

s-ti-cas vi-zu-a-li-za-dos

Manual de Referência do MySQL

Copyright © 1997-2001 MySQL AB

Table of Contents

1 Informações gerais sobre o MySQL

Este é o manual de referência do MySQL para a versão 3.23.52. Como o desenvolvimento do MySQL está em andamento, este manual é atualizado constantemente. Existe uma grande chance desta versão não estar atualizada, a menos que você a esteja lendo online. A versão mais recente deste manual está disponível em <http://www.mysql.com/documentation/> em vários formatos diferentes. Se você está tendo dificuldades para encontrar as informações no manual, tente a versão online em <http://www.mysql.com/documentation/manual.php>.

MySQL é um servidor robusto de bancos de dados SQL (Structured Query Language - Linguagem Estruturada para Pesquisas) muito rápido, multi-tarefa e multi-usuário.

MySQL é um software livre. Licenciado sob a **GNU GENERAL PUBLIC LICENSE** <http://www.gnu.org/>. See [\[Licensing and Support\]](#), page [\[undefined\]](#).

A [home page do MySQL](#) fornece as últimas informações sobre o MySQL.

A seguinte lista descreve algumas seções úteis do manual:

- Para informações sobre a empresa por trás do MySQL, veja [\[What is MySQL AB\]](#), page [\[undefined\]](#).
- Para discussões das capacidades do MySQL, veja [\[Features\]](#), page [\[undefined\]](#).
- Para instruções de instalação, veja [\[Installing\]](#), page [\[undefined\]](#).
- Para dicas sobre a portabilidade do MySQL para novas arquiteturas ou sistemas operacionais, veja [\[Porting\]](#), page [\[undefined\]](#).
- Para informações sobre a atualização da versão 3.22, veja [\[Upgrading-from-3.22\]](#), page [\[undefined\]](#).
- Para um tutorial de introdução ao MySQL, veja [\[Tutorial\]](#), page [\[undefined\]](#).
- Para exemplos de SQL e informações sobre avaliações de desempenho, veja o diretório de benchmarks ('`sql-bench`' na distribuição).
- Para o histórico de novos recursos e correções de erros, veja [\[News\]](#), page [\[undefined\]](#).
- Para uma lista de erros atualmente conhecidos ou mal-funcionamento, veja [\[Bugs\]](#), page [\[undefined\]](#).
- Para projetos futuros, veja [\[TODO\]](#), page [\[undefined\]](#).
- Para ver a lista de todos os colaboradores para este projeto, veja [\[Credits\]](#), page [\[undefined\]](#).

IMPORTANTE:

Relatórios de erros (também chamados bugs), bem como questões e comentários, devem ser enviados para a lista de discussão em mysql@lists.mysql.com. See [\[Bug reports\]](#), page [\[undefined\]](#). O script `mysqlbug` deve ser usado para gerar comunicados de erros.

Em distribuições fonte, o script `mysqlbug` pode ser encontrado no diretório '`scripts`'. Para distribuições binárias, o `mysqlbug` pode ser encontrado no diretório '`bin`'. Se você encontrou um erro de segurança no MySQL, você deve enviar um email para security@mysql.com.

Se você tem alguma sugestão relacionada a acréscimos ou correções neste manual, por favor, envie-a para a equipe do manual em docs@mysql.com.

Este é o manual de referência; ele não fornece instruções gerais sobre SQL ou conceitos de bancos de dados relacionais. Se você deseja informações gerais sobre SQL, veja [\[General-SQL\]](#), page [\[undefined\]](#). Para livros que focam mais especificamente em MySQL, veja [\[MySQL-Books\]](#), page [\[undefined\]](#).

1.1 MySQL, MySQL AB e Open Source

1.1.1 O que é MySQL

MySQL, o mais popular banco de dados SQL Open Source, é fornecido pela MySQL AB. MySQL AB é uma empresa comercial cujo negócios é fornecer serviços relacionados ao banco de dados MySQL. See [\[What is MySQL AB\]](#), page [\[undefined\]](#).

O MySQL é um sistema gerenciador de bancos de dados.

Um banco de dados é uma coleção de dados estruturados. Ele pode ser qualquer coisa desde uma simples lista de compras a uma galeria de imagens ou a grande quantidade de informação da sua rede corporativa. Para adicionar, acessar, e processar dados armazenados em um banco de dados digital, você necessita de um sistema gerenciador de bancos de dados semelhante ao MySQL. Como os computadores são muito bons em lidar com grandes quantidades de dados, o gerenciamento de bancos de dados funciona como a engrenagem central na computação, como utilitários independentes, ou como partes de outras aplicações.

O MySQL é um sistema gerenciador de bancos de dados relacional.

Um banco de dados relacional armazena dados em tabelas separadas embora todos os dados estejam armazenados em um só local. Isso proporciona velocidade e flexibilidade. As tabelas são unidas por relações definidas tornando possível combinar dados de diferentes tabelas nas requisições. A parte SQL do MySQL atende pela "Linguagem estruturada de pesquisas" - a linguagem padrão mais comum usada para acessar bancos de dados.

O é MySQL um software Open Source.

Open Source garante para qualquer pessoa o uso ou modificação do software. Qualquer pessoa pode fazer download do MySQL pela Internet e usá-lo sem ônus. Qualquer pessoa dedicada pode estudar o código fonte e alterá-lo para adequá-lo às suas necessidades. O MySQL usa a GPL (Licença Pública Geral GNU) <http://www.gnu.org>, para definir o que você pode e não pode fazer com o software em diferentes situações. Se sentir desconforto com a GPL ou precisar embutir o MySQL numa aplicação comercial você pode adquirir a versão comercial licenciada conosco.

Por que usar o MySQL?

O MySQL é extremamente rápido, confiável, e fácil de usar. Se isto é o que você está procurando, você deveria experimentá-lo. MySQL também tem um conjunto de recursos muito práticos desenvolvidos com a cooperação de nossos

usuários. Você pode encontrar comparativos de performance do MySQL com outros gerenciadores de bancos de dados na nossa página de benchmark See [\[MySQL Benchmarks\]](#), page [\[MySQL Benchmarks\]](#).

MySQL foi desenvolvido originalmente para lidar com bancos de dados muito grandes de maneira muito mais rápida que as soluções existentes e tem sido usado em ambientes de produção de alta demanda por diversos anos de maneira bem sucedida. Apesar de estar em constante desenvolvimento, o MySQL hoje oferece um rico e proveitoso conjunto de funções. A conectividade, velocidade, e segurança fazem com que o MySQL seja altamente adaptável para acessar bancos de dados na Internet.

As características técnicas do MySQL

Para informações técnicas avançadas, veja [\[Reference\]](#), page [\[Reference\]](#). MySQL é um sistema cliente/servidor que consiste de um servidor SQL multi-tarefa que suporta acessos diferentes, diversos programas clientes e bibliotecas, ferramentas administrativas e diversas interfaces de programação.

Também concedemos o MySQL como uma biblioteca multi-tarefa que você pode ligar à sua aplicação para chegar a um produto mais rápido, menor e mais facilmente gerenciável.

MySQL tem muitos softwares de colaboradores disponível.

É bem provável que sua aplicação ou linguagem favorita já suporte o MySQL.

A pronúncia oficial do MySQL é “Mai Ess Que Ell” (e não MAI-SEQUÊL). Mas tentamos não corrigir as pessoas que dizem MAI-SEQUÊL.

1.1.2 O que é a MySQL AB

MySQL AB é uma empresa sueca que pertence e é administrada pelos fundadores do MySQL e principais desenvolvedores. Nos dedicamos a desenvolver e disseminar nosso banco de dados para novos usuários. MySQL AB é dona do direito autoral referente ao código fonte do servidor MySQL e a marca comercial MySQL. Uma quantidade significativa dos rendimentos de nossos serviços é destinada ao desenvolvimento do MySQL. See [\[What-is\]](#), page [\[What-is\]](#).

A MySQL AB tem obtido lucros a partir do MySQL desde o início. Nós não recebemos capital externo, e todo o dinheiro foi arrecadado por nós.

Estamos procurando parceiros que gostariam de apoiar o desenvolvimento do MySQL para que possamos acelerar o ritmo de desenvolvimento. Se você está interessado, envie um e-mail para partner@mysql.com.

Atualmente, a MySQL AB tem mais de 20 funcionários (<http://www.mysql.com/development/team.html>) na sua folha de pagamento e está crescendo rapidamente.

Nossas principais fontes de recursos são:

- Suporte comercial de alta qualidade para o MySQL fornecido pelos próprios desenvolvedores do MySQL. Se você tem interesse em adquirir um contrato de suporte, por favor visite <https://order.mysql.com/> para ver opções de suporte ou para solicitar suporte.

- Serviços de Consultoria. Nós temos desenvolvedores e consultores em 12 países e parceiros em muitos outros países que podem ajuda-lo com quase qualquer assunto relacionado ao MySQL. Para serviços de consultoria, por favor envie um email descrevendo detalhadamente suas necessidades: info@mysql.com! Se não conseguirmos resolver seu problema, encontraremos um parceiro ou desenvolvedor que possa ajudá-lo.
- Vendemos licenças para usar o MySQL como um Banco de Dados embutido. See [〈undefined〉 \[Cost\], page 〈undefined〉](#). Se você possui um produto comercial e deseja um banco de dados rápido e de alta qualidade, mas não pode tornar seu produto Open Source, poderá comprar o direito de uso do servidor MySQL sobre uma licença comercial normal. As licenças de uso do MySQL são vendidas em <https://order.mysql.com/> ou pelo licensing@mysql.com.
- Propaganda. O <http://www.mysql.com/> é um site de internet muito popular com mais de 10.000.000 page views por mês (Janeiro 2001). Colocando um banner, você estará atingindo vários clientes potenciais na comunidade Open Source, Linux e de Bancos de Dados. Se tiver interesse, envie email para advertising@mysql.com.
- Estamos criando um programa de parcerias que fornecerá serviços MySQL em todos os países. Se estiver interessado em se tornar um parceiro da MySQL AB, por favor visite <http://www.mysql.com/information/partners.html> ou envie um email para partner@mysql.com.
- Fornecemos treinamento através de nossos parceiros. Para maiores informações, por favor envie um email para info@mysql.com.
- A marca MySQL tem, desde 1995, sido associada com velocidade e estabilidade e é conhecida por ser confiável. Caso tenha interesse em usar a marca do MySQL no seu negócio, envie um email para info@mysql.com.

A ideologia do MySQL mostra nossa dedicação ao MySQL e ao Open Source.

Nós desejamos que o MySQL seja:

- O melhor e o mais usado banco de dados no mundo.
- Acessível e disponível para todos.
- Fácil de usar.
- Melhorado continuamente, permanecendo rápido e seguro.
- Divertido para usar e aprimorar.
- Livre de erros (bugs).

A MySQL AB e sua equipe:

- Promovem a filosofia Open Source e suporte à comunidade Open Source.
- Tem como objetivo serem bons cidadãos.
- Tem preferência por parceiros que compartilhem nossos valores e idéias.
- Respondem e-mails e dão suporte.
- São uma empresa virtual, conectada com outras.
- Trabalha contra patentes de sistemas.

1.1.3 Sobre este manual

Este manual é disponível atualmente em versões Texinfo, texto, Info, HTML, PostScript e PDF. O documento original está no formato Texinfo. A versão HTML é produzida automaticamente usando uma versão modificada do `texi2html`. A versão texto e Info são produzidas com `makeinfo`. A versão PostScript é produzida usando `texi2dvi` e `dvips`. A versão PDF é produzida com `pdftex`.

O manual original (em inglês) é escrito e mantido por David Axmark, Michael (Monty) Widenius, Jeremy Cole e Paul DuBois. Para outros colaboradores, veja [\[Credits\]](#), [page](#) [\[Credits\]](#).

1.1.4 Convenções usadas neste manual.

Este manual usa algumas convenções tipográficas:

constant Fonte de largura fixa é usada para nomes de comandos e opções; expressões SQL; bancos de dados, nomes de tabelas e colunas; código C e Perl; e variáveis de ambiente. Exemplo: “Para ver como o `mysqladmin` funciona, execute-o com a opção `--help`.”

`'filename'`

Fonte de largura fixa com aspas é usada para nomes de arquivos e caminhos. Exemplo: “distribuição é instalada sobre o diretório `'/usr/local'`.”

`'c'`

Fonte de largura constante com aspas é também usada para indicar sequências de caracteres. Exemplo: “Para especificar uma máscara, use o caractere `'%'`.”

italic

Fonte Itálica é usada para dar ênfase, *como aqui*.

boldface

Fonte em Negrito é usada para destacar nomes privilegiados (por exemplo, “não permita o privilégio a **process** levemente”) e ocasionalmente indicar **ênfase especial**.

Quando um comando deve ser executado por um programa, ele é indicado por um prompt antes do comando. Por exemplo, `shell>` é um prompt de comando para indicar o seu shell atual e `mysql>` indica um prompt de comando do cliente `mysql`;

```
shell> digite um comando shell aqui
mysql> digite um comando mysql aqui
```

Comandos Shell são mostrados usando a sintaxe do Shell Bourne. Se você usa um shell do estilo `csch`, pode ser necessário alterar algum de seus comandos. Por exemplo, a sequência para configurar uma variável de ambiente e rodar um comando que se parece como abaixo na sintaxe Bourne Shell:

```
shell> NOMEVAR=valor algum_comando
```

Para `csch`, execute a sequência desta forma:

```
shell> setenv NOMEVAR valor
shell> algum_comando
```

Frequentemente, bancos de dados, tabelas, e nomes de colunas devem ser substituídos nos comandos. Para indicar que as substituições são necessárias, este manual usa `nome_db`, `nome_tbl` e `nome_col`. Por exemplo, você pode ver uma expressão assim:

```
mysql> SELECT nome_col FROM nome_bd.nome_tbl;
```

Isso significa que se você estiver trabalhando numa expressão similar, poderá fornecer seu próprio banco de dados, tabela e nomes de colunas, que possivelmente se parecerá com isto:

```
mysql> SELECT nome_autor FROM biblio_bd.lista_autor;
```

Expressões SQL podem ser escritas em maiúsculas ou minúsculas. Quando este manual mostra uma expressão SQL, maiúsculas são usadas para as palavras-chave em questão (para enfatizá-las) e minúsculas são usadas para o resto da expressão. Veja o exemplo na expressão `SELECT` a seguir :

```
mysql> SELECT count(*) FROM nome_tbl;
```

Por outro lado, em uma discussão sobre a função `COUNT()`, a mesma expressão pode ser escrita assim:

```
mysql> select COUNT(*) from nome_tbl;
```

Não existindo o interesse em destacar, todas as palavras-chave são escritas uniformemente em maiúsculas.

Em descrições de sintaxe, colchetes (‘[’ e ‘]’) são usados para indicar palavras ou cláusulas opcionais:

```
DROP TABLE [IF EXISTS] nome_tbl
```

Quando elementos da sintaxe possuem mais de uma alternativa, elas são separados por barras verticais (‘|’). Quando um membro de um conjuntop de opções **pode ser** escolhido, as alternativas são listadas em colchetes (‘[’ e ‘]’):

```
TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)
```

Quando um membro de um conjunto de opções **deve** ser selecionado, as alternativas são listadas em colchetes (‘{’ e ‘}’):

```
{DESCRIBE | DESC} nome_tabela {nome_campo | wild}
```

1.1.5 História do MySQL

Quando começamos, tínhamos a intenção de usar o `mSQL` para conectar às nossas tabelas utilizando nossas rápidas rotinas de baixo nível (ISAM). Entretanto, depois de alguns testes, chegamos a conclusão que o `mSQL` não era rápido e nem flexível o suficiente para nossas necessidades. Isto resultou em uma nova interface SQL para nosso banco de dados, mas com praticamente a mesma Interface API do `mSQL`. Esta API foi escolhida para facilitar a portabilidade para códigos de terceiros.

A derivação do nome MySQL não é bem definida. Nosso diretório base e um grande número de nossas bibliotecas e ferramentas sempre tiveram o prefixo “my” por pelo menos 10 anos. A filha de Monty, alguns anos mais nova que o MySQL, também ganhou o nome My. Qual das duas originou o nome do MySQL continua sendo um mistério, mesmo para nós.

1.1.6 As principais características do MySQL

A seguinte lista descreve algumas das características mais importantes do MySQL:

- Suporte total a multi-threads usando threads diretamente no kernel. Isto significa que se pode facilmente usar múltiplas CPUs, se disponível.

- C, C++, Eiffel, Java, Perl, PHP, Python e Tcl APIs. See [\[Clients\]](#), page [\[undefined\]](#).
- Funciona em diversas plataformas. See [\[Which OS\]](#), page [\[undefined\]](#).
- Aceita diversos tipos de campos: inteiros de 1, 2, 3, 4 e 8 bytes com e sem sinal, FLOAT, DOUBLE, CHAR, VARCHAR, TEXT, BLOB, DATE, TIME, DATETIME, TIMESTAMP, YEAR, SET, and ENUM types. See [\[Column types\]](#), page [\[undefined\]](#).
- Joins muito rápidas usando uma multi-join de leitura única otimizada.
- Completo suporte a operadores e funções nas partes SELECT e WHERE das consultas. Por exemplo:

```
mysql> SELECT CONCAT(first_name, " ", last_name) FROM nome_tabela
        WHERE income/dependents > 10000 AND age > 30;
```

- Funções SQL são implementadas através de uma biblioteca de classes altamente otimizada e com o máximo de performance. Geralmente não há nenhuma alocação de memória depois da inicialização da pesquisa.
- Suporte pleno às cláusulas SQL GROUP BY e ORDER BY. Suporte para funções de agrupamento (COUNT(), COUNT(DISTINCT ...), AVG(), STD(), SUM(), MAX() e MIN()).
- Suporte para LEFT OUTER JOIN e RIGHT OUTER JOIN e sintaxes ANSI, SQL e ODBC.
- Você pode misturar tabelas de bancos de dados diferentes na mesma pesquisa (como na versão 3.22).
- Um sistema de privilégios e senhas que é muito flexível, seguro e que permite verificação baseada em estações/máquinas. Senhas são seguras porque todo o tráfego de senhas é criptografado quando você se conecta ao servidor.
- Suporte ao ODBC (Open-DataBase-Connectivity) para Win32 (com fonte aberto). Todas as funções ODBC 2.5 e muitas outras. Por exemplo, você pode usar o MS Access para conectar ao seu servidor MySQL. See [\[ODBC\]](#), page [\[undefined\]](#)
- Tabelas de disco baseadas em árvores-B extremamente rápidas com compressão de índices.
- São permitidos até 32 índices por tabela. Cada índice pode ser composto de 1 a 16 colunas ou partes de colunas. O tamanho máximo do índice é de 500 bytes (isto pode ser alterado na compilação do MySQL). Um índice pode usar o prefixo de campo com um tipo CHAR ou VARCHAR.
- Registros de tamanhos fixos ou variáveis.
- Tabelas hash em memória que são usadas como tabelas temporárias.
- Lida com bancos de dados enormes. Nós estamos usando MySQL com alguns bancos de dados que contém 50.000.000 registros e nós sabemos de usuários que usam MySQL com 60.000 tabelas e aproximadamente 5.000.000.000 de linhas.
- Todas as colunas têm valores padrão. Você pode usar INSERT para inserir um subconjunto de colunas de tabelas; aquelas colunas que não possuem valores fornecidos explicitamente usam os seus valores padrões.
- Utiliza o GNU Automake, Autoconf, e Libtool para portabilidade.
- Escrito em C e C++. Testado com um amplo faixa de compiladores diferentes.
- Um sistema de alocação de memória muito rápido e baseado em processo(thread).

- Não existem furos ou problemas de memória. O MySQL foi testado com o Purify, um detector de problemas de memória comercial.
- Inclui o `myisamchk`, um utilitário muito rápido para checagem, otimização e reparo de tabelas. Toda a funcionalidade do `myisamchk` está, também, disponível pela interface SQL. See [\[MySQL Database Administration\]](#), page [\[MySQL Database Administration\]](#).
- Suporte total para vários conjuntos de caracteres, que incluem ISO-8859-1 (Latin1), big5, ujis e mais. Por exemplo, os caracteres Escandinavos ‘å’, ‘ä’ e ‘ö’ são permitidos em nomes de tabelas e colunas.
- Todos os dados são armazenados no conjunto de caracteres escolhido. Todas as comparações em colunas de sequenciamento são insensíveis.
- A ordenação é feita de acordo com o conjunto de caracteres escolhido (o modo suéco por padrão). É possível alterar isso quando o servidor MySQL é iniciado. Para ver um exemplo de várias ordenações avançadas, procure pelo código de ordenação Tcheca. O MySQL suporta diversos conjuntos de caracteres que podem ser especificados em tempo de compilação e execução.
- Apelidos em tabelas e colunas são disponíveis como definidos no padrão SQL92.
- `DELETE`, `INSERT`, `REPLACE`, e `UPDATE` retornam o número de linhas que foram alteradas (afetadas). É possível retornar o número de linhas com padrão coincidentes configurando um parâmetro quando estiver conectando ao servidor.
- Nomes de funções não conflitam com nomes de tabelas ou colunas. Por exemplo, `ABS` é um nome de campo válido. A única restrição é que para uma chamada de função, espaços não são permitidos entre o nome da função e o ‘(’ que o segue. See [\[Reserved words\]](#), page [\[Reserved words\]](#).
- Todos os programas MySQL podem ser chamados com as opções `--help` ou `-?` para obter ajuda online.
- O servidor pode apresentar mensagem de erros aos clientes em várias línguas. See [\[Languages\]](#), page [\[Languages\]](#).
- Os clientes podem se conectar ao servidor MySQL usando sockets (tomadas) TCP/IP, sockets Unix, ou Named Pipes (NT).
- O comando específico do MySQL `SHOW` pode ser usado para devolver informações sobre bancos de dados, tabelas e índices. O comando `EXPLAIN` pode ser usado para determinar como o otimizador resolve a consulta.

1.1.7 O MySQL é estável?

Esta seção discute as questões “Quanto estável é o MySQL?” e “Posso depender do MySQL neste projeto?” Tentaremos deixar claro alguns assuntos e responder algumas das questões mais importantes que parecem interessar muitas pessoas. Esta seção foi colocada juntamente com a informação colhida da lista de discussão (que é muito ativa em relatar erros).

Na TcX, o MySQL vem trabalhando sem problema em nossos projetos desde o meio de 1996. Quando o MySQL foi disponibilizado para um público maior, nós fomos notificados que existiam algumas peças de “código sem testes” que foram sendo rapidamente encontrados pelos novos usuários que criavam pesquisas de uma maneira diferente das que nós fazíamos. Cada nova release teve menos problemas de portabilidade que o anterior (mesmo com os novos recursos implementados em cada uma destas versões)

Cada release do MySQL foi sendo usado, e ocorrem problemas somente quando usuários começam a usar as “áreas cinzentas.” Naturalmente, usuários externos não sabem o que são as áreas cinzentas; esta seção tenta indicar aquelas que são conhecidas atualmente. As descrições lidam com a Versão 3.23 do MySQL. Todos os erros conhecidos e relatados são reparados na última versão, com a exceção dos bugs listados na seção de erros, os quais são relacionados ao desenho. See [\[Bugs\]](#), page [\[Bugs\]](#)

O MySQL é escrito em múltiplas camadas e diferentes módulos independentes. Estes módulos estão listados abaixo com indicações de quão bem-testado foi cada um deles.

O handler de tabelas ISAM — Estável

Gerencia armazenamento e restauração de todos os dados no MySQL Versão 3.22 e anteriores. Em todas versões do MySQL, nunca foi relatado um só erro neste código. A única forma conhecida de corromper uma tabela é matar o servidor no meio de uma atualização. Mesmo isto não é suficiente para destruir qualquer dados que não possam ser recuperados, porque todos os dados são atualizados para o disco entre cada query. Não existe nenhum relato de erros sobre perda de dados causados por bugs no MySQL.

O handler de tabelas MyISAM — Estável

Este é novo na versão do MySQL 3.23. É amplamente baseado nos códigos de tabelas ISAM mas possui diversos recursos novos e uteis.

O parser e o analisador léxico — Estável

Não existem relatos de erros neste sistema há muito tempo.

O código C cliente — Estável

Não existem problemas conhecidos. Em versões mais antigas que a 3.20, existiam algumas limitações no tamanho do buffer de envio/recebimento. Como na versão 3.21, o tamanho do buffer agora é dinâmico podendo chegar a um padrão de 16M.

Programas clientes padrões — Estável

Inclui `mysql`, `mysqladmin`, `mysqlshow`, `mysqldump`, e `mysqlimport`.

SQL Básico — Estável

O manipulador do sistema de funções, da classe string e da memória dinamica do SQL basico. Não existem relatos de erros neste sistema.

Otimizador de pesquisas — Estável

Otimizador do Range — Estável

Otimizador de Join — Estável

Locking — Gamma

Esse módulo é muito dependente do sistema. Em alguns sistemas existem certos problemas por utilizar o locking padrão do SO (`fcntl()`). Em alguns casos, você pode executar o daemon do MySQL com o parâmetro `--skip-locking`. São conhecidos alguns problemas ocorridos em alguns sistemas Linux e no SunOS quando utiliza-se sistemas de arquivos montados em NFS.

Threads Linux — Estável

O maior problema encontrado neste módulo foi com a chamada `fcntl()`, que é corrigida usando a opção `--skip-locking` com o `mysqld`. Algumas pessoas

relataram travamentos com a versão 0.5. Se você planeja usar mais de 1000 conexões simultâneas, o LinuxThreads deverá ser recompilado. Embora seja possível rodar todas essas conexões com o LinuxThreads padrão (de qualquer forma, você nunca poderá ir acima de 1021 conexões), o espaço de pilha padrão de 2MB deixa a aplicação instável, e iremos reproduzir um coredump depois de criar 1021 conexões. See [\[Linux\]](#), page [\[Linux\]](#)

Solaris 2.5+ pthreads — Estável

Utilizamos isto para todo nosso trabalho de produção.

MIT-pthreads (Outros Sistemas) — Estável

Não há nenhum erro relatado desde da versão 3.20.15 e nenhum erro conhecido desde a versão 3.20.16. Em alguns sistemas, há um "mal funcionamento" onde algumas operações são muito lentas (uma parada de 1/20 segundos e feita entre cada consulta). É claro que MIT-pthreads pode deixar tudo um pouco lento, mas as instruções **SELECT** baseada em índice geralmente são feitas uma de cada vez, então elas não precisam ser travadas com "mutex locking" nas threads.

Outras implementações de threads — Beta -

A portabilidade para outros sistemas ainda é muito nova e podem existir erros, possivelmente no MySQL, mas a maioria na própria implementação das threads.

LOAD DATA ..., INSERT ... SELECT — Estável

Algumas pessoas achavam que haviam encontrado bugs aqui, mas geralmente são mal-entendidos com relação ao funcionamento desses módulos. Por favor confira o manual antes de relatar problemas!

ALTER TABLE — Estável

Pequenas alterações na versão 3.22.12.

DBD — Estável

Atualmente mantido por Jochen Wiedmann (wiedmann@neckar-alb.de). Obrigado!

mysqlaccess — Estável

Escrito e mantido por Yves Carlier (Yves.Carlier@rug.ac.be). Obrigado!

GRANT — Estável

Grandes alterações feitas no MySQL Versão 3.22.12.

MyODBC (Usa ODBC SDK 2.5) — Gamma

Parece funcionar bem com muitos programas.

Replicação – Beta / Gamma

Ainda estamos trabalhando na replicação, portanto não espere que isso irá ser sólido como uma rocha ainda. Por outro lado, alguns usuários MySQL já estão usando isto com ótimos resultados.

Tabelas BDB – Beta

O código de Bancos de dados Berkeley é muito estável, mas nós ainda estamos melhorando a interface entre o MySQL e as tabelas BDB, portanto ele irá levar algum tempo até ser testado como os outros tipos de tabelas foram.

Tabelas InnoDB – Beta

Esta é uma adição recente ao MySQL. Ela parece funcionar bem e pode ser usada depois de alguns testes iniciais.

Recuperação automática de tabelas MyISAM - Beta

Isso afeta somente o novo código que confere, na inicialização do banco, se a tabela foi fechada corretamente e executa uma conferência/reparo automático da tabela em caso negativo.

MERGE de Tabelas – Beta / Gamma

O uso das chaves em MERGE de tabelas ainda não foi bem testado. A outra parte do código MERGE foi muito bem testada.

FULLTEXT – Beta

A pesquisa textual aparenta funcionar, mas ainda não é amplamente usada.

MySQL AB fornece suporte por e-mail para clientes pagos, mas a lista de discussão MySQL normalmente fornece respostas para questões comuns. Erros são normalmente corrigidos com um patch; para erros sérios, normalmente é lançada uma nova release.

1.1.8 Qual o tamanho das tabelas que o MySQL pode suportar

A Versão 3.22 do MySQL tem suporte para tabelas com limite de tamanho até 4G. Com o novo MyISAM no MySQL versão 3.23 o tamanho máximo foi estendido até 8 milhões de terabytes (2^{63} bytes).

Lembre, entretando, que sistemas operacionais tem seu próprio limite de tamanho para arquivos. Seguem alguns exemplos:

Sistema Operacional	Limite do tamanho do arquivo
Linux-Intel 32 bit	2G, 4G ou mais, depende da versão do Linux
Linux-Alpha	8T (?)
Solaris 2.5.1	2G (possivelmente 4G com patch)
Solaris 2.6	4G
Solaris 2.7 Intel	4G
Solaris 2.7 ULTRA-SPARC	8T (?)

No Linux 2.2 você pode ter tabelas maiores que 2G usando o patch LFS para o sistema de arquivos ext2. No Linux 2.4 já existem patches para o sistema de arquivos ReiserFS para ter suporte a arquivos maiores.

Isto significa que o tamanho da tabela para o MySQL é normalmente limitado pelos sistemas operacionais.

Por padrão, tabelas do MySQL têm um tamanho máximo em torno de 4G. Você pode verificar o tamanho máximo da tabela com o comando `SHOW TABLE STATUS` ou com o `myisamchk -dv nome_tabela` See [\[SHOW\]](#), page [\[undefined\]](#).

Se você precisa de tabelas maiores que 4G (e seu sistema operacional suporta isto), você pode configurar o `AVG_ROW_LENGTH` e o parâmetro `MAX_ROWS` quando você criar sua tabela. See [\[CREATE TABLE\]](#), page [\[undefined\]](#). Você pode também alterar isso mais tarde com `ALTER TABLE`. See [\[ALTER TABLE\]](#), page [\[undefined\]](#)

Se sua tabela grande será somente leitura, você poderá usar o `myisampack` para unir e comprimir várias tabelas em uma. `myisampack` normalmente comprime uma tabela em pelo menos 50%, portanto você pode obter, com isso, tabelas muito maiores. See [\(undefined\)](#) [`myisampack`], page [\(undefined\)](#)

Uma opção é ignorar o limite de tamanho de arquivos do sistema operacional para arquivos de dados MyISAM usando a opção RAID. See [\(undefined\)](#) [`CREATE TABLE`], page [\(undefined\)](#).

Outra solução pode ser a biblioteca MERGE incluída, que lhe permite acessar uma coleção de tabelas idênticas como se fosse apenas uma. See [\(undefined\)](#) [`MERGE`], page [\(undefined\)](#)

1.1.9 Compatibilidade com o ano 2000 (Y2K)

O MySQL não apresenta nenhum problema com o ano 2000 (Y2K compatível)

- O MySQL usa funções de tempo Unix e não tem problemas com datas até o ano 2069; todos os anos de 2 dígitos são estimados na escala 1970 até 2069; o que significa que se você armazenar 01 numa coluna `year`, O MySQL o tratará como 2001.
- Todas as funções de data do MySQL estão no arquivo `'sql/time.cc'` e codificadas com muito cuidado para ser compatível com o ano 2000.
- No MySQL versão 3.22 e posterior, o novo tipo de campo `YEAR` pode armazenar anos 0 e 1901 até 2155 em 1 byte e mostrá-lo usando 2 ou 4 dígitos.

Você pode ter problemas com aplicações que usem o MySQL de uma maneira que não seja segura à Y2K. Por exemplo, muitas aplicações antigas armazenam ou manipulam anos usando valores de 2 dígitos (que é ambíguo) em vez de usar valores de 4 dígitos. Este problema pode ser composto por aplicações que usam valores como em 00 ou 99 como indicadores de valores “perdidos”.

Infelizmente, estes problemas podem ser difíceis de resolver, porque diferentes aplicações podem ser escritas por diferentes programadores, e cada um deles pode usar um diferentes conjunto de convenções e funções manipuladoras de data.

Aqui está uma demonstração simples ilustrando que o MySQL não tem nenhum problema com datas até o ano 2030:

```
mysql> DROP TABLE IF EXISTS y2k;
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> CREATE TABLE y2k (date date, date_time datetime, time_stamp timestamp);
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> INSERT INTO y2k VALUES
-> ("1998-12-31", "1998-12-31 23:59:59", 19981231235959),
-> ("1999-01-01", "1999-01-01 00:00:00", 19990101000000),
-> ("1999-09-09", "1999-09-09 23:59:59", 19990909235959),
-> ("2000-01-01", "2000-01-01 00:00:00", 20000101000000),
-> ("2000-02-28", "2000-02-28 00:00:00", 20000228000000),
-> ("2000-02-29", "2000-02-29 00:00:00", 20000229000000),
-> ("2000-03-01", "2000-03-01 00:00:00", 20000301000000),
-> ("2000-12-31", "2000-12-31 23:59:59", 20001231235959),
-> ("2001-01-01", "2001-01-01 00:00:00", 20010101000000),
```

```

-> ("2004-12-31","2004-12-31 23:59:59",20041231235959),
-> ("2005-01-01","2005-01-01 00:00:00",20050101000000),
-> ("2030-01-01","2030-01-01 00:00:00",20300101000000),
-> ("2050-01-01","2050-01-01 00:00:00",20500101000000);
Query OK, 13 rows affected (0.01 sec)
Records: 13 Duplicates: 0 Warnings: 0

```

```

mysql> SELECT * FROM y2k;
+-----+-----+-----+
| date      | date_time          | time_stamp      |
+-----+-----+-----+
| 1998-12-31 | 1998-12-31 23:59:59 | 19981231235959 |
| 1999-01-01 | 1999-01-01 00:00:00 | 19990101000000 |
| 1999-09-09 | 1999-09-09 23:59:59 | 19990909235959 |
| 2000-01-01 | 2000-01-01 00:00:00 | 20000101000000 |
| 2000-02-28 | 2000-02-28 00:00:00 | 20000228000000 |
| 2000-02-29 | 2000-02-29 00:00:00 | 20000229000000 |
| 2000-03-01 | 2000-03-01 00:00:00 | 20000301000000 |
| 2000-12-31 | 2000-12-31 23:59:59 | 20001231235959 |
| 2001-01-01 | 2001-01-01 00:00:00 | 20010101000000 |
| 2004-12-31 | 2004-12-31 23:59:59 | 20041231235959 |
| 2005-01-01 | 2005-01-01 00:00:00 | 20050101000000 |
| 2030-01-01 | 2030-01-01 00:00:00 | 20300101000000 |
| 2050-01-01 | 2050-01-01 00:00:00 | 00000000000000 |
+-----+-----+-----+
13 rows in set (0.00 sec)

```

Isto mostra que os tipos `DATE` e `DATETIME` não apresentarão problemas com datas futuras (eles irão conseguir trabalhar com datas até o ano 9999).

O tipo `TIMESTAMP`, que é usado para armazenar a hora atual, tem um alcance somente até 2030-01-01. `TIMESTAMP` tem um alcance de 1970 até 2030 em máquinas 32-bits (valor com sinal). Em máquinas de 64-bits ele pode trabalhar com datas até 2106 (valor sem sinal).

Mesmo apesar do MySQL ser compatível com o ano 2000, é de sua responsabilidade fornecer datas que não sejam ambíguas. Veja [\[Y2K issues\]](#), [page](#) para regras do MySQL para lidar com entrada de datas ambíguas (datas contendo valores de ano com 2 dígitos)

1.2 Fontes de Informações Sobre MySQL

1.2.1 Livros sobre o MySQL

Para informações mais recentes sobre livros, com comentários dos usuários, por favor visite <http://www.mysql.com/portal/books/html/index.html>.

Enquanto este manual ainda for o lugar certo para informações técnicas atualizadas, seu objetivo principal será conter tudo que é conhecido sobre o MySQL. Ao mesmo tempo é interessante ter um livro para ler na cama ou enquanto você viaja. Segue aqui uma lista de livros sobre MySQL e assuntos relacionados (em Inglês).

Comprando um livro através dos hyperlinks fornecidos aqui você estará contribuindo para o desenvolvimento do MySQL.

MySQL

Disponível em	Barnes and Noble
Editor	New Riders
Autor	Paul DuBois
Data de Publicação	1st Edition December 1999
ISBN	0735709211
Páginas	800
Preço	\$49.99 US
Exemplos para Download	samp_db distribution
Errata	are available here

Revisado por Michael “Monty” Widenius, Moderador do MySQL.

Em *MySQL*, Paul DuBois introduz a você um guia completo para um dos mais populares bancos de dados relacionais. Paul tem contribuído para a documentação online para o MySQL e é um membro ativo da comunidade MySQL. O principal desenvolvedor do MySQL, Monty Widenius, e uma rede de seus companheiros desenvolvedores revisaram o manuscrito, e forneceram a Paul o tipo de discernimento que ninguém poderia fornecer.

Ao invés de simplesmente dar uma visão geral do MySQL, Paul o ensina grande parte de seus conhecimentos. Através de duas tabelas exemplos utilizadas por todo o livro, ele mostra soluções para problemas que você certamente irá encontrar. Ele o ajuda a integrar MySQL, eficientemente, com ferramentas de terceiros, como PHP e Perl, habilitando-o a gerar páginas Web dinâmicas através de consultas a banco de dados. Ele o ensina a escrever programas que acessam banco de dados MySQL, e também fornece um conjunto completo de referências para tipos de campos, operadores, funções, sintaxe SQL, programando em MySQL, API C, Perl DBI, e API PHP. *MySQL* simplesmente lhe dará informações que você não encontrará em nenhum outro lugar.

Se você usa o MySQL, este livro lhe oferece:

- Uma introdução ao MySQL e SQL.
- Cobertura dos tipos de dados do MySQL e como usá-los.
- Perfeito tratamento de como escrever programas clientes em C.
- Um guia para usar o Perl DBI e APIs PHP para desenvolver aplicações baseadas em linha de comando e Web.
- Dicas em assuntos administrativos como contas de usuários, backup, recuperação em caso de quedas e segurança.
- Ajuda para escolher um Provedor de Serviços de Internet para acesso MySQL.
- Uma referência completa para tipos de dados MySQL, operadores, funções, instruções SQL e utilitários.
- Guias de referência completos para a API C do MySQL, a API do Perl DBI e funções PHP relacionadas ao MySQL.

MySQL & mSQL

Disponível em	Barnes and Noble
Editor	O'Reilly
Autores	Randy Jay Yarger, George Reese & Tim King
Data de Publicação	1st Edition July 1999
ISBN	1-56592-434-7, Order Number: 4347
Páginas	506
Preço	\$34.95

Este livro ensina você como usar o MySQL e mSQL, dois populares e robustos produtos de banco de dados que suportam subsistemas de chave de SQL em ambos os sistemas Linux e Unix. Qualquer um que conheça C básico, Java, Perl ou Python pode escrever um programa para interagir com um banco de dados, podendo ser uma aplicação isolada ou através de uma página Web. Este livro leva você através de todo o processo, da instalação e configuração até a programação de interfaces e administração básica. Inclui farto material de tutorial.

Sams' Teach Yourself MySQL in 21 Days

Disponível em	Barnes and Noble
Editor	Sams
Autores	Mark Maslakowski and Tony Butcher
Data de Publicação	Junho 2000
ISBN	0672319144
Páginas	650
Preço	\$39.99

O *Teach Yourself MySQL in 21 Days* da Sams é indicado para usuários Linux intermediários que desejam entrar na área de bancos de dados. Boa parte do público alvo é formada por desenvolvedores Web que necessitam de um banco de dados para armazenar grande quantidade de informações para que possam ser recuperadas pela Web.

Teach Yourself MySQL in 21 Days é um prático tutorial passo-a-passo. O leitor irá aprender desenvolver e trabalhar essa tecnologia de bancos de dados open source em seu Web site usando exemplos práticos para seguir.

E-Commerce Solutions with MySQL

Disponível em	Barnes and Noble
Editor	Prima Communications, Inc.
Autores	Não disponível
Data de Publicação	Janeiro 2000
ISBN	0761524452
Páginas	500
Preço	\$39.99

Sem descrição disponível.

MySQL and PHP from Scratch

Disponível em	Barnes and Noble
Editor	Que
Autores	N/A
Data de Publicação	September 2000
ISBN	0789724405
Páginas	550
Preço	\$34.99

Este livro disponibiliza de forma conjunta informações sobre instalação, configuração e correção de problemas no Apache, MySQL, PHP3 e IMP em um volume completo. Você aprende também como cada peça é parte de um conjunto de aprendizagem passo-a-passo, para criar um sistema de e-mail baseado em Web. Aprenda a executar o equivalente ao Active Server Pages (ASP) usando PHP3, configurar um site de e-commerce usando um banco de dados e um servidor Web Apache, e criar um sistema de entrada de dados (como em vendas, controle de qualidade de produção, preferências dos clientes, etc) para instalar no seu PC.

Professional MySQL Programming

Disponível em	Barnes and Noble
Editor	Wrox Press, Inc.
Autores	N/A
Data de Publicação	Late 2001
ISBN	1861005164
Páginas	1000
Preço	\$49.99

Sem descrição disponível.

Professional Linux Programming

Disponível em	Barnes and Noble
Editor	Wrox Press, Inc.
Autores	N/A
Data de Publicação	September 2000
ISBN	1861003013
Páginas	1155
Preço	\$47.99

Neste Best-Selling *Beginning Linux Programming* você aprenderá com o vivência e experiência dos autores em desenvolvimento de software para Linux; Você irá acompanhar o desenvolvimento da aplicação exemplo de uma Loja de DVD, com capítulos que o levam a diferentes aspectos de sua implementação. Capítulos individuais cobrem tópicos importantes que vão além do tema central. Todo o foco é feito sobre aspectos praticos da programação, mostrando a importância de se escolher a ferramenta certa para o trabalho, usando-a da forma correta e fazendo da forma correta na primeira vez.

PHP and MySQL Web Development

Disponível em	Barnes and Noble
Editor	Sams

Autores	Luke Welling, Laura Thomson
Data de Publicação	March 2001
ISBN	0672317842
Páginas	700
Preço	\$49.99

PHP and MySQL Web Development traz à você as vantagens de se implementar o MySQL e PHP. As vantagens são detalhadas através das condições estatísticas e de vários estudos de caso. Uma aplicação web prática é desenvolvida ao longo do livro, fornecendo a você quais são as ferramentas necessárias para implementar um banco de dados online e funcional. Cada função é desenvolvida separadamente, o que lhe permite escolher incorporar somente partes que você deseja implementar. Conceitos de programação da linguagem PHP são destacados, incluindo funções das quais amarram o suporte MySQL em um script PHP e tópicos avançados cobrindo manipulação de tabelas.

Livros recomendados pelos desenvolvedores do MySQL

SQL-99 Complete, Really

Disponível em	Barnes and Noble
Editor	CMP Books
Autores	Peter Gultzan, Trudy Pelzer
Data de Publicação	April 1999
ISBN	0879305681
Páginas	1104
Preço	\$55.96

Este livro contém descrições completas dos novos padrões de sintaxe, estrutura de dados, e processos de recuperação de informação de bancos de dados SQL. Como um manual de referência baseado em exemplos, ele inclui todas as funções CLI, informações, tabelas de esquemas e códigos de status, bem como um banco de dados funcional fornecido no disco que acompanha o livro.

C, A reference manual

Disponível em	Barnes and Noble
Editor	Prentice Hall
Autores	Samuel P. Harbison, Guy L. Steele
Data de Publicação	September 1994
ISBN	0133262243
Páginas	480
Preço	\$35.99

Uma nova e aprimorada revisão de referência à linguagem C. Este manual lhe introduz noções de "C Limpo", escrevendo código C que pode ser compilado como um programa C++, estilo de programação C que enfatiza comportamento apropriado, portabilidade, sustentabilidade, e incorpora a emenda 1 do ISO C (1994) que especifica novas facilidades para escrever programas em C portáveis e internacionais.

C++ for Real Programmers

Disponível em	Barnes and Noble
---------------	----------------------------------

Editor	Academic Press, Incorporated
Autores	Jeff Alger, Jim Keogh
Data de Publicação	February 1998
ISBN	0120499428
Páginas	388
Preço	\$39.95

C++ For Real Programmers preenche a lacuna entre C++ como foi descrito nos livros indicados para programadores de nível iniciante e intermediário e o C++ utilizado por programadores experientes. Várias técnicas preciosas são descritas, organizadas em três temas simples: indireção, hierarquia de classes e gerenciamento de memória. Ele também cobre detalhadamente a criação de templates, ponteiros, e técnicas de otimização. O foco do livro é em C++ ANSI independente de compilador.

C++ For Real Programmers é uma revisão do *Secrets of the C++ Masters* e inclui um novo apêndice comparando C++ com Java. O livro vem com um disquete de 3.5" com código fonte para windows.

Algorithms in C

Disponível em	Barnes and Noble
Editor	Addison Wesley Longman, Inc.
Autores	Robert Sedgewick
Data de Publicação	April 1990
ISBN	0201514257
Páginas	648
Preço	\$45.75

Algorithms in C descreve uma variedade de algoritmos em algumas áreas de interesse, incluindo: ordenação, pesquisa, processamento de strings e algoritmos geométricos, gráficos e matemáticos. O livro enfatiza técnicas fundamentais, fornecendo aos leitores as ferramentas para executar, implementar e debugar úteis algoritmos com confiança.

Multithreaded Programming with Pthreads

Available	Barnes and Noble
Publisher	Prentice Hall
Authors	Bil Lewis, Daniel J. Berg
Pub Date	October 1997
ISBN	0136807291
Pages	432
Price	\$34.95

Baseado no best-selling *Threads Primer*, *Multithreaded Programming with Pthreads* dá a você um sólido conhecimento de threads Posix: o que são, como funcionam, quando usá-las e como otimizá-las. Ele mantém a clareza e o humor de *Threads Primer*, mas inclui comparações com as implementações em Win32 e OS/2. Códigos exemplos testados na maioria das plataformas UNIX são disponibilizados com explicações detalhadas de como e porque eles usam threads.

Programming the PERL DBI: Database Programming with PERL

Available	Barnes and Noble
Publisher	O'Reilly & Associates, Incorporated
Authors	Alligator Descartes, Tim Bunce
Pub Date	February 2000
ISBN	1565926994
Pages	400
Price	\$27.96

Programming the Perl DBI é escrito em conjunto com Alligator Descartes, um dos membros mais ativos da comunidade DBI, e como Tim Bunce, o inventor do DBI. Para os iniciantes o livro explica a arquitetura do DBI e mostra como escrever programas baseados em DBI. Para os mais experientes o livro explica nuances do DBI e peculiaridades de cada DBD individual.

Este livro inclui:

- Uma introdução ao DBI e seu desenvolvimento.
- Como criar consultas e casar os parâmetros.
- Trabalhando com banco de dados, drivers e manipulando instruções.
- Técnicas para debugar.
- Cobertura de cada DBD existente.
- Uma referência completa do DBI.

1.2.2 Informações gerais e tutoriais

O seguinte livro foi recomendado por várias pessoas na lista de discussão do MySQL:

Judith S. Bowman, Sandra L. Emerson and Marcy Darnovsky
The Practical SQL Handbook: Using Structured Query Language
Second Edition
Addison-Wesley
ISBN 0-201-62623-3
<http://www.awl.com>

O seguinte livro recebeu algumas recomendações por usuários do MySQL:

Martin Gruber
Understanding SQL
ISBN 0-89588-644-8
Publisher Sybex 510 523 8233
Alameda, CA USA

Um tutorial sobre SQL está disponível na rede em <http://w3.one.net/~jhoffman/sqltut.htm>

1.2.3 Links úteis relacionados ao MySQL

Além dos links a seguir, você pode encontrar e fazer download de vários programas, ferramentas e APIs para MySQL do [Diretorio de Colaboradores](#).

MySQL

Tutoriais e Manuais

MySQL Myths Debunked

MySQL usado no "mundo real".

<http://www.4t2.com/mysql>

Informações sobre a lista de discussão alemã do MySQL.

<http://www2.rent-a-database.de/mysql/>

MySQL handbook em alemão.

<http://www.bitmover.com:8888//home/bk/mysql>

Acesso web ao repositório BitKeeper do MySQL.

<http://www.analysisandsolutions.com/code/mybasic.htm>

Tutorial para iniciantes no MySQL em como instalar e configurar o MySQL numa máquina Windows.

http://www.devshed.com/Server_Side/MySQL/

Vários tutorias sobre MySQL.

<http://mysql.hitstar.com/>

Manual do MySQL em chinês.

<http://www.linuxplanet.com/linuxplanet/tutorials/1046/1/>

Configurando um site web baseado em MySQL.

<http://www.hotwired.com/webmonkey/backend/tutorials/tutorial1.html>

Tutorial Perl-MySQL.

<http://www.iserver.com/support/contrib/perl5/modules.html>

Instalando novos módulos Perl que necessitam de módulos localmente instalados.

<http://www.hotwired.com/webmonkey/databases/tutorials/tutorial4.html>

Tutorial PHP/MySQL.

<http://www.useractive.com/>

Tutorial para MySQL.

Portando MySQL/Usando MySQL em diferentes sistemas

<http://www.entropy.ch/software/macosx/mysql/>

Binários do MySQL para MAC OS X. Inclui informações sobre como construir e usar MySQL no MAC OS X.

<http://xclave.macnn.com/MySQL/>

O Mac OS Xclave. Executando MySQL no Mac OS X.

<http://www.prnet.de/RegEx/mysql.html>

MySQL para Servidor Mac OS X .

<http://www.latencyzero.com/macosx/mysql.html>

Compilando o MySQL para o Mac OS X.

<http://www.essencesw.com/Software/mysql.lib.html>

Novas bibliotecas cliente para o MAC OS Classic (Macintosh).

<http://www.lilback.com/macsql/>

Bibliotecas cliente para o MAC OS Classic (Macintosh).

http://sixk.maniasys.com/index_en.html

MySQL para a plataforma Amiga.

Links relacionados a Perl

<http://dbimysql.photoflux.com/>

FAQ MySQL com DBI Perl.

Fórum de Discussão MySQL

<http://www.weberdev.com/>

Exemplos de usos do MySQL; (Veja os Top 20)

<http://futurerealm.com/forum/futureforum.htm>

FutureForum Web Discussion Software.

Aplicações comerciais que suportam o MySQL

<http://www.supportwizard.com/>

SupportWizard; Ajuda interativa na Web (Este produto inclui uma cópia licenciada do MySQL.)

<http://www.sonork.com/>

Sonork, instant messenger que não é orientado somente à Internet. Ele é focado em redes privadas e em pequenas e médias empresas. O Cliente é gratuito e o servidor também é gratuito para até 5 usuários.

<http://www.stweb.org/>

StWeb - Servidor Web e de aplicações Stratos - Fácil de usar, plataforma cruzada, sistema de desenvolvimento e entrega de aplicações Web para Internet/Intranet. A versão padrão do StWeb tem uma interface nativa para o banco de dados MySQL.

<http://www.rightnowtech.com/>

Right Now Web; Automação Web para serviços à clientes.

<http://www.icaap.org/Bazaar/>

Bazaar; Fóruns de discussão interativas com interface Web.

<http://www.phonesweep.com/>

PhoneSweepT é o primeiro sistema de busca de telefone comercial do mundo. Muitas invasões ocorridas nos últimos anos, não são feitas através da internet, mas através chamadas não autorizadas através de modems. PhoneSweep permite que você encontre estes modems através de repetidas chamadas locais para todos os números de telefones que sua empresa controla. PhoneSweep tem um sistema que pode reconhecer mais de 250 tipos diferentes de programas de acesso

remoto, incluindo Carbon Copy(TM), pcAnywhere(TM), e Windows NT RAS. Todas as informações são armazenadas em um banco de dados SQL. É gerado então um relatório completo detalhando quais serviços forma descobertos em cada numero de telefone em sua empresa.

Cientes SQL e Geradores de Relatórios

urSQL Editor SQL e Utilitário de Consultas. Marcador de sintaxe personalizado, grade de resultados editaveis, exportação de resultados, funções básicas de administração de NySQL, Etc.. Para Windows.

MySQL Data Manager

MySQL Data Manager * é cliente web independente de plataforma (escrita em perl) para servidor MySQL sobre TCP/IP.

<http://ksql.sourceforge.net/>
cliente MySQL para KDE.

<http://www.ecker-software.de>
Um cliene GUI para Windows por David Ecker.

<http://www.icaap.org/software/kiosk/>
Kiosk; um cliente MySQL para gerenciamento de banco de dados. Escrito em Perl. Será uma parte do Bazaar.

<http://www.casestudio.com/>
Ferramenta de desenvolvimento que suporta MySQL 3.23.

<http://home.skif.net/~voland/zeos/eng/index.html>
Zeos - Um cliente que suporta MySQL, Interbase e PostgreSQL.

<http://www.geocities.com/SiliconValley/Ridge/4280/GenericReportWriter/grwhome.html>
Um gerador de relatorios gratuito escrito em Java

<http://www.javaframework.de>
MySQLExport - Exportação instruções de criação do MySQL e dados em diversos formatos (SQL, HTML, CVS, text, ZIP, GZIP...)

<http://dlabs.4t2.com>
M2D, Um cliente administrativo do MySQL para windows. M2D suporta administração de Bancos de Dados MySQL, criação de novos bancos de dados e tabelas, edição e muito mais.

<http://dlabs.4t2.com>
Dexter, um pequeno servidor escrito em Perl pode ser usado como servidor proxy para o MySQL ou como um extensor do banco de dados.

<http://www.scibit.com/Products/Software/Utils/Mascon.asp>
Mascon é um poderoso GUI Win32 para administrar MySQL bancos de dados.

<http://www.rtlabs.com/>
MacSQL Monitor. Gui para Bancos de dados MySQL, ODBC e JDBC para o MAC OS.

Distribuições que incluem o MySQL

<http://www.suse.com/>

SuSE Linux (6.1 e superior)

<http://www.redhat.com/>

RedHat Linux (7.0 e superior)

<http://distro.conectiva.com.br>

Conectiva Linux (4.0 e superior)

Ferramentas de Desenvolvimento Web que suportam MySQL

<http://www.php.net/>

PHP: Uma linguagem script do lado do servidor embutida em HTML.

<http://www.midgard-project.org>

The Midgard Application Server; um ambiente de desenvolvimento Web poderoso baseado em MySQL e PHP.

<http://www.smartworker.org>

SmartWorker é uma plataforma para desenvolvimento de aplicações Web.

<http://xsp.lentus.se/>

XSP: e(X)tendible (S)erver (P)ages e é uma linguagem embutida em tags HTML escrito em Java (anteriormente conhecido como XTAGS.)

<http://www.dbServ.de/>

dbServ é uma extensão para um servidor web para integrar a saída do banco de dados em seu código HTML. Você pode usar qualquer função HTML em sua saída. Somente o cliente poderá te interromper. Funciona como um servidor isolado ou como servlet Java.

<http://www.chilisoft.com/>

Plataforma ASP independente por Chili!Soft

<http://www.voicenet.com/~zellert/tjFM>

Driver JDBC para MySQL.

<http://www.wernhart.priv.at/php/>

Demonstrações de MySQL e PHP.

<http://www.dbwww.com/>

ForwardSQL: Interface HTML para manipular bancos de dados MySQL.

<http://www.daa.com.au/~james/www-sql/>

WWW-SQL: Mostra informações de bancos de dados.

<http://www.minivend.com/minivend/>

Minivend: Um carrinho de compras em Web.

<http://www.heitml.com/>

HeiTML: uma extensão HTML do lado do servidor e uma linguagem 4GL ao mesmo tempo.

- <http://www.metahtml.com/>
Metahtml: Um Linguagem Dinamica de Programação para Aplicações WWW.
- <http://www.binevolve.com/>
VelocityGen para Perl e Tcl.
- <http://hawkeye.net/>
Hawkeye Internet Server Suite.
- <http://www.fastflow.com/>
Network Database Connection para Linux
- <http://www.wdbi.net/>
WDBI: Navegador web como um front end universal para banco de dados que suportam MySQL.
- <http://www.webgroove.com/>
WebGroove Script: compilador HTML e linguagem script do lado do servidor.
- <http://www.ihtml.com/>
Uma linguagem script do lado do servidor.
- <ftp://ftp.igc.apc.org/pub/myodbc/README>
Como usar o MySQL com ColdFusion no Solaris.
- <http://calistra.com/MySQL/>
Administrador ODBC MySQL Calistra.
- <http://www.webmerger.com>
Webmerger - Esta ferramenta CGI le arquivos e gera uma saída dinamica baseada em um conjunto de tags simple. Drivers prontos para MySQL e PostgreSQL através do ODBC.
- <http://phpclub.net/>
PHPclub - Dicas e truques para o PHP.
- <http://www.penguinservices.com/scripts>
Scripts MySQL e Perl.
- <http://www.widgetchuck.com>
The Widgetchuck; Web site com ferramentas.
- <http://www.adcycle.com/>
AdCycle - Software gerenciador de anuncios.
- <http://sourceforge.net/projects/pwpage/>
pwPage - fornece uma ferramenta extremamente rápida e simples para a criação de formulario de banco de dados. Isto é, se existe uma tabela de banco de dados e uma pagina HTML foi construida usando poucas linhas pwPage pode ser imediatamente usado para seleção, inserção, atualização, deleção de dados da tabela e revisão do conteúdo da tabela selecionada.
- <http://www.omnis-software.com/products/studio/studio.html>
OMNIS Studio é uma ferramenta de desenvolvimento rápido de aplicações (RAD).

<http://www.webplus.com>

talentsoft Web+ 4.6 - uma completa e poderosa linguagem de desenvolvimento para usar na criação de aplicações cliente/servidor com base na web sem escrever programas CGI complicados, baixo-nível e que consomem tempo.

Ferramentas de criação de bancos de dados com Suporte MySQL

<http://www.mysql.com/documentation/dezign/>

"DeZign for databases" é uma ferramenta de desenvolvimento de banco de dados que usa diagramas de relacionamento de entidades (ERD).

Servidores Web com Ferramentas MySQL

<ftp://ftp.kcilink.com/pub/>

mod_auth_mysql, Modulo de autenticação para o Apache.

<http://www.roxen.com/>

O Servidor Web Roxen Challenger.

Extensões para outros programas.

http://www.seawood.org/msql_bind/

Suporte MySQL para o BIND (O servidor de nomes da Internet).

<http://www.inet-interactive.com/sendmail/>

Suporte MySQL para o Sendmail e Procmil.

Utilizando o MySQL com outros programas

<http://www.iserver.com/support/addonhelp/database/mysql/msaccess.html>

Utilizando o MySQL com o MS Access.

<http://www.iserver.com/support/contrib/perl5/modules.html>

Instalando novos módulos Perl que exigem módulos instalados localmente.

Links relacionados ao ODBC

<http://www.iodbc.org/>

Gerenciados do driver iODBC popular (libiodbc) agora disponível como Open Source.

<http://users.ids.net/~bjepson/freeODBC/>

As páginas do FreeODBC.

<http://genix.net/unixODBC/>

Os objetivos do projeto unixODBC são desenvolver e promover o unixODBC para ser o padrão definitivo para ODBC na plataforma Linux. Isso inclui suportar suporte GUI para o KDE.

<http://www.sw-soft.com/products/BtrieveODBC/>

Um driver ODBC baseado em MySQL para o Btrieve.

Links relacionados à API

<http://www.jppp.com/>

Componentes para MySQL compatíveis com TDataset parcialmente implementados.

<http://www.riverstyx.net/qpopmysql/>

qpopmysql - Um patch que permite autenticações POP3 para um banco de dados MySQL. Existe também um link para um patch de Paul Khavkine para o Procmail para permitir a qualquer MTA a fazer a entrega a usuarios em um banco de dados MySQL.

<http://www.pbc.ottawa.on.ca>

Gerador de classes Visual Basic para o Active X.

<http://www.essencesw.com/Software/mysql-lib.html>

Novas bibliotecas clientes para o Mac OS Classic (Macintosh).

<http://www.lilback.com/macsql/>

Bibliotecas-cliente para o Macintosh.

<http://www.essencesw.com/Plugins/mysqlplug.html>

Plugin para o REALbasic (para Macintosh)

<http://www.iis.ee.ethz.ch/~neeri/macintosh/gusi-qa.html>

Uma biblioteca que emula sockets BSD e pthreads no Macintosh. Ela pode ser usada se você quiser compilar a biblioteca do cliente MySQL para Mac. Ela provavelmente pode ser usada para portar o MySQL para o Macintosh, mas não conhecemos ninguém que tenha tentado isto.

<http://www.dedecker.net/jessie/scmdb/>

SCMDB - Um add-on para SCM que porta a biblioteca C do MySQL para scheme (SCM). Com esta biblioteca desenvolvedores do scheme podem fazer conexões para um banco de dados MySQL e usar SQL embutido em seus programas.

Outros Links Relacionados ao MySQL

SAT

A Small Application Toolkit (SAT) é uma coleção de utilitários com intenção de simplificar o desenvolvimento de aplicações pequenas, multiusuários e com base em GUI em um ambiente (Microsoft -ou- X) Cliente Windows / Servidor Unix.

<http://www.wix.com/mysql-hosting/>

Registro de provedores Web que suportam o MySQL.

<http://www.softagency.co.jp/mysql/index.en.html>

Links sobre usar o MySQL no Japão/Ásia.

<http://abattoir.cc.ndsu.nodak.edu/~nem/mysql/udf/>

Registro UDF MySQL.

<http://www.open.com.au/products.html>

Sistema Web Comercial de rastreamento de defeitos

<http://www.stonekeep.com/pts/>

PTS: Projeto de Sistema de Rastreamento.

<http://tomato.nvgc.vt.edu/~hroberts/mot>

Serviço e software de sistema de rastreamento.

<http://www.cynergi.net/exportsql/>

ExportSQL: Um script para exportar dados do Access95+.

<http://SAL.KachinaTech.COM/H/1/MYSQL.html>

Entrada MySQL SAL - Aplicações Científicas no Linux (Scientific Applications on Linux).

<http://www.infotech-nj.com/itech/index.shtml>

Uma empresa de consultoria que menciona o MySQL nos direitos da companhia.

<http://www.pmpcs.com/>

PMP Computer Solutions. Desenvolvedores de banco de dados usando MySQL and mSQL.

<http://www.aewa.org/>

Airborne Early Warning Association.

<http://www.dedserious.com/y2kmatrix/>

Testador Y2K.

Interfaces SQL e de Banco de Dados

<http://java.sun.com/products/jdbc/>

A API do JDBC para acesso a banco de dados.

<http://www.gagme.com/mysql>

Patch para mSQL Tcl.

<http://www.amsoft.ru/easysql/>

EasySQL: Um driver de gerenciamento como o ODBC.

<http://www.lightlink.com/hessling/rexxsql.html>

Um interface REXX para banco de dados SQL.

<http://www.mytcl.cx/>

Interface Tlc baseada em tcl-sql com muitos erros corrigidos.

<http://www.binevolve.com/~tdarugar/tcl-sql/>

Interface Tcl.

<http://www.contrib.andrew.cmu.edu/~shadow/sql.html>

Página de Referência SQL com vários links interessantes.

Exemplos de utilizações do MySQL

<http://www.little6.com/about/linux/>

Little6 Inc., Um <<<...online contract and job finding site...>>> que utiliza MySQL, PHP3, e Linux.

- <http://www.delec.com/is/products/prep/examples/BookShelf/index.html>
DELECis - Uma Ferramenta que torna muito fácil a criação de documentação de tabelas gerada automaticamente. Eles têm usado o MySQL como exemplo.
- <http://www.worldrecords.com>
World Records - Um mecanismo de pesquisa para informações sobre música que utiliza o MySQL e PHP.
- <http://www.webtechniques.com/archives/1998/01/note/>
Um banco de dados de contatos usando MySQL e PHP.
- <http://modems.rosenet.net/mysql/>
Calendário comunitário em PHP com interface Web.
- <http://www.odbsoft.com/cook/sources.htm>
Pacote Perl para gerar páginas html de uma estrutura de tabelas SQL e instruções SQL de um formulário HTML.
- <http://www.gusnet.cx/proj/telsql/>
Banco de dados básico de telefones usando DBI/DBD.
- <http://tecfa.unige.ch/guides/java/staf2x/ex/jdbc/coffee-break>
Exemplos JDBC por Daniel K. Schneider.
- <http://www.spade.com/linux/howto/PostgreSQL-HOWTO-41.html>
SQL BNF
- <http://www.ooc.com/>
Object Oriented Concepts Inc; Aplicações CORBA com exemplos em fontes.
- <http://www.pbc.ottawa.on.ca/>
DBWiz: Fornece um exemplo de como gerenciar cursores em VB.
- <http://keilor.cs.umass.edu/pluribus/>
Pluribus é um mecanismo de busca gratuito que melhora a qualidade de seus resultados com o tempo. Pluribus funciona através da gravação de qual das páginas um usuário selecionou entre aquelas retornadas pela consulta. Um usuário vota na página ao selecioná-la; Pluribus utiliza então o seu conhecimento para melhorar a qualidade dos resultados quando outra pessoa envia a mesma consulta (ou similar). Usa PHO e MySQL.
- <http://www.stopbit.com/>
Stopbit -Um site de notícias sobre tecnologia usando MySQL e PHP.
- <http://www.linuxsupportline.com/~kalendar/>
Gerenciador de datas para KDE - O gerenciador possui suporte monousuário (baseado e arquivo) e multiusuário (banco de dados MySQL).
- <http://tim.desert.net/~tim/imger/>
Exemplo de armazenamento/recuperação de imagens com MySQL e CGI.
- <http://www.penguinservices.com/scripts>
Sistema de Carrinho de Compras Online.

<http://www.city-gallery.com/album/>

Old Photo Album - O album é um projeto de historico de fotografia com colaboração popular que gera todas a paginas através de dados armazenados em banco de dados MySQL. As páginas são geradas dinamicamente através de uma interface php3 com o conteúdo do banco de dados. Utiliza imagens e descrições. As imagens são armazenadas em um servidor web para evitar armazená-las em um banco de dados como BLOBs. Todas as outras informações são armazenadas no no servidor MySQL compartilhado.

Links Gerais de Banco de Dados

<http://www.pcslink.com/~ej/dbweb.html>

Database Jump Site

<http://black.hole-in-the.net/guy/webdb/>

Homepage da lista de discussão webdb-l (Web Databases).

<http://www.symbolstone.org/technology/perl/DBI/index.html>

Página dos módulos DBI/DBD do Perl.

<http://www.student.uni-koeln.de/cygwin/>

Ferramentas Cygwin. Unix em cima do Windows.

<http://dbasecentral.com/>

dbasecentral.com; Desenvolvimento e distribuição de sistemas e aplicativos de banco de dados poderosos e fáceis de usar.

<http://www.tek-tips.com/>

Tek-Tips Forums são mais de 800 foruns de suporte p2p não comerciais e independentes voltado à profissionais da Computação. Recursos incluem notificação automática para respostas, uma biblioteca de links, e proteção confidenciais para foruns particulares.

<http://www.public.asu.edu/~peterjn/btree/>

Arvores B: Estrutura de Dados em árvore balanceada.

<http://www.fit.qut.edu.au/~maire/baobab/lecture/sld001.htm>

Conferência sobre Arvores B.

Existem também outros diversos sites Web que utilizam o MySQL. See [\[Users\]](#), [page](#). Envie sugestões para esta lista no email webmaster@mysql.com. Estamos pedindo que você disponibilize uma logomarca do MySQL em algum lugar de seu site se você desejar ter sua página citada aqui. Isto também é válido para ter sua página adicionada na pagina “ferramentas utilizadas” ou algo parecido.

1.2.4 Listas de Discussão MySQL

Esta seção traz a você as listas de discussão do MySQL e da algumas dicas de como utilizá-las

1.2.4.1 As Listas de Discussão MySQL

Para se inscrever na principal lista de discussão sobre MySQL, envie uma mensagem para o endereço de correio eletrônico mysql-subscribe@lists.mysql.com.

Para cancelar a sua inscrição na principal lista de discussão sobre MySQL envie uma mensagem para o endereço de correio eletrônico mysql-unsubscribe@lists.mysql.com.

Somente o endereço para o qual você envia a mensagem é significativa. O assunto e o corpo da mensagem são ignorados.

Se o seu endereço de resposta não for válido, você pode especificar o seu endereço explicitamente, adicionando um hífen ao comando de inscrição ou cancelamento, seguido pelo seu endereço com o caracter '@' do seu endereço substituído por um '='. Por exemplo, para inscrever `seu_nome.dominio` envie uma mensagem para `mysql-subscribe-seu_nome=host.dominio@lists.mysql.com`

Mensagens para mysql-subscribe@lists.mysql.com ou mysql-unsubscribe@lists.mysql.com são processadas automaticamente pelo processador de listas de discussão ezmlm. Informações sobre ezmlm estão disponíveis no [Website do ezmlm](#).

Para enviar uma mensagem para a lista, envie para mysql@lists.mysql.com. No entanto, *não* enviem mensagens de inscrição e cancelamento no mysql@lists.mysql.com por favor, porque qualquer mensagem enviada para o endereço será distribuída automaticamente para milhares de outros usuarios.

Seu site local pode ter muitas inscrições para mysql@lists.mysql.com. Neste caso, ele pode ter uma lista de discussão local, assim as mensagens enviadas para lists.mysql.com para seu site são propagadas para a lista local. Nestes casos, por favor, contate seu administrador de sistema para adicionado ou excluído da lista local do MySQL.

Se você quiser que as mensagens da lista de discussão sejam encaminhadas para uma caixa de correio separada no seu programa de emails, configure um filtro com base nos cabeçalhos das mensagens. Você pode também usar os cabeçalhos `List-ID:` ou `Entregar-Para:` para identificar suas mensagens.

Existe também as seguintes listas de discussão sobre MySQL atualmente:

announce-subscribe@lists.mysql.com (anuncio)

Esta é para anuncio de novas versões do MySQL e programas relacionados. Esta é uma lista com baixo volume na qual todos usuarios do MySQL deveriam se inscrever.

mysql-subscribe@lists.mysql.com (mysql)

A principal lista para discussões MySQL em geral. Note que alguns tópicos são mais bem discutidos em listas mais especializadas. Se você enviar para a lista errada você pode não obter resposta.

mysql-digest-subscribe@lists.mysql.com (mysql-digest)

A lista mysql na forma resumida. Isto significa que você irá receber todas mensagens individuais, enviadas na forma de uma grande mensagem uma única vez ao dia.

bugs-subscribe@lists.mysql.com (bugs)

Nesta lista você somente deve postar um relato de bug completo que acontece de maneira repetitiva usando o script `mysqlbug` (se você estiver executando

no Windows, você deverá incluir uma descrição do sistema operacional e a versão do MySQL). Preferencialmente, você deve testar o problema usando a última versão estável ou de desenvolvimento do MySQL antes de postar um erro! Qualquer um pode repetir o erro apenas usando `mysql test < script` no caso do teste incluído. Todos os erros enviados para esta lista deverão ser corrigidos ou documentados na próxima release do MySQL! Se a solução envolve apenas a alteração de pequena parte do código, será enviado um patch corrigindo o problema.

bugs-digest-subscribe@lists.mysql.com (bugs-digest)

Uma versão resumida da lista bugs.

internals-subscribe@lists.mysql.com (internals)

Uma lista para pessoas que trabalham no código do MySQL. Nesta lista pode-se discutir desenvolvimento do MySQL e pos-patches.

internals-digest-subscribe@lists.mysql.com (internals-digest)

Uma versão resumida da lista internals.

java-subscribe@lists.mysql.com (java)

Discussão sobre MySQL e Java. Maioria sobre o driver JDBC..

java-digest-subscribe@lists.mysql.com java-digest

Uma versão resumida da lista java.

win32-subscribe@lists.mysql.com (win32)

Todas as questões relacionada ao MySQL em sistemas operacionais Microsoft, como o Win95, Win98, NT e Win2000.

win32-digest-subscribe@lists.mysql.com (win32-digest)

Uma versão resumida da lista win32.

myodbc-subscribe@lists.mysql.com (myodbc)

Tudo sobre conectividade do MySQL com ODBC.

myodbc-digest-subscribe@lists.mysql.com (myodbc-digest)

Uma versão resumida da lista myodbc.

plusplus-subscribe@lists.mysql.com (plusplus)

Tudo relacionado à programação da API C++ para o MySQL.

plusplus-digest-subscribe@lists.mysql.com (plusplus-digest)

Uma versão resumida da lista plusplus.

msql-mysql-modules-subscribe@lists.mysql.com (msql-mysql-modules)

Lista sobre o Suporte MySQL no Perl. msql-mysql-modules

msql-mysql-modules-digest-subscribe@lists.mysql.com
(msql-mysql-modules-digest)

Lista resumida sobre a versão do msql-mysql-modules.

Você se inscreve ou cancela a assinatura para todas listas do mesmo jeito que foi descrito acima. Na sua mensagem de assinatura ou cancelamento, coloque apenas o nome da lista apropriada. Por exemplo, para assinar ou cancelar sua assinatura da

lista `myodbc`, envie uma mensagem para `myodbc-subscribe@lists.mysql.com` ou `myodbc-unsubscribe@lists.mysql.com`.

Se você não obtiver uma resposta para suas questões na lista de mensagens, uma opção é pagar pelo suporte da MySQL AB, que irá colocar você em contato direto com desenvolvedores MySQL. See `<undefined>` [Support], page `<undefined>`.

A seguinte tabela mostra algumas listas de mensagens sobre o MySQL que utilizam línguas diferentes do Inglês. Perceba que elas não são operadas pela MySQL AB, portanto, não podemos garantir a qualidade destas.

`mysql-france-subscribe@yahoogroups.com` Lista de mensagens na língua francesa.

`list@tinc.net` Lista de mensagens coreana.
Envie `subscribe mysql your@email.address` para esta lista.

`mysql-de-request@lists.4t2.com` Lista de mensagens alemã.
Envie `subscribe mysql-de your@email.address` para esta lista.
Você pode encontrar informações sobre esta lista de mensagens em <http://www.4t2.com/mysql>.

`mysql-br-request@listas.linkway.com.br` Lista de mensagens em português Envie `subscribe mysql-br your@email.address` para esta lista.

`mysql-alta@elistas.net` Lista de mensagens espanhola.
Envie `subscribe mysql your@email.address` para esta lista.

1.2.4.2 Fazendo perguntas ou relatando erros

Antes de enviar um relato de erro ou uma questão, por favor faça o seguinte:

- Comece pesquisando o manual MySQL online em:
<http://www.mysql.com/documentation/manual.php>
Nós tentaremos manter o manual atualizado, frequentemente atualizando-o com soluções para novos problemas encontrados!
- Pesquise os arquivos das listas de mensagens MySQL:
<http://www.mysql.com/documentation/>
- Você pode também usar a página <http://www.mysql.com/search.html> para pesquisar todas as páginas Web (incluindo o manual) que estão localizados em <http://www.mysql.com/>.

Se você não puder encontrar uma resposta no manual ou nos arquivos, confira com seu expert em MySQL local. Se você continua não encontrando uma resposta para sua questão, vá em frente e leia a próxima seção sobre como enviar email para `mysql@lists.mysql.com`.

1.2.4.3 Como relatar erros ou problemas

Escrever um bom relatório de erro exige paciência, e fazê-lo de forma apropriada economiza tempo para nós e para você. Um bom relatório de erros contendo um teste de caso para o bug irá torná-lo muito mais fácil para corrigi-lo no próximo release. Esta seção irá ajudá-lo

a escrever seu relatório corretamente para que você não perca seu tempo fazendo coisas que não irão ajudar-nos muito ou nada.

Nós encorajamos todo mundo a usar o script `mysqlbug` para gerar um relato de erros (ou um relato sobre qualquer problema), se possível. `mysqlbug` pode ser encontrado no diretório `'scripts'` na distribuição fonte, ou, para uma distribuição binária, no diretório `'bin'` no diretório de instalação do MySQL. Se você não puder utilizar o `mysqlbug`, você pode continuar incluindo todas as informações necessárias listadas nesta seção.

O script `mysqlbug` lhe ajudará a gerar um relatório determinando muitas das seguintes informações automaticamente, mas se alguma coisa importante estiver faltando, por favor forneça-o junto de sua mensagem! Por favor leia esta seção com cuidado e tenha certeza que todas as informações descritas aqui estão incluídas no seu relatório.

O lugar comum para relatar erros e problemas é mysql@lists.mysql.com. Se você fizer um caso com testes que demonstre o bug claramente, pode postá-lo na lista bugs@lists.mysql.com. Note que nesta lista você deverá postar somente um relatório completo, usando o script `mysqlbug`. Se você estiver trabalhando com o MySQL para Windows, você deve incluir uma descrição do sistema operacional e a versão do MySQL. Preferencialmente você deve testar o problema usando a última versão estável ou de desenvolvimento do MySQL antes de postar ! Qualquer um deve conseguir repetir o erro usando apenas `"mysql test < script"` no teste do caso incluso, ou executando o shell ou utilizando scripts perl inclusos no relatório de erro. Todos os erros postados na lista `bugs` serão corrigidos ou documentados na próxima release do MySQL! Se foram necessárias apenas pequenas alterações no código para resolver este problema, nós iremos publicar um patch que corrige o problema.

Lembre-se que é possível responder a uma mensagem contendo muita informação, mas não a uma contendo muito pouca. Frequentemente pessoas omitem fatos porque acreditam que conhecem a causa do problema e assumem que alguns detalhes não importam. Um bom princípio é: Se você está em dúvida sobre declarar alguma coisa, declare-a ! É milhares de vezes mais rápido e menos problemático escrever um pouco de linhas a mais no seu relatório do que ser forçado a perguntar de novo e esperar pela resposta porque você não forneceu informação suficiente da primeira vez.

Os erros mais comuns acontecem porque as pessoas não indicam o número da versão da distribuição do MySQL que estão usando, ou não indicam em qual plataforma elas tem o MySQL instalado (Incluindo o número da versão da plataforma). Essa informação é muito relevante, e em 99% dos casos o relato de erro é inútil sem ela! Frequentemente nós recebemos questões como, "Por que isto não funciona para mim?" então nós vemos que aquele recurso requisitado não estava implementado naquela versão do MySQL, ou que o erro descrito num relatório foi resolvido em uma versão do MySQL mais nova. Algumas vezes o erro é dependente da plataforma; nesses casos, é quase impossível corrigir alguma coisa sem conhecimento do sistema operacional e o número da versão da plataforma.

Lembre-se também de fornecer informações sobre seu compilador, se isto for relacionado ao problema. Frequentemente pessoas encontram erros em compiladores e acreditam que o problema é relacionado ao MySQL. A maioria dos compiladores estão sobre desenvolvimento todo o tempo e tornam-se melhores a cada versão. Para determinar se o seu problema depende ou não do compilador, nós precisamos saber qual compilador foi usado. Note que todo problema de compilação deve ser estimado como relato de erros e, conseqüentemente publicado.

É de grande ajuda quando uma boa descrição do problema é incluída no relato do erro. Isto é, um bom exemplo de todas as coisas que o levou ao problema e a correta descrição do problema. Os melhores relatórios são aqueles que incluem um exemplo completo mostrando como reproduzir o erro ou o problema See [\[Reproducible test case\]](#), page [\[undefined\]](#).

Se um programa produz uma mensagem de erro, é muito importante incluir essas mensagens no seu relatório! Se nós tentarmos procurar por algo dos arquivos usando programas, é melhor que as mensagens de erro relatadas sejam exatamente iguais a que o programa produziu. (Até o caso deve ser observado!) Você nunca deve tentar lembrar qual foi a mensagem de erro; e sim, copiar e colar a mensagem inteira no seu relatório!

Se você tem um problema com o MyODBC, você deve tentar gerar um arquivo para rastreamento de erros (trace) do MyODBC. See [\[MyODBC bug report\]](#), page [\[undefined\]](#).

Por favor lembre-se que muitas das pessoas que lerão seu relatório podem usar um vídeo de 80 colunas. Quando estiver gerando relatórios ou exemplos usando a ferramenta de linha de comando `mysql`, então deverá usar a opção `--vertical` (ou a instrução terminadora `\G`) para saída que irá exceder a largura disponível para este tipo de vídeo (por exemplo, com a instrução `EXPLAIN SELECT`; veja exemplo abaixo).

Por favor inclua a seguinte informação no seu relatório:

- O número da versão da distribuição do MySQL que está em uso (por exemplo, MySQL Version 3.22.22). Você poderá saber qual versão vocês está executando, usando o comando `mysqladmin version`. `mysqladmin` pode ser encontrado no diretório 'bin' sob sua instalação do MySQL.
- O fabricante e o modelo da máquina na qual você está trabalhando.
- O nome do sistema operacional e a versão. Para a maioria dos sistemas operacionais, você pode obter esta informação executando o comando Unix `uname -a`.
- Algumas vezes a quantidade de memória (real e virtual) é relevante. Se estiver em dúvida, inclua esses valores.
- Se você estiver usando uma distribuição fonte do MySQL, é necessário o nome e número da versão do compilador usado. Se você estiver usando uma distribuição binária, é necessário o nome da distribuição.
- Se o problema ocorre durante a compilação, inclua a(s) exata(s) mensagem(s) de erro(s) e também algumas linhas do contexto envolvendo o código no arquivo onde o erro ocorreu.
- Se o `mysqld` finalizou, você deverá relatar também a consulta que travou o `mysqld`. Normalmente você pode encontrar isto executando `mysqld` com o log habilitado. See [\[Using log files\]](#), page [\[undefined\]](#)
- Se alguma tabela do banco de dados estiver relacionado ao problema, inclua a saída de `mysqldump --nodata nome_db nome_tb11 nome_tb12...`. Isto é muito fácil de fazer e é um modo poderoso de obter informações sobre qualquer tabela em um banco de dados que irá ajudar-nos a criar uma situação parecida da que você tem.
- Para problemas relacionados à velocidade ou problemas com instruções `SELECT`, você sempre deve incluir a saída de `EXPLAIN SELECT ...` e ao menos o número de linhas que a instrução `SELECT` produz. Quanto mais informação você fornecer sobre a sua

situação, mais fácil será para alguém ajudar-lo! A seguir um exemplo de um relatório de erros muito bom (ele deve ser postado com o script `mysqlbug`):

Exemplo de execução usando a ferramenta de linha de comando `mysql` (perceba o uso do instrução terminadora `\G` para instruções cuja largura de saída deva ultrapassar 80 colunas):

```
mysql> SHOW VARIABLES;
mysql> SHOW COLUMNS FROM ... \G
      <saída para SHOW COLUMNS>
mysql> EXPLAIN SELECT ... \G
      <saída para EXPLAIN>
mysql> FLUSH STATUS;
mysql> SELECT ...;
      <Uma pequena versão da saída do SELECT,
      incluindo a hora em que a consulta foi executada>
mysql> SHOW STATUS;
      <saída do SHOW STATUS>
```

- Se um erro ou problema ocorrer quando estiver executando o `mysqld`, tente fornecer um script de entrada que irá reproduzir a anomalia. Este script deve incluir qualquer arquivo de fonte necessário. Quanto mais próximo o script puder reproduzir sua situação, melhor. Se você puder fazer uma série de testes repetidos, você poderá postá-lo para bugs@lists.mysql.com para um tratamento de alta prioridade!

Se não puder fornecer o script, você ao menos deve incluir a saída de `mysqladmin variables extended-status processlist` na sua mensagem para fornecer alguma informação da performance do seu sistema.

- Se você não puder produzir um caso de teste em algumas linhas, ou se a tabela de testes for muito grande para ser enviada por email para a lista de mensagens (mais de 10 linhas), você deverá dar um dump de suas tabelas usando o `mysqldump` e criar um arquivo 'README' que descreve seu problema.
Crie um arquivo comprimido de seus arquivos usando `tar` e `gzip` ou `zip`, e use o `ftp` para transferir o arquivo para <ftp://support.mysql.com/pub/mysql/secret/>. E envie uma pequena descrição do problema para bugs@lists.mysql.com.
- Se você achar que o MySQL produziu um resultado estranho para uma consulta, não inclua somente o resultado, mas também sua opinião de como o resultado deve ser, e uma conta descrevendo o base de sua opinião.
- Quando fornecer um exemplo do problema, é melhor usar os nomes de variáveis, nomes de tabelas, etc. utilizados na sua situação atual do que enviar com novos nomes. O problema pode ser relacionado ao nome da variável ou tabela! Esses casos são raros, mas é melhor prevenir do que remediar. Além disso, será mais fácil para você fornecer um exemplo que use sua situação atual, que é o que mais importa para nós. No caso de ter dados que não deseja mostrar para outros, você pode usar o `ftp` para transferi-lo para <ftp://support.mysql.com/pub/mysql/secret/>. Se os dados são realmente confidenciais, e você não deseja mostrá-los nem mesmo para nós, então vá em frente e providencie um exemplo usando outros nome, mas, por favor considere isso como uma única chance.

- Inclua, se possível, todas as opções fornecidas aos programas relevantes. Por exemplo, indique as opções que você utiliza quando inicializa o daemon `mysqld` e aquelas que são utilizadas para executar qualquer programa cliente MySQL. As opções para programas como o `mysqld` e `mysql`, e para o script `configure`, são frequentemente chaves para respostas e são muito relevantes! Nunca é uma má idéia incluí-las de qualquer forma! Se você usa algum módulo, como Perl ou PHP por favor forneça o número da versão deles também.
- Se sua questão é relacionada ao sistema de privilégios, por favor forneça a saída de `mysqlaccess`, a saída de `mysqladmin reload`, e todas as mensagens de erro que você obteve quando tentava conectar! Quando você testar seus privilégios, você deve primeiramente executar `mysqlaccess`. Depois, execute `mysqladmin reload version` e tente conectar com o programa que gerou o problema. `mysqlaccess` pode ser encontrado no diretório `'bin'` sob seu diretório de instalação do MySQL.
- Se você tiver um patch para um erro, isso é bom, mas não assuma que o patch é tudo que precisamos, ou que iremos usá-lo, se você não fornecer algumas informações necessárias, como os casos de testes mostrando o erro que seu patch corrige. Nós podemos encontrar problemas com seu patch ou nós podemos não entendê-lo ao todo; se for assim, não podemos usá-lo.

Se nós não verificarmos exatamente o que o patch quer dizer, nós não poderemos usá-lo. Casos de testes irão ajudar-nos aqui. Mostre que o patch irá cuidar de todas as situações que possam ocorrer. Se nós encontrarmos um caso (mesmo que raro) onde o patch não funcionaria, ele pode ser inútil.

- Palpites sobre o que o erro pode ser, porque ocorre, ou do que ele depende, geralmente estão errados. Mesmo o time MySQL não pode adivinhar antecipadamente tais coisas sem usar um debugger para determinar a causa real do erro.
- Indique na sua mensagem de e-mail que você conferiu o manual de referência e o arquivo de mensagens para que outros saibam que você tentou solucionar o problema.
- Se você obter um **parse error**, por favor confira sua sintaxe com atenção! Se você não conseguiu encontrar nada errado com ela, é extremamente provável que sua versão corrente do MySQL não suporte a consulta que você está utilizando. Se você estiver usando a versão recente e o manual em <http://www.mysql.com/documentation/manual.php> não cobrir a sintaxe que você estiver usando, o MySQL não suporta sua consulta. Neste caso, suas únicas opções são implementar você mesmo a sintaxe ou enviar uma mensagem para mysql-licensing@mysql.com e perguntar por uma oferta para implementá-lo!

Se o manual cobrir a sintaxe que você estiver usando, mas você tiver uma versão mais antiga do MySQL, você deverá conferir o histórico de alterações do MySQL para ver quando a sintaxe foi implementada. Neste caso, você tem a opção de atualizar para uma nova versão do MySQL. See [\(undefined\) \[News\]](#), page [\(undefined\)](#).

- Se você tiver um problema do tipo que seus dados aparecem corrompidos ou você obtém erros quando você acessa alguma tabela em particular, você deverá primeiro checar depois tentar reparar suas tabelas com `myisamchk` ou `CHECK TABLE` e `REPAIR TABLE`. See [\(undefined\) \[MySQL Database Administration\]](#), page [\(undefined\)](#).
- Se você frequentemente obtém tabelas corrompidas, você deve tentar encontrar quando e porque isto acontece! Neste caso, o arquivo

'mysql-data-directory/'hostname'.err' deve conter algumas informações sobre o que aconteceu. See [\[Error log\]](#), page [\[Error log\]](#). Por favor forneça qualquer informação relevante deste arquivo no seu relatório de erro! Normalmente o `mysqld` **NUNCA** deverá danificar uma tabela se nada o finalizou no meio de uma atualização! Se você puder encontrar a causa do fim do `mysqld`, se torna muito mais fácil para nós fornecermos a você uma solução para o problema! See [\[What is crashing\]](#), page [\[What is crashing\]](#).

- Se possível, faça o download e instale a versão mais recente do MySQL para saber se ela resolve ou não o seu problema. Todas versões do MySQL são muito bem testadas e devem funcionar sem problemas! Acreditamos em deixar tudo, o mais compatível possível com as versões anteriores, e você conseguirá mudar de versões MySQL em minutos! See [\[Which version\]](#), page [\[Which version\]](#).

Se você é um cliente de nosso suporte, por favor envie o seu relatório de erros em mysql-support@mysql.com para tratamento de alta prioritário, bem como para a lista de mensagens apropriada para ver se mais alguém teve experiências com (e talvez resolveu) o problema.

Para informações sobre relatar erros no **MyODBC**, veja [\[ODBC Problems\]](#), page [\[ODBC Problems\]](#).

Para soluções a alguns problemas comuns, veja See [\[Problems\]](#), page [\[Problems\]](#).

Quando respostas são enviadas para você individualmente e não para a lista de mensagens, é considerado boa etiqueta resumir as respostas e enviar o resumo para a lista de mensagens para que outras possam ter o benefício das respostas que você recebeu que ajudaram a resolver seu problema!

1.2.4.4 Guia para responder questões na lista de discussão

Se você considerar que sua resposta possa ter um amplo interesse, você pode querer postá-la para a lista de mensagens em vez de responder diretamente para a pessoa que perguntou. Tente deixar sua resposta da forma mais genérica possível para que outras pessoas além da que postou a pergunta possam se beneficiar dela. Quando você postar para a lista, por favor tenha certeza que sua resposta não é uma réplica de uma resposta anterior.

Tente resumir a parte essencial da questão na sua resposta, não se sinta obrigado a citar a mensagem original inteira.

Por favor não poste mensagens a partir de seu browser com o modo HTML ligado! Muitos usuários não leem e-mail com browser!

1.3 Licenciamento e Suporte do MySQL

Esta seção descreve o suporte MySQL e termos de licenciamento:

- Os direitos autorais em que o MySQL é distribuído (see [\[Copyright\]](#), page [\[Copyright\]](#))
- Exemplos de situações ilustrando onde uma licença é necessária (see [\[Licensing examples\]](#), page [\[Licensing examples\]](#))

- Custos de Suporte (see [\(undefined\)](#) [Cost], page [\(undefined\)](#)) e seus benefícios (see [\(undefined\)](#) [Support], page [\(undefined\)](#))
- Custos do licenciamento comercial

1.3.1 Política de licenciamento MySQL

Os termos formais da licença GPL podem ser encontradas em [\(undefined\)](#) [GPL license], page [\(undefined\)](#). Basicamente, nossa política de licenças e interpretação da GPL é desta forma:

Perceba que versões mais antigas do MySQL ainda utilizam uma **licença mais restrita**. Veja a documentação para aquela versão para maiores informações. Se você precisa de uma licença MySQL comercial, porque sua aplicação não se encaixa com a licença GPL, você pode comprar uma em <https://order.mysql.com>

Para uso interno normal, o MySQL não custa nada. Você não tem que nos pagar se não quiser fazê-lo.

Uma licença é necessária se:

- Se você unir um programa, que não é software livre, com código do MySQL server ou clientes que tem o direito autoral baseado na GPL. Isto acontece por exemplo quando você usa o MySQL como um servidor embutido nas suas aplicações ou quando você adiciona extensões não livres para o servidor MySQL. Neste caso, sua aplicação/código irá também se tornar GPL através do esquema GPL que age como um vírus. Licenciando o servidor MySQL da MySQL AB sobre uma licença comercial você irá evitar este problema. Veja <http://www.gnu.org/copyleft/gpl-faq.html>.
- Você tem uma aplicação comercial que trabalha SOMENTE com o MySQL e distribui a aplicação com o servidor MySQL. Isto acontece porque vemos isto como uma união mesmo se é feito através da rede.
- Você tem uma distribuição do MySQL e você não fornece o código fonte para sua cópia do servidor MySQL, como é definido na licença GPL.

NÃO seria necessária uma licença se:

- Não é necessário uma licença para incluir o código cliente em programas comerciais. A parte cliente do MySQL é licenciada com a LGPL GNU Library General Public License. O programa cliente de comandos de linha `mysql` inclui código da biblioteca `readline` que está na GPL.
- Se a forma que você usa o MySQL não necessitar a licença, mas você gosta do MySQL e quer promover e encorajar o desenvolvimento, você certamente será bem vindo para comprar uma licença ou suporte do MySQL.
- Se você usa o MySQL num contexto comercial no qual obtém lucro com seu uso, nós pedimos que você ajude o desenvolvimento do MySQL, comprando algum nível de suporte. Nós sentimos que se o MySQL ajuda seu negócio, é razoável pedir à você que ajude o MySQL. (De outra forma, se você nos perguntar questões de suporte, você não estará somente usando de graça uma coisa em que colocamos muito trabalho, mas, também estará pedindo para nós fornecermos suporte gratuito.)

Para circunstâncias em que uma licença MySQL é necessária, você precisa de uma licença para cada máquina que executar o servidor `mysqld`. Entretanto, uma máquina com vários

processadores conta como uma máquina única, e não existem restrições sobre o número de servidores MySQL que executam em uma máquina ou no número de clientes conectados simultaneamente a um servidor executado naquela máquina!

Se você tiver alguma dúvida se uma licença é necessária ou não para seu uso particular do MySQL, por favor leia isto de novo e contate-nos. See [\[Contact information\]](#), page [\[Contact information\]](#).

Se você necessita de uma licença MySQL, o modo mais fácil para pagar por ele é usar o formulário de licença no servidor seguro da MySQL em <https://order.mysql.com/>. Outras formas de pagamento são discutidas em [\[Payment information\]](#), page [\[Payment information\]](#).

1.3.2 Direitos autorais usados no MySQL

Existem vários direitos autorais diferentes na distribuição do MySQL:

1. O código fonte específico necessário para construir a biblioteca `mysqlclient` é licenciada sobre a LGPL e programas no diretório `'client'` é GPL. Cada arquivo tem um cabeçalho que mostra qual direito autoral é usado para aquele arquivo.
2. A biblioteca cliente e a (GNU `getopt` são cobertas pela “GNU LIBRARY GENERAL PUBLIC LICENSE.” See [\[LGPL license\]](#), page [\[LGPL license\]](#).
3. Algumas partes da fonte (a biblioteca `regex` são cobertas pelos direitos autorais no estilo de Berkeley.
4. Todas as fontes no servidor e a biblioteca (GNU `readline` é coberta pela “GNU GENERAL PUBLIC LICENSE.” See [\[GPL license\]](#), page [\[GPL license\]](#). Isto também está disponível no arquivo `'COPYING'` nas distribuições.

Um objetivo é que a biblioteca cliente SQL deve ser livre o bastante que seja possível adicionar suporte MySQL em produtos comerciais sem uma licença. Para esta razão, nós escolhemos a licença LGPL para o código cliente.

Isto significa que você pode usar o MySQL gratuitamente com qualquer programa que use qualquer tipo de licença livre. O MySQL é também grátis para a utilização por qualquer usuário final para seu uso próprio ou dentro da empresa.

Entretanto, se você usa o MySQL para alguma coisa importante, poderã querer ajudar a segurar seu desenvolvimento comprando licenças ou contrato de suporte. See [\[Support\]](#), page [\[Support\]](#).

1.3.2.1 Possíveis alterações futuras dos direitos autorais

A versão 3.22 do MySQL continua usando uma licença mais restrita. Veja a documentação para aquela versão para maiores informações.

1.3.3 Exemplos de situações de licenciamento

Esta seção descreve algumas situações ilustrando se você deve ou não licenciar o servidor MySQL. Geralmente estes exemplos envolvem o fornecimento do MySQL como uma parte integral do produto.

Perceba que uma licença única do MySQL cobre qualquer número de CPUs e servidores `mysqld` em uma máquina! Não existe limite artificial no número de clientes que conectam ao servidor em nenhum caso.

1.3.3.1 Vendendo produtos que usam o MySQL

Para determinar se você precisa ou não de uma licença MySQL quando vender sua aplicação, você deve se perguntar se a sua aplicação é dependente no uso do MySQL e se você inclui ou não o servidor MySQL com seu produto. Existem diversos casos a considerar:

- Sua aplicação necessita do MySQL para funcionar corretamente ?
- Se seu produto necessita do MySQL, você precisa de uma licença para qualquer máquina que executa o servidor `mysqld`. Por exemplo, se você desenvolveu sua aplicação sobre o MySQL, então você realmente criou um produto comercial que necessita do mecanismo, então precisa de uma licença.
- Se a sua aplicação não necessita do MySQL, você não precisa obter uma licença. Por exemplo, se o uso do MySQL apenas adiciona alguns novos recursos opcionais para seu produto (como gerar log para um banco de dados em vez de simplesmente usar um arquivo texto), ela deve cair no uso normal, e uma licença não é necessária.
- Em outras palavras, você precisa de uma licença se você vender um produto desenvolvido para ser usado especificamente com o MySQL ou que necessita do servidor MySQL para funcionar completamente. Isto é verdadeiro se você fornecer ou não o MySQL para seu cliente como parte da distribuição de seu produto.
- Ela também depende do que você irá fazer pelo cliente. Você planeja fornecer a seu cliente instruções detalhadas sobre como instalar o MySQL com seu software? Então seu produto pode ser contingente no uso do MySQL; se sim, será necessária a compra de uma licença. Se você está simplesmente amarrando em um banco de dados que você espera já estar instalado na época em que seu software foi comprado, então possivelmente não precisa de uma licença.

1.3.3.2 Serviços ISP MySQL

Provedores de serviço de Internet (ISPs) geralmente hospedam servidores MySQL para seus clientes. Com a licença GPL isto não necessita de uma licença.

Por outro lado, nós encorajamos as pessoas a usar ISPs que possuem suporte MySQL, isto dará a eles a confiança de que se eles tiverem algum problema com a sua instalação do MySQL, seu ISP poderá resolver o problema para eles (em alguns casos com a ajuda da equipe de desenvolvimento MySQL).

Todos ISPs que desejam se manter atualizados devem se inscrever na nossa lista de mensagens `announce` para que eles possam ser notificados sobre assuntos que possam ser relevantes para suas instalações MySQL.

Perceba que se o ISP não tem uma licença para o MySQL, ele deve fornecer ao menos acesso de leitura para a fontes da instalação MySQL para que seus clientes possam verificar se foi corretamente corrigido.

1.3.3.3 Executando um servidor web usando o MySQL.

Se você usa o MySQL em conjunto com um servidor Web no Unix, não existe necessidade de pagar para uma licença.

Isto é verdade mesmo se você executa um servidor Web Comercial que usa MySQL, porque você não estará vendendo uma versão do MySQL embutida. Entretanto, neste caso nós gostaríamos que você comprasse suporte MySQL. Porque a MySQL estará ajudando a sua empresa.

1.3.4 Custos de licenciamento e suporte do MySQL

Nossa lista atual de preços para licenças é acessada abaixo. Para fazer uma compra, por favor visite <https://order.mysql.com/>.

Se você pagar com cartão de crédito, a moeda é EURO (European Union Euro) então os preços podem ser um pouco diferentes.

Número de Licenças	Por cópia
1-9	230 EURO
10-24	138 EURO
25-49	117 EURO
50-99	102 EURO
100-249	91 EURO
250-499	76 EURO
500-999	66 EURO

Para um alto volume de compras (OEM), por favor, contate: sales@mysql.com.

Para compras OEM, você deve agir como o intermediário para eventuais problemas ou requisições de seus usuários. Nós também solicitamos que os clientes OEM tenham pelo menos uma contrato estendido de suporte. Note que as licenças OEM somente são aplicadas para produtos onde o usuário não tem acesso direto ao servidor MySQL (Sistema embutido). Em outras palavras, o servidor MySQL deve ser usado com a aplicação que foi fornecida por você.

Se você possui um produto de grande vendagem e baixa margem de lucro, você pode sempre discutir conosco sobre outros termos (por exemplo, um percentual do preço de venda). Se é seu caso, por favor informe seu produto, preço, mercado e qualquer informação que possa ser relevante.

Uma licença paga não é um acordo de suporte e inclui suporte mínimo. Isto significa que nós tentamos responder qualquer questão relevante. Se a resposta existe na documentação, nós iremos direcioná-lo para a seção apropriada. Se você não comprou uma licença ou suporte, provavelmente não iremos respondê-lo.

Se você descobrir o que consideramos de um bug real, iremos corrigí-lo de qualquer maneira. Mas se você paga pelo suporte iremos notificá-lo sobre o estado da correção em vez de apenas consertá-lo em um release posterior.

Suporte mais completo é vendido separadamente. Descrições do que cada nível de suporte oferece é fornecido em [\[Support\]](#), page [\[undefined\]](#). Custos para os vários tipos de suporte comercial são vistos abaixo. Os preços dos níveis de suporte estão em EURO (European Union Euro). Um EURO equivale algo em torno de 1.06 USD.

Tipo de suporte	Custo por ano
Suporte básico por e-mail. See [Basic email support] , page [undefined] .	EURO 200
Suporte estendido por e-mail See [Extended email support] , page [undefined] .	EURO 1000
Suporte com Login See [Login support] , page [undefined] .	EURO 2000
Suporte estendido com Login See [Extended login support] , page [undefined] .	EURO 5000
Suporte telefônico See [Telephone support] , page [undefined] .	EURO 12000

Você pode atualizar de qualquer nível baixo de suporte para um nível mais alto pela diferença de preço entre os dois níveis.

Nós também fornecemos suporte telefônico (na maioria suporte emergencial mas também suporte 24/7). Esta opção de suporte, entretanto, não tem um preço fixo e é negociado para caso a caso. Se você tem interesse nesta opção envie um email para sales@mysql.com e diga-nos sobre suas necessidades.

Entenda que como nossa equipe de vendas é muito ocupada, ela pode tomar algum tempo até que sua requisição seja atendida. Nosso pessoal de suporte entretanto sempre atenderá prontamente às questões de suporte!

1.3.4.1 Informações de pagamento

Atualmente podemos receber pagamento em conta, cheques ou cartões de crédito.

Pagamentos podem ser feitos à:

Postgirot Bank AB
105 06 STOCKHOLM, SWEDEN

MySQL AB
BOX 6434
11382 STOCKHOLM, SWEDEN

Endereço SWIFT : PGSI SESS
Número da conta: 96 77 06 - 3

Especifique: licença e/ou suporte, seu nome e endereço de e-mail.

Na Europa e Japão podem ser usados o EuroGiro (que deve ser mais barato) para a mesma conta.

Se você deseja pagar com cheque, faça-o nominal à “MySQL Finland AB” e envie-o para o endereço abaixo:

MySQL AB
BOX 6434, Torsgatan 21
11382 STOCKHOLM, SWEDEN

Se você deseja pagar com cartão de crédito pela Internet, pode usar [Formulário seguro de licenças da MySQL AB](#).

Você pode também imprimir uma cópia do formulário de licenças, preenchê-lo e enviá-lo por fax para:

+46-8-729 69 05

Se você desejar que nós lhe cobremos, pode usar o formulário de licenças e escrever “bill us” no campo de comentário. Pode também enviar uma mensagem para sales@mysql.com (não mysql@lists.mysql.com!) com as informações de sua empresa e pedir para que nós envie a cobrança.

1.3.4.2 Informações para Contato

Para licenciamento comercial, por favor contate a equipe de licenciamento do MySQL. O método mais preferido é enviar um e-mail para licensing@mysql.com. Envios de Fax também é possível mas pode demorar mais (Fax +46-8-729 69 05).

Se você representa uma empresa que está interessada em ter parcerias com a MySQL, por favor envie um e-mail para partner@mysql.com.

Por enquanto, para respostas precisas para questões técnicas sobre o MySQL você deve **comprar** um de nossos **contratos de suporte**. O Suporte ao MySQL é fornecido pelos desenvolvedores do MySQL portanto a qualidade de atendimento é extremamente alta.

Se você está interessado em colocar um banner de anúncio em nosso site Web, envie um e-mail para advertising@mysql.com.

Se estiver interessado em algum dos empregos listados em nossa seção **empregos**, por favor envie um e-mail para jobs@mysql.com.

Para discussões gerais entre nossos usuários, dirija sua atenção para a **lista de mensagens** apropriada.

Para solicitações de informações gerais, envie e-mail para info@mysql.com.

Para questões ou comentários sobre os trabalhos ou conteúdo de nosso site Web, envie e-mail para webmaster@mysql.com.

1.3.5 Tipos de suporte comercial

Todos os detalhes a seguir fazem parte de todas opções de suporte:

- O suporte é por ano.
- Corrigiremos, ou fornecemos uma forma razoável de contornar a situação para qualquer bug persistente.
- Iremos concentrar um esforço razoável para encontrar e corrigir qualquer outro erro relacionado ao MySQL.
- Quanto maior o nível do contrato de suporte, maior será o esforço para encontrar uma solução para seus problemas.
- Os seguintes itens fazem parte de todos contratos de suporte exceto do suporte básico por email:

Qualquer coisa não relacionadas a erros, como ajudar a otimizar suas queries ou seus sistema, estendendo o MySQL com novas funcionalidades, etc., cobramos 200 EUROS/hora. que é deduzido de seu contrato de suporte. Em outras palavras, se você tem suporte de login (2000 EURO) pode esperar que trabalharemos por volta de 10 horas para ajudar a você com coisas deste tipo.

1.3.5.1 Suporte básico por E-mail

Suporte básico por e-mail é uma opção de suporte muito barata e deve ser entendida mais como um meio de apoiar o desenvolvimento do MySQL do que como uma opção real de suporte. Nós da MySQL fornecemos vários suportes gratuitos em todas as listas diferentes do MySQL, e o dinheiro que obtemos com o suporte básico por e-mail é amplamente usado para tornar isto possível.

Neste nível de suporte, as listas de mensagens do MySQL são métodos preferidos de comunicação. Questões normalmente devem ser enviadas para a lista de mensagens principal (mysql@lists.mysql.com) ou uma de nossas listas regulares (por exemplo, win32@lists.mysql.com para questões relacionadas ao MySQL no windows), como outra pessoa pode ter experimentado o mesmo e resolvido o mesmo problema que você tem. See [\[Asking questions\]](#), page [\[undefined\]](#).

Entretando, comprando suporte básico por e-mail, você também tem acesso ao endereço de suporte mysql-support@mysql.com, que não é disponível como parte do suporte mínimo que você obtêm comprando uma licença do MySQL. Isto significa que para questões especialmente críticas, você também pode enviar a sua mensagem para mysql-support@mysql.com. (Se a mensagem conter dados sensíveis, você deve postar somente para mysql-support@mysql.com.)

LEMBRE-SE! de sempre incluir seu número de registro e data de expiração quando você enviar uma mensagem para mysql-support@mysql.com.

Perceba que se você encontrou um bug crítico e repetitivo e seguiu as regras da seção do manual de como relatar erros e enviá-los para bugs@lists.mysql.com, prometemos tentar corrigí-lo assim que possível, sem levar em consideração seu nível de suporte! See [\[Bug reports\]](#), page [\[undefined\]](#).

Suporte básico por e-mail inclui os seguintes tipos de serviço:

- Se sua dúvida já está solucionada no manual, iremos informá-lo da seção correta em que você pode achar a resposta. Se a resposta não está no manual, nós iremos apontar você na direção certa para resolver seu problema.
- Nós garantimos uma resposta para suas mensagens de e-mail. O que não garantimos é que podemos resolver qualquer problema, pelo menos você receberá uma resposta se pudermos lhe contactar em seu e-mail.
- Iremos ajudar com problemas inesperados quando você instalar o MySQL de uma distribuição binária em plataformas suportadas. Este nível de suporte não cobre a instalação do MySQL de uma distribuição fonte. Plataformas suportadas são aquelas que já são conhecidas em que o MySQL funciona. See [\[Which OS\]](#), page [\[undefined\]](#).
- Nós iremos ajudá-lo com bugs e recursos em falta. Qualquer bugs encontrados são corrigidos no próximo release do MySQL. Se o bug é crítico para você, enviaremos um e-mail com um patch logo que o bug for corrigido. Bugs Críticos sempre tem prioridade alta para nós e garantimos que ele será corrigido logo que possível.
- Suas sugestões para favorecer o desenvolvimento do MySQL serão levadas em consideração. Obtendo o suporte por email você já está ajudando o desenvolvimento do MySQL. Se você desejar mais integração, atualize para um nível mais alto do suporte.

- Se você deseja que nós ajudemos a otimizar seu sistema, deve atualizar para um nível mais alto de suporte.

1.3.5.2 Suporte estendido por E-mail

O suporte estendido por e-mail inclui todos os itens do suporte básico por e-mail com estas vantagens:

- Seu e-mail será atendido antes dos emails dos usuários do suporte básico e usuários não registrados.
- Suas sugestões para o favorecimento do desenvolvimento do MySQL receberão forte consideração. Extensões simples que fazem parte dos objetivos básicos do MySQL são implementados em poucos dias. Obtendo o suporte estendido por e-mail você estará ajudando e contribuindo ao desenvolvimento do MySQL.
- Situações típicas que são cobertas pelo suporte estendido por e-mail:
 - Iremos responder e (com razão) resolver questões relativas a possíveis bugs no MySQL. Logo que um bug for encontrado e corrido, você receberá a correção.
 - Ajudaremos com problemas inesperados quando você instalar o MySQL de uma distribuição fonte ou binária em plataformas suportadas.
 - Responderemos questões sobre recursos não disponíveis e ofereceremos dicas de como contornar tais situações.
 - Forneceremos dicas de como otimizar o `mysqld` para sua situação.
- Será permitido que você influencie a prioridade de itens na lista de coisas à fazer. See [\(undefined\) \[TODO\], page \(undefined\)](#). Isto irá assegurar que os recursos que você realmente necessita serão implementados logo que possível.

1.3.5.3 Suporte por Login

O Suporte por Login inclui todos os detalhes citados no suporte estendido por e-mail com estes acréscimos:

- Suas mensagens serão tratadas antes dos emails do usuarios do suporte estendido por email.
- Suas sugestões para o favorecimento do desenvolvimento do MySQL receberão forte consideração. Extensões realistas que podem ser implementadas em poucas horas e que ifazem parte dos objetivos básicos do MySQL serão implementadas assim que possível.
- Se você possuir um problema muito específico, podemos acessar remotamente seu sistema para resolver seu problema “no local.”
- Como qualquer outro fornecedor de bancos de dados, não podemos garantir que podemos recuperar dados de tabelas danificadas, mas se o pior acontecer, podemos ajudar a recuperar o que for possível. O MySQL prova ser muito estável, mas qualquer coisa é possível devidas as circunstancias que vão além de nosso controle (por exemplo, se seu sistema travar ou alguém finalizar o servidor executando um comando `kill -9`).
- Forneceremos ajuda nas otimizações de seu sistema e queries.

- Você estará apto a contactar um desenvolvedor MySQL (moderadamente) e discutir seus problemas relacionados ao MySQL. Esta opção, entretanto, deve somente ser usada como um último recurso durante uma emergência caso falharmos para compreender o problema por email. Para deixar mais eficiente o uso de nosso tempo precisamos antecipadamente saber de todos os fatos sobre o problema, antes de falarmos no telefone, para trabalhar da maneira mais eficiente possível na solução do problema.

1.3.5.4 Suporte estendido por Login

Suporte estendido por login inclui todos os itens do suporte de Login com as seguintes vantagens:

- Seu e-mail tem a mais alta prioridade possível.
- Examinaremos ativamente seu sistema e vamos ajudar a otimizá-lo e suas queries. Podemos também otimizar e/ou estender o MySQL para melhor encaixar em suas necessidades.
- Você pode também requisitar extensões especiais para você. Por exemplo:

```
mysql> select MINHA_FUNCAO(col1,col2) from tabela;
```
- Forneceremos uma distribuição binária de todos releases importantes do MySQL para seu sistema, assim que tivermos uma conta em um sistema similar. No pior caso, podemos solicitar acesso ao seu sistema para criar uma distribuição binária.
- Se você puder fornecer acomodações e pagar por transporte, podemos até mesmo enviar um desenvolvedor MySQL para visitá-lo e oferecer ajudar com seus problemas. Suporte estendido por Login lhe dá o direito de um encontro por ano. mas somos muito flexíveis para com nossos clientes! Se a visita demorar 16 horas ou mais, as primeiras 8 horas não serão cobradas. Para as 8 horas seguintes, você será cobrado com uma taxa que é pelo menos 20% menor do que os valores padrões.

1.3.5.5 Suporte Telefônico

Suporte telefônico fornece todos os detalhes listados no suporte estendido por login com os seguintes complementos:

- Forneceremos a você uma página web dinâmica exibindo a lista atual de desenvolvedores MySQL que pode ajudá-lo por telefone quando você passar por um problema crítico.
- Para problemas não críticos, você pode solicitar um desenvolvedor MySQL para retornar a ligação em até 48 horas para discutir questões relacionados ao MySQL.

1.3.5.6 Suporte para outros manipuladores de tabela

Para obter suporte para tabelas BDB ou tabelas InnoDB você deve pagar uma taxa adicional de 30% no preço padrão do suporte para cada um dos manipuladores de tabelas que você desejar acrescentar no suporte.

Nós na MySQL AB iremos ajudá-lo a criar um relatório correto de bugs para o manipulador de tabelas e submetê-lo aos desenvolvedores da tabela específica. Também iremos fazer o máximo para garantir que você irá obter uma resposta ou solução em tempo dos desenvolvedores do manipulador de tabelas.

Mesmo se estivermos muito convencidos que podemos resolver a maioria dos problemas de uma maneira precisa, não podemos garantir uma solução rápida para quaisquer problemas que você possa ter com os diferentes manipuladores de tabelas. Iremos entretanto fazer o melhor possível para ajudá-lo a ter seu problemas resolvidos.

1.4 Qual compatibilidade aos padrões o MySQL oferece ?

Esta seção descreve como o MySQL relaciona aos padrões ANSI SQL. MySQL tem muitas extensões aos padrões ANSI SQL, e aqui você descobrirá quais são elas, e como usá-las. Você irá também encontrar informação sobre falta de funcionalidade do MySQL, e como trabalhar com algumas diferenças.

1.4.1 Extensões MySQL para o ANSI SQL92

O MySQL fornece algumas extensões que você provavelmente não irá encontrar em alguns bancos de dados SQL. Fique avisado que se você usá-las, seu código pode não ser mais portátil para outros servidores SQL. Em alguns casos, você pode escrever código que inclui extensões MySQL, mas continua portátil, usando comentários da forma `/*! ...*/`. Neste caso, o MySQL irá analisar e executar o código com o comentário como irá fazer com qualquer outra instrução MySQL, mas outros servidores SQL irão ignorar as extensões. Por exemplo:

```
SELECT /*! STRAIGHT_JOIN */ nome_campo FROM table1,table2 WHERE ...
```

Se você adicionar um número de versão depois do `'!'`, a sintaxe só será executada se a versão do MySQL é igual ou maior que o número de versão usado:

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

O exemplo acima significa que se você tiver uma versão do MySQL 3.23.02 ou mais nova, então o MySQL irá usar a palavra-chave `TEMPORARY`

Extensões MySQL são listadas abaixo:

- Os tipos de campo `MEDIUMINT`, `SET`, `ENUM` e os diferentes tipos `BLOB` e `TEXT`.
- Os atributos de campos `AUTO_INCREMENT`, `BINARY`, `NULL`, `UNSIGNED` e `ZEROFILL`.
- Todas comparações de strings por padrão são caso insensitivo, com classificação ordenada determinada pelo conjunto de caracteres corrente (ISO-8859-1 Latin1 por padrão). Se você não gosta disso você deverá declarar suas colunas com o atributo `BINARY` ou usar o operador `BINARY`, que fazendo com que as comparações sejam feitas de acordo com a ordem `ASCII` usada na máquina servidora do MySQL.
- O MySQL mapeia cada banco de dados em um diretório sob o diretório de dados do MySQL, e tabelas internamente num banco de dados para arquivos no diretório do banco de dados.

Isto tem algumas implicações:

- Nomes de bancos de dados e tabelas são caso sensitivo no MySQL em sistemas operacionais que possuem o sistema de arquivos caso sensitivo (como na maioria dos sistemas Unix). See [\[Name case sensitivity\]](#), page [\[Name case sensitivity\]](#).
- Nomes de Bancos de dados, tabelas, índices, campos ou apelidos pode começar com um dígito (porém não podem consistir somente de dígitos).

- Você pode usar comandos padrão do sistemas para fazer backups, renomear, apagar e copiar tabelas. Por exemplo, para renomear uma tabela, renomeie os arquivos ‘.MYD’, ‘.MYI’ e ‘.frm’. para o nome da tabela correspondente.
- Em algumas instruções SQL, você pode acessar tabelas de diferentes bancos de dados com a sintaxe `nome_bd.nome_tbl`. Alguns servidores SQL fornecem a mesma funcionalidade mas chamam isto de `User space`. O MySQL não suporta tablespaces como em: `create table ralph.my_table...IN minha_tablespace`.
- LIKE é permitido em campos numéricos.
- O uso de INTO OUTFILE e STRAIGHT_JOIN em uma instrução SELECT. See [\[SELECT\]](#), page [\[undefined\]](#).
- A opção SQL_SMALL_RESULT em uma instrução SELECT.
- EXPLAIN SELECT para obter uma descrição de como as tabelas são ligadas.
- A utilização de nomes de índices, índices em um prefixo de um campo, e uso de INDEX ou KEY em uma instrução CREATE TABLE. See [\[CREATE TABLE\]](#), page [\[undefined\]](#).
- O uso de TEMPORARY ou IF NOT EXISTS com CREATE TABLE.
- O uso de COUNT(DISTINCT lista) onde 'lista' é maior que um elemento.
- O uso de CHANGE nome_campo, DROP nome_campo, ou DROP INDEX, IGNORE ou RENAME em uma instrução ALTER TABLE. See [\[ALTER TABLE\]](#), page [\[undefined\]](#).
- O uso de RENAME TABLE. See [\[RENAME TABLE\]](#), page [\[undefined\]](#).
- Utilização de múltiplas cláusulas ADD, ALTER, DROP, ou CHANGE em uma instrução ALTER TABLE.
- O uso de DROP TABLE com as palavras-chave IF EXISTS.
- Você pode remover (drop) múltiplas tabelas com uma instrução única DROP TABLE.
- A cláusula LIMIT da instrução DELETE.
- A cláusula DELAYED das instruções INSERT e REPLACE.
- A cláusula LOW_PRIORITY das instruções INSERT, REPLACE, DELETE e UPDATE.
- O uso de LOAD DATA INFILE. Em alguns casos essa sintaxe é compatível com o Oracle LOAD DATA INFILE. See [\[LOAD DATA\]](#), page [\[undefined\]](#).
- As intruções ANALYZE TABLE, CHECK TABLE, OPTIMIZE TABLE, e REPAIR TABLE.
- A instrução SHOW. See [\[SHOW\]](#), page [\[undefined\]](#).
- Strings podem ser fechadas pelo “” ou ‘’, não apenas pelo ‘’.
- O uso do meta-caractere de escape ‘\’.
- A instrução SET OPTION. See [\[SET OPTION\]](#), page [\[undefined\]](#).
- Você não precisa nomear todos os campos selecionados na parte GROUP BY. Isto fornece melhor performance para algumas consultas específicas, mas muito comuns. See [\[Group by functions\]](#), page [\[undefined\]](#).
- Pode ser especificado ASC e DESC com o GROUP BY.
- Para tornar mais fácil para usuários que venham de outros ambientes SQL, o MySQL suporta apelidos (aliases) para várias funções. Por exemplo, todas funções de string suportam as sintaxes ANSI SQL e ODBC.

- O MySQL entende os operadores `||` e `&&` como `ou(OR)` e `e(AND)` lógicos, como na linguagem de programação C. No MySQL, `||` e `OR` são sinônimos, assim como `&&` e `AND`. Devido a esta ótima sintaxe, o MySQL não suporta o operador ANSI SQL para concatenação de strings `||`; em vez disso, use o `CONCAT()`. Como `CONCAT()` aceita vários argumentos, é fácil converter o uso do operador `||` para MySQL.
- `CREATE DATABASE` or `DROP DATABASE`. See [\[CREATE DATABASE\]](#), page [\[undefined\]](#).
- O operador `%` é um sinônimo para `MOD()`. Isto é, `N % M` é equivalente a `MOD(N,M)`. `%` é suportado para programadores C e para compatibilidade com o PostgreSQL.
- Os operadores `=`, `<>`, `<=`, `<`, `>=`, `>`, `<<`, `>>`, `<=>`, `AND`, `OR` ou `LIKE` podem ser utilizados em comparações de campos a esquerda do `FROM` nas instruções `SELECT`. Por exemplo:


```
mysql> SELECT col1=1 AND col2=2 FROM nome_tabela;
```
- A função `LAST_INSERT_ID()`. See [\[mysql_insert_id\(\)\]](#), page [\[undefined\]](#).
- Os operadores estendidos `REGEXP` e `NOT REGEXP` utilizados em expressões regulares.
- `CONCAT()` ou `CHAR()` com um ou mais de dois argumentos. (No MySQL, estas funções receber qualquer número de argumentos.)
- As funções `BIT_COUNT()`, `CASE`, `ELT()`, `FROM_DAYS()`, `FORMAT()`, `IF()`, `PASSWORD()`, `ENCRYPT()`, `md5()`, `ENCODE()`, `DECODE()`, `PERIOD_ADD()`, `PERIOD_DIFF()`, `TO_DAYS()` ou `WEEKDAY()`.
- Uso de `TRIM()` para cortar substrings. o ANSI SQL só suporta remoção de caracteres únicos.
- As funções do `GROUP BY`: `STD()`, `BIT_OR()` e `BIT_AND()`.
- Uso de `REPLACE` no lugar de `DELETE + INSERT`. See [\[REPLACE\]](#), page [\[undefined\]](#).
- As instruções `FLUSH`, `RESET` e `DO`.
- A possibilidade de configurar variáveis em uma instrução com `:=`:

```
SELECT @a:=SUM(total),@b=COUNT(*),@a/@b AS media FROM tabela_teste;
SELECT @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
```

1.4.2 Diferenças do MySQL comparadas com o ANSI SQL92

Nós tentamos fazer com que o MySQL siguisse os padrões ANSI SQL e o ODBC SQL, mas em alguns casos, o MySQL faz algumas coisas de forma diferente:

- `--` é somente um comentário, se seguido de um espaço em branco. See [\[Missing comments\]](#), page [\[undefined\]](#).
- Para campos `VARCHAR`, espaços extras são removidos quando o valor é armazenado. See [\[Bugs\]](#), page [\[undefined\]](#).
- Em alguns casos, campos `CHAR` são alterados sem perguntas para o tipo de campo `VARCHAR`. See [\[Silent column changes\]](#), page [\[undefined\]](#).
- Privilégios para uma tabela não são negadas automaticamente quando você apaga uma tabela. Você deve usar explicitamente um `REVOKE` para negar privilégios para uma tabela. See [\[GRANT\]](#), page [\[undefined\]](#).
- `NULL AND FALSE` serão interpretados como `NULL` e não como `FALSE`. Isto é porque não acreditamos que é uma boa coisa interpretar várias condições extras neste caso.

1.4.3 Executando o MySQL no modo ANSI

Se você inicializa o `mysqld` com a opção `--ansi`, o seguinte comportamento é alterado no MySQL:

- `||` funciona para concatenação de strings em vez de `OR`.
- Você pode ter qualquer número de espaços entre um nome de função e o `('`. Isto faz com que todos nomes de funções sejam tratadas como palavras reservadas.
- `'` será um caracter identificador (como o caracter aspas `“` do MySQL) e não um caracter de string.
- `REAL` será um sinônimo para `FLOAT` no lugar de um sinônimo de `DOUBLE`.
- O nível de isolamento padrão de um transação é `SERIALIZABLE`. See [\(undefined\) \[SET TRANSACTION\]](#), page [\(undefined\)](#).

Isto é o mesmo que usar `--sql-mode=REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,SERIALIZE,ONLY_FULL_GROUP_BY`.

1.4.4 Funcionalidades perdidas no MySQL

As seguintes funcionalidades não estão presentes na versão atual do MySQL. Para uma lista priorizada indicando quando novas extensões serão adicionadas ao MySQL você deve consultar [A lista do à fazer online\(TODO List\)](#) . Esta é a última versão da lista do à fazer neste manual. See [\(undefined\) \[TODO\]](#), page [\(undefined\)](#).

1.4.4.1 Sub-selects

Atualmente o MySQL suporta somente sub selects da forma `INSERT ... SELECT ...` e `REPLACE ... SELECT ...`. Entretanto, você pode usar a função `IN()` em outros contextos.

Em alguns casos você pode reescrever a query sem uma sub selects:

```
SELECT * FROM table1 WHERE id IN (SELECT id FROM table2);
```

Isto pode ser reescrito como:

```
SELECT table1.* FROM table1,table2 WHERE table1.id=table2.id;
```

As consultas:

```
SELECT * FROM table1 WHERE id NOT IN (SELECT id FROM table2);
```

```
SELECT * FROM table1 WHERE NOT EXISTS (SELECT id FROM table2 where table1.id=table2
```

Podem ser reescritas como:

```
SELECT table1.* FROM table1 LEFT JOIN table2 ON table1.id=table2.id where table2.id
```

Para consulta aninhadas mais complexas você pode, geralmente, criar tabelas temporárias para armazenar as consultas aninhadas. Em alguns casos, entretanto, essa opção não irá funcionar. O mais frequentemente encontrado nestes casos surgem em instruções `DELETE`, para o qual padrão `SQL` não suporta joins (exceto nas sub-selects). Para esta situação existem duas opções disponível enquanto consultas aninhadas forem suportadas pelo MySQL.

A primeira opção é usar uma linguagem de programação procedural (como Perl ou PHP) para enviar uma query `SELECT` para obter as chaves primárias dos registros a serem deletados, e depois usar estes valores para construir uma instrução `DELETE DELETE FROM ... WHERE ... IN (key1,key2, ...)`.

A segunda opção é usar o `interactive SQL` para construir uma série de instruções `DELETE` automaticamente, usando a extensão MySQL `CONCAT()` (em lugar do operador padrão `||`). For example:

```
SELECT CONCAT('DELETE FROM tab1 WHERE pkid = ', tab1.pkid, ');'
FROM tab1, tab2
WHERE tab1.col1 = tab2.col2;
```

Você pode colocar esta consulta em um arquivo script e redirecionar sua saída para o interpretador de linha de comandos `mysql`, canalizando sua saída de volta a uma segunda instância do interpretador:

```
prompt> mysql --skip-column-names mydb < myscript.sql | mysql mydb
```

O MySQL 4.0 suporta deleção de multi-tabelas que pode ser usado para apagar registros de forma eficiente com base em informações de uma tabela ou mesmo de várias tabelas ao mesmo tempo.

1.4.4.2 SELECT INTO TABLE

O MySQL ainda não suporta a extensão SQL Oracle: `SELECT ... INTO TABLE ...`. MySQL suporta a sintaxe ANSI SQL `INSERT INTO ... SELECT ...`, que é basicamente a mesma coisa. See [\[INSERT SELECT\]](#), page [\[undefined\]](#).

```
INSERT INTO tblTemp2 (fldID) SELECT tblTemp1.fldOrder_ID FROM tblTemp1 WHERE
tblTemp1.fldOrder_ID > 100;
```

De maneira alternativa, você pode usar `SELECT INTO OUTFILE...` ou `CREATE TABLE ... SELECT` para resolver seu problema.

1.4.4.3 Transações

Como o MySQL, hoje em dia, suporta transações, a seguinte discussão só é válida se você estiver usando os tipos de tabelas não transacionais. See [\[COMMIT\]](#), page [\[undefined\]](#)

Essa questão é normalmente feita, pelos curiosos e críticos, “Por que o MySQL não é um banco de dados transacional?” ou “Por que o MySQL não suporta transações?”.

O MySQL tem feito uma decisão consciente para suportar outro paradigma para integridade de dados, “operações atômicas.” É de nosso conhecimento e experiência que operações atômicas oferecem integridade igual ou melhor com performance muito mais eficiente. Nós, apesar de tudo, apreciamos e entendemos o paradigma de bancos de dados transacionais e planejamos, nas próximas versões, introduzir tabelas com segurança transacional nos fundamentos de tabelas. Nós iremos dar a nossos usuários a possibilidade de decidir se eles precisam da velocidade das operações atômicas ou se eles precisam usar recursos transacionais na suas aplicações.

Como são utilizados recursos do MySQL para manter de forma rigorosa a integridade e como estes recursos se comparam com o paradigma transacional?

Primeiro, no paradigma transacional, se as suas aplicações são escritas de uma forma que é dependente na chamada de “rollback” em vez de “commit” em situações críticas, então transações são mais convenientes. Além disso, transações asseguram que atualizações inacabadas ou atividades corrompidas não sejam executadas no banco de dados; o servidor

oferece uma oportunidade para fazer um rollback automático e seu banco de dados é mantido.

O MySQL, na maioria dos casos, permite a você resolver potenciais problemas incluindo simples conferências antes das atualizações e executando scripts simples que conferem inconsistências no banco de dados e, automaticamente, repara ou avisa caso isto ocorra. Perceba que apenas usando o log do MySQL ou mesmo adicionando um log extra, pode-se corrigir tabelas perfeitamente sem nenhuma perda de integridade.

Além disso, atualizações transacionais fatais podem ser reescritas para serem atômicas. De fato podemos dizer que todos problemas de integridade que transações resolvem podem ser feitas com `LOCK TABLES` ou atualizações atômicas, assegurando que você nunca irá ter uma finalização automática da tabela, o que é um problema comum em bancos de dados transacionais.

Nem mesmo transações podem prevenir todas as falhas se o servidor cair. Nestes casos mesmo um sistema transacional pode perder dados. A diferença entre sistemas diferentes é apenas em quão pequeno é o lapso de tempo em que eles podem perder dados. Nenhum sistema é 100% seguro, somente “seguro o suficiente.” Mesmo o Oracle, com reputação de ser o mais seguro bancos de dados transacionais, tem relatos de algumas vezes perder dados nestas situações.

Para estar seguro com o MySQL, você apenas deve fazer backups e ter o log de atualizações ligado. Com isto você pode se recuperar de qualquer situação possível com bancos de dados transacionais. É sempre bom ter backups, independente de qual banco de dados você usa.

O paradigma transacional tem seus benefícios e suas desvantagens. Muitos usuários e desenvolvedores de aplicações dependem da facilidade com a qual eles podem codificar contornando problemas onde abortar parece ser, ou é necessário, e podem ter um pouco mais de trabalho com o MySQL por terem que pensar diferente ou escrever mais. Se você é novo no paradigma de operações atômicas ou tem mais familiaridade ou conforto com transações, não acabe por concluir que o MySQL não tenha se preocupado com estes assuntos. Confiança e integridade estão em nossas mentes. Estimativas recentes indicam que existem mais de 1.000.000 de servidores `mysqld` operando atualmente, muitos dos quais estão em ambientes de produção. São muito, muito raros os casos nos quais nossos usuários perderam seus dados, e em quase todos os casos, erros dos usuários estão envolvidos. Isto é, na nossa opinião, a melhor prova da estabilidade e segurança do MySQL.

Ultimamente, em situações onde integridade é de grande importância, as atuais características do MySQL permitem níveis transacionais ou melhor confiança e integridade. Se você bloquear tabelas com `LOCK TABLES` todos as atualizações irão ser adiadas até qualquer verificação de integridade ser feita. Se você só obter um bloqueio de leitura (oposto ao bloqueio de escrita), então leituras e inserções poderão ocorrer. Os novos registros inseridos não poderão ser visualizados por nenhum dos clientes que tiverem um bloqueio de `LEITURA` até eles liberarem estes bloqueios. Com `INSERT DELAYED` você pode enfileirar inserções em uma fila local, até os bloqueios serem liberados, sem que o cliente precise esperar pela inserção completar. See [\[INSERT DELAYED\]](#), page [\[INSERT DELAYED\]](#)

“Atômico”, no sentido em que nós mencionamos, não é mágico. Significa apenas que você pode estar certo que enquanto cada atualização específica está sendo executada, nenhum outro usuário pode interferir com ela, e nunca haverá um rollback automático (que pode acontecer em sistemas baseados em transações se você não tiver muito cuidado). O MySQL

também assegura que nunca ocorrerá uma leitura suja. Você pode encontrar alguns exemplos de como escrever atualizações atômicas na seção `commit-rollback`. See [\[Commit-rollback\]](#), page [\[undefined\]](#).

Estamos pensando muito sobre integridade e performance, e nós acreditamos que nosso paradigma de operações atômicas permitem tanto confiança quanto uma performance extremamente alta, na ordem de tres a cinco vezes a velocidade do mais rápido e mais otimizado dos bancos de dados transacionais. Nós não deixamos as transações porque são muito difíceis para serem feitas. A principal razão de trabalharmos com operações atômicas ao invés de transações é que fazendo isto nós podemos aplicar muitas otimizações de velocidade que de outra forma não seria possível.

Muitos de nossos usuários que tem a velocidade em mente, não estão muito preocupados com transações. Para eles transações não são importantes. Para aqueles usuários que estão preocupados com ou tem curiosidade sobre transações no MySQL, existe um “MySQL Way” como nós citamos acima. Para aqueles que dão mais importância a segurança do que velocidade, recomendamos o uso das tabelas BDB ou InnoDB para todos seus dados críticos. See [\[Table types\]](#), page [\[undefined\]](#).

Nota final: Atualmente nós estamos trabalhando em um esquema seguro de replicação o qual acreditamos ser melhor que qualquer sistema de replicação comercial que conhecemos. Este sistema irá trabalhar com confiança sobre o paradigma de operações atômicas não-transacional. Mantenha-se informado.

1.4.4.4 Stored Procedures e Triggers

Uma Stored Procedure é um conjunto de comandos SQL que podem ser compilados e armazenados no servidor. Uma vez feito isso, os clientes não necessitam reescrever toda a consulta mas podem fazer referência à stored procedure. Isto fornece melhor performance porque a query necessita ser analisada pelo servidor somente uma vez, e necessita menos informação para ser enviada entre o servidor e o cliente. Você também pode elevar o nível conceitual tendo bibliotecas de funções no servidor.

Um trigger é uma stored procedure que é chamado quando um evento em particular ocorre. Por exemplo, você pode instalar uma stored procedure que é disparada toda vez que um registro for apagado de uma tabela de transações que automaticamente apaga o cliente correspondente de uma tabela de clientes quando todas as transações forem removidas.

A linguagem planejada atualizada será apta a armazenar stored procedures mas sem triggers. Trigger normalmente deixa tudo muito mais lento, mesmo consultas que não as utilizam.

Para saber quando o MySQL terá stored procedures, consulte [\[TODO\]](#), page [\[undefined\]](#).

1.4.4.5 Chaves Estrangeiras (Foreign Keys)

Note que as chaves estrangeiras no SQL não são usadas para ligar tabelas, mas na maioria das vezes são usadas para verificar a integridade referencial. Se você deseja obter resultados de múltiplas tabelas de uma instrução `SELECT`, você pode fazer isto ligando tabelas:

```
SELECT * from table1,table2 where table1.id = table2.id;
```

See [\[JOIN\]](#), page [\[example-Foreign keys\]](#), page [\[example-Foreign keys\]](#).

A sintaxe `FOREIGN KEY` existe no MySQL somente para compatibilidade com os comandos `CREATE TABLE` de outros fabricantes, ela não faz nada. A sintaxe `FOREIGN KEY` sem `ON DELETE . . .` é usada na maioria dos casos para documentação. Algumas aplicações ODBC podem usar isto para produzir cláusulas `WHERE` automáticas, mas é normalmente ignorada. `FOREIGN KEY` é algumas vezes usado para verificação, porém, esta verificação é desnecessária na prática se os registros são inseridos nas tabelas na ordem correta. O MySQL suporta somente estas cláusulas porque algumas aplicações dependem delas para existir (não levando em consideração se elas funcionam ou não).

No MySQL, você pode trabalhar contornando o problema do `ON DELETE . . .` ainda não implementado, adicionando as instruções `DELETE` apropriadas a aplicação quando você apagar registros de uma tabela que possui uma chave estrangeira. Na prática isto é tão rápido (em alguns casos, mais rápido) e muito mais portátil do que o uso de chaves estrangeiras.

No futuro próximo, iremos estender a implementação de **CHAVES ESTRANGEIRAS** para que ao menos a informação seja armazenada no arquivo de especificações da tabela e possa ser recuperada com `mysqldump` e ODBC. Em um estágio posterior, nós implementaremos as chaves estrangeiras para aplicações que não podem ser facilmente codificadas para ignorá-la.

1.4.4.6 Razões para não usar chaves estrangeiras

Existem vários problemas com as chaves estrangeiras. É até difícil sabermos por onde começar:

- Chaves estrangeiras tornam a vida muito complicada porque suas definições devem ser armazenadas em um banco de dados e, implementá-las pode acabar com a “graça” de usar arquivos que podem ser movidos, copiados e removidos.
- O impacto de velocidade é terrível em sentenças `INSERT` e `UPDATE`, neste caso, quase todas as verificações de **CHAVES ESTRANGEIRAS** são desnecessários, pois normalmente inserimos registros nas tabelas corretas e na ordem correta.
- Existe também uma necessidade de travar registros em várias tabelas quando estiver atualizando apenas uma tabela, devido ao efeito cascata em todo no banco de dados. É muito mais rápido primeiro apagar registros de uma tabela e posteriormente apagá-los das outras tabelas.
- Você não pode restaurar uma tabela apagando-a totalmente e depois restaurar os registros (de uma nova fonte ou de um backup).
- Se você usa chaves estrangeiras, não poderá remover e restaurar tabelas a não ser que faça isto com muito cuidado em uma ordem muito específica.
- É muito fácil fazer definições circulares “permitidas” que impossibilitam as tabelas de recriar cada tabela com uma simples instrução de criação mesmo se a definição funciona e está em uso.
- É muito fácil de travar regras `FOREIGN KEY . . . ON DELETE` quando estamos codificando uma aplicação. Não é raro ocorrer perda de muitas informações importantes devido apenas a uma regra `ON DELETE` errada ou mal utilizada.

O único aspecto interessante das **CHAVES ESTRANGEIRAS** é que ela dá ao ODBC e alguns outros programas clientes a habilidade de ver como a tabela é conectada e utilizá-la para mostrar diagramas de conexão e ajudar na criação de aplicações.

MySQL logo irá armazenar definições de **CHAVES ESTRANGEIRAS** para que o cliente possa pedi-la e receber uma resposta sobre como a conexão original foi feita. O formato atual dos arquivos `.frm` não tem lugar para elas. Em um estágio posterior nós iremos implementar as chaves estrangeiras para aplicações que não podem ser codificadas para evitá-las.

1.4.4.7 Views

MySQL ainda não suporta views, mas planejamos implementá-las em torno da versão 4.1. Views geralmente são muito úteis para permitir aos usuários acessar uma série de relações como uma tabela (em modo leitura). Vários bancos de dados SQL não permitem atualizar nenhum registro em uma view, mas você tem que fazer as atualizações em tabelas separadas. Como o MySQL é normalmente usado em aplicações e sistemas web onde o autor da aplicação tem total controle no uso de bancos de dados, a maioria de nossos usuários não consideram view muito importante. (pelo menos nenhum teve interesse suficiente para financiar a implementação de views).

Não é necessário o uso de views em MySQL para restringir acesso aos campos já que o MySQL tem um sistema de privilégios bem sofisticado. See [\[Privilege system\]](#), page [\[undefined\]](#).

1.4.4.8 '--' como o início de um comentário

Outros bancos de dados SQL usam '--' para iniciar comentários. O MySQL usa '#' como o caractere para início de comentário, mesmo se a ferramenta de linha de comando `mysql` remover todas linhas que começam com '--'. Você também pode usar o comentário no estilo C `/*isto é um comentário*/` com o MySQL. See [\[Comments\]](#), page [\[undefined\]](#).

O MySQL versão 3.23.3 e superior suporta o estilo de comentário '--' somente se o comentário for seguido por um caractere de espaço. Isto ocorre porque este estilo de comentário causou muitos problemas com queries SQL geradas automaticamente que usavam algo como o código seguinte, onde automaticamente é inserido o valor do pagamento para `!pagamento!`:

```
UPDATE nome_tabela SET credito=credito-!pagamento!
```

O que você acha que irá acontecer quando o valor de `pagamento` for negativo?

Como `1--1` é legal no SQL, nós achamos terrível que '--' signifique início de comentário.

No MySQL Versão 3.23 você pode, entretanto, usar: `1-- Isto é um comentário`

A seguinte discussão somente interessa se você estiver executando uma versão do MySQL inferior a versão 3.23:

Se você tem um programa SQL em um arquivo texto que contém comentários '--' você deverá usar:

```
shell> replace " --" " #" < arquivo-texto-com-comentário.sql \
| mysql banco-de-dados
```

No lugar de:

```
shell> mysql banco-de-dados < arquivo-texto-com-comentario.sql
```

Você também pode editar o próprio arquivo de comandos alterando os comentários '--' para '#':

```
shell> replace " --" " #" -- arquivo-texto-com-comentario.sql
```

Desfaça utilizando este comando:

```
shell> replace " #" " --" -- arquivo-texto-com-comentario.sql
```

1.4.5 Quais são os padrões que o MySQL segue ?

Padrão SQL92. Níveis ODBC 0-2.

1.4.6 Como lidar sem o COMMIT-ROLLBACK

A maioria dos textos a seguir aplicam-se somente para tabelas ISAM, MyISAM e HEAP. Se você utilizar as tabelas transacionais (tabelas BDB ou InnoDB) em uma atualização, você poderá utilizar COMMIT e ROLLBACK no MySQL. See [\[COMMIT\]](#), page [\[COMMIT\]](#).

O problema com a manipulação de COMMIT-ROLLBACK eficientemente com os tipos de tabelas citadas acima é que seria necessário um layout de tabelas completamente diferente que o MySQL usa hoje em dia. O tipo de tabela iria necessitar também de processos extras que fizessem a limpeza automática nas tabelas, e o uso de disco seria muito maior. Isto poderia deixar esses tipos de tabelas 2-4 vezes mais lento que são hoje.

No momento, preferimos implementar a linguagem de servidor SQL (algo parecido com stored procedures). Com isto, você raramente precisaria usar o COMMIT-ROLLBACK. Isto também traria uma performance muito melhor.

Loops que necessitam de transações normalmente podem ser codificados com a ajuda de LOCK TABLES. Se você atualiza registros previamente determinados não haverá necessidade de se usar cursores.

Nós na TcX temos uma maior necessidade de um banco de dados realmente rápido do que um banco de dados 100% genérico. Assim que encontrarmos uma forma de implementar esses recursos sem perda de performance, provavelmente o faremos. No momento, existem várias coisas mais importantes a serem feitas. Verifique o TODO para saber quais as nossas prioridades até o momento. (Clientes com alto nível de suporte podem alterar isto, então as prioridades podem ser alteradas.)

O problema atual é o ROLLBACK. Sem o ROLLBACK, pode-se fazer qualquer tipo de ação COMMIT com LOCK TABLES. Para suportar ROLLBACK com os tipos de tabelas acima, o MySQL teria que ser alterado para armazenar todos registros antigos que foram atualizados e reverter tudo para o ponto de partida se o ROLLBACK fosse utilizado. Para casos simples, isto não é difícil para implementar (o `isamlog` atual pode ser usado para este propósito), mas seria muito mais difícil de implementar ROLLBACK para ALTER/DROP/CREATE TABLE

Para evitar o uso do ROLLBACK, você pode usar a seguinte estratégia:

1. Usar LOCK TABLES ... para travar todas as tabelas que você pretende acessar.
2. Testar condições.
3. Atualizar se tudo estiver bem.
4. Usar UNLOCK TABLES para liberar suas travas.

Este método é, normalmente, muito mais rápido do que usar transações com possíveis ROLLBACKs, mas nem sempre. A única situação em que esta solução não se aplica é quando alguém mata as threads no meio de uma atualização. Neste caso, todos bloqueios serão liberados mas algumas atualizações podem não ter sido executadas.

Você também pode utilizar funções para atualizar registros em uma única operação. Você pode ter uma aplicação muito eficiente utilizando as seguintes técnicas:

- Modificar campos relativos aos seus valores atuais.
- Atualizar somente aqueles campos que foram realmente alterados.

Por exemplo, quando estivermos fazendo atualizações em alguma informação de clientes, atualizamos somente os dados do cliente que foram alterados e testamos somente aqueles que não tiveram dados alterados ou dados que dependam dos dados alterados e que tenham sido alterados comparados ao registro original. O teste para dados alterados é feito com a cláusula WHERE na instrução UPDATE. Se o registro não for atualizado, daremos ao cliente uma mensagem: "Alguns dos dados que você alterou foi alterado por outro usuário". Então nós mostramos o registro antigo e o novo registro na tela para que o usuário possa decidir qual versão ele pode usar.

Isto fornece algo que é parecido com bloqueio de colunas mas bem melhor, porque só atualizamos algumas das colunas, usando valores que são relativos aos seus valores atuais. Isto significa que instruções UPDATE se parecerão com isto:

```
UPDATE nome_tabela SET pay_back=pay_back+'mudança relativa';
```

```
UPDATE customer
SET
  data_corrente='data_corrente',
  endereco='novo endereco',
  telefone='novo telefone',
  dinheiro_ele_nos_deve=dinheiro_ele_nos_deve+'novo_dinheiro'
WHERE
  id_cliente=id AND endereco='endereco antigo' AND telefone='telefone antigo';
```

Como podemos ver, esta forma é muito eficiente e funciona mesmo se outro cliente tiver alterado os valores nas colunas pay_back ou dinheiro_ele_nos_deve.

Em alguns casos, usuários tem necessitado de ROLLBACK e/ou LOCK TABLES com o propósito de gerenciar identificadores únicos para algumas tabelas. Isto pode ser tratado de forma muito mais eficiente usando um campo AUTO_INCREMENT e também a função SQL LAST_INSERT_ID() ou a função da API C mysql_insert_id(). See [\[mysql_insert_id\(\)\]](#), page [\[mysql_insert_id\(\)\]](#).

Na MySQL AB, nós nunca tivemos necessidade de bloqueio no nível de registros porque nós sempre fomos capaz de programar evitando-os. Alguns casos realmente precisam de bloqueio de registros, mas são poucos. Se você deseja bloqueio de registros, você pode usar um campo de sinalização na tabela e fazer como se segue:

```
UPDATE nome_tabela SET row_flag=1 WHERE id=ID;
```

O MySQL retorna 1 para o número de registros atingidos se o registro for encontrado e row_flag já não tinha 1 no registro original.

Você pode pensar como se o MySQL tivesse alterado a consulta acima para:

```
UPDATE nome_tabela SET row_flag=1 WHERE id=ID and row_flag <> 1;
```

1.4.7 Erros conhecidos e deficiências de design no MySQL

Os seguintes problemas são conhecidos e tem prioridade muito alta para serem corrigidos:

- `ANALYZE TABLE` em uma tabela BDB pode, em alguns, casos inutilizar a tabela até que se reinicie o servidor `mysqld`. Quando isto acontecer você irá ver o seguinte tipo de erro no arquivo de erros do MySQL.

```
001207 22:07:56 bdb: log_flush: LSN past current end-of-log
```
- Não execute `ALTER TABLE` em uma tabela BDB em que você estiver executando transações multi-instruções não completadas. (A transação provavelmente será ignorada).
- `ANALYZE TABLE`, `OPTIMIZE TABLE` e `REPAIR TABLE` podem causar problemas em tabelas para as quais você estiver usando `INSERT DELAYED`.
- Fazendo um `LOCK TABLE ..` e `FLUSH TABLES ..` não garante que não existem transações não terminadas em progresso na tabela.
- Tabelas BDB são um pouco lentas para abrir. Se você tiver várias tabelas BDB em um banco de dados, gastará muito tempo para usar o cliente `mysql` no banco de dados se você não estiver usando a opção `-A` ou se você estiver usando `rehash`. Isto é percebido principalmente quando você tiver um cache de tabelas grandes.
- O protocolo de replicação atual não pode lidar com `LOAD DATA INFILE` e caracteres de terminação de linha maiores que 1 caractere.

Os seguintes problemas são conhecidos e serão corrigidos na hora certa:

- No momento `MATCH` funciona somente com instruções `SELECT`.
- Quando estiver usando `SET CHARACTER SET`, não é permitido usar caracteres especiais no nome do banco de dados, tabelas ou campos.
- `DELETE FROM merge_table` usado sem `WHERE` irá apenas apagar o mapeamento para a tabela, não apagando tudo nas tabelas mapeadas.
- Você não pode construir em outro diretório quando estiver utilizando MIT-pthreads. Como isto necessitaria de alterações na MIT-pthreads, nós não estamos aptos a corrigí-la.
- BLOB valores não podem ser usados com confiança em `GROUP BY`, `ORDER BY` ou `DISTINCT`. Somente os primeiros bytes (padrão 1024) `max_sort_length` são usados quando estiver comparando BLOBs nestes casos. Isto pode ser alterado com a opção `-O max_sort_lenght` para `mysqld`. Uma forma de contornar este problema para a maioria dos casos é usar a substring: `SELECT DISTINCT LEFT(blob,2048) FROM nome_tabela`.
- Cálculos são feitos com `BIGINT` ou `DOUBLE` (normalmente, ambos tem o tamanho de 64 bits). Depende da precisão utilizada na função. A regra geral é que funções binárias são feitas com precisão `BIGINT`, `IF` e `ELT()` com precisão `BIGINT` ou `DOUBLE` e o resto com precisão `DOUBLE`. Devemos evitar o uso de valores sem sinal maiores que 63 bits (9223372036854775807) para qualquer outra coisa além de campos binários!
- Todas os campos string, exceto campos do tipo `BLOB` e `TEXT` tem, automaticamente, todos os espaços extras removidos quando recuperados. Para tipos `CHAR`, isto não tem problema, e pode ser considerado como um recurso de acordo com o ANSI SQL92. O problema é que no MySQL, campos `VARCHAR` são tratados desta mesma forma.

- Você só pode ter até 255 colunas ENUM e SET em uma tabela.
- `safe mysqld` redireciona todas as mensagens de `mysqld` para o log `mysqld`. Um problema com isto é que se você executar o `mysqladmin refresh` para fechar e reabrir o log, a `stdout` e a `stderr` continuam redirecionadas para o log antigo. Se você utiliza `--log` extensivamente, deverá editar o `safe mysqld` para logar em `'hostname'.err` em vez de `'hostname'.log`; assim você pode facilmente utilizar o espaço do log antigo apagando-o e executando `mysqladmin refresh`.
- Em instruções UPDATE, colunas são atualizadas da esquerda para a direita. Se você referenciar a uma coluna atualizada, você irá obter o valor atualizado em vez do valor original, por exemplo:

```
mysql> UPDATE nome_tabela SET KEY=KEY+1,KEY=KEY+1;
```

Isto atualiza KEY com 2 no lugar de 1.

- Você não pode usar tabelas temporárias mais que uma vez na mesma query. Por exemplo, a seguinte instrução não funciona.

```
select * from tabela_temporaria, tabela_temporaria as t2;
```

- RENAME não funciona com tabelas temporárias (TEMPORARY).
- O otimizador pode lidar com o DISTINCT de forma diferente se você estiver usando colunas 'escondidas' em uma join ou não. Em uma join, colunas escondidas são contadas como parte do resultado (mesmo se elas não são mostradas) enquanto que em queries normais colunas escondidas não participam na comparação DISTINCT. Nós provavelmente iremos alterar isto no futuro para nunca comparar as colunas escondidas quando executando DISTINCT.

um exemplo disto é:

```
SELECT DISTINCT mp3id FROM band_downloads WHERE userid = 9 ORDER BY id
DESC;
```

e

```
SELECT DISTINCT band_downloads.mp3id, FROM band_downloads,band_mp3
WHERE band_downloads.userid = 9 AND band_mp3.id = band_downloads.mp3id
ORDER BY band_downloads.id DESC;
```

No segundo caso, você pode obter duas linhas idênticas no MySQL 3.23.x na série do resultado (porque o campo escondido 'id' pode variar).

Perceba que isto somente acontece em consultas onde você não tem colunas ORDER BY no resultado, algo não permitido no ANSI SQL.

- Como o MySQL permite trabalhar com tipos de tabelas que não suportam transações (e assim não pode fazer `rollback` em dados) algumas coisas funcionam um pouco diferentes de outros servidores SQL em MySQL (Isto serve para garantir que o MySQL nunca necessitará de um `rollback` para um comando SQL). Porém isto pode ser um pouco estranho em casos que os valores dos campos devem ser verificados na aplicação, mas isto ira fornecer um ótimo ganho de velocidade assim como permite ao MySQL fazer algumas otimizações que de outro modo seriam muito difíceis para serem feitas. Se você informar um valor incorreto em uma coluna, o MySQL, em vez de fazer um `rollback`, armazenará o **melhor valor possível** no campo.
 - Se tentar armazenar um valor fora da faixa em uma coluna numérico, o MySQL irá armazenar o menor ou maior valor possível no campo.

- Se tentar armazenar uma string que não comece com um número em uma coluna numérica, o MySQL irá armazenar 0 na coluna.
- Se você tentar armazenar `NULL` em uma coluna que não aceita valores nulos, MySQL irá armazenar 0 ou '' (string vazia) na coluna. (Este comportamento pode, entretanto, ser alterado com a opção de compilação `-DDONT_USE_DEFAULT_FIELDS`).
- O MySQL permite o armazenamento de alguns valores errados de data em campos do tipo `DATE` e `DATETIME`. (Como 2000-02-31 ou 2000-02-00). Se a data estiver totalmente errada, o MySQL irá armazenar a data 0000-00-00 no campo.
- Se você especificar um valor não suportado para um campo do tipo `enum`, ele será alterado para o valor de erro 'empty string', com valor numérico 0.
- Se você executar uma `PROCEDURE` em uma pesquisa que retorna uma série vazia, em alguns casos a instrução `PROCEDURE` não irá transformar as colunas.
- Criação da tabela do tipo `MERGE` não verifica se as tabelas envolvidas são de tipos compatíveis.
- O MySQL ainda não pode lidar com valores `NaN`, `-Inf` e `Inf` em tipos `double`. Usá-los causará problemas na exportação e importação de dados. Uma solução intermediária é alterar `NaN` para `NULL` (se for possível) e `-Inf` e `Inf` para o valor `double` mínimo ou máximo respectivo possível.
- `LIMIT` em números negativos são tratados como números grandes positivos.
- Se você usar `ALTER TABLE` para primeiro adicionar um índice `UNIQUE` a uma tabela usada em uma tabela `MERGE` e então usar `ALTER TABLE` para adicionar um índice normal na tabela `MERGE`, a ordem das chaves será diferente para as tabelas se existir uma chave antiga não única na tabela. Isto é porque o `ALTER TABLE` coloca chaves `UNIQUE` antes de chaves normais para ser possível detectar chaves duplicadas o mais cedo o possível.

Os seguintes erros são conhecidos em versões mais antigas do MySQL:

- Você pode pendurar um processo se você fizer um `DROP TABLE` em uma tabela entre outras que esteja travada com `LOCK TABLES`.
- No caso seguinte você pode obter um descarrego de memória para o arquivo `core`:
 - Tratamento de inserções com atraso tem deixado inserções pendentes na tabela.
 - `LOCK table` com `WRITE`
 - `FLUSH TABLES`
- Antes da versão 3.23.2 do MySQL um `UPDATE` que atualizava uma chave com um `WHERE` na mesma chave podia falhar porque a chave era usada para procurar por registros e a mesma linha poderia ter encontrado vários itens:

```
UPDATE nome_tabela SET KEY=KEY+1 WHERE KEY > 100;
```

Um modo de contornar este erro é utilizar:

```
mysql> UPDATE nome_tabela SET KEY=KEY+1 WHERE KEY+0 > 100;
```

Isto funcionará porque MySQL não utilizará índices em expressões com a cláusula `WHERE`.

- Antes da versão 3.23 do MySQL, todos os tipos numéricos tratados como campos de pontos fixos. Isto significa que você tem que especificar quantas casas decimais um

campo de ponto flutuante deve ter. Todos os resultados eram retornados com o número correto de casas decimais.

Para erros específicos na plataforma, veja as seções sobre compilação e portabilidade.

1.5 Como comparar o MySQL com outros Bancos de dados

Esta seção compara o MySQL com outros bancos de dados populares.

Esta seção foi escrita pelos desenvolvedores MySQL, então deve ser lida com isto em mente. Não é de nosso conhecimento a existência de erros factuais nesta seção. Se você encontrar algo em que acredita ser um erro, por favor nos contate em docs@mysql.com

Para uma lista de todos os limites suportados, funções e tipos veja a páginas Web `crash-me` em <http://www.mysql.com/information/crash-me.php>.

1.5.1 Como comparar o MySQL com o mSQL

Performance

Para uma verdadeira comparação de velocidade, consulte a suite de benchmark do MySQL See [\[MySQL Benchmarks\]](#), page [\[MySQL Benchmarks\]](#).

Por não existir sobrecarga de criação de threads, um pequeno parser, poucos recursos e segurança simples, mSQL deve ser mais rápido em:

- Testes que conectam e desconectam repetidas vezes, executando queries muito simples durante cada conexão.
- Operações INSERT em tabelas muito simples com poucas colunas e chaves.
- CREATE TABLE e DROP TABLE.
- SELECT em campos que não sejam índice. (Uma varredura de tabela é muito fácil.)

Como estas operações são muito simples, é difícil ser melhor que eles quando você tem uma alta sobrecarga inicial. Depois da conexão ser estabelecida, a performance do MySQL deve ser bem melhor.

Por outro lado, o MySQL é muito mais rápido que o mSQL (e a maioria das outras implementações SQL) no seguinte:

- Operações SELECT complexas.
- Recuperação de grandes resultados (MySQL tem um protocolo melhor, mais rápido e mais seguro)
- Tabelas com strings de tamanho variavel, porque o MySQL tem um handler mais eficiente e pode ter índices em campos VARCHAR
- Manipulação de tabelas com várias colunas.
- Manipulação de tabelas com grande quantidade de registros.
- SELECT com várias expressões.
- SELECT em grande tabelas.
- Tratamento de várias conexões ao mesmo tempo. O MySQL é totalmente multi-threaded. Cada conexão tem sua própria thread, o que significa que nenhuma thread tem que esperar por outra (a menos que uma thread esteja

modificando uma tabela que uma outra thread deseja acessar). No mSQL, uma vez que uma conexão for estabelecida, todos outros devem esperar até que a primeira tenha acabado, sem levar em consideração se a conexão está executando uma query que é curta ou longa. Quando a primeira conexão termina, a próxima pode ser atendida, enquanto todas as outras esperam novamente e etc.

- Joins. O mSQL pode ficar patologicamente lento se você alterar a ordem das tabelas em uma **SELECT**. Na suite de benchmark, o mSQL se mostrou 15000 vezes mais lento que o MySQL. Isto é devido a falta de um otimizador join para ordenar tabelas na ordem otimizada no mSQL. Entretanto, se você colocar as tabelas na ordem exata no mSQL2 e o **WHERE** for simples e usar índice, o join vai ser relativamente rápido! See [\[MySQL Benchmarks\]](#), page [\[MySQL Benchmarks\]](#).
- **ORDER BY** e **GROUP BY**.
- **DISTINCT**.
- Usando tipos **TEXT** ou **BLOB** em campos.

Recursos SQL

- **GROUP BY** e **HAVING**. O mSQL não suporta completamente o **GROUP BY**. O MySQL suporta completamente o **GROUP BY** com **HAVING** e as seguintes funções: **COUNT()**, **AVG()**, **MIN()**, **MAX()**, **SUM()** e **STD()**. **COUNT(*)** é otimizado para retornar rapidamente se o **SELECT** retornar de uma tabela, nenhum outro campo é devolvido e não há uma cláusula **WHERE**. **MIN()** e **MAX()** podem exigir argumentos.
- **INSERT** e **UPDATE** com cálculos. O MySQL pode fazer cálculos em uma instrução **INSERT** ou **UPDATE**. Por exemplo:


```
mysql> UPDATE SET x=x*10+y WHERE x<20;
```
- Apelidos ou alias. MySQL tem suporte a apelidos para os campos.
- Qualificando nomes de colunas. No MySQL, se um nome de coluna é única entre as tabelas usadas em uma query, você não tem que usar o qualificador completo.
- **SELECT** com funções. O MySQL tem várias funções (muitas para serem listadas aqui; veja [\[Functions\]](#), page [\[Functions\]](#)).

Eficiência no Armazenamento em Disco

Ou seja, quão pequena pode ser suas tabelas ?

Existem tipos muito precisos no MySQL, que possibilitam a criação de tabelas que consomem muito pouco espaço em disco. Um exemplo de um tipo de dados MySQL é o **MEDIUMINT** que ocupa 3 bytes. Se você tiver 100.000.000 de registros, economizar um byte por registro pode ser muito importante.

O mSQL2 possui uma variedade mais limitada para tipos de campos, portanto, é mais complicado para obter pequenas tabelas.

Estabilidade

É difícil julgar objetivamente. Para uma discussão sobre estabilidade do MySQL, veja em [\[Stability\]](#), page [\[Stability\]](#).

Nós não temos experiência com a estabilidade do `mSQL`, portanto, não podemos dizer nada sobre isto.

Preço Outro assunto importante é a licença. O MySQL possui uma licença mais flexível que o `mSQL`, e também é mais barato que o `mSQL`. Qualquer produto que você escolha usar, lembre-se de considerar ao menos o pagamento de uma licença ou suporte por e-mail. (Você é obrigado a obter uma licença se incluir o MySQL com um produto que vende.)

Interfaces Perl

O MySQL tem basicamente a mesma interface para o Perl que o `mSQL` com alguns recursos adicionais.

JDBC (Java)

Atualmente, o MySQL tem vários drivers JDBC diferentes:

- O driver `mm`: Um driver JDBC tipo 4 por Mark Matthews mmatthew@ecn.purdue.edu. Este é distribuído sobre a LGPL.
- O driver `Resin`. Este é um driver JDBC comercial open source. <http://www.caucho.com/projects/jdbc-mysql/index.xtp>
- O driver `gwe`: Uma interface JAVA fabricada pela GWE technologies (não é mais suportado).
- O driver `jms`: Um driver `gwe` melhorado por xiaokun Kelvin ZHU X.Zhu@brad.ac.uk (não é mais suportado).
- O driver `twz`: Um driver JDBC tipo 4 por Terrence W. Zellers zellert@voicenet.com. É comercial, porém livre para uso privado e educacional (não é mais suportado).

O driver recomendado é o driver `mm`. O driver `Resin` também pode ser bom (pelo menos nos benchmarks parece bom), mas nós não recebemos informações suficientes sobre ele ainda.

Nós sabemos que o `mSQL` tem um driver JDBC, mas nós temos muito pouca experiência com ele para comparar.

Desenvolvimento

MySQL tem uma equipe de desenvolvedores muito pequena, mas somos bem rápidos codificando em C e C++. Como `threads`, funções, `GROUP BY` e outras recursos ainda não estão implementada em `mSQL`, eles têm muitas coisas para fazer. Para ter uma perspectiva para estas implementações você pode ar uma olhada no arquivo de `HISTÓRICO` do `mSQL` do ultimo ano e compará-lo com a seção `Novidades` do Manual de Referência do MySQL. (see [\[News\]](#), [page](#)). É bem óbvio quem tem desenvolvido mais rapidamente.

Programas utilitários

Ambos `mSQL` e MySQL tem várias ferramentas interessantes de terceiros. Como é muito fácil portar (do `mSQL` para MySQL), quase todas as aplicações interessantes que são disponíveis para `mSQL` também está disponível para MySQL.

O MySQL vem com um programa simples, `mysql2mysql` que corrige diferenças nas sintaxes entre o `mSQL` e o MySQL para as funções C API mais usadas. Por exemplo, ele altera instancias de `mysqlConnect()` para `mysql_connect()`.

Converter um programa cliente de `mSQL` para `MySQL` levam alguns poucos minutos.

1.5.1.1 Como converter ferramentas `mSQL` para o `MySQL`

Por experiência própria, é necessário apenas algumas horas para converter ferramentas como a `mysql-tcl` e `mysqljava` que usam a API em C do `mSQL` para que eles trabalhem com API C do `MySQL`.

O procedimento de conversão é:

1. Execute o script shell `mysql2mysql` na fonte. Isto necessita o programa `replace`, que é distribuído com o `MySQL`.
2. Compile.
3. Corrija todos erros de compilação

As diferenças entre a API C do `mSQL` e a API C do `MySQL` são:

- O `MySQL` usa uma estrutura `MYSQL` como um tipo de conexão, (`mSQL` usa um `int`).
- o `mysql_connect()` recebe um ponteiro para uma estrutura `MYSQL` como um parâmetro. É fácil definir um globalmente ou usar `malloc()` para obter um. `mysql_connect()` também recebe dois parâmetros para especificar o usuário e senha. Você pode configurá-los para `NULL`, `NULL` para uso padrão.
- `mysql_error()` recebe uma estrutura `MYSQL` como um parâmetro. Apenas adicione o parâmetro para código antigo `mysql_error()` se você estiver portanto o código antigo.
- O `MySQL` retorna um número de erro e uma mensagem texto de erro para todos erros. O `mSQL` retorna somente a mensagem de erro.
- Algumas incompatibilidades existem como o resultado do `MySQL` suportar múltiplas conexões para o servidor do mesmo processo.

1.5.1.2 Diferenças entre os protocolos de comunicação do `mSQL` e do `MySQL` Cliente/servidor

Existem bastantes diferenças que é impossível (ou pelo menos, não é fácil) suportar ambos. O que mais caracteriza as diferenças entre o protocolo do `MySQL` e o protocolo do `mSQL` estão listadas abaixo:

- Um buffer de memória pode conter várias linhas de resultado.
- Os buffers de mensagens crescem dinamicamente se a sua query ou o resultado for maior que o buffer atual, chega até a configuração do servidor e o limite do cliente.
- Todos pacotes são numerados para capturar pacotes perdidos ou duplicados.
- Todos valores das colunas são enviados em ASCII. Os tamanhos de colunas e linhas são enviados empacotados em uma codificação binária (1, 2 ou 3 bytes).
- O `MySQL` pode ler no resultado que não está registrado (sem necessidade de armazenar o conjunto inteiro no cliente).
- Se uma simples leitura/escrita demorar mais que 30 segundos, o servidor encerra a conexão.
- Se uma conexão estiver ociosa por 8 horas, o servidor encerra a conexão.

1.5.1.3 Diferenças entre as sintaxes SQL do mSQL 2.0 e do MySQL

Tipos de Campos

O MySQL Tem os seguintes tipos adicionais (entre outros; see [\[CREATE TABLE\]](#), [page](#)):

- Tipo ENUM para um de um conjunto de strings.
- Tipo SET para vários de um conjunto de strings.
- Tipo BIGINT para inteiros com 64bits.

MySQL também suporta os seguintes atributos de tipos adicionais:

- Opção UNSIGNED para colunas com números inteiros.
- Opção ZEROFILL para colunas com números inteiros.
- Opção AUTO_INCREMENT para colunas com números inteiros que são CHAVE PRIMÁRIA. See [\[mysql_insert_id\(\)\]](#), [page](#).
- Valor DEFAULT para todos os tipos de colunas.

mSQL2 tipos de campo mSQL correspondem aos tipo MySQL mostrados abaixo:

Tipo mSQL	Tipo MySQL correspondente
CHAR(<i>tam</i>)	CHAR(<i>tam</i>)
TEXT(<i>tam</i>)	TEXT(<i>tam</i>). <i>tam</i> é o tamanho máximo. E LIKE funciona.
INT	INT. Com muito mais opções!
REAL	REAL. Ou FLOAT. Estão disponíveis as versões 4 e 8 bits.
UINT	INT UNSIGNED
DATE	DATE. Usa o tanto formato ANSI SQL quanto o formato proprietário do mSQL.
TIME	TIME
MONEY	DECIMAL(12,2). Um número de ponto fixo com dois decimais.

Criação de Índices

MySQL Índices podem ser especificados na hora da criação da tabela juntos a instrução CREATE TABLE.

mSQL Índices podem ser criados depois da criação da tabela, com instruções CREATE INDEX separadas.

Para inserir um identificador único na tabela

MySQL Usa o AUTO_INCREMENT como um especificador de tipo de coluna. See [\[mysql_insert_id\(\)\]](#), [page](#).

mSQL

Cria um SEQUENCE em uma tabela e seleciona a coluna `_seq`.

Para obter um identificador único para uma linha

MySQL Adicione uma chave primaria (PRIMARY) ou uma chave UNIQUE para a tabela e a use. Novidades na Versão 3.23.11: Se as chaves primaria (PRIMARY) ou UNIQUE consistirem de somente uma coluna e esta for do tipo inteiro, também podemos nos referenciar a ela como `_rowid`.

mSQL Use a coluna `_rowid`. Observe que `_rowid` pode ser alterado varias vezes, dependendo de alguns fatores.

Obtendo a hora em que uma coluna foi alterada

MySQL Adicionando uma coluna `TIMESTAMP` na tabela. Este campo é configurado automaticamente para receber os valores de data e hora atuais se você não informar algum valor ou fornecer um valor `NULL`.

mSQL Usa o coluna `_timestamp`.

Comparações de Valores NULL

MySQL O MySQL segue o ANSI SQL, portanto qualquer comparação com `NULL` é sempre `NULL`.

mSQL No mSQL, `NULL = NULL` é verdadeiro. Você deve alterar `=NULL` para `IS NULL` e `<>NULL` para `IS NOT NULL` quando portar códigos antigos do mSQL para o MySQL.

Comparações de Strings

MySQL Normalmente, comparações de conjuntos de caracteres são feitas no estilo independente do caso com a forma de classificação determinada pelo conjunto de caracteres atuais (ISO-8859-1 Latin1 por definição padrão). Se você não gosta disto, declare seus campos com o atributo `BINARY`, que deixa as comparações serem realizadas de acordo com a ordenação `ASCII` usada na máquina que hospeda o servidor MySQL.

mSQL Todas as comparações de conjuntos de caracteres são caso sensitivo com ordenação na ordem `ASCII`.

Pesquisas Caso Insensitivo

MySQL `LIKE` é um operador caso-sensitivo ou caso-insensitivo, dependendo dos campos envolvidos. Se possível, o MySQL usa índices se o argumento `LIKE` não iniciar com um meta caracter.

mSQL Usar `CLIKE`.

Manipulando Espaços Excessivos

MySQL Remove todos os espaços no final de colunas `CHAR` e `VARCHAR`. Utilize uma coluna `TEXT` se o comportamento citado não é desejado.

mSQL Mantém os espaços excessivos.

Cláusulas WHERE

MySQL O MySQL prioriza tudo (`AND` é avaliado antes de `OR`). Para obter o compartimento do mSQL no MySQL, use parenteses (como visto no exemplo abaixo).

mSQL Avalia tudo da esquerda para a direita. Isto significa que alguns calculos lógicos com mais de tres argumentos não pode ser expressados. Também significa que você deve alterar algumas queries quando você atualizar para o MySQL. Você pode fazer isto facilmente adicionando parênteses. Suponha que você tenha a seguinte consulta mSQL:

```
mysql> SELECT * FROM tabela WHERE a=1 AND b=2 OR a=3 AND b=4;
```

Para fazer com que o MySQL avalie a sentença da maneira que o mSQL faria, você deve adicionar parênteses:

```
mysql> SELECT * FROM tabela WHERE (a=1 AND (b=2 OR (a=3 AND (b=4))));
```

Controle de Acesso

- MySQL** Tem tabelas para armazenar opções de privilégios (permissões) por usuário, máquina e banco de dados. See [\[Privileges\]](#), page [\[undefined\]](#).
- mSQL** Tem um arquivo 'mSQL.acl' em que você pode conceder privilégios de leitura/escrita para usuários.

1.5.2 Como comparar o MySQL ao PostgreSQL

Quando ler esta seção, por favor perceba que ambos produtos estão continuamente em evolução. Nós na MySQL AB e os desenvolvedores do PostgreSQL estamos ambos trabalhando para fazer os nossos bancos de dados os melhores possíveis, portanto, somos ambos, uma séria escolha sobre qualquer banco de dados comercial.

A seguinte comparação é feita por nós na MySQL AB. Nós tentamos ser o mais preciso e claro possível, mas como não temos pleno conhecimento de todos os recursos do PostgreSQL como temos do MySQL, podemos ter deixado alguns itens errados. Entretanto, iremos corrigi-los que chamarem nossa atenção.

Inicialmente gostaríamos de notar que o PostgreSQL e o MySQL são ambos, produtos intensamente usados, mas com diferentes objetivos, mesmo se ambos empenhassem na compatibilização com o ANSI SQL. Isto significa que para algumas aplicações o MySQL é mais indicado, enquanto para outras, o PostgreSQL possa ser mais. Quando for escolher qual banco de dados usar, você deve inicialmente conferir se o conjunto de recursos do banco de dados satisfaz sua aplicação. Se você precisa de velocidade, o MySQL é provavelmente sua melhor escolha. Se você necessita e alguns recursos extras que somente o PostgreSQL pode oferecer, você deve usar PostgreSQL.

1.5.2.1 Estratégias de desenvolvimento da MySQL e PostgreSQL

Quando adicionamos recursos ao MySQL, orgulhamos de criar uma solução otimizada e definitiva. O código deve ser tão bom que nós não teremos necessidade de alterá-lo em um futuro próximo. Nós também não gostamos de sacrificar velocidade em prol dos recursos, porém faremos o máximo para encontrar uma solução que irá fornecer o máximo de resultados. Isto implica que o desenvolvimento irá ser um pouco demorado, mas o resultado final irá ter uma importância maior. Este tipo de desenvolvimento só é possível porque todo o código do servidor é conferido por poucas (atualmente duas) pessoas antes de ser incluído no servidor MySQL.

Nós na MySQL AB lançamos frequentemente atualizações para que possamos implementar rapidamente novos recursos para nossos usuários. Por isto fazemos um pequeno lançamento a aproximadamente cada 3 semanas, e uma distribuição maior a cada ano. Todos lançamentos são extensamente testados com nossas ferramentas de testes em várias plataformas diferentes.

PostgreSQL é baseado em um kernel com vários colaboradores. Nesta configuração faz sentido priorizar a adição de vários recursos novos, em vez de implementá-los de forma otimizada, porque sempre se pode otimizar itens depois se houver necessidade para isto.

Outra grande diferença entre o MySQL e o PostgreSQL é que quase todo o código no servidor MySQL é feito por desenvolvedores que são contratados pela MySQL AB e continuam trabalhando no código do servidor. As exceções são os mecanismos de transações e a biblioteca regexp.

Este é o contraste com o código do PostgreSQL onde a maioria do código é feita por um grande grupo de pessoas em diferentes situações e condições. Só agora os desenvolvedores do PostgreSQL anunciaram que o grupo de desenvolvimento atual finalmente teria tempo para revisar todo o código na versão atual.

Ambos os métodos de desenvolvimento citados acima tem seus próprios méritos e benefícios. É claro que nós na MySQL pensamos que nosso modelo é melhor pois ele fornece melhor consistência no código, um código reutilizável e otimizado e na nossa opinião, menos erros. Pelo fato de sermos os autores do código do servidor MySQL, nós estamos melhor adaptados para coordenar novos recursos e lançamentos.

1.5.2.2 Comparações de recursos entre o MySQL e PostgreSQL

Na página [crash-me](#) você pode encontrar uma lista das construções e limites dos bancos de dados que podem ser detectados automaticamente com um programa. Perceba entretanto que várias dos limites numéricos podem ser alterados com opções de inicialização para respectivos bancos de dados. O página web acima é extremamente útil quando você quer assegurar que sua aplicação funcione com vários bancos de dados diferentes ou quando você desejar converter sua aplicação de um banco de dados para outro.

O MySQL oferece as seguintes vantagens sobre o PostgreSQL:

- O MySQL é normalmente muito mais rápido que o PostgreSQL. See [\[MySQL-PostgreSQL benchmarks\]](#), page [\[MySQL-PostgreSQL benchmarks\]](#).
- O MySQL tem uma base de usuários muito maior que o PostgreSQL, portanto o código é mais testado e têm sido historicamente mais estável que o PostgreSQL. O MySQL é muito mais usado nos ambientes de produção do que o PostgreSQL, na maioria graças à MySQL AB, formalmente TCX DataKonsult AB, que tem fornecido excelencia na qualidade do do suporte comercial para o MySQL desde o dia de seu lançamento, onde que até recentemente o PostgreSQL não era suportado.
- O MySQL trabalha melhor no Windows do que o PostgreSQL. O MySQL executa como uma aplicação Windows nativa (um serviço no NT/Win2000/WinXP), enquanto o PostgreSQL é executado sobre a emulação cygwin. Nós temos ouvido que o PostgreSQL não é ainda estável no windows mas não temos capacidades para afirmar.
- O MySQL tem mais APIs que outras linguagens e é suportado por mais programas existentes que o PostgreSQL. See [\[Contrib\]](#), page [\[Contrib\]](#).
- O MySQL trabalha em sistemas pesados 24/7. Na maioria das circunstâncias você nunca precisará fazer qualquer limpeza no MySQL. O PostgreSQL não suporta ainda sistemas 24/7 porque em determinado momento deve ser executado o `VACUUM()` para recuperar espaço dos comandos `UPDATE` e `DELETE` e executar analizes estatísticas que são críticas para obter boa performance com o PostgreSQL. `VACCUM()` é também necessário

depois de adicionar vários registros em uma tabela. Em um sistema ocupado com várias alterações, `VACUUM()` deve ser executado muito frequentemente, nos piores casos mesmo muitas vezes ao dia. Durante a execução do `VACUUM()`, que pode durar horas se o banco de dados for grande, o banco de dados num ponto de vista de produtividade fica praticamente morto. A equipe PostgreSQL tem corrigido isto no seu TODO, mas nós assumimos que isto não é uma coisa fácil de ser corrigida permanentemente.

- O recurso de replicação funciona e é usado por sites como:
 - Yahoo Finance (<http://finance.yahoo.com>)
 - Mobile.de (<http://www.mobile.de/>)
 - Slashdot (<http://www.slashdot.org>)
- É incluído nas distribuições MySQL dois tipos diferentes de testes: ‘`mysql-test-run`’ e `crash-me`, bem como o de medida de velocidade. O sistema de testes é ativamente atualizado com código para testar cada novo recurso e com quase todos erros repetidos que chamaram nossa atenção. Nós testamos o MySQL com isto em várias plataformas antes de todos lançamentos. Estes testes são mais sofisticados que qualquer um que vimos para o PostgreSQL, e ele assegura que o MySQL é mantido em um alto padrão.
- Existem muito mais livros impressos sobre MySQL do que sobre PostgreSQL. O’Reilly, Sams, Que e New Rider são todas grandes editores com livros sobre MySQL. Todos recursos MySQL são também documentados no manual on-line, porque, quando um novo recurso é implementado é exigido que os desenvolvedores do MySQL o documentem antes de ser incluído na fonte.
- O MySQL suporta mais funções básicas do ODBC que o PostgreSQL.
- O MySQL tem um `ALTER TABLE` muito mais sofisticado.
- O MySQL tem suporte para tabelas não transacionais para aplicações que necessitam toda velocidade que podem ter. As tabelas podem ser baseadas em memória, tabelas `HEAP` ou baseadas em disco `MyISAM`. See [\[Table types\]](#), page [\[undefined\]](#)
- O MySQL tem suporte para dois manipuladores de tabelas diferentes que suportam transações, `BerkeleyDB` e `InnoDB`. Porque todo mecanismo transacional executa diferentemente sobre condições diferentes, isto fornece ao programador mais opções para encontrar uma solução otimizada para sua configuração. See [\[Table types\]](#), page [\[undefined\]](#)
- Tabelas consolidadas com `MERGE` lhe fornece uma maneira única para criar uma vista instantânea sobre um conjunto de tabelas idênticas e usa-las como uma. Isto é perfeito para sistemas onde você tem arquivos log que você ordena, por exemplo por mês. See [\[MERGE\]](#), page [\[undefined\]](#).
- A opção para comprimir tabelas somente leitura, mas ainda ter acesso direto aos registros na tabela, fornece melhor performance minimizando leituras em disco. Isto é muito útil quando você estiver arquivando coisas. See [\[myisampack\]](#), page [\[undefined\]](#).
- O MySQL tem suporte interno para pesquisas textuais. See [\[Fulltext Search\]](#), page [\[undefined\]](#).
- Você pode acessar vários bancos de dados da mesma conexão (dependendo é claro de seus privilégios).

- O MySQL é codificado desde o início para ser multi-thread enquanto o PostgreSQL usa processos. Troca de contexto e acesso a áreas comuns de armazenamento é muito mais rápido entre threads do que entre processos separados, isto fornece ao MySQL uma grande vantagem em velocidade em aplicações multi-usuárias e também torna muito mais fácil para o MySQL obter vantagem total de sistemas de multiprocessamento simétrico (SMP).
- O MySQL tem um sistema de privilégios muito mais sofisticado do que o PostgreSQL. Enquanto o PostgreSQL suporta permissões somente para INSERT, SELECT e UPDATE/DELETE por usuário em um banco de dados ou uma tabela, o MySQL permite que você defina um conjunto completo de diferentes privilégios nos bancos de dados, tabelas e campos. O MySQL também lhe permite especificar as combinações de privilégios por máquina e usuários. See [\(undefined\) \[GRANT\], page \(undefined\)](#).
- O MySQL suporta um protocolo cliente/servidor comprimido que aumenta a performance em conexões lentas.
- O MySQL trabalha com um conceito de “table handler”, e é o único banco de dados relacional que conhecemos construído neste conceito. Isto permite diferentes tipos de tabela em baixo nível serem trocadas no mecanismo SQL e cada tipo de tabela ser otimizado para diferentes características de performance.
- Todos os tipos de tabelas MySQL (exceto **InnoDB**) são implementados como arquivos (uma tabela por arquivo), que facilita muito o backup, mudança e deleção e mesmo conexões simbólicas entre bancos de dados e tabelas, mesmo se o servidor estiver desligado.
- Ferramentas para corrigir e otimizar tabelas **MyISAM** (o tipo de tabelas MySQL mais comum). Um ferramenta de reparos só é necessária quando uma corrupção física do arquivo de dados acontece, normalmente por uma falha no equipamento. Elas permitem que a maioria dos dados sejam recuperados.
- Atualizar o MySQL é indolor. Quando você estiver atualizando o MySQL, não precisará de apagar/restaurar os seus dados, como você deve fazer na maioria das atualizações do PostgreSQL.

Desvantagens do MySQL em comparação ao PostgreSQL:

- O suporte a transação no MySQL não é tão bem testado como no sistema PostgreSQL.
- Como o MySQL utiliza threads, os quais ainda não estão completos em vários Sistemas Operacionais, deve-se também usar binários de <http://www.mysql.com/downloads>, ou cuidadosamente seguir nossas instruções em http://www.mysql.com/doc/I/n/Installing_source.html para obter um binário otimizado que funcione em todos os casos.
- Bloqueios de tabelas, usados pelas tabelas não transacionais MyISAM é, em muitos casos, mais rápidos que bloqueios de páginas, bloqueios de registros ou versionamento. A desvantagem entretanto é que se alguém não levar em consideração como funciona o bloqueio de tabelas, simplesmente, uma única query que demora muito tempo para executar, pode bloquear uma tabela para atualizações por um grande tempo. Isto pode ser evitado quando for desenhar a aplicação. Se não, pode-se sempre trocar a tabela e utilizar uma das tabelas do tipo transacional. See [\(undefined\) \[Table locking\], page \(undefined\)](#).

- Com o UDF (Funções definidas pelo usuário) pode-se estender o MySQL com funções SQL normais e agregadas, mas isto ainda não é tão fácil e flexível como no PostgreSQL. See [\[Adding functions\]](#), page [\[Adding functions\]](#).
- Atualizações e remoções que executam sobre múltiplas tabelas são mais difíceis para serem feitas no MySQL. Isto irá, entretanto, ser corrigido no MySQL 4.0 com DELETE multi tabelas e UPDATE multi tabelas e no MySQL 4.1 com subselects.

O PostgreSQL atualmente oferece as seguintes vantagens sobre o MySQL:

Note que como conhecemos o mapa do MySQL, nós incluímos na seguinte tabela a versão na qual o MySQL deve suportar o recurso. Infelizmente nós não podemos fazer isto em na comparação anterior, porque nós não conhecemos o mapa do PostgreSQL.

Recurso	versão MySQL
Subselects	4.1
Foreign keys (Chaves Estrangeiras)	4.0 e 4.1
Views	4.2
Stored procedures	4.1
Sistemas de tipos extensivos	Sem planejamento
Unions	4.0
Full join	4.0 or 4.1
Triggers	4.1
Constraint	4.1
Cursors	4.1 or 4.2
Tipos de índices extensivos como R-trees	R-trees estão planejados para 4.2
Tabelas com herança	Sem planejamento

Outras razões para usar o PostgreSQL:

- O uso comum do PostgreSQL é mais próximo ao ANSI SQL em alguns casos.
- O PostgreSQL pode ficar mais rápido codificando coisas como stored procedures.
- PostgreSQL tem uma equipe de desenvolvedores maior que contribuem com o servidor.

Desvantagens do PostgreSQL comparado ao MySQL:

- A cláusula VACUUM() dificulta o uso do PostgreSQL em um ambiente 24/7.
- Somente tabelas transacionais.
- INSERT, DELETE UPDATE muito mais lentos.

Para uma lista completa de desvantagens, você deve também examinar a primeira tabela nesta seção.

1.5.2.3 Comparações de velocidade entre o MySQL e o PostgreSQL

O único benchmark open source que nós conhecemos que pode ser usados para testar a velocidade do MySQL e do PostgreSQL (e outros bancos) é o nosso próprio. Ele pode ser encontrado em <http://www.mysql.com/information/benchmarks.html>.

Várias vezes solicitamos ajuda aos desenvolvedores e alguns usuários do PostgreSQL para estender esse software de benchmark para fazer dele o teste definitivo para Bancos de Dados, mas infelizmente nós não estamos tendo retorno sobre isto.

Por isto, nós, os desenvolvedores do MySQL gastamos várias horas para obter performance máxima do PostgreSQL para os benchmarks, mas como não conhecemos PostgreSQL intimamente, temos certeza que existem detalhes que podem ter sido esquecidos. Nós temos na página de testes em nosso site a documentação de como fizemos os testes para que seja fácil para qualquer um repetir e verificar nossos resultados.

Os testes comparativos são normalmente executados com e sem a opção `--fast`. Quando executado com `--fast` nós estamos tentando usar todos os truques que o servidor pode fazer para que o código execute o mais rápido possível. A idéia é que a execução normal deve mostrar como o servidor deve trabalhar numa configuração padrão e a execução com `--fast` mostra como o servidor comportaria se o desenvolvedor da aplicação usar extensões no servidor para fazer sua aplicação executar mais rápida.

Quando executando com o PostgreSQL e `--fast` nós fazemos um `VACUUM` depois de quase todos maiores `UPDATE` e `DROP TABLE` para deixar o banco de dados em forma perfeita para as `SELECTs` seguintes. O tempo para `VACUUM()` é medido separadamente.

Quando executando com o PostgreSQL 7.1.1 não podemos, entretando, executar com `--fast` porque durante o teste de `INSERT`, o postmaster (daemon do PostgreSQL) finalizou e o banco de dados ficou tão corrompido que se tornou impossível reiniciar o postmaster. Depois disto ter acontecido duas vezes, decidimos adiar o teste com o `--fast` até a próxima versão do PostgreSQL. Os detalhes sobre a máquina na qual executamos o teste pode ser encontrado na página de testes comparativos em nosso site.

Antes de irmos para os outros testes comparativos que conhecemos, gostaríamos de fornecer algumas informações sobre os testes:

É muito fácil escrever um teste demonstrando que ALGUM banco de dados é o melhor do mundo, restringindo o teste apenas a alguma parte em que o banco de dados seja muito bom e não testar outras partes que podem não ser. Se alguém publica os resultados com uma simples figuração as coisas se tornam mais fáceis ainda.

Seria como nós medíssemos a velocidade do MySQL comparada ao PostgreSQL olhando para o resumo de tempo dos testes do MySQL na nossa página. Desta forma o MySQL poderia ser 40 vezes mais rápido que o PostgreSQL, algo que com certeza não é verdade. Poderíamos deixar a coisa ainda pior se pegássemos o teste onde o PostgreSQL atuou pior e falar que o MySQL é mais de 2000 vezes mais rápido que o PostgreSQL.

O caso é que o MySQL faz várias otimizações que o PostgreSQL não faz. É claro que isto também é verdade do outro ladoa. Um otimizador SQL é uma coisa muito complexa e uma empresa pode gastar anos apenas para deixar o otimizador cada vez mais rápido.

Se você estiver consultando os resultados dos testes procure por coisas que sua aplicação utilize e use estes resultados apenas para decidir qual banco de dados será o melhor para a sua aplicação. Os resultados dos testes comparativos também mostram onde um banco de dados em particular não é bom e deve lhe fornecer uma noção sobre coisas que devem ser evitadas e o que você deve fazer de outras maneiras.

Conhecemos dois testes comparativos que mostram o PostgreSQL com mais velocidade que o MySQL. Os dois são testes multi-usuários, um teste que nós da MySQL AB ainda não tivemos tempo para desenvolver e incluir no conjunto de testes comparativos principalmente porque é uma enorme tarefa fazer isto de uma maneira justa para todos os bancos de dados.

Um é o teste comparativo pago da Great Bridge, o qual você pode ler a respeito em: http://www.greatbridge.com/about/press.php?content_id=4.

Este é provavelmente o pior software de testes que já vimos alguém conduzir. Ele não só foi preparado para testar exatamente o que o PostgreSQL tem de melhor, mas além disso foi também injusto contra todos os outros bancos de dados envolvidos no teste.

NOTA: Sabemos que nem mesmo alguns dos principais desenvolvedores do PostgreSQL gostam da maneira que a Great Bridge conduziu os testes comparativos, então não os culpamos pela maneira que os testes foram realizados.

Este comparativo tem sido condenado em várias mensagens postadas em listas de discussões portanto falaremos aqui apenas das coisas que estavam erradas com ele.

- Os testes foram executados com uma cara ferramenta comercial, que torna impossível para uma empresa open source como nós confirmar os resultados, ou mesmo conferir como os testes realmente foram feitos. A ferramenta nem mesmo é uma ferramenta de testes comparativos de verdade, mas uma ferramenta de testes de aplicações e configuração. Dizer que ela é uma ferramenta “padrão” de testes comparativos é esconder a verdade.
- A própria Great Bridge admitiu que eles otimizaram os testes para o banco de dados PostgreSQL (com `VACUUM()` antes dos testes) e preparou os testes, coisa que não foi feita para nenhum dos outros bancos de dados envolvidos. Para dizer “Este processo otimiza índices e libera um pouco de espaço em disco. Os índices otimizados aumentaram a velocidade em alguns casos.” Nossos testes claramente indicam que a diferença em executar várias selects em um banco de dados com e sem `VACUUM()` pode facilmente diferenciar de um fator de dez.
- Os resultados dos testes também foram estranhos. A documentação do teste AS3AP menciona que o teste faz “seleções, joins simples, projeções, agregados, atualizações de uma tupla e atualizações em massa (bulk updates)”.

O PostgreSQL é bom fazendo `SELECTs` e `JOINs` (especialmente depois de um `VACUUM()`), mas não atua tão bem em `INSERTs` ou `UPDATEs`. Os comparativos parecem indicar que somente `SELECTs` foram feitos (ou muito poucas atualizações). Isto pode facilmente explicar porque os resultados foram tão bons para o PostgreSQL neste teste. Os maus resultados para o MySQL obviamente rebaixa este documento.

- Eles executaram o tão chamado benchmark de uma máquina Windows acessando uma máquina Linux sobre ODBC, uma configuração que nenhum usuário de banco de dados comum faria se estivesse executando uma aplicação pesada multi-usuária. Isto testou mais o driver ODBC e o protocolo Windows usado entre os clientes do que o próprio banco de dados.
- Executando o banco de dados contra o Oracle e o MS-SQL (A Great Bridge indicou indiretamente os bancos de dados que eles usaram no teste), eles usaram o ODBC no lugar do protocolo nativo. Qualquer um que já tenha usado o Oracle sabe que aplicações reais devem usar a interface nativa e não ODBC. Fazer um teste através de ODBC e reivindicar que os resultados não tinham relação alguma com as situações de uso na vida real não deve ser tomado como justo. Eles deveriam ter feito dois testes com e sem ODBC para fornecer os fatos reais (depois de contratar especialistas para preparar os bancos de dados envolvidos).
- A Great Bridge faz referências aos testes TPC-C, mas não mencionam em lugar algum que o teste que fizeram não foi realmente um teste TPC-C e eles nem mesmo estavam permitidos a chamá-lo de teste TPC-C. Um teste TPC-C só pode ser conduzido

pelas regras aprovadas pelo conselho TPC (<http://www.tpc.org>). A Great Bridge não fez isto. Com isto, violaram a marca registrada TPC e descreditado seus próprios comparativos. As regras criadas pelo conselho TPC são muito estritas para garantir que ninguém produza falsos resultados ou relatos improváveis. Aparentemente Great Bridge não está interessada em fazer isto.

- Depois do primeiro teste, nós entramos em contato com a Great Bridge e mencionamos algumas das falhas óbvias que cometeram com o MySQL:
 - Utilizaram uma versão de depuração do nosso driver ODBC
 - Executaram em um sistema Linux que não era otimizado para threads
 - Utilizaram uma versão antiga do MySQL quando existia uma nova versão mais recomendada para o uso, já disponível
 - Não inicializaram o MySQL com as opções corretas para uso em sistemas multi-usuários pesados (a instalação padrão do MySQL é preparada para uso mínimo de recursos).

A Great Bridge executou um novo teste, com nosso driver ODBC otimizado e com melhores opções de inicialização para o MySQL, mas recusaram usar a nossa biblioteca glibc atualizada ou nosso binário padrão (usado por 80% de nossos usuários), que utiliza com uma biblioteca glibc corrigida.

Segundo o que sabemos, a Great Bridge não fizeram nada para assegurar que os outros bancos de dados foram configurados corretamente para executar bem em seu ambiente de testes. Nós temos certeza, entretanto, que eles não entraram em contato com a Oracle ou a Microsoft para pedir seu conselho nesta questão ;)

- Os testes comparativos foram pagos pela Great Bridge e eles decidiram publicar somente resultados parciais selecionados (em vez de publicar todo o resultado).

Tim Perdue, um fã assíduo do PostgreSQL e relutante usuário MySQL publicou uma comparação em [phpbuilder](#).

Quando ficamos sabendo da comparação, ligamos para Tim Perdue pois existiam muitas coisas estranhas nos seus resultados. Por exemplo, ele alega que o MySQL tem um problema com cinco usuários nos seus testes, enquanto sabemos que existem usuários com equipamentos similares que usam o MySQL com cerca de 2000 conexões simultâneas executando 400 consultas por segundo. (Neste caso o limite foi a banda de rede, não o banco de dados.)

Nos pareceu que ele estivesse usando um Kernel Linux que tinha algum problema com várias threads, como nos kernels antes da versão 2.4, que possuía um problema com threads em máquinas multi-processadas. Nós documentamos neste manual a forma para corrigir isto e Tim deve estar ciente deste problema.

Outro possível problema pode ter sido uma biblioteca glibc antiga e que Tim não tenha usando um binário MySQL de nosso site, que é utiliza com uma biblioteca glibc correta, porém compilou uma versão própria. Em qualquer dos casos acima, o sintoma poderia ter sido exatamente o que Tim mediu.

Perguntamos ao Tim se poderíamos ter acesso aos dados para que pudéssemos repetir a avaliação e se ele poderia conferir a versão MySQL na máquina para descobrir o que estava errado e ele prometeu nos dar um retorno. Isto ainda não foi feito.

Por causa disto não podemos depositar nossa confiança nestes testes :(

As coisas mudaram com o tempo e os testes acima não são mais relevantes. O MySQL agora tem alguns diferentes manipuladores de tabela com diferente custo-benefício de velocidade/concorência. See [\[Table types\]](#), page [\[Table types\]](#). Seria interessante ver como os testes anteriores poderiam executar com os diferentes tipos de tabelas transacionais no MySQL. É claro que o PostgreSQL também ganhou novos recursos desde que o teste foi feito. Como o teste acima não foi publicamente disponível não existe forma para nós sabermos como o banco de dados poderá atuar nos mesmos testes hoje.

Conclusão:

Os únicos benchmarks que existem hoje que qualquer um pode baixar e executar contra o MySQL e o PostgreSQL são os testes MySQL. Nós, na MySQL, acreditamos que bancos de dados com código aberto devem ser testados com ferramentas de código aberto! Este é o único modo para assegurar que ninguém faça testes que não possam ser reproduzidos e os use para alegar que um banco de dados é melhor que o outro. Sem conhecer todos os fatos é impossível responder às alegações do testador.

O que achamos estranho é que todos os testes que temos visto sobre o PostgreSQL, que é impossível de ser reproduzido, alega que o PostgreSQL é melhor na maioria dos casos enquanto em nossos testes, que qualquer um pode reproduzir, mostra claramente o contrário. Com isto nós não queremos dizer que o PostgreSQL não é bom em várias coisas (ele é!) ou que ele não é mais rápido que o MySQL sobre certas condições. Nós só gostaríamos de ver um teste justo que seja realmente muito bom, para que então pudessemos ter uma competição amigável!

Para maiores informações sobre nosso pacote de testes See [\[MySQL Benchmarks\]](#), page [\[MySQL Benchmarks\]](#).

Estamos trabalhando em um pacote de testes ainda melhor, incluindo testes multi usuários, e uma melhor documentação do que os testes individuais realmente fazem e como adicionar mais testes ao pacote.

1.6 MySQL e o futuro (o TO-DO)

Este apêndice lista os recursos que planejamos implementar no MySQL.

Tudo nesta lista está, aproximadamente, na ordem em que serão feitas. Se você desejar alterar a ordem de prioridades, por favor registre uma licença ou suporte e nos diga o que você quer que seja feito mais rapidamente. See [\[Licensing and Support\]](#), page [\[Licensing and Support\]](#).

O plano é que, no futuro, nós suportaremos completamente o padrão ANSI SQL99, mas com várias extensões úteis. O desafio é fazer isto sem sacrificar a velocidade ou comprometer o código.

1.6.1 Coisas que devem constar na Versão 4.0

Nós planejamos fazer da versão 4.0 uma distribuição “rápida” na qual só adicionamos coisas novas para que outras pessoas possam nos ajudar no desenvolvimento de novos recursos para a versão 4.1. O MySQL versão 4.0 deve levar aproximadamente um mês para que possamos torná-la estável e iniciar o desenvolvimento da versão 4.1. A versão 4.0 deve ter os seguintes recursos:

A seção de novidades para a 4.0 inclui uma lista dos recursos que nós já implementamos na árvore 4.0. See [\[News-4.0.x\]](#), page [\[undefined\]](#).

- Novo formato do arquivo de definição de tabela (arquivos `.frm`). Isto nos permitirá que não tenhamos problemas de bits quando adicionarmos mais opções de tabelas. O formato antigo `.frm` ainda poderá ser usado com o 4.0. Todas novas tabelas, entretanto, usarão o novo formato.

O novo formato de arquivo permitirá adicionar novos tipos de colunas, mais opções para chaves e suporte a **Chave Estrangeira**.

- `mysqld` como uma biblioteca. Isto terá a mesma interface como o cliente padrão MySQL (com uma função extra para configurar apenas parâmetros de inicialização) mas será mais rápido (sem sobrecarga do TCP/IP ou do socket), menor e mais fácil para usar para produtos embutidos.

Poderemos definir em tempo de ligação se queremos usar o modelo cliente/servidor ou uma aplicação isolada apenas definindo com qual biblioteca deve ser ligada.

O `mysqld` irá suportar todos recursos padrão do MySQL e podemos usá-lo num cliente que suporte threads para executar diferentes consultas em cada thread.

- A replicação deve trabalhar com `RAND()` e variáveis de usuário `@var`.
- Backup Online com penalidade mínima de performance. O backup online irá facilitar para adicionar um novo escravo de replicação sem ter que desligar o mestre.
- `DELETE FROM nome_tabela` irá retornar o número de linhas apagadas. Para execução mais rápida use `TRUNCATE nome_tabela`.
- Permite que `DELETE` em tabelas MyISAM use o cache de registros. Para fazer isto, precisaremos atualizar os threads de cache de registros quando atualizarmos o arquivo `.MYD`
- Melhor replicação.
- Mais funções para pesquisas textuais. See [\[Fulltext Features to Appear in MySQL 4.0\]](#), page [\[undefined\]](#).
- Elenco de conjuntos de caracteres e sintaxe para manipular multiplos conjunto.
- Permite usuários para alterar opções de inicialização sem parar o servidor.
- Ajuda para todos comandos no cliente.
- Conexões seguras (com SSL).
- Extender o otimizador para otimizar algumas consultas `ORDER BY key_name DESC`
- `SHOW COLUMNS FROM nome_tabela` (usado pelo cliente `mysql` para permitir expansões de nomes de colunas) não deve abrir a tabela, mas somente o arquivo de definição. Isto irá exigir menos memória e será mais rápido.
- Nova chave de cache
- Quando usar `SET CHARACTER SET` devemos traduzir a toda consulta e não somente strings. Isto irá habilitar os usuários para usarem os caracteres traduzidos no banco de dados, nome de colunas e tabelas.
- Adicionar uma interface portátil sobre `gethostbyaddr_r()` para que possamos alterar `ip_to_hostname()` para não bloquear outras threads enquanto faz consultas DNS.
- Adicionar o método `record_in_range()` para tabelas `MERGE` para ser possível escolher o índice correto quando existir vários que possam ser escolhidas. Devemos também

estender a interface de informações para obter a distribuição de chaves para cada índice, se `analyze` estiver executando em todas sub-tabelas.

- `SET SQL_DEFAULT_TABLE_TYPE=[MyISAM | INNODB | BDB | HEAP]`.

1.6.2 Coisas que devem ser feitas em um futuro bem próximo

- Replicação livre de falhas.
- Subqueries. `select id from t where grp in (select grp from g where u > 100)`
- Tabelas derivadas.

```
select a.col1, b.col2 from (select max(col1) as col1 from root_table ) a,
other_table b where a.col1=b.col1
```

Isto podia ser feito automaticamente criando tabelas temporárias para as tabelas derivadas na duração da consulta.

- Adicionar `PREPARE` de instruções e envio de parâmetros para `mysqld`.
- Estender o protocolo cliente/servidor para suportar avisos.
- Adicionar opções ao protocolo cliente/servidor para obter notas de progresso para comandos de longo tempo de execução.
- Adicionar bancos de dados e nomes reais de tabelas (no caso de apelidos (alias)) à estrutura `MYSQL_FIELD`.
- Não permitir mais que um número definido de threads para executar a recuperação `MyISAM` ao mesmo tempo.
- Alterar `INSERT ... SELECT` para opcionalmente usar inserções concorrentes.
- Implementar `RENAME DATABASE`. Para tornar isto seguro para todos handlers de tabelas, ele funcionará assim:
 - Cria o novo banco de dados.
 - Muda o nome de todas as tabelas para o outro banco de dados, como nós fazemos com o comando `RENAME`.
 - Remove o banco de dados antigo.
- Retornar o tipo do campo original quando fizer `SELECT MIN(column) ... GROUP BY`.
- Múltiplos conjuntos de resultados.
- Alterar o protocolo para permitir transferência binária de valores. Para fazê-lo eficientemente, precisamos adicionar uma API para podermos casar as variáveis.
- Torna possível especificar `long_query_time` com precisão em microsegundos.
- Adicionar um prompt configurável para o cliente de linha de comando `mysql` com opções do tipo banco de dados em uso, hora e data...
- Adicionar conferência do alcance para tabelas `MERGE`.
- Ligar o código `myisampack` no servidor.
- Portar o MySQL para o BeOS.
- Portar os clientes MySQL para LynxOS.
- Adicionar um cache de buffer para chave temporária durante `INSERT/DELETE/ UPDATE` para que possamos fazer uma recuperação fácil se o arquivo de índice encher.

- Se você executar um `ALTER TABLE` em uma tabela que é ligada simbolicamente a outro disco, criar tabelas temporárias neste disco.
- Implementar um tipo `DATE/DATETIME` para manipular fusos horários corretamente para lidarmos mais facilmente com datas em diferentes fusos horários.
- FreeBSD e MIT-pthreads; Deixar threads “dormindo” exigem processamento?
- Conferir se threads travadas exige processamento
- Corrigir o `configure` para que ele possa compilar todas as bibliotecas (como a `MyISAM`) sem threads.
- Adicionar uma opção para periodicamente descarregar páginas chave para tabelas com chaves atrasadas se elas não forem usadas em um certo espaço de tempo.
- Permitir “join” em partes de chaves (questão de otimização).
- `INSERT SQL_CONCURRENT` e `mysqld --concurrent-insert` para fazer inserções concorrentes no final do arquivo se o arquivo for travado para leitura.
- Armazenar definições de chaves `ESTRANGEIRAS` no arquivo `‘.frm’`.
- Cascadear o `DELETE`
- Cursores do lado do servidor.
- Conferir se o `lockd` trabalha com kernels Linux mais modernos; Se não nós teremos que corrigir o `lockd`! Para testar isto, inicie `mysqld` com `--enable-locking` e execute os diferentes pacotes de testes `fork*`. Se eles não retornarem erros, `lockd` funciona.
- Permite variáveis SQL no `LIMIT`, como em `LIMIT @a,@b`.
- Permite atualização de variáveis nas instruções `UPDATE`. Por exemplo: `UPDATE TABLE foo SET @a=a+b, a=@a, b=@a+c`
- Alterar quando variáveis de usuários são atualizadas para que possam ser usadas com `GROUP BY`, como no exemplo seguinte : `SELECT id, @a:=count(*), sum(sum_col)/@a FROM nome_tabela GROUP BY id`.
- Não adicionar valores `DEFAULT` para as colunas. Isto retorna um erro quando se usa um `INSERT` que não contenha uma coluna não tenham um `DEFAULT`.
- Cache de pesquisas e resultados. Isto deve ser feito como um módulo separado que examina cada pesquisa e se a mesma estiver no cache o resultado memorizado deve ser retornado. Quando alguém atualiza uma tabela deve se remover algumas queries do cache. Isto deverá fornecer um grande incremento de velocidade em máquinas com bastante memória onde pesquisas são muito repetidas (como aplicações WWW). Uma idéia seria armazenar no cache somente pesquisas do tipo: `SELECT CACHED ...`
- Corrigir `‘libmysql.c’` para permitir dois comandos `mysql_query()` em uma linha sem ler os resultados ou fornecer uma mensagem de erro quando alguém o fizer.
- Otimizar o tipo `BIT` para ocupar 1 bit (hoje `BIT` ocupa 1 char).
- Checar porque MIT-pthreads `ctime()` não funciona em alguns sistemas FreeBSD.
- Adicionar uma opção `IMAGE` para `LOAD DATA INFILE` para não atualizar campos `TIMESTAMP` e `AUTO_INCREMENT`.
- Adicionar sintaxe `LOAD DATE INFILE... UPDATE`.
 - Para tabelas com chaves primárias, se os dados contêm a chave primária, entradas que combinem com a chave primária são atualizadas do resto das colunas. Entretanto, colunas `FALTANDO` na entrada de dados não são mudadas.

- Para tabelas com chaves primárias que falta alguma parte da chave na entrada de dados, ou que não tenha chave primária, a entrada é tratada como em `LOAD DATA INFILE ... REPLACE INTO`.
- Fazer o `LOAD DATA INFILE` entender sintaxes do tipo:


```
LOAD DATA INFILE 'file_name.txt' INTO TABLE nome_tabela
TEXT_FIELDS (text_field1, text_field2, text_field3)
SET table_field1=concatenate(text_field1, text_field2), table_field3=23
IGNORE text_field3
```

Isto pode ser usado para saltar colunas extras no arquivo texto, ou atualizar colunas baseadas em expressões dos dados lidos...
- `LOAD DATA INFILE 'file_name' INTO TABLE 'nome_tabela' ERRORS TO err_nome_tabela` Esta forma permite que quaisquer erros e avisos sejam logados na tabela `err_nome.tabela`. Essa tabela deve ter uma estrutura como:


```
line_number      - Número da linha no arquivo de dados
error_message    - A mensagem de erro/aviso
e talvez
data_line        - A linha do arquivo de dados
```
- Adicionar real suporte a `VARCHAR` (Já existe suporte para isto no `MyISAM`).
- Saída automática do `mysql` para o `netscape`.
- `LOCK DATABASES`. (com várias opções)
- Alterar ordenação para alocar memória em “grandes partes” para conseguir melhor utilização de memória.
- Tipos `DECIMAL` e `NUMERIC` não podem ler números exponenciais; `Field_decimal::store(const char *from, uint len)` deve ser reescrito para corrigir isto.
- Funções: `ADD_TO_SET(valor,conjunto)` e `REMOVE_FROM_SET(valor,conjunto)`
- Adicionar uso de `t1 JOIN t2 ON ...` e `t1 JOIN t2 USING ...` Atualmente, você só pode usar esta sintaxe com `LEFT JOIN`.
- Adicionar suporte pleno para o tipo `unsigned long long`.
- Várias variáveis a mais para `show status`. Contadores para: Instruções `INSERT/DELETE/UPDATE`. Registros lidos e atualizados. `Selects` em 1 tabela e `selects` com `joins`. Número real de tabelas em uma `select`. Número de consultas `ORDER BY` e `GROUP BY`.
- Se você abortar o `mysql` no meio de uma pesquisa, você deve abrir outra conexão e matar a pesquisa antiga. Alternativamente, deve ser feita uma tentativa para detectar isto no servidor.
- Adicionar uma interface de manipulação para as informações de tabelas para que se possa usá-la como uma tabela de sistema. Pode se tornar um pouco lento se forem requisitadas informações sobre todas as tabelas, mas muito flexível. `SHOW INFO FROM nome_tabela` para informações básicas de tabelas devem também ser implementadas.
- Adicionar suporte para `UNICODE`.
- `NATURAL JOIN` e `UNION JOIN`

- Permitir `select A from crash_me left join crash_me2 using (A)`; Neste caso A é assumido para vir da tabela `crash_me`.
- Corrigir o `ON` e `USING` para funcionar com o tipo de união `JOIN`.
- `CONNECT BY PRIOR . . .`, como Oracle, para psquisas em estruturas hierárquicas.
- `mysqladmin copy database new-database`. – Necessita do comando `COPY` para ser adicionado ao `mysqld`
- Lista de processos deve exibir número de consultas/thread.
- `SHOW HOSTS` exibirá informações sobre o cache de nomes de máquinas.
- Opções `DELETE` e `REPLACE` para a instrução `UPDATE` (esta opção apagará registros quando um erro de chave duplicada for obtido durante a atualização).
- Alterar o formato de `DATETIME` para armazenar frações de segundos.
- Adicionar todos os tipos `ANSI92` e `ODBC 3.0` que ainda faltam.
- Alterar nomes de tabelas de strings vazias para `NULL` para colunas calculadas.
- Não usar 'Item_copy_string' em valores numéricos para evitar conversão de número->string->número no caso de: `SELECT COUNT(*)*(id+0) FROM nome_tabela GROUP BY id`
- Tornar possível o uso da nova biblioteca GNU `regex` em vez da atual (A biblioteca GNU deve ficar muito mais rápida do que a atual).
- Alterar o `ALTER TABLE` para que não aborte clientes que executam `INSERT DELAYED`.
- Corrigir quando as colunas referenciadas em uma cláusula `UPDATE` contém os valores antigos antes da atualização iniciar.
- `myisamchk`, `REPAIR` e `OPTIMIZE TABLE` devem conseguir tratar casos onde os arquivos de dados e/ou índices são links simbólicos.
- Adicionar simulação de `pread()/pwrite()` no Windows para habilitar inserções simultâneas.
- Um analisador de arquivos log que podem analisar informações sobre quais tabelas são acessadas mais frequentemente, qual a frequência que joins de multi-tabelas são executadas, etc. Ele deverá ajudar os usuários a identificar áreas ou projetos de tabelas que poderão ser otimizadas para executar consultas muito mais eficientes.
- Adicionar `SUM(DISTINCT)`
- Adicionar funções de grupo `ANY()`, `EVERY()` e `SOME()`. Em ANSI SQL isto só funciona em colunas booleanas, mas nós podemos extendê-las para trabalhar em qualquer coluna/expressão aplicando: `valor == 0 -> FALSO` e `valor <> 0 -> TRUE`.
- Corrigir o tipo para `MAX(coluna)` ser o mesmo que o tipo da coluna.

```

create table t1 (a DATE);
insert into t1 values (now());
create table t2 select max(a) from t1;
show columns from t2;

```
- Vir com uma nova sintaxe para uma expressão que irá fazer um `UPDATE` no registro se ele existir e fazer um `INSERT` de um novo registro se o mesmo não existir. (Como o `REPLACE` trabalha com `INSERT / DELETE`)

1.6.3 Coisas que em algum dia devem ser feitas

- Implementar função: `get_changed_tables(timeout,table1,table2,...)`
- Atualizações atômicas multi-tabelas, ex: `update items,month set items.price=month.price where items.id=month.id;;`
- Alterar leitura através de tabelas para usar mapeamento de memória quando possível. Atualmente somente tabelas compactadas usam mapeamento de memória.
- Adicionar um novo privilégio '**Show_priv**' para comandos **SHOW**.
- Tornar o código de timestamp automático melhor. Adicionar timestamps para o log de atualizações com `SET TIMESTAMP=#;`
- Usar mutex de leitura/escrita em alguns lugares para obter maior velocidade.
- Suporte pleno a chave estrangeira. Provavelmente implementaremos uma linguagem procedural primeiro.
- Views simples (inicialmente em uma tabela, depois em qualquer expressão).
- Fechar algumas tabelas automaticamente se uma tabela, tabela temporária ou arquivos temporários obtiverem o erro 23 (não pode abrir arquivos suficientes).
- Quando alguém encontra um campo=#, mude todas as ocorrências do campo para #. Agora isto é feito somente para alguns casos simples.
- Alterar todas expressões const com expressões calculadas se possível.
- Chave otimizada = expressão. No momento somente a chave = campo ou a chave = constante são otimizadas.
- Melhorar o código de algumas das funções de cópia
- Alterar '`sql_yacc.yy`' para um analisador em linha para reduzir seu tamanho e obter melhores mensagens de erro (5 dias).
- Alterar o analisador para usar somente uma regra para diferentes números de argumentos em uma função.
- Utilizar nomes de cálculo completos na parte de ordenação. (For ACCESS97)
- UNION, MINUS, INTERSECT e FULL OUTER JOIN. (Atualmente somente o LEFT OUTER JOIN é suportado)
- Permitir UNIQUE em campos que podem ser NULL.
- `SQL_OPTION MAX_SELECT_TIME=#` para colocar um limite de tempo em uma pesquisa.
- Fazer o log de atualizações gravar em um banco de dados.
- LIMIT negativo para recuperar dados do fim.
- Alarmes em funções clientes de conexão, leitura e escrita.
- Por favor, perceba as alterações ao `safe_mysqld`: de acordo com o FSSTND (que o Debian tenta seguir) arquivos PID dever ir em '`/var/run/<progname>.pid`' e arquivos de log em '`/var/log`'. Seria ótimo se você puder colocar o diretório de dados na primeira declaração de "pidfile" e "log", para que a colocação destes arquivos possa ser alterada com uma simples instrução.
- Permitir um cliente requisitar log.
- Adicionar uso de `zlib()` para arquivos `gzip` para funcionar com `LOAD DATA INFILE`
- Corrigir ordenação e agrupamento de colunas BLOB (parcialmente resolvida agora).

- Stored procedures. Atualmente isto não é tratado com muito importante já que as stored procedures não estão muito bem padronizadas ainda. Outro problema é que verdadeiras stored procedures dificultam muito o otimizador e em vários casos o resultado é mais lento que antes. Nós iremos, por outro lado, adicionar uma linguagem de atualização (atômica) simples que pode ser usada para escrever loops e semelhantes no servidor MySQL.
- Alterar para o uso de semáforos quando contar threads. Devemos primeiro implementar uma biblioteca de semáforos para a MIT-pthreads.
- Não atribuir novos valores AUTO_INCREMENT quando alguém configurar uma coluna para 0. No lugar disto usar NULL.
- Adicionar suporte pleno para JOIN com parênteses.
- Como uma alternativa para uma thread / conexão gerencie uma fila de threads para manipular as pesquisas.
- Permitir obter mais de um bloqueio com GET_LOCK. Quando isto for feito, serão, também, tratados os possíveis deadlocks que essa alteração irá acarretar.

Tempo é fornecido de acordo com a quantidade de trabalho, e não tempo real.

1.6.4 Algumas coisas que não temos planos para fazer

- Nada; Planejamos ser totalmente compatíveis com o ANSI 92 / ANSI 99.

2 Instalação do MySQL

Este capítulo descreve como obter e instalar o MySQL:

- Para uma lista de sites no quais você pode obter o MySQL, veja [Getting MySQL](#), page [Getting MySQL](#).
- Para saber quais são as plataformas suportadas, veja em [Which OS](#), page [Which OS](#). Por favor perceba que nem todas as plataformas suportadas são igualmente boas para executar o MySQL. Algumas são mais robustas e eficientes que outras - ver [Which OS](#), page [Which OS](#) para detalhes.
- Várias versões do MySQL estão disponíveis em distribuições binárias e fonte. Nós também fornecemos acesso público à nossa árvore fonte atual para aqueles que desejam ver nossos desenvolvimentos mais recentes e nos ajudar a testar novos códigos. Para determinar que versão e tipo da distribuição você deve usar, veja [Which version](#), page [Which version](#). Se ainda restar dúvidas, use a distribuição binária.
- Instruções de instalação para distribuições binária e fonte são descritos em [Installing binary](#), page [Installing binary](#) e [Installing source](#), page [Installing source](#). Cada conjunto de instruções inclui uma seção sobre problemas específicos de sistemas que você pode precisar.
- Para procedimentos pós-instalação, veja [Post-installation](#), page [Post-installation](#). Estes procedimentos podem ser aplicados caso você use uma distribuição binária ou fonte do MySQL.

2.1 Instalação rápida padrão do MySQL

2.1.1 Instalando o MySQL no Linux

O caminho recomendado para instalar o MySQL no Linux é usando um arquivo RPM. Os RPMs do MySQL atualmente são construídos na versão 6.2 do sistema RedHat Linux mas deve funcionar em outras versões de Linux que suportam `rpm` e usam `glibc`.

Se você tiver problemas com um arquivo RPM, por exemplo, se você receber o erro “`Sorry, the host 'xxxx' could not be looked up`”, veja [Binary notes-Linux](#), page [Binary notes-Linux](#).

Os arquivos RPM que você provavelmente usará são:

- `MySQL-VERSION.i386.rpm`
O servidor MySQL. Você irá precisar disto a não ser que você só deseje somente conectar a um servidor MySQL executando em outra máquina.
- `MySQL-client-VERSION.i386.rpm`
Os programas clientes padrões do MySQL. Provavelmente você sempre instalará este pacote.
- `MySQL-bench-VERSION.i386.rpm`
Testes e comparativos de performances (benchmarks). Necessita Perl e RPMs `mysql-modules`.

- `MySQL-devel-VERSION.i386.rpm`
Bibliotecas e arquivos include necessários se você desejar compilar outros clientes MySQL, como nos módulos Perl.
- `MySQL-VERSION.src.rpm`
Este contém o código fonte para todos os pacotes acima. Ele também pode ser usado para tentar construir RPMs para outras arquiteturas (por exemplo, Alpha ou SPARC).

Para ver todos os arquivos em um pacote RPM, execute:

```
shell> rpm -qp1 MySQL-VERSION.i386.rpm
```

Para realizar uma instalação mínima padrão, execute:

```
shell> rpm -i MySQL-VERSION.i386.rpm MySQL-client-VERSION.i386.rpm
```

Para instalar somente o pacote cliente, execute:

```
shell> rpm -i MySQL-client-VERSION.i386.rpm
```

O RPM coloca dados em `/var/lib/mysql`. O RPM também cria as entradas apropriadas em `/etc/rc.d/` para iniciar o servidor automaticamente na hora do boot. (Isto significa que se você realizou uma instalação anterior, talvez você deseje criar uma cópia do seu arquivo de inicialização instalado anteriormente se você fez alguma alteração no mesmo, para que você não perca suas alterações.)

Depois de instalar o(s) arquivo(s) RPM, o daemon `mysqld` deve estar rodando e você já deve poder iniciar o uso do MySQL. See [\[Post-installation\]](#), page [\[undefined\]](#).

Se alguma coisa der errado, você encontrar maiores informações no capítulo de instalação. See [\[Installing binary\]](#), page [\[undefined\]](#).

2.1.2 Instalando o MySQL no Windows

As seguintes instruções aplicam para distribuições binárias pré-compiladas. Se você fizer o download de uma distribuição fonte, você deve compilá-lo e instalá-lo por conta própria.

Se você não possui uma cópia de uma distribuição MySQL, a primeira coisa a fazer é o download em <http://www.mysql.com/downloads/mysql-3.23.html>.

Se você deseja conectar ao MySQL de algum outro programa, você provavelmente precisará do driver **MyODBC**, que pode ser encontrado na página de download **MyODBC** (<http://www.mysql.com/downloads/api-myodbc.html>).

Para instalar alguma das distribuições, descompacte-a em algum diretório vazio e execute o programa `Setup.exe`.

Por definição, o MySQL-Windows é configurado para ser instalado em `c:\mysql`. Se você deseja instalar o MySQL em outro lugar, instale-o primeiramente em `c:\mysql`, então mova a instalação para onde você preferir. Se você mover o MySQL, deve indicar onde está localizado usando a opção `--basedir` quando iniciar o servidor. Por exemplo, se você mover a distribuição MySQL para `d:\programs\mysql`, você deve iniciar o `mysqld` assim:

```
C:\> D:\programs\mysql\bin\mysqld --basedir D:\programs\mysql
```

Use `mysqld --help` para mostrar todas as opções que o `mysqld` aceita!

Em todas novas versões do MySQL, você pode também criar um arquivo `c:\my.cnf` que guarda todas as opções padrões para o servidor MySQL. Copie o arquivo `\mysql\my-xxxx.cnf` para `c:\my.cnf` e edite-o para atender suas necessidades. Perceba

que você deve especificar todos caminhos com '/' em vez de '\'. Se você usar '\', você precisa especificá-lo dobrado, porque '\' é o caractere escape no MySQL. See [\[Option files\]](#), page [\[undefined\]](#).

Iniciado com o MySQL 3.23.38, a distribuição Windows inclui ambos binários, normal e o **MySQL-Max**. O principal benefício de usar o binário normal `mysqld.exe` é que ele é um pouco mais rápido e usa menos recursos.

Aqui está uma lista dos diferentes servidores MySQL que você pode usar:

<code>mysqld</code>	Compilado com debugger integral e conferência automática de alocação de memória, links simbólicos, BDB e tabelas InnoDB.
<code>mysqld-opt</code>	Binário otimizado sem suporte para tabelas transacionais.
<code>mysqld-nt</code>	Binário otimizado para Nt com suporte para named pipes. você pode executar esta versão no Win98, mas neste caso não são criados named pipes e você deve ter TCP/IP instalado.
<code>mysqld-max</code>	Binário otimizado com suporte para links simbólicos, tabelas BDB e InnoDB.
<code>mysqld-max-nt</code>	Como o <code>mysqld-max</code> , porém compilado com suporte para named pipes.

Incluído a partir da versão 3.23.50, named pipes estará habilitado somente se o `mysqld` é iniciado com a opção `--enable-named-pipe`

Todos os binários acima são otimizados para o processador Pentium Pro mas deve funcionar em qualquer processador Intel \geq i386.

Nota: Se você deseja utilizar as tabelas InnoDB, existem algumas opções de inicialização que devem ser especificadas no seu arquivo 'my.ini' See [\[InnoDB start\]](#), page [\[undefined\]](#)

2.2 Detalhes Gerais de Instalação

2.2.1 Como obter o MySQL

Confira a [home page do MySQL](#) para informações sobre a versão atual e para instruções de download.

Nosso principal espelho de download está localizado em: <http://download.sourceforge.net/mirrors/my>

Se você está interessado em se tornar um site espelho do MySQL, você deve fazer sincronismo anônimo com: `rsync://download.sourceforge.net/mysql/`. Envie e-mail para webmaster@mysql.com nos notificando sobre seu espelho para ser adicionado na lista abaixo.

Se você tiver problemas com o download de nosso site principal, tente usar algum dos espelhos listados abaixo.

Por favor relate espelhos ruins ou desatualizados ao webmaster@mysql.com.

Europa:

- Austria [Univ. of Technology/Vienna] [WWW FTP](#)
- Bulgaria [online.bg/Sofia] [WWW FTP](#)
- Republica Tcheca [Masaryk University in Brno] [WWW FTP](#)
- Republica Tcheca [www.sopik.cz] [WWW](#)
- Republica Tcheca [www.gin.cz] [WWW FTP](#)

- Dinamarca [Borsen] [WWW](#)
- Dinamarca [SunSITE] [WWW](#) [FTP](#)
- Estonia [OKinteractive] [WWW](#)
- França [mtesa.net] [WWW](#)
- França [fastorama.com, Chatenois] [WWW](#) [FTP](#)
- Finlândia [tonnikala.net] [WWW](#)
- Alemanha [Kernelnotes.de, Bonn] [WWW](#) [FTP](#)
- Alemanha [Wolfenbuettel] [WWW](#) [FTP](#)
- Grécia [NTUA, Athens] [WWW](#) [FTP](#)
- Hungria [Xenia] [WWW](#) [FTP](#)
- Hungria [TiszaneT] [WWW](#) [FTP](#)
- Hungria [stop.hu] [WWW](#)
- Islândia [GM] [WWW](#) [FTP](#)
- Itália [feelinglinux.com] [WWW](#)
- Itália [Teta Srl] [WWW](#)
- Itália [tzone.it] [WWW](#)
- Irlanda [Esat Net] [WWW](#) [FTP](#)
- Letônia [linux.lv] [FTP](#)
- Holanda [Silverpoint] [WWW](#)
- Holanda [Widexs BV] [WWW](#) [FTP](#)
- Holanda [ProServe] [WWW](#)
- Polónia [Sunsite] [WWW](#) [FTP](#)
- Polónia [ncservice.com/Gdansk] [WWW](#)
- Portugal [Netc] [WWW](#) [FTP](#)
- Romênia [roedu.net/Bucharest] [FTP](#)
- Rússia [DirectNet] [WWW](#) [FTP](#)
- Rússia [Scientific Center/Chernogolovka] [FTP](#)
- Suécia [Sunet] [WWW](#) [FTP](#)
- Suíça [Sunsite] [WWW](#) [FTP](#)
- Reino Unido [PLiG/UK] [WWW](#) [FTP](#)
- Ucrânia [PACO] [WWW](#) [FTP](#)
- Ucrânia [ISP Alkar Teleport/Dnepropetrovsk] [WWW](#)
- Iugoslávia [bolex.co.yu] [WWW](#) [FTP](#)

America do Norte:

- Canadá [Tryc] [WWW](#)
- USA [Hurricane Electric/San Jose] [WWW](#)
- USA [ValueClick, Los Angeles CA] [WWW](#) [FTP](#)
- EUA [Wisconsin University/Wisconsin] [WWW](#) [FTP](#)

- EUA [LinuxWired/Scottsdale, AZ] [WWW](#) [FTP](#)
- EUA [adgrafix.com/Boston, MA] [WWW](#)
- EUA [netNumina/Cambridge, MA] [WWW](#)
- EUA [Ahaza Systems/Seattle, WA] [WWW](#) [FTP](#)

América do Sul:

- Argentina [bannerlandia.com] [WWW](#) [FTP](#)
- Chile [Vision] [WWW](#)
- Chile [PSINet] [WWW](#) [FTP](#)
- Chile [Tecnoera] [WWW](#)

Asia:

- China [linuxforum.net] [WWW](#)
- China [HKLPG/Hong Kong] [WWW](#)
- China [Gremlins/Hong Kong] [WWW](#) [FTP](#)
- China [shellhung.org/Hong Kong] [WWW](#) [FTP](#)
- Indonesia [incaf.net] [WWW](#)
- Indonesia [web.id] [WWW](#) [FTP](#)
- Japão [Soft Agency] [WWW](#)
- Japão [u-aizu.ac.jp/Aizu] [FTP](#)
- Coreia do Sul [Webiiz] [WWW](#)
- Coreia do Sul [PanworldNet] [WWW](#)
- Singapura [HJC] [WWW](#) [FTP](#)
- Taiwan [TTN] [WWW](#)
- Taiwan [nctu.edu/HsinChu] [WWW](#)

Africa:

- Africa do Sul [Mweb] [WWW](#)
- Africa do Sul [The Internet Solution/Johannesburg] [FTP](#)

2.2.2 Sistemas Operacionais suportados pelo MySQL

Nós utilizamos o GNU Autoconf, para que seja possível portar o MySQL para todos sistemas operacionais modernos com threads Posix funcionando e um compilador C++. (Para compilar somente o código cliente, um compilador C++ é necessário mas threads não.) Nós mesmos usamos e desenvolvemos o software primariamente em Sun Solaris (Versões 2.5 - 2.7) e SuSE Linux Versão 7.x.

Perceba que para alguns sistemas operacionais, o suporte nativo a thread funciona somente nas últimas versões. O MySQL compila com sucesso nas seguintes combinações de sistema operacional/pacote de thread:

- AIX 4.x com threads nativas. See [\(undefined\)](#) [IBM-AIX], page [\(undefined\)](#).
- Amiga.

- BSDI 2.x com o pacote incluído MIT-pthreads. See [\[BSDI\]](#), page [\[undefined\]](#).
- BSDI 3.0, 3.1 e 4.x com threads nativas. See [\[BSDI\]](#), page [\[undefined\]](#).
- DEC Unix 4.x com threads nativas. See [\[Alpha-DEC-UNIX\]](#), page [\[undefined\]](#).
- FreeBSD 2.x com o pacote incluído MIT-pthreads. See [\[FreeBSD\]](#), page [\[undefined\]](#).
- FreeBSD 3.x e 4.x com threads nativas. See [\[FreeBSD\]](#), page [\[undefined\]](#).
- HP-UX 10.20 com o pacote incluído MIT-pthreads. See [\[HP-UX 10.20\]](#), page [\[undefined\]](#).
- HP-UX 11.x com as threads nativas. See [\[HP-UX 11.x\]](#), page [\[undefined\]](#).
- Linux 2.0+ com LinuxThreads 0.7.1+ ou glibc 2.0.7+. See [\[Linux\]](#), page [\[undefined\]](#).
- Mac OS X Server. See [\[Mac OS X\]](#), page [\[undefined\]](#).
- NetBSD 1.3/1.4 Intel e NetBSD 1.3 Alpha (Necessita GNU make). See [\[NetBSD\]](#), page [\[undefined\]](#).
- OpenBSD > 2.5 com threads nativas. OpenBSD < 2.5 com o pacote incluído MIT-pthreads. See [\[OpenBSD\]](#), page [\[undefined\]](#).
- OS/2 Warp 3, FixPack 29 e OS/2 Warp 4, FixPack 4. See [\[OS/2\]](#), page [\[undefined\]](#).
- SGI Irix 6.x com threads nativas. See [\[SGI-Irix\]](#), page [\[undefined\]](#).
- Solaris 2.5 e superior com threads nativas nas plataformas SPARC e x86. See [\[Solaris\]](#), page [\[undefined\]](#).
- SunOS 4.x com o pacote incluído MIT-pthreads. See [\[Solaris\]](#), page [\[undefined\]](#).
- SCO OpenServer com o port recente do pacote FSU Pthreads. See [\[SCO\]](#), page [\[undefined\]](#).
- SCO UnixWare 7.0.1. See [\[SCO Unixware\]](#), page [\[undefined\]](#).
- Tru64 Unix
- Win95, Win98, NT e Win2000. See [\[Windows\]](#), page [\[undefined\]](#).

Perceba que nem todas as plataformas são apropriadas para executar o MySQL. Os seguintes fatores determinam se uma certa plataforma é apropriada para uma missão crítica pesada:

- Estabilidade geral da biblioteca thread. Uma plataforma pode ter excelente reputação, entretanto, se a biblioteca thread é instável no código que é usado pelo MySQL, mesmo se todo o resto for perfeito, o MySQL irá ser tão estável quanto a biblioteca thread.
- A habilidade do kernel e/ou a biblioteca thread tirar vantagem do **SMP** em sistemas multi-processados. Em outras palavras, quando um processo cria uma thread, deve ser possível para aquela thread executar em uma CPU diferente que o processo original.
- A habilidade do kernel e/ou a biblioteca thread executar várias threads que adquire/libera um bloqueio mutex sobre uma pequena região crítica frequentemente sem trocas de contexto excessivos. Em outras palavras, se a implementação de `pthread_mutex_lock()` requisitar a CPU muito rapidamente, isto irá afetar o MySQL

tremendamente. Se esse detalhe não estiver sendo cuidado, adicionar CPUs extras podem deixar o MySQL mais lento.

- Estabilidade e performance geral do sistema de arquivos.
- Habilidade do sistema de arquivos em lidar com arquivos grandes de forma eficiente, se suas tabelas forem grandes.
- Nosso nível de experiência aqui na MySQL AB com a plataforma. Se nós conhecemos bem uma plataforma, introduzimos otimizações/correções específicas para ela habilitadas na hora da compilação. Nós também podemos fornecer conselhos sobre como configurar seu sistema otimizada para o MySQL.
- O volume de testes feitos internamente de configurações similares.
- O número de usuários que tem executado o MySQL com sucesso naquela plataforma em configurações similares. Se esse número for alto, as chances de se ter alguma surpresa específica da plataforma fica muito menor.

Baseado nos critérios acima, as melhores plataformas para a execução do MySQL até este ponto são o x86 com SuSe Linux 7.1, kernel 2.4 e ReiserFS (ou qualquer distribuição Linux similar) e Sparc com Solaris 2.7 ou 2.8. FreeBSD vem em terceiro, mas realmente temos esperanças que ele irá se unir ao clube dos tops uma vez que a biblioteca thread está melhorando. Nós também acreditamos que em certo ponto iremos estar aptos para incluir todas as outras plataformas em que o MySQL compila e executa, mas não tão bem e com o mesmo nível de estabilidade e performance, na categoria superior. Isto necessitará de algum esforço da nossa parte em cooperação com os desenvolvedores dos componentes do Sistema Operacional/Biblioteca que o MySQL depende. Se você tiver interesse em melhorar algum de nossos componentes, está em uma posição para influenciar seu desenvolvimento, e precisa de instruções mais detalhadas sobre o que o MySQL necessita para uma melhor execução, envie um e-mail para internals@lists.mysql.com.

Por favor, perceba que a comparação acima não é para dizer que um SO é melhor ou pior que o outro em geral. Nós estamos falando sobre a escolha de um SO para um propósito dedicado: executar o MySQL, e comparamos as plataformas levando isto em consideração. Desta forma, o resultado desta comparação seria diferente se nós incluíssemos mais detalhes. E em alguns casos, a razão de um SO ser melhor que o outro pode ser simplesmente porque colocamos mais esforço nos testes e otimização para aquela plataforma em particular. Estamos apenas colocando nossas observações para ajudá-lo na decisão de qual plataforma usar o MySQL na sua configuração.

2.2.3 Qual versão do MySQL deve ser usada

A primeira decisão a ser feita é se você deve usar a última versão de desenvolvimento ou a última versão estável:

- Normalmente, se você estiver usando o MySQL pela primeira vez ou tentando portá-lo para algum sistema em que não exista distribuição binária, recomendamos o uso da versão estável (atualmente Versão 3.23.52). Repare que todos os lançamentos do MySQL são conferidos com os testes comparativos de performance e um conjunto extenso de testes antes de cada lançamento.
- Senão, caso você esteja trabalhando com um antigo sistema e quiser atualizá-lo, mas não que correr o risco com uma atualização sem correções, você deve fazê-lo do mesmo ramo

que você está usando (onde apenas o último número da versão é mais novo que o seu). Nós temos tentado corrigir somente erros fatais e torná-los menores, com alterações relativamente seguras para aquela versão.

A segunda decisão a ser feita é se você deseja usar uma distribuição fonte ou binária. Na maioria dos casos provavelmente você deverá usar a distribuição binária, se alguma existir para sua plataforma, será normalmente muito mais fácil para instalar do que a distribuição em código fonte.

Nos seguintes casos você provavelmente será mais bem servido com uma instalação baseada em código fonte:

- Se você deseja instalar o MySQL em algum lugar específico. (O padrão das distribuições binárias é estar “pronto para rodar” em qualquer lugar, mas talvez você deseje ainda mais flexibilidade).
- Para estar apto e satisfazer diferentes requisições dos usuários, estaremos fornecendo duas versões binárias diferentes; Uma compilada com os manipuladores de tabelas não transacionais (um binário rápido e pequeno) e um configurado com as mais importantes opções estendidas como tabelas transacionais. Ambas versões são compiladas da mesma distribuição fonte. Todos clientes MySQL nativos pode conectar com ambas versões do MySQL.

A distribuição binária estendida é marcada com o sufixo `-max` e é configurada com as mesmas opções de `mysqld-max`. `mysqld-max`. See [\[mysqld-max\]](#), page [\[undefined\]](#).

Se você deseja usar o RPM MySQL-Max, primeiramente você deve instalar o RPM MySQL padrão.

- Se você deseja configurar `mysqld` com alguns recursos extras que NÃO estão nas distribuições binárias. Segue abaixo a lista das opções extras mais comuns que você pode querer usar:
 - `--with-berkeley-db`
 - `--with-innodb`
 - `--with-raid`
 - `--with-libwrap`
 - `--with-named-z-lib` (Isto é feito para alguns dos binários)
 - `--with-debug[=full]`
- A distribuição binária padrão é normalmente compilada com suporte para todos conjuntos de caracteres e deve funcionar em uma variedade de processadores para a mesma família do processador.

Se você precisar de um servidor MySQL mais rápido você pode querer recompilá-lo com suporte para somente o conjunto de caracteres que você precisa, usar um compilador melhor (como `pgcc`) ou usar opções de compiladores para usar otimizações para seu processador.

- Se você encontrar um erro e relatá-lo para o time de desenvolvimento do MySQL você provavelmente receberá um patch que será necessário aplicá-lo para a distribuição fonte para ter o bug corrigido.

- Se você deseja ler (e/ou modificar) o código C e C++ que é o MySQL, você pode obter uma distribuição fonte. O código fonte é sempre o manual final. Distribuições fontes também contem mais testes e exemplos que as distribuições binárias.

O esquema de nomes do MySQL usa números de versões que consistem de tres números e um sufixo. Por exemplo, um nome de lançamento como `mysql-3.21.17-beta` é interpretado da seguinte maneira:

- O primeiro número (3) descreve o formato dos arquivos. Todas releases da Versão 3 tem o mesmo formato de arquivo.
- O segundo número (21) é o nível da distribuição. Normalmente existem dois para serem escolhidos. Um é o ramo estável (atualmente 23) e o outro é o ramo de desenvolvimento (atualmente 4.0). Normalmente ambos são estáveis, mas a versão de desenvolvimento pode acontecer coisas estranhas, faltar documentação em novos recursos, ou mesmo não compilar em alguns sistemas.
- O terceiro número (17 é o número da versão do nível de distribuição. Este é incrementado para cada nova distribuição. Normalmente você desejará a última versão para o nível de publicação que tiver escolhido.
- O sufixo (**beta**) indica o nível de estabilidade da versão. Os possíveis sufixo são:
 - **alpha** indica que a versão contém grandes seções de novos códigos que não foram 100% testados. Bugs conhecidos (normalmente não tem nenhum) devem estar documentados na seção News. See [\[News\]](#), page [\[News\]](#). Existem também novos comandos e extensões na maioria das publicações alpha. Desenvolvimento ativo que podem envolver maiores alterações no código pode ocorrer numa versão alpha, mas tudo será testado antes de fazer a publicação. Não podem existir erros conhecidos em nenhuma publicação do MySQL.
 - **beta** significa que todo o novo código foi testado. Não serão adicionados novos recursos que podem causar algum tipo de corrompimento. Não deve existir bugs conhecidos. Uma alteração de versão de alpha para beta ocorre quando não existir nenhum relato de erro fatal com uma versão alpha por pelo menos um mês e não planejamos adicionar nenhum recurso que pode deixar algum antigo comando menos confiável.
 - **gamma** é o beta que já tem sido usado a algum tempo e parece funcionar bem. Apenas pequenas correções são adicionadas. Isto é o que muitas empresas chamam de release.
 - Se não existir um sufixo, significa que esta versão já está sendo executada há algum tempo em diferentes locais sem relatos de erros além dos específicos de certas plataformas. Somente correções de erros críticos são adicionados ao release. Isto é o que chamamos de uma distribuição estável.

Todas as versões do MySQL funcionam sobre nossos testes padrões e comparativos para garantir que eles são relativamente seguros para o uso. Como os testes padrões são estendidos ao longo do tempo para conferir por todos os bugs antigos encontrados, o pacote de testes continua melhorando.

Perceba que todas publicações de versões foram testadas pelo menos com:

Um pacote de testes interna

Faz parte de um sistema de produção para um cliente. Ela tem diversas tabelas com centenas de megabytes de dados.

O pacote de comparativos da MySQL

Este executa uma série de consultas comuns. É também um teste para ver se o último conjunto de otimizações fez o código mais rápido. See [\[MySQL Benchmarks\]](#), page [\[MySQL Benchmarks\]](#).

O teste `crash-me`

Este tenta determinar quais recursos o banco de dados suporta e quais são suas capacidades e limitações. See [\[MySQL Benchmarks\]](#), page [\[MySQL Benchmarks\]](#).

Outro teste é que nós usamos a versão do MySQL mais nova em nosso ambiente de produção interna, em pelo menos uma máquina. Nós temos mais de 100 gigabytes de dados com que trabalhar.

2.2.4 Layouts de Instalação

Esta seção descreve o layout padrão dos diretórios criados pela instalação das distribuições binária e fonte.

Uma distribuição binária é instalada descompactando-a no local de instalação de sua escolha (tipicamente `/usr/local/mysql`) e cria os seguintes diretórios nesses locais:

Diretório	Conteúdo do diretório
<code>'bin'</code>	Programas clientes e o servidor <code>mysqld</code>
<code>'data'</code>	Arquivos Log, bancos de dados
<code>'include'</code>	Arquivos de cabeçalho (headers)
<code>'lib'</code>	Bibliotecas
<code>'scripts'</code>	<code>mysql_install_db</code>
<code>'share/mysql'</code>	Arquivos de mensagem de erro
<code>'sql-bench'</code>	Benchmarks - testes comparativos

Uma distribuição baseada em código fonte é instalada depois de você configurá-la e compilá-la. Por padrão, a instalação copia os arquivos em `/usr/local`, nos seguintes subdiretórios:

Diretório	Conteúdo do diretório
<code>'bin'</code>	Programas clientes e scripts
<code>'include/mysql'</code>	Arquivos de cabeçalho (headers)
<code>'info'</code>	Documentação no formato Info
<code>'lib/mysql'</code>	Bibliotecas
<code>'libexec'</code>	O servidor <code>mysqld</code>
<code>'share/mysql'</code>	Arquivos com mensagens de erros
<code>'sql-bench'</code>	Benchmarks e o teste <code>crash-me</code>
<code>'var'</code>	Bancos de dados e arquivos log

Dentro de um diretório de instalação, o layout de uma instalação baseada em fontes difere de uma instalação binária nas seguintes formas:

- O servidor `mysqld` é instalado no diretório `'libexec'` no lugar de `'bin'`.
- O diretório de dados é `'var'` ao invés de `'data'`.

- `mysql_install_db` é instalado no diretório `‘/usr/local/bin’` ao invés de `‘/usr/local/mysql/scripts’`.
- Os arquivos headers e diretórios de bibliotecas estão em `‘include/mysql’` e `‘lib/mysql’` em vez de `‘include’` e `‘lib’`.

Você pode criar sua própria instalação binária da compilação dos fontes executando o script `‘scripts/make_binary_distribution’`.

2.2.5 Como e quando as atualizações são lançadas ?

O MySQL está evoluindo muito rapidamente na MySQL AB e nós queremos compartilhar isto com outros usuários MySQL. Sempre que temos alguns recursos úteis que outros acham necessário, tentamos fazer um release.

Também tentamos ajudar usuários que solicitam recursos que são de fácil implementação. Tomamos notas do que nossos usuários licenciados gostariam de ter, especialmente do que nossos clientes com suporte estendido desejam e tentamos ajudá-los.

Não existe uma real necessidade para baixar uma nova release. A seção News irá dizer se a nova versão tem alguma coisa que você precisa. See [\[News\]](#), page [\[News\]](#).

Usamos a seguinte política quando estamos atualizando o MySQL:

- Para cada pequena atualização, o último número na versão é incrementado. Quando tiver um maior número de novos recursos ou menor incompatibilidade com versões antigas, o segundo número na versão é incrementado. Quando o formato de arquivo altera, o primeiro número é aumentado.
- Versões estáveis testadas aparecem na média de uma a duas vezes por ano, mas se pequenos bugs são encontrados, uma versão será lançada apenas com as correções dos erros.
- Releases funcionais aparecem na média a cada 1-8 semanas.
- Distribuições binárias para algumas plataformas será feita por nós somente para releases mais importantes. Outras pessoas podem fazer distribuições binárias para outros sistemas mas provavelmente com menos frequência.
- Nós normalmente disponibilizamos os patches logo que localizamos e corrigimos pequenos bugs.
- Para bugs não críticos, mas irritantes, disponibilizamos patches se eles são enviados para nós. De qualquer forma, iremos combinar vários deles em um patch maior.
- Se existitir, por algum motivo, um bug fatal numa versão criaremos uma nova release o mais cedo possível. Gostaríamos que outras empresas fizessem isto também.

A versão estável atual é a 3.23; nós já mudamos o desenvolvimento em atividade para a versão 4.0. Bugs continuarão a ser corrigidos na versão estável. Não acreditamos em um congelamento completo, pois isto abandona a correções de bugs e coisas que “devem ser feitas.” “Alguma coisa congelada” significa que talvez possamos adicionar pequenas coisas que “com certeza não afetará nada que já esteja funcionando.”

2.2.6 Binários MySQL compilados pela MySQL AB

Como um serviço, nós na MySQL AB fornecemos um conjunto de distribuições binárias do MySQL que são compiladas no nosso site ou em sites onde os clientes cordialmente nos dão acesso as suas máquinas.

Estas distribuições são geradas com `scripts/make_binary_distribution` e são configuradas com os seguintes compiladores e opções:

SunOS 4.1.4 2 sun4c com gcc 2.7.2.1

```
CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors" ./configure
--prefix=/usr/local/mysql --disable-shared --with-extra-
charsets=complex --enable-asmbl
```

SunOS 5.5.1 (e superior) sun4u com egcs 1.0.3a or 2.90.27 or gcc 2.95.2 and newer

```
CC=gcc CFLAGS="-O3" CXX=gcc CXXFLAGS="-O3 -felide-constructors
-fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql
--with-low-memory --with-extra-charsets=complex --enable-asmbl
```

SunOS 5.6 i86pc com gcc 2.8.1

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
--with-low-memory --with-extra-charsets=complex
```

Linux 2.0.33 i386 com pgcc 2.90.29 (egcs 1.0.3a)

```
CFLAGS="-O3 -mpentium -mstack-align-double" CXX=gcc CXXFLAGS="-O3
-mpentium -mstack-align-double -felide-constructors -fno-
exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql
--enable-asmbl --with-mysqld-ldflags=-all-static --with-extra-
charsets=complex
```

Linux 2.2.x com x686 com gcc 2.95.2

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro
-felide-constructors -fno-exceptions -fno-rtti" ./configure
--prefix=/usr/local/mysql --enable-asmbl --with-mysqld-
ldflags=-all-static --disable-shared --with-extra-charset=complex
```

SCO 3.2v5.0.4 i386 com gcc 2.7-95q4

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex
```

AIX 2 4 com gcc 2.7.2.2

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex
```

OSF1 V4.0 564 alpha com gcc 2.8.1

```
CC=gcc CFLAGS=-O CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
--with-low-memory --with-extra-charsets=complex
```

Irix 6.3 IP32 com gcc 2.8.0

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex
```

BSDI BSD/OS 3.1 i386 com gcc 2.7.2.1

```
CC=gcc CXX=gcc CXXFLAGS=-O ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex
```

BSDI BSD/OS 2.1 i386 com gcc 2.7.2

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex
```

Qualquer que tenha mais opções otimizadas para qualquer das configurações listadas acima pode sempre enviá-los para a lista de mensagens dos desenvolvedores em internals@lists.mysql.com.

Distribuições RPM que anteceda o MySQL versão 3.22 são contribuições dos usuários. Os RPMs gerados por nós da MySQL AB só começaram a ser fornecidos a partir da versão 3.22 do MySQL.

Se você deseja compilar uma versão para depuração do MySQL, você deve adicionar `--with-debug` ou `--with-debug=full` para as linhas de configuração acima e remover qualquer opção `-fomit-frame-pointer`.

2.3 Instalando uma distribuição com fontes do MySQL

Antes de você continuar com as instalações dos fontes, confira antes se nosso binário está disponível para sua plataforma e se ela funcionará para você. Nós colocamos muito esforço para ter certeza que nossos binários são contruídos com as melhores opções possíveis.

Você precisa das seguintes ferramentas para contruir e instalar o MySQL a partir do código fonte:

- GNU `gunzip` para descompactar a distribuição.
- Algum `tar` razoável que desempacote a distribuição. Sabe-se que o GNU `tar` funciona. Você pode ter alguns problemas com o `tar` da Sun.
- Um compilador ANSI C++ funcional. `gcc` \geq 2.95.2, `egcs` \geq 1.0.2 ou `egcs` 2.91.66, SGI C++, e SunPro C++ são alguns dos compiladores que sabemos que funcionam. A `libg++` não é necessária quando o `gcc` for usado. `gcc` 2.7.x tem um bug que torna impossível compilar alguns arquivos C++ perfeitamente corretos, como o `'sql/sql_base.cc'`. Se você possui somente o `gcc` 2.7.x você deve atualiza-lo para conseguir compilar o MySQL. `gcc` 2.8.1 é também conhecido por ter problemas em algumas plataformas portanto ele deve ser evitado se existir um novo compilador para a plataforma.
`gcc` \geq 2.95.2 é recomendado quando compilar o MySQL Versão 3.23.x.
- Um bom programa `make`. GNU `make` é sempre recomendado e é algumas vezes necessário. Se você tiver problemas, recomendamos tentar o GNU `make` 3.75 ou mais novo.

Se você estiver usando uma versão recente de `gcc`, recente o bastante para entender a opção `-fno-exceptions`, é **MUITO IMPORTANTE** que você a use. De outra forma, você pode compilar um binário que quebra randomicamente. Nós também recomendamos que você use `-felide-constructors` e `-fno-rtti` juntas com `-fno-exception`. Se estiver com dúvidas, faça o seguinte:

```
CFLAGS="-O3" CXX=gcc CXXFLAGS="-O3 -felide-constructors -fno-exceptions -fno-rtti"
```

Na maioria dos sistemas você irá obter um binário rápido e estável com essas opções.

Se você tiver problemas, **SEMPRE USE** `mysqlbug` quando postar questões para mysql@lists.mysql.com. Mesmo se o problema não for um bug, `mysqlbug` recolhe informações do sistema que facilitará aos outros resolverem seu problema. Por não usar `mysqlbug`, você perde a vantagem de ter seu problema resolvido! Você irá encontrar `mysqlbug` no diretório ‘`scripts`’ depois de desempacotar a distribuição. See [\[Bug reports\]](#), page [\[undefined\]](#).

2.3.1 Visão geral da instalação rápida

Os comandos básicos que você deve executar para instalar o MySQL a partir da distribuição fonte são:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
shell> gunzip < mysql-VERSION.tar.gz | tar -xvf -
shell> cd mysql-VERSION
shell> ./configure --prefix=/usr/local/mysql
shell> make
shell> make install
shell> scripts/mysql_install_db
shell> chown -R root /usr/local/mysql
shell> chown -R mysql /usr/local/mysql/var
shell> chgrp -R mysql /usr/local/mysql
shell> cp support-files/my-medium.cnf /etc/my.cnf
shell> /usr/local/mysql/bin/safe_mysqld --user=mysql &
```

Se você deseja ter suporte para tabelas InnoDB, você deve editar o arquivo `/etc/my.cnf` e remover o caractere `#` antes dos parâmetros que iniciam com `innodb_...`. See [\[Option files\]](#), page [\[undefined\]](#). See [\[InnoDB start\]](#), page [\[undefined\]](#).

Se você iniciar de um RPM fonte, então faça o seguinte:

```
shell> rpm --rebuild MySQL-VERSION.src.rpm
```

Isto irá criar um RPM binário que você pode instalar.

Você pode adicionar novos usuários utilizando o script `bin/mysql_setpermission` se você instalar o DBI e módulos Perl `Msq1-MySQL-modules`.

Segue uma descrição mais detalhada.

Para instalar uma distribuição fonte, siga os passos a seguir, então prossiga para [\[Post-installation\]](#), page [\[undefined\]](#), para inicialização do pós-instalação e testes:

1. Escolha o diretório sobre o qual você deseja descompactar a distribuição e vá para ele.
2. Obtenha um arquivo de distribuição de algum dos sites listados em [\[Getting MySQL\]](#), page [\[undefined\]](#).
3. Se você está interessado em usar tabelas Berkeley DB com MySQL, você precisará obter uma versão com o patch do código fonte do Berkeley DB. Por favor leia o capítulo sobre tabelas Berkeley DB antes de continuar. See [\[BDB\]](#), page [\[undefined\]](#).

Distribuições fontes do MySQL são fornecidas como arquivos `tar` compactados e tem nomes como ‘`mysql-VERSION.tar.gz`’, onde `VERSION` é um número como 3.23.52.

4. Adicione um usuário e grupo para o `mysql` executar assim:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
```

Estes comandos adicionam o grupo `mysql` e o usuário `mysql`. A sintaxe para `useradd` e `groupadd` podem mudar um pouco em diferentes versões de Unix. Elas podem também ser chamadas `adduser` e `addgroup`. Você pode escolher outros nomes para o usuário e grupo em vez de `mysql`.

5. Descompacte a distribuição para o diretório corrente:

```
shell> gunzip < /path/to/mysql-VERSION.tar.gz | tar xvf -
```

Este comando cria um diretório com o nome `'mysql-VERSION'`.

6. Mude para o diretório da distribuição descompactada:

```
shell> cd mysql-VERSION
```

Note que agora você deve configurar e construir o MySQL a partir deste diretório raiz da distribuição. Você não pode construí-lo em um diretório diferente.

7. Configure o release e compile tudo:

```
shell> ./configure --prefix=/usr/local/mysql
shell> make
```

Quando você executar `configure`, você pode desejar especificar algumas opções. Execute `./configure --help` para uma lista das opções. [\(undefined\) \[configure options\]](#), [page \(undefined\)](#), discute algumas das opções mais usadas.

Se o `configure` falhar, e você for enviar uma mensagem para mysql@lists.mysql.com para pedir ajuda, por favor, inclua qualquer linhas de `'config.log'` que você acha que pode ajudar a resolver o problema. Também inclua as últimas linhas da saída de `configure` se o `configure` abortar. Envie o relatório de erros usando o script `mysqlbug`. See [\(undefined\) \[Bug reports\]](#), [page \(undefined\)](#).

Se a compilação falhar, veja [\(undefined\) \[Compilation problems\]](#), [page \(undefined\)](#), para uma ajuda com um varios problemas comuns.

8. Instalar tudo:

```
shell> make install
```

Você deve executar este comando como `root`.

9. Crie as tabelas de permissões do MySQL (necessárias só se você não tiver instalado o MySQL anteriormente):

```
shell> scripts/mysql_install_db
```

Note que as versões do MySQL anteriores à versão 3.22.10 iniciam o servidor MySQL quando você executa `mysql_install_db`. Isto não acontece mais!

10. Altere o dono dos binários para `root` e do diretório dados para o usuário que irá executar o `mysqld`:

```
shell> chown -R root /usr/local/mysql
shell> chown -R mysql /usr/local/mysql/var
shell> chgrp -R mysql /usr/local/mysql
```

O primeiro comando altera o atributo de propriedade dos arquivos para o usuário `root`, o segundo altera o atributo de propriedade do diretório de dados para o usuário `mysql`, e o terceiro altera o atributo de grupo para o grupo `mysql`.

11. Se você deseja instalar suporte para a interface Perl DBI/DBD, veja [\(undefined\)](#) [Perl support], page [\(undefined\)](#).
12. Se você deseja que o MySQL inicie automaticamente quando você ligar sua máquina, você pode copiar `support-files/mysql.server` para o local onde seu sistema tem seus arquivos de inicialização. Mais informações podem ser encontradas no próprio script `support-files/mysql.server` e em [\(undefined\)](#) [Automatic start], page [\(undefined\)](#).

Depois de tudo ter sido instalado, você deve iniciar e testar sua distribuição:

```
shell> /usr/local/mysql/bin/safe_mysqld --user=mysql &
```

Se o comando falhar imediatamente com `mysqld daemon ended` então você pode achar alguma informação no arquivo `'diretório-dados-mysql/'nome_maquina'.err'`. A razão pode ser que você já possua outro servidor `mysqld` sendo executado. See [\(undefined\)](#) [Multiple servers], page [\(undefined\)](#).

See [\(undefined\)](#) [Post-installation], page [\(undefined\)](#).

2.3.2 Aplicando patches

Algumas vezes patches aparecem na lista de mensagens ou são colocados na [Área de patches](#) do site web do MySQL.

Para aplicar um patch da lista de mensagens, salve a mensagem em que o patch aparece em um arquivo, mude para o diretório raiz da sua distribuição fonte de seu MySQL e execute estes comandos:

```
shell> patch -p1 < patch-file-name
shell> rm config.cache
shell> make clean
```

Patches do site FTP são distribuídos como arquivos texto ou como arquivos compactados com `gzip`. Aplique um patch no formato texto como mostrado acima para patches da lista de mensagens. Para aplicar um patch compactado, mude para o diretório raiz da árvore fonte do MySQL e execute estes comandos:

```
shell> gunzip < patch-file-name.gz | patch -p1
shell> rm config.cache
shell> make clean
```

Depois de aplicar um patch siga as instruções para uma instalação normal a partir dos fontes começando com o passo `./configure`. Depois de executar o passo `make install`, reinicie seu servidor MySQL.

Você pode precisar derrubar algum servidor atualmente em execução antes de executar `make install`. (Use `mysqladmin shutdown` para fazer isto.) Alguns sistemas não lhe permitem instalar uma nova versão do programa se ele substitui algum que estiver em execução.

2.3.3 Opções típicas do configure

O script `configure` fornece uma grande gama de controle sobre como você configura sua distribuição MySQL. Normalmente você faz isto usando opções na linha de comando do `configure`. Você também pode alterar `configure` usando algumas variáveis de ambiente. See [\(undefined\)](#) [Environment variables], page [\(undefined\)](#). Para uma lista de opções suportadas pelo `configure`, execute este comando:

```
shell> ./configure --help
```

Algumas das opções mais usadas normalmente com o `configure` estão descritas a seguir:

- Para compilar apenas as bibliotecas clientes do MySQL e programas clientes e não o servidor, use a opção `--without-server`:

```
shell> ./configure --without-server
```

Se você não possui um compilador C++, `mysql` não irá compilar (ele é o programa cliente que exige C++). Neste caso, você pode remover o código no `configure` que testa pelo compilador C++ e executar `./configure` com a opção `--without-server`. O passo da compilação continuará tentando construir `mysql`, mas você pode ignorar as advertências sobre `'mysql.cc'`. (Se o `make` parar, tente `make -k` para continuar com o resto da compilação mesmo se erros ocorrerem.)

- Se você não deseja que seus arquivos de logs e diretórios de bancos de dados fiquem localizados sobre `'/usr/local/var'`, use o comando `configure`; algo parecido com um destes:

```
shell> ./configure --prefix=/usr/local/mysql
shell> ./configure --prefix=/usr/local \
    --localstatedir=/usr/local/mysql/data
```

O primeiro comando altera o diretório instalação para que tudo seja instalado sobre `'/usr/local/mysql'` em vez do padrão `'/usr/local'`. O segundo comando preserva o diretório da instalação padrão, mas altera a localização padrão para diretórios de bancos de dados (normalmente `'/usr/local/var'`) e altera para `/usr/local/mysql/data`.

- Se você estiver usando Unix e deseja que o arquivo socket do MySQL fique em um diretório diferente do padrão (normalmente no diretório `'/tmp'` ou `'/var/run'`) use o comando `configure` da seguinte forma:

```
shell> ./configure --with-unix-socket-path=/usr/local/mysql/tmp/mysql.sock
```

Perceba que o arquivo fornecido deve ter um caminho absoluto ! Você também pode, mais tarde, alterar a localização de `'mysql.sock'` usando os arquivos de opções do MySQL. See [\[Problems with mysql.sock\]](#), page [\[undefined\]](#)

- Se você deseja compilar programas linkeditados estaticamente (por exemplo, para criar uma distribuição binária, obter mais velocidade, ou evitar problemas com algumas distribuições Red Hat Linux), execute `configure` desta forma:

```
shell> ./configure --with-client-ldflags=-all-static \
    --with-mysqld-ldflags=-all-static
```

- Se você estiver usando `gcc` e não tem `libg++` ou `libstdc++` instalados você pode dizer ao `configure` para usar o `gcc` como seu compilador C++:

```
shell> CC=gcc CXX=gcc ./configure
```

Quando você usar como seu compilador C++, ele não irá ligar com o `libg++` ou `libstdc++`.

Segue algumas configurações de variáveis de ambiente comuns, dependendo do compilador que você estiver usando:

```
gcc 2.7.2.1      CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors"
egcs 1.0.3a     CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors -fno-
exceptions -fno-rtti"
```

```
gcc 2.95.2          CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -
mpentiumpro -felide-constructors -fno-exceptions -fno-rtti"
pgcc 2.90.29 ou    CFLAGS="-O3 -mpentiumpro -mstack-align-double" CXX=gcc
mais novo          CXXFLAGS="-O3 -mpentiumpro -mstack-align-double -felide-
constructors -fno-exceptions -fno-rtti"
```

Na maioria dos casos você pode obter um binário MySQL razoavelmente otimizado usando as opções acima e adicionar as seguintes opções para a linha de configuração:

```
--prefix=/usr/local/mysql --enable- assembler --with-mysqld-ldflags=-all-static
```

A linha completa de configuração deverá ser, em outras palavras, algo como o seguinte para todas as versões recentes do gcc:

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro -felide-constructors
```

Os binários que fornecemos no site Web MySQL em <http://www.mysql.com> são todos compilados com otimização plena e deve ser perfeito para a maioria dos usuários. See [\[MySQL binaries\]](#), page [\[MySQL binaries\]](#). Existem algumas coisas que você pode alterar para criar um binário ainda mais rápido, mas isto é somente para usuários avançados. See [\[Compile and link options\]](#), page [\[Compile and link options\]](#).

Se a construção falhar e produzir erros sobre seu compilador ou linkeditor não estarem aptos para criarem a biblioteca compartilhada 'libmysqlclient.so.r#' ('r#' é um número de versão), você pode evitar este problema fornecendo a opção `--disable-share` para o `configure`. Neste caso, `configure` não construirá uma biblioteca `libmysqlclient.so.*` compartilhada.

- Você pode configurar o MySQL para não usar valores de campos DEFAULT para campos não-NULL (isto é, campos que não podem ser NULL). Com isto, sentenças INSERT geram um erro a menos que você especifique explicitamente valores para todos os campos que necessitem um valor não- NULL. Para suprimir o uso de valores default, execute o `configure` desta forma:

```
shell> CXXFLAGS=-DDONT_USE_DEFAULT_FIELDS ./configure
```

- Por padrão, o MySQL usa o conjunto de caracteres ISO-8859-1 (Latin1). Para alterar o conjunto padrão, use a opção `--with-charset`:

```
shell> ./configure --with-charset=CHARSET
```

CHARSET pode ser um de `big5`, `cp1251`, `cp1257`, `czech`, `danish`, `dec8`, `dos`, `euc_kr`, `gb2312`, `gbk`, `german1`, `hebrew`, `hp8`, `hungarian`, `koi8_ru`, `koi8_ukr`, `latin1`, `latin2`, `sjis`, `swe7`, `tis620`, `ujis`, `usa7`, ou `win1251ukr`. See [\[Character sets\]](#), page [\[Character sets\]](#).

Se você deseja converter os caracteres entre o servidor e o cliente, você deve dar uma olhada no comando `SET OPTION CHARACTER SET`. See [\[SET OPTION\]](#), page [\[SET OPTION\]](#).

Cuidado: Se você alterar o conjunto de caracteres depois de ter criado qualquer tabela, você deve executar `myisamchk -r -q` em cada tabela. Seus índices podem ser ordenados incorretamente. (Isto pode acontecer se você instalar o MySQL, criar algumas tabelas, depois reconfigurar o MySQL para usar um conjunto diferente de caracteres e reinstalá-lo).

Com a opção `--with-extra-charset=LISTA` você pode definir qual conjunto de caracteres adicionais deve ser compilado no servidor.

Aqui LISTA é uma lista de conjuntos de caracteres separados por espaços, complex para incluir todos caracteres que não podem ser carregados dinamicamente ou `all` para incluir todos os conjuntos nos binários.

- Para configurar o MySQL com código para depuração, use a opção `--with-debug`:

```
shell> ./configure --with-debug
```

Isto inclui uma alocação segura de memória que pode encontrar alguns erros e fornecer saída sobre o que está acontecendo. See [\(undefined\) \[Debugging server\]](#), page [\(undefined\)](#).

- Se seus programas clientes usam threads, você precisará também compilar uma versão thread-safe da biblioteca cliente do MySQL com as opções do `configure --enable-thread-safe-client`. Isto irá criar uma biblioteca `libmysqlclient_r` com o qual você deverá ligar suas aplicações que fazem uso de threads. See [\(undefined\) \[Threaded clients\]](#), page [\(undefined\)](#).
- Opções que pertençam a sistemas particulares podem ser encontrados na seção com detalhes específicos de sistemas neste manual. See [\(undefined\) \[Operating System Specific Notes\]](#), page [\(undefined\)](#).

2.3.4 Instalando Através da Árvore Fonte de Desenvolvimento

CUIDADO: Você deve ler esta seção somente se você estiver interessado em nos ajudar a testar nossos novos códigos. Se você só deseja deixar o MySQL funcionando em seus sistema, você deve usar uma distribuição padrão (pode ser uma distribuição binária ou fonte).

Para obter nossa mais nova árvore de desenvolvimento, use estas instruções:

1. Faça download do **BitKeeper** em <http://www.bitmover.com/cgi-bin/download.cgi>. Você precisará do **Bitkeeper** 2.0 ou posterior para acessar nosso repositório.
2. Siga as instruções para instalá-lo.
3. Depois que o **BitKeeper** estiver instalado, use este comando se você deseja clonar o ramo MySQL 3.23:

```
shell> bk clone bk://work.mysql.com:7000 mysql
```

Para clonar o ramo 4.0, use este comando:

```
shell> bk clone bk://work.mysql.com:7001 mysql-4.0
```

O download inicial da árvore fonte pode demorar um pouco, dependendo da velocidade de sua conexão; seja paciente.

4. Você precisará do GNU `autoconf`, `automake`, `libtool` e `m4` para executar o próximo conjunto de comandos. Caso apareçam alguns erros estranhos durante este estágio, confira se você realmente tem a `libtool` instalada!

```
shell> cd mysql
shell> bk -r edit
shell> aclocal; autoheader; autoconf; automake;
shell> ./configure # Adicione suas opções favoritas aqui
shell> make
```

Uma coleção de nossos scripts de configuração padrões está localizada no subdiretório 'BUILD/'. Se preferir, você pode usar 'BUILD/compile-pentium-debug'. Para compilar em uma arquitetura diferente, modifique o script removendo opções que são específicas da arquitetura Pentium.

5. Quando a construção estiver pronta, execute `make install`. Seja cuidadoso com isto em uma máquina de produção; o comando pode sobrescrever sua versão atual instalada. Se você tem outra instalação do MySQL, nós recomendamos que você execute `./configure` com valores diferentes para as opções `prefix`, `tcp-port` e `unix-socket-path` que as usadas pelo seu servidor em produção.
6. Seja rígido com sua nova instalação e tente fazer com que os novos recursos falhem. Inicie executando `make test`. See [\[MySQL test suite\]](#), page [\[MySQL test suite\]](#).
7. Se você chegar ao estágio `make` e a distribuição não compilar, por favor relate-o para bugs@lists.mysql.com. Se você instalou as últimas versões das ferramentas GNU exigidas, e elas falharam tentando processar nossos arquivos de configuração, por favor informe isto também. Entretanto, se você executar `aclocal` e obtêm um erro de `command not found` não o reporte. Tenha certeza que todas as ferramentas necessárias estejam instaladas e que sua variável `PATH` esteja corretamente configurada para que sua shell possa encontrá-la.
8. Depois da operação inicial `bk clone` para obter a árvore fonte, você deve executar `bk pull` periodicamente para obter as atualizações.
9. Você pode examinar o histórico de alterações para a árvore com todos os diffs usando `bk sccstool`. Se você ver alguns diffs estranhos ou código sobre o qual você tenha alguma dúvida, não hesite em enviar um e-mail para internals@lists.mysql.com. Além disso, se você pensar que tem uma idéia melhor em como fazer algo, envie um email para o mesmo endereço com um patch. `bk diffs` irá produzir um patch para você após fazer as alterações no código fonte. Se você não tiver tempo para codificar sua idéia, apenas envie uma descrição.
10. **BitKeeper** tem um ótimo utilitário de ajudar que você pode acessar via `bk helptool`.

2.3.5 Problemas com a compilação

Todos programas MySQL compilam de forma limpa sem alertas no solaris usando `gcc`. Em outros sistemas, alertas podem ocorrer devido a diferenças em arquivos include dos sistemas. Veja [\[MIT-pthreads\]](#), page [\[MIT-pthreads\]](#) para avisos que podem ocorrer usando MIT-pthreads. Para outros problemas, confira a lista abaixo.

A solução para vários problemas envolve reconfiguração. Se você precisa reconfigurar, faça notas do seguinte:

- Se `configure` é executado depois dele já ter sido chamado, ele pode usar informação que foi colhida durante a chamada anterior. Esta informação é armazenada no arquivo `'config.cache'`. Quando `configure` inicia, ele procura por este arquivo, lê seu conteúdo, se ele existir, assumindo que aquela informação continua correta. Essa conjectura é inválida quando você reconfigurar.
- Cada vez que você executa `configure`, você deve executar `make` de novo para recompilar. Entretanto, você pode desejar remover primeiro antigos arquivos objeto de construções anteriores, porque eles foram compilados usando diferentes opções de configuração.

Para prevenir antigas informações de configurações ou arquivos objetos de serem usados, execute estes comandos antes de re-executar `configure`:

```
shell> rm config.cache
shell> make clean
```

Uma alternativa, seria executar `make distclean`

A lista abaixo descreve alguns dos problemas compilando o MySQL que tem sido encontrados com mais frequência:

- Se você obtém erros quando 'sql_yacc.cc' como os mostrados abaixo, você provavelmente tem de falta de memória ou espaço de swap:

```
Internal compiler error: program cc1plus got fatal signal 11
ou
Out of virtual memory
ou
Virtual memory exhausted
```

O problema é que `gcc` necessita de grande quantidade de memória para compilar 'sql_yacc.cc' com funções inline. Tente executando `configure` com a opção `--with-low-memory`:

```
shell> ./configure --with-low-memory
```

Esta opção adiciona `-fno-inline` na a linha de compilação se você estiver usando `gcc` e `-O0` se você estiver usando outro programa. Você deve tentar a opção `--with-low-memory` mesmo se você tiver muita memória e espaço de swap que você ache ser suficiente para não ocorrer erros. Este problema tem ocorrido mesmo em sistemas com boas configurações de hardware e a opção `--with-low-memory` geralmente corrige isto.

- Por padrão, `configure` escolhe `c++` como o nome do compilador e GNU `c++` liga com `-lg++`. Se você estiver usando `gcc`, este comportamento pode causar problemas durante a compilação, como o seguinte:

```
configure: error: installation or configuration problem:
C++ compiler cannot create executables.
```

Você podem também ter problemas durante a compilação relacionados à `g++`, `libg++` ou `libstdc++`.

Uma causa destes problemas é que você pode não ter `g++` ou você pode ter `g++` mas não ter o `libg++` ou o `libstdc++`. De uma olhada no arquivo 'config.log'. Ele deve conter a razão exata do porque seu compilador `c++` não funciona! Para trabalhar evitando estes problemas, você pode usar `gcc` como seu compilador `C++`. Tente configurar a variável de ambiente `CXX` para "gcc -O3". Por exemplo:

```
shell> CXX="gcc -O3" ./configure
```

Isto funciona porque `gcc` compila código fonte `C++` tão bem quanto `g++` faz, mas não faz a ligação em `libg++` ou `libstdc++` por padrão.

Outra forma de corrigir estes problemas, com certeza, é instalando `g++`, `libg++` e `libstdc++`.

- Se sua compilação falhar com erros, como um dos seguintes, você deve atualizar sua versão de `make` para GNU `make`:

```
making all in mit-pthreads
make: Fatal error in reader: Makefile, line 18:
Badly formed macro assignment
```

```

or
make: file 'Makefile' line 18: Must be a separator (:
or
pthread.h: No such file or directory

```

O Solaris e o FreeBSD são conhecidos por terem alguns problemas com o `make`.

O GNU `make` versão 3.75 irá funcionar.

- Se você deseja definir algumas opções que devem ser usadas pelo seu compilador C ou C++, adicione as opções para as variáveis de ambiente `CFLAGS` e `CXXFLAGS`. Você pode também especificar os nomes do compilador a ser usado da mesma forma utilizando `CC` e `CXX`. Exemplo:

```

shell> CC=gcc
shell> CFLAGS=-O3
shell> CXX=gcc
shell> CXXFLAGS=-O3
shell> export CC CFLAGS CXX CXXFLAGS

```

Olhe em [\(undefined\) \[MySQL binaries\]](#), [page \(undefined\)](#) para uma lista de definição de opções que tenham sido úteis em vários sistemas.

- Se você recebeu uma mensagem de erro como esta, é necessário atualizar o compilador `gcc`:

```

client/libmysql.c:273: parse error before '__attribute__'

```

O `gcc` 2.8.1 funciona, mas recomendamos o uso do `gcc` 2.95.2 ou `egcs` 1.0.3a em seu lugar.

- Se você obtém erros como estes vistos abaixo enquanto estiver compilando o `mysqld`, o `configure` não detectou corretamente o tipo do último argumento para `accept()`, `getsockname()` ou `getpeername()`:

```

cxx: Error: mysql.cc, line 645: In this statement, the referenced
      type of the pointer value "&length" is "unsigned long", which
      is not compatible with "int".
new_sock = accept(sock, (struct sockaddr *)&cAddr, &length);

```

Para corrigir isto, edite o arquivo `config.h` (que é gerado pelo `configure`). Procure por estas linhas:

```

/* Define as the base type of the last arg to accept */
#define SOCKET_SIZE_TYPE XXX

```

Altere `XXX` para `size_t` ou `int`, dependendo de seu sistema operacional. (Perceba que você deverá fazer isto cada vez que você executar `configure`, porque `configure` regenera `config.h`.)

- O arquivo `sql_yacc.cc` é gerado pelo `sql_yacc.yy`. Normalmente o processo de construção não necessita criar `sql_yacc.cc`, porque o MySQL já vem com uma cópia pré-gerada. Entretanto, se você necessita recriá-lo você pode encontrar este erro:

```

"sql_yacc.yy", line xxx fatal: default action causes potential...

```

Isto é um indício de que sua versão do `yacc` é deficiente. Provavelmente você precisará instalar o `bison` (a versão GNU de `yacc`) e usá-lo no lugar do `yacc`.

- Se você necessita depurar `mysqld` ou um cliente MySQL, execute `configure` com a opção `--with-debug`, então recompile e ligue seus clientes com a nova biblioteca cliente. See [\(undefined\) \[Debugging client\]](#), [page \(undefined\)](#).

2.3.6 Notas MIT-pthreads

Esta seção descreve alguns dos detalhes envolvidos no uso de MIT-pthreads.

Note que no Linux você NÃO deve usar MIT-pthreads mas instalar LinuxThreads! See [\[Linux\]](#), page [\[undefined\]](#).

Se seu sistema não fornece suporte nativo a thread, você precisará construir o MySQL usando o pacote MIT-pthreads. Isto inclui antigos sistemas FreeBSD, SunOS 4.X, Solaris 2.4 e anteriores entre outros. See [\[Which OS\]](#), page [\[undefined\]](#).

- Na maioria dos sistemas, você pode forçar o uso de MIT-pthreads executando o `configure` com a opção `--with-mit-threads`:

```
shell> ./configure --with-mit-threads
```

Construção em um diretório não fonte não é suportado com o uso de MIT-pthreads, porque nós queremos minimizar nossas alterações para este código.

- As verificações que determinam se MIT-pthreads será usado ou não, ocorrerá somente durante a parte do processo de configuração que trata com o código do servidor. Se você configurou a distribuição usando `--without-server` para construir somente o código cliente, clientes não irão saber se o MIT-pthreads está sendo usado e irá usar conexões socket Unix por padrão. Como os sockets Unix não funcionam sob MIT-pthreads, isto significa que você precisará usar `-h` ou `--host` quando executar programas clientes.
- Quando o MySQL é compilado usando MIT-pthreads, travas de sistema são desabilitadas por padrão por razões de performance. Você pode dizer ao servidor para usar travas de sistema com a opção `--use-locking`.
- Algumas vezes o comando `pthread bind()` falha ao ligar a um socket sem nenhuma mensagem de erro (pelo menos no Solaris). O resultado é que todas conexões ao servidor falham. Por exemplo:

```
shell> mysqladmin version
mysqladmin: connect to server at '' failed;
error: 'Can't connect to mysql server on localhost (146)'
```

A solução para isto é matar o servidor `mysqld` e reiniciá-lo. Isto só aconteceu conosco quando forçamos uma queda do servidor e fizemos uma reinicialização imediata.

- Com MIT-pthreads, a chamada de sistema `sleep()` não é interrompível com `SIGINT` (break). Isto só é percebido quando você executa `mysqladmin --sleep`. Você deve esperar pela chamada `sleep()` para terminar, antes da interrupção ser servida e o processo parar.
- Na ligação, você pode receber mensagens de alerta como estes (pelo menos no Solaris); elas podem ser ignoradas:

```
ld: warning: symbol '_iob' has differing sizes:
(file /my/local/pthreads/lib/libpthread.a(findfp.o) value=0x4;
file /usr/lib/libc.so value=0x140);
/my/local/pthreads/lib/libpthread.a(findfp.o) definition taken
ld: warning: symbol '__iob' has differing sizes:
(file /my/local/pthreads/lib/libpthread.a(findfp.o) value=0x4;
file /usr/lib/libc.so value=0x140);
/my/local/pthreads/lib/libpthread.a(findfp.o) definition taken
```

- Alguns outros alertas também podem ser ignorados:

```
implicit declaration of function 'int strtoll(...)'
```

- Não colocamos `readline` para funcionar com MIT-pthreads. (Isto não é necessário, mas pode ser interessante para alguns.)

2.4 Configurações e Testes Pós-instalação

Uma vez instalado o MySQL (de uma distribuição binária ou fonte), você deve inicializar as tabelas de concessões, iniciar o servidor e ter certeza que o servidor está funcionando bem. Você pode também desejar que o servidor inicie e pare automaticamente quando seu sistema iniciar e desligar.

Normalmente você instala as tabelas de concessões e inicia o servidor assim para instalações baseadas em uma distribuição fonte:

```
shell> ./scripts/mysql_install_db
shell> cd diretorio_instalacao_mysql
shell> ./bin/safe_mysqld --user=mysql &
```

Para uma distribuição binária (sem ser pacotes RPM ou PKG), faça isto:

```
shell> cd diretorio_instalacao_mysql
shell> ./bin/mysql_install_db
shell> ./bin/safe_mysqld --user=mysql &
```

Isto cria o banco de dados `mysql` que irá armazenar todos privilégios do banco de dados, o banco de dados `test` que você poderá usar para testar o MySQL e também entradas de privilégio para o usuário que usa o `mysql_install_db` e o usuário `root` (sem senhas). Isto também inicia o servidor `mysqld`.

`mysql_install_db` não irá sobrescrever nenhuma tabela de privilégios antiga, então deve ser seguro executá-lo em quaisquer circunstâncias. Se você não deseja ter o banco de dados `test` você pode removê-lo com `mysqladmin -u root drop test`.

Testes são geralmente facilmente feitos de um diretório raiz da distribuição MySQL. Para uma distribuição binária, este é seu diretório de instalação (normalmente algo como `/usr/local/mysql`). Para uma distribuição fonte, este é o diretório principal da sua árvore fonte do MySQL.

Nos comandos mostrados abaixo nesta seção e nas seguintes subseções, `BINDIR` é o caminho para a localização na qual os programas como `mysqladmin` e `safe_mysqld` estão instalados. Para uma distribuição binária este é o diretório `'bin'`. Para uma distribuição fonte, `BINDIR` é provavelmente `'/usr/local/bin'`, a menos que você especifique um diretório de instalação diferente de `'/usr/local'` quando você executa `configure`. `EXECDIR` é a localização na qual o servidor `mysqld` está instalado. Para uma distribuição binária, isto é o mesmo que `BINDIR`. Para uma distribuição fonte, `EXECDIR` é provavelmente `'/usr/local/libexec'`.

Testes são descritos em detalhes abaixo:

1. Se necessário, inicie o servidor `mysqld` e configure as tabelas de concessões iniciais contendo os privilégios que determinam como os usuários estão permitidos a conectar ao servidor. Isto é feito normalmente com o script `mysql_install_db`:

```
shell> scripts/mysql_install_db
```

Normalmente, `mysql_install_db` precisa ser executado somente na primeira vez que você instala o MySQL. Portanto, se você estiver atualizando uma instalação existente,

you can skip this step. (however, `mysql_install_db` is really safe to use and it won't update any table that already exists, so if you're not sure what you're doing, you can always run `mysql_install_db`.)

`mysql_install_db` creates six tables (`user`, `db`, `host`, `tables_priv`, `columns_priv` and `func`) in the `mysql` database. A description of the initial privileges is provided in [\(undefined\) \[Default privileges\], page \(undefined\)](#). In short, these privileges allow the `root` user to do anything in MySQL, and allow any user to create or use databases with the name `'test'` or starting with `'test_'`.

If you don't configure the privilege tables, the following error will appear in the log when you start the server:

```
mysqld: Can't find file: 'host.frm'
```

The error above can also occur with a binary distribution of MySQL if you don't start MySQL by running `./bin/safe_mysqld!` See [\(undefined\) \[safe_mysqld\], page \(undefined\)](#).

You must run `mysql_install_db` as `root`. However, if you prefer, you can run the MySQL server as a user (non-`root`) without privileges, as long as the user can read and write files in the database directory. Instructions for running MySQL as a user without privileges are detailed in [\(undefined\) \[Alterando usuários MySQL\], page \(undefined\)](#).

If you have problems with `mysql_install_db`, see [\(undefined\) \[mysql_install_db\], page \(undefined\)](#).

There are some alternatives for running the `mysql_install_db` script as it is provided in the MySQL distribution:

- You can edit `mysql_install_db` before running it, to change the initial privileges that are installed in the privilege tables. This is useful if you want to install MySQL on several machines with the same privileges. In this case, it's likely that you'll only need to add some extra `INSERT` statements to the `mysql.user` and `mysql.db` tables.
- If you want to change the privilege tables after installing them, you can run `mysql_install_db`, then use `mysql -u root mysql` to connect to the privilege tables as `root` and use SQL statements to modify them directly.
- It's possible to recreate the privilege tables completely after they've already been created. You can do this if you've already installed the tables but want to recreate them after editing `mysql_install_db`.

For more information about these alternatives, see [\(undefined\) \[Default privileges\], page \(undefined\)](#).

2. Start the MySQL server like this:

```
shell> cd diretorio_instalacao_mysql
shell> bin/safe_mysqld &
```

If you have problems starting the server, see [\(undefined\) \[Starting server\], page \(undefined\)](#).

3. Use `mysqladmin` para verificar se o servidor está em execução. Os seguintes comandos fornecem um teste simples para conferir se o servidor está em funcionamento e respondendo às conexões:

```
shell> BINDIR/mysqladmin version
shell> BINDIR/mysqladmin variables
```

A saída de `mysqladmin version` varia muito pouco dependendo de sua plataforma e versão do MySQL, mas deve ser similar a esta mostrada abaixo:

```
shell> BINDIR/mysqladmin version
mysqladmin Ver 8.14 Distrib 3.23.32, for linux on i586
Copyright (C) 2000 MySQL AB & MySQL Finland AB & TCX DataKonsult AB
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL license
```

```
Server version          3.23.32-debug
Protocol version        10
Connection              Localhost via Unix socket
TCP port                3306
UNIX socket             /tmp/mysql.sock
Uptime:                 16 sec
```

```
Threads: 1 Questions: 9 Slow queries: 0 Opens: 7 Flush tables: 2 Open table
```

Para ter uma idéia do que você pode fazer com `BINDIR/mysqladmin`, invoque-o com a opção `--help`.

4. Verifique se você pode desligar o servidor:

```
shell> BINDIR/mysqladmin -u root shutdown
```

5. Verifique que você possa reiniciar o servidor. Faça isto usando `safe_mysqld` ou chamado o `mysqld` diretamente. Por exemplo:

```
shell> BINDIR/safe_mysqld --log &
```

Se o `safe_mysqld` falhar, tente executá-lo do diretório de instalação do MySQL (se você já não estiver lá). Se não funcionar, veja [\(undefined\) \[Starting server\], page \(undefined\)](#).

6. Execute alguns testes básicos para verificar se o servidor está funcionando. A saída deve ser similar ao mostrado abaixo:

```
shell> BINDIR/mysqlshow
+-----+
| Databases |
+-----+
| mysql     |
+-----+
```

```
shell> BINDIR/mysqlshow mysql
Database: mysql
+-----+
| Tables |
+-----+
| columns_priv |
```

```

| db          |
| func       |
| host       |
| tables_priv|
| user       |
+-----+

```

```

shell> BINDIR/mysql -e "select host,db,user from db" mysql
+-----+-----+-----+
| host | db      | user |
+-----+-----+-----+
| %    | test   |      |
| %    | test_% |      |
+-----+-----+-----+

```

Também existe uma suite de benchmark no diretório ‘`sql-bench`’ (sob o diretório de instalação do MySQL) que você pode usar para comparar como o MySQL se comporta em diferentes plataformas. O diretório ‘`sql-bench/Results`’ contém os resultados de várias execuções em diferentes bancos de dados e plataformas. Para executar todos testes, execute estes comandos:

```

shell> cd sql-bench
shell> run-all-tests

```

Se você não possui o diretório ‘`sql-bench`’, você provavelmente está usando uma distribuição binária RPM. (Distribuições fontes RPMs incluem o diretório com os benchmarks.) Neste caso, você deve primeiramente instalar a suite de benchmark antes de poder usá-lo. A partir da versão 3.22 do MySQL, começaram a existir arquivos RPMs de benchmark chamados ‘`mysql-bench-VERSION-i386.rpm`’ que contém código ie dados de benchmark.

Se você tem uma distribuição fonte, você também pode executar os testes no subdiretório ‘`tests`’. Por exemplo, para executar ‘`auto_increment.tst`’, faça isto:

```

shell> BINDIR/mysql -vfv test < ./tests/auto_increment.tst

```

Os resultados esperados são mostrados no arquivo ‘`./tests/auto_increment.res`’.

2.4.1 Problemas Executando o `mysql_install_db`

O propósito do script `mysql_install_db` é gerar novas tabelas de privilégios. Ele não irá afeter nenhum outro dado! Ele também não fará nada se você já tem a tabela de privilégio do MySQL instalada.

Se você deseja refazer suas tabelas de privilégios, você deve desligar o servidor `mysqld`, se ele já está executando, então faça assim:

```

mv diretorio-dados-mysql/mysql diretorio-dados-mysql/mysql-old
mysql_install_db

```

Esta seção relaciona alguns problemas que podem ser encontrados ao executar `mysql_install_db`:

`mysql_install_db` não instala as tabelas de permissões

Você pode descobrir que o `mysql_install_db` falha ao instalar as tabelas de permissões e termina depois de mostrar as seguintes mensagens:

```
starting mysqld daemon with databases from XXXXXX
mysql daemon ended
```

Neste caso, você deve examinar o arquivo de log com muito cuidado! O log deve se encontrar no diretório 'XXXXXX' nomeado pela mensagem de erro, e deve indicar porque `mysqld` não inicializa. Se você não entende o que aconteceu, inclua o log quando você postar um relato de erro usando `mysqlbug`! See [\(undefined\)](#) [Bug reports], page [\(undefined\)](#).

Já existe um daemon `mysqld` sendo executado

Neste caso, provavelmente não será necessário executar o `mysql_install_db`. Você deve executar o `mysql_install_db` somente uma vez, quando você instalar o MySQL da primeira vez.

Instalar um segundo daemon `mysqld` não funciona quando um daemon

estiver em execução.

Isto pode acontecer quando você já tiver uma instalação do MySQL existente, mas deseja colocar uma nova instalação em um diferente lugar (por exemplo, para testes, ou talvez você simplesmente deseja executar duas instalações ao mesmo tempo). Geralmente o problema que ocorre quando você tenta executar o segundo servidor é que ele tenta usar o mesmo socket e porta que o outro. Neste caso você irá obter a mensagem de erro: `Can't start server: Bind on TCP/IP port: Address already in use` ou `Can't start server : Bind on unix socket....` See [\(undefined\)](#) [Installing many servers], page [\(undefined\)](#).

Você não tem direito de escrita no diretório '/tmp'

Se você não tem direito de escrita para criar um arquivo socket no local padrão (em '/tmp') ou permissão para criar arquivos temporários em '/tmp,' você irá obter um erro quando executar `mysql_install_db` ou quando iniciar ou usar `mysqld`.

Você pode especificar socket e diretório temporário diferentes, como segue:

```
shell> TMPDIR=/algum_dir_tmp/
shell> MYSQL_UNIX_PORT=/algum_dir_tmp/mysqld.sock
shell> export TMPDIR MYSQL_UNIX_PORT
```

See [\(undefined\)](#) [Problems with `mysql.sock`], page [\(undefined\)](#).

'algum_dir_tmp' deve ser o caminho para o mesmo diretório no qual você tem permissão de escrita. See [\(undefined\)](#) [Environment variables], page [\(undefined\)](#).

Depois disto você deve estar apto para executar `mysql_install_db` e iniciar o servidor com estes comandos:

```
shell> scripts/mysql_install_db
shell> BINDIR/safe_mysqld &
```

`mysqld` falha imediatamente

Se você estiver executando RedHat Versão 5.0 com uma versão de `glibc` anterior a 2.0.7-5 você deve ter certeza que você instalou todos os patches para a `glibc`! Existe muita informação sobre isto nos arquivos das listas de mensagens do MySQL. Links para os arquivos de correio estão disponíveis online em

<http://www.mysql.com/documentation/>. Veja também [\(undefined\)](#) [Linux], page [\(undefined\)](#).

Você pode também iniciar o `mysqld` manualmente usando a opção `--skip-grant-tables` e adicionar a informação de privilégios usando o `mysql`:

```
shell> BINDIR/safe_mysqld --skip-grant-tables &
shell> BINDIR/mysql -u root mysql
```

Do `mysql`, execute manualmente os comandos SQL em `mysql_install_db`. Tenha certeza de executar `mysqladmin flush_privileges` ou `mysqladmin reload` após dizer ao servidor para recarregar as tabelas de permissões.

2.4.2 Problemas inicializando o Servidor MySQL

Se você for usar tabelas que suportem transações (BDB, InnoDB), primeiro deve-se criar um arquivo `my.cnf` e configurar opções de inicialização para os tipos de tabelas que você planeja usar. See [\(undefined\)](#) [Table types], page [\(undefined\)](#).

Geralmente, você inicia o servidor `mysqld` de uma das três maneiras:

- Invocando `mysql.server`. Este script é usado primariamente na inicialização e finalização do sistema, e é descrito de forma mais completa em [\(undefined\)](#) [Automatic start], page [\(undefined\)](#).
- Invocando `safe_mysqld`, que tenta determinar as opções apropriadas para `mysqld` e então executá-lo com estas opções. See [\(undefined\)](#) [safe_mysqld], page [\(undefined\)](#).
- No Windows NT você deve instalar o `mysqld` como um serviço, como abaixo:

```
bin\mysqld-nt --install # Instala o MySQL como um serviço
```

Você pode iniciar/parar o `mysqld` como abaixo:

```
NET START mysql
NET STOP mysql
```

Perceba que neste caso você não pode usar outras opções para o `mysqld`!

Você pode remover o serviço da seguinte maneira:

```
bin\mysqld-nt --remove # remove MySQL as a service
```

- Invocando o `mysqld` diretamente.

Quando o daemon `mysqld` inicia, ele altera o diretório para o diretório de dados. É neste diretório que ele espera gravar arquivos de log e o arquivo pid (com o ID do processo) e onde ele espera encontrar os bancos de dados.

A localização do diretório de dados é especificada quando a distribuição é compilada. Entretanto, se o `mysqld` espera encontrar o diretório de dados em lugar diferente de onde ele realmente está no seu sistema, ele não funcionará corretamente. Se você tiver problemas com caminhos incorretos você pode encontrar quais opções o `mysqld` permite e quais são as configurações do caminho padrão chamando o `mysqld` com a opção `--help`. Você pode sobrescrever os padrões especificando os caminhos corretos como argumentos de linha de comando ao `mysqld`. (Estas opções também podem ser usadas com o `safe_mysqld`).

Normalmente você precisaria indicar ao `mysqld` somente o diretório base sob o qual o MySQL é instalado. Você pode fazer isso usando a opção `--basedir`. Você pode também usar `--help` para conferir o efeito das opções para se alterar o caminho (perceba que `--help` deve ser a opção final do comando `mysqld`. Por exemplo:

```
shell> EXECDIR/mysqld --basedir=/usr/local --help
```

Uma vez que você determina as configurações de caminho que você deseja, inicie o servidor sem a opção `--help`.

Qualquer que tenha sido o método utilizado para iniciar o servidor, se houver falha na inicialização, confira o arquivo de log para ver se você pode entender o porquê. Arquivos log estão localizados no diretório dados (normalmente `/usr/local/mysql/data` para uma distribuição binária, `/usr/local/var` para uma distribuição fonte, `\mysql\data\mysql.err` no Windows.) Procure no diretório de dados por arquivos com nomes no formato `'nome_maquina.err'` e `'nome_maquina.log'` onde `nome_maquina` é o nome do servidor. Então confira as últimas linhas destes arquivos:

```
shell> tail nome_maquina.err
shell> tail nome_maquina.log
```

Se você encontrar algo como o seguinte no arquivo log:

```
000729 14:50:10 bdb: Recovery function for LSN 1 27595 failed
000729 14:50:10 bdb: warning: ./test/t1.db: No such file or directory
000729 14:50:10 Can't init databases
```

Significa que você não inicializou o `mysqld` com `--bdb-no-recover` e o Berkeley DB encontrou algo errado com seus arquivos log quando ele tentou recuperar seus bancos de dados. Para poder continuar, você deve mover o antigo arquivo log Berkeley DB do diretório do banco de dados para outro lugar, onde poderá examiná-los posteriormente. Os arquivos log são nomeados `'log.0000000001'`, onde o número irá incrementar com o tempo.

Se você estiver executando o `mysqld` com suporte a tabelas BDB e o `mysqld` falhar no início, pode ser devido a alguns problemas com o arquivo de recuperação BDB. Neste caso você pode tentar iniciar o `mysqld` com `--bdb-no-recover`. Se isto ajudar, então você pode remover todos os arquivos `'log.*'` do diretório de dados e tentar iniciar o `mysqld` novamente.

Se você obter o seguinte erro, significa que algum outro programa (ou outro servidor `mysqld`) já está usando a porta TCP/IP ou socket `mysqld` está tentando usar:

```
Can't start server: Bind on TCP/IP port: Address already in use
ou
Can't start server : Bind on unix socket...
```

Use `ps` para ter certeza que você não tem outro servidor `mysqld` em execução. Se você não consegue encontrar outro servidor, você pode tentar executar o comando `telnet sua_maquina numero_porta_tcp-ip` e apertar ENTER várias vezes. Se você não obter uma mensagem como `telnet: Unable to connect to remote host: Connection refused`, algo está usando a mesma porta TCP/IP que o `mysqld` está tentando usar. Veja [\[mysql-install.db\]](#), [page <undefined>](#) e [\[Multiple servers\]](#), [page <undefined>](#).

Se o `mysqld` está atualmente em execução, você pode verificar as configurações que ele está usando executando este comando:

```
shell> mysqladmin variables
```

ou

```
shell> mysqladmin -h 'your-host-name' variables
```

Se o `safe_mysqld` inicia o servidor mas você não consegue se conectar a ele, tenha certeza que você tem uma entrada no arquivo `/etc/hosts` que parece com isto:

127.0.0.1 localhost

Este problema só ocorre em sistemas que não possuem uma biblioteca thread funcional e para o qual o MySQL deve estar configurado para usar MIT-pthreads.

Se você não consegue iniciar o `mysqld` você pode tentar criar um arquivo para rastreamento de erros (trace) para encontrar o problema. See [\[Making trace files\]](#), page [\[undefined\]](#).

Se você estiver utilizando tabelas InnoDB, procure pelas opções específicas de inicialização do InnoDB. See [\[InnoDB start\]](#), page [\[undefined\]](#).

Se você estiver usando tabelas BDB (Berkeley DB), você deve se familiarizar com as diferentes opções específicas de inicialização do BDB. [\[BDB start\]](#), page [\[undefined\]](#).

2.4.3 Inicializando e parando o MySQL automaticamente.

Os scripts `mysql.server` e `safe_mysqld` podem ser usados para iniciar o servidor automaticamente na inicialização do sistema. `mysql.server` também pode ser usado para parar o servidor.

O script `mysql.server` pode ser usado para inicializar ou parar o servidor utilizando-o com os argumentos `start` ou `stop`:

```
shell> mysql.server start
shell> mysql.server stop
```

`mysql.server` pode ser encontrado no diretório `'share/mysql'` sob o diretório de instalação do MySQL ou no diretório `'support-files'` da árvore fonte do MySQL.

Antes do `mysql.server` iniciar o servidor, ele vai para o diretório de instalação do MySQL, e então chama o `safe_mysqld`. Você pode precisar editar o `mysql.server` se tiver uma distribuição binária instalada em um local não-padrão. Modifique-o para chamar o diretório (`cd`) apropriado antes de executar o `safe_mysqld`. Se você deseja que o servidor seja executado com um usuário específico, adicione uma linha `user` apropriada para o arquivo `'/etc/my.cnf'`, como será visto posteriormente nesta seção.

`mysql.server stop` desliga o servidor MySQL enviando um sinal para ele. Você pode desligar o servidor manualmente executando `mysqldadmin shutdown`.

Você pode querer adicionar esses comandos `start` e `stop` nos lugares apropriados de seus arquivos `'/etc/rc.*'` quando você começar a usar o MySQL para aplicações de produção. Perceba que se você modifica `mysql.server`, e atualizar seu MySQL, sua versão modificada será sobrescrita, portanto você deve criar uma cópia de sua versão editada.

Se o seu sistema usa `'/etc/rc.local'` para iniciar scripts externos, você deve adicionar o seguinte a ele:

```
/bin/sh -c 'cd /usr/local/mysql ; ./bin/safe_mysqld --user=mysql &'
```

Você também pode adicionar opções para `mysql.server` em um arquivo global `'/etc/my.cnf'`. Um típico arquivo `'/etc/my.cnf'` pode parecer com isto:

```
[mysqld]
datadir=/usr/local/mysql/var
socket=/var/tmp/mysql.sock
port=3306
user=mysql
```

```
[mysql.server]
basedir=/usr/local/mysql
```

O script `mysql.server` entende as seguintes opções: `datadir`, `basedir` e `pid-file`.

A seguinte tabela mostra quais grupos de opções cada script de inicialização lê dos arquivos de opções:

Script	Grupos de opções
<code>mysqld</code>	<code>mysqld</code> e <code>server</code>
<code>mysql.server</code>	<code>mysql.server</code> , <code>mysqld</code> e <code>server</code>
<code>safe_mysqld</code>	<code>mysql.server</code> , <code>mysqld</code> e <code>server</code>

See [\[Option files\]](#), page [\[undefined\]](#).

2.5 Atualizando/Desatualizando o MySQL

Você sempre pode mover os arquivos de formato e de dados do MySQL entre diferentes versões na mesma arquitetura enquanto você tiver versão base do MySQL. A versão base atual é 3. Se você alterar o conjunto de caracteres quando executar o MySQL (o que também pode alterar a ordem de classificação), você deve executar `myisamchk -r -q` em todas tabelas. De outra forma seus índices podem não ser corretamente ordenados.

Se você tem receio de novas versões, você sempre pode renomear seu antigo `mysqld` para algo como `mysqld-número-da-versão-antiga`. Se o seu novo `mysqld` comportar de maneira inesperada, você simplesmente pode desliga-lo e reiniciar com seu antigo `mysqld`!

Quando fizer uma atualização, faça backup de seus bancos de dados antigos.

Se depois de uma atualização, você tiver problemas com programas clientes recompilados como `Commands out of sync` ou “core dumps” inesperados, você provavelmente usou um arquivo de cabeçalho ou de biblioteca antigo na compilação de seus programas. Neste caso você deve conferir a data de seu arquivo `mysql.h` e da biblioteca `libmysqlclient.a` para verificar que eles são da nova distribuição MySQL. Se não, por favor, recompile seus programas!

Se você tiver problemas com a inicialização do novo servidor `mysqld` ou caso você não consiga conectar sem uma senha, confira se o seu arquivo `my.cnf` é o mesmo da antiga instalação! Você pode conferir com isto: `nome-programa --print-defaults`. Se isto não produzir outra saída além do nome do programa, você tem um arquivo `my.cnf` ativo que está afetando o funcionamento do programa!

É uma boa idéia reconstruir e reinstalar a distribuição `Msql-MySQL-modules` sempre que instalar uma nova versão do MySQL, particularmente se você perceber sintomas tais como todos os scripts DBI falharem depois de atualizar o MySQL.

2.5.1 Atualizando da versão 3.22 para 3.23

A Versão 3.23 do MySQL suporta tabelas do novo tipo `MyISAM` e do antigo tipo `ISAM`. Você não necessita converter suas antigas tabelas para usá-las com a versão 3.23. Por padrão, todas novas tabelas serão criadas usando o tipo `MyISAM` (a menos que você inicie o `mysqld` com a opção `--default-table-type=isam`). Você pode alterar uma tabela `ISAM` para uma

tabela MyISAM com `ALTER TABLE nome_tabela TYPE=MyISAM` ou com o script Perl `mysql_convert_table_format`.

Os clientes versões 3.22 e 3.21 irão trabalhar sem quaisquer problemas com um servidor versão 3.23.

As seguintes listas dizem o que você deve conferir quando atualizar para a versão 3.23:

- Todas tabelas que usam o conjunto de caracteres `tis620` devem ser corrigidos com `myisamchk -r` ou `REPAIR TABLE`.
- Se você fizer um `DROP DATABASE` em um banco de dados ligado simbolicamente, a ligação e o banco de dados original serão apagados. (Isto não acontece na 3.22 porque ele não detecta a chamada de sistema `readlink`).
- `OPTIMIZE TABLE` agora funciona somente para tabelas **MyISAM**. Para outros tipos de tabelas, você pode usar `ALTER TABLE` para otimizar a tabela. Durante o `OPTIMIZE TABLE` a tabela é, agora, bloqueada para outras threads.
- O cliente MySQL `mysql` é, agora, inicializado por padrão com a opção `--no-named-commands (-g)`. Esta opção pode ser desabilitada com `--enable-named-commands (-G)`. Isto pode causar problemas de incompatibilidade em alguns casos, por exemplo, em scripts SQL que usam comandos sem ponto e vírgula! Comandos longos continuam funcionando.
- Se você estiver usando a ordem de classificação de caracteres `alemã`, você deve reparar todas suas tabelas com `isamchk -r`, porque foram feitas alterações na sua ordem de classificação!
- O tipo padrão de retorno de `IF` irá agora depender de ambos argumentos e não apenas do primeiro argumento.
- `AUTO_INCREMENT` não irá funcionar com números negativos. A razão para isto é que números negativos causaram problemas quando o `-1` passa para `0`. `AUTO_INCREMENT` é, agora, tratado em um nível mais baixo para tabelas MyISAM e é muito mais rápido que antes. Para tabelas MyISAM números antigos também não são mais reusados, mesmo se você apagar algumas linhas da tabela.
- `CASE`, `DELAYED`, `ELSE`, `END`, `FULLTEXT`, `INNER`, `RIGHT`, `THEN` e `WHEN` agora são palavras reservadas.
- `FLOAT(X)` agora é um tipo de ponto flutuante verdadeiro e não um valor com um número fixo de decimais.
- Quando estiver declarando `DECIMAL(tamanho,dec)` o argumento `tamanho` não inclui mais um lugar para o símbolo do ponto decimal.
- Uma string `TIME` agora deve estar em um dos seguintes formatos: `[[[DAYS] [H]H:]MM:]SS[.fraction]` ou `[[[[[H]H]H]H]MM]SS[.fraction]`
- `LIKE` agora compara strings usando as mesmas regras de comparação de caracteres de `'='`. Se você precisa do antigo comportamento, você pode compilar o MySQL com a opção `CXXFLGAS=-DLIKE_CMP_TOUPPER`.
- `REGEXP` agora é caso insensitivo para strings normais (não binárias).
- Quando for necessário dar manutenção/reparar tabelas deve ser usado `CHECK TABLE` ou `myisamchk` para tabelas MyISAM (`.MYI`) e `isamchk` para tabelas ISAM (`.ISM`).
- Se desejar que os arquivos `mysqldump` sejam compatíveis entre as versões 3.22 e 3.23 do MySQL, não deve ser usados as opções `--opt` ou `--full` com o `mysqldump`.

- Confira todas suas chamadas à `DATE_FORMAT()` para ter certeza que exista um `'%'` antes de cada caractere formatador. (Versões mais antigas que o MySQL 3.22 aceitaivam esta sintaxe.)
- `mysql_fetch_fields_direct` agora é uma função (era uma macro) e ela retorna um ponteiro para um `MYSQL_FIELD` no lugar de um `MYSQL_FIELD`.
- `mysql_num_fields()` não pode mais ser usada em um objeto `MYSQL*` (agora é uma função que obtém `MYSQL_RES*` como um argumento. Agora deve ser usado `mysql_field_count()`).
- No MySQL Versão 3.22, a saída de `SELECT DISTINCT ...` era na maioria das vezes ordenada. Na Versão 3.23, você deve usar `GROUP BY` ou `ORDER BY` para obter a saída ordenada.
- `SUM()` agora retorna `NULL`, em vez de 0 se não existir registros coincidentes. Isto é de acordo com o ANSI SQL.
- Um `AND` ou `OR` com valores `NULL` agora retornam `NULL` no lugar de 0. Isto afetará, em grande parte, pesquisas que usam `NOT` em uma expressão `AND/OR` como `NOT NULL = NULL`. `LPAD()` e `RPAD` reduzirão a string resultante se ela for maior que o tamanho do argumento.

2.5.2 Atualizando da versão 3.21 para 3.22

Nada que afetaria a compatibilidade foi alterada entre a versão 3.21 e 3.22. A única dificuldade é que novas tabelas que são criadas com colunas do tipo `DATE` usarão a nova forma de armazenar a data. Você não pode acessar esses novos campos com uma versão antiga de `mysqld`.

Depois de instalar o MySQL versão 3.22, você deve iniciar o novo servidor e depois executar o script `mysql_fix_privilege_tables`. Isto adicionará os novos privilégios que você precisará para usar o comando `GRANT`. Se você se esquecer disto, sera retornado o erro `Access denied` quando você tentar usar `ALTER TABLE`, `CREATE INDEX` ou `DROP INDEX`. Se o seu usuário `root` do MySQL necessita uma senha, você deve fornecê-la como um argumento para o `mysql_fix_privilege_tables`.

A interface API C para `mysql_real_connect()` foi alterada. Se você tem um programa cliente antigo que chama essa função, você deve colocar um 0 para o novo argumento `db` (ou recodificar o cliente para enviar o elemento `db` para conexões mais rápidas). Você também deve chamar `mysql_init()` antes de chamar `mysql_real_connect()`! Esta alteração foi feita para permitir à nova função `mysql_options()` salvar opções na estrutura do manipulador do `MYSQL`.

A variável `key_buffer` do `mysqld` mudou de nome para `key_buffer_size`, mas você ainda pode usar o antigo nome nos seus arquivos de inicialização.

2.5.3 Atualizando da versão 3.20 para 3.21

Se você estiver executando uma versão mais antiga que a Versão 3.20.28 e deseja mudar para a versão 3.21 você deve fazer o seguinte:

Inicie o servidor `mysqld` versão 3.21 com `safe_mysqld --old-protocol` para usá-lo com clientes de uma distribuição da versão 3.20 Neste caso, a nova função cliente `mysql_errno()`

não irá retornar erro do servidor, somente `CR_UNKNOWN_ERROR` (mas isto funciona para erros de clientes) e o servidor usa a forma antiga da função `password()` para verificação, ao invés da nova.

Se você **NÃO** estiver usando a opção `--old-protocol` para `mysqld`, você precisará fazer as seguir alterações:

- Todo o código cliente deve ser recompilado. Se você usa o ODBC, deve obter o novo driver **MyODBC 2.x**.
- O script `scripts/add_long_password` deve ser executado para converter o campo `Password` na tabela `mysql.user` para `CHAR(16)`.
- Todas as senhas devem ser reatribuídas na tabela `mysql.user` (para obter 62-bits no lugar de senhas 31-bits).
- O formato das tabelas não foi alterado, então não é preciso converter nenhuma tabela.

A versão do MySQL 3.20.28 e superiores podem manipular o novo formato da tabela de `usuários` sem afetar os clientes. Se você tem uma versão do MySQL mais nova que 3.20.28, senhas não irão mais funcionar se você converter a tabela de `usuários`. Por segurança, você primeiro deve fazer uma atualização para a versão 3.20.28, pelo menos, e então atualizar para a versão 3.21.

O novo código cliente trabalha com um servidor `mysqld 3.20.x`, portanto se houver problemas com 3.21.x você deve usar o antigo servidor 3.20.x sem a necessidade de recompilar os clientes novamente.

Se você não está usando a opção `--old-protocol` para o `mysqld`, antigos clientes irão emitir a seguinte mensagem de erro:

```
ERROR: Protocol mismatch. Server Version = 10 Client Version = 9
```

A nova interface PERL DBI/DBD também suporta a antiga interface `mysqlperl`. A única alteração que deve ser feita se você usa o `mysqlperl` é alterar os argumentos para a função `connect()`. Os novos argumentos são: `host`, `database`, `user`, `password` (os argumentos `user` e `password` foram alterados de lugar). See [\[Perl DBI Class\]](#), page [\[undefined\]](#).

As seguintes alterações podem afetar consultas em antigas aplicações:

- `HAVING` deve ser especificada antes de qualquer cláusula `ORDER BY`.
- Os parâmetros para `LOCATE()` foram trocados.
- Agora existem algumas palavras reservadas novas. As mais notáveis são `DATE TIME` e `TIMESTAMP`.

2.5.4 Atualizando para outra arquitetura

Se você estiver usando o MySQL Versão 3.23, você pode copiar os arquivos `.frm`, `.MYI` e `.MYD` entre diferentes arquiteturas que suportem o mesmo formato de ponto flutuante. (O MySQL cuida de cada detalhe de troca de bytes.)

Os arquivos `ISAM` de dados e índices (`*.ISD` e `*.ISM` respectivamente) são dependentes da arquitetura e em alguns casos dependentes do Sistema Operacional. Se você deseja mover suas aplicações para outra máquina que tem uma arquitetura ou SO diferentes da sua máquina atual, você não deve tentar mover um banco de dados simplesmente copiando os arquivos para a outra máquina. Use o `mysqldump`.

Por padrão, o `mysqldump` irá criar um arquivo cheio de declarações SQL. Você pode então transferir o arquivo para a outra máquina e alimentá-la como uma entrada para o cliente `mysql`.

Utilize `mysqldump --help` para ver quais opções estão disponíveis. Se você está movendo os dados para uma versão mais nova do MySQL, você deve usar `mysqldump --opt` com a nova versão para obter uma descarga rápida e compacta.

A mais fácil (mas não a mais rápida) forma para mover um banco de dados entre duas máquinas é executar os seguintes comandos na máquina em que o banco de dados se encontra:

```
shell> mysqladmin -h 'nome da outra maquina' create nome_bd
shell> mysqldump --opt nome_bd \
    | mysql -h 'nome da outra maquina' nome_bd
```

Se você deseja copiar um banco de dados de um máquina remota sobre uma rede lenta, pode ser usado:

```
shell> mysqladmin create nome_bd
shell> mysqldump -h 'nome de outra maquina' --opt --compress nome_bd \
    | mysql nome_bd
```

O resultado pode também ser armazenado em um arquivo, depois transfira o arquivo para a máquina destino e carregue o arquivo no banco de dados. Por exemplo você pode descarregar um banco de dados para um arquivo na máquina origem desta forma:

```
shell> mysqldump --quick nome_bd | gzip > nome_bd.contents.gz
```

(O arquivo criado neste exemplo está compactado.) Transfira o arquivo contendo o conteúdo do banco de dados para a máquina destino e execute estes comandos:

```
shell> mysqladmin create nome_bd
shell> gunzip < nome_bd.contents.gz | mysql nome_bd
```

Também pode ser usado `mysqldump` e `mysqlimport` para ajudar na transferência do banco de dados. Para grandes tabelas, isto é muito mais rápido do que usar simplesmente `mysqldump`. Nos comandos abaixo, `DUMPDIR` representa o caminho completo do diretório que você utiliza para armazenar a saída de `mysqldump`.

Primeiro, crie o diretório para os arquivos de saída e descarregue o banco de dados:

```
shell> mkdir DUMPDIR
shell> mysqldump --tab=DUMPDIR nome_bd
```

Depois transfira os arquivos no diretório `DUMPDIR` para algum diretório correspondente na máquina destino e carregue os arquivos no MySQL assim:

```
shell> mysqladmin create nome_bd           # cria o banco de dados
shell> cat DUMPDIR/*.sql | mysql nome_bd   # cria tabelas no banco de dados
shell> mysqlimport nome_bd DUMPDIR/*.txt   # carrega dados nas tabelas
```

Não se esqueça de copiar o banco de dados `mysql` também, porque é nele que as tabelas de permissões (`user`, `db` e `host`) são armazenadas. Você pode ter que executar comandos como o usuário `root` do MySQL na nova máquina até que você tenha o banco de dados `mysql` no lugar.

Depois de importar o banco de dados `mysql` para a nova máquina, execute `mysqladmin flush-privileges` para que o servidor recarregue as informações das tabelas de permissões.

2.6 Notas específicas para os Sistemas Operacionais

2.6.1 Notas Linux (Todas as versões)

As notas abaixo a respeito da **glibc** aplicam-se somente na situação quando o MySQL é construído por você mesmo. Se você está executando Linux em uma máquina x86, na maioria dos casos é muito melhor para você usar nosso binário. Nós ligamos nossos binários com a melhor versão alterada da **glibc**, podemos escolher as melhores opções do compilador, em uma tentativa de torná-la funcional para um servidor muito exigido. Portanto se você ler o texto abaixo, e está em dúvida sobre o que deve fazer, tente usar o nosso binário primeiro para ver se ele preenche suas necessidades, e preocupe-se com uma construção própria apenas se você descobrir que nosso binário não é bom o suficiente para você. Neste caso, iríamos apreciar se fosse feita uma observação sobre isto, para que possamos fazer uma melhor versão binária da próxima vez. Para um usuário comum, mesmo para configurações com várias conexões concorrentes e/ou tabelas excedendo o limite de 2 GB, nosso binário é, na maioria das vezes, a melhor escolha.

O MySQL usa LinuxThreads no Linux. Se você usa uma versão do Linux que não tenha a **glibc2**, você deve instalar LinuxThreads antes de tentar compilar o MySQL. Você pode obter o LinuxThreads em <http://www.mysql.com/Downloads/Linux>.

NOTA: Temos visto alguns problemas estranhos com o Linux 2.2.14 e MySQL em sistemas SMP; Se você tem um sistema SMP, recomendamos a atualização para o Linux 2.4! Seu sistema ficará mais rápido e mais estável fazendo isto!

Perceba que as versões da **glibc** iguais ou anteriores à Versão 2.1.1 tem um bug fatal no tratamento do `pthread_mutex_timedwait`, que é usado quando for feito um `INSERT DELAYED`. Recomendamos não usar `INSERT DELAYED` antes de atualizar a **glibc**.

Se você planeja ter mais de 1000 conexões simultâneas, será necessário fazer algumas alterações na LinuxThreads, recompila-a e religue o MySQL ao novo `'libpthread.a'`. Aumente `PTHREAD_THREADS_MAX` em `'sysdeps/unix/sysv/linux/bits/local_lim.h'` para 4096 e abaixe o `STACK_SIZE` no `'linuxthreads/internals.h'` para 256KB. Os caminhos são relativos à raiz da **glibc**. Note que o MySQL não será estável com cerca de 600-1000 conexões se o valor de `STACK_SIZE` for o padrão de 2MB.

Se você tiver um problema com o MySQL, no qual ele não consiga abrir vários arquivos ou conexões, pode ser que você não tenha configurado o Linux para lidar com o número de arquivos suficiente.

No Linux 2.2 e posteriores, você pode conferir o valor para a alocação dos arquivos fazendo:

```
cat /proc/sys/fs/file-max
cat /proc/sys/fs/dquot-max
cat /proc/sys/fs/super-max
```

Se você possui mais de 16M de memória, deve ser adicionado o seguinte no seu script de boot (`'/etc/rc/boot.local'` no SuSE):

```
echo 65536 > /proc/sys/fs/file-max
echo 8192 > /proc/sys/fs/dquot-max
echo 1024 > /proc/sys/fs/super-max
```

Você também pode executar os comandos acima da linha de comando como root, mas neste caso, os antigos limites voltarão a ser usados na próxima vez que o computador for reiniciado.

Deve também ser adicionado ao `/etc/my.cnf`:

```
[safe_mysqld]
open-files-limit=8192
```

Os parâmetros acima permitem o MySQL criar até 8192 conexões + arquivos.

A constante `STACK_SIZE` na `LinuxThreads` controla o espaçamento das pilhas threads no espaço de endereçamento. Ela necessita ser grande o bastante para que tenha espaço o suficiente para a pilha de cada thread, mas pequena o bastante para manter a pilha de alguma thread executando dos dados globais `mysqld`. Infelizmente, a implementação Linux de `mmap()`, como descobrimos em experiências, irá desmapear uma região já mapeada se você solicitar o mapeamento de um endereço já em uso, zerando os dados de toda a página ao invés de retornar. um erro. Portanto a segurança do `mysqld` ou qualquer outra aplicação baseada em threads depende do comportamento gentil do código que cria as threads. O usuário deve tomar medidas para certificar-se que o número de threads em funcionamento em qualquer hora seja suficientemente baixo para que as pilhas das threads permaneçam longe do monte global. Com `mysqld` você deve reforçar este comportamento "gentil" configurando um valor razoável para a variável `max_connections`.

Se você mesmo construiu o MySQL e não deseja confusões corrigindo `LinuxThreads`, você deve configurar `max_connections` para um valor máximo de 500. Ele ainda deve ser menor se você tiver uma chave grande para o buffer, grandes tabelas heap, ou outras coisas que fazem o `mysqld` alocar muita memória ou se você estiver executando um kernel 2.2 com o patch de 2GB. Se você estiver usando nosso binário ou RPM versão 3.23.25 ou posterior, você pode seguramente configurar `max_connections` para 1500, assumindo que não há uma grande chave de buffer ou tabelas heap com grande quantidade de dados. Quanto mais você reduz `STACK_SIZE` em `LinuxThreads` mais threads você pode criar seguramente. Recomendamos os valores entre 128K e 256K.

Se você usa várias conexões simultâneas, você pode sofrer com um "recurso" do kernel 2.2 que penaliza um processo por bifurcar-se ou clonar um filho na tentativa de prevenir um ataque de separação. Isto faz com que o MySQL não consiga fazer uma bom escalonamento, quando o número de clientes simultâneos cresce. Em sistemas com CPU única, temos visto isto se manifestar em uma criação muito lenta das threads, tornando a conexão ao MySQL muito lenta. Em sistemas de múltiplas CPUs, temos observado uma queda gradual na velocidade das consultas quando o número de clientes aumenta. No processo de tentar encontrar uma solução, recebemos um patch do kernel de um de nossos usuários, que alega fazer muita diferença para seu site. O patch está disponível aqui (<http://www.mysql.com/Downloads/Patches/linux-fork.patch>). Atualmente temos feito testes extensivos deste patch nos sistemas de desenvolvimento e produção. A performance do MySQL obtem uma melhora significativa, sem causar problemas e atualmente o recomendamos para nossos usuários que continuando trabalhando com servidores muito carregados em kernels 2.2. Este detalhe foi corrigido no kernel 2.4, portanto, se você não está satisfeito com a performance atual do seu sistema, melhor do que aplicar um patch ao seu kernel 2.2, pode ser mais fácil simplesmente atualizar para o 2.4, que lhe dará também uma melhora em seu sistemas SMP em adição à correção do bug discutido aqui.

Estamos testando o MySQL no kernel 2.4 em uma máquina com 2 processadores e descobrimos que o MySQL escala muito melhor - virtualmente, não há nenhuma perda de desempenho no throughput das consultas até cerca de 1000 clientes, e o fator da escala do MySQL (computado com a razão do throughput máximo para o throughput de cada cliente.) foi de 180%. Temos observado resultados similares em sistemas com 4 processadores - virtualmente não há perda de desempenho quando o número de clientes é incrementado até 1000 e o fator da escala foi de 300%. Portanto para um servidor SMP muito carregado nós definitivamente recomendamos o kernel 2.4. Nós descobrimos que é essencial executar o processo `mysqld` com a mais alta prioridade possível no kernel 2.4 para obter performance máxima. Isto pode ser feito adicionando o comando `renice -20 $$` ao `safe_mysqld`. Nos nossos testes em uma máquina com 4 processadores, o aumento da prioridade nos deu 60% de aumento no throughput com 400 clientes.

Atualmente estamos tentando coletar mais informações sobre como o MySQL atua no kernel 2.4 em sistemas com 4 e 8 processadores. Se você tem acesso a um sistema deste porte e tem feito alguns benchmarks, por favor envie um email para docs@mysql.com com os resultados - iremos incluí-los neste manual.

Existe outro detalhe que afeta muito a performance do MySQL, especialmente em sistemas multi processados. A implementação de mutex em LinuxThreads na **glibc-2.1** é muito ruim para programas com várias threads que travam o mutex por um tempo curto. Em um sistema SMP, ironicamente, se você liga o MySQL com **LinuxThreads** sem modificações, removendo processadores da máquina, a performance do MySQL é melhorada em alguns casos. Para corrigir este comportamento, disponibilizamos um patch para **glibc 2.1.3**, em [linuxthreads-2.1-patch](#)

Com a **glibc-2.2.2**, o MySQL versão 3.23.36 irá usar o mutex adaptativo, que é muito melhor, mesmo que o patch na **glibc-2.1.3**. Avisamos, entretanto, que sobre algumas condições, o código mutex no **glibc-2.2.2** overspins, que prejudica a performance do MySQL. A chance desta condição pode ser reduzida mudando a prioridade do processo `mysqld` para a prioridade mais alta. Nós também corrigimos o comportamento overspin com um patch, disponível em <http://www.mysql.com/Downloads/Linux/linuxthreads-2.2.2.patch.here>. Ele combina a correção do overspin, número máximo de threads e espaçamento das pilhas em um único patch. Você precisará aplicá-lo no diretório `linuxthreads` com `patch -p0 </tmp/linuxthreads-2.2.2.patch`. Acreditamos que será incluído de alguma forma nos futuros lançamentos da `glibc-2.2`. De qualquer forma, se você ligar com `glibc-2.2.2`, ainda será necessário corrigir `STACK_SIZE` e `PTHREAD_THREADS_MAX`. Temos esperanças que os padrões serão corrigidos para valores mais aceitáveis para configurações pesadas do MySQL no futuro, então sua construção poderá ser reduzida a `./configure; make; make install`.

Recomendamos que você use os patches acima para construir uma versão estática especial de `libpthread.a` e use-a somente para ligações estáticas com o MySQL. Sabemos que os patches são seguros para o MySQL e pode melhorar significamente sua performance, mas não podemos dizer nada sobre outras aplicações. Se você ligar outras aplicações coma a versão modificada da biblioteca ou construir uma versão alterada compartilhada e instalá-la no seu sistema, você estará fazendo por sua conta e risco e tenha atenção com outras aplicações que dependem de `LinuxThreads`.

Se você passar por problemas estranhos durante a instalação do MySQL ou com travamentos de alguns utilitários comuns, é muito provável que eles são relacionados a problemas de bibliotecas ou compilador. Se for este o caso, o uso de nosso binário será a solução.

Um problema conhecido com a distribuição binária é que com antigos sistemas Linux que usam `libc` (como o RedHat 4.x ou Slackware), você obterá alguns problemas não fatais com resolução de nomes. See [\[Binary notes-Linux\]](#), page [\[undefined\]](#).

Quando estiver usando LinuxThreads você verá um mínimo de três processos em execução. Estes são de fato, threads. Existirá uma thread para o gerenciador LinuxThreads, uma thread para lidar com conexões e uma thread para tartar de alarmes e sinais.

Perceba que o kernel Linux e a biblioteca LinuxThread pode por padrão ter apenas 1024 threads. Isto significa que você pode ter até 1021 conexões ao MySQL em um sistema sem correção. A página <http://www.volano.com/linuxnotes.html> contém informações sobre como contornar este limite.

Se você ver um processo `mysqld` daemon finalizado com `ps`, isto normalmente significa que você encontrou um bug no MySQL ou que tenha uma tabela corrompida. See [\[Crashing\]](#), page [\[undefined\]](#).

Para obter um descarga do core no Linux se o `mysqld` finalizar com um sinal SIGSEGV, você pode iniciar o `mysqld` com a opção `--core-file`. Perceba que provavelmente você também precisará aumentar o `core file size` adicionando `ulimit -c 1000000` para `safe_mysqld` ou iniciar `safe_mysqld` com `--core-file-sizes=1000000`, See [\[safe_mysqld\]](#), page [\[undefined\]](#).

Se você estiver ligando seu próprio cliente MySQL e obter o erro:

```
ld.so.1: ./my: fatal: libmysqlclient.so.4: open failed: No such file or directory
```

Quando executá-los, o problema pode ser evitado com um dos seguintes métodos:

- Ligue o cliente com a seguinte opção (no lugar de `-Lpath`): `-Wl,r/path-libmysqlclient.so`.
- Copie `libmysqlclient.so` para `'/usr/lib'`.
- Adicione o caminho do diretório onde `libmysqlclient.so` está localizado para a variável de ambiente `LD_RUN_PATH` antes de executar seu cliente.

Se você estiver usando o compilador Fujitsu (`fcc / FCC`) você terá alguns problemas compilando o MySQL porque os arquivos de cabeçalho Linux são muito orientados ao `gcc`.

A seguinte linha `configure` deve funcionar com `fcc/FCC`:

```
CC=fcc CFLAGS="-O -K fast -K lib -K omitfp -Kpreex -D_GNU_SOURCE -DCONST=const -DNO
```

2.6.1.1 Notas Linux para distribuições binárias

O MySQL necessita pelo menos do Linux versão 2.0

A versão binária é ligada com `-static`, que significa que você normalmente não precisa se preocupar com qual versão das bibliotecas do sistema você tem. Você não precisa instalar LinuxThreads. Um programa ligado com a opção `-static` é um pouco maior que um programa ligado dinamicamente e também um pouco mais rápido (3-5%). Um problema, entretanto, é que você não pode usar funções definidas pelo usuário (UDF) com um programa ligado estaticamente. Se você for escrever ou usar funções UDF (isto é algo para programadores C ou C++), você deve compilar o MySQL, usando ligações dinâmicas.

Se você estiver usando um sistema baseado em `libc` (em vez de um sistema `glibc2`), você, provavelmente, terá alguns problemas com resolução de nomes de máquinas e `getpwnam()` com a versão binária. (Isto é porque o `glibc` infelizmente depende de algumas bibliotecas externas para resolver nomes de máquinas e `getpwent()`, mesmo quando compilado com `-static`). Neste caso, você provavelmente obterá a seguinte mensagem de erro quando executar `mysql_install_db`:

```
Sorry, the host 'xxxx' could not be looked up
```

ou o seguinte erro quando você tentar executar `mysqld` com a opção `--user`:

```
getpwnam: No such file or directory
```

Você pode resolver este problema usando de um dos modos seguintes:

- Obtenha uma distribuição fonte do MySQL (uma distribuição RPM ou `tar.gz`) e a instale.
- Execute `mysql_install_db --force`; Isto não executará o teste `resolveip` no `mysql_install_db`. O lado ruim é que você não poderá usar nomes de máquinas nas tabelas de permissões; você deve usar números IP no lugar (exceto para `localhost`). Se você estiver usando uma release antiga do MySQL que não suporte `--force`, você deve remover o teste `resolveip` no `mysql_install` com um editor.
- Inicie `mysqld` com `su` no lugar de usar `--user`.

As distribuições binárias Linux-Intel e RPM do MySQL são configuradas para o máximo de desempenho possível. Nós sempre tentamos usar o compilador mais rápido e estável disponível.

Suporte MySQL ao Perl exige Perl Versão 5.004.03 ou mais novo.

Em algumas versões 2.2 do kernel Linux, você pode obter o erro `Resource temporarily unavailable` quando você faz várias novas conexões para um servidor `mysqld` sobre TCP/IP.

O problema é que o Linux tem um atraso entre o momento em que você fecha um socket TCP/IP até que ele seja realmente liberado pelo sistema. Como só existe espaço para um número finito de slots TCP/IP, você irá obter o erro acima se você tentar fazer muitas novas conexões TCP/IP durante um pequeno tempo, como quando você executa o benchmark do MySQL `'test-connect'` sobre TCP/IP.

Nós enviamos emails sobre este problema várias vezes para diferentes listas de discussão Linux mas nunca conseguimos resolver este problema apropriadamente.

A única 'correção' conhecida, para este problema é usar conexões persistentes nos seus clientes ou usar sockets, se você estiver executando o servidor de banco de dados e clientes na mesma máquina. Nós esperamos que o kernel Linux 2.4 corrija este problema no futuro.

2.6.1.2 Notas Linux x86

O MySQL exige a versão 5.4.12 ou mais nova da `libc`. Sabe-se que funciona com a `libc` 5.4.46. A versão 2.0.6 e posterior da `glibc` também deve funcionar. Existem alguns problemas com os RPMs `glibc` da RedHat, portanto se você tiver problemas, confira se existe alguma atualização! Sabemos que os RPMs `glibc` 2.0.7-19 e 2.0.7-29 funcionam.

Em algumas distribuições Linux mais antigas, `configure` pode produzir um erro como este:

```
Syntax error in sched.h. Change _P to __P in the /usr/include/sched.h file.
See the Installation chapter in the Reference Manual.
```

Faça apenas o que a mensagem de erro diz e adicione um caractere sublinhado para a macro `_P` que tem somente um caractere sublinhado e então tente novamente.

Você pode obter alguns avisos quando estiver compilando; os mostrados abaixo podem ser ignorados:

```
mysqld.cc -o objs-thread/mysqld.o
mysqld.cc: In function 'void init_signals()':
mysqld.cc:315: warning: assignment of negative value '-1' to 'long unsigned int'
mysqld.cc: In function 'void * signal_hand(void *)':
mysqld.cc:346: warning: assignment of negative value '-1' to 'long unsigned int'
```

No Debian GNU/Linux, se você deseja que o MySQL inicie automaticamente quando o sistema iniciar, faça o seguinte:

```
shell> cp support-files/mysql.server /etc/init.d/mysql.server
shell> /usr/sbin/update-rc.d mysql.server defaults 99
```

O `mysql.server` pode ser encontrado no diretório `'share/mysql'` sob o diretório de instalação MySQL ou no diretório `'support-files'` da árvore fonte MySQL.

Se o `mysqld` sempre descarregar um core na inicialização, o problema pode ser que você tenha um antigo `'/lib/libc.a'`. Tente renomeá-lo depois remova `'sql/mysqld'` e faça um novo `make install` e tente novamente. Este problema foi relatado em algumas instalações Slackware.

Se você obter o seguinte erro quando ligar o `mysqld`, significa que seu `'libg++.a'` não está instalado corretamente:

```
/usr/lib/libc.a(putc.o): In function '_IO_putc':
putc.o(.text+0x0): multiple definition of '_IO_putc'
```

Você pode evitar o uso de `'libg++.a'` executando `configure` desta forma:

```
shell> CXX=gcc ./configure
```

2.6.1.3 Notas Linux SPARC

Em algumas implementações, `readdir_r()` está quebrada. O sintoma é que `SHOW DATABASES` sempre retorna um conjunto vazio. Isto pode ser corrigido removendo `HAVE_READDIR_R` do `'config.h'` depois de configurar e antes de compilar.

Para alguns problemas será necessário corrigir a sua instalação Linux. O patch pode ser encontrado em <http://www.mysql.com/Downloads/patches/Linux-sparc-2.0.30.diff>. Este patch é para a distribuição Linux `'sparclinux-2.0.30.tar.gz'` que está disponível em `vger.rutgers.edu` (Uma versão de Linux que nunca foi fundido com o 2.0.30 oficial). Você também deve instalar LinuxThreads Versão 0.6 ou mais nova.

2.6.1.4 Notas Linux Alpha

O MySQL Versão 3.23.12 é a primeira versão do MySQL que é testada no Linux-Alpha. Se você planeja usar o MySQL no Linux-Alpha, você deve ter certeza que possui esta versão ou mais nova.

Temos testado o MySQL no Alpha com nossos pacotes de benchmarks e testes, e ele parece funcionar muito bem. A principal coisa que ainda não tivemos tempo de testar é como as coisas funcionam com muitos usuários simultâneos.

Quando nós compilamos o binários MySQL padrões, nós estávamos usando SuSE 6.4, kernel 2.2.13-SMP, Compilador C Compaq (V6.2-504) e compilador C++ Compaq (V6.3-005) em uma máquina Compaq DS20 com um processador Alpha EV6.

Você pode encontrar os compiladores acima em <http://www.support.compaq.com/alpha-tools>). Usando estes compiladores, em vez do gcc, obtemos 9-14 % de melhora na performance com MySQL.

Note que a linha de configuração otimiza o binário para a CPU atual; isto significa que você só pode utilizar nosso binário se você tiver um processador Alpha EV6. Nós também compilamos estaticamente para evitar problemas de bibliotecas.

```
CC=ccc CFLAGS="-fast" CXX=cxx CXXFLAGS="-fast -noexceptions -nortti" ./configure --
```

Se você deseja usar egcs a seguinte linha de configuração funcionou para nós:

```
CFLAGS="-O3 -fomit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -fomit-frame-pointer -felid
```

Alguns problemas conhecidos quando executamos o MySQL no Linux-Alpha:

- Debugar aplicações baseadas em threads como o MySQL não irá funcionar com gdb 4.18. Você deve fazer download e usar o gdb 5.0!
- Se você tentar ligar o `mysqld` estaticamente quando usar o `gcc`, a imagem resultante irá descarregar um arquivo core no início. Em outras palavras, **NÃO** use `--with-mysqld-ldflags=-all-static` com `gcc`.

2.6.1.5 Notas Linux PowerPC

O MySQL deve funcionar no MkLinux com o mais novo pacote `glibc` (testado com `glibc 2.0.7`).

2.6.1.6 Notas Linux MIPS

Para ter o MySQL funcionando no Qube2. (Linux Mips), você precisará das bibliotecas `glibc` mais novas (Sabemos que `glibc-2.0.7.29C2` funciona). Você também deve usar o compilador egcs C++ (`egcs-1.0.2-9`, `gcc 2.95.2` ou mais nova).

2.6.1.7 Notas Linux IA64

Para conseguir compilar o MySQL no Linux Ia64, usamos a seguinte linha de compilação: Usando `gcc-2.96`:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -fno-omit-frame-p
```

No Ia64 os binários do cliente MySQL estão usando bibliotecas compartilhadas. Isto significa se você instalar nossa distribuição binárias em algum outro lugar diferente de `‘/usr/local/mysql’` você precisa modificar o `‘/etc/ld.so.conf’` ou adicionar o caminho da o diretório onde está localizado o `‘libmysqlclient.so’` na variável de ambiente `LD_LIBRARY_PATH`.

See [\[Link errors\]](#), page [\[Link errors\]](#).

2.6.2 Notas Windows

Esta seção descreve a instalação e uso do MySQL no Windows. Esta informação também é fornecida no arquivo 'README' que acompanha a distribuição MySQL para Windows.

2.6.2.1 Iniciando o MySQL no Win95 / Win98

O MySQL usa TCP/IP para conectar um cliente a um servidor. (Isto permitirá que qualquer máquina na sua rede se conecte a seu servidor MySQL.) Por causa disto, você deve instalar o TCP/IP na sua máquina antes de iniciar o MySQL. Você pode encontrar TCP/IP no seu CD-ROM do Windows.

Perceba que se você estiver usando uma versão antiga do Win95 (por exemplo, OSR2). É preferível que você use um pacote antigo Winsock! Para o MySQL é necessário o Winsock 2! Você pode obter o Winsock mais novo em <http://www.microsoft.com>. O Win98 tem a nova biblioteca Winsock 2, portanto o que foi dito acima não se aplica para o Win98.

Para iniciar o servidor `mysqld`, você deve iniciar uma janela do Prompt do MS-DOS e digitar:

```
C:\> C:\mysql\bin\mysqld
```

Isto irá iniciar o `mysqld` em segundo plano, sem uma janela.

Você pode finalizar o servidor MySQL executando:

```
C:\> C:\mysql\bin\mysqladmin -u root shutdown
```

Perceba que o Win95 e o Win98 não suporta criação de named pipes. No Win95 e Win98, você pode usar somente named pipes para se conectar a um servidor MySQL remoto, executando em uma máquina Windows NT server. (É claro que o servidor MySQL deve também suportar named pipes. Por exemplo, usando `mysqld-opt` sobre NT não irá permitir conexões named pipe. Você deve usar o `mysqld-nt` ou `mysqld-max-nt`.)

Se o `mysqld` não iniciar, por favor, confira o arquivo '`\mysql\data\mysql.err`' para ver se o servidor escreveu alguma mensagem que possa indicar a causa do problema. Você pode também tentar iniciar o servidor com `mysqld --standalone`; Neste caso, você pode obter alguma informação útil na tela que pode ajudar a resolver o problema.

A última opção é iniciar o `mysqld` com `--standalone --debug`. Neste caso o `mysqld` irá escrever em um arquivo log '`C:\mysql.trace`' que deve conter a razão pela qual o `mysqld` não inicia. See [\[Making trace files\]](#), page [\[Making trace files\]](#).

2.6.2.2 Iniciando o MySQL no NT / Win2000

A seção Win95/Win98 também se aplica para o MySQL no NT/Win2000 com as seguintes diferenças:

Para ter o MySQL funcionando com TCP/IP no NT, você deve instalar o service pack 3 (ou mais novo)!

Perceba que tudo a seguir que se aplique ao NT, funcionará também para o Win2000!

Para NT/Win2000, o nome do servidor é `mysqld-nt`. Normalmente você deve instalar o MySQL como um serviço no NT/Win2000:

```
C:\> C:\mysql\bin\mysqld-nt --install
```

ou

```
C:\> C:\mysql\bin\mysqld-max-nt --install
```

(No Windows NT, você pode instalar qualquer dos binários do servidor como um serviço, mas somente os que tem nomes que terminam com `-nt.exe` fornecem suporte para named pipes.)

Você pode iniciar e parar o serviço MySQL com estes comandos:

```
C:\> NET START mysql
C:\> NET STOP mysql
```

Note que neste caso você não pode usar qualquer outra opção para `mysqld-nt`!

Você pode também executar `mysqld-nt` como um programa a parte no NT caso você precise iniciar o `mysqld-nt` com algumas opções! Se você inicia o `mysqld-nt` sem opções no NT, `mysqld-nt` tenta iniciar a ele próprio como um serviço com as opções padrões do serviço. Se você parou o `mysqld-nt`, você deve iniciá-lo com `NET START mysql`.

O serviço é instalado com o nome `MySQL`. Uma vez instalado, ele deve ser iniciado usando o utilitário Services Control Manager (SCM) encontrado no Painel de Controle, ou usando o comando `NET START MySQL`. Se algumas opções são desejadas, elas devem ser especificadas como “Parâmetros de inicialização” no utilitário SCM antes de iniciar o serviço MySQL. Uma vez em execução, `mysqld-nt` pode ser parado usando `mysqladmin`, ou do utilitário SCM ou usando o comando `NET STOP MySQL`. Se você utiliza o SCM para parar o `mysqld-nt`, existe uma estranha mensagem do SCM sobre `mysqld shutdown normally`. Quando você executa como um serviço, `mysqld-nt` não tem acesso ao console e então as mensagens não podem ser vistas.

No NT você pode obter as seguintes mensagens de erro do serviço:

Permission Denied	Significa que ele não pode encontrar <code>mysqld-nt.exe</code> .
Cannot Register	Significa que o caminho é incorreto.
Failed to install service.	Significa que o serviço já está instalado ou que o Service Control Manager está em mau estado.

Se você tiver problemas para instalar `mysqld-nt` como um serviço, tente iniciá-lo com o caminho completo:

```
C:\> C:\mysql\bin\mysqld-nt --install
```

Se isto não funcionar, você pode ter o `mysqld-nt` iniciando corretamente corrigindo o caminho no registro!

Se você não deseja iniciar o `mysqld-nt` como um serviço, você pode iniciá-lo como mostrado abaixo:

```
C:\> C:\mysql\bin\mysqld-nt --standalone
```

ou

```
C:\> C:\mysql\bin\mysqld --standalone --debug
```

Este último exemplo fornece um arquivo para depuração em ‘`C:\mysqld.trace`’. See [\(undefined\)](#) [Making trace files], page [\(undefined\)](#).

2.6.2.3 Executando o MySQL no Windows

O MySQL suporta TCP/IP em todas plataformas Windows e named pipes no NT. O padrão é usar named pipes para instalações locais no NT e TCP/IP para todos outros casos se o cliente tiver TCP/IP instalado. O nome da máquina especifica qual protocolo é usado:

Nome da Máquina	Protocolo
NULL (nenhum)	No NT, tente o named pipes antes; se isto não funcionar, use TCP/IP. No Win95/Win98, TCP/IP é usado.
.	Named pipes
localhost	TCP/IP para máquina atual
hostname	TCP/IP

Você pode forçar um cliente MySQL a usar named pipes especificando a opção `--pipe` ou especificando `.` como o nome da máquina. Utilize a opção `--socket` para especificar o nome do pipe.

Perceba que a partir da versão 3.23.50, named pipes só estão habilitados se o `mysqld` for iniciado com a opção `--enable-named-pipe`. Isto é porque alguns usuários tiveram problemas desligando o servidor MySQL quando este utilizava named pipes.

Você pode testar se o MySQL está funcionando ou não executando os seguintes comandos:

```
C:\> C:\mysql\bin\mysqlshow
C:\> C:\mysql\bin\mysqlshow -u root mysql
C:\> C:\mysql\bin\mysqladmin version status proc
C:\> C:\mysql\bin\mysql test
```

Se o `mysqld` está lento para responder a suas conexões no Win95/Win98, provavelmente existe um problema com seu DNS. Neste caso, inicie o `mysqld` com `--skip-name-resolve` e use somente `localhost` e números IP nas tabelas de permissões do MySQL. Você também pode evitar o DNS quando estiver conectando em um servidor MySQL `mysqld-nt` executando no NT, utilizando o argumento `--pipe` para especificar o uso de named pipes. Isto funciona para a maioria de clientes MySQL.

Existem duas versões da ferramenta de linha de comando MySQL:

<code>mysql</code>	Compilado em Windows nativo, que oferece capacidades de edição de texto muito limitadas.
<code>mysqlc</code>	Compilado com o compilador Cygnus GNU, que oferece edição <code>readline</code> .

Se você desejar usar o `mysqlc.exe`, deve copiar o `'C:\mysql\lib\cygwinb19.dll'` para o diretório `system` do seu Windows (`'\windows\system'` ou um lugar parecido).

Os privilégios padrões no Windows dão a todos usuários locais privilégios totais para todos os bancos de dados sem necessidade de especificar uma senha. Para deixar o MySQL mais seguro, você deve configurar uma senha para todos os usuário e remover a linha na tabela `mysql.user` que tem `Host='localhost'` e `User=''`.

Você também deve adicionar uma senha para o usuário `root`. O exemplo seguinte inicia removendo o usuário anônimo que pode ser usando por qualquer um para acessar o banco de dados `test`, e então configura uma senha para o usuário `root`:

```
C:\> C:\mysql\bin\mysql mysql
mysql> DELETE FROM user WHERE Host='localhost' AND User='';
mysql> QUIT
C:\> C:\mysql\bin\mysqladmin reload
C:\> C:\mysql\bin\mysqladmin -u root password sua_senha
```

Depois de configurar a senha, se você desejar desligar o servidor `mysqld`, você pode usar o seguinte comando:

```
C:\> mysqladmin --user=root --password=sua_senha shutdown
```

Se você estiver usando a antiga versão shareware do MySQL versão 3.21 no Windows, o comando acima irá falhar com um erro: `parse error near 'SET OPTION password'`. A correção é feita atualizando a versão do MySQL, que está disponível livremente.

Com as versões atuais do MySQL você pode facilmente adicionar novos usuários e alterar privilégios com os comandos `GRANT` e `REVOKE`. See [\[GRANT\]](#), page [\[undefined\]](#).

2.6.2.4 Conectando em um MySQL remoto do Windows com SSH

Aqui temos notas sobre como conectar a um servidor MySQL através de uma conexão remota e segura usando o SSH (por David Carlson dcarlson@mplcomm.com):

- Instale um cliente SSH na sua máquina Windows. Como um usuário, o melhor opção paga que encontrei é o SecureCRT da <http://www.vandyke.com/>. Outra opção é o `f-secure` da <http://www.f-secure.com/>. Você também pode encontrar algumas versões livres no Google em http://directory.google.com/Top/Computers/Security/Products_and_Tools/Cryptography/SSH/Clients/Windows/.
- Inicie seu cliente SSH Windows. Configure `Host_Name = IP_ou_Nome_servidormysql`. Configure `userid=seu_userid` para logar no seu servidor (provavelmente não o mesmo que seu usuário/senha do MySQL).
- Configure a porta de acesso. E também faça um acesso remoto (Configure `local_port: 3306, remote_host: ip_ou_nomeservidormysql, remote_port: 3306`) ou um acesso local (configure `port: 3306, host: localhost, remote port: 3306`).
- Salve tudo, senão você terá que refazer tudo da próxima vez.
- Logue ao seu servidor com a sessão SSH que acabou de ser criada.
- Na sua máquina Windows, inicie algumas aplicações ODBC (como o Access).
- Crie um novo arquivo no Windows e ligue ao MySQL usando o driver ODBC da mesma forma que você normalmente faz, EXCETO pelo fato de digitar `localhost` para a máquina servidora MySQL — não `nomeservidormysql`.

Você agora deve ter uma conexão ODBC ao MySQL, criptografada com SSH.

2.6.2.5 Dividindo dados entre diferentes discos no Win32

A partir do MySQL versão 3.23.16, o `mysqld-max` e servidores `mysql-max-nt` na distribuição MySQL são compilados com a opção `-DUSE_SYMDIR`. Isto permite que você coloque um banco de dados em discos diferentes adicionando um link simbólico para ele (em forma similar aos links simbólicos no Unix).

No Windows, você cria um link simbólico para um banco de dados criando um arquivo que contem o caminho para o diretório de destino e salvando-o no diretório `'mysql_data'` com o arquivo `'database.sym'`. Note que o link simbólico só será usada se o diretório `'mysql_data_dir\database'` não existir.

Por exemplo, se o diretório de dados do MySQL é `'C:\mysql\data'` e você precisa ter o banco de dados `foo` localizado em `'D:\data\foo'`, você deve criar o arquivo `'C:\mysql\data\foo.sym'` que contém o texto `D:\data\foo\`. Depois disto, todas tabelas criadas no banco de dados `foo` serão criadas no `'D:\data\foo'`.

Note que devido a penalidade que você tem na velocidade quando abre todas as tabelas, nós não habilitamos esta opção por padrão, mesmo se você compilar o MySQL com suporte

a isto. Para habilitar links simbólicos você deve colocar no seu arquivo `my.cnf` ou `my.ini` a seguinte entrada:

```
[mysqld]
use-symbolic-links
```

No MySQL 4.0 nós habilitaremos links simbólicos por padrão. Então você deve usar a opção `skip-symlink` se você desejar desabilitá-las.

2.6.2.6 Compilando clientes MySQL no Windows

Em seus arquivos fontes, você deve incluir `'windows.h'` antes de incluir `'mysql.h'`:

```
#if defined(_WIN32) || defined(_WIN64)
#include <windows.h>
#endif
#include <mysql.h>
```

Você também pode ligar seu código coma biblioteca dinâmica `'libmysq.lib'`, que é apenas um wrapper para carregar em `'libmysql.dll'` sobre demanda, ou ligar com a biblioteca estática `'mysqlclient.lib'`.

Perceba que como as bibliotecas `mysqlclient` são compiladas como bibliotecas `threaded`, você também deve compilar seu código para ser `multi-threaded`!

2.6.2.7 MySQL-windows comparado com o unix MySQL

O MySQL-Windows tem provado ser muito estável. Esta versão do MySQL tem os mesmos recursos que sua versão correspondente Unix com as seguintes exceções:

Win95 e threads

O Win95 perde aproximadamente 200 bytes de memória principal para cada thread criada. Cada conexão no MySQL cria uma nova thread, portanto você não deve executar o `mysqld` por um longo tempo no Win95 se seu servidor lida com várias conexões! WinNT e Win98 não sofrem deste bug.

Leituras simultâneas

O MySQL depende das chamadas `pread()` e `pwrite()` para estar apto a misturar `INSERT` e `SELECT`. Atualmente nós usamos mutexes para emular `pread()/pwrite()`. Nós iremos, a longo prazo, trocar o nível da interface de arquivos com uma interface virtual para que nós possamos usar a interface `readfile()/writefile()` no NT para obter mais velocidade. A implementação atual limita o número de arquivos abertos que o MySQL pode usar para 1024, o que significa que você não conseguirá executar tantas threads simultâneas no NT como no Unix.

Leitura de blocos

O MySQL usa uma leitura de blocos para cada conexão. Isto significa que:

- Uma conexão não irá ser desconectada automaticamente depois de 8 horas, como acontece com a versão Unix do MySQL.
- Se uma conexão trava, é impossível a finaliza-la sem matar o MySQL.
- `mysqladmin kill` não irá funcionar em uma conexão adormecida.

- `mysqladmin shutdown` não pode abortar enquanto existirem conexões adormecidas.

Planejamos corrigir este problema quando nossos desenvolvedores Windows tiverem conseguido um boa solução.

Funções UDF

Até o momento, o MySQL-Windows não suporta funções definidas pelo usuário.

DROP DATABASE

Você não pode remover um banco de dados que está em uso por alguma thread.

Matando o MySQL do gerenciador de tarefas

Você não pode matar o MySQL do gerenciador de tarefas ou com o utilitário `shutdown` no Win95. Você deve desligá-lo com `mysqladmin shutdown`.

Nomes case-insensitivo

Nomes de arquivos são caso-insensitivo no Windows, portanto, nomes de bancos de dados e tabelas também são caso insensitivo no MySQL para Windows. A única restrição é que os nomes de bancos de dados e tabelas devem usar o mesmo caso em uma sentença fornecida. See [\(undefined\)](#) [Name case sensitivity], page [\(undefined\)](#).

O caracter de diretório '\'

Componentes de nomes de caminho no Win95 são separados pelo caracter '\ ' o qual também é o caractere de escape no MySQL. Se você estiver usando `LOAD DATA INFILE` ou `SELECT ... INTO OUTFILE`, você deve dobrar o caractere '\':

```
mysql> LOAD DATA INFILE "C:\\tmp\\skr.txt" INTO TABLE skr;
mysql> SELECT * INTO OUTFILE 'C:\\tmp\\skr.txt' FROM skr;
```

Uma alternativa é usar o estilo de nome de arquivos do Unix com caracteres '/':

```
mysql> LOAD DATA INFILE "C:/tmp/skr.txt" INTO TABLE skr;
mysql> SELECT * INTO OUTFILE 'C:/tmp/skr.txt' FROM skr;
```

erro: Can't open named pipe

Se você utiliza uma versão 3.22 do MySQL no NT com o os clientes MySQL mais novos, será apresentado o seguinte erro:

```
error 2017: can't open named pipe to host: . pipe...
```

Isto ocorre porque a versão do MySQL usa named pipes no NT por padrão. Você pode evitar este erro usando a opção `--host=localhost` para os novos clientes MySQL ou criar um arquivo de opções 'c:\my.cnf' que contenha a seguinte informação:

```
[client]
host = localhost
```

A partir da versão 3.23.50, named pipes são habilitados somente se o `mysqld` é iniciado com a opção `--enable-name-pipe`.

Erro Access denied for user

Se você obtém o erro `Access denied for user: 'some-user@unknown' to database 'mysql'` quando acessar um servidor MySQL na mesma máquina, significa que o MySQL não pode resolver seu nome de máquina corretamente.

Para corrigir isto, você deve criar um arquivo '`\windows\hosts`' com a seguinte informação:

```
127.0.0.1      localhost
```

ALTER TABLE

Enquanto você estiver executando uma cláusula `ALTER TABLE`, a tabela será bloqueada para o uso por outras threads. Isto é relacionado com o fato de que no Windows, você não pode apagar um arquivo que está sendo usado por outras threads. (No futuro, iremos encontrar alguma forma de contornar este problema.)

DROP TABLE em uma tabela que está em uso por uma tabela MERGE não irá funcionar.

O manipulador `MERGE` faz seu mapeamento de tabelas escondido do MySQL. Como o Windows não permite apagar arquivos que estão abertos, você deve primeiramente descarregar todas tabelas `MERGE` (com `FLUSH TABLES`) ou apagar a tabela `MERGE` antes de apagar a tabela. Iremos corrigir isto quando introduzirmos as `VIEWS`.

Aqui estão alguns assuntos em aberto para qualquer um que queira nos ajudar com a versão Windows:

- Criar um servidor `MYSQL.DLL` mono-usuário. Isto deve incluir tudo de um servidor MySQL padrão, exceto criação de threads. Isto deixará o MySQL muito mais fácil para ser usado em aplicações que não necessitam de um verdadeiro cliente/servidor e não necessitam acessar o servidor de outras máquinas.
- Adicionar alguns ícones agradáveis para o start e shutdown na instalação do MySQL.
- Criar uma ferramenta para gerenciar entradas de registro para as opções de inicialização do MySQL. A leitura de entradas no registro já está codificada em '`mysqld.cc`', mas ela deve ser refeita para ser mais orientada a parâmetros. A ferramenta também deve estar apta para atualizar o arquivo de opções '`C:\my.cnf`' se o usuário preferir usa-lo no lugar do registro.
- Quando registrar o `mysqld` como um serviço com `--install` (no NT) seria ótimo se você também pudesse adicionar opções padrões na linha de comando. Para o momento, a solução é listar os parâmetros no arquivo '`C:\my.cnf`'.
- Seria muito interessante conseguir matar o `mysqld` do gerenciador de tarefas. Para o momento, deve ser usado o `mysqladmin shutdown`.
- Portar o `readline` para Windows para uso na ferramenta de linha de comando `mysql`.
- Versões GUI dos clientes MySQL padrões (`mysql`, `mysqlshow`, `mysqladmin` e `mysqldump`) seria ótimo.
- Seria muito bom se as funções de leitura e escrita no socket em '`net.c`' fosse interrompíveis. Isto tornaria possível matar threads abertas com `mysqladmin kill` no Windows.
- `mysqld` sempre inicia na localidade "C" e não na padrão. Gostaríamos de ter o `mysqld` usando a localização atual para a ordem de classificação.
- Implementar funções UDF com `.DLLs`.
- Adicionar macros para usar os métodos mais rápidos de incremento/decremento de threads seguras fornecidos pelo Windows.

Outros detalhes específicos do Windows são descritos no arquivo 'README' que acompanha a distribuição MySQL-Windows.

2.6.3 Notas Solaris

No Solaris, você deve ter problemas mesmo antes de descompactar a distribuição MySQL! O `tar` do Solaris não pode tratar grandes nomes de arquivos, portanto você pode ver um erro deste tipo quando descompactar o MySQL:

```
x mysql-3.22.12-beta/bench/Results/ATIS-mysql_odbc-NT_4.0-cmp-db2,informix,ms-sql,m
tar: directory checksum error
```

Neste caso, você deve usar o GNU `tar` (`gtar`) para desempacotar a distribuição. Você pode encontrar uma cópia pré-compilada para Solaris em <http://www.mysql.com/Downloads/>.

As threads nativas da Sun funcionam somente no Solaris 2.5 e superior. Para a versão 2.4 e anteriores, o MySQL irá automaticamente usar MIT-pthreads. See [\[MIT-pthreads\]](#), page [\[undefined\]](#).

Se você obter o seguinte erro de configure:

```
checking for restartable system calls... configure: error can not run test
programs while cross compiling
```

Isto significa que alguma coisa está errada com a instalação de seu compilador! Neste caso você deve atualizar seu compilador para uma versão mais nova. Você também pode resolver este problema inserindo a seguinte linha no arquivo 'config.cache':

```
ac_cv_sys_restartable_syscalls=${ac_cv_sys_restartable_syscalls='no'}
```

Se você está usando Solaris em um SPARC, o compilador recomendado é o `gcc 2.95.2`. Você pode encontrá-lo em <http://gcc.gnu.org/>. Perceba que `egcs 1.1.1` e `gcc 2.8.1` não são estáveis no SPARC!

A linha do `configure` recomendado quando usando `gcc 2.95.2` é:

```
CC=gcc CFLAGS="-O3" \
CXX=gcc CXXFLAGS="-O3 -felide-constructors -fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql --with-low-memory --enable-assembler
```

Se você possui um `ultra sparc`, você pode obter 4% a mais de performance adicionando "`-mcpu=v8 -Wa,-xarch=v8plusa`" para a `CFLAGS` e `CXXFLAGS`.

Se você possui o compilador Sun Workshop (Fortre) 5.3 (ou mais novo), você pode executar `configure` da seguinte forma:

```
CC=cc CFLAGS="-Xa -fast -xO4 -native -xstrconst -mt" \
CXX=CC CXXFLAGS="-noex -xO4 -mt" \
./configure --prefix=/usr/local/mysql --enable-assembler
```

Nos benchmarks do MySQL, nós obtemos um aumento de 6% em um `Ultrasparc` quando usado o Sun Workshop 5.3 comparado ao uso do `gcc` com as opções `-mcpu`.

Se você tiver um problema com `fdatasync` ou `sched_yield`, você pode corrigir isto adicionando `LIBS=-lrt` para a linha de configuração

O seguinte parágrafo é relevante somente para compiladores mais antigos que o `WorkShop 5.3`:

Você também pode ter que editar o script `configure` para alterar esta linha:

```
#if !defined(__STDC__) || __STDC__ != 1
```

para isto:

```
#if !defined(__STDC__)
```

Se você ligar `__STDC__` com a opção `-Xc`, o compilador Sun não pode compilar com o arquivo de cabeçalho `'pthread.h'` do Solaris. Isto é um bug da Sun (compilador corrompido ou arquivo include corrompido).

Se o `mysqld` emitir a mensagem de erro mostrada abaixo quando você executá-lo, você deve tentar compilar o MySQL com o compilador Sun sem habilitar a opção multi-thread (`-mt`):

```
libc internal error: _rmutex_unlock: rmutex not held
```

Adicione `-mt` a `CFLAGS` e `CXXFLAGS` e tente novamente.

Se você obter o seguinte erro quando estiver compilando o MySQL com `gcc`, significa que seu `gcc` não está configurado para sua versão de Solaris:

```
shell> gcc -O3 -g -O2 -DDEBUG_OFF -o thr_alarm ...
./thr_alarm.c: In function 'signal_hand':
./thr_alarm.c:556: too many arguments to function 'sigwait'
```

A coisa apropriada para fazer neste caso é obter a versão mais nova do `gcc` e compilá-lo com seu compilador `gcc` atual! Ao menos para o Solaris 2.5, a maioria das versões binárias de `gcc` tem arquivos inúteis e antigos que irão quebrar todos programas que usam threads (e possivelmente outros programas)!

O Solaris não fornece versões estáticas de todas bibliotecas de sistema (`libpthreads`) e `libdl`), portanto você não pode compilar o MySQL com `--static`. Se você tentar fazer isto, receberá o erro:

```
ld: fatal: library -ldl: not found
```

ou

```
undefined reference to 'dlopen'
```

ou

```
cannot find -lrt
```

Se vários processos tentar conectar muito rapidamente ao `mysqld`, você verá este erro no log do MySQL:

```
Error in accept: Protocol error
```

Você deve tentar iniciar o servidor com a opção `--set-variable back_log=50` como uma solução para esta situação. See [\[Command-line options\]](#), page [\[undefined\]](#).

Se você está ligando seu próprio cliente MySQL, você deve obter o seguinte erro quando tentar executá-lo:

```
ld.so.1: ./my: fatal: libmysqlclient.so.#: open failed: No such file or directory
```

O problema pode ser evitado por um dos seguintes métodos:

- Ligue o cliente com a seguinte opção (em vez de `-Lpath`): `-Wl,r/full-path-to-libmysqlclient.so`.
- Copie o arquivo `'libmysqlclient.so'` para `'/usr/lib'`.

- Adicione o caminho do diretório onde `libmysqlclient.so` está localizado à variável de ambiente `LD_RUN_PATH` antes de executar seu cliente.

Quando estiver usando a opção do configure `--with-libwrap`, você também deve incluir as bibliotecas que o `libwrap.a` necessita:

```
--with-libwrap="/opt/NUtcpwrapper-7.6/lib/libwrap.a -lnsl -lsocket
```

Se você tiver problemas com o configure tentando ligar com `-lz` e você não tem a `zlib` instalada, você terá duas opções:

- Se você deseja usar o protocolo de comunicação de compactado você precisará obter e instalar a `zlib` from ftp.gnu.org.
- Configure com `--with-named-z-libs=no`.

Se você estiver usando o `gcc` e tiver problemas carregando funções UDF no MySQL, tente adicionar `-lgcc` para a linha de ligação para a função UDF.

Se você deseja que o MySQL inicie automaticamente, você pode copiar `support-files/mysql.server` para `/etc/init.d` e criar um link simbólico para ele, chamado `/etc/rc.3.d/S99mysql.server`.

2.6.3.1 Notas Solaris 2.7/2.8

Você pode utilizar normalmente um binário Solaris 2.6 no Solaris 2.7 e 2.8. A maioria dos detalhes do Solaris 2.6 também se aplicam ao Solaris 2.7 e 2.8.

Note que o MySQL versão 3.23.4 e superiores devem estar aptos para autodetectar novas versões do Solaris e habilitar soluções para os problemas seguintes!

Solaris 2.7 / 2.8 tem alguns bugs nos arquivos include. Você pode ver o seguinte erro quando você usa o `gcc`:

```
/usr/include/widec.h:42: warning: 'getwc' redefined
/usr/include/wchar.h:326: warning: this is the location of the previous
definition
```

Se isto ocorrer, você pode fazer o seguinte para corrigir o problema:

Copie `/usr/include/widec.h` para `.../lib/gcc-lib/os/gcc-version/include` e mude a linha 41 :

```
#if !defined(lint) && !defined(__lint)
```

para

```
#if !defined(lint) && !defined(__lint) && !defined(getwc)
```

Uma alternativa é editar o `/usr/include/widec.h` diretamente. Desta forma, depois de fazer a correção, você deve remover o `config.cache` e executar o `configure` novamente !

Se você obter erros como estes quando você executar o `make`, é porque o `configure` não encontrou o arquivo `curses.h` (provavelmente devido ao erro no arquivo `/usr/include/widec.h`):

```
In file included from mysql.cc:50:
/usr/include/term.h:1060: syntax error before ','
/usr/include/term.h:1081: syntax error before ';'

```

A solução para isto é fazer uma das seguintes opções:

- Configure com `CFLAGS=-DHAVE_CURSES_H CXXFLAGS=-DHAVE_CURSES_H ./configure`.
- Edite o `/usr/include/widec.h` como indicado acima e re-execute o `configure`.
- Remova a linha `#define HAVE_TERM` do arquivo `config.h` e execute `make` novamente.

Se o seu ligador tiver problemas para encontrar o `-lz` quando ligar ao seu programa cliente, provavelmente o problema é que seu arquivo `libz.so` está instalado em `/usr/local/lib`. Você pode corrigir isto usando um dos seguintes métodos:

- Adicione `/usr/local/lib` ao `LD_LIBRARY_PATH`.
- Adicione um link para `libz.so` a partir de `/lib`.
- Se você estiver usando o Solaris 8, você pode instalar a `zlib` opcional do CD de distribuição do Solaris 8.
- Configure o MySQL com a opção `--with-named-z-libs=no`.

2.6.3.2 Notas Solaris x86

No Solaris 2.8 no x86, `mysqld` irá descarregar um core se você executar um `'strip'` no mesmo. Se você estiver usando `gcc` ou `egcs` no Solaris X86 e você tiver problemas com descarregos de core, você deve utilizar o seguinte comando `configure`:

```
CC=gcc CFLAGS="-O3 -fomit-frame-pointer -DHAVE_CURSES_H" \
CXX=gcc \
CXXFLAGS="-O3 -fomit-frame-pointer -felide-constructors -fno-exceptions -fno-rtti -fno-rtti -fno-rtti" \
./configure --prefix=/usr/local/mysql
```

Isto irá evitar problemas com a biblioteca `libstdc++` e com exceções C++.

Se isto não ajudar, você pode compilar uma versão com debug e executá-lo com um arquivo de rastreamento (`trace`) ou sob `gdb`. See [\(undefined\) \[Using gdb on mysqld\]](#), page [\(undefined\)](#).

2.6.4 Notas BSD

2.6.4.1 Notas FreeBSD

FreeBSD 3.x é recomendado para execução do MySQL uma vez ue o pacote `thread` é muito mais integrado.

A mais fácil e portanto a forma preferida para instalá-lo é usar as portas `mysql-server` e `mysql-client` disponíveis em <http://www.freebsd.org>.

Usando-as você obtém:

- Um MySQL funcional, com todas as otimizações conhecidas para trabalhar na sua versão habilitada do FreeBSD.
- Configuração e construção automática.
- Scripts de inicialização instalados em `/usr/local/etc/rc.d`.
- Habilidade para ver quais arquivos estão instalados com `pkg_info -L`. E para remover todos com `pkg_delete` se você não quiser mais o MySQL na máquina.

É recomendado que você utilize MIT-threads no FreeBSD 2.x e threads nativas nas Versões 3 e superiores. É possível executar com threads nativas em algumas versões antigas (2.2.x) mas você pode encontrar problemas ao finalizar o `mysqld`.

O ‘Makefile’ do MySQL necessita o GNU make (**gmake**) para funcionar. Se você deseja compilar o MySQL, antes você precisará instalar o GNU make.

Tenha certeza que sua configuração de resolução de nomes esteja correta. De outra forma você vai ter atrasos na resolução ou falhas quando conectar ao **mysqld**.

Tenha certeza que a entrada **localhost** no arquivo ‘**/etc/hosts**’ esteja correta (de outra forma você irá ter problemas conectando ao banco de dados). O arquivo ‘**/etc/hosts**’ deve iniciar com a linha:

```
127.0.0.1      localhost localhost.seu.dominio
```

Se você notar que o **configure** irá usar MIT-pthreads, você deve ler as notas sobre o MIT-pthreads. See [\[MIT-pthreads\]](#), page [\[MIT-pthreads\]](#).

Se você obter um erro do **make install** dizendo que ele não consegue encontrar ‘**/usr/include/pthreads**’, o **configure** não percebeu que você necessita da MIT-pthreads. Isto é corrigido com os seguintes comandos:

```
shell> rm config.cache
shell> ./configure --with-mit-threads
```

O FreeBSD é também conhecido por ter um limite muito baixo para o manipulador de arquivos. See [\[Not enough file handles\]](#), page [\[Not enough file handles\]](#). Descomente a seção **ulimit -n** no **safe_mysqld** ou aumente os limites para o usuário **mysqld** no **/etc/login.conf** (e reconstrua-o com **cap_mkdb /etc/login.conf**). Também tenha certeza que você configurou a classe apropriada para este usuário no arquivo de senhas (**password**) se você não estiver usando o padrão (use: **chpass nome_usuario_mysqld**). See [\[safe_mysqld\]](#), page [\[safe_mysqld\]](#).

Se você tiver problemas com a data atual no MySQL, configurar a variável **TZ** provavelmente ajudará. See [\[Environment variables\]](#), page [\[Environment variables\]](#).

Para obter um sistema seguro e estável você deve usar somente kernels FreeBSD que estejam marcados com **-STABLE**.

2.6.4.2 Notas NetBSD

Para compilar no NetBSD você precisa do GNU **make**. De outra forma o compilador quebraria quando o **make** tentasse executar **lint** em arquivos C++.

2.6.4.3 Notas OpenBSD

2.6.4.4 Notas OpenBSD 2.5

No OpenBSD Versão 2.5, você pode compilar o MySQL com threads nativas com as seguintes opções:

```
CFLAGS=-pthread CXXFLAGS=-pthread ./configure --with-mit-threads=no
```

2.6.4.5 Notas OpenBSD 2.8

Nossos usuários relataram que o OpenBSD 2.8 tem um bug nas threads que causa problemas com o MySQL. Os desenvolvedores do OpenBSD já corrigiram o problema, mas em 25 de Janeiro de 2001 a correção foi disponível apenas no ramo “-current”. Os sintomas deste bug nas threads são: resposta lenta, alta carga, alto uso de CPU e quedas do servidor.

2.6.4.6 Notas BSD/OS

2.6.4.7 Notas BSD/OS Versão 2.x

Se você obter o seguinte erro quando estiver compilando o MySQL, seu valor `ulimit` para memória virtual é muito baixo:

```
item_func.h: In method 'Item_func_ge::Item_func_ge(const Item_func_ge &)':
item_func.h:28: virtual memory exhausted
make[2]: *** [item_func.o] Error 1
```

Tente usar `ulimit -v 80000` e executar o `make` novamente. Se isto não funcionar e você estiver usando o `bash`, tente trocar para `csh` ou `sh`; alguns usuários BSDI relataram problemas com `bash` e `ulimit`.

Se você utiliza `gcc`, você pode também ter de usar a opção `--with-low-memory` para o `configure` estar apto a compilar o `'sql_yacc.cc'`.

Se você tiver problemas com a data atual no MySQL, configurar a variável `TZ` provavelmente ajudará. See [\(undefined\) \[Environment variables\], page \(undefined\)](#).

2.6.4.8 Notas BSD/OS Version 3.x

Atualize para BSD/OS Versão 3.1. Se isto não for possível, instale BSDIpatch M300-038.

Use o seguinte comando quando configurar o MySQL:

```
shell> env CXX=shlicc++ CC=shlicc2 \
./configure \
--prefix=/usr/local/mysql \
--localstatedir=/var/mysql \
--without-perl \
--with-unix-socket-path=/var/mysql/mysql.sock
```

O comando seguinte também funciona:

```
shell> env CC=gcc CXX=gcc CXXFLAGS=-O3 \
./configure \
--prefix=/usr/local/mysql \
--with-unix-socket-path=/var/mysql/mysql.sock
```

Você pode alterar as localizações dos diretórios se você desejar, ou apenas usar os padrões não especificando nenhuma localização.

Se você tiver problemas com performance sob alta carga, tente usar a opção `--skip-thread-priority` para `mysqld`! Isto irá executar todas as threads com a mesma prioridade; no BSDI versão 3.1, isto fornece melhor performance (pelo menos até o BSDI corrigir seu organizador de threads).

Se você obter o erro `virtual memory exhausted` enquanto estiver compilando, deve tentar usar `ulimit -v 80000` e executar `make` novamente. Se isto não funcionar e você estiver usando `bash`, tente trocar para `csh` ou `sh`; alguns usuários BSDI relataram problemas com `bash` e `ulimit`.

2.6.4.9 Notas BSD/OS Versão 4.x

O BSDI Versão 4.x tem alguns bugs relacionados às threads. Se você deseja usar o MySQL nesta versão, você deve instalar todas as correções relacionadas às threads. Pelo menos a M400-23 deve estar instalada.

Em alguns sistemas BSDI versão 4.x, você pode ter problemas com bibliotecas compartilhadas. O sintoma é que você não pode executar nenhum programa cliente, por exemplo, `mysqladmin`. Neste caso você precisa reconfigurar o MySQL, para ele não usar bibliotecas compartilhadas, com a opção `--disable-shared`.

Alguns clientes tiveram problemas no BSDI 4.0.1 que o binário do `mysqld` não conseguia abrir tabelas depois de um tempo em funcionamento. Isto é porque alguns bugs relacionados a biblioteca/sistema fazem com que o `mysqld` altere o diretório atual sem nenhuma informação!

A correção é atualizar para a 3.23.34 ou depois de executar `configure` remova a linha `#define HAVE_REALPATH` de `config.h` antes de executar o `make`.

Perceba que com isso você não pode fazer um link simbólico de um diretório de banco de dados para outro diretório ou fazer um link simbólico a uma tabela para outro banco de dados no BSDI! (Criar um link simbólico para outro disco funciona).

2.6.5 Notas Mac OS X

2.6.5.1 Mac OS X Public Beta

O MySQL deve funcionar sem problemas no Mac OS X Public Beta (Darwin). Você não precisa dos patches `pthread` para este SO.

2.6.5.2 Mac OS X Server

Antes de tentar configurar o MySQL no MAC OS X server, primeiro você deve instalar o pacote `pthread` encontrado em: <http://www.prnet.de/RegEx/mysql.html>.

Nosso binário para o Mac OS X é compilado no Rhapsody 5.5 com a seguinte linha de configuração:

```
CC=gcc CFLAGS="-O2 -fomit-frame-pointer" CXX=gcc CXXFLAGS="-O2 -fomit-frame-pointer"
```

Você deve precisar adicionar também apelidos (alias) para seu arquivo de shell para acessar `mysql` e `mysqladmin` da linha de comando:

```
alias mysql '/usr/local/mysql/bin/mysql'
alias mysqladmin '/usr/local/mysql/bin/mysqladmin'
```

2.6.6 Outras Notas Unix

2.6.6.1 Notas HP-UX para distribuições binárias

Alguma das distribuições binárias do MySQL para HP-UX é distribuída como um arquivo depot da HP e como um arquivo tar. Para usar o arquivo depot você deve estar executando pelo menos o HP-UX 10.x para ter acesso às ferramentas de arquivos depot da HP.

A versão HP do MySQL foi compilada em um servidor HP 9000/8xx sob HP-UX 10.20, usando MIT-pthreads. Sob esta configuração o MySQL funciona bem. O MySQL Versão 3.22.26 e mais novas também podem ser construídas com o pacote thread nativo da HP.

Outras configurações que podem funcionar:

- HP 9000/7xx executando HP-UX 10.20+
- HP 9000/8xx executando HP-UX 10.30

As seguintes configurações definitivamente não funcionarão:

- HP 9000/7xx ou 8xx executando HP-UX 10.x where $x < 2$
- HP 9000/7xx ou 8xx executando HP-UX 9.x

Para instalar a distribuição, utilize um dos comandos abaixo, onde `/path/to/depot` é o caminho completo do arquivo depot:

- Para instalar tudo, incluindo o servidor, cliente e ferramentas de desenvolvimento:

```
shell> /usr/sbin/swinstall -s /path/to/depot mysql.full
```

- Para instalar somente o servidor:

```
shell> /usr/sbin/swinstall -s /path/to/depot mysql.server
```

- Para instalar somente o pacote cliente:

```
shell> /usr/sbin/swinstall -s /path/to/depot mysql.client
```

- Para instalar somente as ferramentas de desenvolvimento:

```
shell> /usr/sbin/swinstall -s /path/to/depot mysql.developer
```

O depot copia os binários e bibliotecas em `/opt/mysql` e dados em `/var/opt/mysql`. O depot também cria as entradas apropriadas em `/etc/init.d` e `/etc/rc2.d` para iniciar o servidor automaticamente na hora do boot. Obviamente, para instalar o usuário deve ser o `root`.

Para instalar a distribuição HP-UX tar.gz, você deve ter uma cópia do GNU tar.

2.6.6.2 Notas HP-UX Versão 10.20

Existem alguns pequenos problemas quando compilamos o MySQL no HP-UX. Nós recomendamos que você use o gcc no lugar do compilador nativo do HP-UX, porque o gcc produz um código melhor!

Nós recomendamos o uso do gcc 2.95 no HP-UX. Não utilize opções de alta otimização (como `-O6`) já que isto pode não ser seguro no HP-UX.

Perceba que as MIT-pthreads não podem ser compiladas com o compilador HP-UX porque ele não pode compilar arquivos `.S` (assembler).

A seguinte linha de configuração deve funcionar:

```
CFLAGS="-DHPUX -I/opt/dce/include" CXXFLAGS="-DHPUX -I/opt/dce/include -felide-cons
```

Se você mesmo está compilando gcc 2.95, você não deve ligá-lo com as bibliotecas DCE (`libdce.a` ou `libcma.a`) se você deseja compilar o MySQL com MIT-pthreads. Se você misturar os pacotes DCE e MIT-pthreads, você irá obter um `mysqld` com o qual não poderá conectar. Remova as bibliotecas DCE enquanto você compila gcc 2.95!

2.6.6.3 Notas HP-UX Versão 11.x

Para HP-UX Versão 11.x nós recomendamos o MySQL Versão 3.23.15 ou posterior.

Por causa de alguns bugs críticos nas bibliotecas padrão do HP-UX, você deve instalar as seguintes correções antes de tentar executar o MySQL no HP-UX 11.0:

```
PHKL_22840 Streams cumulative
PHNE_22397 ARPA cumulative
```

Isto irá resolver um problema que tem como retorno EWOLDBLOCK de `recv()` e `EBADF` de `accept()` em aplicações threads.

Se você estiver usando `gcc 2.95.1` em um sistema HP-UX 11.x sem correções, você obterá o erro:

```
In file included from /usr/include/unistd.h:11,
                 from ../include/global.h:125,
                 from mysql_priv.h:15,
                 from item.cc:19:
/usr/include/sys/unistd.h:184: declaration of C function ...
/usr/include/sys/pthread.h:440: previous declaration ...
In file included from item.h:306,
                 from mysql_priv.h:158,
                 from item.cc:19:
```

O problema é que o HP-UX não define consistentemente a `pthread_atfork()`. Ela tem protótipos conflitantes em `'/usr/include/sys/unistd.h':184` e `'/usr/include/sys/pthread.h':440` (detalhes abaixo).

Uma solução é copiar `'/usr/include/sys/unistd.h'` em `'mysql/include'` e editar `'unistd.h'` alterando-o para coincidir com a definição em `'pthread.h'`. Aqui está o diff:

```
183,184c183,184
<     extern int pthread_atfork(void (*prepare)(), void (*parent)(),
<                               void (*child)());
---
>     extern int pthread_atfork(void (*prepare)(void), void (*parent)(void),
>                               void (*child)(void));
```

Depois disto, a seguinte linha configure deve funcionar:

```
CFLAGS="-fomit-frame-pointer -O3 -fpic" CXX=gcc CXXFLAGS="--felide-constructors -fno-
```

Segue algumas informações que um usuário do HP-UX Versão 11.x nos enviou sobre compilação do MySQL com o compilador HP-UX:

```
Environment:
  proper compilers.
    setenv CC cc
    setenv CXX aCC
  flags
    setenv CFLAGS -D_REENTRANT
    setenv CXXFLAGS -D_REENTRANT
    setenv CPPFLAGS -D_REENTRANT
% aCC -V
aCC: HP ANSI C++ B3910B X.03.14.06
% cc -V /tmp/empty.c
```

```

cpp.ansi: HP92453-01 A.11.02.00 HP C Preprocessor (ANSI)
ccom: HP92453-01 A.11.01.00 HP C Compiler
cc: "/tmp/empty.c", line 1: warning 501: Empty source file.

```

```

configuration:
./configure --with-pthread \
--prefix=/source-control/mysql \
--with-named-thread-libs=-lpthread \
--with-low-memory

```

added '#define _CTYPE_INCLUDED' to include/m_ctype.h. This symbol is the one defined in HP's /usr/include/ctype.h:

```

/* Don't include std ctype.h when this is included */
#define _CTYPE_H
#define __CTYPE_INCLUDED
#define _CTYPE_INCLUDED
#define _CTYPE_USING /* Don't put names in global namespace. */

```

- Eu tive que usar a opção de tempo de compilação `-D_REENTRANT` para que o compilador reconheça o protótipo para `localtime_r`. Uma forma alternativa é fornecer o protótipo para `localtime_r`. Mas eu queria pegar outros erros sem precisar executá-los. Não tinha certeza onde eu precisaria dele, portanto eu o adicionei a todas as opções.
- As opções de otimização utilizadas pelo MySQL (`-O3`) não é reconhecida pelos compiladores HP. Eu não alterei as opções.

Se você obter o seguinte erro do `configure`

```

checking for cc option to accept ANSI C... no
configure: error: MySQL requires a ANSI C compiler (and a C++ compiler). Try gcc. S

```

Confira se você não tem o caminho para o compilador K&R antes do caminho para o compilador C e C++ do HP-UX.

2.6.6.4 Notas IBM-AIX

Deteção automática de `xlc` está faltando no `Autoconf`, portando um comando `configure` deste tipo é necessário quando estiver compilando o MySQL (Este exemplo usa o compilador IBM):

```

export CC="xlc_r -ma -O3 -qstrict -qoptimize=3 -qmaxmem=8192 "
export CXX="xlc_r -ma -O3 -qstrict -qoptimize=3 -qmaxmem=8192"
export CFLAGS="-I /usr/local/include"
export LDFLAGS="-L /usr/local/lib"
export CPPFLAGS=$CFLAGS
export CXXFLAGS=$CFLAGS

```

```

./configure --prefix=/usr/local \
--localstatedir=/var/mysql \
--sysconfdir=/etc/mysql \
--sbindir='/usr/local/bin' \
--libexecdir='/usr/local/bin' \

```

```
--enable-thread-safe-client \
--enable-large-files
```

Acima estão as opções usadas para compilar a distribuição MySQL que pode ser encontrada em <http://www-frec.bull.com/>.

Se você alterar o -O3 para -O2 na linha de configuração acima, você também deve remover a opção -qstrict (isto é uma limitação no compilador C da IBM).

Se você estiver usando gcc ou egcs para compilar o MySQL, você **DEVE** usar a opção -fno-exceptions, já que o manipulador de exceções no gcc/egcs não é seguro para threads! (Isto foi testado com egcs 1.1). Existem também alguns problemas conhecidos com o assembler da IBM que pode gerar código errado quando usado com gcc.

Nós recomendamos a seguinte linha do configure com egcs e gcc 2.95 no AIX:

```
CC="gcc -pipe -mcpu=power -Wa,-many" \
CXX="gcc -pipe -mcpu=power -Wa,-many" \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql --with-low-memory
```

O -Wa,-many é necessário para o compilador ser bem sucedido. IBM está ciente deste problema mas não está com pressa de corrigi-lo devido ao fato do problema poder ser contornado. Nós não sabemos se o -fno-exceptions é necessário com gcc 2.9.5, mas como o MySQL não utiliza exceções e a opção acima gera código mais rápido, recomendamos que você sempre use esta opção com o egcs/gcc.

Se você tiver algum problema com código assembler tente alterar o -mcpu=xxx para o seu processador. Normalmente power2, power ou powerpc podem ser usados, de uma maneira alternativa você pode precisar usar 604 ou 604e. Não tenho certeza mas acredito que usar "power" deve satisfazer a maioria dos casos, mesmo em uma máquina power2.

Se você não sabe qual é o seu processador, utilize o comando "uname -m", isto irá fornecer a você uma string que parece com "000514676700", com um formato de xyyyyyyymmss onde xx e ss são sempre 0s, yyyyyy é o ID único do sistema e mm é o ID da CPU Planar. Uma tabela destes valores podem ser encontrados em http://www.rs6000.ibm.com/doc_link/en_US/a_doc_lib/cmds/aixcmds5/uname.htm. Isto irá lhe fornecer um tipo de máquina e um modelo de máquina que você pode usar para determinar que tipo de cpu você tem.

Se você tiver problemas com sinais (MySQL finaliza sem notificação sob alta carga) você pode ter encontrado um bug de SO com threads e sinais. Neste caso você pode dizer ao MySQL para não usar sinais configurando-o com:

```
shell> CFLAGS=-DDONT_USE_THR_ALARM CXX=gcc \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti -DDONT_USE_THR_ALARM" \
./configure --prefix=/usr/local/mysql --with-debug --with-low-memory
```

Isto não afeta a performance do MySQL, mas tem o efeito colateral que você não pode matar clientes que estão "dormindo" em uma conexão com mysqladmin kill ou mysqladmin shutdown. Neste caso, o cliente morrerá quando ele chegar no próximo comando.

Em algumas versões do AIX, ligando com libbind.a faz o getservbyname descarregar core. Isto é erro no AIX e deve ser relatado para a IBM.

Para o AIX 4.2.1 e gcc você tem que fazer as seguintes alterações.

Depois de configurar, edite o 'config.h' e 'include/my_config.h' e altere a linha que diz

```
#define HAVE_SNPRINTF 1
para
#undef HAVE_SNPRINTF
```

E finalmente, no 'mysqld.cc' você precisa adicionar um protótipo para initgroups.

```
#ifdef _AIX41
extern "C" int initgroups(const char *,int);
#endif
```

2.6.6.5 Notas SunOS 4

No SunOS 4, é necessário a MIT-pthreads para compilar o MySQL, o que significa que você precisa do GNU make.

Alguns sistemas SunOS 4 tem problemas com bibliotecas dinâmicas e libtool. Você pode usar a seguinte linha do configure para evitar este problema:

```
shell> ./configure --disable-shared --with-mysqld-ldflags=-all-static
```

Quando compilando readline, você pode obter alguns avisos sobre definições duplicadas que podem ser ignoradas.

Ao compilar o mysqld, vão existir alguns alertas sobre implicit declaration of function que também podem ser ignoradas.

2.6.6.6 Notas Alpha-DEC-UNIX (Tru64)

Se você está usando o egcs 1.1.2 no Digital Unix, você atualizar par o gcc 2.95.2, já que o egcs no DEC tem vários erros graves !

Quando compilando programas com threads no Digital Unix, a documentação recomenda usar a opção -pthread para cc e cxx e as bibliotecas -lmach -lexc (em adição para -lpthread). Você deve executar o configure parecido com isto:

```
CC="cc -pthread" CXX="cxx -pthread -O" \
./configure --with-named-thread-libs="-lpthread -lmach -lexc -lc"
```

Quando compilando o mysqld, você deve ver alguns avisos como estes:

```
mysqld.cc: In function void handle_connections():
mysqld.cc:626: passing long unsigned int *' as argument 3 of
accept(int,sockaddr *, int *)'
```

Você pode ignorar estes alertas com segurança. Eles ocorrem porque o configure só pode detectar erros e não alertas.

Se você inicia o servidor diretamente da linha de comando, você pode ter problemas com a finalização do servidor ao sair (log out). (Quando você sai, seu processo superior recebe um sinal SIGHUP.) Se isto acontecer, tente iniciar o servidor desta forma:

```
shell> nohup mysqld [options] &
```

nohup faz com que o comando que o segue ignore qualquer sinal SIGHUP enviado pelo terminal. De forma alternativa, inicie o servidor executando safe_mysqld, o qual invoca o mysqld usando nohup por você. See [<undefined> \[safe_mysqld\]](#), page [<undefined>](#).

Se você tiver problemas quando compilar mysys/get-opt.c, apenas remova a linha #define _NO_PROTO do inicio do arquivo!

Se você estiver utilizando o compilador CC da Compaq, a seguinte linha de configuração deverá funcionar:

```
CC="cc -pthread"
CFLAGS="-O4 -ansi_alias -ansi_args -fast -inline speed all -arch host"
CXX="cxx -pthread"
CXXFLAGS="-O4 -ansi_alias -ansi_args -fast -inline speed all -arch host"
export CC CFLAGS CXX CXXFLAGS
./configure \
--prefix=/usr/local/mysql \
--with-low-memory \
--enable-large-files \
--enable-shared=yes \
--with-named-thread-libs="-lpthread -lmach -lexc -lc"
gnumake
```

Se você tiver problemas com a libtool, ao compilar com bibliotecas compartilhadas como no exemplo acima, quando estiver ligando ao `mysqld`, você deve conseguir contornar este problema usando:

```
cd mysql
/bin/sh ../libtool --mode=link cxx -pthread -O3 -DDEBUG_OFF \
-O4 -ansi_alias -ansi_args -fast -inline speed \
-speculate all \ -arch host -DUNDEF_HAVE_GETHOSTBYNAME_R \
-o mysql mysql.o readline.o sql_string.o completion_hash.o \
../readline/libreadline.a -lcurses \
../libmysql/.libs/libmysqlclient.so -lm
cd ..
gnumake
gnumake install
scripts/mysql_install_db
```

2.6.6.7 Notas Alpha-DEC-OSF1

Se você tiver problemas com compilação e tem o DEC CC e o gcc instalados, tente executar o `configure` desta forma:

```
CC=cc CFLAGS=-O CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql
```

Se você tiver problemas com o arquivo `'c_asm.h'`, você pode criar e usar um arquivo `'c_asm.h'` 'burro' com:

```
touch include/c_asm.h
CC=gcc CFLAGS=-I./include \
CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql
```

Perceba que os seguintes problemas com o programa `ld` pode ser corrigido fazendo o download do último kit de atualização da DEC (Compaq) de <http://ftp.support.compaq.com/public/unix/>.

Com o OSF1 V4.0D e o compilador "DEC C V5.6-071 no Digital Unix V4.0 (Rev. 878)" o compilador tem alguns comportamentos estranhos (símbolos `asm` indefinidos). `/bin/ld` também aparece estar quebrado (problemas com erros `_exit undefined` ocorrendo ao ligar

no `mysqld`). Neste sistema, temos compilado o MySQL com a seguinte linha `configure`, depois de substituir `/bin/ld` com a versão do OSF 4.0C:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
```

Com o compilador da Digital "C++ V6.1-029", o seguinte deve funcionar:

```
CC=cc -pthread
CFLAGS=-O4 -ansi_alias -ansi_args -fast -inline speed -speculate all -arch host
CXX=cxx -pthread
CXXFLAGS=-O4 -ansi_alias -ansi_args -fast -inline speed -speculate all -arch host -
export CC CFLAGS CXX CXXFLAGS
./configure --prefix=/usr/mysql/mysql --with-mysqld-ldflags=-all-static --disable-s
```

Em algumas versões do OSF1, a função `alloca()` está quebrada. Corrija isto removendo a linha no `'config.h'` que define `'HAVE_ALLOCA'`.

A função `alloca()` pode também ter um protótipo incorreto em `/usr/include/alloca.h`. O alerta resultante deste erro pode ser ignorado.

`configure` irá usar a seguinte biblioteca `thread` automaticamente: `--with-named-thread-libs="-lpthread -lmach -lexc -lc"`.

Quando usar o `gcc`, você também pode tentar executar `configure` desta forma:

```
shell> CFLAGS=-D_PTHREAD_USE_D4 CXX=gcc CXXFLAGS=-O3 ./configure ....
```

Se você tiver problemas com sinais (MySQL finalizar inesperadamente sobre alta carga), você pode ter encontrado um erro com threads e sinais no SO. Neste caso você pode dizer ao MySQL para não usar sinais configurando-o com:

```
shell> CFLAGS=-DDONT_USE_THR_ALARM \
CXXFLAGS=-DDONT_USE_THR_ALARM \
./configure ...
```

Isto não afeta a performance do MySQL, mas tem efeitos colaterais que não permitem finalizar clientes que estão "dormindo" em uma conexão com `mysqladmin kill` ou `mysqladmin shutdown`. Neste caso o cliente irá morrer quando ele receber o próximo comando.

Com `gcc 2.95.2`, você provavelmente encontrará o seguinte erro de compilação:

```
sql_acl.cc:1456: Internal compiler error in 'scan_region', at except.c:2566
Please submit a full bug report.
```

Para corrigir isto você deve alterar para o diretório `sql` e fazer um "corta e cola" da última linha `gcc`, mas altere `-O3` para `-O0` (ou adicione `-O0` imediatamente depois de `gcc` se você não tiver algumas opção `-O` na sua linha de compilação.) Depois disto feito você deve apenas voltar ao diretório superior e executar `make` novamente.

2.6.6.8 Notas SGI Irix

Se você estiver usando Irix Versão 6.5.3 ou mais novo, o `mysqld` só irá conseguir criar threads se você executá-lo como um usuário com privilégios de `CAP_SCHED_MGT` (como `root`) ou dar ao servidor `mysqld` este privilégio com o seguinte comando shell:

```
shell> chcap "CAP_SCHED_MGT+epi" /opt/mysql/libexec/mysqld
```

Você pode precisar indefinir algumas coisas em `'config.h'` depois de executar `configure` e antes de compilar.

Em algumas implementações Irix, a função `alloca()` está quebrada. Se o servidor `mysqld` morrer em alguma instrução `SELECT`, remova as linhas de `'config.h'` que definem `HAVE_ALLOC` e `HAVE_ALLOC_H`. Se `mysqladmin create` não funciona, remova a linha do `'config.h'` que define `HAVE_READDIR_R`. Você também deve precisar remover a linha `HAVE_TERM_H`.

A SGI recomenda que você instale todos os patches desta página: <http://support.sgi.com/surfzone/patches/>
No mínimo, você deve instalar o último rollup do kernel, o último rollup `rld`, e o último rollup `libc`.

Definitivamente você precisará de todos patches POSIX nesta página, para suporte `pthread`:

http://support.sgi.com/surfzone/patches/patchset/6.2_posix.rps.html

Se você obter o seguinte erro quando estiver compilando o `'mysql.cc'`:

```
"/usr/include/curses.h", line 82: error(1084): invalid combination of type
```

Digite o seguinte no diretório topo da sua árvore fonte do MySQL:

```
shell> extra/replace bool curses_bool < /usr/include/curses.h > include/curses.h
shell> make
```

Existem relatos de problemas com organização de threads. Se somente uma thread estiver executando, o sistema fica lento. Pode se evitar isto iniciando outro cliente. Isto pode acarretar num crescimento de 2 para 10 vezes na velocidade de execução para a outra thread. Isto é um problema não compreendido com threads Irix; você deve improvisar para encontrar soluções até que isto seja resolvido.

Se você estiver compilando com `gcc`, você pode usar o seguinte comando `configure`:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql --enable-thread-safe-client --with-named-thre
```

No Irix 6.5.11 com Irix C nativo e compiladores C++ ver. 7.3.1.2, o seguinte irá funcionar

```
CC=cc CXX=CC CFLAGS='-O3 -n32 -TARG:platform=IP22 -I/usr/local/include \
-L/usr/local/lib' CXXFLAGS='-O3 -n32 -TARG:platform=IP22 \
-I/usr/local/include -L/usr/local/lib' ./configure --prefix=/usr/local/mysql \
--with-berkeley-db --with-innodb \
--with-libwrap=/usr/local --with-named-curses-libs=/usr/local/lib/libncurses.a
```

2.6.6.9 Notas SCO

A versão atual foi testado somente nos sistemas “sco3.2v5.0.4” e “sco3.2v5.0.5”. A versão para o “sco 3.2v4.2” também tem tido muito progresso.

Até o momento o compilador recomendado no OpenServer é o `gcc 2.95.2`. Com isto você deve estar apto a compilar o MySQL apenas com:

```
CC=gcc CXX=gcc ./configure ... (opções)
```

1. Para o OpenServer 5.0.X você precisa usar o GDS no Skunkware 95 (95q4c). Isto é necessário porque o GNU `gcc 2.7.2` no Skunkware 97 não possui o GNU `as`. Você também pode usar o `egcs 1.1.2` ou mais novo <http://www.egcs.com/>. Se você estiver usando `egcs 1.1.2` você deve executar o seguinte comando:

```
shell> cp -p /usr/include/pthread/stdtypes.h /usr/local/lib/gcc-lib/i386-pc-sco3
```

2. Você precisa do GCC versão 2.5.x para este produto e do sistema de desenvolvimento. Eles são necessários nesta versão do SCO Unix. Você não pode usar apenas o sistema GCC Dev.
3. Você deve obter o pacote FSU Pthreads e instalá-lo primeiro. Pode ser obtido em http://www.cs.wustl.edu/~schmidt/ACE_wrappers/FSU-threads.tar.gz. Você pode também obter um pacote precompilado de <http://www.mysql.com/Downloads/SCO/FSU-threads>.
4. FSU Pthreads pode ser compilado com SCO Unix 4.2 com tcpip, ou OpenServer 3.0 ou OpenDesktop 3.0 (OS 3.0 ODT 3.0), com o Sistema de Desenvolvimento da SCO instalado usando uma boa versão do GCC 2.5.x ODT ou OS 3.0, no qual você necessitará de uma boa versão do GCC 2.5.x. Existem vários problemas sem uma boa versão. Esta versão do produto necessita do sistema de Desenvolvimento SCO Unix. Sem ele, você estará perdendo as bibliotecas e o editor de ligação necessário.
5. Para construir a FSU Pthreads no seu sistema, faça o seguinte:
 - a. Execute `./configure` no diretório `'threads/src'` e selecione a opção SCO OpenServer. Este comando copia `'Makefile.SCO5'` para `'Makefile'`.
 - b. Execute `make`.
 - c. Para instalar no diretório padrão `'/usr/include'`, use o usuário root, depois mude para o diretório `'thread/src'` e execute `make install`
6. Lembre de usar o GNU `make` quando estiver construindo o MySQL.
7. Se você não iniciou o `safe_mysql` como root, você provavelmente só irá obter, por padrão, os 110 arquivos abertos por processo. O `mysqld` irá gravar uma nota sobre isto no arquivo `log`.
8. Com o SCO 3.2V5.0.5, você deve usar o FSU Pthreads versão 3.5c ou mais nova. Você deve também usar o `gcc 2.95.2` ou mais novo.

O seguinte comando `configure` deve funcionar:

```
shell> ./configure --prefix=/usr/local/mysql --disable-shared
```

9. Com SCO 3.2V4.2, você deve usar FSU Pthreads versão 3.5c ou mais nova. O seguinte comando `configure` deve funcionar:

```
shell> CFLAGS="-D_XOPEN_XPG4" CXX=gcc CXXFLAGS="-D_XOPEN_XPG4" \
./configure \
--prefix=/usr/local/mysql \
--with-named-thread-libs="-lgthreads -lsocket -lgen -lgthreads" \
--with-named-curses-libs="-lcurses"
```

Você pode ter alguns problemas com alguns arquivos de inclusão. Neste caso, você pode encontrar novos arquivos de inclusão específicos do SCO em <http://www.mysql.com/Downloads/SCO/SCO-3.2v4.2-includes.tar.gz>. Você deve descompactar este arquivo no diretório `'include'` da sua árvore fonte do MySQL.

Notas de desenvolvimento SCO:

- O MySQL deve detectar automaticamente FSU Pthreads e ligar o `mysqld` com `-lgthreads -lsocket -lgthreads`.
- As bibliotecas de desenvolvimento SCO são re-entrantes nas FSU Pthreads. A SCO diz que suas bibliotecas de funções são re-entrantes, então elas devem ser re-entrantes com as FSU-Pthreads. FSU Pthreads no OpenServer tentam usar o esquema SCO para criar bibliotecas re-entrantes.

- FSU Pthreads (ao menos a versão em <http://www.mysql.com>) vem ligada com GNU malloc. Se você encontrar problemas com uso de memória, tenha certeza que o 'gmalloc.o' esteja incluído em 'libgthreads.a' e 'libgthreads.so'.
- Na FSU Pthreads, as seguintes chamadas de sistema são compatíveis com pthreads: read(), write(), getmsg(), connect(), accept(), select() e wait().
- O CSSA-2001-SCO.35.2 (O patch é listado de costume como patch de segurança erg711905-dscr_remap ver 2.0.0) quebra FSU threads e deixa o mysqld instável. Você deve remove-lo se você deseja executar o mysqld em uma máquina OpenServer 5.0.6.

Se você deseja instalar o DBI no SCO, você deve editar o 'Makefile' em DBI-xxx e cada subdiretório.

Note que o exemplo abaixo considera o gcc 2.95.2 ou mais novo:

```

OLD:                                NEW:
CC = cc                             CC = gcc
CCCDLFLAGS = -KPIC -W1,-Bexport     CCCDLFLAGS = -fpic
CCDLFLAGS = -wl,-Bexport           CCDLFLAGS =

LD = ld                              LD = gcc -G -fpic
LDDLFLAGS = -G -L/usr/local/lib     LDDLFLAGS = -L/usr/local/lib
LDFLAGS = -belf -L/usr/local/lib    LDFLAGS = -L/usr/local/lib

LD = ld                              LD = gcc -G -fpic
OPTIMISE = -Od                       OPTIMISE = -O1

OLD:
CCCFLAGS = -belf -dy -w0 -U M_XENIX -DPERL_SCO5 -I/usr/local/include

NEW:
CCFLAGS = -U M_XENIX -DPERL_SCO5 -I/usr/local/include

```

Isto é porque o carregador dinâmico Perl não irá carregar os módulos DBI se elas foram compiladas com `icc` ou `cc`.

Perl trabalha melhor quando compilado com `cc`.

2.6.6.10 Notas SCO Unixware Version 7.0

Você deve usar uma versão de MySQL pelo menos tão recente quando a Versão 3.22.13 porque esta versão corrige alguns problemas de portabilidade sob o Unixware.

Nós temos compilado o MySQL com o seguinte comando `configure` no UnixWare Versão 7.0.1:

```
CC=cc CXX=CC ./configure --prefix=/usr/local/mysql
```

Se você deseja usar o `gcc`, deverá ser usado o `gcc 2.95.2` ou mais novo.

2.6.7 Notas OS/2

O MySQL usa poucos arquivos aberto. Por isto, você deve adicionar uma linha parecida com a abaixo em seu arquivo 'CONFIG.SYS':

```
SET EMXOPT=-c -n -h1024
```

Se você não fizer isto, provavelmente vai ter o seguinte erro:

```
File 'xxxx' not found (Errcode: 24)
```

Quando usar o MySQL com OS/2 Warp 3, o FixPack 29 ou superior é necessário. Com OS/2 Warp 4, FixPack 4 ou acima é necessário. Isto é uma exigência da biblioteca Pthreads. O MySQL deve estar instalado em uma partição que suporta nomes longos de arquivos como no HPFS, FAT32, etc.

O script 'INSTALL.COM' deve ser executado pelo próprio 'CMD.EXE' do OS/2 e opde não funcionar com shells substitutas como o '4OS2.EXE'.

O script 'scripts/mysql-install-db' foi renomeado. Agora ele é chamado 'install.cmd' e é um script REXX, que irá atualizar as configurações padrões de segurança do MySQL e criar os ícones na WorkPlace Shell para o MySQL.

Suporte a módulos dinâmicos é compilado mas não totalmente testado. Módulos dinâmicos devem ser compilados usando a biblioteca run-time Pthreads.

```
gcc -Zdll -Zmt -Zcrt.dll=pthrdrt1 -I../include -I../regex -I.. \
-o example udf_example.cc -L../lib -lmysqlclient udf_example.def
mv example.dll example.udf
```

Nota: Devido a limitações no OS/2, o nome do módulo UDF não deve exceder 8 caracteres. Módulos são armazenados no diretório '/mysql2/udf'; o script safe-mysqld.cmd irá colocar este diretório na variável de ambiente BEGINLIBPATH. Quando usando módulos UDF, extensões específicas são ignoradas — considere-se que seja '.udf'. Por exemplo, no Unix, o módulo compartilhado deve ser nomeado 'example.so' e você deve carregar uma função dele desta forma:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME "example.so";
```

No OS/2, o módulo deve ter o nome de 'example.udf', mas você não deve especificar a extensão do módulo:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME "example";
```

2.6.8 Notas BeOS

Realmente temos interesse que o MySQL funcione no BeOS, mas infelizmente não temos ninguém que saiba BeOS ou tenha tempo para fazer uma versão.

Nós estamos interessados em encontrar alguém que faça uma versão, e iremos ajudar com qualquer questão técnica que possa ter durante o desenvolvimento.

Nós já falamos com alguns desenvolvedores BeOS que disseram que o MySQL está 80% portado para o BeOS, mas nós não sabemos qual a situação no momento.

2.6.9 Notas Novell Netware

Estamos realmente interessados em deixar o MySQL funcionando no NetWARE, mas infelizmente não temos ninguém que saiba Netware ou tenha tempo para fazer uma versão.

Nós estamos interessados em encontrar alguém que faça a versão, e iremos ajudá-lo com qualquer questão técnica que possa ter durante o desenvolvimento.

3 Introdução ao MySQL: Um Tutorial MySQL

Este capítulo fornece um tutorial de introdução ao MySQL demonstrando como usar o programa cliente `mysql` para criar e usar um banco de dados simples. `mysql` (algumas vezes apresentado como o “terminal monitor” ou apenas “monitor”) é um programa interativo que lhe permite conectar a um servidor MySQL, executar consultas e visualizar os resultados. `mysql` pode também ser executado em modo batch: você coloca suas consultas em um arquivo, depois diz ao `mysql` para executar o conteúdo do arquivo. Cobrimos aqui ambas as formas de utilizar o `mysql`.

Para ver uma lista de opções conhecidas pelo `mysql`, chame-o com a opção `--help`:

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

Este capítulo presume que o `mysql` está instalado na sua máquina e que um servidor MySQL está disponível para quem puder conectar. Se isto não for verdade, contate seu administrador MySQL. (Se *you* é o administrador, você precisará consultar outras seções deste manual.)

Este capítulo descreve todo o processo de configuração e uso de um banco de dados. Se você estiver interessado em apenas acessar um banco de dados já existente, poderá pular as seções que descrevem como criar o banco de dados e suas respectivas tabelas.

Como este capítulo é um tutorial, vários detalhes são necessariamente deixados de fora. Consulte as seções relevantes do manual para mais informações sobre os tópicos cobertos aqui.

3.1 Conectando e desconectando do servidor

Para conectar ao servidor, normalmente você precisará fornecer um nome de usuário quando o `mysql` for chamado e, na maioria dos casos, uma senha. Se o servidor executa em uma máquina diferente de onde você está, você também precisará especificar um nome de máquina. Contate seu administrador para saber quais parâmetros de conexão você deve usar para conectar (isto é, qual máquina, usuário e senha usar). Uma vez que você saiba quais os parâmetros corretos, você deve estar pronto para conectar da seguinte forma:

```
shell> mysql -h servidor -u usuario -p
Enter password: *****
```

Os asteriscos (*****) representam sua senha; digite-a quando o `mysql` mostrar o prompt `Enter password:`.

Se isto funcionar, você deve ver algumas informações iniciais seguidas de um prompt `mysql>`

```
shell> mysql -h servidor -u usuario -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

Type 'help' for help.
```

```
mysql>
```

O prompt lhe diz que o `mysql` está pronto para que você digite os comandos.

Algumas instalações MySQL permitem aos usuários de se conectarem como usuários anônimos ao servidor executando na máquina local. Se isto é o caso na sua máquina, você deve conseguir conectar ao servidor chamando o `mysql` sem qualquer opção:

```
shell> mysql
```

Depois de você conectar com sucesso, você pode desconectar a qualquer hora digitando `QUIT` no prompt `mysql>`:

```
mysql> QUIT
Bye
```

Você também pode desconectar pressionando Control-D.

A maioria dos exemplos nas seções seguintes assumem que você já está conectado ao servidor. Isto é indicado pelo prompt `mysql>`.

3.2 Fazendo Consultas

Tenha certeza que você está conectado ao servidor, como discutido na seção anterior. Isto feito, não será selecionado nenhum banco de dados para trabalhar, mas não tem problemas. Neste momento, é mais importante saber um pouco sobre como fazer consultas do que já criar tabelas, carregar dados para elas, e recuperar dados delas. Esta seção descreve os princípios básicos da entrada de comandos, usando diversas consultas você pode tentar se familiarizar com o funcionamento do `mysql`.

Aqui está um comando simples que solicita ao servidor seu número de versão e a data atual. Digite-o como visto abaixo seguindo o prompt `mysql>` e digite a tecla RETURN:

```
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 várias coisas sobre o `mysql`:

- Um comando normalmente consiste de uma instrução SQL seguida por um ponto e vírgula. (Existem algumas exceções onde um ponto e vírgula não é necessário. `QUIT` mencionado anteriormente, é um deles. Sabemos de outros mais tarde.)
- Quando você emite um comando, o `mysql` o envia para o servidor para execução e mostra os resultados, depois imprime o turo `mysql>` para indicar que está pronto para outro comando.
- O `mysql` mostra a saída da consulta como uma tabela (linhas e colunas). A primeira linha contém rótulos para as colunas. As linhas seguintes são o resultado da consulta. Normalmente, rótulos de colunas são os nomes das colunas que você busca das tabelas do banco de dados. Se você está recuperando o valor de uma expressão no lugar de uma coluna de tabela (como no exemplo já visto), o `mysql` rotula a coluna usando a própria expressão.
- O `mysql` mostra quantas linhas foram retornadas e quanto tempo a consulta levou para executar, o que lhe dá uma vaga idéia da performance do servidor. Estes valores

são impreciso porque eles representam tempo de relógio (Não tempo de CPU ou de máquina), e porque eles são afetados pelos fatores como a carga do servidor e latência de rede. (Para resumir, a linha “rows in set” não é mostrada nos exemplos seguintes deste capítulo.)

Palavras Chave podem ser entradas em qualquer caso de letra. As seguintes consultas são equivalentes:

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

Aqui está outra consulta. Ela demonstra que você pode usar o `mysql` como uma calculadora simples:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+-----+-----+
| SIN(PI()/4) | (4+1)*5 |
+-----+-----+
| 0.707107 | 25 |
+-----+-----+
```

Os comandos mostrados até agora têm sido instruções relativamente pequenas, de uma linha. Você pode também entrar com múltiplas instruções em uma única linha. Basta finalizar 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 não necessita estar todo em uma única linha, então comandos extensos que necessitam de várias linhas não são um problema. O `mysql` determina onde sua instrução termina através do ponto e vírgula terminador, e não pelo final da linha de entrada. (Em outras palavras, o `mysql` aceita entradas de livre formato: Ele coleta linhas de entrada mas não as executa até chegar o ponto e vírgula.)

Aqui está uma instrução simples usando múltiplas linhas:

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

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

Neste exemplo, note como o prompt altera de `mysql>` para `->` depois de você entrar a primeira linha de uma consulta com múltiplas linhas. Isto é como o `mysql` indica que ainda não achou uma instrução completa e está esperando pelo resto. O prompt é seu amigo, porque ele fornece um retorno valioso. Se você usa este retorno, você sempre estará ciente do que o `mysql` está esperando.

Se você decidir que não deseja executar um comando que está no meio do processo de entrada, cancele-o digitando `\c`:

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

Note o prompt aqui também. Ele troca para o `mysql>` depois de você digitar `\c`, fornecendo retorno para indicar que o `mysql` está pronto para um novo comando.

A seguinte tabela mostra cada dos prompts que você pode ver e resume o que ele significa sobre o estado em que o `mysql` se encontra:

Prompt	Significado
<code>mysql></code>	Pronto para novo comando.
<code>-></code>	Esperando pela próxima linha de comando com múltiplas linhas.
<code>'></code>	Esperando pela próxima linha, coletando uma string que comece com uma aspas simples ('').
<code>"></code>	Esperando pela próxima linha, coletando uma string que comece com aspas duplas (").

É muito comum instruções multi-linhas ocorrem por acidente quando você pretende publicar um comando em uma única linha, mas esquece o ponto e vírgula terminador. Neste caso, o `mysql` espera por mais entrada:

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

Se isto ocorrer com você (acha que entrou uma instrução mas a única resposta é um prompt `->`), o mais provável é que o `mysql` está esperando pelo ponto e vírgula. Se você não perceber o que o prompt está lhe dizendo, você pode parar por um tempo antes de entender o que precisa fazer. Entre com um ponto e vírgula para completar a instrução, e o `mysql` irá executá-la:

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

O prompt `'>` e `">` ocorrem durante a coleta de strings. No MySQL, você pode escrever strings utilizando os caracteres `'` ou `"` (por exemplo, `'hello'` ou `"goodbye"`), e o `mysql` permite a entrada de strings que consomem múltiplas linhas. Quando você ver um prompt `'>` ou `">`, significa que você digitou uma linha contendo uma string que começa com um caracter de aspas `'` ou `"` mas ainda não entrou com a aspas que termina a string. Isto

é bom se você realmente está entrando com uma string com múltiplas linhas, mas qual é a probabilidade disto acontecer ? Não muita. Geralmente, os prompts `'>` e `">` indicam que você, por algum descuido, esqueceu algum caracter de aspas. Por exemplo:

```
mysql> SELECT * FROM minha_tabela WHERE nome = "Smith AND idade < 30;
">
```

Se você entrar esta sentença `SELECT`, apertar `ENTER` e esperar pelo resultado, nada irá acontecer. Em vez de se perguntar o porquê desta query demorar tanto tempo, perceba a pista fornecida pelo prompt `">`. Ele lhe diz que o `mysql` espera pelo resto de uma string não terminada. (Você ve o erro na declaração? Falta a segunda aspas na string `"Smith`.)

O que fazer neste ponto ? A coisa mais simples é cancelar o comando. Entretanto, você não pode simplesmente digitar `\c` neste caso, porque o `mysql` o intrerpreta como parte da string que está coletando! Digite o caracter de aspas para fechar (então o `mysql` sabe que você fechou a string), então digite `\c`:

```
mysql> SELECT * FROM minha_tabela WHERE nome = "Smith AND idade < 30;
"> "\c
mysql>
```

O prompt volta para `mysql>`, indicando que o `mysql` está pronto para um novo comando. É importante saber o que os prompts `'>` e o `">` significam, porque se você entrar sem querer com uma string sem terminação, quaisquer linhas seguintes que forem digitadas serão ignoradas pelo `mysql` — incluindo uma linha contendo `QUIT!` Isto pode ser um pouco confuso, especialmente se você não sabe que você precisa fornecer as aspas finais antes poder cancelar o comando atual.

3.3 Criação e Utilização de um Banco de Dados

Agora que você já sabe como entrar com os comandos, é hora de acessar um banco de dados. Suponha que você tenha diversos animais de estimação em sua casa (menagerie) e você gostaria de ter o registro de vários tipos de informações sobre eles. Você pode fazer isto criando tabelas para armazenar seus dados e carregá-los com a informação desejada. Depois você pode responder diferentes tipos de questões sobre seus animais recuperando dados das tabelas. Esta seção mostrará como:

- Criar um banco de dados
- Criar uma tabela
- Carregar dados na tabela
- Recuperar dados de uma tabela de várias maneiras
- Usar múltiplas tabelas

O banco de dados menagerie será simples (deliberadamente), mas não é difícil pensar em situações na vida real em que um tipo similar de banco de dados pode ser usado. Por exemplo, um banco de dados deste tipo pode ser usado por um fazendeiro para gerenciar seu estoque de animais, ou por um veterinário para gerenciar registros de seus pacientes. Uma distribuição do menagerie contendo algumas das consultas e dados de exemplos usados nas seções seguintes podem ser obtidas do site Web do MySQL. Estão disponíveis tanto em **compressed formato tar** como em **formato Zip** .

Utilize a instrução `SHOW` para saber quais bancos de dados existem atualmente no servidor:

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

A lista de bancos de dados provavelmente será diferente na sua máquina, mas os bancos de dados `mysql` e `test` provavelmente estarão entre eles. O banco de dados `mysql` é necessário porque ele descreve privilégios de acessos de usuários. O banco de dados `test` é geralmente fornecido como um espaço para que os usuários possam fazer testes.

Se o banco de dados `test` existir, tente acessá-lo:

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

Perceba que o `USE`, como o `QUIT`, não necessitam de um ponto e vírgula. (Você pode terminar tais declarações com uma ponto e vírgula se gostar; isto não importa) A instrução `USE` é especial em outra maneira, também: Ela deve ser usada em uma única linha.

Você pode usar o banco de dados `test` (Se você tiver acesso a ele) para os exemplos que seguem mas qualquer coisa que você criar neste banco de dados pode ser removido por qualquer um com acesso a ele. Por esta razão, você provavelmente deve pedir permissão ao seu administrador MySQL para usar um banco de dados próprio. Suponha que você o chame de `menagerie`. O administrador precisa executar um comando como este:

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

onde `seu_usuario_mysql` é o nome do usuário MySQL atribuído a você.

3.3.1 Criando e Selecionando um Banco de Dados

Se o administrador criar seu banco de dados quando configurar as suas permissões, você pode começar a usá-lo. Senão, você mesmo precisa criá-lo:

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

No Unix, nomes de bancos de dados são caso sensível (ao contrário das palavras chave SQL), portanto você deve sempre fazer referência ao seu banco de dados como `menagerie` e não `Menagerie`, `MENAGERIE` ou outra variação. Isto também é verdade para nomes de tabelas. (No Windows, esta restrição não se aplica, entretanto você deve referenciar os bancos de dados e tabelas usando o mesmo caso em toda a parte da consulta.)

Criar um banco de dados não o seleciona para o uso; você deve fazer isso de forma explícita. Para fazer o `menagerie` o banco de dados atual, use o comando:

```
mysql> USE menagerie
Database changed
```

Seu banco de dados necessita ser criado somente uma única vez, mas você deve selecioná-lo para o uso cada vez que você iniciar uma sessão `mysql`. Você pode fazer isso usando a instrução `USE` como visto acima. Uma forma alternativa é selecionar o banco de dados na linha de comando quando você chamar o `mysql`. Apenas especifique seu nome depois de qualquer parâmetro de conexão que você pode precisar fornecer. Por exemplo:

```
shell> mysql -h servidor -u usuario -p menagerie
Enter password: *****
```

Perceba que `menagerie` não é sua senha no comando mostrado. Se você precisar passar sua senha na linha de comando depois da opção `-p`, você deve fazê-lo sem usar espaços (por exemplo, `-pminhasenha` e não como em `-p minhasenha`). Entretanto, colocando sua senha na linha de comando não é recomendado, porque isto expõe sua senha permitindo que outro usuário utilize a sua máquina.

3.3.2 Criando uma Tabela

Criar o banco de dados é a parte fácil, mas neste ponto ele está vazio, como o `SHOW TABLES` mostrará:

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

A parte mais difícil é decidir qual a estrutura que seu banco de dados deve ter: quais tabelas você precisará e que colunas estarão em cada uma delas.

Você irá precisar de uma tabela para guardar um registro para cada um de seus animais de estimação. Esta tabela pode ser chamada `pet`, e ela deve conter, pelo menos, o nome de cada animal. Como o nome por si só não é muito interessante, a tabela deverá conter outras informações. Por exemplo, se mais de uma pessoa na sua família também tem animais, você pode desejar listar cada dono. Você pode também desejargravar algumas informações descritivas básicas como espécie e sexo.

Que tal a idade? Pode ser do interesse, mas não é uma boa coisa para se armazenar em um banco de dados. A idade muda à medida em que o tempo passa, o que significa que você sempre terá de atualizar seus registros. Em vez disso, é melhor armazenar um valor fixo como a data de nascimento. Então, sempre que você precisar da idade, basta você calculá-la como a diferença entre a data atual e a data de aniversário. O MySQL fornece funções para fazer aritmética de datas, então isto não é difícil. Armazenando datas de aniversário no lugar da idade também oferece outras vantagens:

- Você pode usar o banco de dados para tarefas como gerar lembretes para aniversários que estão chegando. (Se você pensa que este tipo de query é algo bobo, perceba que é a mesma questão que você perguntar no contexto de um banco de dados comercial para identificar clientes para quais você precisará enviar cartão de aniversário, para um toque pessoal assistido pelo computador.)
- Você pode calcular a idade em relação a outras datas diferente da data atual. Por exemplo, se você armazenar a data da morte no banco de dados, você poderá facilmente calcular qual a idade que o bicho tinha quando morreu.

Você provavelmente pode pensar em outros tipos de informações que poderão ser úteis na tabela `pet`, mas as identificadas até o momento são suficientes por agora: `nome(name)`, `dono(owner)`, `espécie(species)`, `sexo(sex)`, `data de nascimento(birth)` e `data da morte(death)`.

Utilize a sentença `CREATE TABLE` para especificar o layout de sua tabela:

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

`VARCHAR` é uma boa escolha para os campos `name`, `owner`, e `species` porque os valores da coluna são de tamanho variável. Os tamanhos destas colunas não precisam necessariamente

de ser os mesmos e não precisam ser 20. Você pode escolher qualquer tamanho de 1 a 255, o que você achar melhor. (Se você não fizer uma boa escolha e depois precisar de um campo maior, o MySQL fornece o comando `ALTER TABLE`.)

O sexo dos animais podem ser representados em várias formas, por exemplo, "m" e "f" ou mesmo "macho" e "fêmea". É mais simples usar os caracteres "m" e "f".

O uso do tipo de dados `DATE` para as colunas `birth` e `death` são obviamente a melhor escolha.

Agora que você criou uma tabela, a instrução `SHOW TABLES` deve produzir alguma saída:

```
mysql> SHOW TABLES;
+-----+
| Tables in menagerie |
+-----+
| pet                |
+-----+
```

Para verificar se sua tabela foi criada da forma que você esperava, utilize a instrução `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 na sua tabela ou de que tipos elas são.

3.3.3 Carregando dados em uma tabela

Depois de criar sua tabela, você precisará povoá-la. As instruções `LOAD DATA` e `INSERT` são úteis para isto.

Suponha que seu registro de animais possa ser descrito como é abaixo: (Observe que o MySQL espera datas no formato `AAAA-MM-DD`; isto pode ser diferente do que você está acostumado.)

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1998-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	

Como você está começando com uma tabela vazia, uma forma simples de povoá-la é criar um arquivo texto contendo uma linha para cada um de seus animais, e depois carregar o conteúdo do arquivo para a tabela com uma simples instrução.

Você pode criar um arquivo texto ‘pet.txt’ contendo um registro por linha, com valores separado por tabulações e na mesma ordem em que as colunas foram listadas na instrução `CREATE TABLE`. Para valores em falta (como sexo desconhecido ou data da morte para animais que ainda estão vivos), você pode usar valores `NULL`. Para representá-lo em seu arquivo texto, use `\N`. Por exemplo, o registro para Whistler the bird podem parecer com isto (onde o espaço em branco entre os valores é um simples caractere de tabulação):

```
Whistler      Gwen          bird          \N           1997-12-09      \N
```

Para carregar o arquivo texto ‘pet.txt’ na tabela `pet`, use este comando:

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

Você pode especificar o valor do separador de colunas e o marcador de final de linha explicitamente na instrução `LOAD DATA` se você desejar. Mas os valores omitidos são suficientes para a instrução ler o arquivo ‘pet.txt’ corretamente.

Quando você desejar adicionar novos registros um a um, a instrução `INSERT` é usada. Na sua forma mais simples, você fornece valores para cada coluna, na ordem em que as colunas foram listadas na instrução `CREATE TABLE`. Suponha que Diane tenha um novo hamster chamado Puffball. Você pode adicionar um registro utilizando uma instrução `INSERT` desta forma:

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

Perceba que os valores de string e datas são especificados aqui como strings com aspas. Com o `INSERT` você também pode inserir `NULL` diretamente para representar um valor em falta. Não pode ser usado `\N` como você fez com `LOAD DATA`.

A partir deste exemplo, você deverá perceber que existem várias outras formas envolvidas para carregar seus registros inicialmente utilizando diversas instruções `INSERT` do que uma simples instrução `LOAD DATA`.

3.3.4 Recuperando Informações de uma Tabela

A instrução `SELECT` é usada para recuperar informações de uma tabela. A forma geral da instrução é:

```
SELECT o_que_mostrar
FROM de_qual_tabela
WHERE condições_para_satisfazer
```

`o_que_mostrar` indica o que você deseja ver. Isto pode ser uma lista de colunas ou `*` para indicar “todas colunas.” `de_qual_tabela` indica a tabela de onde você deseja recuperar os dados. A cláusula `WHERE` é opcional. Se estiver presente, `condições_para_satisfazer` especificam as condições que os registros devem satisfazer para fazer parte do resultado.

3.3.4.1 Selecionando Todos os Dados

A forma mais simples do `SELECT` recuperar tudo de uma tabela:

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

Esta forma do `SELECT` é útil se você deseja ver sua tabela inteira, como agora, depois de você acabar de carregá-la com os dados iniciais. Como podemos ver, a saída mostrada revela um erro no seu arquivo de dados: Bowser nasceu depois de morto ! Consultando seus papéis originais de pedigree, descobriu que o ano correto do nascimento é 1989, não 1998.

Existem pelo menos duas formas de corrigir isto:

- Edite o arquivo 'pet.txt' para corrigir o erro, depois limpe a tabela e recarregue-o usando `DELETE` e `LOAD DATA`:

```
mysql> SET AUTOCOMMIT=1; # Usado para a recriação da tabela
mysql> DELETE FROM pet;
mysql> LOAD DATA LOCAL INFILE "pet.txt" INTO TABLE pet;
```

Entretanto, se você fizer isto, você também deve refazer a entrada para Puffball.

- Corrigir somente o registro errado com uma instrução `UPDATE`:

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

Como foi mostrado acima, é fácil recuperar uma tabela inteira. Mas normalmente você não vai quer fazer isto, particularmente quando a tabela ficar grande. Normalmente você estará mais interessado em ter a resposta de uma questão em particular, no qual você especifica detalhes da informação que deseja. Vamos ver algumas consultas de seleção nos termos das questões sobre seus animais.

3.3.4.2 Selecionando Registros Específicos

Você pode selecionar apenas registros específicos da sua tabela. Por exemplo, se você deseja verificar a alteração que fez na data de nascimento do Bowser, selecione o registro desta forma:

```
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 o ano foi gravado corretamente agora como 1989 e não 1998.

Comparações de strings normalmente são caso insensitivo, então você pode especificar o nome como "bowser", "BOWSER", etc. O resultado da pesquisa será o mesmo.

Você pode especificar condições em qualquer coluna, não apenas no name. Por exemplo, se você deseja saber quais foram os animais que nasceram depois de 1998, teste o campo birth:

```
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 encontrar cadelas (dog/f):

```
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 consulta anterior utiliza o operador lógico AND (e). Existe também um operador OR (ou):

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

AND e OR podem ser misturados. Se você fizer isto, é uma ótima idéia usar parênteses para indicar quais condições devem ser agrupadas:

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

3.3.4.3 Selecionando Colunas Específicas

Se você não desejar ver todo o registro de sua tabela, especifique as colunas em que você estiver interessado, separado por vírgulas. Por exemplo, se você deseja saber quando seus animais nasceram, selecione as colunas name e birth:

```
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 saber quem são os donos dos animais, use esta consulta:

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

```

+-----+
| owner |
+-----+
| Harold |
| Gwen   |
| Harold |
| Benny  |
| Diane  |
| Gwen   |
| Gwen   |
| Benny  |
| Diane  |
+-----+

```

Entretanto, perceba que a query simplesmente retornou o campo `owner` de cada registro, e alguns deles apareceram mais de uma vez. Para minimizar a saída, recupere cada registro apenas uma vez, adicionando a palavra chave `DISTINCT`:

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

```

+-----+
| owner |
+-----+
| Benny  |
| Diane  |
| Gwen   |
| Harold |
+-----+

```

Você pode usar uma cláusula `WHERE` para combinar seleção de registros com seleção de colunas. Por exemplo, para obter a data de nascimento somente dos gatos e cachorros, utilize esta query:

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

```

3.3.4.4 Ordenando Registros

Você deve ter percebido nos exemplos anteriores que os registros retornados não são mostrados de forma ordenada. Entretanto, normalmente é mais fácil examinar a saída da consulta quando os registros são ordenados com algum sentido. Para ordenar o resultado, utilize uma cláusula `ORDER BY`.

Aqui está o dia de nascimento dos animais, ordenado 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 ordenação na ordem reversa, adicione a palavra chave `DESC` (descendente) ao nome da coluna que deve ser ordenada:

```

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 ordenar por múltiplas colunas. Por exemplo, para ordenar o tipo de animal, depois por dia de nascimento dentro do tipo de animal com os mais novos primeiro, utilize a seguinte consulta:

```

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

Perceba que a palavra chave DESC aplica somente para o nome da coluna precedente (`birth`); valores na coluna `species` continuam ordenados na ordem ascendente.

3.3.4.5 Cálculo de Datas

O MySQL fornece várias funções que você pode usar para realizar cálculos em datas, por exemplo, para calcular idades ou extrair partes de datas.

Para determinar quantos anos cada um do seus animais tem, compute a diferença do ano da data atual e a data de nascimento (`birth`), depois subtraia se a o dia/mês da data atual for anterior ao dia/mês da data de nascimento. A consulta seguinte, mostra, para cada animal, a data de nascimento, a data atual e a idade em anos.

```
mysql> SELECT name, birth, CURRENT_DATE,
-> (YEAR(CURRENT_DATE)-YEAR(birth))
-> - (RIGHT(CURRENT_DATE,5)<RIGHT(birth,5))
-> AS age
-> FROM pet;
```

name	birth	CURRENT_DATE	age
Fluffy	1993-02-04	2001-08-29	8
Claws	1994-03-17	2001-08-29	7
Buffy	1989-05-13	2001-08-29	12
Fang	1990-08-27	2001-08-29	11
Bowser	1989-08-31	2001-08-29	11
Chirpy	1998-09-11	2001-08-29	2
Whistler	1997-12-09	2001-08-29	3
Slim	1996-04-29	2001-08-29	5
Puffball	1999-03-30	2001-08-29	2

Aqui, `YEAR()` separa a parte do ano de uma data e `RIGHT()` separa os cinco caracteres mais a direita que representam a parte da data MM-DD. A parte da expressão que compara os valores MM-DD resulta em 1 ou 0, o qual ajusta a diferença do ano um ano abaixo se `CURRENT_DATE` ocorrer mais cedo, no ano, que `birth`. A expressão completa é um tanto deslegante, então um apelido (`age`) é usado para obter uma saída mais significativa.

A consulta funciona, mas o resultado pode ser mais compreensível se os registros forem apresentados em alguma ordem. Isto pode ser feito adicionando uma cláusula `ORDER BY name` para ordenar a saída pelo nome:

```
mysql> SELECT name, birth, CURRENT_DATE,
-> (YEAR(CURRENT_DATE)-YEAR(birth))
-> - (RIGHT(CURRENT_DATE,5)<RIGHT(birth,5))
-> AS age
-> FROM pet ORDER BY name;
```

name	birth	CURRENT_DATE	age
Bowser	1989-08-31	2001-08-29	11
Buffy	1989-05-13	2001-08-29	12
Chirpy	1998-09-11	2001-08-29	2
Claws	1994-03-17	2001-08-29	7
Fang	1990-08-27	2001-08-29	11
Fluffy	1993-02-04	2001-08-29	8
Puffball	1999-03-30	2001-08-29	2
Slim	1996-04-29	2001-08-29	5
Whistler	1997-12-09	2001-08-29	3

Para ordenar a saída por age em vez de name, é só utilizar uma cláusula `ORDER BY` diferente:

```
mysql> SELECT name, birth, CURRENT_DATE,
-> (YEAR(CURRENT_DATE)-YEAR(birth))
-> - (RIGHT(CURRENT_DATE,5)<RIGHT(birth,5))
-> AS age
-> FROM pet ORDER BY age;
```

name	birth	CURRENT_DATE	age
Chirpy	1998-09-11	2001-08-29	2
Puffball	1999-03-30	2001-08-29	2
Whistler	1997-12-09	2001-08-29	3
Slim	1996-04-29	2001-08-29	5
Claws	1994-03-17	2001-08-29	7
Fluffy	1993-02-04	2001-08-29	8
Fang	1990-08-27	2001-08-29	11
Bowser	1989-08-31	2001-08-29	11
Buffy	1989-05-13	2001-08-29	12

Uma consulta similar pode ser usada para determinar a idade na morte para animais que morreram. Para determinar quais são os animais, confira se o valor de `death` não é `NULL`. Depois para estes com valores não-`NULL`, compute a diferença entre os valores dos campos `death` e `birth`:

```
mysql> SELECT name, birth, death,
-> (YEAR(death)-YEAR(birth)) - (RIGHT(death,5)<RIGHT(birth,5))
-> AS age
-> FROM pet WHERE death IS NOT NULL ORDER BY age;
```

```

+-----+-----+-----+-----+
| name   | birth       | death       | age  |
+-----+-----+-----+-----+
| Bowser | 1989-08-31 | 1995-07-29 | 5   |
+-----+-----+-----+-----+

```

A query usa `death IS NOT NULL` em vez de `death != NULL` porque `NULL` é um valor especial. Isto será explicado depois. See [Working with NULL](#), page [Working with NULL](#).

E se você desejar saber quais animais fazem aniversário no próximo mês? Para este tipo de cálculo, ano e dia são irrelevantes; você simplesmente deseja extrair a parte do mês da coluna `birth`. O MySQL fornece diversas funções para extrair partes da data, como em `YEAR()`, `MONTH()` e `DAYOFMONTH()`. `MONTH` é a função apropriada aqui. Para ver como ela funciona, execute uma consulta simples que mostre o valor de `birth` e `MONTH(birth)`:

```

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            |
+-----+-----+-----+

```

Encontrar animais com aniversário no próximo mês também é fácil. Suponha que o mês atual é abril. Então o valor do mês é 4 e você procura por animais nascidos em Maio (mês 5) assim:

```

mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
+-----+-----+
| name | birth       |
+-----+-----+
| Buffy | 1989-05-13 |
+-----+-----+

```

Existe uma pequena complicação se o mês atual é Dezembro, é claro. Você não pode apenas adicionar um para o número do mês (12) e procurar por animais nascidos no mês 13, porque não existe tal mês. O certo seria procurar por animais nascidos em Janeiro (mês 1).

Você pode também escrever uma consulta para que funcione sem importar qual é o mês atual. Assim você não tem que usar um número de mês em particular na consulta. `DATE_ADD()` permite adicionar um intervalo de tempo para uma data fornecida. Se você adicionar um mês para o valor de `NOW()`, então extrair a parte do mês com `MONTH()`, o resultado é o mês no qual você deseja procurar por aniversários:

```

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

```

Uma maneira diferente para realizar a mesma tarefa é adicionar 1 para obter o mês seguinte ao atual (depois de usar a função módulo (MOD) para o valor do mês retornar 0 se ele for 12):

```
mysql> SELECT name, birth FROM pet
       -> WHERE MONTH(birth) = MOD(MONTH(NOW()), 12) + 1;
```

Perceba que MONTH retorna um número entre 1 e 12. E MOD(*alguma_coisa*,12) retorna um número entre 0 e 11. Então a adição tem que ser feita depois do MOD(), senão iríamos de Novembro (11) para Janeiro (1).

3.3.4.6 Trabalhando com Valores Nulos (NULL)

O valor NULL pode ser surpreendente até você usá-lo. Conceitualmente, NULL significa valor em falta ou valor desconhecido e é tratado de uma forma diferente de outros valores. Para testar o valor NULL, você não pode usar os operadores de comparações aritméticas como em =, <, ou !=. Para demonstrar para você mesmo, tente executar a seguinte consulta:

```
mysql> SELECT 1 = NULL, 1 != NULL, 1 < NULL, 1 > NULL;
+-----+-----+-----+-----+
| 1 = NULL | 1 != NULL | 1 < NULL | 1 > NULL |
+-----+-----+-----+-----+
|      NULL |      NULL |      NULL |      NULL |
+-----+-----+-----+-----+
```

Claramente você não obterá resultados significativos destas comparações. Utilize os operadores IS NULL e IS NOT NULL no lugar:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----+-----+
| 1 IS NULL | 1 IS NOT NULL |
+-----+-----+
|          0 |          1 |
+-----+-----+
```

No MySQL, 0 ou NULL significa falso e o resto é verdadeiro. O valor verdadeiro por o padrão em uma operação booleana é 1.

Este tratamento especial de NULL é porque, na seção anterior, foi necessário determinar quais animais não estavam mais vivos usando `death IS NOT NULL` no lugar de `death != NULL`.

3.3.4.7 Combinação de padrões

O MySQL fornece combinação de padrões do SQL bem como na forma de combinação de padrões baseado nas expressões regulares estendidas similares àquelas usadas pelos utilitários Unix como o `vi`, `grep` e `sed`.

A combinação de padrões SQL lhe permite você usar `_` para coincidir qualquer caractere simples e `%` para coincidir um número arbitrário de caracteres (incluindo zero caracter). No MySQL, padrões SQL são caso insensitivo por padrão. Alguns exemplos são vistos abaixo. Perceba que você não usa `=` ou `!=` quando usar padrões SQL; use os operadores de comparação `LIKE` ou `NOT LIKE` neste caso.

Para encontrar nomes começando 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 com o final 'fy':

```
mysql> SELECT * FROM pet WHERE name LIKE "%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 um '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, use o caracter '_':

```
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 combinação de padrões fornecido pelo MySQL usa expressões regulares estendidas. Quando você testa por uma combinação para este tipo de padrão, utilize os operadores REGEXP e NOT REGEXP (ou RLIKE e NOT RLIKE, que são sinônimos).

Algumas características das expressões regulares estendidas são:

- '.' combina qualquer caractere único
- Uma classe de caracteres '[...]' combina qualquer caractere que consta dentro dos colchetes. Por exemplo, '[abc]' combina com 'a', 'b', ou 'c'. Para nomear uma sequência de caracteres utilize um traço. '[a-z]' combina com qualquer letra minúscula e '[0-9]' combina com qualquer dígito.
- '*' combina nenhuma ou mais instâncias de sua precedência. Por exemplo, 'x*' combina com qualquer número de caracteres 'x', '[0-9]*' combina com qualquer número de dígitos e '.*' combina com qualquer número de qualquer coisa.

- Expressões regulares caso sensitivo, mas você pode usar uma classe de caracteres para combinar maiúsculas/minúsculas se você desejar. Por exemplo, '[aA]' combina o 'a' minúsculo ou maiúsculo e '[a-zA-Z]' combina qualquer letra em ambos casos.
- O padrão combina se ele ocorre em algum lugar no valor sendo testado. (Padrões SQL combinam somente se eles combinarem com todo o valor.)
- Para fazer com que um padrão deva combinar com o começo ou o fim de um valor sendo testado, utilize '^' no começo ou '\$' no final do padrão.

Para demonstrar como expressões regulares extendidas funcionam, as consultas com LIKE mostradas acima foram reescritas abaixo usando REGEXP.

Para encontrar nomes começando com 'b', utilize '^' para combinar com o começo do nome:

```
mysql> SELECT * FROM pet WHERE name REGEXP "^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 |
+-----+-----+-----+-----+-----+-----+
```

Antes da versão 3.23.4 do MySQL, REGEXP era caso sensitivo, e a consulta anterior não iria retornar nenhum registro. Para combinar letras 'b' maiúsculas e minúsculas, utilize esta consulta:

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

A partir do MySQL 3.23.4, para forçar uma comparação REGEXP com caso sensitivo, utilize a palavra-chave BINARY para tornar uma das strings em uma string binárias. Esta consulta irá combinar somente com 'b's minúsculos no começo de um nome:

```
mysql> SELECT * FROM pet WHERE name REGEXP BINARY "^b";
```

Para encontrar nomes finalizados com 'fy', utilize '\$' para combinar com o 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 um 'w' minúsculo ou maiúsculo, utilize esta consulta:

```
mysql> SELECT * FROM pet WHERE name REGEXP "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       |
+-----+-----+-----+-----+-----+-----+
```

Como uma expressão regular extendida encontra padrões coincidentes se eles ocorrem em qualquer lugar no valor comparado, não é necessário utilizar, na consulta anterior, nenhum

metacaracter em nenhum dos lados do padrão para fazê-lo coincidir com todo o valor, como seria feito se fosse utilizado o padrão SQL.

Para encontrar nomes contendo exatamente cinco caracteres, utilize ‘^’ e ‘\$’ para combinar com o começo e fim do nome e cinco instâncias de ‘.’ entre eles.

```
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 pode também escrever a consulta anterior utilizando o operador ‘{n}’ “repete-n-vezes”:

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

3.3.4.8 Contando Registros

Bancos de dados normalmente são usados para responder a perguntas, “Qual a frequência que certo tipo de dados ocorre em uma tabela?” Por exemplo, você deve querer saber quantos animais tem, ou quantos animais cada dono tem, ou você pode querer fazer vários outros tipos de censos com seus animais.

Contando o número total de animais que você tem é a mesma questão como em “Quantos registros existem na tabela `pet`?” porque existe um registro por animal. A função `COUNT()` conta o número de resultados não-NULL, portanto a pesquisa para contar seus animais parecerá com isto:

```
mysql> SELECT COUNT(*) FROM pet;
+-----+
| COUNT(*) |
+-----+
|          9 |
+-----+
```

Logo, você recuperará os nomes das pessoas que possuam animais. Você pode usar `COUNT()` se você desejar encontrar quantos animais cada dono possui:

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
+-----+-----+
| owner | COUNT(*) |
+-----+-----+
| Benny |         2 |
| Diane |         2 |
| Gwen  |         3 |
| Harold |         2 |
+-----+-----+
```

Perceba o uso de GROUP BY para agrupar todos os registros para cada owner (dono). Sem ele, você teria 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 GROUP BY são úteis para personalizar seus dados de diversas maneiras. Os seguintes exemplos mostram diferentes maneiras para realizar operações de censo nos 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 combinando 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 |
+-----+-----+-----+
```

Não é necessário selecionar uma tabela inteira quando estiver usando COUNT(). Por exemplo, a consulta anterior, quando realizada apenas procurando por cachorros e gatos, se parece com isto:

```
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 saber o número de animais por sexo somente de animais com sexo conhecido:

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

3.3.4.9 Utilizando Múltiplas Tabelas

A tabela `pet` mantém informações de quais animais você tem. Se você deseja gravar outras informações sobre eles como eventos em suas vidas, tais como visitas ao veterinário ou sobre suas crias, você necessitará de outra tabela. Como esta tabela deve se parecer? Ela precisa:

- Conter o nome do animal para que você saiba a qual animal pertence o evento.
- Uma data para que você saiba quando ocorreu o evento.
- Um campo para descrever o evento.
- Um campo com o tipo de evento, se você desejar classificá-los por categoria.

Dadas estas considerações, a instrução `CREATE TABLE` para a tabela `event` deve se parecer com isto:

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
-> type VARCHAR(15), remark VARCHAR(255));
```

Como na tabela `pet`, é mais fácil carregar os registros iniciais criando um arquivo texto delimitado por tabulações contendo a informação:

Fluffy	1995-05-15	litter	4 kittens, 3 female, 1 male
Buffy	1993-06-23	litter	5 puppies, 2 female, 3 male
Buffy	1994-06-19	litter	3 puppies, 3 female

Chirpy	1999-03-21	vet	needed beak straightened
Slim	1997-08-03	vet	broken rib
Bowser	1991-10-12	kennel	
Fang	1991-10-12	kennel	
Fang	1998-08-28	birthday	Gave him a new chew toy
Claws	1998-03-17	birthday	Gave him a new flea collar
Whistler	1998-12-09	birthday	First birthday

Carregue os registros usando:

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

Baseado no que você já aprendeu com as consultas realizadas na tabela `pet`, você deve estar apto para realizar pesquisas na tabela `event`; os princípios são o mesmo. Mas quando a tabela `event`, sozinha, é insuficiente para responder às suas questões?

Suponha que você deseje saber as idades de cada animal quando eles tiveram cria. A tabela `event` indica quando isto ocorre, mas para calcular a idade da mãe, você precisará da data de nascimento dela. Como isto está armazenado na tabela `pet`, você precisará das duas tabelas para a consulta:

```
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

Existem várias coisas que devem ser percebidas sobre esta consulta:

- A cláusula `FROM` lista as duas tabelas porque a consulta precisa extrair informação de ambas.
- Quando combinar (unir) informações de múltiplas tabelas, você precisa especificar como registros em uma tabela podem ser coincidadas com os registros na outra. Isto é simples porque ambas possuem uma coluna `name`. A consulta utiliza a cláusula `WHERE` para coincidir registros nas duas tabelas baseadas nos valores de `name`.
- Como a coluna `name` ocorre em ambas tabelas, você deve especificar qual a tabela a que você está se referindo. Isto é feito usando o nome da tabela antes do nome da coluna separados por um ponto (`.`).

Você não precisa ter duas tabelas diferentes para realizar uma união. Algumas vezes é útil unir uma tabela a ela mesma, se você deseja comparar registros em uma tabela com outros registros na mesma tabela. Por exemplo, para encontrar pares entre seus animais, você pode unir a tabela `pet` com ela mesma para formar pares de machos e fêmeas de acordo com as espécies:

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

```

Nesta consulta, nós especificamos apelidos para os nomes das tabelas para conseguir referenciar às colunas e manter com qual instância da tabela cada coluna de referência está associada.

3.4 Obtendo Informações Sobre Bancos de Dados e Tabelas

E se você esquecer o nome de um banco de dados ou tabela, ou como é a estrutura de uma certa tabela (por exemplo, como suas colunas são chamadas)? O MySQL resolve este problema através de diversas instruções que fornecem informações sobre os bancos de dados e as tabelas que ele suporta.

Você já viu `SHOW DATABASES`, que lista os bancos de dados gerenciados pelo servidor. Para saber qual banco de dados está sendo usado atualmente, utilize a função `DATABASE()`:

```

mysql> SELECT DATABASE();
+-----+
| DATABASE() |
+-----+
| menagerie |
+-----+

```

Se você ainda não selecionou nenhum banco de dados ainda, o resultado estará em branco. Para saber quais tabelas o banco de dados atual contém (por exemplo, quando você não tem certeza sobre o nome de uma tabela), utilize este comando:

```

mysql> SHOW TABLES;
+-----+
| Tables in menagerie |
+-----+
| event |
| pet |
+-----+

```

Se você deseja saber sobre a estrutura de uma tabela, o comando `DESCRIBE` é útil; ele mostra informações sobre cada uma das colunas da 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 | |

```

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

A coluna `Field` (campo) indica o nome da coluna, `Type` é o tipo de dados para a coluna, `Null` indica se a coluna pode conter valores nulos (`NULL`), `key` indica se a coluna é indexada ou não e `Default` especifica o valor padrão da coluna.

Se você tem índices em uma tabela, `SHOW INDEX FROM tbl_nome` traz informações sobre eles.

3.5 Exemplos de Consultas Comuns

Aqui estão os exemplos de como resolver problemas comuns com o MySQL.

Alguns dos exemplos usam a tabela `shop` para armazenar o preço de cada item (`article`) para certas revendas (`dealers`). Supondo que cada revenda tenha um preço fixo por artigo, então (`article, dealer`) é uma chave primária para os registros.

Inicie a ferramenta de linha de comando `mysql` e selecione um banco de dados:

```
mysql o-nome-do-seu-banco-de-dados
```

(Na maioria das instalações do MySQL, você pode usar o banco de dados 'test').

Você pode criar a tabela exemplo assim:

```
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);
```

Feito isto, os dados de exemplo são:

```
mysql> 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 |
+-----+-----+-----+
```

3.5.1 O Valor Máximo para uma Coluna

“Qual é o maior número dos itens?”

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

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

```

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

```

3.5.2 O Registro que Armazena o Valor Máximo para uma Certa Coluna

“Encontre o número, fornecedor e preço do ítem mais caro.”

No SQL ANSI isto é feito facilmente com uma sub-consulta:

```

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

```

No MySQL (que ainda não suporta sub-selects), faça isto em dois passos:

1. Obtenha o valor do preço máximo da tabela com uma instrução `SELECT`.
2. Usando este valor componha a pesquisa:

```

SELECT article, dealer, price
FROM   shop
WHERE  price=19.95

```

Outra solução é ordenar todos os registros por preço de forma descendente e obtenha somente o primeiro registro utilizando a cláusula específica do MySQL `LIMIT`:

```

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

```

NOTA: Se existir diversos ítems mais caros (por exemplo, cada um por 19,95) a solução `LIMIT` mostra somente um deles !

3.5.3 Máximo da Coluna por Grupo

“Qual é o maior preço por ítem?”

```

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 |
+-----+-----+

```

3.5.4 As Linhas Armazenando o Group-wise Máximo de um Certo Campo

“Para cada ítem, encontre o(s) fornecedor(s) com o maior preço.”

No SQL ANSI, poderíamos fazer uma sub-consulta, desta forma:

```
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 fazê-lo em diversos passos:

1. Obtenha a lista de (article,maxprice).
2. Para cada ítem obtenha os registros correspondentes que tenham o maior preço.

Isto pode ser feito facilmente 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 shop read;

INSERT INTO tmp SELECT article, MAX(price) FROM shop GROUP BY article;

SELECT shop.article, dealer, shop.price FROM shop, tmp
WHERE shop.article=tmp.article AND shop.price=tmp.price;

UNLOCK TABLES;

DROP TABLE tmp;
```

Se você não usar uma tabela TEMPORÁRIA, você deve bloquear também a tabela 'tmp'.

“Posso fazer isto com uma única query?”

Sim, mas somente com um truque ineficiente que eu chamo de “truque MAX-CONCAT”:

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

O último exemplo pode, é claro, ser feito de uma maneira mais eficiente fazendo a separação da coluna concatenada no cliente.

3.5.5 Utilizando Variáveis de Usuário

Você pode usar variáveis de usuários no MySQL para lembrar de resultados sem a necessidade de armazená-las em variáveis no cliente. See [\[Variables\]](#), page [\(undefined\)](#).

Por exemplo, para encontrar os itens com os preços mais altos e mais baixos você pode fazer:

```
select @min_price:=min(price),@max_price:=max(price) from shop;
select * from shop where price=@min_price or price=@max_price;
```

```
+-----+-----+-----+
| article | dealer | price |
+-----+-----+-----+
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+-----+-----+-----+
```

3.5.6 Utilizando Chaves Estrangeiras

Você não precisa de chaves estrangeiras para unir 2 tabelas.

A única coisa que o MySQL não faz é CONFERIR para ter certeza que as chaves que você usa realmente existem na(s) tabela(s) referenciadas e ele não apaga automaticamente registros da tabela com uma definição de chave estrangeira. Se você utiliza suas chaves da normalmente, irá funcionar bem:

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

3.5.7 Pesquisando em Duas Chaves

O MySQL ainda não otimiza quando você pesquisa em duas chaves diferentes combinadas com OR (Pesquisa em uma chave com diferentes partes OR é muito bem otimizadas).

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

A razão é que nós ainda não tivemos tempos para fazer este tratamento de uma maneira eficiente no caso geral. (A manipulação do AND é, em comparação, completamente geral e funciona muito bem).

No momento você pode resolver isto de maneira eficiente usando uma tabela TEMPORÁRIA. Este tipo de otimização é também muito boa se você estiver utilizando consultas muito complicadas no qual o servidor SQL faz as otimizações na ordem errada.

```
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;
```

A maneira descrita acima para resolver esta consulta é uma união (UNION) de duas consultas.

3.5.8 Calculando Visitas Diárias

A seguir demonstramos uma idéia de como usar as funções binárias de agrupamento para calcular o número de dias por mês que um usuário tem visitado uma página web.

```
CREATE TABLE t1 (year YEAR(4), month INT(2) UNSIGNED ZEROFILL, day INT(2) UNSIGNED ZEROFILL);
INSERT INTO t1 VALUES(2000,1,1),(2000,1,20),(2000,1,30),(2000,2,2),(2000,2,23),(2000,2,30);

SELECT year,month,BIT_COUNT(BIT_OR(1<<day)) AS days FROM t1 GROUP BY year,month;
```

que retornará:

```
+-----+-----+-----+
| year | month | days |
+-----+-----+-----+
| 2000 |    01 |    3 |
| 2000 |    02 |    2 |
+-----+-----+-----+
```

O exemplo acima calcula quantos dias diferentes foram usados para uma combinação fornecida de mês/ano, com remoção automática de entradas duplicadas.

3.6 Utilizando mysql em Modo Batch

Nas seções anteriores, você usou `mysql` interativamente para fazer consultas e ver os resultados. Você pode também executar `mysql` no modo batch. Para fazer isto, coloque os comando que você deseja executar em um arquivo, e diga ao `mysqld` para ler sua entrada do arquivo:

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

Se você precisa especificar parâmetros de conexão na linha de comando, o comando deve parecer com isto:

```
shell> mysql -h host -u user -p < batch-file
Enter password: *****
```

Quando você utilizar o `mysql` desta forma, você estará criando um arquivo script, depois executando o script.

Por que usar um script? Existem várias razões:

- Se você executa uma query repetidamente (digamos, todos os dias ou todas as semanas), transformá-lo em um script permite que você não o redigite toda vez que o executa.
- Você pode gerar novas consultas a partir das já existentes copiando e editando os arquivos de script.
- O modo batch pode também ser útil quando você estiver desenvolvendo uma consulta, particularmente para comandos de múltiplas linhas ou sequências de comandos com várias instruções. Se você cometer um erro, não será necessário redigitar tudo. Apenas edite seu arquivo script e corrija o erro, depois diga ao `mysql` para executá-lo novamente.
- Se você tem uma query que produz muita saída, você pode encaminhar a saída através de um paginador.

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

- Você pode capturar a saída em um arquivo para processamento posterior:

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

- Você pode distribuir seu script para outras pessoas para que elas possam executar os comandos também.
- Algumas situações não permitem uso interativo, por exemplo, quando você executa uma consulta através de um processo automático (`cron job`). Neste caso, você deve usar o modo batch.

A formato padrão de saída é diferente (mais conciso) quando você executa o `mysql` no modo batch do que quando você o usa interativamente. Por exemplo, a saída de `SELECT DISTINCT species FROM pet` se parece com isto quando você o executa interativamente:

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

Mas fica assim quando você o executa no modo batch:

```
species
bird
cat
dog
hamster
snake
```

Se você desejar obter o formato de saída interativa no modo batch, utilize `mysql -t`. Para mostrar a saída dos comandos que são executados, utilize `mysql -vvv`.

3.7 Consultas de Projetos Gêmeos

Em Analytikerna e Lentus, nós estamos fazendo os sistemas e trabalho de campo para um grande projeto de pesquisa. Este projeto é uma colaboração entre o Instituto de Medicina

Ambiental em Karolinksa Institutet Stockholm e a Seção de Pesquisa Clínica em Envelhecimento e Psicologia na University of Southern California.

O projeto envolve uma parte de seleção onde todos os gêmeos na Suécia mais velhos que 65 anos são entrevistados por telefone. Gêmeos que preenchem certos critérios passam para o próximo estágio. Neste estágio posterior, gêmeos que desejam participar são visitados por uma equipe de doutores/enfermeiros. Alguns dos consultas incluem exames físicos e neuropsicológico, testes de laboratório, imagem neural, determinação do estado psicológico e coletas de histórico familiar. Adicionalmente, dados são coletados em fatores de riscos médicos e ambientais.

Mais informações sobre o estudos dos gêmeos pode ser encontrados em:

<http://www.imm.ki.se/TWIN/TWINUKW.HTM>

A parte posterior do projeto é administrada com uma interface Web escrita utilizando a linguagem Perl e o MySQL.

Cada noite todos dados das entrevistas são movidos para um banco de dados MySQL.

3.7.1 Encontrando Todos Gêmeos Não-distribuídos

A seguinte consulta é usada para determinar quem vai na segunda parte do projeto:

```
select
    concat(p1.id, p1.tvab) + 0 as tvid,
    concat(p1.christian_name, " ", p1.surname) as Name,
    p1.postal_code as Code,
    p1.city as City,
    pg.abrev as Area,
    if(td.participation = "Aborted", "A", " ") as A,
    p1.dead as dead1,
    l.event as event1,
    td.suspect as tsuspect1,
    id.suspect as isuspect1,
    td.severe as tsevere1,
    id.severe as isevere1,
    p2.dead as dead2,
    l2.event as event2,
    h2.nurse as nurse2,
    h2.doctor as doctor2,
    td2.suspect as tsuspect2,
    id2.suspect as isuspect2,
    td2.severe as tsevere2,
    id2.severe as isevere2,
    l.finish_date
from
    twin_project as tp
/* For Twin 1 */
left join twin_data as td on tp.id = td.id and tp.tvab = td.tvab
left join informant_data as id on tp.id = id.id and tp.tvab = id.tvab
left join harmony as h on tp.id = h.id and tp.tvab = h.tvab
left join lentus as l on tp.id = l.id and tp.tvab = l.tvab
```

```

/* For Twin 2 */
left join twin_data as td2 on p2.id = td2.id and p2.tvab = td2.tvab
left join informant_data as id2 on p2.id = id2.id and p2.tvab = id2.tvab
left join harmony as h2 on p2.id = h2.id and p2.tvab = h2.tvab
left join lentus as l2 on p2.id = l2.id and p2.tvab = l2.tvab,
person_data as p1,
person_data as p2,
postal_groups as pg
where
/* p1 gets main twin and p2 gets his/her twin. */
/* ptvab is a field inverted from tvab */
p1.id = tp.id and p1.tvab = tp.tvab and
p2.id = p1.id and p2.ptvab = p1.tvab and
/* Just the scening survey */
tp.survey_no = 5 and
/* Skip if partner died before 65 but allow emigration (dead=9) */
(p2.dead = 0 or p2.dead = 9 or
 (p2.dead = 1 and
  (p2.death_date = 0 or
   (((to_days(p2.death_date) - to_days(p2.birthdate)) / 365)
    >= 65))))
and
(
/* Twin is suspect */
(td.future_contact = 'Yes' and td.suspect = 2) or
/* Twin is suspect - Informant is Blessed */
(td.future_contact = 'Yes' and td.suspect = 1 and id.suspect = 1) or
/* No twin - Informant is Blessed */
(ISNULL(td.suspect) and id.suspect = 1 and id.future_contact = 'Yes') or
/* Twin broken off - Informant is Blessed */
(td.participation = 'Aborted'
 and id.suspect = 1 and id.future_contact = 'Yes') or
/* Twin broken off - No inform - Have partner */
(td.participation = 'Aborted' and ISNULL(id.suspect) and p2.dead = 0))
and
l.event = 'Finished'
/* Get at area code */
and substring(p1.postal_code, 1, 2) = pg.code
/* Not already distributed */
and (h.nurse is NULL or h.nurse=00 or h.doctor=00)
/* Has not refused or been aborted */
and not (h.status = 'Refused' or h.status = 'Aborted'
 or h.status = 'Died' or h.status = 'Other')
order by
tvid;

```

Algumas explicações:

`concat(p1.id, p1.tvab) + 0 as tvid`

N queremos ordenar o `id` e o `tvab` concatenados na ordem numérica. Adicionando 0 ao resultado faz o MySQL tratar o resultado como um número.

coluna `id` Esta identifica um par de gêmeos. Ela é uma chave em todas as tabelas.

column `tvab`

Esta identifica um gêmeo em um par. Ela pode ter um valor de 1 ou 2.

column `ptvab`

Esta é o inverso de `tvab`. Quando `tvab` é 1 este campo é 2 e vice versa. Ela existe para poupar digitação e tornar mais fácil para o MySQL otimizar a query.

Esta consulta demonstra, entre outras coisas, como fazer buscas em uma tabela a partir da mesma tabela com uma uniao (`p1` e `p2`). No exemplo, isto é usado para conferir se um par de um gêmeo morreu antes de 65 anos. Se for verdade, a linha não é retornada.

Tudo acima existe em todas as tabelas com informações relacionada aos gêmeos. Nós temos uma chave em ambos `id, tvab` (todas as tabelas) e `id, ptvab` (`person_data`) para tornar as consultas mais rápidas.

Na nossa máquina de produção (Um UltraSPARC 200MHz), esta consulta retorna entre 150-200 linhas e gasta menos que um segundo.

O número atual de registros nas tabelas usadas acima:

Tabela	Registros
<code>person_data</code>	71074
<code>lentus</code>	5291
<code>twin_project</code>	5286
<code>twin_data</code>	2012
<code>informant_data</code>	663
<code>harmony</code>	381
<code>postal_groups</code>	100

3.7.2 Mostrando uma Tabela sobre a Situação dos Pares Gêmeos

Cada entrevista termina com um código da situação chamado `event`. A consulta mostrada abaixo é usada para mostrar uma tabela sobre todos pares gêmeos combinados por evento. Ela indica em quantos pares ambos gêmeos terminaram, em quantos pares um gêmeo terminou e o outro foi recusado e assim por diante.

```
select
    t1.event,
    t2.event,
    count(*)
from
    lentus as t1,
    lentus as t2,
    twin_project as tp
where
    /* We are looking at one pair at a time */
    t1.id = tp.id
    and t1.tvab=tp.tvab
```

```

        and t1.id = t2.id
        /* Just the sceening survey */
        and tp.survey_no = 5
        /* This makes each pair only appear once */
        and t1.tvab='1' and t2.tvab='2'
group by
        t1.event, t2.event;

```

3.8 Utilizando MySQL com Apache

A seção Contrib inclui programas que lhe permite autenticar seus usuários a partir de um banco de dados MySQL e também permite logar seus arquivos em uma tabela MySQL. See [\[Contrib\]](#), page [\[Contrib\]](#).

Você pode alterar o formato de log do Apache para ser facilmente lido pelo MySQL colocando o seguinte no arquivo de configuração do Apache:

```

LogFormat \
    "%h",%{Y%m%d%H%M%S}t,%>s,"%b",\{"Content-Type"}o", \
    "%U",\{"Referer"}i",\{"User-Agent"}i"

```

No MySQL você pode fazer algo assim:

```

LOAD DATA INFILE '/local/access_log' INTO TABLE nome_tabela
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\';

```

4 Administração de Bancos de Dados MySQL

4.1 Configurando o MySQL

4.1.1 Opções da Linha de Comando do `mysqld`

O `mysqld` aceita as seguintes opções de linha de comando:

- `--ansi` Utilizar a sintaxe ANSI SQL no lugar da sintaxe MySQL. See [\(undefined\)](#) [ANSI mode], page [\(undefined\)](#).
- `-b, --basedir=path` Encaminho para o diretório de instalação. Todos os caminhos normalmente são resolvidos em relação a este.
- `--big-tables` Permite grandes conjuntos de resultados salvando todos os conjuntos temporários em um arquivo. Ele resolve a maioria dos erros 'table full', mas também abaixa a velocidade das consultas nas quais as tabelas em memória seriam suficientes. Desde a Versão 3.23.2, o MySQL é capaz de resolver isto automaticamente usando memória para pequenas tabelas temporárias e trocando para o disco as tabelas, quando for necessário.
- `--bind-address=IP` Endereço IP para ligar.
- `--character-sets-dir=path` Diretório onde estão os conjuntos de caracteres. See [\(undefined\)](#) [Conjunto de caracteres], page [\(undefined\)](#).
- `--chroot=path` Muda o diretório raiz do daemon `mysqld` durante a inicialização. Medida de segurança recomendada. Limita os comandos `LOAD DATA INFILE` e `SELECT ... INTO OUTFILE`.
- `--core-file` Grava um arquivo `core` se o `mysqld` morrer. Para alguns sistemas você deve também especificar `--core-file-size` para `safe_mysqld`. See [\(undefined\)](#) [`safe_mysqld`], page [\(undefined\)](#)
- `-h, --datadir=path` Encaminha para o diretório raiz dos bancos de dados.
- `--default-character-set=charset` Configura o conjunto de caracteres padrão. See [\(undefined\)](#) [Conjunto de caracteres], page [\(undefined\)](#).
- `--default-table-type=type` Configura o tipo de tabela padrão. See [\(undefined\)](#) [Tipos de tabelas], page [\(undefined\)](#).

- `--debug[...]=`

Se o MySQL está configurado com `--with-debug`, você pode usar esta opção para obter um arquivo de rastreamento indicando o que o `mysqld` está fazendo. See [\(undefined\)](#) [Criando arquivos de rastreamento], page [\(undefined\)](#).
- `--delay-key-write-for-all-tables`

Não atualiza buffers das chaves entre escritas em nenhuma tabela MyISAM. See [\(undefined\)](#) [Parâmetros do servidor], page [\(undefined\)](#).
- `--enable-locking`

Habilita o bloqueio do sistema. Perceba que se usar esta opção em um sistema que não possui um `lockd()` completamente funcional (como no Linux) você pode fazer com que o `mysqld` entre em deadlock.
- `--enable-named-pipe`

Habilita suporte para named pipes (somente no NT/Win2000/XP).
- `-T, --exit-info`

Esta é uma máscara binária com diferentes parâmetros que pode ser usada para depurar o servidor `mysqld`; Esta opção não deve ser usada por alguém que não a conheça muito bem!
- `--flush`

Atualiza todas as alterações no disco depois de cada comando SQL. Normalmente o MySQL só faz a escrita de todas as alterações no disco depois de cada comando SQL e deixa o sistema operacional lidar com a sincronização com o disco. See [\(undefined\)](#) [Falhas], page [\(undefined\)](#).
- `-, --help`

Mostra uma pequena ajuda e sai.
- `--init-file=arquivo`

Lê comandos SQL do arquivo especificado na inicialização.
- `-L, --language=...`

Mensagens de erro do cliente na língua especificada. Pode ser fornecido como um caminho completo. See [\(undefined\)](#) [Línguas], page [\(undefined\)](#).
- `-l, --log[=arquivo]`

Log de conexões e consultas ao arquivo. See [\(undefined\)](#) [Log de consultas], page [\(undefined\)](#).
- `--log-isam[=arquivo]`

Log de todas alterações ISAM/MyISAM no arquivo (usado somente quando estiver depurando bancos ISAM/MyISAM).
- `--log-slow-queries[=arquivo]`

Log de todas as consultas que levam mais de `long_query_time` segundos de execução para um arquivo. See [\(undefined\)](#) [Log de consultas lentas], page [\(undefined\)](#).
- `--log-update[=arquivo]`

Log de atualizações para `file.#` onde `#` é um número único se não for fornecido. See [\(undefined\)](#) [Log de atualização], page [\(undefined\)](#).

--log-long-format

Log de algumas informações extra ao log de atualizações. Se você estiver usando `--log-slow-queries` então consultas que não estão usando índices são logadas ao log de consultas lentas.

--low-priority-updates

Operações de alterações das tabelas (`INSERT/DELETE/UPDATE`) irão ter prioridade menor do que as `selects`. Isto também pode ser feito usando `{INSERT | REPLACE | UPDATE | DELETE} LOW_PRIORITY ...` para baixar a prioridade de somente uma consulta, ou `SET OPTION SQL_LOW_PRIORITY_UPDATES=1` para alterar a prioridade em uma única thread. See [\(undefined\)](#) [Bloqueio de tabelas], page [\(undefined\)](#).

--memlock

Bloqueia o processo `mysqld` na memória. Isto funciona somente se o seu sistema suportar a chamada de sistema `mklockall()` (como no Solaris). Isto pode ajudar se você tiver um problema no qual o sistema operacional faz com que o `mysqld` faça a troca em disco.

--myisam-recover [=opção[,opção...]] onde opção é qualquer combinação de `DEFAULT`, `BACKUP`, `FORCE` ou `QUICK`. Você também pode configurar isto explicitamente para "" se você deseja desabilitar esta opção. Se esta opção for usada, o `mysqld` irá conferir na abertura se a tabela está marcada como quebrada ou se a tabela não foi fechada corretamente. (A última opção funciona somente se você estiver executando com `--skip-locking`). Se este for o caso `mysqld` irá executar uma conferência na tabela. Se a tabela estiver corrompida, o `mysqld` irá tentar repará-la.

As seguintes opções afetam no funcionamento da reparação.

<code>DEFAULT</code>	O mesmo que não fornecer uma opção para <code>--myisam-recover</code> .
<code>BACKUP</code>	Se os dados da tabela foram alterados durante a recuperação, salve um backup do arquivo de dados 'nome_tabela.MYD' como 'nome_tabela_dia_hora.BAK'.
<code>FORCE</code>	Execute a recuperação mesmo se perdermos mais de uma linha do arquivo .MYD.
<code>QUICK</code>	Não confira as linhas na tabela se não existir nenhum bloco apagado.

Antes da tabela ser reparada automaticamente, o MySQL irá adicionar uma nota no log de erros. Se você desejar que a recuperação da maioria das coisas não tenha a intervenção de algum usuário, devem ser usadas as opções `BACKUP, FORCE`. Isto irá forçar um reparo de uma tabela mesmo se alguns registros forem apagados, mas ele manterá o arquivo de dados antigo como um backup para que você possa examinar posteriormente o que aconteceu.

--pid-file=path

Encaminha para o arquivo pid usado pelo `safe_mysqld`.

-P, --port=...

Número da porta para conexões TCP/IP.

- o, --old-protocol**
Utilize o protocolo 3.20 para compatibilidade com alguns clientes muito antigos. See [\[Upgrading-from-3.20\]](#), page [\[undefined\]](#).
- one-thread**
Use somente uma thread (para depuração sobre Linux). See [\[Debugging server\]](#), page [\[undefined\]](#).
- O, --set-variable var=opção**
Fornecer um valor para uma variável. **--help** lista as variáveis. Você pode encontrar uma descrição completa para todas as variáveis na seção **SHOW VARIABLES** deste manual. See [\[SHOW VARIABLES\]](#), page [\[undefined\]](#). A seção de sintonia dos parâmetros do servidor inclui informações sobre como otimizá-los. See [\[Server parameters\]](#), page [\[undefined\]](#).
- safe-mode**
Salta alguns estágios de otimização. Implica **--skip-delay-key-write**.
- safe-show-database**
Não mostra os bancos de dados nos quais o usuário não tenha privilégios.
- safe-user-create**
Se isto estiver ativo, um usuário não pode criar novos usuários com o comando **GRANT**, se o usuário não ter o privilégio de **INSERT** na tabela `mysql.user` ou em alguma coluna desta tabela.
- skip-concurrent-insert**
Desliga a habilidade de selecionar e inserir ao mesmo tempo em tabelas **MyISAM**. (Isto só é usado se você achar que encontrou um erro neste recurso).
- skip-delay-key-write**
Ignore a opção `delay_key_write` para todas as tabelas. See [\[Parâmetros do servidor\]](#), page [\[undefined\]](#).
- skip-grant-tables**
Esta opção faz com que o servidor não use o sistema de privilégio. Isto dá a todos *acesso pleno* a todos os bancos de dados! (Você pode dizer a um servidor em execução para iniciar a usar as tabelas de permissão novamente executando `mysqladmin flush-privileges` ou `mysqladmin reload`.)
- skip-host-cache**
Nunca utiliza cache para nomes de máquina para resoluções de nomes mais rápidos, mas pesquisa o servidor DNS em todas conexões. See [\[DNS\]](#), page [\[undefined\]](#)
- skip-locking**
Não utilizar bloqueio de sistema. Para usar `isamchk` ou `myisamchk` você deve desligar o servidor. See [\[Stability\]](#), page [\[undefined\]](#). Perceba que na Versão 3.23 do MySQL pode ser usado **REPAIR** e **CHECK** para reparar/conferir tabelas **MyISAM**.

--skip-name-resolve

Nomes de máquinas não são resolvidos. Todos os valores da coluna `Host` nas tabelas de permissões devem conter números IP ou `localhost`. See [\[DNS\]](#), page [\[undefined\]](#).

--skip-networking

Não escutar conexões TCP/IP. Toda interação com `mysqld` deve ser feito através de sockets Unix. Esta opção é altamente recomendada para sistemas onde requisições locais são permitidas. See [\[DNS\]](#), page [\[undefined\]](#)

--skip-new

Não utilizar rotinas novas, possivelmente erradas. Implica `--skip-delay-key-write`. Isto também configura o tipo de tabela padrão como ISAM. See [\[ISAM\]](#), page [\[undefined\]](#).

--skip-symlink

Não apague ou renomeie arquivos de ligação simbólica para o diretório de dados.

--skip-safemalloc

Se o MySQL é configurado com `--with-debug=full`, todos os programas irão verificar a memória por erros para cada alocação e liberação de memória. Como esta consistência é muito lenta, você pode evitá-la, quando você não precisar de verificar a memória, usando esta opção.

--skip-show-database

Não permite o comando 'SHOW DATABASE', a menos que o usuário tenha privilégio **process**.

--skip-stack-trace

Não gravar rastreamentos de pilha. Esta opção é útil quando você estiver executando o `mysqld` sob um depurador. See [\[Depurando o servidor\]](#), page [\[undefined\]](#).

--skip-thread-priority

Desabilita o uso de prioridade das threads para um tempo de resposta mais rápido.

--socket=path

Arquivo socket para usar em conexões locais no lugar do padrão `/tmp/mysql.sock`.

--sql-mode=opção [, opção [, opção ...]]

Opção pode ser qualquer combinação de: `REAL_AS_FLOAT`, `PIPES_AS_CONCAT`, `ANSI_QUOTES`, `IGNORE_SPACE`, `SERIALIZE`, `ONLY_FULL_GROUP_BY`. Ela também pode ficar vazia ("") se você desejar limpá-la.

Especificar todas as opções acima é o mesmo que usar `-ansi`. Com estas opções, pode se ligar somente os modos SQL necessários. See [\[Modo ANSI\]](#), page [\[undefined\]](#).

transaction-isolation= { READ-UNCOMMITTED | READ-COMMITTED | REPEATABLE-READ | SERIALIZABLE }

Configura o nível de isolamento da transação padrão. See [\[SET TRANSACTION\]](#), page [\[undefined\]](#).

- t, --tmpdir=caminho
Encaminha para arquivos temporários. Ele pode ser útil se o seu diretório padrão /tmp está em uma partição muito pequena para armazenar tabelas temporárias.
- u, --user=nome_usuario
Executar o servidor mysqld como o usuário nome_usuario. Esta opção é *obrigatória* quando o mysqld é iniciado como usuário root.
- V, --version
Gera saída com a informação da versão e sai.
- W, --warnings
Imprime avisos como *Aborted connection...* no arquivo .err. See [\(undefined\)](#) [Erros de comunicação], page [\(undefined\)](#).

4.1.2 Arquivo de Opções my.cnf

O MySQL pode, desde a versão 3.22, ler as opções padrões de inicialização para o servidor e para clientes dos arquivos de opções.

O MySQL le opções padrões dos seguintes arquivos no Unix:

Nome do arquivo	Propósito
/etc/my.cnf	opções globais
DATADIR/my.cnf	opções específicas do servidor
defaults-extra-file	O arquivo especificado com <code>--defaults-extra-file=#</code>
~/my.cnf	opções específicas do usuário

DATADIR é o diretório de dados do MySQL (normalmente `'/usr/local/mysql/data'` para instalações binárias ou `'/usr/local/var'` para instalações de código fonte). Perceba que este é o diretório que foi especificado na hora da configuração, não o especificado com `--datadir` quando o mysqld inicia! (`--datadir` não tem efeito sobre o local onde o servidor procura por arquivos de opções, porque ele procura por eles antes de processar qualquer argumento da linha de comando.)

O MySQL lê os opções padrões dos seguintes arquivos no windows:

Nome do Arquivo	Propósito
windows-system-directory\my.ini	opções globais
C:\my.cnf	opções globais
C:\mysql\data\my.cnf	opções específicas do servidor

Perceba que no windows, você deve especificar todos os caminhos com / no lugar de \. Se for utilizado o \, será necessário digitá-lo duas vezes, pois o \ é o caractere de escape no MySQL.

O MySQL tenta ler os arquivos de opções na ordem listada acima. Se múltiplos arquivos de opções existirem, uma opção especificada em um arquivo lido depois recebe a precedência sobre a mesma opção especificada em um arquivo lido anteriormente. Opções especificadas na linha de comando recebem a precedência sobre opções especificadas em qualquer arquivo de opções. Algumas opções podem ser especificadas usando variáveis de ambiente. Opções especificadas na linha de comando ou nos arquivos de opção tem precedência sobre valores nas variáveis de ambiente. See [\(undefined\)](#) [Variáveis de ambiente], page [\(undefined\)](#).

Os seguintes programas suportam arquivos de opções: `mysql`, `mysqladmin`, `mysqld`, `mysqldump`, `mysqlimport`, `mysql.server`, `myisamchk` e `myisampack`

Você pode usar arquivos de opções para especificar qualquer opção estendida que o programa suporte! Execute o programa com `--help` para obter uma lista das opções disponíveis.

Um arquivo de opções pode conter linhas na seguinte forma:

#comentario

Linhas de comentário iniciam com o caractere '#' ou ';' . Linhas vazias são ignoradas.

[grupo] grupo é o nome do programa ou grupo para o qual você irá configurar as opções. Depois de uma linha de grupo, qualquer linha de opção ou `set-variable` são referentes ao grupo até o final do arquivo de opções ou outra linha de início de grupo.

opção Isto é equivalente à `--opção` na linha de comando.

opção=valor

Isto é equivalente à `--opção=valor` na linha de comando.

set-variable = variável=valor

Isto é equivalente à `--set-variable variável=valor` na linha de comando. Esta sintaxe deve ser usada para configurar uma variável `mysqld`.

O grupo `client` permite especificar opções para todos clientes MySQL (não o `mysqld`). Este é o grupo perfeito de se usar para especificar a senha que você usa para conectar ao servidor. (Mas tenha certeza que o arquivo de opções só pode ser lido e gravado por você) Perceba que para opções e valores, todos espaços em branco são automaticamente apagados. Você pode usar a sequência de escape '\b', '\t', '\n', '\r', '\\ e '\s' no valor da string ('\s' == espaço).

Aqui está um típico arquivo de opções globais.

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
set-variable = key_buffer_size=16M
set-variable = max_allowed_packet=1M

[mysqldump]
quick
```

Aqui está um típico arquivo de opções do usuário

```
[client]
# A senha seguinte será enviada para todos clientes MySQL
password=minha_senha

[mysql]
no-auto-rehash
```

```
set-variable = connect_timeout=2

[mysqlhotcopy]
interactive_timeout
```

Se você tem uma distribuição fonte, você encontrará arquivos de exemplo de configuração chamados 'my-xxxx.cnf' no diretório 'support-files'. Se você tem uma distribuição binária olhe no diretório de instalação 'DIR/support-file', onde DIR é o caminho para o diretório de instalação (normalmente '/usr/local/mysql'). Atualmente existem arquivos de configuração para sistemas pequenos, médios, grandes e enormes. Você pode copiar 'my-xxxx.cnf' para seu diretório home (renomeie a cópia para '.my.cnf' para experimentar).

Todos os clientes MySQL que suportam arquivos de opções aceitam opções:

-no-defaults	Não lê nenhum arquivo de opções.
-print-defaults	Imprima o nome do programa e todas opções.
-defaults-file=caminho-para-arquivo-padrão	Utilize somente o arquivo de configuração especificado.
-defaults-extra-file=caminho-para-arquivo-padrão	Leia este arquivo de configuração depois do arquivo de configuração global mas antes do arquivo de configuração do usuário.

Perceba que as opções acima devem vir primeiro na linha de comando para funcionar! `--print-defaults` pode, no entanto, ser usado logo depois dos comandos `--defaults-xxx-file`.

Notas para desenvolvedores: O tratamento de arquivos de opções é implementado simplesmente processando todos as opções coincidentes (isto é, opções no grupo apropriado) antes de qualquer argumento da linha de comando. Isto funciona bem para programas que usam a última instância de uma opção que é especificada diversas vezes. Se você tem um programa antigo que trata opções especificadas várias vezes desta forma mas não lê arquivos de opções, você só precisa adicionar duas linhas para lhe dar esta capacidade. Verifique o código fonte de qualquer um dos clientes MySQL padrão para ver como fazer isto.

Nos scripts shell você pode usar o comando 'my_print_defaults' para analisar os arquivos config:

```
shell> my_print_defaults client mysql
--port=3306
--socket=/tmp/mysql.sock
--no-auto-rehash
```

A saída acima contem todas opções para os grupos 'client' e 'mysql'.

4.1.3 Instalando Vários Servidores na Mesma Máquina

Em alguns casos você pode precisar de vários daemons (servidores) `mysqld` diferentes executando na mesma máquina. Você pode por exemplo desejar executar uma nova versão do MySQL para testar junto com uma versão mais antiga que está em produção. Outro caso é quando você quiser dar acesso a diferentes usuários em diferentes servidores `mysqld` gerenciados por eles mesmos.

Uma maneira de ter um novo servidor executando é iniciando-o com um socket e porta diferentes, como segue:

```
shell> MYSQL_UNIX_PORT=/tmp/mysqld-new.sock
shell> MYSQL_TCP_PORT=3307
shell> export MYSQL_UNIX_PORT MYSQL_TCP_PORT
shell> scripts/mysql_install_db
shell> bin/safe_mysqld &
```

O apêndice das variáveis de ambiente incluem uma lista de outras variáveis de ambiente que você pode usar e que afetam o `mysqld`. See [\(undefined\) \[Variáveis de ambiente\]](#), page [\(undefined\)](#).

A forma descrita acima é a maneira mais rápida e suja que normalmente é usado para testes. A melhor coisa é que com isto todas as conexões que você faz na shell acima irão automaticamente ser direcionadas para o novo servidor.

Se você precisar fazer isto permanentemente, você deve criar um arquivo de opções para cada servidor. See [\(undefined\) \[Arquivos de opção\]](#), page [\(undefined\)](#). No script de inicialização que é executado no tempo de boot (`mysql.server`) você deve especificar ambos os servidores:

```
safe_mysqld --default-file=path-to-option-file
```

Pelo menos as seguintes opções devem ser diferentes por servidor:

```
port=#
```

```
socket=path
```

```
pid-file=path
```

As seguintes opções devem ser diferentes, se elas forem usadas:

```
log=path
```

```
log-bin=path
```

```
log-update=path
```

```
log-isam=path
```

```
bdb-logdir=path
```

Se você precisar de mais performance, você também pode diferenciar:

```
tmpdir=path
```

```
bdb-tmpdir=path
```

See [\(undefined\) \[Opções de linha de comando\]](#), page [\(undefined\)](#).

Se você estiver instalando versões binárias do MySQL (arquivos `.tar`) e iniciá-los com `./bin/safe_mysql` então na maioria dos casos a única opção que você precisa adicionar/alterar são os argumentos `socket` e `port` do `safe_mysqld`.

4.1.4 Executando Múltiplos Servidores MySQL na Mesma Máquina

Existem circunstâncias em que você pode desejar executar vários servidores na mesma máquina. Por exemplo, você pode precisar testar um novo release MySQL enquanto deixa a configuração de sua produção existente inalterada. Ou você pode ser um provedor de serviços de Internet que deseja fornecer instalações independentes de MySQL para clientes diferentes.

Se você deseja executar múltiplos servidores, a forma mais fácil é compilá-los com diferentes portas TCP/IP e arquivos socket para que ambos não escutem na mesma porta TCP/IP ou arquivo socket. See [\[mysqld_multi\]](#), page [\[mysqld_multi\]](#).

Considere que um servidor existente está configurado para a porta e arquivo socket padrões. Então configure o novo servidor com o comando `configure` assim:

```
shell> ./configure --with-tcp-port=número_porta \
                --with-unix-socket-path=nome_arquivo \
                --prefix=/usr/local/mysql-3.22.9
```

Aqui `número_porta` e `nome_arquivo` deve ser diferente que o número da porta e o caminho do arquivo socket padrões e o valor `--prefix` deve especificar um diretório de instalação diferente daquele usado pelo servidor existente.

Você pode conferir o socket usado por qualquer servidor MySQL em execução com este comando:

```
shell> mysqladmin -h hostname --port=número_porta variáveis
```

Se você especificar “localhost” como o nome da máquina, `mysqladmin` irá por padrão usar sockets Unix no lugar de TCP/IP.

Se você tem um servidor MySQL executando na porta que você usou, você obterá uma lista de algumas das variáveis configuráveis mais importantes no MySQL, incluindo o nome do socket.

Não é necessário recompilar um novo servidor MySQL apenas para iniciar com uma porta ou socket diferentes. Você pode alterar a porta e o socket a ser usado especificando-os no tempo de execução como opções para o `safe_mysqld`:

```
shell> /path/to/safe_mysqld --socket=nome_arquivo --port=número_porta
```

`mysqld_multi` pode também utilizar o `safe_mysqld` (ou `mysqld`) como um argumento e passar as opções de um arquivo de configuração para o `safe_mysqld` e `mysqld`.

Se você executar o novo servidor no mesmo diretório do banco de dados que o outro servidor com o log habilitado, você também deve especificar o nome dos arquivos log para o `safe_mysqld` com `--log`, `--log-update`, ou `--log-slow-queries`. Se não, ambos servidores podem tentar escrever no mesmo arquivo de log.

AVISO: Normalmente você nunca deve ter dois servidores que atualizam dados no mesmo banco de dados! Se seu SO não suporta bloqueio de sistema a prova de falhas, isto pode provocar surpresas indesejáveis.

Se você deseja usar outro diretório de banco de dados para o segundo servidor, você pode usar a opção `--datadir=caminho` para o `safe_mysqld`.

NOTE também, que iniciando vários servidores MySQL (`mysqlds`) em diferentes máquinas e deixá-los acessar um diretório de dados sob NFS, é normalmente uma **MÁ IDÉIA!** O problema é que o NFS se tornará um gargalo, tornando o sistema lento. Ele não se destina para este tipo de uso. E você ainda não terá como ter certeza que dois ou mais `mysqlds` não estão interferindo uns com os outros. No momento não existe nenhuma plataforma que faria, com 100% de segurança, o bloqueio de arquivos (daemon

Quando você for conectar a um servidor MySQL que esteja executando com uma porta diferente daquela que foi compilada no seu cliente, você pode usar um dos seguintes métodos:

- Inicie o cliente com `--host 'nome_máquina' --port=número_porta` para conectar com TCP/IP, ou `[--host localhost] --socket=nome_arquivo` para conectar através de um socket Unix.
- Em programas C ou Perl, você pode especificar os argumentos de porta ou socket quando conectar a um servidor MySQL.
- Se você estiver usando o módulo Perl DBD::mysql você pode ler as opções dos arquivos de opções do MySQL. See [\[Option files\]](#), page [\[undefined\]](#).

```
$dsn = "DBI:mysql:test;mysql_read_default_group=client;mysql_read_default_
file=/usr/local/mysql/data/my.cnf"
$dbh = DBI->connect($dsn, $user, $password);
```

- Configure as variáveis de ambiente `MYSQL_UNIX_PORT` e `MYSQL_TCP_PORT` para apontar para o socket Unix e porta TCP/IP antes de iniciar seus clientes. Se você normalmente utiliza uma porta ou socket específico, você deve colocar os comandos para configurar as variáveis de ambiente no arquivo `‘.login’`. See [\[Environment variables\]](#), page [\[undefined\]](#).
- Especifique o socket e porta TCP/IP padrões no arquivo `‘.my.cnf’` no seu diretório home. See [\[Option files\]](#), page [\[undefined\]](#).

4.2 Detalhes Gerais de Segurança e o Sistema de Acesso

O MySQL tem um sistema de segurança/privilégios avançado mas não padrão. A próxima seção descreve como ele funciona.

4.2.1 Segurança Geral

Qualquer um usando o MySQL em um computador conectado à internet deve ler esta seção para evitar os erros de segurança mais comuns.

Discutindo segurança, nós enfatizamos a a necessidade de proteger completamente o servidor (não simplesmente o servidor MySQL) contra todos os tipos de ataques aplicáveis: eavesdropping, altering, playback e denial of service. Não cobriremos todos os aspectos de disponibilidade e tolerância a falhas aqui.

O MySQL utiliza a segurança baseado em Listas de Controle de Acesso (ACL) para todas conexões, consultas e outras operações que um usuário pode tentar realizar. Existe também algum suporte para conexões criptografadas SSL entre clientes MySQL e servidores. Vários dos conceitos discutidos aqui não são específicos do MySQL; as mesmas idéias podem ser aplicadas para a maioria das aplicações.

Quando executando o MySQL, siga estes procedimentos sempre que possível:

- **NUNCA CONCEDA A ALGUÉM (EXCETO AO USUÁRIO ROOT DO MySQL) ACESSO TABELA user NO BANCO DE DADOS mysql!** A senha criptografada é a senha real no MySQL. Se você conhece a senha listada na tabela `user` para um determinado usuário, você pode facilmente logar como este usuário se tiver acesso à máquina relacionada para aquela conta.
- Aprenda o sistema de controle de acessos do MySQL. Os comandos `GRANT` e `REVOKE` são usados para controlar o acesso ao MySQL. Não conceda mais privilégios do que o necessário. Nunca conceda privilégios para todas as máquinas.

Checklist:

- Tente `mysql -u root`. Se você conseguir conectar com sucesso ao servidor sem a solicitação de uma senha, você tem problemas. Qualquer um pode conectar ao seu servidor MySQL como o usuário `root` com privilégios plenos! Revise as instruções de instalação do MySQL, prestando atenção particularmente ao item sobre configuração da senha do usuário `root`.
- Utilize o comando `SHOW GRANTS` e confira para ver quem tem acesso a o que. Remova aqueles privilégios que não são necessários utilizando o comando `REVOKE`.
- Não mantenha nenhuma senha de texto puro no seu banco de dados. Quando seu computador fica comprometido, o intruso pode obter a lista completa de senhas e utilizá-las. No lugar utilize a função `MD5()` ou qualquer função de embaralhamento de via única.
- Não escolha senhas de dicionários. Existem programas especiais para quebrá-las. Mesmo senhas como “`xfish98`” não são boas. Muito melhor seria “`duag98`” que contém a mesma palavra `fish` mas digitada uma letra a esquerda em um teclado QWERTY convencional. Outro método seria usar “`Mhall`” que é obtido dos primeiros caracteres de cada palavra na frase “`Mary has a little lamb`”. Isto é fácil de lembrar e digitar, mas dificulta que alguém que não a conheça a advinhe.
- Invista em um firewall. Ele protege você de pelo menos 50% de todos os tipos de exploits em qualquer software. Coloque o MySQL atrás do firewall ou em uma zona desmilitarizada (DMZ).

Checklist:

- Tente examinar suas portas da Internet utilizando alguma ferramenta como o `nmap`. O MySQL utiliza a porta 3306 por padrão. Esta porta não deve ser acessível para máquinas não confiáveis. Outra maneira simples para conferir se sua porta do MySQL está aberta ou não é tentar o seguinte comando de alguma máquina remota, onde `nome_máquina` é o nome da máquina ou o endereço IP de seu servidor MySQL:

```
shell> telnet nome_máquina 3306
```

Se você obter uma conexão e alguns caracteres, a porta está aberta e deve ser fechada no seu firewall ou roteador, a menos que você realmente tenha uma boa razão para mantê-la aberta. Se o `telnet` apenas parar ou a conexão for recusada, tudo está bem; a porta está bloqueada.

- Não confie em nenhum dado incluídos pelos seus usuários. Eles podem tentar enganar seu código entrando com caracteres especiais ou sequências de escape nos formulários Web, URLs ou qualquer aplicação que você construa. Tenha certeza que sua aplicação continua segura se um usuário entrar com algo do tipo “`; DROP DATABASE mysql;`”. Este é um exemplo extremo, mas grandes falhas de segurança ou perda de dados podem ocorrer como o resultado de hackers utilizando técnicas similares, se você não estiver preparado para eles.

Também lembre de conferir dados numéricos. Um erro comum é proteger somente as strings. Em alguns casos as pessoas pensam que se um banco de dados contém somente dados disponíveis publicamente, ele não precisa ser protegido. Isto não é verdade. No mínimo ataques do tipo denial-of-service podem ser feitos nestes bancos de dados. A maneira mais simples para proteger deste tipo de ataque é usar apóstrofes em torno

das contantes numéricas: `SELECT * FROM tabela WHERE ID='234'` em vez de `SELECT * FROM table WHERE ID=234`. O MySQL automaticamente converte esta string para um número e corta todos os símbolos não-numéricos dela.

Checklist:

- Todas aplicações Web:
 - Tente inserir ‘’ e ‘’ em todos seus formulários Web. Se você obter qualquer tipo de erro do MySQL, investigue o problema imediatamente.
 - Tente modificar qualquer URL dinâmica adicionando %22 (‘’), %23 (#) e %27 (‘’) na URL.
 - Tente modificar os tipos de dados nas URLs dinâmicas de numérico para caractere contendo caracteres dos exemplos anteriores. Sua aplicação deve ser segura contra estes ataques e similares.
 - Tente inserir caracteres, espaços e símbolos especiais no lugar de número nos campos numéricos. Sua aplicação deve removê-los antes de passá-los para o MySQL ou sua aplicação deve gerar um erro. Passar valores não verificados ao MySQL é extramente perigoso!
 - Confira o tamanho dos dados antes de passá-los ao MySQL.
 - Considere ter sua aplicação conectando ao banco de dados utilizando um usuário diferente do que é utilizado com propósitos administrativos. Não forneça às suas aplicações mais privilégios de acesso do que elas necessitam.
- Usuários do PHP:
 - Confira a função `addslashes`. No PHP 4.0.3, uma função `mysql_escape_string()` está disponível e é baseada na função com o mesmo nome da API C do MySQL.
- Usuários do API C do MySQL:
 - Confira a chamada API `mysql_escape_string()`.
- Usuários do MySQL:
 - Confira os modificadores `escape` e `quote` para consultas streams.
- Usuários do Perl DBI:
 - Confira o método `quote()` ou utilize aspas simples ou duplas.
- Usuários do Java JDBC:
 - Utilize um objeto `PreparedStatement` e aspas simples ou duplas.
- Não transmita dados sem criptografia na Internet. Estes dados são acessíveis para todos que tenham o tempo e habilidade para interceptá-lo e usá-lo para seu propósito próprio. No lugar, utilize um protocolo de criptografia como o SSL ou SSH. O MySQL suporta conexões SSL interno desde a versão 3.23.9. O repasse de portas do SSH pode ser usado para criar um tunel criptografado (e com compressão) para a comunicação.
- Aprenda a usar os utilitários `tcpdump` e `strings`. Para a maioria dos casos você pode conferir se o fluxo de dados do MySQL está ou não criptografado utilizando um comando parecido com este:


```
shell> tcpdump -l -i eth0 -w - src or dst port 3306 | strings
```

(Isto funciona sobre Linux e deve funcionar com pequenas modificações sob outros sistemas.) Alerta: Se você não ver dados não significa sempre que esteja criptografado. Se você necessita de alta segurança, você deve consultar um especialista em segurança.

4.2.2 Como Tornar o MySQL Seguro contra Crackers

Quando você conectar a um servidor MySQL, você normalmente deve usar uma senha. A senha não é transmitida em texto puro sobre a conexão, porém o algoritmo de criptografia não é muito forte e com algum esforço um atacante engenhoso pode quebrar a senha se ele conseguir capturar o tráfego entre o cliente e o servidor. Se a conexão entre o cliente e o servidor passar por uma rede não confiável, você deve usar um tunnel SSH para criptografar a comunicação.

Todas outras informações são transferidas como texto que podem ser lido por qualquer um que consiga ver a conexão. Se você se preocupa com isto, você pode usar o protocolo de compressão (No MySQL versão 3.22 e superiores) para dificultar um pouco as coisas. Para deixar tudo ainda mais seguro você deve usar `ssh`. Você pode encontrar um cliente `ssh` open source em <http://www.openssh.org>, e um cliente `ssh` comercial em <http://www.ssh.com>. Com isto, você pode obter uma conexão TCP/IP criptografada entre um servidor MySQL e um cliente MySQL.

Para deixar um sistema MySQL seguro, você deve considerar as seguintes sugestões:

- Utilize senhas para todos os usuários MySQL. Lembre-se que qualquer um pode logar como qualquer outra pessoa simplesmente com `mysql -u outro_usuario nome_bd` se `outro_usuario` não tiver senha. Isto é um procedimento comum com aplicações cliente/servidor que o cliente pode especificar qualquer nome de usuário. Você pode alterar a senha de todos seus usuários editando o script `mysql_install_db` antes de executá-lo ou somente a senha para o usuário `root` do MySQL desta forma:

```
shell> mysql -u root mysql
mysql> UPDATE user SET Password=PASSWORD('nova_senha')
        WHERE user='root';
mysql> FLUSH PRIVILEGES;
```

- Não execute o daemon do MySQL como o usuário `root` do Unix. Isto é muito perigoso, porque qualquer usuário com privilégios `FILE` estará apto a criar arquivos como o `root` (por exemplo, `~root/.bashrc`). Para prevenir esta situação, `mysqld` irá recusar a execução como `root` a menos que ele seja especificado diretamente usando a opção `--user=root`.

O `mysqld` pode ser executado como um usuário normal sem privilégios. Você pode também criar um novo usuário Unix `mysql` para tornar tudo mais seguro. Se você executar o `mysqld` como outro usuário Unix, você não precisará alterar o usuário `root` na tabela `user`, porque nomes de usuário do MySQL não tem nada a ver com nomes de usuários Unix. Para iniciar o `mysqld` como outro usuário Unix, adicione uma linha `user` que especifica o nome de usuário para o grupo `[mysqld]` do arquivo de opções `/etc/my.cnf` ou o arquivo de opções `my.cnf` no diretório de dados do servidor. Por exemplo:

```
[mysqld]
user=mysql
```

Estas opções configuram o servidor para iniciar como o usuário designado quando você o inicia manualmente ou usando `safe_mysqld` ou `mysql.server`. Para maiores detalhes, veja [\(undefined\)](#) [Changing MySQL user], page [\(undefined\)](#).

- Não suportar links simbólicos para tabelas (Isto pode ser desabilitado com a opção `--skip-symlink`. Isto é muito importante caso você execute o `mysqld` como `root`, assim qualquer um que tenha acesso à escrita aos dados do diretório do `mysqld` podem apagar qualquer arquivo no sistema! See [\(undefined\)](#) [Symbolic links to tables], page [\(undefined\)](#).
- Verifique se o usuário Unix que executa o `mysqld` é o único usuário com privilégios de leitura/escrita nos diretórios de bancos de dados.
- Não forneça o privilégio `process` para todos os usuários. A saída de `mysqladmin processlits` mostra as consultas atualmente em execução, portanto qualquer usuário que consiga executar este comando deve ser apto a ver se outro usuário entra com uma consulta do tipo `UPDATE user SET password=PASSWORD('não_seguro')`.

O `mysqld` reserva uma conexão extra para usuários que tenham o privilégio `process`, portanto o usuário `root` do MySQL pode logar e verificar alguns detalhes mesmo se todas as conexões normais estiverem em uso.

- Não conceda o privilégio `file` a todos os usuários. Qualquer usuário que possua este privilégio pode gravar um arquivo em qualquer lugar no sistema de arquivos com os privilégios do daemon `mysqld`! Para tornar isto um pouco mais seguro, todos os arquivos gerados com `SELECT ... INTO OUTFILE` são lidos por todos, e não se pode sobrescrever arquivos existentes.

O privilégio `file` pode também ser usado para ler qualquer arquivo acessível para o usuário Unix com o qual o servidor está sendo executado. Pode ocorrer abusos como, por exemplo, usar `LOAD DATA` para carregar o arquivo `‘/etc/passwd’` em uma tabela, que pode então ser lido com `SELECT`.

- Se você não confia em seu DNS, você deve utilizar números IP no lugar de nomes de máquinas nas tabelas de permissão. De qualquer forma, você deve ter muito cuidado ao criar entradas de concessão utilizando valores de nomes de máquinas que contenham metacaractes!
- Se você deseja restringir o número de conexões para um único usuário, você pode fazê-lo configurando a variável `max_user_connections` no `mysqld`.

4.2.3 Opções de Inicialização para o `mysqld` em Relação a Segurança.

As seguintes opções do `mysqld` afetam a segurança:

`--safe-show-database`

Com esta opção, `SHOW DATABASES` retorna somente os bancos de dados nos quais o usuário tem algum tipo de privilégio.

`--safe-user-create`

Se for habilitado, um usuário não consegue criar novos usuários com o comando `GRANT`, se o usuário não tiver privilégio de `INSERT` na tabela `mysql.user`. Se você desejar fornecer a um usuário acesso para só criar novos usuários com

privilégios que o usuário tenha direito a conceder, você deve dar ao usuário o seguinte privilégio:

```
GRANT INSERT(user) on mysql.user to 'usuarior'@hostname';
```

Isto irá assegurar que o usuário não poderá alterar nenhuma coluna de privilégios diretamente, mas tem que usar o comando `GRANT` para conceder direitos para outros usuários.

`--skip-grant-tables`

Esta opção desabilita no servidor o uso do sistema de privilégios. Isto dá a todos os usuários *acesso total* a todos os bancos de dados! (Você pode dizer a um servidor em execução para para uar as tabelas de permissões executando `mysqladmin flush-privileges` ou `mysqladmin reload`.)

`--skip-name-resolve`

Nomes de máquinas não são resolvidos. Todos os valores da coluna `Host` nas tabelas de permissões devem ser números IP ou `localhost`.

`--skip-networking`

Não permitir conexões TCP/IP sobre a rede. Todas as conexões para `mysqld` devem ser feitas via Sockets Unix. Esta opção não é possível em sistemas que usam MIT-pthreads, porque o pacote MIT-pthreads não suporta sockets Unix.

`--skip-show-database`

Com esta opção a instrução `SHOW DATABASES` não retorna nada.

4.2.4 O Que o Sistema de Privilégios Faz

A função primária do sistema de privilégios do MySQL é autenticar um usuário a partir de uma determinada máquina e associar este usuário com privilégios a banco de dados como como `select`, `insert`, `update` e `delete`.

Funcionalidades adicionais incluem a habilidade de ter um usuário anônimo e conceder privilégio para funções específicas do MySQL como em `LOAD DATA INFILE` e operações administrativas.

4.2.5 Como o Sistema de Privilégios Funciona

O Sistema de privilégios do MySQL garante que todos usuários possam fazer exatamente aquilo que é permitido. Quando você conecta a um servidor MySQL, sua identidade é determinada pela **maquina de onde você conectou** e o **nome de usuário que você especificou**. O sistema concede privilégios de acordo com sua identidade e **com o que você deseja fazer**. O MySQL considera tanto os nomes de máquinas como os nomes de usuários porque existem poucas razões para assumir que um determinado nome de usuário pertence a mesma pessoa em todo lugar na Internet. Por exemplo, o usuário `bill` que conecta de `whitehouse.gov` não deve necessariamente ser a mesma pessoa que o usuário `bill` que conecta da `microsoft.com`. O MySQL lida com isto, permitindo a distinção de usuários em diferentes máquinas que podem ter o mesmo nome: Você pode conceder a `bill` um conjunto de privilégios para conexões de `whitehouse.gov` e um conjunto diferente de privilégios para conexões de `microsoft.com`.

O controle de acesso do MySQL é composto de dois estágios:

- 1o Estágio: O servidor confere se você pode ter acesso ou não.
- 2o. Estágio: Assumindo que você pode conectar, o servidor verifica cada requisição feita para saber se você tem ou não privilégios suficientes para realizar a operação. Por exemplo, se você tentar selecionar linha de uma tabela em um banco de dados ou apagar uma tabela do banco de dados, o servidor se certifica que você tem o privilégio **select** para a tabela ou o privilégio **drop** para o banco de dados.

O servidor utiliza as tabelas `user`, `db` e `host` no banco de dados `mysql` em ambos estágios do controle de acesso. Os campos nestas tabelas de permissão são detalhados abaixo:

Nome da Tabela	User	db	host
Campos de escopo	Host	Host	Host
	User	Db	Db
	Password	User	
Campos de privilégio	Select_priv	Select_priv	Select_priv
	Insert_priv	Insert_priv	Insert_priv
	Update_priv	Update_priv	Update_priv
	Delete_priv	Delete_priv	Delete_priv
	Index_priv	Index_priv	Index_priv
	Alter_priv	Alter_priv	Alter_priv
	Create_priv	Create_priv	Create_priv
	Drop_priv	Drop_priv	Drop_priv
	Grant_priv	Grant_priv	Grant_priv
	References_priv		
	Reload_priv		
	Shutdown_priv		
	Process_priv		
	File_priv		

No segundo estágio do controle de acesso (verificação da solicitação), o servidor pode, se a solicitação envolver tabelas, consultar adicionalmente as tabelas `tables_priv` e `columns_priv`. Os campos nestas tabelas são mostrados abaixo:

Nome da tabela	tables_priv	columns_priv
Campos de escopo	Host	Host
	Db	Db
	User	User
	Table_name	Table_name Column_name
Campos de privilégio	Table_priv	Column_priv
	Column_priv	
Outros campos	Timestamp	Timestamp
	Grantor	

Cada tabela de permissões contém campos de escopo e campos de privilégios.

Campos de escopo determinam o escopo de cada entrada nas tabelas, isto é, o contexto no qual a entrada se aplica. Por exemplo, uma entrada na tabela `user` com valores `Host` e `User` de `'thomas.loc.gov'` e `'bob'` devem ser usados para autenticar conexões feitas ao servidor por `bob` da máquina `thomas.loc.gov`. De maneira similar, uma entrada na tabela `db` com campos `Host`, `User` e `Db` de `'thomas.loc.gov'`, `'bob'` e `'reports'` devem ser usados quando `bob` conecta da máquina `thomas.loc.gov` para acessar o banco de dados `reports`. As tabelas `tables_priv` e `columns_priv` contêm campos de escopo indicando as combinações de tabelas ou tabela/coluna para o qual cada entrada se aplica.

Para propósitos de verificação de acessos, comparações de valores `Host` são caso insensitivo, valores `User`, `Password`, `Db` e `Table_name` são caso sensitivo. Valores `Column_name` são caso insensitivo no MySQL versão 3.22.12 ou posterior.

Campos de privilégios indicam os privilégios concedidos por uma entrada na tabela, isto é, quais operações podem ser realizadas. O servidor combina as informações de várias tabelas de concessão para formar uma descrição completa dos privilégios de um usuário. As regras usadas para fazer isto são descritas em [\[Request access\]](#), page [\[undefined\]](#).

Campos de escopo são strings, declaradas como mostrado abaixo; os valores padrão para cada é a string vazia:

Nome do Campo	Tipo
Host	CHAR(60)
User	CHAR(16)
Password	CHAR(16)
Db	CHAR(64) (CHAR(60) para as tabelas <code>tables_priv</code> e <code>columns_priv</code>)
Table_name	CHAR(60)
Column_name	CHAR(60)

Nas tabelas `user`, `db` e `host`, todos campos de privilégios são declarados como `ENUM('N','Y')` --- cada um pode ter um valor de `'N'` ou `'Y'` e o valor padrão é `'N'`.

Nas tabelas `tables_` e `columns_priv`, os campos de privilégios são declarados como campos `SET`:

Nome da Tabela	Nome Campo	Possíveis elementos do conjunto
<code>tables_priv</code>	<code>Table_priv</code>	<code>'Select'</code> , <code>'Insert'</code> , <code>'Update'</code> , <code>'Delete'</code> , <code>'Create'</code> , <code>'Drop'</code> , <code>'Grant'</code> , <code>'References'</code> , <code>'Index'</code> , <code>'Alter'</code>
<code>tables_priv</code>	<code>Column_priv</code>	<code>'Select'</code> , <code>'Insert'</code> , <code>'Update'</code> , <code>'References'</code>
<code>columns_priv</code>	<code>Column_priv</code>	<code>'Select'</code> , <code>'Insert'</code> , <code>'Update'</code> , <code>'References'</code>

De maneira resumida, o servidor utiliza as tabelas de permissões desta forma:

- Os campos de escopo da tabela `user` determinam quando permitir ou aceitar conexões. Para conexões permitidas, qualquer privilégio concedido em uma tabela `user` indica o privilégio global (superusuário) do usuário. Estes privilégios se aplicam a **todos** os bancos de dados no servidor.
- As tabelas `db` e `host` são usadas juntas:
 - Os campos de escopo da tabela `db` determinam quais usuários podem acessar determinados bancos de dados de máquinas determinadas. Os campos de privilégios determinam quais operações são permitidas.

- A tabela `host` é usada como uma extensão da tabela `db` quando você quer que uma certa entrada na tabela `db` seja aplicada a diversas máquinas. Por exemplo, se você deseja que um usuário esteja apto a usar um banco de dados a partir de diversas máquinas em sua rede, deixe o campo `Host` vazio no registro da tabela `db`, então popule a tabela `Host` com uma entrada para cada uma das máquinas. Este mecanismo é descrito com mais detalhes em [\[Request access\]](#), [page](#) [\[undefined\]](#).
- As tabelas `tables_priv` e `columns_priv` são similares à tabela `db`, porém são mais finas: Elas se aplicam ao nível de tabelas e colunas em vez do nível dos bancos de dados.

Perceba que os privilégios administrativos (**reload**, **shutdown** e etc) são especificados somente na tabela `user`. Isto ocorre porque operações administrativas são operações no próprio servidor e não são específicas e não específicas dos bancos de dados, portanto não existe razão para listar tais privilégios nas outras tabelas de permissão. De fato, somente a tabela `user` necessita ser consultada para determinar se você pode ou não realizar uma operação administrativa.

O privilégio **file** também só é especificado na tabela `user`. Ele não é um privilégio administrativo, mas sua habilidade para ler ou escrever arquivo no servidor é independente do banco de dados que você está acessando.

O servidor `mysqld` le o conteúdo das tabelas de permissões uma vez, quando é iniciado. Alterações nas tabelas de permissões tem efeito como indicado em [\[Privilege changes\]](#), [page](#) [\[undefined\]](#).

Quando você modifica o conteúdo das tabelas de permissões, é uma boa idéia ter certeza que suas alterações configuraram os privilégios da forma desejada. Para ajuda no diagnostico de problemas, veja [\[Access denied\]](#), [page](#) [\[undefined\]](#). Para conselhos sobre assuntos de segurança, See [\[Security\]](#), [page](#) [\[undefined\]](#).

Uma ferramenta de diagnóstico útil é o script `mysqlaccess`, que Yves Carlier fornece na distribuição MySQL. Chame `mysqlaccess` com a opção `--help` para descobrir como ele funciona. Perceba que o `mysqlaccess` confere o acesso usando somente as tabelas `user`, `db` e `host`. Ele não confere privilégios no nível de tabelas ou colunas.

4.2.6 Privilégios Fornecidos pelo MySQL

Informações sobre privilégios de usuários são armazenados nas tabelas `user`, `db`, `host`, `tables_priv` e `columns_priv` no banco de dados chamado `mysql`. O servidor MySQL lê o conteúdo destas tabelas quando ele inicia e sob as circunstâncias indicadas em [\[Privilege changes\]](#), [page](#) [\[undefined\]](#).

Os nomes usados neste manual que se referem-se aos privilégios fornecidos pelo MySQL são vistos abaixo juntos com o nome da coluna associada com cada privilégio nas tabelas de permissão e o contexto em que o privilégio aplica-se:

Privilégio	Coluna	Contexto
select	<code>Select_priv</code>	tabelas
insert	<code>Insert_priv</code>	tabelas
update	<code>Update_priv</code>	tabelas
delete	<code>Delete_priv</code>	tabelas

index	Index_priv	tabelas
alter	Alter_priv	tabelas
create	Create_priv	bancos de dados, tabelas, ou índices
drop	Drop_priv	bancos de dados ou tabelas
grant	Grant_priv	bancos de dados ou tabelas
references	References_priv	bancos de dados ou tabelas
reload	Reload_priv	administração do servidor
shutdown	Shutdown_priv	administração do servidor
process	Process_priv	administração do servidor
file	File_priv	acesso à arquivos no servidor

Os privilégios **select**, **insert**, **update** e **delete** permitem realizar operações em registros nas tabelas existentes em um banco de dados.

Instruções **SELECT** necessitam do privilégio **select** somente se ele precisar recuperar registros de uma tabela. Você pode executar certas instruções **SELECT** mesmo sem permissão para acessar algum dos bancos de dados no servidor. Por exemplo, você pode usar o cliente `mysql` como uma simples calculadora:

```
mysql> SELECT 1+1;
mysql> SELECT PI()*2;
```

O privilégio **index** permite a criação ou remoção de índices.

O privilégio **alter** permite utilizar **ALTER TABLE**.

Os privilégios **create** e **drop** permitem a criação de novos bancos de dados e tabelas, ou a remoção de bancos de dados e tabelas existentes.

Perceba que se for concedido o privilégio **drop** no banco de dados `mysql` para algum usuário, este usuário pode remover o banco de dados no qual os privilégios de acesso do MySQL estão armazenados!

O privilégio **grant** permite a você fornecer a outros usuários os privilégios que você mesmo possui.

O privilégio **file** fornece permissão para ler e escrever arquivos no servidor usando instruções **LOAD DATA INFILE** e **SELECT ... INTO OUTFILE**. Qualquer usuário que tenha este privilégio pode ler ou gravar qualquer arquivo que o servidor MySQL possa ler ou escrever.

Os privilégios restantes são usados para operações administrativas, que são realizadas utilizando o programa `mysqladmin`. A tabela abaixo mostra quais comandos do `mysqladmin` cada privilégio administrativos permite a execução:

Privilégio	Comandos permitidos
reload	<code>reload</code> , <code>refresh</code> , <code>flush-privileges</code> , <code>flush-hosts</code> , <code>flush-logs</code> e <code>flush-tables</code>
shutdown	<code>shutdown</code>
process	<code>processlist</code> , <code>kill</code>

O comando **reload** diz ao servidor para recarregar as tabelas de permissões. O comando **refresh** descarrega todas as tabelas e abre e fecha os arquivos de log. **flush-privileges** é um sinônimo para **reload**. Os outros comandos **flush-*** realizam funções similares ao **refresh** mas são mais limitados no escopo e podem ser preferíveis em alguns casos. Por exemplo, se você deseja descarregar apenas os arquivos log, **flush-logs** é uma melhor escolha do que **refresh**.

O comando **shutdown** desliga o servidor.

O comando **processlist** mostra informações sobre as threads em execução no servidor. O comando **kill** mata threads no servidor. Você sempre poderá mostrar ou matar suas próprias threads, mas você precisa do privilégio **process** para mostrar ou matar threads iniciadas por outros usuários. See [\(undefined\)](#) [KILL], page [\(undefined\)](#)

É uma boa idéia em geral conceder privilégios somente para aqueles usuários que necessitem deles, mas você deve ter muito cuidado ao conceder certos privilégios:

- O privilégio **grant** permite aos usuários repassarem seus privilégios a outros usuários. Dois usuários com diferentes privilégios e com o privilégio **grant** conseguem combinar seus privilégios.
- O privilégio **alter** pode ser usado para subverter o sistema de privilégios renomeando as tabelas.
- O privilégio **file** pode ser usado com abuso para ler qualquer arquivo de leitura no servidor em uma tabela de banco de dados, o conteúdo pode ser acessando utilizando **SELECT**. Isto inclui o conteúdo de todos os bancos de dados hospedados pelo servidor!
- O privilégio **shutdown** pode ser utilizado para negar inteiramente serviços para outros usuários, terminando o servidor.
- O privilégio **process** pode ser usado para ver o texto das consultas atualmente em execução, incluindo as consultas que configuram ou alteram senhas.
- Privilégios no banco de dados **mysql** pode ser utilizado para alterar senhas e outras informações de privilégio de acesso. (Senhas são armazenadas criptografadas, portanto um usuário malicioso não pode simplesmente lê-las para saber as senhas em texto puro). Se fosse possível acessar a coluna **password** do banco **mysql.user**, seria possível logar ao servidor MySQL como outro usuário. (Com privilégios suficientes, o mesmo usuário pode trocar a senha por outra diferente.)

Existem algumas coisas que você não pode fazer com o sistema de privilégios do MySQL:

- Você não pode especificar explicitamente que um determinado usuário deve ter acesso negado. Você não pode explicitamente comparar um usuário e depois recusar sua conexão.
- Você não pode especificar que um usuário tenha privilégios para criar ou remover tabelas em um banco de dados, mas não possa criar ou remover o banco de dados.

4.2.7 Conectando ao Servidor MySQL

Programas clientes do MySQL geralmente necessitam de parâmetros de conexão quando você precisar acessar um servidor MySQL: a máquina na qual você deseja se conectar, seu nome de usuário e sua senha. Por exemplo, o cliente **mysql** pode ser iniciado desta forma (argumentos opcionais são colocados entre '[' e ']'):

```
shell> mysql [-h nome_máquina] [-u nome_usuario] [-psua_senha]
```

Formas alternativas das opções **-h**, **-u** e **-p** são **--host=nome_máquina**, **--user=nome_usuario** e **--password=sua_senha**. Perceba que não existe *espaço* entre **-p** ou **--password=** e a senha que deve vir a seguir.

NOTA: Especificar a senha na linha de comando não é seguro! Qualquer usuário no seu sistema pode saber sua senha digitando um comando do tipo: **ps auxww**. See [\(undefined\)](#) [Option files], page [\(undefined\)](#).

O `mysql` utiliza valores padrão para parâmetros de conexão que não são passados pela linha de comando:

- O nome padrão da máquina (hostname) é `localhost`.
- O nome de usuário padrão é o mesmo nome do seu usuário no Unix.
- Nenhuma senha é fornecida se faltar o parâmetro `-p`.

Então, para um usuário Unix `joe`, os seguintes comandos são equivalentes:

```
shell> mysql -h localhost -u joe
shell> mysql -h localhost
shell> mysql -u joe
shell> mysql
```

Outros clientes MySQL comportam-se de forma similar.

Em sistemas Unix, você pode especificar valores padrões diferentes para serem usados quando você faz uma conexão, assim você não precisa digitá-los na linha de comando sempre que chamar o programa cliente. Isto pode ser feito de várias maneiras:

- Podem ser especificados parâmetros de conexão na seção `[client]` do arquivo de configuração `.my.cnf` no seu diretório home. A seção relevante do arquivo deve se parecer com isto:

```
[client]
host=nome_máquina
user=nome_usuario
password=senha_usuario
```

See [\(undefined\) \[Option files\]](#), page [\(undefined\)](#).

- Você pode especificar parâmetros de conexão utilizando variáveis de ambiente. O nome de máquina pode ser especificado para o `mysql` utilizando a variável `MYSQL_HOST`. O nome do usuário MySQL pode ser especificado utilizando `USER` (isto é somente para Windows). A senha pode ser especificada utilizando `MYSQL_PWD` (mas isto não é seguro; veja a próxima seção). See [\(undefined\) \[Environment variables\]](#), page [\(undefined\)](#).

4.2.8 Controle de Acesso, Estágio 1: Verificação da Conexão

Quando você tenta se conectar a um servidor MySQL, o servidor aceita ou rejeita a conexão baseado na sua identidade e se pode ou não verificar sua identidade fornecendo a senha correta. Senão, o servidor nega o acesso a você completamente. De outra forma, o servidor aceita a conexão, entra no estágio 2 e espera por requisições.

Sua identidade é baseada em duas partes de informação:

- A máquina de onde está conectando
- Seu nome de usuário no MySQL

A conferência da identidade é feita utilizando os tres campos de escopo da tabela `user` (`Host`, `User` e `Password`). O servidor aceita a conexão somente se uma entrada na tabela `user` coincidir com a máquina, nome de usuário e a senha fornecidos.

Valores dos campos escopo na tabela `user` podem ser especificados como segue:

- Um valor `Host` deve ser um nome de máquina ou um número IP ou `'localhost'` para indicar a máquina local.

- Você pode utilizar os metacaracteres '%' e '_' no campo **Host**.
- Um valor **Host** de '%' coincide com qualquer nome de máquina.
- Um valor **Host** em branco significa que o privilégio deve ser adicionado com a entrada na tabela **host** que coincide com o nome de máquina fornecido. Você pode encontrar mais informações sobre isto no próximo capítulo.
- Como no MySQL Versão 3.23, para valores **Host** especificados como números IP, você pode especificar uma máscara de rede indicando quantos bits de endereço serão usados para o número da rede. Por exemplo:

```
GRANT ALL PRIVILEGES on db.* to david@'192.58.197.0/255.255.255.0';
```

Isto permitirá que todos a se conectarem a partir de determinado IP cuja condição seguinte seja verdadeira:

```
IP_usuario & máscara_rede = ip_maquina.
```

No exemplo acima todos IPs no Intervalo 192.58.197.0 - 192.58.197.255 podem se conectar ao servidor MySQL.

- Metacaracteres não são permitidos no campo **User**, mas você pode especificar um valor em branco, que combina com qualquer nome. Se a entrada na tabela **user** que casa com uma nova conexão tem o nome do usuário em branco, o usuário é considerado como um usuário anônimo (o usuário sem nome), em vez do nome que o cliente especificou. Isto significa que um nome de usuário em branco é usado para todas as verificações de acessos durante a conexão. (Isto é, durante o estágio 2).
- O campo **Password** pode ficar em branco. O que não significa que qualquer senha possa ser usada, significa que o usuário deve conectar sem especificar uma senha.

Valores de **Password** que não estão em branco são apresentados como senhas criptografadas. O MySQL não armazena senhas na forma de texto puro para qualquer um ver. Em vez disso, a senha fornecida por um usuário que está tentando se conectar é criptografada (utilizando a função **PASSWORD()**). A senha criptografada é então usada quando o cliente/servidor estiver conferindo se a senha é correta (Isto é feito sem a senha criptografada sempre trafegando sobre a conexão.) Perceba que do ponto de vista do MySQL a senha criptografada é a senha REAL, portanto você não deve passá-la para ninguém! Em particular, não forneça a usuários normais acesso de leitura para as tabelas no banco de dados **mysql**!

Os exemplos abaixo mostram várias combinações de valores de **Host** e **User** nos registros da tabela **user** aplicando a novas conexões:

Valor em host	Valor em user	Conexões casadas com o registro
'thomas.loc.gov'	'fred'	fred, conectando de thomas.loc.gov
'thomas.loc.gov'	' '	Qualquer usuário, conectando de thomas.loc.gov
'%'	'fred'	fred, conectando a partir de qualquer máquina
'%'	' '	Qualquer usuário, conectando a partir de qualquer máquina
'%.loc.gov'	'fred'	fred, conectando de qualquer máquina do domínio loc.gov
'x.y.%'	'fred'	fred, conectando de x.y.net, x.y.com,x.y.edu, etc. (Isto provavelmente não é útil)

'144.155.166.177'	'fred'	fred, conectando da máquina com endereço IP 144.155.166.177
'144.155.166.%'	'fred'	fred, conectando de qualquer máquina na sub-rede de classe C 144.155.166
'144.155.166.0/255.255.255:0'		o mesmo que no exemplo anterior

Como você pode usar valores coringas de IP no campo `Host` (por exemplo, `'144.155.166.%'` combina com todas máquinas em uma subrede), existe a possibilidade que alguém possa tentar explorar esta capacidade nomeando a máquina como `144.155.166.algumlugar.com`. Para evitar tais tentativas, O MySQL desabilita a combinação com nomes de máquina que iniciam com dígitos e um ponto. Portanto se você possui uma máquina nomeada como `1.2.foo.com`, este nome nunca irá combinar com uma coluna `Host` das tabelas de permissões. Somente um número IP pode combinar com um valor coringa de IP.

Uma conexão de entrada pode coincidir com mais de uma entrada na tabela `user`. Por exemplo, uma conexão a partir de `thomas.loc.gov` pelo usuário `fred` pode combinar com diversas das entradas vistas acima. Como o servidor escolhe qual entrada usar se mais de uma coincide? O servidor resolve esta questão ordenando a tabela `user` no tempo de inicialização, depois procura pelas entradas na ordem da classificação quando um usuário tenta se conectar. A primeira entrada que coincidir é a que será usada.

A ordenação da tabela `user` funciona da forma mostrada a seguir. Suponha que a tabela `user` se pareça com isto:

```
+-----+-----+
| Host      | User      | ...
+-----+-----+
| %         | root      | ...
| %         | jeffrey   | ...
| localhost | root      | ...
| localhost |           | ...
+-----+-----+
```

Quando o servidor lê a tabela, ele ordena as entradas com os valores mais específicos de `Host` primeiro (`'%'` na coluna `Host` significa “qualquer máquina” e é menos específico). Entradas com o mesmo valor `Host` são ordenadas com os valores mais específicos de `User` primeiro (um valor em branco na coluna `User` significa “qualquer usuário” e é menos específico). O resultado da tabela `user` ordenada ficaria assim:

```
+-----+-----+
| Host      | User      | ...
+-----+-----+
| localhost | root      | ...
| localhost |           | ...
| %         | jeffrey   | ...
| %         | root      | ...
+-----+-----+
```

Quando uma conexão é iniciada, o servidor procura entre as entradas ordenadas e utiliza a primeira entrada coincidente. Para uma conexão a partir de `localhost` feito por `jeffrey`, as entradas com `'localhost'` na coluna `Host` coincide primeiro. Destas, a entrada com o nome do usuário em branco combina com o nome da máquina e o nome do usuário. (A entrada `'%'/jeffrey` também casaria, mas ela não é a primeira entrada coincidente na tabela.

Aqui está outro exemplo. Suponha que a tabela `user` fosse assim:

```
+-----+-----+
| Host          | User      | ...
+-----+-----+
| %             | jeffrey   | ...
| thomas.loc.gov |          | ...
+-----+-----+
```

A tabela ordenada pareceria com isto:

```
+-----+-----+
| Host          | User      | ...
+-----+-----+
| thomas.loc.gov |          | ...
| %             | jeffrey   | ...
+-----+-----+
```

Uma conexão a partir de `thomas.loc.gov` feita por `jeffrey` coincide com a primeira entrada, no entanto, uma conexão de `whitehouse.gov` feita por `jeffrey` coincidiria com a segunda entrada na tabela.

Um erro comum é pensar que para um determinado usuário, todas as entradas que citam explicitamente este usuário serão usadas primeiro quando o usuário tentar encontrar uma combinação para a conexão. Simplesmente isto não é verdade. O exemplo anterior ilustra isto, onde uma conexão de `thomas.loc.gov` feita por `jeffrey` combina primeiro não com a entrada contendo 'jeffrey' no valor do campo `user`, mas sim pela entrada sem o nome de usuário!

Se você tiver problemas conectando ao servidor, imprima a tabela `user` e ordene-a na manualmente para ver onde se deu o primeiro coincidência de valores.

4.2.9 Controle de Acesso, Estágio 2: Verificação da Requisição

Uma vez estabelecida uma conexão, o servidor entra no 2o estágio. Para cada requisição que vem na conexão, o servidor verifica se você tem privilégios suficientes para realizá-la, baseado nas operações que você deseja fazer. É aqui que os campos de concessões nas tabelas de permissões entram em ação. Estes privilégios pode vir de qualquer uma das tabelas `user`, `db`, `host`, `tables_priv` ou `columns_priv`. As tabelas de permissões são manipuladas com os comandos `GRANT` e `REVOKE`. See [\[GRANT\]](#), page [\[GRANT\]](#). (Você pode achar útil fazer referencia a [\[Privileges\]](#), page [\[Privileges\]](#), que lista os campos presentes em cada uma das tabelas de permissões.)

A tabela `user` concede privilégios que são especificados por você em uma base global e que se aplicam sem importar qual é o banco de dados atual. Por exemplo, se a tabela `user` concede a alguém o privilégio **delete**, este usuário pode apagar linhas de qualquer banco de dados no servidor! Em outras palavras, privilégios na tabela `user` são privilégios de superusuário. O correto é conceder privilégios na tabela `user` apenas para superusuários tais como os administradores de servidor ou de bancos de dados. Para outros usuários, você deve deixar os privilégios na tabela `user` configurados para 'N' e conceder privilégios somente em bancos de dados específicos, utilizando as tabelas `db` e `host`.

As tabelas `db` e `host` concedem privilégios para bancos de dados específicos. Valores nos campos de escopo podem ser especificados como a seguir:

- Os metacaracteres '%' e '_' podem ser usados nos campos `Host` e `Db` de ambas tabelas.
- O valor '%' em `Host` na tabela `db` significa “qualquer máquina.” Um valor em branco em `Host` na tabela `db` significa “consulte a tabela `host` para informação adicional.”
- O valor '%' ou em branco no campo `Host` na tabela `host` significa “qualquer máquina.”
- O valor '%' ou em branco no campo `Db` de ambas as tabelas significa “qualquer banco de dados.”
- O valor em branco no campo `User` em ambas tabelas coincide com o usuário anônimo.

As tabelas `db` e `host` são lidas e ordenadas quando o servidor inicia (ao mesmo tempo que ele lê a tabela `user`). A tabela `db` é ordenada nos campos de escopo `Host`, `Db` e `User` e a tabela `host` é ordenada nos campos de escopo `Host` e `Db`. Assim como na tabela `user`, a ordenação coloca os valores mais específicos no início e os menos específicos por último, e quando o servidor procura por entradas coincidentes, ele usa a primeira combinação que encontrar.

As tabelas `tables_priv` e `columns_priv` concedem privilégios específicos para tabelas e campos. Valores nos campos escopo podem ser especificados como a seguir:

- Os meta caracteres '%' e '_' podem ser usados no campo `Host` de ambas tabelas.
- O valor '%' ou em branco no campo `Host` em ambas tabelas significam “qualquer máquina”
- Os campos `Db`, `Table_name` e `Column_name` não podem conter meta caracteres ou serem brancos em ambas tabelas.

As tabelas `tables_priv` e `columns_priv` são ordenadas nos campos `Host`, `DB` e `User`. Isto é parecido com a ordenação da tabela `db`, no entanto, a ordenação é mais simples porque somente o campo `Host` pode conter meta caracteres.

O processo de verificação da requisição é descrito abaixo. (Se você já está familiarizado com o código de verificação de acesso, você irá perceber que a descrição aqui é um pouco diferente do algoritmo usado no código. A descrição é equivalente ao que o código realmente faz; ele só é diferente para tornar a explicação mais simples.)

Para requisições administrativas (`shutdown`, `reload`, etc.), o servidor confere somente a entrada da tabela `user`, porque ela é a única tabela que especifica privilégios administrativos. O acesso é concedido se o registro permitir a operação requisitada ou negado caso o contrário. Por exemplo, se você deseja executar `mysqladmin shutdown` mas a entrada em sua tabela `user` não lhe concede o privilégio `shutdown`, o acesso é negado mesmo sem consultar as tabelas `db` ou `host`. (elas não contém o campo `Shutdown_priv`, portanto não existe esta necessidade.)

Para requisições relacionadas aos bancos de dados (`insert`, `update`, etc.), o servidor primeiro confere os privilégios globais do usuário consultando as entradas da tabela `user`. Se a entrada permitir a operação requisitada, o acesso é concedido. Se os privilégios globais na tabela `user` são insuficientes, o servidor determina os privilégios específicos de banco de dados para o usuário consultando as tabelas `db` e `host`:

1. O servidor consulta a tabela `db` por uma combinação nos campos `Host`, `Db` e `User`. Os campos `Host` e `User` são comparados com o nome da máquina e o nome do usuário que faz a requisição. O campo `Db` é comparado com o banco de dados que o usuário deseja acessar. Se não existir entradas coincidentes para o `Host` e `User`, o acesso é negado.

2. Se existir uma combinação com a entrada da tabela `db` e seu campo `Host` não estiver em branco, aquela entrada define os privilégios específicos do banco de dados do usuário.
3. Se o registro coincidente da tabela `db` tiver o campo `Host` em branco, significa que a tabela `host` enumera quais máquinas são permitidas acessar o banco de dados. Neste caso, uma consulta adicional é feita na tabela `host` para encontrar um valores coincidentes nos campos `Host` e `Db`. Se nenhuma entrada na tabela `host` coincide, o acesso é negado. Se existir uma coincidência, os privilégios específicos de bancos de dados para o usuário são computados como a interseção (*não* a união!) dos privilégios nas entradas das tabelas `db` e `host`, isto é, os privilégios que são 'Y' em ambas entradas. (Desta forma você pode conceder privilégios gerais em entradas na tabela `db` e então restringi-los em uma base de máquina a máquina utilizando as entradas da tabela `host`.)

Depois de determinar os privilégios específicos do banco de dados concedido pelas entradas nas tabelas `db` e `host`, o servidor os adiciona aos privilégios globais concedidos pela tabela `user`. Se o resultado permitir a operação requisitada, o acesso será concedido. De outra forma, o servidor consulta os privilégios de tabelas e campos do usuário nas tabelas `tables_priv` e `columns_priv` e os adiciona aos privilégios do usuário. O acesso será permitido ou negado baseado no resultado.

Expresso em termos booleanos, a descrição precedente de como os privilégios de um usuário são calculados podem ser resumido assim:

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
```

Ele pode não ser aparente porque, se os privilégios da entrada global de `user` são inicialmente insuficientes para a operação requisitada, o servidor adiciona estes privilégios mais tarde aos privilégios específicos de banco de dados, tabelas e colunas. A razão é que uma requisição pode exigir mais que um tipo de privilégio. Por exemplo, se você executar uma instrução `INSERT ... SELECT`, você precisa dos privilégios `insert` e `select`. Seu privilégio pode ser tal que a entrada da tabela `user` concede um privilégio e a entrada da tabela `db` concede o outro. Neste caso, você tem os privilégios necessários para realizar a requisição, mas o servidor não pode obtê-los de ambas as tabelas por si próprio; os privilégios concedidos pelas entradas em ambas as tabelas de ser combinados.

A tabela `host` pode ser usada para manter uma lista dos servidores seguros.

Na `Tcx`, a tabela `host` contém uma lista de todas as máquina na rede local. A elas são concedidos todos os privilégios.

Você pode também usar a tabela `host` para indicar máquinas que *não* são seguras. Suponha que você tenha uma máquina `public.your.domain` que está localizada em uma área pública que você não considera segura. Você pode permitir o acesso a todas as máquinas de sua rede exceto a esta máquina usando entradas na tabela `host` desta forma:

```
+-----+-----+-----+
| Host                | Db | ...
+-----+-----+-----+
| public.your.domain | %  | ... (todos os privilégios configurados para 'N')
| %.your.domain      | %  | ... (todos os privilégios configurados para 'Y')
```

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

Naturalmente, você deve sempre testar suas entradas nas tabelas de permissões (por exemplo, usar o `mysqlaccess` para ter certeza que os privilégios de acesso estão atualmente configurados da forma que você imagina.

4.2.10 Causas dos Erros de Acesso Negado

Se você encontrar erros de **Acesso Negado** (Access denied) quando tentar conectar-se ao servidor MySQL, a lista abaixo indica alguns caminhos que você pode seguir para corrigir o problema:

- Depois de instalar o MySQL, você executou o script `mysql_install_db` para configurar o conteúdo inicial das tabelas de permissões? Se não, faça isto. See [\(undefined\)](#) [Default privileges], page [\(undefined\)](#). Teste os privilégios iniciais executando este comando:

```
shell> mysql -u root test
```

O servidor deve deixar você conectar sem erros. Você também deve assegurar que exista o arquivo `'user.MYD'` no diretório do banco de dados do MySQL. Normalmente ele fica em `'CAMINHO/var/mysql/user.MYD'`. onde `CAMINHO` é o caminho para a raiz da instalação do MySQL.

- Depois de terminar uma instalação, você deve conectar ao servidor e configurar seus usuários e suas permissões de acesso.

```
shell> mysql -u root mysql
```

O servidor deve permitir a conexão pois o usuário `root` MySQL vem inicialmente configurado sem senha. Isto também é um risco de segurança, portanto configurar a senha do usuário `root` é algo que deve ser feito enquanto você configura os outros usuários do MySQL.

Se você tentar se conectar como `root` e obter este erro:

```
Access denied for user: '@unknown' to database mysql
```

isto significa que você não possui um registro na tabela `user` com o valor `'root'` no campo `User` e que o `mysqld` não pode resolver o nome de máquina do cliente. Neste caso, você deve reiniciar o servidor com a opção `--skip-grant-tables` e editar seu arquivo `'/etc/hosts'` ou o `'\windows\hosts'` para adicionar uma entrada para sua máquina.

- Se você obter um erro como o seguinte:

```
shell> mysqladmin -u root -pXXXX ver
Access denied for user: 'root@localhost' (Using password: YES)
```

Significa que você está usando uma senha errada. See [\(undefined\)](#) [Passwords], page [\(undefined\)](#).

Se você esqueceu a senha de `root`, você pode reiniciar o `mysqld` com a opção `--skip-grant-tables` para alterar a senha. Você pode saber mais sobre esta opção posteriormente nesta mesma seção do manual.

Se você obter o erro acima mesmo se não tiver configurado uma senha, significa que você tem algum arquivo `my.ini` configurado para passar alguma senha incorreta. See [\(undefined\)](#) [Option files], page [\(undefined\)](#). Você pode evitar o uso de arquivos de opções com a opção `--no-defaults`, como a seguir:

```
shell> mysqladmin --no-defaults -u root ver
```

- Se você atualizou uma instalação existente do MySQL de um versão anterior à versão 3.22.11 para a Versão 3.22.11 ou posterior, você executou o script `mysql_fix_privilege_tables` ? Se não faça isto. A estrutura das tabelas de permissões alteraram com a Versão 3.22.11 do MySQL quando a instrução `GRANT` se tornou funcional.
- Se os seus privilégios parecerem alterados no meio de uma sessão, pode ser que o superusuário os alterou. A recarga das tabelas de permissões afeta novas conexões dos clientes, mas ela também afeta conexões existentes como indicado em [\[Privilege changes\]](#), page [\[undefined\]](#).
- Se você não consegue fazer a sua senha funcionar, lembre-se que você deve usar a função `PASSWORD()` se você configurar a senha com instruções `INSERT`, `UPDATE` ou `SET PASSWORD`. A função `PASSWORD()` é desnecessária se você especificar a senha usando a instrução `GRANT ... IDENTIFIED BY` ou o comando `mysqladmin password`. See [\[undefined\]](#) [\[Passwords\]](#), page [\[undefined\]](#).
- `localhost` é um sinônimo para seu nome de máquina local, e é também a máquina padrão em que clientes tentam se conectar se você não especificar explicitamente o nome da máquina. Entretanto, conexões para `localhost` não funcionam se você estiver executando em um sistema que utilize MIT-pthreads (conexões `localhost` são feitas utilizando sockets Unix, que não são suportadas pelas MIT-pthreads). Para evitar este problema nestes sistemas, você deve utilizar a opção `--host` para nomear explicitamente o servidor. Isto fará uma conexão TCP/IP ao servidor `mysqld`. Neste caso, você deve ter seu nome de máquina real nos registros da tabela `user` no servidor. (Isto é verdadeiro mesmo se você estiver executando um programa cliente na mesma máquina que o servidor.)
- Se você obter o erro `Access denied` quando tentando conectar ao banco de dados com `mysql -u nome_usuario _nome_bd`, você pode ter um problema com a tabela `user`. Verifique isto executando `mysql -u root mysql` e usando esta sentença SQL:


```
mysql> SELECT * FROM user;
```

O resultado deve incluir uma entrada com as colunas `Host` e `User` combinando com o nome de seu computador e seu nome de usuário no MySQL.
- A mensagem de erro `Access denied` irá dizer a você com qual usuário você está tentando se logar, a máquina que está tentando conectar e se você está utilizando uma senha ou não. Normalmente, você deve ter um registro na tabela `user` que combine exatamente com o nome de máquina e o nome de usuário que forem fornecidos na mensagem de erro. Por exemplo, se você obter uma mensagem de erro que contenha `Using password: NO`, isto significa que você está tentando se conectar sem uma senha.
- Se você obter o seguinte erro quando estiver tentando conectar de uma máquina diferente da que o servidor MySQL estiver executando, então não deve existir um registro na tabela `user` que combine com esta máquina:

```
Host ... is not allowed to connect to this MySQL server
```

Você pode corrigir isto utilizando a ferramenta de linha de comando `mysql` (no servidor!) para adicionar um registro à tabela `user`, `db` ou `host` para coincidir com o usuário e nome de máquina de onde você está tentando conectar, depois execute o comando `mysqladmin flush-privileges`. Se você não estiver executando o MySQL Versão 3.22 e você não sabe o número IP ou o nome da máquina da qual estiver conectando, você

deve colocar uma entrada com o valor '%' na coluna `Host` da tabela `user` e reiniciar o `mysqld` com a opção `--log` na máquina onde é executado o servidor. Depois tente conectar a partir da máquina cliente, a informação no log do MySQL irá indicar como você está realmente conectando. (Então troque o '%' na tabela `user` com o nome da máquina mostrado pelo log. De outra forma você teria um sistema que seria inseguro.)

Outra razão para este erro no Linux pode ser porque você está utilizando uma versão binária do MySQL que é compilada com uma versão diferente da `glibc` que você está usando. Neste caso você deve atualizar seu `SO/Glibc` ou fazer o download da versão fonte do MySQL e compilá-la. Um RPM fonte é, normalmente, fácil de compilar e instalar, logo, isto não é um grande problema.

- Se você obter uma mensagem de erro onde o nome da máquina não é exibido ou, no lugar do nome da máquina existir um IP, mesmo se você tenta a conexão com um nome de máquina:

```
shell> mysqladmin -u root -pXXXX -h some-hostname ver
Access denied for user: 'root' (Using password: YES)
```

Isto significa que o MySQL obteve algum erro quando tentava resolver o IP para um nome de máquina. Neste caso você pode executar `mysqladmin flush-hosts` para zerar o cache DNS interno. See [\[DNS\]](#), page [\[undefined\]](#).

Algumas soluções permanentes são:

- Tente descobrir o que está errado com seu servidor DNS e corrija os erros.
 - Especifique números IPs no lugar de nomes nas tabelas de privilégios do MySQL.
 - Inicie o `mysqld` com `--skip-name-resolve`.
 - Inicie o `mysqld` com `--skip-host-cache`.
 - Conecte à `localhost` se você estiver executando o servidor e o cliente na mesma máquina.
 - Coloque os nomes das máquinas clientes em `/etc/hosts`.
- Se `mysql -u root test` funciona mas `mysql -h nome_servidor -u root test` resultar em `Access denied`, então você pode não ter o nome correto para a sua máquina na tabela `user`. Um problema comum é quando o valor de `Host` na entrada da tabela `user` especifica um nome de máquina não qualificado, mas as rotinas de resolução de nomes de seu sistema retornam um nome qualificado completo do domínio (ou vice-versa). Por exemplo, se você tem uma entrada com o nome 'tcx' na tabela `user`, mas seu DNS diz ao MySQL que o nome da máquina é 'tcx.subnet.se', a entrada não irá funcionar. Tente adicionar um registro à tabela `user` que contenha o número IP de sua máquina como o valor da coluna `Host`. (Uma alternativa, seria adicionar um registro à tabela `user` com o valor de `Host` contendo um metacaracter, por exemplo, 'tcx.%'. Entretanto, o uso de nomes de máquinas terminando com '%' é *inseguro* e não é recomendado!)
 - Se `mysql -u nome_usuario test` funciona mas `mysql -u nome_usuario outro_bd` não funcionar, você não possui uma entrada para `outro_bd` listado na tabela `db`.
 - Se `mysql -u nome_usuario nome_bd` funciona quando executado no próprio servidor, mas `mysql -u nome_máquina -u nome_usuario nome_bd` não funciona quando executado em outra máquina cliente, você não possui o nome da máquina cliente listado na tabela `user` ou na tabela `db`.

- Se você não estiver entendendo porque obtém **Access denied**, remova da tabela **user** todas as entradas da coluna **Host** que contenham meta caracteres (entradas que contenham '\$' ou '_'). Um erro muito comum é inserir uma nova entrada com **Host='%'** e **User='algum usuário'**, pensando que isto irá permitir a você especificar **localhost** para conectar da mesma máquina. A razão disto não funcionar é que os privilégios padrões incluem uma entrada com **Host='localhost'** e **User=''**. Como esta entrada tem o valor **'localhost'** em **Host** que é mais específica que **'%'**, ela é usada no lugar da nova entrada quando se conectar de **localhost**! O procedimento correto é inserir uma segunda entrada com **Host='localhost'** e **User='algum_usuario'**, ou remover a entrada com **Host='localhost'** e **User= ''**.

- Se você obter o seguinte erro, você pode ter um problema com a tabela **db** ou a tabela **host**:

Access to database denied

Se a entrada selecionada da tabela **db** tiver um valor vazio na coluna **Host**, tenha certeza que exista uma ou mais entradas correspondentes na tabela **host** especificando quais máquinas aplicam-se à tabela **db**.

Se você obter o erro quando estiver utilizando comandos SQL **SELECT ... INTO OUTFILE** ou **LOAD DATA INFILE**, a entrada na tabela **user** provavelmente não tem o privilégio **file** habilitado.

- Lembre-se que programas clientes irão usar parâmetros de conexões especificados em arquivos de configuração ou variáveis ambientais. See [\(undefined\)](#) [Environment variables], page [\(undefined\)](#). Se parecer que algum cliente está enviando parâmetros errados para a conexão e você não os especificou na linha de comando, verifique seu ambiente e o arquivo **my.cnf** no seu diretório home. Você pode também conferir os arquivos de configurações do servidor MySQL, apesar de não ser interessante gravar configurações de cliente nestes arquivos. See [\(undefined\)](#) [Option files], page [\(undefined\)](#). Se você obter a mensagem de acesso negado (**Access denied**) quando estiver executando um cliente sem opções, tenha certeza que você não especificou uma senha antiga em nenhum de seus arquivos de opções! See [\(undefined\)](#) [Option files], page [\(undefined\)](#).
- Se você fizer alterações para as tabelas de permissões diretamente (utilizando uma instrução **INSERT** ou **UPDATE**) e suas alterações parecem ser ignoradas, lembre que você deve usar uma instrução **FLUSH PRIVILEGES** ou executar um comando **mysqladmin flush-privileges** para o servidor ler novamente as tabelas com os privilégios. De outra forma, suas alterações não farão efeito até que o servidor seja reiniciado. Lembre-se que depois de configurar a senha de **root** com um comando **UPDATE**, não será necessário especificar a senha até que você atualize os privilégios, pois o servidor ainda não saberá que você alterou a senha!
- Se você tiver problemas de acesso com Perl, PHP, Python ou um programa ODBC, tente conectar ao servidor com **mysql -u nome_usuario nome_bd** ou **mysql -u nome_usuario -psua_senha nome_bd**. Se você consegue conectar com o cliente **mysql**, existe algum problema com seu programa e não o acesso aos privilégios (Note que não espaço entre **-p** e a senha; você também pode utilizar a sintaxe **--password=sua_senha** para especificar a senha. Se você utilizar a opção **-p** sozinha, o MySQL irá lhe solicitar a senha.)
- Para testar, inicie o daemon **mysqld** com a opção **--skip-grant-tables**. Então você

pode alterar as tabelas de permissões do MySQL e utilizar o script `mysqlaccess` para conferir se suas modificações fizeram o não o efeito desejado. Quando você estiver satisfeito com suas alterações, execute `mysqladmin flush-privileges` para dizer ao servidor `mysqld` para iniciar utilizando as novas tabelas com os privilégios. **Nota:** Recarregar as tabelas de permissões sobrescreve a opção `--skip-grant-tables`. Isto lhe permite dizer ao servidor para começar a utilizar as tabelas de permissões novamente sem reiniciá-lo.

- Se tudo mais falhar, inicie o servidor `mysqld` com uma opção de depuração (por exemplo, `--debug=d,general,query`). Isto irá imprimir informações de máquinas e usuários sobre tentativas de conexões, e também informações sobre cada comando disparado. See [\[Making trace files\]](#), page [\[undefined\]](#).
- Se você tiver outros problemas com as tabelas de permissões do MySQL e sente que deve enviar o problema para a lista de discussão, sempre forneça um descarga das tabelas de permissões do seu MySQL. Você pode descarregar as tabelas com o comando `mysqldump mysql`. Como sempre, envie seus problemas utilizando o script `mysqlbug`. See [\[Bug reports\]](#), page [\[undefined\]](#). Em alguns casos você pode precisar reiniciar o `mysqld` com a opção `--skip-grant-tables` para executar o `mysqldump`.

4.3 Gerenciamento das Contas dos Usuários no MySQL

4.3.1 A Sintaxe de GRANT e REVOKE

```
GRANT tipo_priv [(column_list)] [, tipo_priv [(column_list)] ...]
  ON {nome_tabela | * | *.* | nome_bd.*}
  TO nome_usuario [IDENTIFIED BY 'password']
    [, nome_usuario [IDENTIFIED BY 'password'] ...]
  [WITH GRANT OPTION]
```

```
REVOKE tipo_priv [(column_list)] [, tipo_priv [(column_list)] ...]
  ON {nome_tabela | * | *.* | nome_bd.*}
  FROM nome_usuario [, nome_usuario ...]
```

O comando `GRANT` é implementado no MySQL versão 3.22.11 ou posterior. Para versões anteriores do MySQL, a instrução `GRANT` não faz nada.

Os comandos `GRANT` e `REVOKE` permitem aos administradores do sistema criar usuários e conceder e revogar direitos aos usuários do MySQL em quatro níveis de privilégios:

Nível Global

Privilégios globais aplicam para todos os bancos de dados em um determinado servidor. Estes privilégios são armazenados na tabela `mysql.user`.

Nível dos bancos de dados

Privilégios de bancos de dados aplicam-se a todas as tabelas em um determinado banco de dados. Estes privilégios são armazenados nas tabelas `mysql.db` e `mysql.host`.

Nível das tabelas

Privilégios de tabelas aplicam-se a todas as colunas em uma determinada tabela. Estes privilégios são armazenados na tabela `mysql.tables_priv`.

Nível das colunas

Privilégios de colunas aplicam-se a uma única coluna em uma determinada tabela. Estes privilégios são armazenados na tabela `mysql.columns_priv`.

Se você fornecer uma permissão para usuários que não existem, este usuário é criado. Para exemplos de como `GRANT` funciona, veja [\(undefined\) \[Adding users\], page \(undefined\)](#).

Para as instruções `GRANT` e `REVOKE`, `tipo_priv` pode ser especificado como um dos seguintes:

<code>ALL PRIVILEGES</code>	<code>FILE</code>	<code>RELOAD</code>
<code>ALTER</code>	<code>INDEX</code>	<code>SELECT</code>
<code>CREATE</code>	<code>INSERT</code>	<code>SHUTDOWN</code>
<code>DELETE</code>	<code>PROCESS</code>	<code>UPDATE</code>
<code>DROP</code>	<code>REFERENCES</code>	<code>USAGE</code>

`ALL` é um sinônimo para `ALL PRIVILEGES`. `REFERENCES` ainda não foi implementado. `USAGE` é atualmente um sinônimo para “sem privilégios”. Pode ser usado quando você desejar criar um usuário que não tenha privilégios.

Para anular o privilégio `grant` de um usuário, utilize o valor `tipo_priv` de `GRANT OPTION`:

```
REVOKE GRANT OPTION ON ... FROM ...;
```

Os únicos valores de `tipo_priv` que você pode especificar para uma tabela são `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE`, `DROP`, `GRANT`, `INDEX` e `ALTER`.

Os únicos valores de `tipo_priv` que você pode especificar para uma coluna (isto é, quando você usar uma cláusula `column_list`) são `SELECT`, `INSERT` e `UPDATE`.

Você pode configurar privilégios globais utilizando a sintaxe `ON *.*`. Você pode configurar privilégios de bancos de dados utilizando a sintaxe `ON nome_bd.*`. Se você especificar `ON *` e estiver com algum banco de dados aberto, será configurado os privilégios somente para este banco de dados. (**AVISO:** Se você especificar `ON *` e você *não* possui um banco de dados aberto, irá afetar os privilégios globais!).

Para acomodar concessões de direitos para usuários de máquinas arbitrárias, o MySQL suporta a especificação do valor `user_name` no formato `usuário@máquina`. Se você desejar especificar uma string `user` contendo caracteres especiais (como o `'-`), ou uma string contendo caracteres especiais ou meta caracteres (como o `'%`), você pode colocar o usuário ou o nome de máquina entre aspas (por exemplo, `'usuário-teste'@'máquina-teste'`).

Você pode especificar meta caracteres no nome da máquina. Por exemplo, `user@%.loc.gov` se aplica a `user` para qualquer máquina no domínio `loc.gov`, e `user@"144.155.166.%"` se aplica a `user` em qualquer máquina na subrede de classe C `144.155.166`.

O formato simples `user` é sinônimo de `user@"%"`. **NOTA:** Se você permite o acesso de usuários anônimos ao seu servidor MySQL (que é o padrão), você deve também adicionar todos usuários locais como `user@localhost` porque, de outra forma, a entrada de usuário anônimo para a máquina local na tabela `mysql.user` irá ser usada quando o usuário tentar a conexão ao servidor MySQL da máquina local! Usuários anônimos são definidos inserindo entradas com `User=''` na tabela `mysql.user`. Você pode verificar se isto se aplica a você executando esta instrução:

```
mysql> SELECT Host,User FROM mysql.user WHERE User='';
```

No momento, `GRANT` suporta somente nomes de máquinas, tabelas bancos de dados e colunas até 60 caracteres. Um nome de usuário pode ter até 16 caracteres.

Os privilégios para uma tabela ou coluna são formados através do OU lógico dos privilégios em cada um dos quatro níveis de privilégios. Por exemplo, se a tabela `mysql.user` especifica que um usuário tem um privilégio global **select**, isto não pode ser negado por uma entrada no nível de banco de dados, tabela ou coluna.

Os privilégios para uma coluna podem ser calculados da seguinte forma:

```

privilégios globais
OR (privilégios de banco de dados AND privilégios de máquina)
OR privilégios de tabela
OR privilégios de coluna

```

Na maioria dos casos, os direitos a um usuário são atribuídos em apenas um dos níveis de privilégios, portanto a vida normalmente não é tão complicada como mostrado acima. Os detalhes do procedimento de verificação dos privilégios são apresentados em [\[Privilege system\]](#), page [\[undefined\]](#).

Se você concede privilégios para uma combinação de usuário e máquina que não existem na tabela `mysql.user`, um registro é adicionado e permanece lá até ser removido com um comando `DELETE`. Em outras palavras, `GRANT` pode criar registros na tabela `user`, mas `REVOKE` não as removerá; para removê-las você deve usar a instrução explícita `DELETE`.

Na Versão 3.22.12 ou posterior do MySQL, se um novo usuário é criado ou se você possui privilégios de concessão globais, a senha do usuário será especificada utilizando a cláusula `IDENTIFIED BY`, se uma for dada. Se o usuário já possui uma senha, ela é trocada pela nova.

CUIDADO: Se você criar um novo usuário mas não especificar uma cláusula `IDENTIFIED BY`, o usuário não possuirá uma senha. Isto não é seguro.

Senhas podem também ser configuradas com o comando `SET PASSWORD`. See [\[SET OPTION\]](#), page [\[undefined\]](#).

Se você conceder privilégios para um banco de dados, uma entrada na tabela `mysql.db` é criada se necessário. Quando todos os privilégios para o banco de dados forem removidos com `REVOKE`, este registro é removido.

Se um usuário não tem privilégios em uma tabela, a tabela não é mostrada quando o usuário solicita uma lista de tabelas (com a instrução `SHOW TABLES` por exemplo).

A cláusula `WITH GRANT OPTION` dá ao usuário habilidade de fornecer à outros usuários quaisquer privilégios que ele tenha em um nível específico de privilégio. Você deve ter cuidado ao fornecer o privilégio **grant**, pois dois usuários podem se unir para unir privilégios!

Você não pode conceder a outro usuário um privilégio que não possua; o privilégio **grant** possibilita fornecer somente os privilégios que possuir.

Esteja ciente que quando conceder a um usuário o privilégio **grant** em um nível particular de privilégios, qualquer privilégio que o usuário já possua (ou seja fornecido no futuro!) nesse nível também pode ser concedido por este usuário. Suponha que você conceda a um usuário o privilégio **insert** em um banco de dados. Se você conceder o privilégio **select** no banco de dados e especificar `WITH GRANT OPTION`, o usuário além de poder repassar o privilégio **select** poderá também repassar o **insert**. Se você concede o privilégio **update** para o usuário no banco de dados, o usuário poderá conceder os privilégios **insert**, **select** e **update**.

Você não deve conceder privilégios **alter** a um usuário comum. Se você fizer isto, o usuário pode tentar enganar o sistema de privilégios renomeando tabelas!

Perceba que se você estiver utilizando privilégios de tabelas ou colunas, mesmo que para apenas um usuário, o servidor examina os privilégios de tabelas e colunas para todos os usuários e isto irá deixar o MySQL um pouco mais lento.

Quando o `mysqld` inicia, todos os privilégios são lidos na memória. Privilégios de bancos de dados, tabelas e colunas são iniciados um vez, e privilégios ao nível de usuário fazem efeito na próxima vez que o usuário conectar. Modificações nas tabelas de permissões que você realiza utilizando `GRANT` ou `REVOKE` são percebidas pelo servidor imediatamente. Se você modificar as tabelas de permissões manualmente (utilizando `INSERT`, `UPDATE`, etc), você deve executar uma instrução `FLUSH PRIVILEGES` ou executar `mysqladmin flush-privileges` para dizer ao servidor para recarregar as tabelas de permissões. See [\[Privilege changes\]](#), page [\[undefined\]](#).

As maiores diferenças entre o ANSI SQL e versões MySQL de `GRANT` são:

- No MySQL privilégios são fornecidos para uma combinação de usuário e máquina e não somente para um usuário.
- O ANSI SQL não possui privilégios no nível global ou de bancos de dados, e não suporta todos os tipos de privilégios que o MySQL suporta. O MySQL não suporta os privilégios `TRIGGER`, `EXECUTE` ou `UNDER` do ANSI SQL.
- Os privilégios do ANSI SQL são estruturados em uma maneira hierárquica. Se você remover um usuário, todos os privilégios do usuário são removidos. No MySQL os privilégios concedidos não são removidos automaticamente, mas você deve removê-los se necessário.
- Se no MySQL você possuir o privilégio `INSERT` em somente parte das colunas em uma tabela, você pode executar instruções `INSERT` na tabela; As colunas em que você não tem o privilégio `INSERT` irão receber seus valores padrões. O SQL ANSI necessita que você tenha o privilégio `INSERT` em todas as colunas.
- Quando você remove uma tabela no ANSI SQL, todos os privilégios para a tabela são removidos. Se você remover um privilégio no SQL ANSI, todos os privilégios que foram concedidos baseado neste privilégio são também removidos. No MySQL, privilégios só podem ser removidos com comandos `REVOKE` explícitos ou manipulando as tabelas de permissões do MySQL.

4.3.2 Nomes de Usuários e Senhas do MySQL

Existem várias diferenças entre a forma que nomes de usuários e senhas são usados pelo MySQL e a forma que são usados pelo Unix ou Windows:

- Nomes de usuários, como usado pelo MySQL para propósitos de autenticação, não tem nenhuma relação com os nomes de usuários do Unix (nomes de login) ou nomes de usuários Windows. A maioria dos clientes MySQL, por padrão, tentam se conectar utilizando o nome de usuário atual do Unix como o nome de usuário no MySQL, mas isto existe somente por conveniência. Programas clientes permite especificar um nome diferente com as opções `-u` ou `--user`. Isto significa que você não pode tornar um banco de dados seguro a menos que todos os usuários do MySQL possuam senhas. Qualquer um pode tentar se conectar ao servidor utilizando qualquer nome, e eles se conectarão com qualquer nome que não possua uma senha.
- Nomes de usuários MySQL podem ter o tamanho de até 16 caracteres; Nomes de usuário Unix normalmente são limitados até 8 caracteres.

- Senhas MySQL não tem nenhuma relação com senhas Unix. Não existe nenhuma associação entre a senha em que você utiliza para logar-se a uma máquina Unix e a senha que é utilizada para acessar um banco de dados na mesma máquina.
- O MySQL criptografa senhas utilizando um algoritmo diferente que o utilizado pelo processo de login do Unix. Veja as descrições das funções `PASSWORD()` e `ENCRYPT()` em [\[Miscellaneous functions\]](#), page [\[undefined\]](#). Perceba que mesmo que a senha é armazenada 'embaralhada', o conhecimento da sua senha 'embaralhada' é o suficiente para conseguir se conectar ao servidor MySQL!

Usuários MySQL e seus privilégios são criados normalmente com o comando `GRANT`, See [\[GRANT\]](#), page [\[undefined\]](#).

Quando você se conecta a um servidor MySQL com um cliente de linha de comando você pode especificar a senha com `--password=sua-senha`. See [\[Connecting\]](#), page [\[undefined\]](#).

```
mysql --user=monty --password=guess nome_do_banco
```

Se você deseja que o cliente lhe solicite a senha, deve ser especificado o parâmetro `--password` sem nenhum argumento

```
mysql --user=monty --password nome_do_banco
```

ou no formato curto:

```
mysql -u monty -p nome_do_banco
```

Perceba que no último exemplo a senha **não** é 'nome_do_banco'.

Se você deseja usar a opção `-p` para fornecer uma senha você deve fazer assim:

```
mysql -u monty -pguess nome_do_banco
```

Em alguns sistemas, a chamada da biblioteca que é utilizada pelo MySQL para solicitar por uma senha corta automaticamente a senha para 8 caracteres. Internamente o MySQL não limita o tamanho limite da senha.

4.3.3 Quando as Alterações nos Privilégios tem Efeito

Quando o `mysqld` inicia, todas o conteúdo das tabelas de permissões são lidos em memória e tem efeito neste momento.

As modificações das tabelas de permissões que você realiza utilizando `GRANT`, `REVOKE` ou `SET PASSWORD` são imediatamente reconhecidas pelo servidor.

Se você alterar as tabelas de permissões manualmente (utilizando `INSERT`, `UPDATE`, etc), você deve executar a instrução `FLUSH PRIVILEGES` ou executar `mysqladmin flush-privileges` ou `mysqladmin reload` para dizer ao servidor para recarregar as tabelas de permissões. De outra forma suas alterações *não terão efeito* até que o servidor seja reiniciado. Se você alterar as tabelas de permissões manualmente mas se esquecer de recarregar os privilégios, suas alteração vão parecer não ter feito nenhuma diferença!

Quando o servidor reconhecer que as tabelas de permissões foram alteradas, conexões existentes são afetadas da seguinte forma:

- Alterações nos privilégios de tabelas e colunas fazem efeito com a próxima requisição do cliente.
- Alterações nos privilégios de bancos de dados fazem efeito no próximo comando `USE nome_bd`.

Alterações nos privilégios globais e alterações de senhas fazem efeito na próxima vez que o cliente conectar.

4.3.4 Configurando os Privilégios Iniciais do MySQL

Depois de instalar o MySQL, você configura os privilégios iniciais dos acessos executando `scripts/mysql_install_db`. See [\[Quick install\]](#), page [\[undefined\]](#). O script `mysql_install_db` inicia o servidor `mysqld`, depois inicializa as tabelas de permissões com a seguinte configuração dos privilégios:

- O usuário `root` do MySQL é criado como um superusuário que pode fazer qualquer coisa. Conexões devem ser feitas através da máquina local.
NOTA: A senha inicial de `root` é vazia, portanto qualquer um que conectar como `root sem senha` terá direito a todos os privilégios.
- Um usuário anônimo é criado e pode fazer o que desejar com bancos de dados com nome `'test'` ou iniciando com `'test_'`. Conexões devem ser feitas da máquina local. Isto significa que usuários locais podem se conectar sem senha e serem tratados como usuários anônimos.
- Outros privilégios são negados. Por exemplo, usuários normais não podem executar `mysqladmin` ou `mysqladmin processlist`.

NOTA: Os privilégios padrões são diferentes no Windows. See [\[Windows running\]](#), page [\[undefined\]](#).

Como sua instação inicialmente é parcialmente aberta, uma das primeiras coisas que você deve fazer é especificar uma senha para o usuário `root` do MySQL. Você pode fazer isto como a seguir (perceba que a senha foi especificada utilizando a função `PASSWORD()`):

```
shell> mysql -u root mysql
mysql> UPDATE user SET Password=PASSWORD('nova_senha')
        WHERE user='root';
mysql> FLUSH PRIVILEGES;
```

Você pode, nas versões 3.22 e superiores do MySQL, utilizar a instrução `SET PASSWORD`:

```
shell> mysql -u root mysql
mysql> SET PASSWORD FOR root=PASSWORD('nova_senha');
```

Outra forma de configurar a senha é utilizando o comando `mysqladmin`:

```
shell> mysqladmin -u root password nova_senha
```

Somente usuários com acesso de escrita/atualização ao banco de dados `mysql` podem alterar a senha de outros usuários. Todos os usuários comuns (não os anônimos) podem alterar somente a própria senha com um dos comandos acima ou com `SET PASSWORD=PASSWORD('nova_senha')`.

Perceba que se você atualizar a senha na tabela `user` diretamente utilizando o primeiro método, você deve dizer ao servidor para reler as tabelas de permissões (com `FLUSH PRIVILEGES`), de outra forma a alteração não seria notificada.

Uma vez que a senha de `root` foi configurada, você deve informar a senha quando se conectar ao servidor MySQL como `root`.

Você pode desejar deixar a senha de `root` em branco para que você não precise especificá-la quando realizar configurações adicionais ou testes. Entretanto, tenha certeza de configurá-la antes de utilizar sua instalação para qualquer ambiente de produção.

Veja o script `scripts/mysql_install_db` para ver como são configurados os privilégios padrões. Você pode usar isto como uma base para ver como adicionar outros usuários.

Se você deseja que os privilégios iniciais sejam diferentes do descrito acima, é possível modificar o script `mysql_install_db` antes de executá-lo.

Para recriar as tabelas de permissões completamente, remova todos os arquivos `.frm` `.MYI` e `.MYD` no diretório contendo o banco de dados `mysql`. (Este é o diretório chamado `'mysql'` sob o diretório do banco de dados, que é listado quando você executa `mysqld --help`.) Depois execute o script `mysql_install_db`, possivelmente depois de editá-lo para criar os privilégios desejáveis.

NOTA: Para versões do MySQL mais antigas que a versão 3.22.10, você não deve apagar os arquivos `.frm`. Se você fizer isso acidentalmente, você deve voltá-los a partir de sua distribuição MySQL antes de executar `mysql_install_db`.

4.3.5 Adicionando Novos Usuários ao MySQL

Existem duas maneiras de adicionar usuários: utilizando instruções `GRANT` ou manipulando as tabelas de permissões do MySQL diretamente. O método preferido é utilizar instruções `GRANT`, porque elas são mais concisas e menos propensas a erros. See [\[GRANT\]](#), page [\[GRANT\]](#).

Existem vários programas de colaboradores como o `phpmyadmin` que pode ser utilizado para criar e administrar usuários. See [\[Contrib\]](#), page [\[Contrib\]](#).

Os exemplos abaixo mostram como usar o cliente `mysql` para configurar novos usuários. Estes exemplos assumem que privilégios são configurados de acordo com os padrões descritos na seção anterior. Isto significa que para fazer alterações, você deve se conectar na mesma máquina em que o `mysqld` está executando, você deve se conectar com o usuário `root`, e o usuário `root` deve ter os privilégios `inster` ao banco de dados `mysql` e o administrativo `reload`. Também, se você alterou a senha do usuário `root`, você deve especificá-la para os comandos `mysql` abaixo.

Você pode adicionar novos usuários utilizando instruções `GRANT`:

```
shell> mysql --user=root mysql
mysql> GRANT ALL PRIVILEGES ON *.* TO monty@localhost
      IDENTIFIED BY 'alguma_senha' WITH GRANT OPTION;
mysql> GRANT ALL PRIVILEGES ON *.* TO monty@"%"
      IDENTIFIED BY 'alguma_senha' WITH GRANT OPTION;
mysql> GRANT RELOAD,PROCESS ON *.* TO admin@localhost;
mysql> GRANT USAGE ON *.* TO dummy@localhost;
```

Estas instruções `GRANT` configuram três novos usuários:

monty Um superusuário completo que pode conectar ao servidor de qualquer lugar, mas deve utilizar uma senha `'alguma_senha'` para fazer isto. Perceba que devemos utilizar instruções `GRANT` para `monty@localhost` e `monty@"%`. Se nós não adicionarmos a entrada com `localhost`, a entrada para o usuário anônimo para `localhost` que é criada por `mysql_install_db` irá tomar precedência quando nos conectarmos da máquina local, porque ele contém um campo `Host` com um valor mais específico e também vem antes na ordenação da tabela `user`.

- admin** Um usuário que possa conectar de `localhost` sem uma senha e que é concedido os privilégios administrativos **reload** e **process**. Isto permite ao usuário a execução dos comandos `mysqladmin reload`, `mysqladmin refresh` e `mysqladmin flush-*`, bem como o `mysqladmin processlist`. Nenhum privilégio relacionado aos bancos de dados é concedido. (Depois eles podem ser adicionados utilizando instruções `GRANT` adicionais.)
- dummy** Um usuário que pode conectar sem uma senha, mas somente na máquina local. Os privilégios globais são todos configurados para 'N' --- o tipo de privilégio `USAGE` permite a criação de um usuário sem privilégios. Considera-se que você irá conceder privilégios específicos de bancos de dados mais tarde.

Também é possível adicionar a mesma informação de acesso do usuário diretamente, utilizando instruções `INSERT` e depois dizendo ao servidor para recarregar as tabelas de permissões:

```
shell> mysql --user=root mysql
mysql> INSERT INTO user VALUES('localhost','monty',PASSWORD('alguma_senha'),
    'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user VALUES('%','monty',PASSWORD('alguma_senha'),
    'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user SET Host='localhost',User='admin',
    Reload_priv='Y', Process_priv='Y';
mysql> INSERT INTO user (Host,User>Password)
    VALUES('localhost','dummy','');
mysql> FLUSH PRIVILEGES;
```

Dependendo da sua versão do MySQL, você pode precisar utilizar um número diferente de valores 'Y' acima (versões anteriores à versão 3.22.11 tem menos campos de privilégios). Para o usuário `admin`, a maior sintaxe legível de `INSERT` que está disponível a partir da versão 3.22.11 é a utilizada.

Note que para configurar um superusuário, você só precisa criar uma entrada na tabela `user` com os campos de privilégios configurados para 'Y'. Não é necessário gerar entradas nas tabelas `db` ou `host`.

Os campos de privilégios na tabela `user` não foram exibidas explicitamente na última instrução `INSERT` (para o usuário `dummy`), logo a estes campos são atribuídos o valor padrão 'N'. Isto é a mesma coisa que o `GRANT USAGE` faz.

O seguinte exemplo adiciona um usuário `custom` que pode conectar das máquinas `localhost`, `server.domain` e `whitehouse.gov`. Ele deseja acessar o banco de dados `bankaccount` somente de `localhost`, o banco de dados `expenses` somente de `whitehouse.gov` e o banco de dados `customer` de todas as três máquinas. Ele também deseja utilizar a senha `stupid` das três máquinas.

Para configurar os privilégios deste usuário utilizando instruções `GRANT`, execute estes comandos:

```
shell> mysql --user=root mysql
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ON bankaccount.*
    TO custom@localhost
    IDENTIFIED BY 'stupid';
```

```
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
      ON expenses.*
      TO custom@whitehouse.gov
      IDENTIFIED BY 'stupid';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
      ON customer.*
      TO custom@%'
      IDENTIFIED BY 'stupid';
```

A razão para que nós não concedamos privilégios para o usuário 'custom' é que desejamos fornecer ao usuário acesso ao MySQL a partir da máquina local com sockets Unix e da máquina remota 'whitehouse.gov' com TCP/IP.

Para configurar os privilégios do usuário modificando as tabelas de permissões diretamente, utilize estes comandos (perceba o FLUSH PRIVILEGES no final):

```
shell> mysql --user=root mysql
mysql> INSERT INTO user (Host,User,Password)
      VALUES('localhost','custom',PASSWORD('stupid'));
mysql> INSERT INTO user (Host,User,Password)
      VALUES('server.domain','custom',PASSWORD('stupid'));
mysql> INSERT INTO user (Host,User,Password)
      VALUES('whitehouse.gov','custom',PASSWORD('stupid'));
mysql> INSERT INTO db
      (Host,Db,User,Select_priv,Insert_priv,Update_priv,Delete_priv,
      Create_priv,Drop_priv)
      VALUES
      ('localhost','bankaccount','custom','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
      (Host,Db,User,Select_priv,Insert_priv,Update_priv,Delete_priv,
      Create_priv,Drop_priv)
      VALUES
      ('whitehouse.gov','expenses','custom','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
      (Host,Db,User,Select_priv,Insert_priv,Update_priv,Delete_priv,
      Create_priv,Drop_priv)
      VALUES('%','customer','custom','Y','Y','Y','Y','Y','Y');
mysql> FLUSH PRIVILEGES;
```

As primeiras três instruções INSERT adicionam entradas na tabela `user` que permite ao usuário `custom` conectar a partir de várias máquinas com a senha determinada, mas não concede permissões ao mesmo (todos os privilégios são configurados com o valor padrão de 'N'). As próximas três instruções INSERT adicionam entradas na tabela `db` que concedem privilégios à `custom` para os bancos de dados `bankaccount`, `expenses` e `customer`, mas só quando acessados a partir das máquinas apropriadas. Normalmente, quando as tabelas de permissões são modificadas diretamente, o servidor deve ser informado para recarregá-las (com FLUSH PRIVILEGES) para que as alterações nos privilégios tenham efeito.

Se você deseja fornecer a um usuário específico acesso de qualquer máquina em um determinado domínio, você pode utilizar uma instrução GRANT como a seguir:

```
mysql> GRANT ...
      ON *.*
```

```
TO meunomedeusuuario@"%.meudominio.com"
IDENTIFIED BY 'minhasenha';
```

Para realizar a mesma coisa modificando diretamente as tabelas de permissões, faça isto:

```
mysql> INSERT INTO user VALUES ('%.meudominio, 'meunomedeusuuario'
    PASSWORD('minhasenha'),...);
mysql> FLUSH PRIVILEGES;
```

Você pode também utilizar `xmysqladmin`, `mysql_webadmin` e mesmo `xmysql` para inserir, alterar e atualizar valores nas tabelas de permissões. Você pode encontrar estes utilitários em [Diretório de Colaboradores do Website do MySQL](#).

4.3.6 Configurando Senhas

Na maioria dos casos você deve utilizar `GRANT` para configurar seus usuários e senhas, portanto, as informações exibidas a seguir são aplicadas somente para usuários avançados. See [\[GRANT\]](#), page [\[GRANT\]](#).

Os exemplos nas seções precedentes ilustram um princípio importante: quando você armazena uma senha não-vazia utilizando `INSERT` ou `UPDATE` você deve utilizar a função `PASSWORD()` para criptografá-la. Isto é porque a tabela `user` armazena senhas na forma criptografada, e não como texto puro. Se você esquecer deste fato, é provável que você possa tentar configurar senhas desta forma:

```
shell> mysql -u root mysql
mysql> INSERT INTO user (Host,User,Password)
    VALUES('%', 'jeffrey', 'biscuit');
mysql> FLUSH PRIVILEGES;
```

O resultado é que o valor `'biscuit'` é armazenado como a senha na tabela `user`. Quando o usuário `jeffrey` tentar se conectar ao servidor utilizando esta senha, o cliente `mysql` a criptografa utilizando a função `PASSWORD()`, gerando um vetor de autenticação baseado em uma senha **criptografada** e um número randômico, obtido do servidor, e envia o resultado ao servidor. O servidor usa o valor do campo `password` na tabela `user` (que é o valor `'biscuit'` **não criptografado**) para realizar os mesmos cálculos e comparar os resultados. A comparação falha e o servidor rejeita a conexão:

```
shell> mysql -u jeffrey -pbiscuit test
Access denied
```

As senhas devem ser criptografadas quando elas são inseridas na tabela `user`, portanto a instrução `INSERT` deveria ter sido informada no seguinte formato:

```
mysql> INSERT INTO user (Host,User,Password)
    VALUES('%', 'jeffrey', PASSWORD('biscuit'));
```

Você deve também utilizar a função `PASSWORD()` quando utilizar instruções `SET PASSWORD`:

```
mysql> SET PASSWORD FOR jeffrey@"%" = PASSWORD('biscuit');
```

Se você configurar senhas utilizando a instrução `GRANT ... IDENTIFIED BY` ou o comando `mysqladmin password`, a função `PASSWORD()` é desnecessária. Ambos tomam o cuidado de criptografar a senha para você, então você deve especificar a senha `'biscuit'` desta forma:

```
mysql> GRANT USAGE ON *.* TO jeffrey@"%" IDENTIFIED BY 'biscuit';
```

ou

```
shell> mysqladmin -u jeffrey password biscuit
```

NOTA: PASSWORD() não realiza criptografia de senhas da mesma forma em que as senhas do Unix são criptografadas. Você não deve assumir que se sua senha Unix e a do MySQL são as mesmas, o resultado de PASSWORD() irá resultar no mesmo valor criptografado como o armazenado no arquivo de senhas do Unix. See [\(undefined\) \[User names\]](#), page [\(undefined\)](#).

4.3.7 Mantendo Sua Senha Segura

Não é aconselhável especificar uma senha de uma forma que a exponha e possa ser descoberta por outros usuários. Os métodos que você pode usar para especificar sua senha quando executar programas clientes são listados abaixo, juntamente com as determinações de riscos de cada método:

- Nunca forneça a um usuário normal acesso à tabela `mysql.user`. O conhecimento de uma senha criptografada possibilita a conexão como este usuário. As senhas só estão embaralhadas para que não seja possível chegar à senha real que foi usada (acontece muito a utilização de senhas similares em outras aplicações).
- Uso da opção `-psua_senha` ou `--password=sua_senha` na linha de comando. Isto é conveniente mas inseguro, porque sua senha se torna visível para programas de informação do sistema (como no `ps`) que pode ser chamado por outros usuários para exibir linhas de comando. (clientes MySQL normalmente gravam zeros em cima do argumento da linha de comando durante sua sequência de inicialização, mas ainda existe um breve intervalo no qual o valor está visível.)
- Uso das opções `-p` ou `--password` (sem especificar o valor `sua_senha`). Neste caso, o programa cliente solicita a senha do terminal:

```
shell> mysql -u user_name -p
Enter password: *****
```

Os caracteres '*' representam sua senha.

É mais seguro digitar sua senha desta forma do que especificá-la na linha de comando porque ela não fica visível a outros usuários. Entretanto este método de digitar uma senha é válido somente para programas que você executa de forma interativa. Se você deseja chamar um cliente de um script que não execute interativamente, não existirá oportunidade de digitar a senha do terminal. Em alguns sistemas, você pode descobrir que a primeira linha do seu script é lida e interpretada (incorretamente) como sua senha!

- Armazenar a sua senha em um arquivo de configuração. Por exemplo, você pode listar sua senha na seção `[client]` do arquivo `.my.cnf` no seu diretório home:

```
[client]
password=sua_senha
```

Se você armazenar sua senha em um arquivo `.my.cnf`, o arquivo não pode ser lido por seu grupo ou pelos outros usuários. Tenha certeza que o modo de acesso do arquivo é 400 ou 600 See [\(undefined\) \[Option files\]](#), page [\(undefined\)](#).

- Você pode armazenar sua senha na variável de ambiente `MYSQL_PWD`, mas este método deve ser considerado extremamente inseguro e não deve ser usado. Algumas versões de `ps` incluem uma opção para exibir o ambiente de processos em execução; sua senha estaria em texto puro para a leitura para todos os usuários. Mesmo em sistemas sem

esta versão do `ps`, seria imprudência assumir que não existe outro método para observar o ambiente de processos. See [\[Environment variables\]](#), page [\[undefined\]](#).

Em resumo, os métodos mais seguros seriam que o programa cliente solicitasse a senha ou especificar a senha em um arquivo `‘.my.cnf’` corretamente protegido.

4.4 Prevenção de Disastres e Recuperação

4.4.1 Backups dos Bancos de Dados

Como as tabelas do MySQL são armazenadas como arquivos, é mais fácil realizar um backup. Para obter um backup consistente, faça um `LOCK TABLES` nas tabelas relevantes seguido por `FLUSH TABLES` para as tabelas. See [\[LOCK TABLES\]](#), page [\[undefined\]](#). See [\[FLUSH\]](#), page [\[undefined\]](#). Você só precisa de um bloqueio de leitura; isto possibilita outras threads a continuarem a pesquisar nas tabelas enquanto você copia os arquivos no diretório do banco de dados. O `FLUSH TABLE` é necessário para garantir que todas as páginas ativas de índices serão escritas em disco antes de iniciar o backup.

Se você desejar realizar um backup ao nível da linguagem SQL de um tabela, você pode utilizar `SELECT INTO OUTFILE` ou `BACKUP TABLE`. See [\[SELECT\]](#), page [\[undefined\]](#). See [\[BACKUP TABLE\]](#), page [\[undefined\]](#).

Outra maneira de efetuar um backup de um banco de dados é utilizar o programa `mysqldump` ou o script `mysqlhotcopy`. See [\[mysqldump\]](#), page [\[undefined\]](#). See [\[mysqlhotcopy\]](#), page [\[undefined\]](#).

1. Fazer um backup completo dos bancos de dados:

```
shell> mysqldump --tab=/caminho/para/algum/dir --opt --full
```

ou

```
shell> mysqlhotcopy database /caminho/para/algum/dir
```

Você também pode simplesmente copiar os arquivos das tabelas (`*.frm`, `*.MYD`) e os arquivos `*.MYI`) quando o servidor não estiver atualizando nada. O script `mysqlhotcopy` utiliza este método.

2. Interrompa o `mysqld` caso ele esteja em execução, depois inicie-o com a opção `--log-update[=nome_arquivo]`. See [\[Update log\]](#), page [\[undefined\]](#). Os arquivos de log atualizados fornecem a informação necessária para replicar alterações ao banco de dados que forem feitas depois do ponto em que você executou `mysqldump`.

Se você necessita restaurar alguma coisa, tente primeiro recuperar suas tabelas utilizando `REPAIR TABLE` ou `myisamchk -r`. Isto deve funcionar em 99.9% de todos os caso, Se o `myisamchk` falhar, tente o seguinte procedimento: (Isto só irá funcionar se você iniciou o MySQL com `--log-update`, See [\[Update log\]](#), page [\[undefined\]](#)):

1. Restaure o backup original feito com o `mysqldump`.
2. Execute o seguinte comando para re-executar as atualizações armazenadas no log binário:

```
shell> mysqlbinlog hostname-bin.[0-9]* | mysql
```

Se você estiver utilizando o log atualizado, pode utilizar:

```
shell> ls -l -t -r hostname.[0-9]* | xargs cat | mysql
```

O comando `ls` é usado para obter todos os arquivos de log na ordem correta.

Você pode também fazer backups seletivos com `SELECT * INTO OUTFILE 'nome_arquivo' FROM nome_tabela` e restaurar com `LOAD DATA INFILE 'nome_arquivo' REPLACE...`. Para evitar registros duplicados, você precisará de um chave `PRIMARY KEY` ou uma `UNIQUE` na tabela. A palavra chave `REPLACE` substitui os antigos registros com os novos quando um novo registro duplica um antigo registro em uma chave de valores únicos.

Se você tiver problemas de performance realizando backups no seu sistema, você pode resolver isto configurando uma replicação e fazendo os backups na máquina escrava no lugar da master. See [\[Replication Intro\]](#), page [\[undefined\]](#).

Se você estiver utilizando um sistema de arquivos Veritas, você pode fazer:

1. Executar em um cliente (perl ?) `FLUSH TABLES WITH READ LOCK`
2. Bifurcar uma shell ou executar em outro cliente `mount vfxs snapshot`.
3. Executar no primeiro cliente `UNLOCK TABLES`
4. Copiar arquivos do snapshot
5. Desmontar snapshot

4.4.2 Sintaxe de BACKUP TABLE

```
BACKUP TABLE nome_tabela[,nome_tabela...] TO '/caminho/para/diretório/backup'
```

Faz uma cópia de todos os arquivos de tabela para o diretório de backup que é o mínimo necessário para restaurá-lo. Atualmente só funciona para tabelas `MyISAM`. Para tabela `MyISAM`, copia os arquivos `.frm` (definições) e `.MYD` (dados). O arquivo de índice pode ser reconstruído a partir destes dois.

Antes de utilizar este comando, por favor veja See [\[Backup\]](#), page [\[undefined\]](#).

Durante o backup, o bloqueio de leitura (read lock) será usado para cada tabela, uma de cada vez, à medida que o backup é realizado. Se você deseja fazer backup de diversas tabelas como um snapshot, você deve primeiro usar `LOCK TABLES` obtendo um bloqueio de leitura para cada tabela no grupo.

O comando retorna uma tabela com as seguintes colunas:

Coluna	Valor
Table	Nome da Tabela
Op	Sempre "backup"
Msg_type	Um dos seguintes: <code>status</code> , <code>error</code> , <code>info</code> ou <code>warning</code> .
Msg_text	A mensagem.

Note que o comando `BACKUP TABLE` está disponível somente no MySQL versão 3.23.25 e posterior.

4.4.3 Sintaxe de RESTORE TABLE

```
RESTORE TABLE nome_tabela[,nome_tabela...] FROM '/caminho/para/diretório/backup'
```

Restaura as tabelas utilizando o backup feito com `BACKUP TABLE`. Tabelas existentes não serão reescritas - se você tentar restaurar sobre uma tabela existente, obterá um erro. A restauração demora mais tempo do que o backup pois é necessário reconstruir o índice.

Quanto mais chaves tiver, mais demorado será. Como no comando `BACKUP TABLE`, atualmente só funciona com tabelas MyISAM.

O comando retorna uma tabela com as seguintes colunas:

Coluna	Valor
Table	Nome da Tabela
Op	Sempre “restore”
Msg_type	Um dos seguintes: <code>status</code> , <code>error</code> , <code>info</code> ou <code>warning</code> .
Msg_text	A mensagem.

4.4.4 Sintaxe de CHECK TABLE

```
CHECK TABLE nome_tabela[,nome_tabela...] [opção [opção...]]
```

```
opção = QUICK | FAST | MEDIUM | EXTENDED | CHANGED
```

`CHECK TABLE` funciona somente em tabelas MyISAM. Em tabelas MyISAM é a mesma coisa que executar `myisamchk -m nome_tabela` na tabela.

Se você não especificar nenhuma opção, `MEDIUM` é usado.

Verifica se existem erros na(s) tabela(s). Para as tabelas MyISAM as estatísticas das chaves são atualizadas. O comando retorna uma tabela com as seguintes colunas:

Coluna	Valor
Table	Nome da Tabela.
Op	Sempre “check”.
Msg_type	Um dos seguintes: <code>status</code> , <code>error</code> , <code>info</code> , or <code>warning</code> .
Msg_text	A mensagem.

Note que você pode obter várias linhas de informações para cada tabela conferida. A última linha irá ser do tipo `Msg_type status` e normalmente deve estar `OK`. Se você não obteve `OK` ou `Not checked`, deve ser executado, normalmente, um reparo da tabela. See [\[Table maintenance\], page `<undefined>`](#). `Not checked` significa que a tabela de um dado TIPO disse ao MySQL que não havia nenhuma necessidade de verificá-la.

Os diferentes tipos de consistências continuam para os seguintes:

Tipo	Significado
QUICK	Não busca os registros verificando ligações incorretas.
FAST	Só confere tabelas que não foram fechadas corretamente.
CHANGED	Só verifica as tabelas que foram alteradas desde a última conferência ou que não foram fechadas corretamente.
MEDIUM	Busca os registros para verificanado que ligações removidas estão ok. Isto também calcula uma chave de conferência para os registros e verifica isto com um checksum calculado para as chaves.
EXTENDED	Faz uma busca completa nas chaves para todas as chaves em cada registro. Isto assegura que a tabela está 100% consistente, mas pode demorar muito tempo para executar!

Para tabelas MyISAM de tamanho dinâmico, uma verificação iniciada sempre fará uma verificação `MEDIUM`. Para registros de tamanho estático nós saltamos a busca de registros para `QUICK` e `FAST` já que os registros estão raramente corrompidos.

Você pode combinar opções de consistência como em:

```
CHECK TABLE test_table FAST QUICK;
```

O qual só fará uma verificação rápida na tabela se ela não for fechada corretamente.

NOTA: em alguns casos `CHECK TABLE` irá alterar a tabela! Isto acontece se a tabela estiver marcada como 'corrupted' (corrompida) ou 'not closed properly' (não foi fechada corretamente) mas o `CHECK TABLE` não encontrar nenhum problema na tabela. Neste caso, `CHECK TABLE` irá marcar a tabela como ok.

Se uma tabela estiver corrompida, é preferível que seja um problema nos índices e não na parte de dados. Todos os tipos de consistência acima sempre confere os índices e deve então encontrar a maioria dos erros.

Se você só quiser conferir uma tabela que acredita estar ok, você não deve utilizar nenhuma opção para o comando check ou utilizar a opção `QUICK`. O último deve ser utilizado quando você estiver com pressa e o risco do `QUICK` não encontrar um erro no arquivo de dados for mínimo (Na maioria dos casos o MySQL pode encontrar, sob utilização normal, qualquer erro no arquivo de dados. Se isto ocorrer, então a tabela será marcada como 'corrupted', neste caso a tabela não poderá ser utilizada até ser reparada).

`FAST` e `CHANGED` são normalmente chamados a partir de um script (um exemplo é ser executado a partir do cron) Se você desejar conferir suas tabelas de tempos em tempos. Na maioria dos casos o `FAT` é uma opção melhor que `CHANGED`. (O único caso em que isto não acontece é se houver suspeitas de bug no código do MyISAM.).

`EXTENDED` deve ser utilizado somente depois de ter executado um check normalmente, mas continuar obtendo erros de uma tabela quando o MySQL tenta atualizar um registro ou encontrar um registro pela chave (isto seria muito difícil ocorrer caso uma conferência normal tenha executado com sucesso!).

Algumas coisas relatadas pela verificação de tabelas, não podem ser corrigidas automaticamente:

- `Found row where the auto_increment column has the value 0.`

Isto significa que você possui na tabela um registro onde o campo índice que utiliza o recurso `auto_increment` contem o valor 0. (É possível criar um registro onde a coluna de auto incremento seja 0 definindo explicitamente 0 em uma instrução `UPDATE`)

Isto não é exatamente um erro, mas pode causar problemas se você decidir descarregar a tabela e restaurá-la ou executar um `ALTER TABLE` na tabela. Neste caso a coluna de auto incremento irá alterar seu valor, de acordo com as regras das colunas de auto incremento, que pode causar problemas como um erro de chave duplicada.

Para se livrar do alerta, basta executar uma instrução `UPDATE` para configurar a coluna para algum outro valor diferente de 0.

4.4.5 Sintaxe de REPAIR TABLE

```
REPAIR TABLE nome_tabela[,nome_tabela...] [QUICK] [EXTENDED]
```

`REPAIR TABLE` funciona somente em tabelas MyISAM e é a mesma coisa que executar `myisamchk -r nome_tabela` na tabela.

Normalmente você nunca deve executar este comando, mas se um desastre ocorrer você vai precisar recuperar seus dados de uma tabela MyISAM utilizando `REPAIR TABLE`. Se as suas tabelas estiverem muito corrompidas, você deve encontrar a razão para isto! See [\[Crashing\]](#), page [\[undefined\]](#). See [\[MyISAM table problems\]](#), page [\[undefined\]](#).

REPAIR TABLE repara uma tabela possivelmente corrompida. O comando retorna uma tabela com as seguintes colunas:

Coluna	Valor
Table	Nome da Tabela
Op	Sempre "repair"
Msg_type	Um dos seguintes: status , error , info ou warning .
Msg_text	A mensagem.

Perceba que você pode obter várias linhas de informações para cada tabela recuperada. A última linha será de `Msg_type status` e normalmente deve exibir OK. Se o retorno não for OK, você pode tentar reparar a tabela com `myisamchk -o`, já que REPAIR TABLE ainda não implementa todas as opções de `myisamchk`. Futuramente iremos torná-lo mais flexível.

Se o parâmetro QUICK for especificado o MySQL tentará REPARAR somente a árvore de índices.

Se você utilizar EXTENDED o MySQL criará o índice, registro a registro em vez de criar um índice de uma vez com ordenação; Isto pode ser melhor que a ordenação em chaves de tamanho fixo se você tiver grandes chaves do tipo `char()` que compactam muito bem.

4.4.6 Utilizando myisamchk para Manutenção de Tabelas e Recuperação em Caso de Falhas

A partir do MySQL versão 3.23.13 você pode mandar verificar as tabelas MyISAM com o comando CHECK TABLE. See [\[CHECK TABLE\]](#), page [\[CHECK TABLE\]](#). Pode-se reparar tabelas com o comando REPAIR TABLE. See [\[REPAIR TABLE\]](#), page [\[REPAIR TABLE\]](#).

Para verificar/reparar tabelas MyISAM (.MYI e .MYD) você deve utilizar o utilitário `myisamchk`. Para consistir/reparar tabelas ISAM (.ISM e .ISD) você deve usar o utilitário `isamchk`. See [\[Tipos de Tabelas\]](#), page [\[Tipos de Tabelas\]](#).

No texto a seguir iremos comentar sobre o `myisamchk`, mas tudo também se aplica ao antigo `isamchk`.

Você pode utilizar o utilitário `myisamchk` para obter informações sobre suas tabelas de bancos de dados, verificá-las, repará-las ou otimizá-las. As seguintes seções descrevem como executar `myisamchk` (incluindo uma descrição de suas opções), como montar um calendário de manutenção, e como utilizar o `myisamchk` para executar suas várias funções.

Você pode, na maioria dos casos, utilizar o comando OPTIMIZE TABLES para otimizar e reparar tabelas, mas não é tão rápido e confiável (no caso real de erros fatais) como o `myisamchk`. Por outro lado, OPTIMIZE TABLE é mais fácil de usar e você não tem que se preocupar com a recarrega das tabelas. See [\[OPTIMIZE TABLE\]](#), page [\[OPTIMIZE TABLE\]](#).

Mesmo os reparos realizados pelo `myisamchk` são bastante seguros, porém é sempre uma boa idéia fazer um backup dos dados ANTES de realizar um reparo (ou qualquer coisa que fará grandes alterações em alguma tabela)

4.4.6.1 Sintaxe de chamada do myisamchk

`myisamchk` é chamado desta forma:

```
shell> myisamchk [opções] nome_tabela
```

As opções especificam o que você deseja que o `myisamchk` faça. Elas são descritas abaixo. (Você também pode obter a lista das opções com `myisamchk --help`.) Sem opções, o `myisamchk` simplesmente checa sua tabela. Para obter maiores informações ou dizer ao `myisamchk` para tomar ações corretivas, especifique as opções descritas abaixo e nas seções seguintes.

`nome_tabela` é o nome da tabela do banco de dados que você deseja verificar/reparar. Se você executar o `myisamchk` em algum lugar diferente do diretório do banco de dados, você deve especificar o caminho para o arquivo, porque `myisamchk` não faz idéia de onde seu banco de dados se encontra. Na verdade, `myisamchk` não se importa se os arquivos estão localizados em um diretório de banco de dados; você pode copiar os arquivos que correspondem a uma tabela de banco de dados em outra localização e realizar neste outro lugar as operações corretivas.

Você pode nomear várias tabelas na linha de comando do `myisamchk` se você desejar. Você também pode especificar um nome como um arquivo de índice (com o sufixo `‘.MYI’`), que lhe permite especificar todas as tabelas em um diretório utilizando o padrão `‘*.MYI’`. Por exemplo, se você está em um diretório de banco de dados, você pode checar todas as tabelas no diretório desta forma:

```
shell> myisamchk *.MYI
```

Se você não estiver no diretório do banco de dados, você pode verificar todas as tabelas existentes especificando o caminho para o diretório:

```
shell> myisamchk /caminho/para/banco_de_dados/*.MYI
```

Você pode verificar todas as tabelas em todos os bancos de dados especificando um meta-caracter com o caminho para o diretório de banco de dados do MySQL:

```
shell> myisamchk /caminho/para/diretório_dados/*/*.MYI
```

A maneira recomendada para conferir todas as tabelas rapidamente é:

```
myisamchk --silent --fast /caminho/para/diretório_dados/*/*.MYI
isamchk --silent /caminho/para/diretório_dados/*/*.ISM
```

Se você quiser conferir todas as tabelas e reparar todas que estiverem corrompidas, pode utilizar linha a seguir:

```
myisamchk --silent --force --fast --update-state -O key_buffer=64M -O sort_
buffer=64M -O read_buffer=1M -O write_buffer=1M /caminho/para/diretório_
dados/*/*.MYI
isamchk --silent --force -O key_buffer=64M -O sort_buffer=64M -O read_buffer=1M -
O write_buffer=1M /caminho/para/diretório_dados/*/*.ISM
```

A linha acima assume que você tem mais de 64 MB de memória livre.

Perceba que se você obter um erro do tipo:

```
myisamchk: warning: 1 clients is using or hasn't closed the table properly
```

Isto significa que você está tentando verificar uma tabela que está sendo atualizada por outro programa (como o servidor `mysqld`) que ainda não fechou o arquivo ou que finalizou sem fechar o arquivo corretamente.

Se seu `mysqld` está em execução, você deve forçar o sincronismo e fechamento de todas as tabelas com `FLUSH TABLES` e assegurar que ninguém mais esteja utilizando as tabelas quando for executar o `myisamchk`. No MySQL versão 3.23 a forma mais simples de evitar este problema é utilizar `CHECK TABLE` no lugar de `myisamchk` para verificar as tabelas.

4.4.6.2 Opções Genéricas do `myisamchk`

`myisamchk` suporta as seguintes opções.

`-#` ou `--debug=debug_options`

Saída do log de depuração. A string `debug_options` geralmente é `'d:t:o,nomearquivo'`.

`-?` ou `--help`

Exibe uma mensagem de ajuda e sai.

`-O var=opção, --set-variable var=opção`

Configura o valor de uma variável. As variáveis possíveis e seus valores padrões para o `myisamchk` podem ser examinados com `myisamchk --help`

<code>key_buffer_size</code>	523264
<code>read_buffer_size</code>	262136
<code>write_buffer_size</code>	262136
<code>sort_buffer_size</code>	2097144
<code>sort_key_blocks</code>	16
<code>decode_bits</code>	9

`sort_buffer_size` é utilizado quando as chaves são reparadas pela ordenação das chaves, que é o caso normal quando você utiliza `--recover`.

`key_buffer_size` é utilizado quando você estiver conferindo a tabela com `--extended-check` ou quando as chaves são reparadas inserindo-as registro a registro na tabela (como com inserts normais). O reparo através de buffer de chaves (key buffer) é utilizado nos seguintes casos:

- Se você utilizar `--safe-recover`.
- Se você estiver utilizando um índice `FULLTEXT`.
- Se os arquivos temporários necessários para ordenar as chaves forem maior que o dobro do tamanho de quando se criasse o arquivo de chaves diretamente. Isto é o caso quando se tem chaves `CHAR`, `VARCHAR` ou `TEXT` tao grandes quanto necessário pela ordenação para armazenar todas as chaves durante o processo. Se você tiver muito espaço temporário e puder forçar o `myisamchk` a reparar por ordenação você pode utilizar a opção `--sort-recover`.

Reparação através do buffer de chaves (key buffer) economiza muito mais espaço em disco do que utilizando ordenação, mas é muito mais lenta.

Se você deseja uma reparação mais rápida, configure as variáveis acima para cerca de 1/4 da sua memória disponível. Você pode configurar as variáveis para valores altos, pois somente um dos buffers acima será utilizado a cada vez.

`-s` ou `--silent`

Modo discreto ou silencioso. Escreve a saída somente quando um erro ocorre. Você pode utilizar `-s` duas vezes (`-ss`) para deixar o `myisamchk` mais silencioso.

`-v` ou `--verbose`

Modo prolixo. Gera mais informação de saída. Ele pode ser utilizado com `-d` e `-e`. Utilize `-v` múltiplas vezes (`-vv`, `-vvv`) para gerar mais saída!

-V ou --version

Exibe a versão do `myisamchk` e sai.

-w ou, --wait

No lugar de gerar um erro se a tabela estiver bloqueada, espere até que a tabela fique livre antes de continuar. Perceba que se você estiver utilizando `mysqld` na tabela com `--skip-locking`, a tabela só pode ser trancada por outro comando `myisamchk`.

4.4.6.3 Opções de Verificação do `myisamchk`

-c ou --check

Confere por erros na tabela. Esta é a operação padrão se você não estiver utilizando opções que a anulam.

-e ou --extend-check

Verifica a tabela de forma completa (que é bastante lento se você tiver vários índices). Esta opção deve ser usada somente em casos extremos. Normalmente, `myisamchk` ou `myisamchk --medium-check` deve, na maioria dos casos, estar apto a encontrar quaisquer erros na tabela.

Se você estiver utilizando `--extended-check` e tiver muita memória, você deve aumentar um pouco o valor de `key_buffer_size`!

-F ou --fast

Verifica apenas tabelas que não foram fechadas corretamente.

-C ou --check-only-changed

Verifica apenas tabelas que foram alteradas desde a última verificação.

-f ou --force

Reinicia o `myisamchk` com `-r` (reparos) na tabela, se `myisamchk` encontrar quaisquer erros na tabela.

-i ou --information

Exibe informações e estatísticas sobre a tabela que estiver sendo verificada.

-m ou --medium-check

Mais rápido que `extended-check`, mas encontra somente 99.99% de todos os erros. Deve, entretando, ser bom o bastante para a maioria dos casos.

-U ou --update-state

Armazena no arquivo `‘.MYI’` quando a tabela foi verificada e se a tabela falhou. Isto deve ser utilizado para obter o benefício integral da opção `--check-only-changed`, mas você não deve utilizar esta opção se o servidor `mysqld` esta usando a tabela e o `mysqld` esta sendo executado com `--skip-locking`.

-T ou --read-only

Não marca as tabelas como verificadas. Isto é útil se você utiliza o `myisamchk` para verificar uma tabela que esteja em uso por alguma outra aplicação que não utiliza bloqueios (como no `mysqld --skip-locking`).

4.4.6.4 Opções de Reparos do `myisamchk`

As seguintes opções são usadas se você iniciar o `myisamchk` com `-r` ou `-o`:

`-D #` ou `--data-file-length=#`

Tamanho máximo do arquivo de dados (ao recriar arquivos de dados quando eles estão 'cheios').

`-e` ou `--extend-check`

Tenta recuperar todos registros possíveis do arquivo de dados. Normalmente isto irá encontrar também várias linhas com lixo. Não utiliza esta opção a menos que esteja em desespero total.

`-f` ou `--force`

Sobrescreve antigos arquivos temporários (`nome_tabela,TMD`) em vez de abortar.

`-k #` ou `keys-used=#`

Se você estiver utilizando ISAM, diz ao manipulador de tabelas do ISAM para atualizar somente os primeiros `#` índices. Se você estiver utilizando MyISAM, informa quais chaves usar, onde cada bit seleciona uma chave (a primeira chave possui o bit 0). Isto pode ser utilizado para inserções mais rápidas! Índices desativados podem ser reativados utilizando `myisamchk -r`.

`-l` ou `--no-symlinks`

Não segue links simbólicos. Normalmente o `myisamchk` repara a tabela para qual um link simbólico aponta. Esta opção não existe no MySQL 4.0 pois o MySQL 4.0 não irá remover links simbólicos durante os reparos.

`-r` ou `--recover`

Pode concertar quase tudo exceto chaves únicas que não são únicas (Que é um erro extremamente indesejável com tabelas ISAM/MyISAM). Se você deseja recuperar uma tabela, esta é primeira opção a ser tentada. Somente se o `myisamchk` relatar que a tabela não pode ser recuperada pelo `-r` você deve tentar então a opção `-o`. (Perceba que no caso indesejável de `-r` falhar, o arquivo de dados continuará intacto.) Se você possui muita memória, você deve aumentar o tamanho de `sort_buffer_size`!

`-o` ou `--safe-recover`

Utiliza um antigo método de recuperação (le através de todos registros na ordem e atualiza todas as árvores de índices baseado nos registros encontrados); esta opção é muito mais lenta que `-r`, mas pode tratar vários casos indesejáveis que o `-r` não consegue tratar. Este método de recuperação também utiliza muito menos espaço em disco que `-r`. Normalmente sempre se deve tentar, primeiro, um reparo com `-r`, e somente se ele falhar, usar `-o`.

Se você possui muita memória, você deve aumentar o tamanho de `sort_buffer_size`!

`-n` ou `--sort-recover`

Força o uso de ordenação do `myisamchk` para resolver as chaves mesmo se os arquivos temporários forem muito grandes. Isto não terá efeito algum se você tiver chaves textuais na tabela.

- `--character-sets-dir=...`
Diretório onde conjuntos de caracteres são armazenados.
- `--set-character-set=name`
Altere o conjunto de caracteres usado pelo índice
- `.t` ou `--tmpdir=path`
Caminho para armazenar arquivos temporários. Se isto não for configurado, `myisamchk` irá usar a variável de ambiente `TMPDIR` para isto.
- `-q` ou `--quick`
Reparo rápido sem modificar o arquivo de dados. Pode ser fornecido um segundo `-q` para forçar o `myisamchk` para modificar o arquivo de dados original no caso de chaves duplicadas.
- `-u` ou `--unpack`
Descompacta arquivo empacotado com o `myisampack`.

4.4.6.5 Outras Opções de `myisamchk`

Outras ações que o `myisamchk` pode fazer, além de reparar e conferir tabelas:

- `-a` ou `--analyze`
Analisa a distribuição das chaves. Isto habilita o otimizador de joins para a melhor escolha da ordem em deve-se unir as tabelas e quais chaves devem ser usadas: `myisamchk --describe --verbose nome_tabela'` ou utilizar `SHOW KEYS` no MySQL.
- `-d` ou `--description`
Exibe algumas informações sobre a tabela.
- `-A` ou `--set-auto-increment [=valor]`
Força o `auto_increment` a iniciar a partir deste ponto ou de um valor mais alto. Se nenhum valor for fornecido, então é configurado o próximo valor de auto incremento até o valor mais alto já utilizado da chave + 1.
- `-S` ou `--sort-index`
Ordena os blocos da árvore de índices na ordem: mais alto ao mais baixo. Isto irá otimizar pesquisas e tornará a exploração de tabela por chave mais rápida.
- `-R` ou `--sort-records=#`
Ordena registros de acordo com um índice. Isto deixa seus dados muito mais localizados e pode acelerar operações `SELECTs` e `ORDER BY` neste índice. (Ele pode ser muito lento para ordenar na primeira vez!) Para encontrar os números de índices de uma tabela, utilize `SHOW INDEX`, que exibe os índices de uma tabela na mesma ordem que o `myisamchk` os enxerga. Índices são numerados a partir do 1.

4.4.6.6 Uso de Memória do `myisamchk`

Alocação de memória é importante quando você executa o `myisamchk`. `myisamchk` não utiliza mais memória do que você especifica com a opção `-O`. Se você irá utilizar o `myisamchk`

em grandes arquivos, você deve decidir primeiro quanta memória deseja usar. O valor padrão é utilizar somente 3MB para correções. Utilizando valores maiores, o `myisamchk` pode operar mais rapidamente. Por exemplo, se você tiver mais que 32M de memória RAM, você pode utilizar opções tais como esta (em adição às várias outras que podem ser especificadas):

```
shell> myisamchk -O sort=16M -O key=16M -O read=1M -O write=1M ...
```

Utilizando `-O sort=16M` provavelmente é suficiente para a maioria dos casos.

Certifique-se que o `myisamchk` utiliza arquivos temporários em `TMPDIR`. Se `TMPDIR` aponta para um sistema de arquivos em memória, você pode facilmente obter erros de memória. Se isto acontecer, configure `TMPDIR` para apontar para algum diretório com mais espaço e reinicie o `myisamchk`.

Quando reparando, o `myisamchk` também precisará de bastante espaço em disco:

- Dobra-se o tamanho do arquivo de registros (o original e uma cópia). Este espaço não é necessário se for feito um reparo com `--quick`, já que neste caso somente o arquivo de índices será recriado. Este espaço é necessário no mesmo disco que se encontra o arquivo de registros original!
- Espaço para o novo arquivo de índice que substitui o antigo. O arquivo de índices antigo é truncando no início, portanto, normalmente este espaço é ignorado. Este espaço é necessário no mesmo disco que o arquivo de índice original!
- Quando utilizando `--recover` ou `--sort-recover` (mas não quando usando `--safe-recover`, será necessário espaço para um buffer de ordenação de: $(\text{maior_chave} + \text{tamanho_do_ponteiro_de_registro}) * \text{número_de_registros} * 2$. Você pode conferir o tamanho das chaves e o `tamanho_do_ponteiro_de_registro` com `myisamchk -dv tabela`. Este espaço é alocado no disco temporário (especificado por `TMPDIR` ou `--tmpdir=#`).

Se você tiver um problema com espaço em disco durante o reparo, pode-se tentar usar `--safe-recover` em vez de `--recover`.

4.4.6.7 Uso do `myisamchk` para Recuperação em Caso de Falhas

Se você executa o `mysqld` com a opção `--skip-locking` (que é o padrão em alguns sistemas, como o Linux), você não pode utilizar com segurança o `myisamchk` para conferir uma tabela se o `mysqld` estiver utilizando a mesma tabela. Se você pode ter certeza que ninguém está acessando as tabelas através do `mysqld` enquanto você executa o `myisamchk`, você só tem que executar o `mysqladmin flush-tables` antes de iniciar a verificação das tabelas. Se você não tem certeza, então você deve desligar o `mysqld` enquanto verifica as tabelas. Se você executa o `myisamchk` enquanto o `mysqld` estiver atualizando as tabelas, você pode obter um altera que a tabela está corrompida mesmo se não estiver.

Se você não estiver utilizando `--skip-locking`, pode usar o `myisamchk` para conferir as tabelas a qualquer hora. Enquanto você faz isto, todos os clientes que tentarem atualizar a tabela irão esperar até que o `myisamchk` esteja pronto, antes de continuar.

Se você utilizar o `myisamchk` para reparar ou otimizar tabelas, você **DEVE** sempre assegurar que o servidor `mysqld` não esteja utilizando a tabela (Isto também aplica se você utiliza `--skip-locking`). Se você não desligar o `mysql`, você deve, pelo menos, fazer um `mysqladmin flush-tables` antes de executar o `myisamchk`.

Este capítulo descreve como checar e lidar com dados corrompidos nos bancos de dados MySQL. Se suas tabelas corromperem com frequência deve ser encontrada a razão para isto! See [\[Falhas\]](#), page [\[Falhas\]](#).

A seção de tabelas MyISAM contém motivos do porque uma tabela pode estar corrompida. See [\[MyISAM table problems\]](#), page [\[MyISAM table problems\]](#).

Quando se realizar recuperação devido a falhas, é importante entender que cada tabela `nome_tabela` em um banco de dados corresponde a tres arquivos no diretório do banco de dados:

Arquivo	Propósito
'nome_tabela.frm'	Arquivo com definições da tabela (form)
'nome_tabela.MYD'	Arquivo de dados
'nome_tabela.MYI'	Arquivo de índices

Cada um destes três tipos de arquivos está sujeito a corrupção de várias formas, mas problemas ocorrem mais frequentemente em arquivos de dados e índices.

O `myisamchk` trabalha criando uma cópia do arquivo de dados `.MYD` linha a linha. Ele termina o estágio de reparos removendo o antigo arquivo `.MYD` e renomeando o novo arquivo com nome original. Se for utilizada a opção `--quick`, `myisamchk` não cria um arquivo `.MYD` temporário, mas assume que o arquivo `.MYD` está correto e somente gera um novo arquivo índice sem mexer no arquivo de dados. Isto é seguro, pois o `myisamchk` detecta automaticamente se o arquivo `.MYD` está corrompido e aborda o reparo neste caso. Você pode também fornecer duas opções `--quick` para o `myisamchk`. Neste caso, o `myisamchk` não aborta em alguns erros (como chaves duplicadas) mas tenta resolvê-los modificando o arquivo `.MYD`. Normalmente o uso de duas opções `--quick` é útil somente se você tiver muito pouco espaço em disco para realizar um reparo normal. Neste caso você deve pelo menos fazer um backup antes de executar o `myisamchk`.

4.4.6.8 Como Verificar Erros em Tabelas

Para conferir uma tabela MyISAM, utilize os seguintes comandos:

`myisamchk nome_tabela`

Encontra 99.99% de todos os erros. O que ele não pode encontrar é corrompimento que envolva **SOMENTE** o arquivo de dados (que não é comum). Se você desejar conferir uma tabela, você deve executar normalmente o `myisamchk` sem opções ou com as opções `-s` ou `--silent`.

`myisamchk -m nome_tabela`

Encontra 99.999% de todos os erros. Ele verifica primeiramente erros em todas as entradas do índice e então le todos os registros. Ele calcula um checksum para todas as chaves nos registros e verifica se o checksum é o mesmo que o checksum das chaves na árvore de índices.

`myisamchk -e nome_tabela`

Realiza a verificação completa de todos os dados (`-e` significa “conferência estendida”). Ele faz uma conferência lendo todas as chaves de cada registro para verificar se eles realmente apontam para o registro correto. Isto pode demorar **MUITO** tempo em uma tabela grande com várias chaves. `myisamchk` normalmente irá parar depois do primeiro erro que encontrar. Se você deseja

obter mais informações, pode adicionar a opção `--verbose (-v)`. Isto faz o `myisamchk` continuar a percorrer a tabela até um máximo de 20 erros. Em utilização normal, um simples `myisamchk` (sem argumentos além do nome da tabela) é suficiente.

```
myisamchk -e -i nome_tabela
```

Como o comando anterior, mas a opção `-i` diz ao `myisamchk` para exibir algumas informações estatísticas também.

4.4.6.9 Como Reparar Tabelas

Na seção seguinte nós só falaremos do uso do `myiasmchk` em tabelas MyISAM (extensões `.MYI` e `.MYD`). Se você estiver usando tabelas ISAM (extensões `.ISM` e `.ISD`), você deve usar a ferramenta `isamchk`.

A partir do MySQL versão 3.23.14, você pode reparar tabelas MyISAM com o comando `REPAIR TABLE`. See [\(undefined\) \[REPAIR TABLE\], page \(undefined\)](#).

Os sintomas de uma tabela corrompida incluem pesquisas que abortam inesperadamente e erros como estes:

- 'nome_tabela.frm' is locked against change
- Can't find file 'nome_tabela.MYI' (Errcode: ###)
- Unexpected end of file
- Record file is crashed
- Got error ### from table handler

Para obter mais informações sobre o erro você pode executar `perror ###`. Aqui estão os erros mais comuns que indicam um problema com a tabela:

```
shell> perror 126 127 132 134 135 136 141 144 145
126 = Index file is crashed / Wrong file format
127 = Record-file is crashed
132 = Old database file
134 = Record was already deleted (or record file crashed)
135 = No more room in record file
136 = No more room in index file
141 = Duplicate unique key or constraint on write or update
144 = Table is crashed and last repair failed
145 = Table was marked as crashed and should be repaired
```

Perceba que o erro 135, não é um erro que pode ser corrigido por um simples reparo. Neste caso você deve fazer:

```
ALTER TABLE tabela MAX_ROWS=xxx AVG_ROW_LENGTH=yyy;
```

Em outros casos, você deve reparar suas tabelas. `myisamchk` pode normalmente detectar a maioria das coisas que estiverem erradas.

O processo de reparo envolve até quatro estágios, descritos abaixo. Antes de começar, você deve mudar para o diretório do banco de dados e conferir as permissões dos arquivos de tabelas. Tenha certeza que eles possam ser lidos pelo usuário do Unix com o qual `mysqld` é executado (e para você, porque você precisa acessar os arquivos que está conferindo).

Se não estiverem, você precisa alterar os arquivos, eles também devem ter a permissão de escrita para você.

Se você estiver utilizando o MySQL versão 3.23.16 e superior, você pode (e deve) usar os comandos `CHECK` e `REPAIR` para conferir e corrigir tabelas MyISAM. See [\[CHECK TABLE\]](#), page [\[CHECK TABLE\]](#). See [\[REPAIR TABLE\]](#), page [\[REPAIR TABLE\]](#).

A seção do manual sobre manutenção de tabelas inclui as opções para `isamchk/myisamchk`. See [\[Table maintenance\]](#), page [\[Table maintenance\]](#).

A seguinte seção são para os casos onde o comando acima falhar ou se você desejar usar os recursos estendidos que o `isamchk` e `myisamchk` fornecem.

Se você for reparar uma tabela da linha de comandos, deve primeiro desligar o servidor `mysqld`. Perceba que quando você executa `mysqladmin shutdown` em um servidor remoto, o servidor `mysqld` irá continuar funcionando por um tempo depois do `mysqladmin` retornar, até que todas as queries parem e todas as chaves sejam descarregadas no disco.

Estágio 1: Verificando suas tabelas

Execute `myisamchk *.MYI` ou `myisamchk -e *.MYI` se você tiver tempo disponível. Utilize a opção `-s` (silencioso) para suprimir informações desnecessárias.

Se o servidor `mysqld` parar, deve ser utilizada a opção `--update` para dizer ao `myisamchk` marcar a tabela como 'checada'.

Você deve reparar somente as tabelas em que o `myisamchk` indicar um erro. Para tais tabelas, vá para o estágio 2.

Se você obter erros estranhos na verificação (como nos erros `out of memory`), ou se o `myisamchk` quebrar, vá para o estágio 3.

Estágio 2: Reparo simples e seguro

NOTA: Se você deseja que os reparos sejam mais rápidos, devem ser usadas as opções: `-O sorf_buffer=# -O key_buffer=#` (onde `#` seria 1/4 da memória disponível) para todos comandos `isamchk/myisamchk`.

Primeiro, tente usar `myisamchk -r -q nome_tabela` (`-r -q` significa "modo de recuperação rápida"). Ele tentará reparar o arquivo de índice sem mexer no arquivo de dados. Se o arquivo de dados estiver normal e os links apagados apontam nas localizações corretas dentro do arquivo de dados, isto deve funcionar e a tabela será corrigida. Inicie o reparo da próxima tabela. Outra maneira seria utilizar os seguintes procedimentos:

1. Faça um backup do arquivo de dados antes de continuar.
2. Utilize `myisamchk -r nome_tabela` (`-r` significa modo de "recuperação"). Isto removerá registros incorretos e deletados do arquivo de dados e reconstrói o arquivo de índices.
3. Se o passo anterior falhar, utilize `myisamchk --safe-recover nome_tabela`. O modo de recuperação segura utiliza um método de recuperação antiga que trata de alguns casos que o modo de recuperação comum não consegue (porém é mais lento).

Se você obter erros estranhos no reparo (como em erros `out of memory`), ou se o `myisamchk` falhar, vá para o estágio 3.

Estágio 3: Reparo difícil

Você só deve atingir este estágio se o primeiro bloco de 16K do arquivo de índice estiver destruído ou conter informações incorretas, ou se o arquivo de índice não existir. Neste caso, é necessário criar um novo arquivo de índice. Faça como a seguir:

1. Mova o arquivo de dados para algum lugar seguro.
2. Use o arquivo de descrição de tabelas para criar novos arquivos (vazios) de dados e índices:

```
shell> mysql nome_bd
mysql> SET AUTOCOMMIT=1;
mysql> TRUNCATE TABLE nome_tabela;
mysql> quit
```

Se sua versão do MySQL não possuir TRUNCATE TABLE, utilize DELETE FROM nome_tabela.

3. Copie o antigo arquivo de dados de volta para o novo arquivo de dados criado. (Não só mova o antigo arquivo de volta para o novo arquivo; você deve uma cópia no caso de algo der errado.)

Volte ao estágio 2. `myisamchk -r -q` deve funcionar agora. (Isto não deve ser um loop eterno.)

Estágio 4: Reparo muito difícil

Você deve atingir este estágio somente se o arquivo de descrição também falhar. Isto nunca deve acontecer, porque o arquivo de descrição não é alterado depois da tabela ser criada:

1. Restaure o arquivo de descrição de um backup e volte ao estágio 3. Você pode também restaurar o arquivo de índice e voltar ao estágio 2. No último caso, você deve iniciar com `myisamchk -r`.
2. Se você não tem um backup mas sabe exatamente como a tabela foi criada, crie uma cópia da tabela em outro banco de dados. Remova o novo arquivo de dados, e então mova a descrição e arquivos de índice do outro banco de dados para o banco de dados com problemas. Isto lhe fornece um novo arquivos índice e descrição, mas mantém o arquivo de dados da mesma forma. Volte ao estágio 2 e tente reconstruir o arquivo de índices.

4.4.6.10 Otimização de Tabelas

Para agrupar registros fragmentados e eliminar perda de espaço resultante de remoções ou atualizações de registros, execute `myisamchk` no modo de recuperação:

```
shell> myisamchk -r nome_tabela
```

Você pode otimizar uma tabela da mesma forma utilizando a instrução SQL OPTIMIZE TABLE. OPTIMIZE TABLE faz o reparo de tabelas, analisa chaves e também ordena a árvore de índices para fazer pesquisas por chave mais rápidas. Também não existem possibilidade de interação não desejável entre o utilitário e o servidor, porque o servidor faz todo o trabalho quando você utiliza OPTIMIZE TABLE. See [\[OPTIMIZE TABLE\]](#), page [\[undefined\]](#).

`myisamchk` também tem um número de outras opção que podem ser usadas para melhorar a performance de uma tabela:

```
-S, --sort-index
-R num_indice, --sort-records=num_indice
-a, --analyze
```

Para uma descrição completa da opção. See [\[myisamchk syntax\]](#), page [\[undefined\]](#).

4.4.7 Configurando um Regime de Manutenção das Tabelas

A partir do MySQL Versão 3.23.13, você pode conferir tabelas MyISAM com o comando `CHECK TABLE`. See [\[CHECK TABLE\]](#), page [\[CHECK TABLE\]](#). Você pode reparar tabelas com o comando `REPAIR TABLE`. See [\[REPAIR TABLE\]](#), page [\[REPAIR TABLE\]](#).

É uma boa idéia verificar as tabelas regularmente em vez de esperar que ocorram problemas. Para propósitos de manutenção você pode utilizar o `myisamchk -s` para verificar as tabelas. A opção `-s` (abreviação de `--silent`) faz com que o `myisamchk` execute em modo silencioso, exibindo mensagens somente quando ocorrem erros.

É também uma boa idéia verificar as tabelas quando o servidor inicia. Por exemplo, sempre que a máquina reinicia no meio de uma atualização, você normalmente precisará conferir todas as tabelas que podem ter sido afetadas. (Isto é uma “tabela com falhas esperadas”.) Você pode adicionar um teste ao `safe_mysqld` que executa `myisamchk` para conferir todas as tabelas que foram modificadas durante as últimas 24 horas se existir um arquivo `‘.pid’` (process ID) antigo depois do último reboot. (O arquivo `‘.pid’` é criado pelo `mysqld` quando ele inicia e removido quando ele termina normalmente. A presença de um arquivo `‘.pid’` durante a inicialização do sistema indica que o `mysqld` terminou de forma anormal.)

Um teste ainda melhor seria verificar qualquer tabela cuja a data da última modificação é mais recente que a do arquivo `‘.pid’`.

Você também deve verificar suas tabelas regularmente durante a operação normal do sistema. Na MySQL AB, nós executamos uma tarefa agendada `cron` para conferir todas as nossas tabelas importantes uma vez por semana utilizando uma linha com esta no arquivo `‘crontab’`:

```
35 0 * * 0 /diretório/do/myisamchk --fast --silent /diretório/de/dados/*/*.MYI
```

Isto exhibe informações sobre tabelas com falhas para que possamos examiná-las e repará-las quando necessário.

Como nós não estamos tendo tabelas com falhas inesperadas (tabelas corrompidas por razões diferentes de problemas de hardware) por vários anos (isto realmente é verdade), uma vez por semana é mais que suficiente para nós.

Nós recomendamos que para iniciar, você execute `myisamchk -s` a cada noite em todas as tabelas que foram atualizadas durante as últimas 24 horas, até que você confie no MySQL como nós confiamos.

Normalmente você não precisará de tanta manutenção em suas tabelas MySQL. Se você estiver alterando tabelas com registros de tamanho dinâmico (tabelas com colunas `VARCHAR`, `BLOB` ou `TEXT`) ou tem tabelas com vários registros apagados você pode desejar de tempos em tempos (uma vez ao mês?) desfragmentar/recuperar espaço das tabelas.

Você pode fazer isto utilizando `OPTIMIZE TABLE` nas tabelas em questão ou se você puder desligar o servidor `mysqld` por um tempo faça:

```
isamchk -r --silent --sort-index -O sort_buffer_size=16M */*.ISM
myisamchk -r --silent --sort-index -O sort_buffer_size=16M */*.MYI
```

4.4.8 Obtendo Informações sobre as Tabelas

Para obter uma descrição de uma tabela ou estatísticas sobre ela, utilize os comandos mostrados abaixo, nós explicaremos algumas das informações em mais detalhes posteriormente:

`myisamchk -d nome_tabela`

Executa o `myisamchk` no “modo descritivo” para produzir uma descrição de sua tabela. Se você iniciar o servidor MySQL utilizando a opção `--skip-locking`, `myisamchk` pode relatar um erro para uma tabela que está sendo atualizada enquanto é executado. Entretanto, como o `myisamchk` não altera a tabela no modo de descrição, não existem riscos de destruição de dados.

`myisamchk -d -v nome_tabela`

Para produzir mais informações sobre o que `myisamchk` está fazendo, adicione `-v` para solicitar a execução em modo verbose.

`myisamchk -eis nome_tabela`

Exibe somente as informações mais importantes de uma tabela. Ele é lento porque é necessário ler a tabela inteira.

`myisamchk -eiv nome_tabela`

Isto se parece com `-eis`, mas lhe diz o que está sendo feito.

Exemplo da saída de `myisamchk -d`

```
MyISAM file:      company.MYI
Record format:   Fixed length
Data records:    1403698 Deleted blocks:      0
Recordlength:    226
```

table description:

Key	Start	Len	Index	Type
1	2	8	unique	double
2	15	10	multip.	text packed stripped
3	219	8	multip.	double
4	63	10	multip.	text packed stripped
5	167	2	multip.	unsigned short
6	177	4	multip.	unsigned long
7	155	4	multip.	text
8	138	4	multip.	unsigned long
9	177	4	multip.	unsigned long
	193	1		text

Exemplo da saída de `myisamchk -d -v` :

```
MyISAM file:      company
Record format:   Fixed length
File-version:     1
Creation time:    1999-10-30 12:12:51
Recover time:     1999-10-31 19:13:01
Status:           checked
Data records:     1403698 Deleted blocks:      0
```

```
Datafile parts:          1403698 Deleted data:          0
Datafilepointer (bytes):    3 Keyfile pointer (bytes):    3
Max datafile length: 3791650815 Max keyfile length: 4294967294
Recordlength:              226
```

table description:

Key	Start	Len	Index	Type	Rec/key	Root	Blocksize
1	2	8	unique	double	1	15845376	1024
2	15	10	multip.	text packed stripped	2	25062400	1024
3	219	8	multip.	double	73	40907776	1024
4	63	10	multip.	text packed stripped	5	48097280	1024
5	167	2	multip.	unsigned short	4840	55200768	1024
6	177	4	multip.	unsigned long	1346	65145856	1024
7	155	4	multip.	text	4995	75090944	1024
8	138	4	multip.	unsigned long	87	85036032	1024
9	177	4	multip.	unsigned long	178	96481280	1024
	193	1		text			

Exemplo da saída de `myisamchk -eis`:

Checking MyISAM file: company

```
Key:  1:  Keyblocks used:  97% Packed:    0% Max levels:  4
Key:  2:  Keyblocks used:  98% Packed:   50% Max levels:  4
Key:  3:  Keyblocks used:  97% Packed:    0% Max levels:  4
Key:  4:  Keyblocks used:  99% Packed:   60% Max levels:  3
Key:  5:  Keyblocks used:  99% Packed:    0% Max levels:  3
Key:  6:  Keyblocks used:  99% Packed:    0% Max levels:  3
Key:  7:  Keyblocks used:  99% Packed:    0% Max levels:  3
Key:  8:  Keyblocks used:  99% Packed:    0% Max levels:  3
Key:  9:  Keyblocks used:  98% Packed:    0% Max levels:  4
Total:  Keyblocks used:  98% Packed:   17%
```

```
Records:          1403698 M.recordlength:    226 Packed:          0%
Recordspace used:   100% Empty space:          0% Blocks/Record:  1.00
Record blocks:     1403698 Delete blocks:         0
Recorddata:        317235748 Deleted data:         0
Lost space:         0 Linkdata:          0
```

User time 1626.51, System time 232.36

Maximum resident set size 0, Integral resident set size 0

Non physical pagefaults 0, Physical pagefaults 627, Swaps 0

Blocks in 0 out 0, Messages in 0 out 0, Signals 0

Voluntary context switches 639, Involuntary context switches 28966

Exemplo da saída de `myisamchk -eiv`:

Checking MyISAM file: company

```
Data records: 1403698 Deleted blocks: 0
```

- check file-size

- check delete-chain

block_size 1024:

index 1:

```

index 2:
index 3:
index 4:
index 5:
index 6:
index 7:
index 8:
index 9:
No recordlinks
- check index reference
- check data record references index: 1
Key: 1: Keyblocks used: 97% Packed: 0% Max levels: 4
- check data record references index: 2
Key: 2: Keyblocks used: 98% Packed: 50% Max levels: 4
- check data record references index: 3
Key: 3: Keyblocks used: 97% Packed: 0% Max levels: 4
- check data record references index: 4
Key: 4: Keyblocks used: 99% Packed: 60% Max levels: 3
- check data record references index: 5
Key: 5: Keyblocks used: 99% Packed: 0% Max levels: 3
- check data record references index: 6
Key: 6: Keyblocks used: 99% Packed: 0% Max levels: 3
- check data record references index: 7
Key: 7: Keyblocks used: 99% Packed: 0% Max levels: 3
- check data record references index: 8
Key: 8: Keyblocks used: 99% Packed: 0% Max levels: 3
- check data record references index: 9
Key: 9: Keyblocks used: 98% Packed: 0% Max levels: 4
Total: Keyblocks used: 9% Packed: 17%

- check records and index references
[LOTS OF ROW NUMBERS DELETED]

Records:          1403698  M.recordlength:    226  Packed:          0%
Recordspace used: 100%    Empty space:       0%  Blocks/Record:  1.00
Record blocks:   1403698  Delete blocks:     0
Recorddata:     317235748  Deleted data:     0
Lost space:      0        Linkdata:          0

User time 1639.63, System time 251.61
Maximum resident set size 0, Integral resident set size 0
Non physical pagefaults 0, Physical pagefaults 10580, Swaps 0
Blocks in 4 out 0, Messages in 0 out 0, Signals 0
Voluntary context switches 10604, Involuntary context switches 122798

```

Aqui estão os tamanhos dos arquivos de dados e índices para a tabela utilizada nos exemplos anteriores:

```

-rw-rw-r-- 1 monty tcx 317235748 Jan 12 17:30 company.MYD
-rw-rw-r-- 1 davida tcx 96482304 Jan 12 18:35 company.MYM

```

Explicações para os tipos de informações que o `myisamchk` produz são fornecidas abaixo. O “keyfile” é o arquivo de índices. “Registro” e “linha” são sinônimos:

ISAM file Nome do arquivo (índice) ISAM.

Isam-version

Versão do formato ISAM. Atualmente sempre 2.

Creation time

Quando o arquivo de dados foi criado.

Recover time

Quando foi a última vez que o arquivo de índices/dados foi reconstruído.

Data records

Quantos registros existem na tabela.

Deleted blocks

Quantos blocos apagados continuam alocando espaço. Você pode otimizar sua tabela para minimizar este espaço. See [\(undefined\) \[Otimização\]](#), page [\(undefined\)](#).

Datafile: Parts

Para formato de registros dinâmicos, isto indica quantos blocos de dados existem. Para uma tabela otimizada sem registros fragmentados, isto é o mesmo que **Data records**.

Deleted data

Quantos bytes de dados deletados não recuperados existem. Você pode otimizar sua tabela para minimizar este espaço. See [\(undefined\) \[Otimização\]](#), page [\(undefined\)](#).

Datafile pointer

O tamanho do ponteiro do arquivo de dados, em bytes. Ele normalmente possui 2, 3, 4 ou 5 bytes. A maioria das tabelas trabalham com 2 bytes, mas isto ainda não pode ser controlado pelo MySQL ainda. Para tabelas fixas, isto é um endereço de registro. Para tabelas dinâmicas, isto é um endereço de byte.

Keyfile pointer

O tamanho de um ponteiro de arquivo de índices, em bytes. Ele normalmente possui 1, 2 ou 3 bytes. A maioria das tabelas trabalham com 2 bytes, mas isto é calculado automaticamente pelo MySQL. Ele é sempre um endereço de bloco.

Max datafile length

Qual tamanho o arquivo de dados (arquivos `.MYD`) pode atingir, em bytes.

Max keyfile length

Qual tamanho o arquivo de índices (`.MYI`) pode atingir, em bytes.

Recordlength

Quanto espaço cada registro ocupa, em bytes.

Record format

O formato utilizado para armazenar as linhas da tabelas. Os exemplos vistos abaixo utilizam **Fixed length** (tamanho fixo). Outros valores possíveis são **Compressed**(compactado) e **Packed**(empacotado).

table description

Uma lista de todas as chaves na tabela. Para cada chave, alguma informação de baixo nível é apresentada:

Key	O Número desta chave.
Start	Onde, no registro, esta parte do índice inicia.
Len	Qual o tamanho desta parte do índice. Para números empacotados, isto deve sempre ser o tamanho total da coluna. Para strings, deve ser mais curto que o tamanho total da coluna indexada, porque você pode indexar um prefixo de uma coluna string.
Index	unique ou multip. (múltiplos). Indica se um valor pode ou não existir várias vezes neste índice.
Type	Que tipo de dados esta parte do índice tem. Isto é um tipo de dados ISAM com as opções packed , stripped ou empty .
Root	Endereço do bloco de índice raiz.
Blocksize	O tamanho de cada bloco de índice. O tamanho padrão é 1024, mas o valor pode ser alterado na compilação.
Rec/key	Este é um valor estatístico utilizado pelo otimizador. Ele diz quantos registros existem por valor para esta chave. Uma chave única sempre tem um valor de 1. Ele pode ser atualizado depois que uma tabela é carregada (ou muito alterada) com myisamchk -a . Se isto não for completamente atualizado, um valor padrão de 30 é fornecido.

No primeiro exemplo acima, a nona chave é uma chave multi partes com duas partes.

Keyblocks used

Qual o percentual de bloco de chaves são usados. Como a tabela usada nos exemplos foi reorganizada com **myisamchk**, os valores são muito altos (muito próximos do máximo teórico).

Packed O MySQL tenta empacotar chaves com um sufixo comum. Isto pode ser usado somente para chaves **CHAR/VARCHAR/DECIMAL**. Para strings grandes como nomes, isto pode reduzir significativamente o espaço utilizado. No terceiro exemplo acima, a quarta chave possui 10 caracteres e uma redução de 60% no espaço é obtida.

Max levels

Qual a profundidade da árvore-B para esta chave. Grandes tabelas com chaves longas resultam em valores altos.

Records Quantos registros existem na tabela.

M.recordlength

A média de tamanho do registro. Para tabelas com registros de tamanho fixo, isto é o tamanho exato do registro.

Packed O MySQL corta espaços do final de strings. O valor **Packed** indica o percentual de economia alcançado fazendo isto.

Recordspace used

Qual percentual do arquivo de dados é usado.

Empty space

Qual percentual do arquivo de dados não é usado.

Blocks/Record

Número médio de blocos por registro (isto é, de quantos links um registro fragmentado é composto). Sempre será 1 para tabelas de formato fixo. Este valor deve permanecer o mais próximo possível de 1.0. Se ele aumentar, você pode reorganizar a tabela com `myisamchk`. See [\(undefined\)](#) [Otimização], page [\(undefined\)](#).

Recordblocks

Quantos blocos (links) são utilizados. Para formatos fixos, este é o mesmo que o número de registros.

Deleteblocks

Quantos blocos (links) foram excluídos.

Recorddata

Quantos bytes no arquivo de dados são usados.

Deleted data

Quantos bytes no arquivo de dados foram apagados (sem uso).

Lost space

Se um registro é atualizado para um tamanho menor, algum espaço é perdido. Isto é a soma de todas estas perdas, em bytes.

Linkdata Quando o formato de tabela dinâmica é utilizado, fragmentos de registros são ligados com ponteiros (4 a 7 bytes cada). **Linkdata** é a soma do montante de armazenamento utilizado por todos estes ponteiros.

Se uma tabela foi compactada com `myisampack`, `mysiamchk -d` exibe informações adicionais sobre cada coluna da tabela. Veja [\(undefined\)](#) [`myisampack`], page [\(undefined\)](#), para um exemplo desta informação e uma descrição do que ela significa.

4.5 Administração de Bancos de Dados e Referência de Linguagem

4.5.1 Sintaxe de OPTIMIZE TABLE

```
OPTIMIZE TABLE nome_tabela[,nome_tabela]...
```

`OPTIMIZE TABLE` deve ser usado se você apagou uma grande parte de uma tabela ou se você fez várias alterações à uma tabela com registros de tamanho variável (tabelas que tenham campos do tipo `VARCHAR`, `BLOB` ou `TEXT`). Registros apagados são mantidos em uma lista de ligações e operações `INSERT` subsequentes reutilizam posições de registros antigos. Você

pode utilizar `OPTIMIZE TABLE` para reclamar o espaço inutilizado e para desfragmentar o arquivo de dados.

No momento `OPTIMIZE TABLE` só funciona em tabelas **MyISAM** e BDB. Para tabelas BDB, `OPTIMIZE TABLE` é atualmente mapeado para `ANALYZE TABLE`. See [\(undefined\) \[ANALYZE TABLE\]](#), page [\(undefined\)](#).

Você pode ter a otimização de tabelas trabalhando em outros tipos de tabelas iniciando o `mysqld` com `--skip-new` ou `--safe-mode`, mas neste caso, `OPTIMIZE TABLE` é mapeado apenas para `ALTER TABLE`.

`OPTIMIZE TABLE` funciona da seguinte forma:

- Se a tabela tem registros excluídos ou divididos, repara a tabela.
- Se as páginas de índice não estão ordenadas, ordene-as.
- Se as estatísticas não estão atualizadas (e o reparo não pode ser feito ordenando o índice), atualize-as.

`OPTIMIZE TABLE` para tabelas MyISAM é equivalente a executar `myisamchk --quick --check-changed-tables --sort-index --analyze` na tabela.

Perceba que a tabela estará bloqueada durante o tempo em que `OPTIMIZE TABLE` estiver executando.

4.5.2 Sintaxe de ANALYZE TABLE

```
ANALYZE TABLE nome_tabela[,nome_tabela...]
```

Analisa e armazena a distribuição de chaves para a tabela. Durante a análise a tabela é bloqueada com uma trava de leitura. Isto funciona em tabelas MyISAM e BDB.

Isto seria equivalente a executar `myisamchk -a` na tabela.

O MySQL utiliza a distribuição de chaves armazenadas para decidir em que ordem tabelas devem ser unidas quando alguém faz um join em alguma coisa diferente de uma constante.

O comando retorna uma tabela com as seguintes colunas:

Coluna	Valor
Table	Nome da Tabela
Op	Sempre "analyze"
Msg.type	Um dos seguintes: <code>status</code> , <code>error</code> , <code>info</code> ou <code>warning</code> .
Msg.text	A mensagem.

Você pode verificar a distribuição de chaves armazenadas com o comando `SHOW INDEX`. See [\(undefined\) \[SHOW DATABASE INFO\]](#), page [\(undefined\)](#).

Se a tabela não foi alterada deste o último comando `ANALYZE TABLE`, a tabela não será analisada novamente.

4.5.3 Sintaxe de FLUSH

```
FLUSH opções [,opções]
```

Você deve utilizar o comando `FLUSH` se desejar limpar algum dos caches internos que o MySQL usa. Para executar `FLUSH`, você deve ter o privilégio **RELOAD**.

`opções` podem ser qualquer uma das seguintes:

HOSTS	Esvazia as tabelas de cache de nomes de máquinas. Você deve descarregar as tabelas de nomes de máquinas se alguma de suas máquinas receber um número IP diferente ou se você obter a mensagem de erro <code>Host ... is blocked</code> . Quando mais de <code>max_connect_erros</code> erros ocorrerem em um registro para uma determinada máquina enquanto se conecta ao servidor MySQL, o MySQL assume que algo está errado e bloqueia futuras requisições desta máquina. A descarga na tabela de nomes de máquinas permite à máquina se conectar novamente. See (undefined) [Máquina bloqueada], page (undefined) .) Você pode iniciar o <code>mysqld</code> com <code>-O max_connection_errors=999999999</code> para evitar esta mensagem de erro.
LOGS	Fecha e reabre todos os arquivos de log. Se você tiver especificado o arquivo de logs de atualizações ou um arquivo de log binário sem uma extensão, o número de extensão do arquivo log será sempre incrementado de um em relação ao arquivo anterior. Se você usou uma extensão no nome do arquivo, o MySQL irá fechar e reabrir o arquivo de log de atualizações. See (undefined) [Log de atualização], page (undefined) . Isto é a mesma coisa que enviar o sinal <code>SIGHUP</code> para o servidor <code>mysqld</code> .
PRIVILEGES	Recarrega os privilégios das tabelas de permissões no banco de dados <code>mysql</code> .
TABLES	Fecha todas as tabelas abertas e força o fechamento de todas as tabelas em uso
[TABLE TABLES] nome_tabela [,nome_tabela...]	Descarga somente das tabelas fornecidas.
TABLES WITH READ LOCK	Fecha todas tabelas abertas e bloqueia todas tabelas para todos os bancos de dados com leitura até que alguém execute <code>UNLOCK TABLES</code> . Isto é uma maneira muito conveniente para fazer backups se você possui um sistema de arquivos, como Veritas, que pode fazer uma imagem instantânea (snapshot) de um certo momento.
STATUS	Reinicia a maioria das variáveis de status para zero. Isto é algo que deve ser usado somente para depurar uma consulta.

Você pode também acessar cada um dos comandos vistos acima com o utilitário `mysqladmin`, utilizando os comandos `flush-hosts`, `flush-logs`, `reload` ou `flush-tables`.

Também de uma olhada no comando `RESET` usado com a replicação. See [\(undefined\) \[Replicação SQL\], page \(undefined\)](#).

4.5.4 Sintaxe de KILL

```
KILL thread_id
```

Cada conexão ao `mysqld` executa em uma thread separada. Você pode ver quais threads estão em execução com o comando `SHOW PROCESSLIST` e matar uma thread com o comando `KILL thread_id`.

Se você tiver o privilégio **process**, você pode ver e matar qualquer thread. Caso contrário, você pode ver e matar somente suas próprias threads.

Você também pode usar os comandos `mysqladmin processlist` e `mysqladmin kill` para examinar e matar threads.

Quando você utiliza um KILL, um sinal (flag) `kill` específico é configurado para a thread.

Na maioria dos casos pode levar algum tempo para a thread morrer pois o sinal `kill` só é checado em intervalos específicos.

- Nos loops `SELECT`, `ORDER BY` e `GROUP BY`, o sinal é checado depois de ler um bloco de registros. Se o sinal `kill` está habilitado a instrução é abortada.
- Na execução de um `ALTER TABLE` o sinal `kill` é conferido antes de cada bloco de registros ser lido da tabela original. Se o sinal `kill` foi habilitado, o comando é abortado e a tabela temporária apagada.
- Ao fazer um `UPDATE TABLE` and `DELETE TABLE`, o sinal de `kill` é conferido depois de que cada bloco é lido e depois de cada atualização ou remoção de registro. Se o sinal `kill` está habilitado, a instrução é abortada. Note que se você não estiver utilizando transações, as alterações não irão ser desfeitas!
- `GET_LOCK()` irá aborar com `NULL`.
- Uma thread `INSERT DELAYED` irá rapidamente descarregar todos registros que estiverem em memória e morrer.
- Se a thread estiver no manipulador de bloqueio de tabelas (status: `Locked`), o bloqueio de tabela será abortado rapidamente.
- Se a thread estiver esperando por espaço livre em disco numa chamada `write`, a escrita é abortada com uma mensagem de espaço em disco insuficiente.

4.5.5 Sintaxe de SHOW

```

SHOW DATABASES [LIKE wild]
ou SHOW [OPEN] TABLES [FROM nome_bd] [LIKE wild]
ou SHOW [FULL] COLUMNS FROM nome_tabela [FROM nome_bd] [LIKE wild]
ou SHOW INDEX FROM nome_tabela [FROM nome_bd]
ou SHOW TABLE STATUS [FROM nome_bd] [LIKE wild]
ou SHOW STATUS [LIKE wild]
ou SHOW VARIABLES [LIKE wild]
ou SHOW LOGS
ou SHOW [FULL] PROCESSLIST
ou SHOW GRANTS FOR usuario
ou SHOW CREATE TABLE nome_tabela
ou SHOW MASTER STATUS
ou SHOW MASTER LOGS
ou SHOW SLAVE STATUS

```

`SHOW` fornece informações sobre bancos de dados, tabelas, colunas ou informações do estado do servidor. Se a parte `LIKE wild` é usada, a string `wild` pode ser uma string que usa os meta caracteres `'%` e `'_'` do SQL.

4.5.5.1 Recuperando Informações sobre Bancos de Dados, Tabelas, Colunas e Índices

Você pode usar `nome_bd.nome_tabela` como uma alternativa para a sintaxe `nome_tabela FROM nome_bd`. Estas duas declarações são equivalentes:

```
mysql> SHOW INDEX FROM minhatabela FROM meudb;
mysql> SHOW INDEX FROM meubd.minhatabela;
```

`SHOW DATABASES` lista os bancos de dados no servidor MySQL. Você também pode obter esta lista utilizando o comando `mysqlshow`.

`SHOW TABLES` lista as tabelas em um banco de dados específico. Esta lista também pode ser obtida utilizando o comando `mysqlshow nome_db`.

NOTA: Se um usuário não possui nenhum privilégio para uma tabela, a tabela não será mostrada na saída de `SHOW TABLES` ou `mysqlshow nome_db`.

`SHOW OPEN TABLES` lista as tabelas que estão abertas no cache de tabelas. See [\[Cache de Tabelas\]](#), page [\[Cache de Tabelas\]](#). O campo `Comment` diz quantas vezes a tabela está em `cached` e `in_use`.

`SHOW COLUMNS` lista as colunas em uma determinada tabela. Se você especificar a opção `FULL`, também irá obter os privilégios que você possui para cada coluna. Se os tipos de colunas forem diferentes do que você esperava baseando na declaração `CREATE TABLE`, perceba que o MySQL algumas vezes altera os tipos das colunas. See [\[Mudança de tipos de colunas\]](#), page [\[Mudança de tipos de colunas\]](#).

A declaração `DESCRIBE` fornece informação similar à `SHOW COLUMNS`. See [\[DESCRIBE\]](#), page [\[DESCRIBE\]](#).

`SHOW FIELDS` é um sinônimo para `SHOW COLUMNS` e `SHOW KEYS` um sinônimo para `SHOW INDEX`. Você também pode listar as colunas ou índices de uma tabela com `mysqlshow nome_db nome_tabela` ou `mysqlshow -k nome_bd nome_tabela`.

`SHOW INDEX` retorna a informação de índice em um formato que lembra bem a chamada `SQLStatistics` do ODBC. As seguintes colunas são retornadas:

Coluna	Significado
Table	Nome da tabela.
Non_unique	0 se o índice não puder conter duplicidades.
Key_name	Nome do índice.
Seq_in_index	Número da sequência da coluna no índice, à partir de 1.
Column_name	Nome da coluna.
Collation	Como a coluna é ordenada no índice. No MySQL, pode ter valores 'A' (Ascendente) ou NULL (Not sorted).
Cardinality	Número de valores únicos no índice. Isto é atualizado executando <code>isamchk -a</code> .
Sub_part	Número de caracteres indexados se a coluna só é a indexada parcialmente. NULL se a chave inteira for indexada.
Comment	Vários comentários. No momento, ele diz se o índice é textual ou não.

Perceba que como o `Cardinality` é contado baseado nas estatísticas armazenadas como inteiros, ele pode não ser exato para tabelas pequenas.

4.5.5.2 SHOW TABLE STATUS

`SHOW TABLE STATUS [FROM nome_bd] [LIKE wild]`

`SHOW TABLE STATUS` (introduzido na versão 3.23) funciona como o `SHOW STATUS`, mas fornece muitas informações sobre cada tabela. Você também pode obter esta lista utilizando o comando `mysqlshow --status nome_bd`. As seguintes colunas são retornadas:

Coluna	Significado
Name	Nome da tabela.
Type	Tipo da tabela. See (undefined) [Table types] , page (undefined) .
Row_format	O formato de armazenamento do registro (Fixed (Fixo), Dynamic(dinâmico), ou Compressed (Compactado)).
Rows	Número de registros.
Avg_row_length	Tamanho médio do registro.
Data_length	Tamanho do arquivo de dados.
Max_data_length	Tamanho máximo do arquivo de dados.
Index_length	Tamanho do arquivo de índice.
Data_free	Número de bytes alocados mas não utilizados.
Auto_increment	Próximo valor do auto incremento.
Create_time	Quando a tabela foi criada.
Update_time	A última vez que arquivo de dados foi atualizado.
Check_time	A última vez que a tabela foi verificada.
Create_options	Opções extras usadas com <code>CREATE TABLE</code> .
Comment	O Comentário utilizado quando a tabela é criada (ou alguma informação do porquê do MySQL não poder acessar a informação da tabela).

Tabelas InnoDB irão relatar o espaço livre no tablespace no comentário da tabela.

4.5.5.3 SHOW STATUS

`SHOW STATUS` fornece informações de status do servidor (como `mysqladmin extended-status`). A saída é parecida com o que está exibido abaixo, apesar dos números e formatos provavelmente serem diferentes:

Variable_name	Value
Aborted_clients	0
Aborted_connects	0
Bytes_received	155372598
Bytes_sent	1176560426
Connections	30023
Created_tmp_disk_tables	0
Created_tmp_tables	8340
Created_tmp_files	60
Delayed_insert_threads	0
Delayed_writes	0
Delayed_errors	0
Flush_commands	1

Handler_delete	462604
Handler_read_first	105881
Handler_read_key	27820558
Handler_read_next	390681754
Handler_read_prev	6022500
Handler_read_rnd	30546748
Handler_read_rnd_next	246216530
Handler_update	16945404
Handler_write	60356676
Key_blocks_used	14955
Key_read_requests	96854827
Key_reads	162040
Key_write_requests	7589728
Key_writes	3813196
Max_used_connections	0
Not_flushed_key_blocks	0
Not_flushed_delayed_rows	0
Open_tables	1
Open_files	2
Open_streams	0
Opened_tables	44600
Questions	2026873
Select_full_join	0
Select_full_range_join	0
Select_range	99646
Select_range_check	0
Select_scan	30802
Slave_running	OFF
Slave_open_temp_tables	0
Slow_launch_threads	0
Slow_queries	0
Sort_merge_passes	30
Sort_range	500
Sort_rows	30296250
Sort_scan	4650
Table_locks_immediate	1920382
Table_locks_waited	0
Threads_cached	0
Threads_created	30022
Threads_connected	1
Threads_running	1
Uptime	80380

As variáveis de estado listadas acima tem o seguinte significado:

Variável	Significado
Aborted_clients	Número de conexões abortadas porque o cliente morreu sem fechar a conexão corretamente. See [Erros de Comunicação] , page [Erros de Comunicação] .

Aborted_connects	Número de tentativas que falharam ao tentar a conexão ao servidor MySQL. See (undefined) [Erros de Comunicação], page (undefined) .
Bytes_received	Número de bytes recebidos por todos os clientes.
Bytes_sent	Número de bytes enviados para todos os clientes..
Com_xxxx	Número de vezes que os comandos xxx foram executados.
Connections	Número de tentativas de conexão ao servidor MySQL.
Created_tmp_disk_tables	Número de tabelas temporárias implícitas em disco criadas durante a execução de instruções.
Created_tmp_tables	Número de tabelas temporárias implícitas na memória criadas durante execuções de instruções.
Created_tmp_files	Quantos arquivos temporários o mysqld criou.
Delayed_insert_threads	Número de threads para tratamento de insertdelayed que estão em uso.
Delayed_writes	Número de registros escritos com INSERT DELAYED.
Delayed_errors	Número de registros escritos com INSERT DELAYED onde algum erro ocorreu (provavelmente duplicate key).
Flush_commands	Número de comandos FLUSH executados.
Handler_delete	Número de vezes que um registro foi apagado da tabela.
Handler_read_first	Número de vezes que a primeira entrada foi lida de um índice. Se este valor for alto, sugere que o servidor está fazendo várias leituras de índices, por exemplo, SELECT col1 FROM foo, assumindo que col1 é indexado.
Handler_read_key	Número de requisições para ler um registro baseado em uma chave. Se este valor for alto, é uma boa indicação que suas pesquisas e tabelas estão indexadas corretamente.
Handler_read_next	Número de requisições para ler o próximo registro na ordem da chave. Este valor será aumentado se você consultar uma coluna de índice com uma faixa restrita. Ele também aumentará se forem feitas busca nos índices.
Handler_read_rnd	Número de requisições para ler um registro baseado em uma posição fixa. O valor será alto se você estiver executando várias pesquisas que exigem ordenação do resultado.
Handler_read_rnd_next	Número de requisões para ler o próximo registro no arquivo de dados. Será alto se você estiver fazendo várias buscas na tabela. Geralmente sugere que suas tabelas não estão corretamente indexadas ou que suas pesquisas não foram escritas para tirar vantagem dos índices existentes.
Handler_update	Número de requisições para atualizar um registro em uma tabela.
Handler_write	Número de requisições para inserir um registro em uma tabela.
Key_blocks_used	O número de blocos utilizados no cache das chaves.
Key_read_requests	O número de requisições para ler um bloco de chaves do cache.
Key_reads	O número de leituras físicas de blocos de chaves do disco.
Key_write_requests	O número de requisições para gravar um bloco de chaves no cache.

<code>Key_writes</code>	O número de escritas físicas de um bloco de chaves para o disco.
<code>Max_used_connections</code>	O número máximo de conexões simultâneas que foram usadas.
<code>Not_flushed_key_blocks</code>	Blocos de chaves no cache de chaves que foi alterado mas ainda não foi descarregado para o disco.
<code>Not_flushed_delayed_rows</code>	Número de registros esperando para serem escritos em filas <code>INSERT DELAY</code> .
<code>Open_tables</code>	Número de tabelas abertas.
<code>Open_files</code>	Número de arquivos abertos.
<code>Open_streams</code>	Número de fluxos abertos (usado principalmente para logs).
<code>Opened_tables</code>	Número de tabelas que foram abertas.
<code>Select_full_join</code>	Número de joins sem chaves (Se for 0, você deve conferir com cuidado o índice de suas tabelas).
<code>Select_full_range_join</code>	Número de joins onde foram usadas pesquisas segmentadas na tabela de referencia.
<code>Select_range</code>	Número de joins onde foram usadas faixas da primeira tabela. (Normalmente não é crítica mesmo se o valor estiver alto.)
<code>Select_scan</code>	Número de joins onde fizemos uma busca completa na primeira tabela.
<code>Select_range_check</code>	Número de joins sem chaves onde o uso de chave foi conferido após cada registro (Se for 0, o índice de suas tabelas deve ser conferido com cuidado)
<code>Questions</code>	Número de consultas enviadas para o servidor.
<code>Slave_open_temp_tables</code>	Número de tabelas temporárias atualmente abertas pela thread escrava.
<code>Slow_launch_threads</code>	Número de threads que levaram mais tempo do que <code>slow_launch_time</code> para serem criadas.
<code>Slow_queries</code>	Número de consultas que levaram mais tempo que <code>long_query_time</code> . See [Log de consultas lentas] , page [Log de consultas lentas] .
<code>Sort_merge_passes</code>	Número de ifusões feitas pelo algoritmo de ordenação. Se este valor for alto você deve considerar o aumento de <code>sort_buffer</code> .
<code>Sort_range</code>	Número de ordenações que foram feitas com limites.
<code>Sort_rows</code>	Número de registros ordenados.
<code>Sort_scan</code>	Número de ordenações que foram feitas lendo a tabela.
<code>Table_locks_immediate</code>	Número de vezes que um travamento de tabela foi obtido de maneira automática.
<code>Table_locks_waited</code>	Número de vezes que um bloqueio de tabela não pôde ser obtido imediatamente e foi preciso esperar. Se o valor for alto, e você tiver problemas de performance, suas consultas devem ser otimizadas e depois dividir sua(s) tabela(s) ou usar replicação. Disponível à partir da versão 3.23.33
<code>Threads_cached</code>	Número de threads no cache de threads.
<code>Threads_connected</code>	Número de conexões atuais abertas.
<code>Threads_created</code>	Número de threads criadas para lidar com conexões.
<code>Threads_running</code>	Número de threads que não estão dormindo.

Uptime Quantos segundos o servidor está funcionando.

Alguns comentários sobre a tabela acima:

- Se `Opened_tables` for grande, provavelmente sua variável `table_cache` está muito pequena.
- Se `key_reads` for grande, provavelmente sua variável `key_buffer_size` provavelmente está muito pequena. O índice de acertos do cache pode ser calculado com `key_reads/key_read_requests`.
- Se `Handler_read_rnd` for grande, provavelmente você possui várias consultas que exigem do MySQL fazer busca em tabelas inteiras ou você tem joins que não utilizam chaves corretamente.
- Se `Threads_created` for grande você pode desejar aumentar a variável `thread_cache_size`.

4.5.5.4 SHOW VARIABLES

`SHOW VARIABLES [LIKE wild]`

`SHOW VARIABLES` exibe os valores de algumas variáveis de sistema do MySQL. Você pode também obter esta informação utilizando o comando `mysqladmin variables`. Se os valores padrões não são adequados, você pode configurar a maioria destas variáveis utilizando opções de linha de comando quando o `mysqld` iniciar. See [line options-snt](#) [Opções de linha de comando], [page line options-pg](#).

A saída parece com o exibido abaixo, porém o formato e os números podem divergir:

Nome_variável	Valor
<code>ansi_mode</code>	<code>OFF</code>
<code>back_log</code>	<code>50</code>
<code>basedir</code>	<code>/my/monty/</code>
<code>bdb_cache_size</code>	<code>16777216</code>
<code>bdb_log_buffer_size</code>	<code>32768</code>
<code>bdb_home</code>	<code>/my/monty/data/</code>
<code>bdb_max_lock</code>	<code>10000</code>
<code>bdb_logdir</code>	
<code>bdb_shared_data</code>	<code>OFF</code>
<code>bdb_tmpdir</code>	<code>/tmp/</code>
<code>binlog_cache_size</code>	<code>32768</code>
<code>concurrent_insert</code>	<code>ON</code>
<code>connect_timeout</code>	<code>5</code>
<code>datadir</code>	<code>/my/monty/data/</code>
<code>delay_key_write</code>	<code>ON</code>
<code>delayed_insert_limit</code>	<code>100</code>
<code>delayed_insert_timeout</code>	<code>300</code>
<code>delayed_queue_size</code>	<code>1000</code>
<code>flush</code>	<code>OFF</code>
<code>flush_time</code>	<code>0</code>
<code>have_bdb</code>	<code>YES</code>
<code>have_innodb</code>	<code>YES</code>

have_raid	YES	
have_ssl	NO	
init_file		
interactive_timeout	28800	
join_buffer_size	131072	
key_buffer_size	16776192	
language	/my/monty/share/english/	
large_files_support	ON	
log	OFF	
log_update	OFF	
log_bin	OFF	
log_slave_updates	OFF	
long_query_time	10	
low_priority_updates	OFF	
lower_case_table_names	0	
max_allowed_packet	1048576	
max_binlog_cache_size	4294967295	
max_connections	100	
max_connect_errors	10	
max_delayed_threads	20	
max_heap_table_size	16777216	
max_join_size	4294967295	
max_sort_length	1024	
max_tmp_tables	32	
max_write_lock_count	4294967295	
myisam_recover_options	DEFAULT	
myisam_sort_buffer_size	8388608	
net_buffer_length	16384	
net_read_timeout	30	
net_retry_count	10	
net_write_timeout	60	
open_files_limit	0	
pid_file	/my/monty/data/donna.pid	
port	3306	
protocol_version	10	
record_buffer	131072	
query_buffer_size	0	
safe_show_database	OFF	
server_id	0	
skip_locking	ON	
skip_networking	OFF	
skip_show_database	OFF	
slow_launch_time	2	
socket	/tmp/mysql.sock	
sort_buffer	2097116	
table_cache	64	
table_type	MYISAM	
thread_cache_size	4	
thread_stack	65536	

tmp_table_size	1048576	
tmpdir	/tmp/	
version	3.23.29a-gamma-debug	
wait_timeout	28800	
+-----+-----+		

Cada opção é descrita abaixo. Valores para tamanhos de buffer, comprimento e tamanho de pilha são fornecidos em bytes. Você pode especificar valores com sufixos 'K' ou M para indicar o valor em kilobytes ou megabytes. Por exemplo, 16M indica 16 Megabytes. Não importa se os sufixos estão em letras maiúsculas ou minúsculas; 16M e 16m são equivalentes:

ansi_mode.

Está ligado (ON) se o `mysqld` foi iniciado com `--ansi`. See `<undefined>` [Modo ANSI], page `<undefined>`.

back_log O número de requisições de conexões que o MySQL pode suportar. Isto entra em cena quando a thread principal do MySQL recebe **MUITAS** solicitações de conexões em um espaço curto de tempo. Eles tomam algum tempo (porém muito pouco) da a thread principal para conferir a conexão e iniciar uma nova thread. O valor `back_log` indica quantas requisições podem ser empilhadas durante este breve tempo antes do MySQL parar de responder a novas requisições. Você isó precisa aumentá-lo se espera um número alto de conexões em um curto período de tempo

Em outras palavras, este valor é o tamanho da fila de escuta para novas conexões TCP/IP. Seu sistema operacional tem o próprio limite para o tamanho desta fila. A página do manual Unix da chamada de sistema `listen(2)` deve fornecer maiores detalhes. Confira a documentação do seus SO para saber o valor máximo para esta variável. Tentativas de configurar `back_log` maior do que o limite de seu sistema operacional serão ineficazes.

basedir O valor da opção `--basedir`.

bdb_cache_size

O buffer que é alocado para o cache de índice e registros de tabelas BDB. Se você não utiliza tabelas BDB, deve iniciar o `mysqld` com a opção `--skip-bdb` para evitar desperdício de memória para este cache.

bdb_log_buffer_size

O buffer que é alocado para o cache de índice e registros de tabelas BDB. Se você não utiliza tabelas BDB, deve configurá-la com 0 ou iniciar o `mysqld` com a opção `--skip-bdb` para evitar desperdício de memória para este cache.

bdb_home O valor para a opção `--bdb-home`.

bdb_max_lock

O número máximo de bloqueios (1000 por padrão) que podem ser feitas em uma tabela BDB. Você deve ser aumentá-la se obter erros do tipo: `bdb: Lock table is out of available locks` ou `Got error 12 from ...` quando são necessárias longas transações ou quando o `mysqld` precisar examinar vários registros para calcular a pesquisa.

bdb_logdir

O valor da opção `--bdb-logdir`.

bdb_shared_data

Está ligada (ON) se você estiver utilizando `--bdb-shared-data`.

bdb_tmpdir

O valor da opção `--bdb-tmpdir`.

binlog_cache_size. O tamanho do cache para armazenar instruções

SQL para o log binário durante uma transação. Se você geralmente utiliza transações grandes, multi-instruções, você pode aumentar este valor para obter mais performance. See [\(undefined\)](#) [COMMIT], page [\(undefined\)](#).

character_set

O conjunto de caracteres padrão.

character_sets

Os conjuntos de caracteres suportados.

concurrent_inserts

Se ON (ligado, por padrão), o MySQL permitirá o uso de INSERT em tabelas MyISAM ao mesmo tempo em que são executadas consultas SELECT. Você pode desligar esta opção iniciando `mysqld` com `--safe` ou `--skip-new`.

connect_timeout

O número de segundos que o servidor `mysqld` espera para um pacote de conexão antes de responder com Bad handshake.

datadir O valor da opção `--datadir`.**delay_key_write**

Se habilitado (valor padrão), o MySQL utilizará a opção `delay_key_write` com CREATE TABLE. Isto significa que o buffer de chaves das tabelas com esta opção não serão descarregadas a cada atualização do índice, mas somente quando a tabela é fechada. Isto irá aumentar bem a velocidade de escrita em chaves, mas você deve adicionar verificação automática de todas as tabelas com `myisamchk --fast --force` se você usá-lo. Note que se você iniciar o `mysqld` com a opção `--delay-key-write-for-all-tables` significa que todas as tabelas serão tratadas como se fossem criadas com a opção `delay_key_write`. Você pode limpar esta configuração iniciando `mysqld` com `--skip-new` ou `--safe-mode`.

delayed_insert_limit

Depois de inserir `delayed_insert_limit` registros, o agente que cuida de INSERT DELAYED irá conferir se existem instruções SELECT pendentes. Se sim, ele permite a execução destas antes de continuar.

delayed_insert_timeout

Quanto tempo uma thread INSERT DELAYED deve esperar por instruções INSERT antes de terminar.

delayed_queue_size

Qual tamanho deve ser alocado para a fila (em linhas) para lidar com INSERT DELAYED. Se a fila encher, algum cliente que executar INSERT DELAYED irá esperar até existir espaço na fila novamente.

- flush** É habilitado (ON) se você iniciar o MySQL com a opção `--flush`.
- flush_time**
Se esta variável for configurada com um valor diferente de zero, então a cada `flush_time` segundos todas tabelas serão fechadas (para economizar recursos e sincronizar dados com o disco). Recomendamos esta opção somente em sistemas com Win95, Win98 ou outros sistemas com poucos recursos.
- have_bdb** YES se o `mysqld` suportar tabelas Berkeley DB. DISABLED se a opção `--skip-bdb` for usada.
- have_innodb**
YES se o `mysqld` suportar tabelas InnoDB. DISABLED se a opção `--skip-innodb` for usada.
- have_raid**
YES se o `mysqld` suportar a opção RAID.
- have_ssl** YES se o `mysqld` suportar SSL (criptografia) no protocolo cliente/ servidor.
- init_file**
O nome do arquivo especificado com a opção `--init-file` quando você iniciar o servidor. Este é um arquivo das instruções SQL que você deseja que o servidor execute quando é iniciado.
- interactive_timeout**
O número de segundos que o servidor espera por atividade em uma conexão antes de fechá-la. Um cliente interativo é definido como um cliente que utiliza a opção `CLIENT_INTERACTIVE` para `mysql_real_connect()`. Veja também `wait_timeout`.
- join_buffer_size**
O tamanho do buffer que é utilizado para full joins (joins que não utilizam índices). O buffer é alocado uma vez para cada full join entre duas tabelas. Aumente este valor para obter um full join mais rápido quando a adição de índices não for possível. (Normalmente a melhor forma de obter joins rápidas é adicionar índices.)
- key_buffer_size**
Blocos de índices são buferizados e compartilhados por todas as threads. `key_buffer_size` é o tamanho do buffer utilizado para indexar blocos.

Aumente-o para lidar melhor com os índices (para todas as leituras e escritas múltiplas) para o máximo possível 64M em uma máquina com 256M que executa, principalmente, o MySQL é bastante comum. Entretanto, se você deixar este valor muito grande (mais que 50% da sua memória total?) seu sistema pode iniciar a paginar e se tornar MUITO lento. Lembre-se que como o MySQL não utiliza cache de leitura de dados, será necessário deixar algum espaço para o cache do sistema de arquivos para o Sistema Operacional.

Você pode verificar a performance do buffer de chaves executando `show status` e examinar as variáveis `Key_read_requests`, `Key_reads`, `Key_write_requests` e `Key_writes`. A razão de `Key_reads/Key_read_request` deve normalmente

ser < 0.01 . O `Key_write/Key_write_requests` é normalmente próximo de 1 se você estiver utilizando na maioria updates/deletes mas deve ser bem menor se você tender a fazer atualizações que afetam várias outras ao mesmo tempo ou se você estiver utilizando `delay_key_write`. See [\(undefined\) \[SHOW\]](#), page [\(undefined\)](#).

Para obter ainda mais velocidade quando estiver gravando vários registros ao mesmo tempo, utilize `LOCK TABLES`. See [\(undefined\) \[LOCK TABLES\]](#), page [\(undefined\)](#).

`language` A linguagem utilizada para mensagens de erro.

`large_file_support`

Se o `mysqld` foi compilado com opções para suporte a grandes arquivos.

`locked_in_memory`

Se o `mysqld` foi travado na memória com `--memlock`

`log` Se o log de todas as consultas está habilitado.

`log_update`

Se o log de atualizações está habilitado.

`log_bin` Se o log binários está habilitado.

`log_slave_updates`

Se as atualizações do escravo devem ser logadas.

`long_query_time`

Se uma consulta demorar mais que isto (em segundos), o contador `Slow_queries` ser incrementado. Se você estiver utilizando `--log-slow-queries`, a consulta será logada ao arquivo de consultas lentas. See [\(undefined\) \[Slow query log\]](#), page [\(undefined\)](#).

`lower_case_table_names`

Se estiver configurado para 1, nomes de tabelas são armazenados em letras minúsculas no disco e nomes de tabelas serão caso-insensitivo. See [\(undefined\) \[Name case sensitivity\]](#), page [\(undefined\)](#).

`max_allowed_packet`

O valor máximo de um pacote. O buffer de mensagens é iniciado por `net_buffer_length` bytes, mas pode crescer até `max_allowed_packet` bytes quando for necessário. Este valor por padrão é pequeno, para obter pacotes maiores (possivelmente errados). Você deve incrementar este valor se você estiver usando colunas BLOB grandes. Ele deve tão grande quanto o maior BLOB que você deseja utilizar. O protocolo atual limita o `max_allowed_packet` à 16M.

`max_binlog_cache_size`

Se uma transação multi-instruções necessitar de mais que este montante de memória, será obtido o erro "Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage" ("Transação multi-instruções necessita mais que 'max_binlog_cache_size' bytes de armazenamento").

max_binlog_size

Disponível a partir da 3.23.33. Se uma escrita ao log binário (replicação) exceder o valor fornecido, rotacione os logs. Você não pode configurá-lo para menos de 1024 bytes ou mais que 1 GB. O valor padrão é 1 GB.

max_connections

O Número de clientes simultâneos permitidos. Aumentar este valor aumente o número de descritores de arquivos que o `mysqld` necessita. Veja abaixo os comentários sobre os limites de descritores de arquivos. See [\(undefined\)](#) [Muitas conexões], page [\(undefined\)](#).

max_connect_errors

Se houver mais que este número de conexões interrompidas a partir de uma máquina está máquina terá as próximas conexões bloqueadas. Você pode desbloquear uma máquina com o comando `FLUSH HOSTS`.

max_delayed_threads

Não inicie mais do que este número de threads para lidar com instruções `INSERT DELAYED`. Se você tentar inserir dados em uma nova tabela depois que todas as threads `INSERT DELAYED` estiverem em uso, o registro será inserido como se o atributo `DELAYED` não fosse especificado.

max_heap_table_size

Não permita criação de tabelas heap maiores que este valor.

max_join_size

Joins que provavelmente forem ler mais que `max_join_size` registros retornam um erro. Configure este valor se os seus usuários tendem a realizar joins que não possuem uma cláusula `WHERE`, que tomam muito tempo, e retornam milhões de registros.

max_sort_length

O número de bytes utilizados para ordenar valores `BLOB` ou `TEXT` (somente os primeiros `max_sort_length` bytes de cada valor são usados; o resto é ignorado).

max_user_connections

O valor máximo de conexões ativas para um único usuário (0 = sem limite).

max_tmp_tables

(Esta opção ainda não faz nada.) Número máximo de tabelas temporárias que um cliente pode manter abertas ao mesmo tempo.

max_write_lock_count

Depois desta quantidade de bloqueios de escrita, permite que alguns bloqueios de leitura sejam executados.

myisam_recover_options

O valor da opção `--myisam-recover`.

myisam_sort_buffer_size

O buffer que é alocado ao ordenar o índice quando estiver fazendo um `REPAIR` ou estiver criando índices com `CREATE INDEX` ou `ALTER TABLE`.

myisam_max_extra_sort_file_size.

Se a criação do arquivo temporário para criação rápida de índices fosse este valor maior que quando for usado o cache de chaves, de preferência ao método de cache de chaves. Isto é usado principalmente para forçar que longas chaves de caracteres em tabelas grandes usem o método de cache de chaves mais lenta para criar o índice. **NOTE** que este parâmetro é fornecido em megabytes!

myisam_max_sort_file_size

O tamanho máximo do arquivo temporário que é permitido ao MySQL usar enquanto recria os índices (durante REPAIR, ALTER TABLE ou LOAD DATA INFILE). Se o tamanho do arquivo for maior que isto, o índice será criado através do cache de chaves (que é mais lento). **NOTE** que este parâmetro é fornecido em megabytes!

net_buffer_length

O buffer de comunicações é configurado para este tamanho entre queries. Isto não deve ser alterado normalmente, mas se você tem muito pouca memória, pode configurá-lo para o tamanho esperado de uma consulta. (Isto é, o tamanho esperado das instruções SQL enviadas pelos clientes. Se as instruções excederem este valor, o buffer é aumentado automaticamente, até max_allowed_packet bytes.)

net_read_timeout

Número de segundos para esperar por mais dados de uma conexão antes de abortar a leitura. Perceba que quando nós não esperamos dados de uma conexão, o tempo máximo de espera é definido pelo write_timeout. Veja também slave_read_timeout.

net_retry_count

Se uma leitura na porta de comunicações for interrompida, tente novamente net_retry_count vezes antes de parar. Este valor deve ser bem alto no FreeBSD já que interrupções internas são enviadas para todas as threads.

net_write_timeout

Número de segundos para esperar pela escrita de um bloco em uma conexão antes de abortar a escrita.

open_files_limit

Se não for 0, então o mysqld usará este valor para reservar descritores de arquivos para usar com setrlimit(). Se este valor for 0 então o mysqld irá reservar max_connections*5 ou max_connections + table_cache*2 (que é maior) número de arquivos. Você deve tentar aumentar isto se o mysqld fornecer a você o erro 'Too many open files' (Muitos arquivos abertos).

pid_file O valor da opção --pid-file.

port O valor da opção --port.

protocol_version

A versão do protocolo usada pelo servidor MySQL.

record_buffer

Cada thread que faz uma leitura sequencial aloca um buffer deste tamanho para cada tabela lida. Se você fizer várias leituras sequenciais, você pode desejar aumentar este valor.

record_rnd_buffer

Ao ler registros na ordem depois de uma ordenação, os registros são lidos através deste buffer para evitar pesquisas em disco. Se não for configurado, recebe o valor de `record_buffer`.

query_buffer_size

A alocação inicial do buffer de consultas. Se a maioria de suas consultas são grandes (como quando se insere blobs), você deve aumentá-lo!

safe_show_databases

Não exibe bancos de dados nos quais o usuário não tem nenhum privilégios. Isto pode melhorar a segurança se você se preocupa com o fato das pessoas estarem aptas a ver quais bancos de dados outros usuários possuem. Veja também `skip_show_databases`.

server_id

O valor da opção `--server-id`.

skip_locking

Está desligado (OFF) se o `mysqld` usar bloqueio externo.

skip_networking

Está ligado (ON) se somente permitimos conexões locais (socket).

skip_show_databases

Isto previne usuários de fazerem `SHOW DATABASES` se eles não possuem o privilégio `PROCESS_PRIV`. Isto pode aumentar a segurança se você se preocupa com o fato das pessoas poderem ver quais bancos de dados outros usuários possuem. Veja também `safe_show_databases`.

slave_read_timeout

Número de segundos para esperar por mais dados de uma conexão de mestre/escravo antes de abortar a leitura.

slow_launch_time

Se a criação de threads demorar mais que este valor (em segundos), o contador `Slow_launch_threads` será incrementado.

socket

O socket Unix utilizado pelo servidor.

sort_buffer

Cada thread que precisar fazer uma ordenação aloca um buffer deste tamanho. Aumente este valor para operações `ORDER BY` ou `GROUP BY` mais rápidas. See [\[Arquivos temporários\]](#), page [\[Arquivos temporários\]](#).

table_cache

O número de tabelas abertas para todas as threads. Aumentar este valor aumenta o número de descritores de arquivos que o `mysql` necessita. O MySQL precisa de dois descritores de arquivos para cada tabela única aberta. Veja

abaixo os comentários sobre os limites do descritor de arquivos. Você pode conferir se necessita aumentar o cache de tabela conferindo a variável `Opened_tables`. See [\[SHOW\]](#), page [\[SHOW\]](#). Se esta variável for grande e você não faz muitos `FLUSH TABLES` (que apenas força todas as tabelas a serem fechadas e reabertas), então você deve aumentar o valor desta variável.

Tenha certeza que seu sistema operacional pode lidar com o número de descritores de arquivos abertos configurado em `table_cache`. Se o `table_cache` estiver configurado muito alto, o MySQL pode executar sem os descritores de arquivos e recusar conexões, falhar ao realizar consultas e se tornar muito instável. Para informações sobre como o cache de tabelas funciona, veja [\[Table cache\]](#), page [\[Table cache\]](#).

`table_type`

O tipo padrão de tabelas.

`thread_cache_size`

Quantas threads devem ser mantidas em cache para reutilização. Quando um cliente desconecta, as threads dos clientes são colocadas no cache se não existir mais de `thread_cache_size` threads que antes. Todas novas threads serão obtidas primeiramente do cache, e só quando o cache estiver vazio uma nova thread é criada. Esta variável pode ser aumentada para melhorar a performance se você tiver várias conexões novas. (Normalmente isto não dá uma melhora de performance notável se você possuir uma boa implementação de threads.) Examinando as diferenças entre `Connections` e `Threads_create` pode ser visto o quão eficiente é o cache de threads atual.

`thread_concurrency`

No Solaris, `mysqld` irá chamar `thr_setconcurrency()` com este valor. `thr_setconcurrency=()` permite que a aplicação forneça ao sistema de threads uma dica com o número desejado de threads que devem ser executados ao mesmo tempo.

`thread_stack`

O tamanho da pilha para cada thread. Vários dos limites detectados pelo teste `crash-me` são dependentes deste valor. O padrão é grande o suficiente para operações normais. See [\[Benchmarks do MySQL\]](#), page [\[Benchmarks do MySQL\]](#).

`timezone` O fuso horário para este servidor.

`tmp_table_size`

Se uma tabela temporária em memória exceder este tamanho, o MySQL irá a convertê-la automaticamente para uma tabela `MyISAM` em disco. Aumente o valor de `tmp_table_size` se você fizer várias consultas `GROUP BY` avançadas e você tiver muita memória.

`tmpdir` O diretório utilizado para arquivos temporários e tabelas temporárias.

`version` O número da versão do servidor.

wait_timeout

O número de segundos que o servidor espera pela atividade em uma conexão antes de fechá-la. Veja também `interactive_timeout`.

A seção do manual que descreve a sintonia do MySQL contém algumas informações de como sintonizar as variáveis acima. See [\[Parâmetros do servidor\]](#), page [\[Parâmetros do servidor\]](#).

4.5.5.5 SHOW LOGS

`SHOW LOGS` exibe estatísticas sobre os arquivos log existentes. Atualmente ele só exibe informações sobre arquivos de log Berkeley DB.

- `File` exibe o caminho completo para o arquivo de log
- `Type` exibe o tipo do arquivo log (BDB para arquivos de log Berkeley DB)
- `Status` exibe o status do arquivo log (FREE se o arquivo pode ser removido, ou IN USE se o arquivo é necessário para o subsistema de transações)

4.5.5.6 SHOW PROCESSLIST

`SHOW PROCESSLIST` exibe quais threads estão em execução. Esta informação também pode ser obtida utilizando o comando `mysqladmin processlist`. Se você possuir o privilégio `process`, poderá ver todas as threads. Senão só é possível ver as próprias threads. See [\[KILL\]](#), page [\[KILL\]](#). Se você não utiliza a opção `FULL`, então somente os primeiros 100 caracteres de cada query serão exibidos.

Este comando é muito útil caso você obtenha a mensagem de erro 'too many connections' e deseja saber o que está ocorrendo. O MySQL reserva uma conexão extra por cliente com o privilégio `Process_priv` para garantir que você sempre consiga logar e conferir o sistema (assumindo que este privilégio não foi concedido para todos os usuários).

4.5.5.7 SHOW GRANTS

`SHOW GRANTS FOR usuário` lista os comandos concedidos que devem ser usados para duplicar os direitos de um usuário.

```
mysql> SHOW GRANTS FOR root@localhost;
+-----+
| Grants for root@localhost |
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+-----+
```

4.5.5.8 SHOW CREATE TABLE

Exibe uma instrução `CREATE TABLE` que irá criar a seguinte tabela:

```
mysql> show create table t\G
***** 1. row *****
      Table: t
Create Table: CREATE TABLE t (
  id int(11) default NULL auto_increment,
  s char(60) default NULL,
```

```

PRIMARY KEY (id)
) TYPE=MyISAM

```

`SHOW CREATE TABLE` irá citar os nomes de colunas e tabelas de acordo com a opção `SQL_QUOTE_SHOW_CREATE`. [\[SET OPTION SQL_QUOTE_SHOW_CREATE\]](#), [page <undefined>](#).

4.6 Localização do MySQL e Utilização Internacional

4.6.1 O Conjunto de Caracteres Utilizado para Dados e Ordenação

Por padrão, o MySQL utiliza o conjunto de caracteres ISO-8859-1 (Latin1) com ordenação de acordo com o sueco/finlandês. Este também é o conjunto de caracteres aplicável nos EUA e oeste da Europa.

Todos os binários padrões do MySQL são compilados com `--with-extra-charsets=complex`. Isto adicionará código a todos os programas padrões para estarem aptos a lidar com o conjuntos de caracteres `latin1` e todos os multi-byte no binário. Outros conjuntos de caracteres serão carregados de um arquivo de definições de conjuntos de caracteres quando necessários.

O conjunto de caracteres determina quais são os caracteres permitidos em nomes e qual a forma de ordenação por cláusulas `ORDER BY` e `GROUP BY` da instrução `SELECT`.

Você pode alterar o conjunto de caracteres com a opção `--default-character-set` na inicialização do servidor. Os conjuntos de caracteres disponíveis dependem dos parâmetros `--with-charset=charset` e `--with-extra-charset= list-of-charset | complex | all` e os arquivos de configurações de conjuntos de caracteres listados em `'SHAREDIR/charsets/Index'`. See [\[Opções de configurações\]](#), [page <undefined>](#).

Se o conjunto de caracteres for alterado durante a execução do MySQL (que também pode alterar a ordenação), deve-se executar o `myisamchk -r -q` em todas as tabelas. De outra forma seus índices podem não ser ordenados corretamente.

Quando um cliente conecta a um servidor MySQL, o servidor envia o conjunto de caracteres padrão em uso ao cliente. O cliente irá alternar para o uso deste conjunto de caracteres nesta conexão.

Deve ser utilizado `mysql_real_escape_string()` quando desejar ignorar sequências de caracteres em uma consulta SQL. `mysql_real_escape_string()` é idêntico à antiga função `mysql_escaped_string()`, exceto pelo fato de usar a manipulador de conexão MySQL como o primeiro parâmetro.

Se o cliente for compilado com o caminho diferente daquele onde o servidor está instalado e o usuário que configurou o MySQL não incluiu todos os conjuntos de caracteres no binários do MySQL, deve ser especificado para o cliente onde ele pode encontrar os conjuntos de caracteres adicionais que serão necessários se o servidor executar com um conjunto de caracteres diferente do cliente.

Isto pode ser especificado colocando em um arquivo de opções do MySQL:

```
[client]
```

```
character-sets-dir=/usr/local/mysql/share/mysqlCharsets
```

onde o caminho aponta para onde os conjuntos de caracteres dinâmicos do MySQL são armazenados.

Pode-se forçar o cliente a usar conjuntos de caracteres específicos especificando:

```
[client]
```

```
default-character-set=nome-conjunto-caracteres
```

mas normalmente isto nunca será necessário.

4.6.2 Mensagens de Erros em Outras Línguas

`mysqld` pode exibir mensagens de erros nas seguintes línguas: Tcheco, Dinamarquês, Holandês, Inglês (padrão), Estonian, Francês, Alemão, Grego, Húngaro, Italiano, Japonês, Coreano, Norueguês, Norueguês-ny, Polonês, Português, Romeno, Russo, Eslovaco, Espanhol e Sueco.

Para iniciar o `mysqld` com uma língua particular, use uma das opções: `--language=língua` ou `-L língua`. Por exemplo:

```
shell> mysqld --language=swedish
```

ou:

```
shell> mysqld --language=/usr/local/share/swedish
```

Perceba que todos as línguas são especificados em minúsculas.

Os arquivos de linguagens estão localizados (por padrão) em `'mysql.base.dir/share/LINGUA/'`.

Para atualizar o arquivo com mensagens de erros, deve-se editar o arquivo `'errmsg.txt'` e executar o seguinte comando para gerar o arquivo `'errmsg.sys'`:

```
shell> comp_err errmsg.txt errmsg.sys
```

Se você atualizar o MySQL para uma versão mais nova, lembre-se de repetir as alterações no novo arquivo `'errmsg.txt'`.

4.6.3 Adicionando um Novo Conjunto de Caracteres

Para adicionar outro conjunto de caracteres ao MySQL, utilize o seguinte procedimento.

Decida se o conjunto é simples ou complexo. Se o conjunto de caracteres não necessitar do uso de rotinas especiais de classificação de strings para ordenação e também não necessitar de suporte à caracteres multi-byte, será simples. Se ele necessitar de alguns destes recursos, será complexo.

Por exemplo, `latin1` e `danish` são conjuntos simples de caracteres enquanto `big5` ou `czech` são conjuntos de caracteres complexos.

Na seguinte seção, assumimos que você nomeou seu conjunto de caracteres como `MYSET`.

Para um conjunto de caracteres simples use o seguinte:

1. Adicione `MYSET` para o final do arquivo `'sql/shareCharsets/Index'` Associe um número único ao mesmo.
2. Crie o arquivo `'sql/shareCharsets/MYSET.conf'`. (O arquivo `'sql/shareCharsets/latin1.conf'` pode ser utilizado como base para isto).

A sintaxe para o arquivo é muito simples:

- Comentários iniciam com um caractere '#' e continuam até o fim da linha.
- Palavras são separadas por quantidades arbitrárias de espaços em brancos.
- Ao definir o conjunto de caracteres, cada palavra deve ser um número no formato hexadecimal
- O vetor `ctype` obtêm as primeiras 257 palavras. Os vetores `to_lower`, `to_upper` e `sort_order` obtêm, cada um, as 256 palavras seguintes.

See [\[Vetor de caracteres\]](#), page [\[Vetor de caracteres\]](#).

3. Adicione o nome do conjunto de caracteres às listas `CHARSETS_AVAILABLE` e `COMPILED_CHARSETS` no `configure.in`.
4. Reconfigure, recompile e teste.

Para um conjunto de caracteres complexo faça o seguinte:

1. Crie o arquivo `'strings/ctype-MYSET.c'` na distribuição fonte do MySQL.
2. Adicione `MYSET` ao final do arquivo `'sql/share/charsets/Index'`. Associe um número único a ele.
3. Procure por um dos arquivos `'ctype-*.c'` existentes para ver o que precisa ser definido, por exemplo `'strings/ctype-big5.c'`. Perceba que os vetores no seu arquivo deve ter nomes como `ctype_MYSET`, `to_lower_MYSET` e etc. Isto corresponde aos arrays no conjunto simples de caracteres - See [\[Vetor de caracteres\]](#), page [\[Vetor de caracteres\]](#) - para um conjunto de caracteres complexo.
4. Próximo ao topo do arquivo, coloque um comentário especial como este:

```
/*
 * This comment is parsed by configure to create ctype.c,
 * so don't change it unless you know what you are doing.
 *
 * .configure. number_MYSET=MYNUMBER
 * .configure. strxfrm_multiply_MYSET=N
 * .configure. mbmaxlen_MYSET=N
 */
```

O programa `configure` utiliza este comentário para incluir o conjunto de caracteres na biblioteca MySQL automaticamente.

As linhas `strxfrm_multiply` e `mbmaxlen` serão explicadas nas próximas seções. Só as inclua se você precisar de funções de ordenação de strings ou das funções de conjuntos de caracteres multi-byte, respectivamente.

5. Você deve então criar algumas das seguintes funções:
 - `my_strncoll_MYSET()`
 - `my_strcoll_MYSET()`
 - `my_strxfrm_MYSET()`
 - `my_like_range_MYSET()`

See [\[Ordenação de caracteres\]](#), page [\[Ordenação de caracteres\]](#).

6. Adicione o nome do conjunto de caracteres às listas `CHARSETS_AVAILABLE` e `COMPILED_CHARSETS` no `configure.in`.
7. Reconfigure, recompile e teste.

O arquivo 'sql/share/charsets/README' fornece algumas instruções a mais.

Se você desejar ter o seu conjunto de caracteres incluído na distribuição MySQL, envie um email com um patch para internals@lists.mysql.com.

4.6.4 Os vetores de definições de caracteres

`to_lower[]` e `to_upper[]` são vetores simples que definem os caracteres minúsculos e maiúsculos correspondentes a cada membro do conjunto de caracteres. Por exemplo:

```
to_lower['A'] deve conter 'a'
to_upper['a'] deve conter 'A'
```

`sort_order[]` é um mapa indicando como os caracteres devem ser ordenados para propósitos de comparação e ordenação. Para vários conjuntos de caracteres, isto é o mesmo que `to_upper[]` (que significa ordenar em caso insensitivo). O MySQL ordenará caracteres baseado no valor de `sort_order[caractere]`. Para regras mais complicadas de ordenação, veja a discussão sobre ordenação de string abaixo. See [\[Ordenação de strings\]](#), page [\[Ordenação de strings\]](#).

`ctype[]` é um vetor com valores binários, com um elemento para cada caracter. (Note que `to_lower[]`, `to_upper[]` e `sort_order[]` são indexados pelo valor do caracter, mas o `ctype[]` é indexado pelo valor do caracter + 1. Este é um antigo legado para tratamento de EOF.)

Pode-se encontrar as seguintes máscaras binárias de definições em 'm_ctype.h':

```
#define _U      01      /* Maiúsculo */
#define _L      02      /* Minúsculo */
#define _N      04      /* Numeral (digito) */
#define _S      010     /* Caractere de espaço */
#define _P      020     /* Pontuação */
#define _C      040     /* Caractere de controle */
#define _B      0100    /* Branco */
#define _X      0200    /* Digito hexadecimal */
```

A entrada `ctype[]` para cada caracter deve ser a união dos valores da máscara binária que descrevem o caracter. Por exemplo, 'A' é um caracter maiúsculo (`_U`) bem como um dígito hexadecimal (`_X`), portanto `ctype['A'+1]` deve conter o valor:

```
_U + _X = 01 + 0200 = 0201
```

4.6.5 Suporte à Ordenação de Strings

Se as regras de ordenação para a sua linguagem forem muito complexas para serem tratadas com uma simples tabela `sort_order[]`, será necessário o uso das funções de ordenação de strings.

No momento, a melhor documentação sobre isto são os conjuntos de caracteres que já estão implementados. Confira os conjuntos de caracteres `big5`, `czech`, `gbk`, `sjis` e `tis160` para exemplos.

Você deve especificar o valor `strxfrm_multiply_MYSET=N` no comentário especial no topo do arquivo. `N` deve ser configurado para a razão máxima que as strings podem crescer durante `my_strxfrm_MYSET` (ele deve ser um inteiro positivo).

4.6.6 Suporte à Caracteres Multi-byte

Se você deseja adicionar suporte para novos conjuntos de caracteres que incluem caracteres multi-byte, você precisa usar as funções para caracteres multi-byte.

No momento, a melhor documentação sobre isto são os conjuntos de caracteres que já estão implementados. Confira os conjuntos de caracteres `euc_kr`, `gb2312`, `gbk`, `sjis` e `ujis` para exemplos. Eles são implementados no arquivo `ctype-'conj_caracter'.c` no diretório `'strings'`

Você deve especificar o valor `mbmaxlen_MYSET=N` no comentário especial no topo do arquivo. `N` deve ser configurado como o tamanho em bytes do maior caracter no conjunto.

4.7 Utilitários e Scripts do Lado Servidor MySQL

4.7.1 Visão Geral dos Scripts e Utilitários do Lado Servidor

Todos os clientes MySQL que comunicam com o servidor utilizando a biblioteca `mysqlclient` usam as seguintes variáveis de ambiente:

Nome	Descrição
<code>MYSQL_UNIX_PORT</code>	O socket padrão; utilizado para conexões à <code>localhost</code>
<code>MYSQL_TCP_PORT</code>	A porta TCP/IP padrão
<code>MYSQL_PWD</code>	A senha padrão
<code>MYSQL_DEBUG</code>	Opções de depuração-rastreamento ao debugar
<code>TMPDIR</code>	O diretório onde tabelas e arquivos temporários são criados

A utilização de `MYSQL_PWD` é insegura. See [\(undefined\) \[Conectando\]](#), page [\(undefined\)](#).

O cliente `'mysql'` utiliza o arquivo nomeado na variável de ambiente `MYSQL_HISTFILE` para salvar o histórico da linha de comando. O valor padrão para o arquivo de histórico é `'$HOME/.mysql_history'` onde `$HOME` é o valor da variável de ambiente `HOME`. See [\(undefined\) \[Variáveis de ambiente\]](#), page [\(undefined\)](#).

Todos os programas MySQL possuem várias opções diferentes, entretanto, todo programa MySQL fornece uma opção `--help` que pode ser usada para obter uma descrição completa das diferentes opções do programa. Por exemplo, experimente `mysql --help`.

Você pode sobrepor ignorar as opções padrões para todos os programas clientes com um arquivo de opções. [\(undefined\) \[Arquivo de opções\]](#), page [\(undefined\)](#).

A lista abaixo descreve brevemente os programas MySQL.

`myisamchk`

Utilitário para descrever, conferir, otimizar e reparar tabelas MySQL. Como o `myisamchk` tem muitas funções, eles são descritos em seu próprio capítulo. See [\(undefined\) \[Administração do Banco de Dados MySQL\]](#), page [\(undefined\)](#).

`make_binary_distribution`

Cria uma edição binária de um MySQL compilado. Isto pode ser enviado por FTP para `'/pub/mysql/Incoming'` em `support.mysql.com` para a conveniência de outros usuários MySQL.

msql2mysql

Um script shell que converte programas mSQL para MySQL. Ele não consegue lidar com todos os casos, mas é um bom começo quando uma conversão for necessária.

mysqlaccess

Um script que confere os privilégios de acesso para uma combinação de máquina, usuário e banco de dados.

mysqladmin

Utilitário para realizar operações administrativas, como a criação ou remoção de bancos de dados, recarga das tabelas de permissões, descarga de tabelas ao disco e reabertura de arquivos log. **mysqladmin** pode também ser utilizado para exibir informações de versão, processos e estado do servidor. See [\[mysqladmin\]](#), page [\[mysqladmin\]](#).

mysqlbug O script para relatar erros no MySQL. Este script deve sempre ser utilizado quando for necessário preencher um relatório de erros para a lista do MySQL.

mysqld O servidor (daemon) SQL. Deve sempre estar em execução.

mysqldump

Descarrega um banco de dados MySQL em um arquivo no formato de instruções SQL ou como arquivos texto separado por tabulações. Versão melhorada do freeware produzido originalmente por Igor Romanenko. See [\[mysqldump\]](#), page [\[mysqldump\]](#).

mysqlimport

Importa arquivos texto em suas respectivas tabelas utilizando LOAD DATA INFILE. See [\[mysqlimport\]](#), page [\[mysqlimport\]](#).

mysqlshow

Exibe informações sobre bancos de dados, tabelas, colunas e índices.

mysql_install_db

Cria as tabelas de permissões do MySQL com os privilégios padrões. Este comando normalmente é executado somente na primeira vez quando o MySQL é instalado em um sistema.

replace Um programa utilitário que é utilizado pelo **msql2mysql**, mas também pode ser utilizado para aplicações gerais. **replace** altera strings em arquivos ou na entrada padrão. Utiliza uma máquina de estado finito para coincidir primeiramente com strings maiores. Pode ser utilizado para trocar strings. Por exemplo, este comando troca a e b nos arquivos fornecidos:

```
shell> replace a b b a -- arquivo1 arquivo2 ...
```

4.7.2 safe_mysqld, a capa do mysqld

safe_mysqld é a maneira recomendada para iniciar um daemon **mysqld** no Unix. **safe_mysqld** adiciona alguns recursos de segurança tais como reiniciar o servidor quando um erro ocorrer e log de informações de tempo de execução a um arquivo log.

Se você não utilizar `--mysqld=#` ou `--mysql-version=#` o `safe_mysqld` irá utilizar um executável chamado `mysqld-max` se ele existir. Se não, `safe_mysqld` irá iniciar o `mysqld`. Isto torna muito fácil utilizar o `mysqld-max` no lugar do `mysqld`; basta copiar `mysqld-max` no mesmo diretório do `mysqld` e ele será utilizado.

Normalmente o script `safe_mysqld` nunca deve ser editado, em vez disto, coloque as opções para o `safe_mysqld` na seção `[safe_mysqld]` no arquivo `my.cnf`. O `safe_mysqld` irá ler todas as opções das seções `[mysqld]`, `[server]` e `[safe_mysqld]` dos arquivos de opções. See [\[Arquivo de opções\], page \[undefined\]](#).

Note que todas as opções na linha de comando para o `safe_mysqld` são passadas para o `mysqld`. Se você deseja usar algumas opções no `safe_mysqld` que o `mysqld` não suporte, você deve especificá-las no arquivo de opções.

A maioria das opções para `safe_mysqld` são as mesmas que as do `mysqld`. See [line options-snt \[Opções de linha de comando\], page line options-pg](#).

`safe_mysqld` suporta as seguintes opções:

```
--basedir=caminho
--core-file-size=#
    Tamanaho do arquivo core que o mysqld poderá criar. Passado para ulimit
    -c.

--datadir=caminho
--defaults-extra-file=caminho
--defaults-file=caminho
--err-log=caminho
--ledir=caminho
    Caminho para mysqld

--log=caminho
--mysqld=versão_do_mysqld
    Nome da versão do mysqld no diretório ledir que você deseja iniciar.

--mysqld-version=versão
    Similar ao --mysqld= mas aqui você só fornece o sufixo para o mysqld. Por
    exemplo, se você utiliza --mysqld-version=max, o safe_mysqld irá iniciar a
    versão ledir/mysqld-max. Se o argumento para --mysqld-version estiver
    vazio, ledir/mysqld será usado.

--no-defaults
--open-files-limit=#
    Número de arquivos que o mysqld poderá abrir. Passado para ulimit -n.
    Perceba que será necessário iniciar safe_mysqld como root para isto funcionar
    corretamente!

--pid-file=caminho
--port=#
--socket=caminho
--timezone=#
    Configura a variável de fuso horário (TZ) para o valor deste parâmetro.

--user=#
```

O script `safe_mysqld` é gravável, portanto ele deve estar apto para iniciar um servidor que foi instalado de uma fonte ou uma versão binária do MySQL, mesmo se o servidor estiver instalado em localizações um pouco diferentes. `safe_mysqld` espera uma destas condições ser verdadeira:

- O servidor e o banco de dados pode ser encontrado relativo ao diretório de onde o `safe_mysqld` foi chamado. `safe_mysqld` procura dentro de seu diretório de trabalho pelos diretórios `'bin'` e `'data'` (para distribuições binárias) ou pelos diretórios `'libexec'` e `'var'` (para distribuições baseadas em código fonte). Esta condição deve ser satisfeita se você executar o `safe_mysqld` a partir do seu diretório da instalação do MySQL (por exemplo, `'/usr/local/mysql'` para uma distribuição binária).
- Se o servidor e os bancos de dados não forem encontrados relativos ao diretório de trabalho, `safe_mysqld` tenta localizá-lo utilizando caminhos absolutos. Localizações típicas são `'/usr/local/libexec'` e `'/usr/local/var'`. As localizações atuais foram determinadas quando a distribuição foi construída da qual vem o `safe_mysqld`. Eles dever estar corretas se o MySQL foi instalado na localização padrão.

Como o `safe_mysqld` tentará encontrar o servidor e o banco de dados relativo a seu diretório de trabalho, você pode instalar uma distribuição binária do MySQL em qualquer lugar, desde de que o `safe_mysqld` seja iniciado a partir do diretório da instalação:

```
shell> cd diretório_instalação_mysql
shell> bin/safe_mysqld &
```

Se o `safe_mysqld` falhar, mesmo se invocado a partir do diretório de instalação do MySQL, você pode modificá-lo para usar o caminho para o `mysqld` e as opções de caminho que seriam corretas para seu sistema. Perceba que se você atualizar o MySQL no futuro, sua versão modificada de `safe_mysqld` será sobrescrita, portanto, você deve fazer uma cópia de sua versão editada para que você a possa reinstalar.

4.7.3 `mysqld_multi`, programa para gerenciar múltiplos servidores MySQL

`mysqld_multi` gerencia vários processos `mysqld` executando em diferentes sockets UNIX e portas TCP/IP.

O programa irá pesquisar pelo(s) grupo(s) chamado(s) `[mysqld#]` no `my.cnf` (ou no arquivo fornecido no parâmetro `--config-file=...`), onde `#` pode ser qualquer número positivo a partir de 1. Estes grupos devem ser os mesmos que o grupo `[mysqld]` usual (ex.: opções para o `mysqld`, veja o manual do MySQL para informações mais detalhadas sobre este grupo), mas com as opções de portas, sockets e etc. que são necessárias para cada processo separado do `mysqld`. O número no nome do grupo tem outra função; ele pode ser utilizado para iniciar, parar ou relatar alguns servidores `mysqld` específicos com este programa. Veja o uso e as opções abaixo para maiores informações.

```
Uso: mysqld_multi [OPÇÕES] {start|stop|report} [GNR,GNR,GNR... ]
ou mysqld_multi [OPÇÕES] {start|stop|report} [GNR-GNR,GNR,GNR-GNR,... ]
```

O GNR acima significa o número do grupo. Você pode iniciar, parar ou relacionar qualquer GNR ou vários deles ao mesmo tempo. (Veja `--example`). A lista dos GNR podem ser separadas por vírgulas, ou pelo sinal de menos (-), sendo que o ultimo significa que todos os GNRS entre GNR1-GNR2 serão afetados. Sem o argumento GNR todos os grupos

encontrados serão iniciados, parados ou listados. Perceba que você não deve ter nenhum espaço em branco na lista GNR. Qualquer coisa depois de um espaço em branco é ignorado. `mysqld_multi` suporta as seguintes opções:

```
--config-file=...
    Arquivo de configuração alternativo. NOTA: Isto não irá afetar as próprias
    opções do programa (grupo [mysqld_multi]), mas somente grupos [mysqld#].
    Sem esta opção tudo será procurado a partir do arquivo my.cnf.

--example
    Fornece um exemplo de um arquivo de configuração.

--help
    Exibe esta ajuda e sai.

--log=...
    Arquivo Log. Deve ser informado o caminho completo e o nome do arquivo log.
    NOTA: se o arquivo existir, tudo será anexado.

--mysqladmin=...
    Binário mysqladmin a ser usado para o desligamento do servidor.

--mysqld=...
    Binário mysqld a ser usado. Lembre-se que você também pode fornecer safe_
    mysqld a esta opção. As opções são passadas ao mysqld. Apenas tenha certeza
    que o mysqld está localizado na sua variável de ambiente PATH ou corrija o
    safe_mysqld.

--no-log
    Imprime na saída padrão em vez do arquivo log. Por padrão o arquivo log
    sempre fica ligado.

--password=...
    Senha do usuário para o mysqladmin.

--tcp-ip
    Conecta ao(s) servidor(es) MySQL através de porta TCP/IP no lugar de socket
    UNIX. Isto afeta a ação de desligar e relatar. Se um arquivo socket estiver
    faltando, o servidor pode ainda estar executando, mas só pode ser acessado
    através da porta TCP/IP. Por padrão a conexão é feita através de socket UNIX.

--user=...
    Usuário MySQL para o mysqladmin.

--version
    Exibe o número da versão e sai.
```

Algumas notas sobre `mysqld_multi`:

- Tenha certeza que o usuário MySQL, que finalizar os serviços `mysqld` (e.g. utilizando o `mysqladmin`) tem a mesma senha e usuário para todos os diretórios de dados acessados (para o banco de dados 'mysql'). E tenha certeza que o usuário tem o privilégio 'Shutdown_priv'! Se você possui diversos diretórios de dados e vários bancos de dados 'mysql' com diferentes senhas para o usuário 'root' do MySQL, você pode desejar criar um usuário comum 'multi-admin' para cada um que utilize a mesma senha (veja abaixo). Exemplo de como fazer isto:

```
shell> mysql -u root -S /tmp/mysql.sock -psenha_root -e
"GRANT SHUTDOWN ON *.* TO multi_admin@localhost IDENTIFIED BY 'multipass'"
See <undefined> [Privileges], page <undefined>.
```

Você deve fazer isto para cada servidor `mysqld` executando em cada diretório de dados, que você tem (Apenas altere o socket, `-S=...`)

- `pid-file` é muito importante, se você estiver utilizando `safe_mysqld` para iniciar o `mysqld` (ex. `--mysqld=safe_mysqld`) Todos os `mysqld` devem ter seus próprios `pid-file`. A vantagem de utilizar o `safe_mysqld` no lugar de executar diretamente o `mysqld` é que `safe_mysqld` guarda todos os processos e irá reiniciá-los, se um processo do `mysqld` falhar devido a um sinal `kill -9`, ou similar. (Como um falha de segmentação, que nunca pode acontecer com o MySQL.) Por favor note que pode ser necessário executar o script `safe_mysqld` de um lugar específico. Isto significa que você pode ter que alterar o diretório atual para um diretório específico antes de iniciar o `mysqld_multi`. Se você tiver problemas ao iniciar, por favor veja o script `safe_mysqld`. Verifique especialmente as linhas:

```
-----
-----
MY_PWD='pwd' Check if we are starting this relative (for the binary
release) if test -d /data/mysql -a -f ./share/mysql/english/errmsg.sys
-a -x ./bin/mysqld
-----
-----
See <undefined> [safe_mysqld], page <undefined>.
```

O teste acima deve ser bem sucedido, ou você pode encontrar problemas.

- Esteja certo do perigoso de iniciar múltiplos `mysqlds` no mesmo diretório de dados. Utilize diretórios de dados diferentes, a menos que você realmente **SAIBA** o que está fazendo!
- O arquivo de socket e a porta TCP/IP devem ser diferentes para cada `mysqld`.
- O primeiro e quinto grupo `mysqld` foram intencionalmente deixados de lado no exemplo. Você pode ter lacunas no arquivo de configuração. Isto lhe permite mais flexibilidade. A ordem na qual os `mysqlds` são iniciados ou desligados depende da ordem em que eles aparecem no arquivo de configuração.
- Quando você desejar referenciar a um grupo específico utilizando GNR com este programa, basta utilizar o número no fim do nome do grupo (`[mysqld# <==>`).
- Você pode desejar utilizar a opção `'--user'` para o `mysqld`, mas para isto você precisa ser o usuário `root` quando iniciar o script `mysqld_multi`. Não importa se a opção existe no arquivo de configuração; você receberá apenas um alerta se você não for o superusuário e o `mysqlds` for iniciado com a **SUA** conta no Unix. **IMPORTANTE:** Tenha certeza que o `pid-file` e o diretório de dados é acessível para leitura e escrita (+execução para o diretório) para **ESTE** usuário UNIX que iniciará o processo `mysqld`. **NÃO** utilize a conta de `root` para isto, a menos que você **SAIBA** o que está fazendo!
- **MAIS IMPORTANTE:** Tenha certeza que você entendeu os significados das opções que são passadas para os `mysqlds` e porque **VOC PRECISARIA** ter processos `mysqld` separados. Iniciando múltiplos `mysqlds` em um diretório de dados **NÃO IRÁ** melhorar a performance em um sistema baseado em threads.

See [\(undefined\)](#) [Servidores múltiplos], page [\(undefined\)](#).

Este é um exemplo do arquivo de configuração para o funcionamento do `mysqld_multi`.

```
# Este arquivo provavelmente deve estar em seu diretório home (~/.my.cnf) ou /etc/m
# Version 2.1 by Jani Tolonen
```

```
[mysqld_multi]
mysqld      = /usr/local/bin/safe_mysqld
mysqladmin  = /usr/local/bin/mysqladmin
user        = multi_admin
password    = multipass

[mysqld2]
socket      = /tmp/mysql.sock2
port        = 3307
pid-file    = /usr/local/mysql/var2/hostname.pid2
datadir     = /usr/local/mysql/var2
language    = /usr/local/share/mysql/english
user        = john

[mysqld3]
socket      = /tmp/mysql.sock3
port        = 3308
pid-file    = /usr/local/mysql/var3/hostname.pid3
datadir     = /usr/local/mysql/var3
language    = /usr/local/share/mysql/swedish
user        = monty

[mysqld4]
socket      = /tmp/mysql.sock4
port        = 3309
pid-file    = /usr/local/mysql/var4/hostname.pid4
datadir     = /usr/local/mysql/var4
language    = /usr/local/share/mysql/estonia
user        = tonu

[mysqld6]
socket      = /tmp/mysql.sock6
port        = 3311
pid-file    = /usr/local/mysql/var6/hostname.pid6
datadir     = /usr/local/mysql/var6
language    = /usr/local/share/mysql/japanese
user        = jani
```

See [\(undefined\)](#) [Arquivo de opções], page [\(undefined\)](#).

4.7.4 myisampack, O Gerador de Tabelas Compactadas Somente Leitura

`myisampack` é usado para compactar tabelas MyISAM, e `pack_isam` é usado para compactar tabelas ISAM. Como as tabelas ISAM estão ultrapassadas, nós iremos discutir aqui somente

sobre o `myisampack`, mas tudo dito sobre `myisampack` também pode ser verdadeiro para o `pack_isam`.

`myisampack` trabalha compactando cada coluna na tabela separadamente. A informação necessária para descompactar colunas é lida em memória quando a tabela é aberta. Isto resulta em uma performance muito melhor quando estiver acessando registros individuais, porque você precisará descompactar somente um registro, não um bloco muito maior do disco como faz o `Stacker` no MS-DOS. Normalmente, `myisampack` compacta o arquivo de dados 40%-70%.

O MySQL utiliza mapeamento de memória (`nmap()`) em tabelas compactadas e retorna ao uso normal de leitura e escrita se `nmap()` não funcionar.

Atualmente existem duas limitações com `myisampack`:

- Depois da compactação, a tabela deve trabalhar somente para leitura.
- `myisampack` também pode compactar colunas BLOB ou TEXT. O antigo `pack_isam` não pode fazer isto.

A correção destas limitações está em nossa lista de pendências (TODO), mas com baixa prioridade.

`myisampack` é chamado desta forma:

```
shell> myisampack [opções] nome_arquivo ...
```

Cada `nome_arquivo` deve ter o nome de um arquivo de índice (`‘.MYI’`). Se você não se encontra em um diretório de bancos de dados, você deve especificar o caminho completo para o arquivo. Pode-se omitir a extensão `‘.MYI’`.

`myisampack` suporta as seguintes opções:

`-b, --backup`

Realiza um backup da tabela como `nome_tabela.OLD`.

`-#, --debug=debug_options`

Log da saída de depuração. A string `debug_options` geralmente é `‘d:t:o, nome_arquivo’`.

`-f, --force`

Força a compactação da tabela mesmo se ela se tornar maior ou se o arquivo temporário existir. `myisampack` cria um arquivo temporário chamado `‘nome_tabela.TMD’` enquanto ele compacta a tabela. Se você matar o `myisampack` o arquivo `‘.TMD’` não pode ser removido. Normalmente, `myisampack` sai com um erro se ele descobrir que `‘nome_tabela.TMD’` existe. Com `--force`, `myisampack` compacta a tabela de qualquer maneira.

`-?, --help`

Exibe uma mensagem de ajuda e sai.

`-j nome_tabela_grande, --join=nome_tabela_grande`

Une todas as tabelas nomeadas na linha de comando em uma única tabela `nome_tabela_grande`. Todas tabelas que forem combinadas DEVEM ser idênticas (mesmos nomes de colunas e tipos, alguns índices, etc.).

`-p #, --packlength=#`

Especifica o comprimento do tamanho de armazenamento, em bytes. O valor deve ser 1, 2 ou 3. (`myisampack` armazena todas as linhas com ponteiros de

tamanhos 1, 2 ou 3 bytes. Na maioria dos casos normais, `myisampack` pode determinar o valor correto do tamanho antes de começar a compactar o arquivo, mas ele pode notificar durante o processo de compactação que ele pode ter usado um tamanho menor. Neste caso `myisampack` irá exibir uma nota dizendo que a próxima vez que você compactar o mesmo arquivo você pode utilizar um registro de tamanho menor.)

- `-s, --silent`
Modo silencioso. Escreve a saída somente quando algum erro ocorrer.
- `-t, --test`
Não compacta realmente a tabela, apenas testa a sua compactação.
- `-T dir_name, --tmp_dir=dir_name`
Utiliza o diretório especificado como a localização em que serão gravadas as tabelas temporárias.
- `-v, --verbose`
Modo verbose. Escreve informação sobre o progresso e resultado da compactação.
- `-V, --version`
Exibe informação de versão e sai.
- `-w, --wait`
Espera e tenta novamente se a tabela estiver em uso. Se o servidor `mysqld` foi iniciado com a opção `--skip-locking`, não é uma boa idéia chamar `myisampack` se a tabela puder ser atualizada durante o processo de compactação.

A seqüência de comandos mostrados abaixo ilustra uma típica seção de compactação de tabelas:

```
shell> ls -l station.*
-rw-rw-r-- 1 monty  my      994128 Apr 17 19:00 station.MYD
-rw-rw-r-- 1 monty  my      53248  Apr 17 19:00 station.MYI
-rw-rw-r-- 1 monty  my       5767  Apr 17 19:00 station.frm
```

```
shell> myisamchk -dvv station
```

```
MyISAM file:      station
Isam-version:    2
Creation time:   1996-03-13 10:08:58
Recover time:    1997-02-02  3:06:43
Data records:    1192 Deleted blocks:      0
Datafile: Parts: 1192 Deleted data:      0
Datafile pointer (bytes): 2 Keyfile pointer (bytes): 2
Max datafile length: 54657023 Max keyfile length: 33554431
Recordlength:    834
Record format: Fixed length
```

```
table description:
```

Key	Start	Len	Index	Type	Root	Blocksize	Rec/key
-----	-------	-----	-------	------	------	-----------	---------

1	2	4	unique	unsigned long	1024	1024	1
2	32	30	multip.	text	10240	1024	1

Field	Start	Length	Type
-------	-------	--------	------

1	1	1	
2	2	4	
3	6	4	
4	10	1	
5	11	20	
6	31	1	
7	32	30	
8	62	35	
9	97	35	
10	132	35	
11	167	4	
12	171	16	
13	187	35	
14	222	4	
15	226	16	
16	242	20	
17	262	20	
18	282	20	
19	302	30	
20	332	4	
21	336	4	
22	340	1	
23	341	8	
24	349	8	
25	357	8	
26	365	2	
27	367	2	
28	369	4	
29	373	4	
30	377	1	
31	378	2	
32	380	8	
33	388	4	
34	392	4	
35	396	4	
36	400	4	
37	404	1	
38	405	4	
39	409	4	
40	413	4	
41	417	4	
42	421	4	
43	425	4	
44	429	20	
45	449	30	

```

46  479  1
47  480  1
48  481  79
49  560  79
50  639  79
51  718  79
52  797  8
53  805  1
54  806  1
55  807  20
56  827  4
57  831  4

```

```

shell> myisampack station.MYI
Compressing station.MYI: (1192 records)
- Calculating statistics

```

```

normal:      20  empty-space:      16  empty-zero:      12  empty-fill:    11
pre-space:   0  end-space:      12  table-lookups:   5  zero:          7
Original trees: 57  After join: 17
- Compressing file
87.14%

```

```

shell> ls -l station.*
-rw-rw-r--  1 monty  my          127874 Apr 17 19:00 station.MYD
-rw-rw-r--  1 monty  my           55296 Apr 17 19:04 station.MYI
-rw-rw-r--  1 monty  my           5767 Apr 17 19:00 station.frm

```

```

shell> myisamchk -dvv station

```

```

MyISAM file:      station
Isam-version:     2
Creation time:    1996-03-13 10:08:58
Recover time:     1997-04-17 19:04:26
Data records:     1192  Deleted blocks:      0
Datafile: Parts:  1192  Deleted data:        0
Datafilepointer (bytes): 3  Keyfile pointer (bytes): 1
Max datafile length: 16777215  Max keyfile length: 131071
Recordlength:    834
Record format:    Compressed

```

```

table description:

```

Key	Start	Len	Index	Type	Root	Blocksize	Rec/key
1	2	4	unique	unsigned long	10240	1024	1
2	32	30	multip.	text	54272	1024	1

Field	Start	Length	Type	Huff tree	Bits
1	1	1	constant	1	0
2	2	4	zerofill(1)	2	9

3	6	4	no zeros, zerofill(1)	2	9
4	10	1		3	9
5	11	20	table-lookup	4	0
6	31	1		3	9
7	32	30	no endspace, not_always	5	9
8	62	35	no endspace, not_always, no empty	6	9
9	97	35	no empty	7	9
10	132	35	no endspace, not_always, no empty	6	9
11	167	4	zerofill(1)	2	9
12	171	16	no endspace, not_always, no empty	5	9
13	187	35	no endspace, not_always, no empty	6	9
14	222	4	zerofill(1)	2	9
15	226	16	no endspace, not_always, no empty	5	9
16	242	20	no endspace, not_always	8	9
17	262	20	no endspace, no empty	8	9
18	282	20	no endspace, no empty	5	9
19	302	30	no endspace, no empty	6	9
20	332	4	always zero	2	9
21	336	4	always zero	2	9
22	340	1		3	9
23	341	8	table-lookup	9	0
24	349	8	table-lookup	10	0
25	357	8	always zero	2	9
26	365	2		2	9
27	367	2	no zeros, zerofill(1)	2	9
28	369	4	no zeros, zerofill(1)	2	9
29	373	4	table-lookup	11	0
30	377	1		3	9
31	378	2	no zeros, zerofill(1)	2	9
32	380	8	no zeros	2	9
33	388	4	always zero	2	9
34	392	4	table-lookup	12	0
35	396	4	no zeros, zerofill(1)	13	9
36	400	4	no zeros, zerofill(1)	2	9
37	404	1		2	9
38	405	4	no zeros	2	9
39	409	4	always zero	2	9
40	413	4	no zeros	2	9
41	417	4	always zero	2	9
42	421	4	no zeros	2	9
43	425	4	always zero	2	9
44	429	20	no empty	3	9
45	449	30	no empty	3	9
46	479	1		14	4
47	480	1		14	4
48	481	79	no endspace, no empty	15	9
49	560	79	no empty	2	9
50	639	79	no empty	2	9
51	718	79	no endspace	16	9

52	797	8	no empty	2	9
53	805	1		17	1
54	806	1		3	9
55	807	20	no empty	3	9
56	827	4	no zeros, zerofill(2)	2	9
57	831	4	no zeros, zerofill(1)	2	9

A informação exibida pelo `myisampack` é descrita abaixo:

normal O número de colunas para qual nenhum empacotamento extra é utilizado.

empty-space

O número de colunas contendo valores que são somente espaços; estes ocuparão apenas 1 bit.

empty-zero

O número de colunas contendo valores que são somente 0's binários; ocuparão 1 bit.

empty-fill

O número de colunas inteiras que não ocupam a faixa completa de bytes de seu tipo; estes são alteradas para um tipo menor (por exemplo, uma coluna `INTEGER` pode ser alterada para `MEDIUMINT`).

pre-space

O número de colunas decimais que são armazenadas com espaços a esquerda. Neste caso, cada valor irá conter uma contagem para o número de espaços.

end-space

O número de colunas que tem muitos espaços extras. Neste caso, cada valor conterá uma contagem para o número de espaços sobrando.

table-lookup

A coluna tem somente um pequeno número de valores diferentes, que são convertidos para um `ENUM` antes da compressão Huffman.

zero

O número de colunas em que todos os valores estão zerados.

Original trees

O número inicial de árvores Huffman.

After join

O número de árvores Huffman distintas que sobram depois de unir árvores para poupar espaço de cabeçalho.

Depois que uma tabela foi compactada, `myisamchk -dvv` exibe informações adicionais sobre cada campo:

Type O tipo de campo deve conter as seguintes descrições:

constant Todas linhas tem o mesmo valor.

no endspace

Não armazena espaços no fim.

no endspace, not_always

Não armazena espaços no fim e não faz compactação de espaços finais para todos os valores.

no endspace, no empty

Não armazena espaços no fim. Não armazena valores vazios.

table-lookup

A coluna foi convertida para um ENUM.

zerofill(n)

Os n bytes mais significativos no valor são sempre 0 e não são armazenados.

no zeros Não armazena zeros.**always zero**

Valores zero são armazenados em 1 bit.

Huff tree A árvore Huffman associada com o campo.

Bits O número de bits usado na árvore Huffman.

Depois de ter executado `pack_isam/myisampack` você deve executar o `isamchk/myisamchk` para recriar o índice. Neste momento você pode também ordenar os blocos de índices para criar estatísticas necessárias para o otimizador do MySQL trabalhar de maneira mais eficiente.

```
myisamchk -rq --analyze --sort-index nome_tabela.MYI
isamchk -rq --analyze --sort-index nome_tabela.ISM
```

Depois de instalar a tabela compactada no diretório de banco de dados MySQL você deve fazer `mysqladmin flush-tables` para forçar o `mysqld` a iniciar usando a nova tabela.

Se você desejar descompactar uma tabela compactada, você pode fazer isto com a opção `--unpack` para o `isamchk` ou `myisamchk`.

4.7.5 `mysqld-max`, Um servidor `mysqld` estendido

`mysqld-max` é o servidor MySQL (`mysqld`) configurado com as seguintes opções de configuração:

Opção	Comentário
<code>--with-server-suffix=-max</code>	Adiciona um sufixo à string de versão <code>mysqld</code>
<code>--with-bdb</code>	Suporte para tabelas Berkeley DB (BDB)
<code>--with-innodb</code>	Suporte a tabelas InnoDB.
<code>CFLAGS=-DUSE_SYMDIR</code>	Suporte a links simbólicos para Windows.

Você pode encontrar os binários do MySQL-max em <http://www.mysql.com/downloads/mysql-max-3.23.html>.

A distribuição binária Windows MySQL 3.23 inclui tanto o binário `mysqld.exe` padrão e o binário `mysqld-max.exe`. <http://www.mysql.com/downloads/mysql-3.23.html>. See [\[Instalação no Windows\]](#), page [\[Instalação no Windows\]](#).

Note que como o Berkeley DB e InnoDB não estão disponíveis para todas plataformas, alguns dos binários Max podem não ter suporte para os mesmos. Você pode conferir quais tipos de tabelas são suportadas executando a seguinte consulta:

```
mysql> show variables like "have_%";
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_bdb      | YES   |
| have_innodb   | NO    |
| have_isam     | YES   |
| have_raid     | NO    |
| have_ssl      | NO    |
+-----+-----+
```

O significado dos valores são:

Vallor	Significado.
YES	A opção está ativa e é utilizada.
NO	O MySQL não está compilado com suporte a esta opção.
DISABLED	A opção xxx está desabilitada porque o <code>mysqld</code> foi iniciado com <code>--skip-xxxx</code> ou porque não foi iniciado com todas as opções necessárias para habilitar esta opção. Neste caso o arquivo <code>hostname.err</code> deve conter uma razão do pela qual a opção está desabilitada.

NOTA: Para conseguir criar tabelas InnoDB você **DEVE** editar suas opções de inicialização para incluir ao menos a opção `innodb_data_file_path`. See [\(undefined\)](#) [Iniciando InnoDB], page [\(undefined\)](#).

Para obter melhor performance para tabelas BDB, você deve adicionar algumas opções de configuração para elas também. See [\(undefined\)](#) [Iniciando BDB], page [\(undefined\)](#).

`safe_mysqld` tentará iniciar automaticamente qualquer binário `mysqld` com o prefixo `-max`. Isto faz com que seja fácil testar um outro binário `mysqld` em uma instalação existente. Apenas execute o `configure` com as opções desejadas e, então, instale o novo binário `mysqld` como `mysqld-max` no mesmo diretório onde seu antigo binário `mysqld` está. See [\(undefined\)](#) [`safe_mysqld`], page [\(undefined\)](#).

O RPM `mysqld-max` utiliza o recurso `safe_mysqld` já mencionado. Ele apenas instala o executável `mysqld-max` e o `safe_mysqld` usará automaticamente este executável quando o `safe_mysqld` for reiniciado.

A tabela a seguir mostra quais tipos de tabelas nossos binários padrões **MySQL-Max** incluem:

Sistema	BDB	InnoDB
AIX 4.3	N	Y
HP-UX 11.0	N	Y
Linux-Alpha	N	Y
Linux-Intel	Y	Y
Linux-Ia64	N	Y
Solaris-intel	N	Y
Solaris-sparc	Y	Y
SCO OSR5	Y	Y
UnixWare	Y	Y
Windows/NT	Y	Y

4.8 Utilitários e Scripts do Lado do Cliente do MySQL

4.8.1 Visão Geral dos Utilitários e Scripts do Lado do Cliente

Todos clientes MySQL que comunicam com o servidor utilizando a biblioteca `mysqlclient` utilizam as seguintes variáveis de ambiente:

Nome	Descrição
<code>MYSQL_UNIX_PORT</code>	O socket padrão, utilizado para conexões ao <code>localhost</code>
<code>MYSQL_TCP_PORT</code>	A porta TCP/IP padrão
<code>MYSQL_PWD</code>	A senha padrão
<code>MYSQL_DEBUG</code>	Opções de depuração-ratreamento durante depuração
<code>TMPDIR</code>	O diretório onde tabelas e arquivos temporários são criados

A utilização de `MYSQL_PWD` é insegura. See [\(undefined\) \[Connecting\], page \(undefined\)](#).

O cliente `'mysql'` utiliza o arquivo nomeado na variável de ambiente `MYSQL_HISTFILE` para salvar o histórico da linha de comando. O valor padrão para o arquivo de histórico é `'$HOME/.mysql_history'`, onde `$HOME` é o valor da variável de ambiente `HOME`. See [\(undefined\) \[Environment variables\], page \(undefined\)](#).

Todos os programas MySQL podem receber várias opções diferentes. Entretanto, todo programa MySQL fornece a opção `--help` que você pode utilizar para obter uma descrição completa das diferentes opções do programa. Por exemplo, experimente `mysql --help`

Você pode sobrepor todas as opções padrões para programas cliente padrões com um arquivo de opções. [\(undefined\) \[Option files\], page \(undefined\)](#)

A lista abaixo descreve resumidamente os programas MySQL:

`myisamchk`

Utilitário para descrever, verificar, otimizar e reparar tabelas MySQL. Como o `myisamchk` tem várias funções, ele é descrito em seu próprio capítulo. See [\(undefined\) \[Administração de Banco de Dados MySQL\], page \(undefined\)](#).

`make_binary_distribution`

Gera uma distribuição binária do MySQL compilado. Ela pode ser enviado por FTP para `'/pub/mysql/Incoming'` em `support.mysql.com` para a conveniência de outros usuários MySQL.

`mysql2mysql`

Um script shell que converte programas `mSQL` para MySQL. Ele não lida com todos os casos, mas ele fornece um bom início para a conversão.

`mysqlaccess`

Um script que verifica os privilégios de acesso para uma combinação de nome de máquina, usuário e banco de dados.

`mysqladmin`

Utilitário para realizar operações administrativas, tais como criação ou remoção de bancos de dados, recarga das tabelas de permissões, descarga de tabelas em disco e reabertura dos arquivos log. `mysqladmin` também pode ser usado para exibir informações de versão, processos e estado do servidor. See [\(undefined\) \[mysqladmin\], page \(undefined\)](#).

- mysqlbug** O script para relatar erros no MySQL. Este script deve ser usado sempre que for for relatar algum bug para a lista MySQL.
- mysqld** O daemon SQL. Deve estar sempre em execução.
- mysqldump**
Descarrega um banco de dados MySQL em um arquivo como instruções SQL ou como arquivo texto separado por tabulação. Versão aprimorada do free-ware escrito originalmente por Igor Romanenko. See [\(undefined\)](#) [[mysqldump](#)], page [\(undefined\)](#).
- mysqlimport**
Importa arquivos texto em suas tabelas respectivas utilizando LOAD DATA INFILE. See [\(undefined\)](#) [[mysqlimport](#)], page [\(undefined\)](#).
- mysqlshow**
Exibe informações sobre bancos de dados, tabelas, colunas e índices.
- mysql_install_db**
Cria as tabelas de permissões do MySQL com os privilégios padrões. Isto é normalmente executado somente uma vez, quando estiver instalando o MySQL em um sistema.
- replace** Um programa utilitário que é usado pelo `mysql2mysql`, mas que também pode ser aplicável mais genericamente. `replace` altera conjuntos de caracteres. Utiliza uma máquina de estado finito para comparar strings maiores primeiro. Pode ser usada para trocar conjuntos de caracteres. Por exemplo, este comando troca a e b nos arquivos dados:

```
shell> replace a b b a -- arquivo1 arquivo2 ...
```

4.8.2 A Ferramenta de Linha de Comando

O `mysql` é uma shell SQL simples (com capacidades GNU `readline`). Ele suporta usos interativos e não interativos. Quando usado interativamente, os resultados das consultas são apresentadas no formato de tabela ASCII. Quando não usado interativamente (como um filtro por exemplo), o resultado é apresentado em um formato separado por tabulações. (O formato de saída pode ser alterado utilizando opções da linha de comando.) Você pode executar scripts desta forma:

```
shell> mysql database < script.sql > saida.tab
```

Se você tiver problemas devido a memória insuficiente no cliente, utilize a opção `--quick!` Isto força o `mysql` a utilizar `mysql_use_result()` no lugar de `mysql_store_result()` para recuperar o conjunto de resultados.

Utilizar o `mysql` é muito fácil. Inicie-o como mostrado a seguir: `mysql banco_de_dados` ou `mysql --user=nome_usuario --password=sua_senha banco_de_dados`. Digite uma instrução SQL, termine-a com `;`, `\g`, ou `\G` e pressione RETURN/ENTER.

O `mysql` Suporta as seguintes opções:

- `-, --help`
Exibe esta ajuda e sai.

- A, `--no-auto-rehash`
Sem reprocessamento automático. O 'rehash' deve ser usado se o usuário desejar que o cliente `mysql` complete as tabelas e campos. Esta opção é usada para acelerar a inicialização do cliente.
- B, `--batch`
Exibe resultados com o caractere de tabulação como o separador, cada registro em uma nova linha. Não utiliza o arquivo de histórico.
- `--character-sets-dir=...`
Diretório onde os conjuntos de caracteres estão localizados.
- C, `--compress`
Utiliza compactação no protocolo cliente/servidor.
- #, `--debug[=...]`
Log de Depuração. O padrão é 'd:t:o,/tmp/mysql.trace'.
- D, `--database=...`
Qual banco de dados usar. Isto geralmente é útil em um arquivo `my.cnf`.
- `--default-character-set=...`
Configura o conjunto de caracteres padrão.
- e, `--execute=...`
Executa o comando e sai. (Saída parecida com `--batch`)
- E, `--vertical`
Exibe a saída de uma consulta (linhas) verticalmente. Sem esta opção você também pode forçar esta saída terminando suas instruções com `\G`.
- f, `--force`
Continue mesmo se for obtido um erro SQL.
- g, `--no-named-commands`
Comandos nomeados serão desabilitados. Utilize somente a forma `*`, ou use comandos nomeados apenas no começo da linha terminada com um ponto-e-vírgula (;). Desde a versão 10.9, o cliente agora inicia com esta opção habilitada por padrão! Com a opção -g, entretando, comandos de formato longo continuarão funcionando na primeira linha.
- G, `--enable-named-commands`
Comandos nomeados são **habilitados**. Comandos de formato longo são aceitos assim como os comandos reduzidos `*`.
- i, `--ignore-space`
Ignore caractere de espaço depois de nomes de funções.
- h, `--host=...`
Conectar à máquina especificada.
- H, `--html`
Produza saída HTML.

- L, --skip-line-numbers**
Não escreva número da linha para os erros. Útil quando se deseja comparar arquivos com resultados que incluem mensagens de erro.
- no-pager**
Desabilita paginação e impressão na saída padrão. Veja também a ajuda interativa (\h).
- no-tee** Desabilita arquivo de saída. Veja também a ajuda interativa (\h).
- n, --unbuffered**
Descarrega e atualiza o buffer depois de cada pesquisa.
- N, --skip-column-names**
Não escrever nomes de colunas nos resultados.
- O, --set-variable var=option**
Fornecer um valor a uma variável. **--help** lista as variáveis.
- o, --one-database**
Atualiza somente o banco de dados padrão. Isto é útil para evitar atualização em outros bancos de dados no log de atualizações.
- pager [=...]**
Tipo de saída. O padrão é sua variável de ambiente **PAGER**. Paginadores válidos são: **less**, **more**, **cat** [>nome_arquivo], etc. Veja também a ajuda interativa (\h). Esta opção não funciona no modo batch. A opção **pager** funciona somente no UNIX.
- p[password], --password [=...]**
Senha a ser usada ao conectar ao servidor. Se uma senha não é fornecida na linha de comando, lhe será solicitada uma. Perceba que se você utilizar o formato curto **-p** você não pode ter um espaço entre a opção e a senha.
- P --port=...**
Número da porta TCP/IP para usar na conexão.
- q, --quick**
Não faz cache do resultado, imprime linha a linha. Isto pode deixar o servidor mais lento se a saída for suspensa. Não usa arquivo de histórico.
- r, --raw** Exibe valores de colunas sem conversão de escapes. Utilizado com **--batch**
- s, --silent**
Opção para ser mais silencioso.
- S --socket=...**
Arquivo socket para ser utilizado na conexão.
- t --table**
Saída no formato de tabela. Isto é padrão no modo não-batch.
- T, --debug-info**
Exibe alguma informação de depuração na saída.

- `--tee=...`
Anexa tudo no arquivo de saída. Veja também a ajuda interativa (`\h`). Não funciona no modo batch.
- `-u, --user=#`
Usuário para login diferente do usuário atual do sistema.
- `-U, --safe-updates[=#], --i-am-a-dummy[=#]`
Permite somente que UPDATE e DELETE utilizem chaves. Veja abaixo para maiores informações sobre esta opção. Você pode zerar esta opção se possui-la no arquivo `my.cnf` utilizando `--safe-updates=0`.
- `-v, --verbose`
Modo verbose (`-v -v -v` fornece o formato de saída da tabela).
- `-V, --version`
Gera saída com informação de versão e sai.
- `-w, --wait`
Espera e repete em vez de sair se a conexão estiver inacessível.

Você também pode configurar as seguintes variáveis com `-O` ou `--set-variable`:

Nome variável	Padrão	Descrição
<code>connect_timeout</code>	0	Número de segundos antes de esgotar o tempo da conexão
<code>max_allowed_packet</code>	16777216	Tamanho máximo do pacote para enviar/receber do servidor
<code>net_buffer_length</code>	16384	Tamanho do buffer para comunicação TCP/IP e socket
<code>select_limit</code>	1000	Limite automático para SELECT quando utilizar <code>--i-am-a-dummy</code>
<code>max_join_size</code>	1000000	Limite automático para registros em uma join quando utilizar <code>--i-am-a-dummy</code> .

Se você digitar 'help' na linha de comando, `mysql` irá exibir os comandos que ele suporta:

```
mysql> help
```

```
MySQL commands:
```

```
help      (\h)   Display this text.
?         (\h)   Synonym for 'help'.
clear     (\c)   Clear command.
connect   (\r)   Reconnect to the server. Optional arguments are db and host.
edit      (\e)   Edit command with $EDITOR.
ego       (\G)   Send command to mysql server, display result vertically.
exit      (\q)   Exit mysql. Same as quit.
go        (\g)   Send command to mysql server.
nopager   (\n)   Disable pager, print to stdout.
notee     (\t)   Don't write into outfile.
pager     (\P)   Set PAGER [to_pager]. Print the query results via PAGER.
print     (\p)   Print current command.
quit      (\q)   Quit mysql.
rehash    (\#)   Rebuild completion hash.
```

```

source (\.)      Execute a SQL script file. Takes a file name as an argument.
status (\s)     Get status information from the server.
tee (\T)       Set outfile [to_outfile]. Append everything into given outfile.
use (\u)       Use another database. Takes database name as argument.

```

Das opções acima, o pager é o único que só funciona no UNIX.

O comando `status` lhe fornece algumas informações sobre a conexão e o servidor que está utilizando. Se você estiver executando no modo `--safe-updates`, `status` irá também imprimir os valores para as variáveis `mysql` que afetam suas consultas.

Uma opção útil para iniciantes (introduzido no MySQL versão 3.23.11) é o `--safe-updates` (ou `--i-am-a-dummy` para usuários que alguma vez executaram um `DELETE FROM nome_tabela` mas esqueceram da cláusula `WHERE`). Quando utilizar esta opção, o `mysql` envia o seguinte comando ao servidor MySQL quando abrir a conexão.

```

SET SQL_SAFE_UPDATES=1,SQL_SELECT_LIMIT=#select_limit#,
    SQL_MAX_JOIN_SIZE=#max_join_size#

```

onde `#select_limit#` e `#max_join_size#` são variáveis que podem ser configuradas da linha de comando `mysql`. See [\[SET OPTION\]](#), page [\[undefined\]](#).

O efeito da opção acima é:

- Você não tem permissão de utilizar uma instrução `UPDATE` ou `DELETE` se você não possuir uma chave na parte `WHERE`. Pode-se, entretanto, forçar um `UPDATE/DELETE` utilizando `LIMIT`:

```

UPDATE nome_tabela SET campo_ao_chave=# WHERE campo_ao_chave=# LIMIT 1;

```

- Todos resultados maiores são limitados automaticamente a `#select_limit#` linhas.
- `SELECT`'s que provavelmente precisarão examinar mais que `#max_join_size` combinações de linhas serão abortadas.

Algumas dicas úteis sobre o cliente `mysql`:

Alguns dados são muito mais legíveis quando exibido verticalmente, em vez da saída do tipo caixa horizontal comum. Por exemplo: Textos longos, que incluem várias linhas, são muito mais fáceis de serem lidos com saída vertical.

```

mysql> select * from mails where length(txt) < 300 limit 300,1\G
***** 1. row *****
msg_nro: 3068
date: 2000-03-01 23:29:50
time_zone: +0200
mail_from: Monty
reply: monty@no.spam.com
mail_to: "Thimble Smith" <tim@no.spam.com>
subj: UTF-8
txt: >>>> "Thimble" == Thimble Smith writes:

```

```

Thimble> Hi. I think this is a good idea. Is anyone familiar with UTF-8

```

```

Thimble> or Unicode? Otherwise I'll put this on my TODO list and see what
Thimble> happens.

```

```

Yes, please do that.

```

```

Regards,
Monty
  file: inbox-jani-1
  hash: 190402944
1 row in set (0.09 sec)

```

- Para o log, você pode utilizar a opção `tee`. O `tee` pode ser iniciado com a opção `--tee=...`, ou pela linha de comando de maneira interativa com o comando `tee`. Todos os dados exibidos na tela serão anexados no arquivo fornecido. Isto também pode ser muito útil para propósitos de depuração. O `tee` pode ser desabilitado da linha de comando com o comando `notee`. Executando `tee` novamente o log é reiniciado. Sem um parâmetro o arquivo anterior será usado. Perceba que `tee` irá atualizar os resultados dentro do arquivo depois de cada comando, pouco antes da linha de comando reaparecer esperando pelo próximo comando.
- Navegar ou pesquisar os resultados no modo interativo em algum programa do UNIX como o `less`, `more` ou outro similar, é agora possível com a opção `--pager [=...]`. Sem argumento, o cliente `mysql` irá procurar pela variável de ambiente `PAGER` e configurar `pager` para este valor. `pager` pode ser iniciado a partir da linha de comando interativa com o comando `pager` e desabilitado com o comando `nopager`. O comando recebe um argumento opcional e o `pager` será configurado com ele. O comando `pager` pode ser chamado com um argumento, mas isto requer que a opção `--pager` seja usada, ou o `pager` será usado com a saída padrão. `pager` funciona somente no UNIX, uma vez que é utilizado a função `popen()`, que não existe no Windows. No Windows a opção `tee` pode ser utilizada, entretanto ela pode não ser cômoda como `pager` pode ser em algumas situações.
- Poucas dicas sobre `pager`: Você pode usá-lo para gravar em um arquivo:

```
mysql> pager cat > /tmp/log.txt
```

e os resultados irão somente para um arquivo. Você também pode passar qualquer opções para os programas que você deseja utilizar com `pager`:

```
mysql> pager less -n -i -S
```

Note a opção `'-S'` exibida acima. Você pode achá-la muito útil quando navegar pelos resultados; experimente com a opção com saída a horizontal (finalize os comandos com `'\g'`, ou `;`) e com saída vertical (final dos comandos com `'\G'`). Algumas vezes um resultado com um conjunto muito largo é difícil ser lido na tela, com a opção `-S` para `less`, você pode navegar nos resultados com o `less` interativo da esquerda para a direita, evitando que linhas maiores que sua tela continuem na próxima linha. Isto pode tornar o conjunto do resultado muito mais legível. você pode alterar o modo entre ligado e desligado com o `less` interativo com `'-S'`. Veja o `'h'`(help) para mais ajuda sobre o `less`.

- Por último (a menos que você já entendeu este assunto com os exemplos acima ;) você pode combinar maneiras muito complexas para lidar com os resultados, por exemplo, o seguinte enviaria os resultados para dois arquivos em dois diferentes diretórios, em dois discos diferentes montados em `/dr1` e `/dr2`, e ainda exibe o resultado na tela via `less`:

```
mysql> pager cat | tee /dr1/tmp/res.txt | tee /dr2/tmp/res2.txt | less -
n -i -S
```

- Você também pode combinar as duas funções acima; tenha o `tee` habilitado, o `pager` configurado para `'less'` e você estará apto a navegar nos resultados no `less` do `unix` e ainda ter tudo anexado em um arquivo ao mesmo tempo. A diferença entre `UNIX tee` usado com o `pager` e o `tee` embutido no cliente `mysql` é que o `tee` embutido funciona mesmo se você não tiver o comando `UNIX tee` disponível. O `tee` embutido também loga tudo que é exibido na tela, e o `UNIX tee` usado com `pager` não loga completamente. Por último o `tee` interativo é mais cômodo para trocar entre os modos `on` e `off`, quando você desejar logar alguma coisa em um arquivo, mas deseja estar apto para desligar o recurso quando necessário.

4.8.3 `mysqladmin`, Administrando um Servidor MySQL

Um utilitário para realizar operações administrativas. A sintaxe é:

```
shell> mysqladmin [OPÇÕES] comando [opção_do_comando] comando...
```

Você pode obter uma lista das opções que sua versão do `mysqladmin` suporta executando `mysqladmin --help`.

O `mysqladmin` atual suporta os seguintes comandos:

`create databasename`

Cria um novo banco de dados.

`drop databasename`

Apaga um banco de dados e todas suas tabelas.

`extended-status`

Fornecer uma mensagem estendida sobre o estado do servidor.

`flush-hosts`

Atualiza todos os nomes de máquinas que estiverem no cache.

`flush-logs`

Atualiza todos os logs.

`flush-tables`

Atualiza todas as tabelas.

`flush-privileges`

Recarrega tabelas de permissões (mesmo que `reload`).

`kill id,id,...`

Mata threads do MySQL.

`password` Configura uma nova senha. Altera a antiga senha para nova senha.

`ping` Checa se o `mysqld` está ativo.

`processlist`

Exibe lista de threads ativas no servidor.

`reload` Recarrega tabelas de permissão.

`refresh` Atualiza todas as tabelas e fecha e abre arquivos de log.

`shutdown` Desliga o servidor.

slave-start

Inicia thread de replicação no slave.

slave-stop

Termina a thread de replicação no slave.

status

Fornecer uma mensagem curta sobre o estado do servidor.

variables

Exibe variáveis disponíveis.

version

Obtêm informação de versão do servidor.

Todos comandos podem ser reduzidos para seu prefixo único. Por exemplo:

```
shell> mysqladmin proc stat
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
| Id | User | Host      | db | Command      | Time | State | Info |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 6  | monty | localhost |    | Processlist  | 0    |      |      |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
Uptime: 10077  Threads: 1  Questions: 9  Slow queries: 0  Opens: 6  Flush tables: 1
```

O resultado do comando `mysqladmin status` possui as seguintes colunas:

Uptime	Número de segundos que o servidor MySQL está funcionando.
Threads	Número de threads ativas (clientes).
Questions	Número de solicitações dos clientes desde que o <code>mysqld</code> foi iniciado.
Slow queries	Consultas que demoram mais que <code>long_query_time</code> segundos. See [Log de consultas lentas] , page [Log de consultas lentas] .
Opens	Quantas tabelas foram abertas pelo <code>mysqld</code> .
Flush tables	Número de comandos <code>flush...</code> , <code>refresh</code> e <code>reload</code> .
Open tables	Número de tabelas abertas atualmente.
Memory in use	Memória alocada diretamente pelo código do <code>mysqld</code> (disponível somente quando o MySQL é compilado com <code>--with-debug=full</code>).
Max memory used	Memória máxima alocada diretamente pelo código do <code>mysqld</code> (disponível somente quando o MySQL é compilado com <code>--with-debug=full</code>).

Se você executa um `mysqladmin shutdown` em um socket (em outras palavras, em um computador onde o `mysqld` está executando), `mysqladmin` irá esperar até que o `arquivo-pid` do MySQL seja removido para garantir que o servidor `mysqld` parou corretamente.

4.8.4 Usando `mysqlcheck` para Manutenção de Tabelas e Recuperação em Caso de Falhas

Desde o MySQL versão 3.23.38 você estará apto a usar a nova ferramenta de reparos e verificação de tabelas MyISAM. A diferença para o `myisamchk` é que o `mysqlcheck` deve ser usado quando o servidor `mysqld` estiver em funcionamento, enquanto o `myisamchk` deve ser usado quando ele não estiver. O benefício é que você não precisará mais desligar o servidor `mysqld` para verificar ou reparar suas tabelas.

O `mysqlcheck` utiliza os comandos do servidor MySQL `CHECK`, `REPAIR`, `ANALYZE` e `OPTIMIZE` de um modo conveniente para o usuário.

Existem três modos alternativos de chamar o `mysqlcheck`:

```
shell> mysqlcheck [OPÇÕES] database [tabelas]
shell> mysqlcheck [OPÇÕES] --databases DB1 [DB2 DB3...]
shell> mysqlcheck [OPÇÕES] --all-databases
```

Pode ser usado de uma maneira muito similar ao `mysqldump` quando o assunto for quais bancos de dados e tabelas devem ser escolhidas.

O `mysqlcheck` tem um recurso especial comparado aos outros clientes; o comportamento padrão, verificando as tabelas (`-c`), pode ser alterado renomeando o binário. Se você deseja ter uma ferramenta que repare as tabelas como o procedimento padrão, você deve copiar o `mysqlcheck` para o disco com um outro nome, `mysqlrepair`, ou crie um link simbólico com o nome `mysqlrepair`. Se você chamar `mysqlrepair` agora, ele irá reparar as tabelas como seu procedimento padrão.

Os nomes que podem ser utilizados para alterar o comportamento padrão do `mysqlcheck` são:

```
mysqlrepair:   A opção padrão será -r
mysqlanalyze:  A opção padrão será -a
mysqloptimize: A opção padrão será -o
```

As opções disponíveis para o `mysqlcheck` estão listadas aqui, por favor verifique o que a sua versão suporta com o `mysqlcheck --help`.

`-A, --all-databases`

Verifica todos os bancos de dados. Isto é o mesmo que `--databases` com todos os bancos de dados selecionados.

`-1, --all-in-1`

Em vez de fazer uma consulta para cada tabela, execute todas as consultas separadamente para cada banco de dados. Nomes de tabelas estarão em uma lista separada por vírgula.

`-a, --analyze`

Analise as tabelas fornecidas.

`--auto-repair`

Se uma tabela checada está corrompida, ela é corrigida automaticamente. O reparo será feito depois que todas as tabelas tiverem sido checadas e forem detectadas tabelas corrompidas.

`-#, --debug=...`

Log de saída de depuração. Normalmente é `'d:t:o,filename'`

`--character-sets-dir=...`

Diretório onde estão os conjuntos de caracteres.

`-c, --check`

Verifica erros em tabelas

`-C, --check-only-changed`

Verifica somente tabelas que foram alteradas desde a última conferência ou que não foram fechada corretamente.

- compress**
Utilize compressão no protocolo server/cliente.
- ?, --help**
Exibe esta mensagem de ajuda e sai.
- B, --databases**
Para verificar diversos bancos de dados. Perceba a diferença no uso; Neste caso nenhuma tabela será fornecida. Todos os argumentos são tratados como nomes de bancos de dados.
- default-character-set=...**
Configura o conjunto de caracteres padrão.
- F, --fast**
Verifica somente as tabelas que não foram fechadas corretamente
- f, --force**
Continue mesmo se nós obtermos um erro de sql.
- e, --extended**
Se você estiver utilizando esta opção com CHECK TABLE, irá garantir que a tabela está 100 por cento consistente, mas leva bastante tempo.
Se você utilizar esta opção com REPAIR TABLE, ele irá executar um comando de reparos na tabela, que não só irá demorar muito tempo para executar, mas também pode produzir muitas linhas de lixo.
- h, --host=...**
Conecta à máquina.
- m, --medium-check**
Mais rápido que verificação estendida, mas encontra somente 99.99 de todos os erros. Deve resolver a maioria dos casos.
- o, --optimize**
Otimizador de tabelas
- p, --password[=...]**
Senha para usar ao conectar ao servidor. Se a senha não for fornecida será solicitada no terminal.
- P, --port=...**
Número de porta para usar para conexão.
- q, --quick**
Se esta opção for utilizada com CHECK TABLE, evita a busca de registros verificando links errados. Esta é a conferência mais rápida.
Se você estiver utilizando esta opção com REPAIR TABLE, ela tentará reparar somente a árvore de índices. Este é o método de reparo mais rápido para uma tabela.
- r, --repair**
Pode corrigir quase tudo exceto chaves únicas que não são únicas.

- `-s, --silent`
Exibe somente mensagens de erro.
- `-S, --socket=...`
Arquivo socket para usar na conexão.
- `--tables` Sobrepõe a opção `--databases (-B)`.
- `-u, --user=#`
Usuário para o login, se não for o usuário atual.
- `-v, --verbose`
Exibe informação sobre os vários estágios.
- `-V, --version`
Exibe informação sobre a versão e sai.

4.8.5 mysqldump, Descarregando estrutura de tabelas e dados

Utilitário para descarregar um banco de dados ou uma coleção de bancos de dados para backup ou transferencia para outro servidor SQL (Não necessariamente um servidor MySQL). A descarga irá conter instruções SQL para cria a tabela e/ou popular a tabela.

Se a idéia é backup do servidor, deve ser considerada a utilização do `mysqlhotcopy`. See [\[mysqlhotcopy\]](#), page [\[mysqlhotcopy\]](#).

```
shell> mysqldump [OPÇÕES] banco_de_dados [tabelas]
OR      mysqldump [OPÇÕES] --databases [OPÇÕES] BD1 [BD2 BD3...]
OR      mysqldump [OPÇÕES] --all-databases [OPÇÕES]
```

Se você não fornecer nenhuma tabela ou utilizar o `--databases` ou `--all-databases`, todo(s) o(s) banco(s) de dados será(ão) descarregado(s).

Você pode obter uma lista das opções que sua versão do `mysqldump` suporta executando `mysqldump --help`.

Perceba que se você executar o `mysqldump` sem a opção `--quick` ou `--opt`, o `mysqldump` irá carregar todo o conjunto do resultado na memória antes de descarregar o resultado. Isto provavelmente será um problema se você está descarregando um banco de dados grande.

Note que se você estiver utilizando uma cópia nova do programa `mysqldump` e se você for fazer uma descarga que será lida em um servidor MySQL muito antigo, você não deve utilizar as opções `--opt` ou `-e`.

`mysqldump` suporta as seguintes opções:

- `--add-locks`
Adicione `LOCK TABLES` antes de `UNLOCK TABLE` depois de cada descarga de tabelas. (Para obter inserções mais rápidas no MySQL.)
- `--add-drop-table`
Adicione um `drop table` antes de cada instrução `create`.
- `-A, --all-databases`
Descarrega todos os bancos de dados. Isto irá ser o mesmo que `--databases` com todos os bancos de dados selecionados.
- `-a, --all` Inclui todas as opções do `create` específicas do MySQL.

- allow-keywords**
Permite criação de nomes de colunas que são palavras chaves. Isto funciona utilizando o nome da tabela como prefixo em cada nome de coluna.
- c, --complete-insert**
Utilize instruções de insert completas (com nomes de colunas).
- C, --compress**
Compacta todas as informações entre o cliente e o servidor se ambos suportarem a compactação.
- B, --databases**
Para descarregar diversos bancos de dados. Perceba a diferença no uso. Neste caso nenhuma tabela é fornecida. Todos argumentos são estimados como nomes de bancos de dados. `USE nome_bd;` será incluído na saída antes de cada banco de dados novo.
- delayed**
Insere registros com o comando `INSERT DELAYED`.
- e, --extended-insert**
Utiliza a nova sintaxe multilinhas `INSERT`. (Fornece instruções de inserção mais compactas e mais rápidas.)
- #, --debug[=option_string]**
Rastreia a utilização do programa (para depuração).
- help** Exibe uma mensagem de ajuda e sai.
- fields-terminated-by=...**
- fields-enclosed-by=...**
- fields-optionally-enclosed-by=...**
- fields-escaped-by=...**
- lines-terminated-by=...**
Estas opções são usadas com a opção `-T` e tem o mesmo significado que as cláusulas correspondentes em `LOAD DATA INFILE` See [\(undefined\)](#) [`LOAD DATA`], page [\(undefined\)](#).
- F, --flush-logs**
Atualiza o arquivo de log no servidor MySQL antes de iniciar a descarga.
- f, --force,**
Continue mesmo se obter um erro de SQL durante uma descarga de tabela.
- h, --host=..**
Descarrega dados do servidor MySQL na máquina especificada. A máquina padrão é `localhost`.
- l, --lock-tables.**
Bloqueia todas as tabelas antes de iniciar a descarga. As tabelas são bloqueadas com `READ LOCAL` para permitir inserções concorrentes no caso de tabelas `MyISAM`.
- n, --no-create-db**
'`CREATE DATABASE /*!32312 IF NOT EXISTS*/ nome_bd;`' não será colocado na saída. A linha acima será adicionada se a opção `--databases` ou `--all-databases` for fornecida.

- t, --no-create-info**
Não grava informações de criação de tabelas (A instrução CREATE TABLE.)
- d, --no-data**
Não grava nenhuma informação de registros para a tabela. Isto é muito útil se você desejar apenas um dump da estrutura da tabela!
- opt** O mesmo que **--quick --add-drop-table --add-locks --extended-insert --lock-tables**. Fornece a descarga mais rápida para leitura em um servidor MySQL.
- pyour_pass, --password[=sua_senha]**
A senha para usar quando conectando ao servidor. Se não for especificado a parte '=sua_senha', o `mysqldump` irá perguntar por uma senha.
- P port_num, --port=porta_num**
O número da porta TCP/IP usado para conectar a uma máquina. (Isto é usado para conexões a máquinas diferentes de `localhost`, na qual sockets Unix são utilizados.)
- q, --quick**
Não utiliza buffers para as consultas, descarrega diretamente para saída padrão. Utilize `mysql_use_result()` para fazer isto.
- r, --result-file=...**
Direcione a saída para um determinado arquivo. Esta opção deve ser usada no MSDOS porque previne a conversão de nova linha '\n' para '\n\r' (nova linha + retorno de carro).
- S /path/to/socket, --socket=/caminho/para/arquivo_socket**
O arquivo socket que será utilizado quando conectar à `localhost` (que é a máquina padrão).
- tables** Sobrepõe a opção **--databases (-B)**.
- T, --tab=path-to-some-directory**
Cria um arquivo `nome_tabela.sql`, que contém os comandos SQL CREATE e um arquivo `nome_tabela.txt`, que contém os dados, para cada tabela dada. **NOTA:** Isto só funcionará se `mysqldump` for executado na mesma máquina do daemon `mysqld`. O formato do arquivo `.txt` é feito de acordo com as opções **--fields-xxx** e **--lines--xxx**.
- u user_name, --user=user_name**
O nome do usuário do MySQL para usar ao conectar ao servidor. O valor padrão é seu nome de usuário no Unix.
- O var=option, --set-variable var=option**
Configura o valor de uma variável. As variáveis possíveis são listadas abaixo.
- v, --verbose**
Modo verbose. Exibe mais informações sobre o que o programa realiza.
- V, --version**
Exibe informações de versão e sai.

`-w, --where='where-condition'`

Descarrega somente registros selecionados. Perceba que as aspas são obrigatórias:

```
--where=user='jimf' "-wuserid>1" "-wuserid<1"
```

`-O net_buffer_length=#, where # < 16M`

Quando estiver criando instruções de inserções em múltiplas linhas (com a opção `--extended-insert` ou `--opt`), `mysqldump` irá criar linhas até o tamanho de `net_buffer_length`. Se você aumentar esta variável, você também deve se assegurar que a variável `max_allowed_packet` no servidor MySQL é maior que a `net_buffer_length`.

O uso mais comum do `mysqldump` é provavelmente para fazer backups de bancos de dados inteiros. See [\[Backup\]](#), page [\[Backup\]](#).

```
mysqldump --opt banco_dados > arquivo-backup.sql
```

Você pode ler de volta no MySQL com:

```
mysql banco_dados < arquivo-backup.sql
```

ou

```
mysql -e "source /patch-to-backup/backup-file.sql" banco_de_dados
```

Entretanto, é muito útil também popular outro servidor MySQL com informações de um banco de dados:

```
mysqldump --opt banco_dados | mysql ---host=máquina-remota -C banco_dados
```

É possível descarregar vários bancos de dados com um comando:

```
mysqldump --databases banco_dados1 [banco_dados2 banco_dados3...] > meus_bancosdedados.sql
```

Se desejar descarregar todos os bancos de dados, pode-se utilizar:

```
mysqldump --all-databases > todos_bancos_dados.sql
```

4.8.6 `mysqlhotcopy`, Copiando Bancos de Dados e Tabelas MySQL

O `mysqlhotcopy` é um script perl que utiliza `LOCK TABLES`, `FLUSH TABLES` e `cp` ou `scp` para fazer um backup rápido de um banco de dados. É a maneira mais rápida para fazer um backup do banco de dados e de algumas tabelas mas ele só pode ser executado na mesma máquina onde os diretórios dos bancos de dados estão.

```
mysqlhotcopy nome_bd [/caminho/para/novo_diretório]
```

```
mysqlhotcopy nome_bd_2 ... nome_bd_2 /caminho/para/novo_diretório
```

```
mysqlhotcopy nome_bd./regex/
```

`mysqlhotcopy` suporta as seguintes opções:

`-?, --help`

Exibe uma tela de ajuda e sai

`-u, --user=#`

Usuário para fazer login no banco de dados

- `-p, --password=#`
Senha para usar ao conectar ao servidor
- `-P, --port=#`
Porta para usar ao conectar ao servidor local
- `-S, --socket=#`
Qual socket usar ao conectando a um servidor local
- `--allowold`
Não aborta se o alvo já existir (renomeie-o para `_old`)
- `--keepold`
Não apaga alvos anteriores (agora renomeados) quando pronto
- `--noindices`
Não inclui arquivos de índices na cópia para deixar o backup menor e mais rápido. Os índices podem ser reconstruídos mais tarde com `myisamchk -rq..`
- `--method=#`
Método para copiar (`cp` ou `scp`).
- `-q, --quiet`
Seja silencioso exceto em erros
- `--debug` Habilita depuração
- `-n, --dryrun`
Relata ações sem realizá-las
- `--regexp=#`
Copia todos bancos de dados com nomes que coincidem com a expressão regular
- `--suffix=#`
Sufixo para nomes de bancos de dados copiados
- `--checkpoint=#`
Insere entrada de ponto de controle um uma `bd.tabela` especificada
- `--flushlog`
Atualiza logs uma vez que todas as tabelas estiverem bloqueadas.
- `--tmpdir=#`
Diretório Temporário (em vez de `/tmp`).

Você pode utilizar `perldoc mysqlhotcopy` para obter uma documentação mais completa de `mysqlhotcopy`.

`mysqlhotcopy` lê os grupos `[client]` e `[mysqlhotcopy]` dos arquivos de opções.

Para poder executar `mysqlhotcopy` é necessário acesso de escrita ao diretório de backup, privilégio `SELECT` nas tabelas que desejar copiar e o privilégio `Reload` no MySQL (para poder executar `FLUSH TABLES`).

4.8.7 `mysqlimport`, Importando Dados de Arquivos Texto

`mysqlimport` fornece uma interface de linha de comando para a instrução SQL `LOAD DATA INFILE`. A maioria das opções aceitas correspondem diretamente às opções de `LOAD DATA INFILE`. See [\(undefined\) \[LOAD DATA\]](#), page [\(undefined\)](#).

`mysqlimport` é chamado desta maneira:

```
shell> mysqlimport [opções] banco_de_dados arquivo_texto1 [arquivo_texto2....]
```

Para cada arquivo texto passado na linha de comando, `mysqlimport` remove qualquer extensão do nome do arquivo e utiliza o resultado para determinar para qual tabela os dados do arquivo serão importados. Por exemplo, arquivos chamados `'patient.txt'`, `'patient.text'` e `'patient'` serão importados para uma tabela chamada `patient`.

`mysqlimport` suporta as seguintes opções:

`-c, --columns=...`

Esta opção recebe uma lista de nomes de campos separados por vírgula como um argumento. A lista de campos é utilizada para criar um comando `LOAD DATA INFILE` adequado que é então passado ao MySQL. See [\(undefined\) \[LOAD DATA\]](#), page [\(undefined\)](#).

`-C, --compress`

Compacta todas as informações entre o cliente e o servidor se ambos suportarem compressão.

`-#, --debug[=option_string]`

Rastreia o programa (para depuração).

`-d, --delete`

Esvazie a tabela antes de importar o arquivo texto.

`--fields-terminated-by=...`

`--fields-enclosed-by=...`

`--fields-optionally-enclosed-by=...`

`--fields-escaped-by=...`

`--lines-terminated-by=...`

Estas opções tem o mesmo significado que as cláusulas correspondentes para `LOAD DATA INFILE`. See [\(undefined\) \[LOAD DATA\]](#), page [\(undefined\)](#).

`-f, --force`

Ignorar erros. Por exemplo, se uma tabela para um arquivo texto não existir, continue processando quaisquer arquivos restantes. Sem `--force`, `mysqlimport` sai se uma tabela não existir.

`--help` Exibe uma mensagem de ajuda e sai.

`-h host_name, --host=host_name`

Importa dados para o servidor MySQL na máquina referida. A máquina padrão é `localhost`.

`-i, --ignore`

Veja a descrição para a opção `--replace`.

- l, --lock-tables
Bloqueia **TODAS** as tabelas para escrita antes de processar qualquer arquivo texto. Isto garante que todas as tabelas são sincronizadas no servidor.
- L, --local
Lê arquivos de entrada do cliente. Por padrão, é assumido que os arquivos texto estão no servidor se você conectar à `localhost` (máquina padrão).
- p, --password[=*sua_senha*]
Senha para conectar ao servidor. Se você não especificar a parte '*sua_senha*', o `mysqlimport` irá pedir por uma senha.
- P *port_num*, --port=*port_num*
O número da porta TCP/IP para usar quando conectar a uma máquina. (isto é usado para conexões a máquinas diferentes de `localhost`, onde são utilizados sockets Unix).
- r, --replace
As opções `--replace` e `--ignore` controlam o tratamento de registros de entrada que duplicam registros existentes em valores de chaves únicas. Se você especificar `--replace`, novos registros substituirão registros que tiverem o mesmo valor na chave única. Se você especificar `--ignore`, registros de entrada que duplicariam um registro existente em um valor de chave única são saltados. Se você não especificar nenhuma das duas opções, um erro ocorrerá quando um valor de chave duplicado for encontrado e o resto do arquivo texto será ignorado.
- s, --silent
Modo silencioso. Gera saída somente quando ocorrer algum erro.
- S */path/to/socket*, --socket=*/path/to/socket*
O arquivo socket para usar ao conectar à `localhost` (máquina padrão).
- u *user_name*, --user=*user_name*
O nome de usuário MySQL para usar ao conectar ao servidor. O valor padrão é seu nome de usuário atual no Unix.
- v, --verbose
Modo verbose. Gera mais informações na saída.
- V, --version
Exibe informação sobre a versão e sai.

Abaixo um exemplo da utilização de `mysqlimport`:

```
$ mysql --version
mysql Ver 9.33 Distrib 3.22.25, for pc-linux-gnu (i686)
$ uname -a
Linux xxx.com 2.2.5-15 #1 Mon Apr 19 22:21:09 EDT 1999 i586 unknown
$ mysql -e 'CREATE TABLE impctest(id INT, n VARCHAR(30))' test
$ ed
a
100      Max Sydow
101      Count Dracula
.
```

```

w imptest.txt
32
q
$ od -c imptest.txt
0000000  1  0  0  \t  M  a  x          S  y  d  o  w  \n  1  0
0000020  1  \t  C  o  u  n  t          D  r  a  c  u  l  a  \n
0000040
$ mysqlimport --local test imptest.txt
test.imptest: Records: 2 Deleted: 0 Skipped: 0 Warnings: 0
$ mysql -e 'SELECT * FROM imptest' test
+-----+-----+
| id  | n                |
+-----+-----+
| 100 | Max Sydow       |
| 101 | Count Dracula   |
+-----+-----+

```

4.8.8 Exibindo Bancos de Dados, Tabelas e Colunas

`mysqlshow` pode ser usado para exibir rapidamente quais bancos de dados existem, suas tabelas, e o nome das colunas da tabela.

Como o programa `mysql` você pode obter as mesmas informações com comandos `SHOW`. See [\(undefined\) \[SHOW\], page \(undefined\)](#).

`mysqlshow` é chamado assim:

```
shell> mysqlshow [OPÇÕES] [banco_dados [tabela [coluna]]]
```

- Se nenhum banco de dados é fornecido, todos os bancos de dados encontrados são exibidos.
- Se nenhuma tabela é fornecida, todas as tabelas encontradas no banco de dados são exibidas.
- Se nenhuma coluna for fornecida, todas colunas e tipos de colunas encontrados na tabela são exibidos.

Note que em versões mais novas do MySQL, você só visualiza as tabelas/bancos de dados/colunas para quais você tem algum privilégio.

Se o último argumento conter uma shell ou um meta-caracter do SQL, (*, ?, % ou _) somente o que coincidir com o meta-caracter é exibido. Isto pode causar alguma confusão quando alguém tentar exibir as colunas para uma tabela com um _, neste caso o `mysqlshow` exibe somente os nomes de tabelas que casarem com o padrão. Isto é facilmente corrigido adicionando um % extra na linha de comando (como um argumento separador).

4.8.9 perror, Explicando Códigos de Erros

`perror` pode ser usado para exibir mensagen(s) de erro. `perror` pode ser chamado desta forma:

```
shell> perror [OPÇÕES] [CÓDIGO_ERRO [CÓDIGO_ERRO...]]
```

Por exemplo:

```
shell> perror 64 79
Error code 64: Machine is not on the network
Error code 79: Can not access a needed shared library
```

`perror` pode ser usado para exibir uma descrição de um código de erro do sistema, ou um código de erro do manipulador de tabelas MyISAM/ISAM. As mensagens de erro são na maioria dependentes do sistema.

4.8.10 Como Executar Comandos SQL a Partir de um Arquivo Texto

O cliente `mysql` normalmente é usado de maneira interativa, desta forma:

```
shell> mysql banco_dados
```

Entretanto, também é possível colocar seus comandos SQL em um arquivo e dizer ao `mysql` para ler a entrada a partir deste arquivo. Para fazer isto, crie um arquivo texto `'arquivo_texto'` contendo os comandos que você deseja executar. Então execute o `mysql` como exibido abaixo:

```
shell> mysql banco_dados < arquivo_texto
```

Você também pode iniciar seu arquivo texto com uma instrução `USER nome_bd`. Neste caso, não é necessário especificar o nome do banco de dados na linha de comando:

```
shell> mysql < arquivo_texto
```

See [Side Scripts-snt \[Scripts do Lado do Cliente\]](#), page [Side Scripts-pg](#).

4.9 Os Arquivos Log do MySQL

O MySQL tem vários arquivos de log diferentes que podem ajudá-lo a descobrir o que está acontecendo dentro do `mysqld`:

O log de erros	Problemas encontrados iniciando, executando ou parando o <code>mysqld</code> .
O log isam	Documenta todas alterações a tabelas ISAM. Usado somente para depuração do código isam.
O log de consultas	Conexões estabelecidas e consultas executadas.
O log de atualizações	Desatualizado: Armazena todas as instruções que alteram dados.
O log binário	Armazena todas as instruções que alteram qualquer coisa. Usada também para replicação.
O log para consultas lentas	Armazena todas queries que levaram mais de <code>long_query_time</code> para executar ou que não usaram índices.

Todos logs podem ser encontrados no diretório de dados do `mysqld`. Você pode forçar o `mysqld` a reabrir os arquivos de log (ou em alguns casos trocar para um novo log) executando `FLUSH LOGS`. See [\(undefined\) \[FLUSH\]](#), page [\(undefined\)](#).

4.9.1 O Log de Erros

`mysqld` escreve todos erros para `stderr`, que o script `safe_mysqld` redireciona para um arquivo chamado `'nome_máquina'.err`. (No Windows, o `mysqld` o escreve diretamente em `'\mysql\data\mysql.err'`).

Ele contém informações indicando quando o `mysqld` foi iniciado e parado e também os erros críticos encontrados durante a execução. Se o `mysqld` finalizar inesperadamente e o `safe_mysqld` precisar reiniciar o `mysqld`, o `safe_mysqld` irá escrever uma linha `restarted mysqld` neste arquivo. Este log também armazena alertas de tabelas que necessitam ser verificadas automaticamente ou reparadas.

Em alguns sistemas operacionais, o log de erro irá conter registros de pilha de onde o `mysqld` finalizou. Isto pode ser usado para saber onde e como o `mysqld` morreu. See [\[Utilizando registros de pilha\]](#), page [\[undefined\]](#).

4.9.2 O Log de Erros Genéricos

Se você deseja saber o que acontece com `mysqld`, você deve iniciá-lo com a opção `--log[=arquivo]`. Isto irá documentar todas conexões e consultas no arquivo log (por padrão nomeado `'nome_máquina'.log`). Este log pode ser muito útil quando você suspeitar de um erro em um cliente e deseja saber exatamente o que o `mysqld` acha que o cliente enviou.

Por padrão, o script `mysql.server` inicia o servidor MySQL com a opção `-l`. Se você precisar melhorar a performance quando iniciar o uso do MySQL em um ambiente de produção, pode remover a opção `-l` do `mysql.server` ou alterá-lo para `--log-bin`.

As entradas neste log são escritas quando o `mysqld` recebe as questões. Pode estar diferente da ordem em que as instruções são executadas. Isto está em contraste com o log de atualizações e o log binário nos quais as consultas são escritas depois de serem executadas, mas que quaisquer travas sejam liberadas.

4.9.3 O Log de Atualizações

NOTA: O log de atualizações foi substituído pelo log binário. See [\[Log binário\]](#), page [\[undefined\]](#). Com ele você pode fazer qualquer coisa que poderia ser feito com o log de atualizações.

Quando iniciado com a opção `--log-update[=nome_arquivo]`, o `mysqld` grava um arquivo log contendo todos os comandos SQL que atualizam dados. Se nenhum arquivo for fornecido, o nome da máquina é usado. Se um nome de arquivo for fornecido, mas não possuir o caminho, o arquivo é gravado no diretório de dados. Se `'nome_arquivo'` não possuir uma extensão, o `mysqld` irá criar os arquivos com os nomes desta forma: `'nome_arquivo.###'`, onde `###` é um número que é incrementado cada vez que `mysqladmin refresh`, `mysqladmin flush-logs` ou a instrução `FLUSH LOGS` forem executados ou o servidor for reiniciado.

NOTA: Para o esquema acima funcionar, você não pode criar seus próprios arquivos com o mesmo nome que os do log de atualização + algumas extensões que podem ser tratadas como números, no diretório usado pelo log de atualização!

Se forem utilizadas as opções `--log` ou `-l`, o `mysqld` escreve um log geral com o nome de arquivo `'nome_máquina.log'`, e o reinício e a recarga não geram um novo arquivo de log (embora ele seja fechado e reaberto). Neste caso você pode copiá-lo (no Unix) usando:

```
mv nome_máquina.log nome_máquina-antigo.log
mysqladmin flush-logs
cp nome_máquina-antigo.log para-diretório-backup
rm nome_máquina-antigo.log
```

O log de atualização é inteligente pois registra somente instruções que realmente alteram dados. Portanto, um UPDATE ou um DELETE com uma cláusula WHERE que não encontre nenhum registro não é escrito no log. Ele salta até instruções UPDATE que atribui a uma coluna o mesmo valor que ela possuía.

O registro da atualização é feito imediatamente após uma consulta estar completa mas antes que as bloqueios sejam liberados ou que algum commit seja feito. Isto garante que o log seja escrito na ordem de execução.

Se você desejar atualizar um banco de dados a partir de arquivos de logs de atualização, você pode fazer o seguinte (assumindo que seus logs de atualização estejam nomeados na forma 'nome_arquivo.###'):

```
shell> ls -l -t -r nome_arquivo.[0-9]* | xargs cat | mysql
```

ls é utilizado para obter todos os arquivos de log na ordem correta.

Isto pode ser útil se você tiver que recorrer a arquivos de backup depois de uma falha e desejar refazer as atualizações que ocorreram entre a hora do backup e a falha.

4.9.4 O Log Binário de Atualizações

A intenção é que o log binário deve substituir o log de atualizações, portanto nós recomendamos que você troque para este formato de log o mais rápido possível!

O log binário contém toda informação que é disponível no log de atualização em um formato mais eficiente. Ele também contém informações sobre quanto tempo as consultas levaram para atualizar o banco de dados.

O log binário é também usado para replicar um mysqld escravo a partir de um mestre. See [\[Replicação\]](#), page [\[undefined\]](#).

Quando iniciado com a opção `--log-bin[=nome_arquivo]`, o mysqld escreve um arquivo de log contendo todos comandos SQL que atualizam dados. Se nenhum arquivo for fornecido, ele aponta para o nome da máquina seguido de `-bin`. Se for fornecido o nome do arquivo, mas ele não tiver o caminho, o arquivo é escrito no diretório de dados.

Se você fornecer uma extensão à `--log-bin=nome_arquivo.extensão`, a extensão será removida sem aviso.

O mysqld irá acrescentar uma extensão ao nome de arquivo do log binário que é um número que é incrementado cada vez que `mysqladmin refresh`, `mysqladmin flush-logs`, a instrução `FLUSH LOGS` forem executados ou o servidor for reiniciado.

Você pode utilizar as seguintes opções ao mysqld para afetar o que é documentado pelo log binário:

<code>binlog-do-db=nome_banco_dados</code>	Diz ao servidor para registrar atualizações para o banco de dados especificado e excluir todos os outros não mencionados explicitamente. (Exemplo: <code>binlog-do-db=algum_banco_dados</code>)
<code>binlog-ignore-db=nome_banco_dados</code>	Diz ao master que atualizações a um determinado banco de dados não devem ser registradas no log binário (Exemplo: <code>binlog-ignore-db=algum_banco_dados</code>)

Para saber quais arquivos binários foram usados, o `mysqld` irá criar também um arquivo de índice para o log binário que contém o nome de todos os arquivos de log binário usados. Por padrão este arquivo tem o mesmo nome que o arquivo de log binário, com a extensão `'.index'`. Você pode alterar o nome do arquivo de índice do log binário com a opção `--log-bin-index=[nome_arquivo]`.

Se estiver sendo usado replicação, os arquivos de log binário antigos não devem ser apagados até ter certeza que nenhum escravo irá mais precisar deles. Uma forma de fazer isto é o utilizar `mysqladmin flush-logs` uma vez por dia e então remover qualquer log com mais de 3 dias.

Você pode examinar o arquivo de log binário com o comando `mysqlbinlog`. Por exemplo, você pode atualizar um servidor MySQL a partir de um log binário como mostrado a seguir:

```
mysqlbinlog arquivo-log | mysql -h nome_servidor
```

Você também pode utilizar o programa `mysqlbinlog` para ler o log binário diretamente de um servidor MySQL remoto !

`mysqlbinlog --help` irá lhe fornecer mais informações de como usar este programa!

Se você estiver utilizando `BEGIN [WORK]` ou `SET AUTOCOMMIT=0`, você deve utilizar o log binário do MySQL para backups no lugar do antigo log de atualização.

O Log binário é feito imediatamente depois que uma consulta terminar mas antes que os bloqueios sejam liberados ou algum commit seja feito. Isto garante que o log seja feito na ordem de execução.

Todas atualizações (`UPDATE`, `DELETE` ou `INSERT`) que alteram uma tabela transacional (como tabelas BDB) são armazenadas no cache até um `COMMIT`. Quaisquer atualizações a uma tabela não transacional são armazenadas no log binário de uma vez. Todas as threads irão, no início, alocar um buffer de `binlog_cache_size` para registrar consultas. Se uma consulta é maior que o registro, a thread irá criar um arquivo temporário para lidar com a mesma. O arquivo temporário será apagado quando a thread terminar.

O `max_binlog_cache_size` pode ser usado para restringir o tamanho total usado para armazenar uma consulta multi-transacional.

Se você estiver utilizando o log de atualização ou o binário, inserções concorrentes não funcionarão juntas com `CREATE ... INSERT` e `INSERT ... SELECT`. Isto é para garantir que você possa recriar uma cópia exata de suas tabelas aplicando o log em um backup.

4.9.5 O Log para Consultas Lentas

Quando iniciado com a opção `--log-slow-queries[=file_name]` o `mysqld` escreve em um arquivo log contendo todos os comandos SQL que levam mais de `long_query_time` para executar. O tempo para obter os bloqueios de tabelas iniciais não são contados como tempo de execução.

O log de consultas lentas é gerado depois que uma query é executada e depois de todas as bloqueios serem liberados. Ela pode estar em ordem diferente da que as instruções foram executadas.

Se nenhum nome de arquivo for fornecido, o padrão é o nome da máquina com o sufixo `-slow.log`. Se um nome de arquivo for especificado, mas não conter o caminho, o arquivo é gravado no diretório de dados.

O log para queries lentas pode ser usado para encontrar queries que levam muito tempo para executar e que devem ser candidatas a otimização. Com um log muito grande, isto pode ser uma tarefa difícil. Você pode utilizar o log de consultas lentas através do comando `mysqldumpslow` para obter um resumo das consultas que aparecem no log.

Se a opção `--log-long-format` estiver sendo usada, então as consultas que não estiverem utilizando índices serão escritas. See [line options-snt \[Opções de linha de comando\]](#), page [line options-pg](#).

4.9.6 O Log de Manutenção de Arquivo

O MySQL tem vários arquivos de log que possibilitam ver o que está ocorrendo com mais facilidade. See [\[Arquivos de Log\]](#), page [\[Arquivos de Log\]](#). Porém de tempos em tempos deve ser feita uma limpeza nos arquivos de logs do MySQL para que eles não ocupem muito do espaço do disco.

Ao utilizar o MySQL com arquivos log, você necessitará de tempos em tempos remover antigos arquivos de log e dizer ao MySQL para logar com novos arquivos. See [\[Backup\]](#), page [\[Backup\]](#).

Em uma instalação Linux RedHat), você pode usar o script `mysql-log-rotate` para isto. Se você instalou o MySQL de uma distribuição RPM, o script deve ter sido instalado automaticamente. Perceba que você deve ter cuidado com isto se você estiver utilizando o log para replicação!

Em outros sistemas você deve instalar um pequeno script que será executado pelo `cron` para lidar com os arquivos de log.

Você pode forçar o MySQL a iniciar utilizando novos arquivos de log usando `mysqladmin flush-logs` ou utilizando o comando SQL `FLUSH LOGS`. Se você usa o MySQL Versão 3.21 deve utilizar o comando `mysqladmin refresh`.

O comando acima faz o seguinte:

- Se o o log padrão (`--log`) ou log de consultas lentas (`--log-slow-queries`) forem utilizados, fecha e reabre o arquivo de log. (`'mysql.log'` e `'hostname'-slow.log'` como padrão).
- Se o log de atualização (`--log-update`) é usado, fecha o log de atualização e abre um novo arquivo log com uma sequência numérica mais alta.

Se você só estiver utilizando o log de atualização, você tem apenas que atualizar os logs e então mover os arquivos de log antigos para um backup. Se você estiver utilizando o log normal, você pode fazer algo assim:

```
shell> cd diretório-dados-mysql
shell> mv mysql.log mysql.old
shell> mysqladmin flush-logs
```

e então fazer um backup e remover o `'mysql.old'`.

4.10 Replicação no MySQL

Este capítulo descreve os vários recursos da replicação no MySQL. Ele serve como uma referência para as opções disponíveis na replicação. Você será introduzido a replicação e aprenderá como implementá-la. Em direção ao final, existem algumas questões mais perguntadas (FAQ) e descrições de problemas e como resolvê-los.

4.10.1 Introdução

Um motivo de se usar replicação é para ganhar robustez e velocidade. No caso de robustez você pode ter dois sistemas e pode fazer um troca para o backup se tiver problemas com o mestre. A velocidade extra é alcançada enviando uma parte de consultas que não sejam de atualização para o servidor de replicação. É claro que isto funciona somente se consultas que não fazem atualizações forem em maior número, o que é o caso normal.

A partir da versão 3.23.15, o MySQL suporta replicação de uma via internamente. Um servidor atua como o mestre, enquanto o outro atua como escravo. Note que um servidor pode exercer a função do mestre em um momento e de escravo em outro. O servidor mestre mantém um log binário de atualizações (See [\[Log binário\]](#), page [\[Log binário\]](#).) e um arquivo índice para logs binários para manter os registro da rotatividade dos logs. O escravo, na conexão, informa ao mestre onde parou desde a última atualização propagada com sucesso, realiza a atualização e então para e espera o mestre informar sobre novas atualizações.

Note que se você estiver replicando um banco de dados, todas atualizações neste banco de dados deve ser feito através do mestre !

Outro benefício de utilizar replicação é que pode-se obter backups instantâneos do sistema fazendo backups no escravo em vez de fazê-los no mestre. See [\[Backup\]](#), page [\[Backup\]](#).

4.10.2 Visão Geral da Implementação da Replicação

A replicação no MySQL é baseia-se no fato do servidor manter o registro de todas as alterações de seus bancos de dados (atualizações, deleções, etc) no log binário. (See [\[Log binário\]](#), page [\[Log binário\]](#).) e do(s) servidor(es) escravo(s) ler(em) as consultas salvas no log binário do servidor mestre para que assim execute as mesmas consultas nos seus dados replicados.

É **muito importante** entender que o log binário é simplesmente um registro iniciando a partir de um ponto fixo no tempo (o momento que você habilitou o log binário). Quaisquer escravos que você configure necessitará de cópias de todos os dados do seu mestre com eles existiam no momento em que o log binário foi habilitado no mestre. Se você iniciar os escravos com dados que não estejam de acordo com o que está no mestre **quando o log binário foi iniciado**, seus escravos falharão.

Em uma próxima versão (4.0), não existirá a necessidade de manter uma cópia dos dados para novos escravos que você deseje configurar através da funcionalidade do backup instantâneo sem ter a necessidade de bloqueios. Entretanto, neste momento, é necessário travar todas escritas através de uma trava de leitura global ou desligar o master enquanto faz cópia.

Uma vez que o escravo foi configurado corretamente e está em execução, ele simplesmente conectará ao mestre e esperará por atualizações nos processos. Se o mestre for desligado ou o escravo perder conectividade com seu mestre, ele tentará conectar a cada `master-connect-retry` segundos até conseguir reconectar e resumir as atualizações.

Cada escravo mantém registro de onde parou. O servidor mestre não tem conhecimento de quando escravos existem ou quais estão atualizados em um determinado momento.

A próxima seção explica o processo de configuração mestre/escravo com mais detalhes.

4.10.3 Como Configurar a Replicação

Abaixo está uma descrição rápida de como configurar uma replicação completa em seu servidor MySQL atual. Ele assume que você deseja replicar todos os bancos de dados e nunca configurou uma replicação anteriormente. Você precisará desligar seu servidor mestre rapidamente para completar os passos delineados abaixo.

1. Certifique-se que você possui uma versão recente do MySQL instalado no servidor mestre e no(s) escravo(s).

Utilize a versão 3.23.29 ou superior. Releases anteriores utilizavam um formato de log binário diferente e possuía erros que foram corrigidos em versões atuais. Por favor, não relate erros até que você tenha verificado que o problema esteja presente na última versão.

2. Configure um usuário especial para a replicação na máquina master com o privilégio FILE e permissão para conectar a partir de todos os escravos. Se a função deste usuário é somente para a replicação, então você não precisará fornecer nenhum privilégio adicional para ele.

Por exemplo, para criar um usuário chamado `repl` que pode acessar seu mestre de qualquer máquina, você deve utilizar este comando:

```
GRANT FILE ON *.* TO repl@"%" IDENTIFIED BY '<senha>';
```

3. Desligue o MySQL no mestre.

```
mysqladmin -u root -p<senha> shutdown
```

4. Faça uma cópia de todos os dados existentes em seu servidor mestre.

A maneira mais fácil de fazer isto (no Unix) é simplesmente usar o comando `tar` para produzir um arquivo de todo o seu diretório de dados. A localização exata do diretório de dados depende de sua instalação.

```
tar -cvf /tmp/mysql-snapshot.tar /caminho/para/diretório-dados
```

Usuários windows podem usar o Winzip ou algum software similar para criar um arquivo do diretório de dados.

5. No arquivo `my.cnf` no mestre adicione `log-bin` e `server-id=número_único` na seção `[mysqld]` e reinicie o serviço. É muito importante que o que a identificação do servidor na máquina escrava seja diferente da identificação no mestre. Pense em `server-id` como alguma coisa similar a endereços IP - ele identifica unicamente a instância do servidor na comunidade de parceiros de replicação.

```
[mysqld]
log-bin
server-id=1
```

6. Reinicie o MySQL na máquina mestre.
7. Adicione o seguinte no `my.cnf` no(s) escravo(s):

```
master-host=<nome do mestre>
master-user=<nome do usuário de replicação>
master-password=<senha do usuário de replicação>
master-port=<porta TCP/IP para o mestre>
server-id=<algum número único entre 2 e 2^32-1>
```

trocando os valores entre `<>` com o que for relacionado ao seu sistema.

`server-id` deve ser diferente para cada servidor participante da replicação. Se você não especificar um `server-id`, ele será configurado para 1 se você não definiu `master-host`, senão ele será configurado para 2. Perceba que no caso da omissão de `server-id` o mestre irá recusar conexões de todos os escravos, e o escravo recusará a conexão a um mestre. Assim, omitindo `server-id` é bom somente para um backup com um log binário.

8. Coloque os dados copiados anteriormente no seu diretório de dados no(s) escravo(s). Tenha certeza que os privilégios nos arquivos e diretórios estão corretos. O usuário que executa o MySQL deve estar apto para ler e escrever nos bancos de dados, da mesma forma que no mestre.
9. Restart the slave(s).

Depois de completados os procedimentos acima, o(s) escravo(s) devem se conectar ao mestre e pegar todas as atualizações que ocorreram desde que o backup foi restaurado.

Se você esqueceu de configurar o `server-id` no escravo você irá obter o seguinte erro no arquivo de log:

```
Warning: one should set server_id to a non-0 value if master_host is set.
The server will not act as a slave.
```

Se você esqueceu fazer isto no mestre, os escravos não vão conseguir se conectar ao mestre.

Se um escravo não está apto para replicar por alguma razão, você encontrará a mensagem de erro no arquivo de log de erros no escravo.

Uma vez que um escravo está replicando, você encontrará um arquivo chamado `master.info` no mesmo diretório do seu log de erros. O arquivo `master.info` é usado pelo escravo para manter o registro de quanto foi processado do log binário do mestre. **Não** remova ou edite o arquivo, a menos que você realmente saiba o que está fazendo. Mesmo neste caso, é mais aconselhável usar o comando `CHANGE MASTER TO`.

4.10.4 Recursos de Replicação e Problemas Conhecidos

Abaixo uma explicação do que é e o que não é suportado:

- A Replicação será feita corretamente com valores `AUTO_INCREMENT`, `LAST_INSERT_ID` e `TIMESTAMP`.
- `RAND()` nas atualizações não replica corretamente. Utilize `RAND(alguma_expr_não_randômica)` se estiver replicando atualizações com `RAND()`. Você pode, por exemplo, usar `UNIX_TIMESTAMP()` para o argumento para `RAND()`.
- Devem ser usados os mesmos conjuntos de caracteres (`--default-character-set`) no mestre e no escravo. Se não, você pode obter erros de chaves duplicadas no escravo, pois uma chave que é tratada como única no mestre pode não ser a mesma no outro conjunto de caracteres.
- `LOAD DATA INFILE` será tratado corretamente enquanto o arquivo ainda continuar no servidor mestre no momento da propagação da atualização. `LOAD LOCAL DATA INFILE` será ignorado na replicação.
- Consultas de atualização que usam variáveis de usuários (ainda) não são seguras para replicar.

- Comandos `FLUSH` não são armazenados no log binário e por isso, não são replicadas para os escravos. Normalmente isto não é um problema já que `FLUSH` não altera nada. Entretanto, isto significa que se você atualizar as tabelas de privilégios do MySQL diretamente sem usar a instrução `GRANT` e replicar o banco de dados de privilégios do MySQL, você deve fazer um `FLUSH PRIVILEGES` nos escravos para fazer com que os novos privilégios tenham efeito.
- A partir da versão 3.23.29 tabelas temporárias são replicadas corretamente com a exceção do caso no qual o servidor escravo é desligado (não apenas a thread escrava da replicação), e algumas tabelas temporárias estão abertas e são usadas nas atualizações subsequentes. Para lidar com este problema, antes de desligar o escravo, execute `SLAVE STOP` e então verifique a variável `Slave_open_temp_tables` para ver se ela é 0, então execute `mysqladmin shutdown`. Se o número não for 0, reinicie a thread escrava com `SLAVE START` e veja se você terá mais sorte na próxima vez. Existirá uma solução melhor, mas teremos que esperá-la até a versão 4.0. Em versões anteriores, tabelas temporárias não eram replicadas corretamente - nós recomendamos que você atualize sua versão ou execute `SET SQL_LOG_BIN=0` nos seus clientes antes de todas as consultas com tabelas temporárias.
- O MySQL suporta somente um mestre e vários escravos. Na versão 4.x adicionaremos um algoritmo de votação para trocar automaticamente o mestre se alguma coisa estiver errada com o mestre atual. Iremos também introduzir processos agentes para ajudar a fazer o balanceamento de carga enviando consultas com `SELECTS` para diferentes escravos.
- A partir da versão 3.23.26 é seguro conectar servidores em um relacionamento circular com `log-slave-updates` habilitado. Perceba, entretanto, que várias queries não irão funcionar corretamente neste tipo de configuração a menos que o código do cliente seja escrito para tomar cuidado dos potenciais problemas que podem ocorrer em diferentes sequências em servidores diferentes.

Isto significa que você pode fazer uma configuração parecida com o seguinte:

```
A -> B -> C -> A
```

Esta configuração funcionará se você fizer somente atualizações não conflitantes entre as tabelas. Em outras palavras, se você inserir dados em A e C, você nunca poderá inserir uma linha em A que pode ter uma chave conflitante com uma linha inserida em C. Você também não deve atualizar as mesmas linhas em dois servidores se a ordem em que as atualizações são aplicadas importar.

Note que o formato de log foi alterado na Versão 3.23.26 portanto escravos anteriores a versão 3.23.26 não conseguirão ler estes logs.

- Se houver um erro em uma consulta no escravo, a thread escrava irá terminar e uma mensagem irá aparecer no arquivo `.err`. Você deve então conectar a um escravo manualmente, corrigir a causa do erro (por exemplo, tabela não existente), e então executar o comando `sql SLAVE START` (disponível a partir da versão 3.23.16). Na versão 3.23.15, será necessário reiniciar o servidor.
- Se a conexão para o mestre for perdida, o escravo irá tentar conectar de novo imediatamente e no caso de falhas, a cada `master-connect-retry` (padrão 60) segundos. Por causa disto, é seguro desligar o mestre, e então reiniciá-lo depois de um tempo. O escravo também está apto para lidar com interrupções de rede.

- Desligar o escravo (corretamente) também é seguro, pois mantém sinais de onde parou. Desligamentos incorretos podem produzir problemas, especialmente se o cache de disco não foi sincronizado antes do sistema morrer. Seu sistema de tolerância a falhas será melhorado se você possuir um bom No-Break ou UPS.
- Se o mestre está escutando em uma porta não padrão, você também deve especificá-la com o parâmetro `master-port` no `my.cnf`.
- Na versão 3.23.15, todas as tabelas e bancos de dados serão replicados. A partir da versão 3.23.16, você pode restringir a replicação para um conjunto de bancos de dados com diretivas `replicate-do-db` no `my.cnf` ou apenas excluir um conjunto de bancos de dados com `replicate-ignore-db`. Até a versão 3.23.23, existia um bug que não lidava corretamente com `LOAD DATA INFILE` se você fizesse isto em um banco de dados que foi excluído da replicação.
- A partir da Versão 3.23.16, `SET SQL_LOG_BIN = 0` irá desligar a replicação (binárias) no mestre e `SET SQL_LOG_BIN=1` irá ligá-la novamente - você deve possuir o privilégio `process` para fazer isto.
- A partir da versão 3.23.19, você pode limpar vestígios de replicações anteriores quando algo estiver errada acontecer e você deseja iniciar corretamente com os comandos `FLUSH MASTER` e `FLUSH SLAVE`. Na versão 3.23.26 nós os renomeamos respectivamente para `RESET MASTER` e `RESET SLAVE` para deixar claro o que eles fazem. As variantes antigas `FLUSH` ainda funcionam, para manter compatibilidade.
- A partir da versão 3.23.21, você pode utilizar `LOAD TABLE FROM MASTER` para backup de rede e configurar inicialmente a replicação. Recentemente, recebemos vários relatos de bugs relacionados e estamos investigando, portanto nós recomendamos que você o utilize somente para testes até que se torne mais estável.
- A partir da versão 3.23.23, você pode alterar os mestres e ajustar a posição do log com `CHANGE MASTER TO`.
- A partir da versão 3.23.23, você pode dizer ao mestre que atualizações em determinados bancos de dados não devem ser registradas no log binário com `binlog-ignore-db`.
- A partir da versão 3.23.26, você pode utilizar `replicate-rewrite-db` para dizer ao escravo para fazer atualizações de um banco de dados no mestre para outro com um nome diferente no escravo.
- A partir da versão 3.23.28, você pode utilizar `PURGE MASTER LOGS TO 'log-name'` para se livrar de logs antigos enquanto o escravo estiver funcionando.

4.10.5 Opções da Replicação no `my.cnf`

Se você estiver utilizando replicação, recomendamos o uso da versão 3.23.30 do MySQL ou superior. Versões antigas funcionam, mas elas possuem alguns bugs e faltam alguns recursos.

Você deve utilizar a opção `server-id` no mestre e no escravo. Isto configura um único id de replicação. Deve ser escolhido um valor único no intervalo de 1 a $2^{32}-1$ para cada mestre e escravo. Example: `server-id=3`

A tabela a seguir tem as opções que podem ser utilizadas no **MESTRE**:

Opção	Descrição
<code>log-bin=nome_arquivo</code>	Grava em um log binário de atualizações em uma localização específica. Perceba que se você fornecer um parâmetro com uma extensão (por exemplo, <code>log-bin=/mysql/logs/replication.log</code>) as versões até 3.23.24 não irão funcionar direito durante a replicação se você fizer <code>FLUSH LOGS</code> . O problema é corrigido na versão 3.23.25. Se você estiver utilizando este tipo de nome de log, <code>FLUSH LOGS</code> será ignorado no log binário. Para limpar o log, execute <code>FLUSH MASTER</code> , e não se esqueça de executar <code>FLUSH SLAVE</code> em todos os escravos. Na versão 3.23.26 e em versões posteriores você deve utilizar <code>RESET MASTER</code> e <code>RESET SLAVE</code> .
<code>log-bin-index=filename</code>	Como o usuário pode disparar o comando <code>FLUSH LOGS</code> , precisamos saber qual log está ativo no momento e quais foram rotacionados e em qual sequência. Esta informação é armazenada no arquivo de índice do log binário. O padrão é <code>'nome_máquina'.index</code> . Você pode utilizar esta opção se deseja se rebelar. Exemplo: <code>log-bin-index=db.index</code> .
<code>sql-bin-update-same</code>	Se configurado, atribuir um valor a <code>SQL_LOG_BIN</code> irá atribuir automaticamente ao parâmetro <code>SQL_LOG_UPDATE</code> o mesmo valor e vice versa.
<code>binlog-do-db=nome_bd</code>	Diz ao mestre que ele deve logar as atualizações para o log binário se o banco de dados atual for <code>'nome_bd'</code> . Todos os outros bancos de dados são ignorados. Note que se você utilizar esta opção você deve se assegurar que só poderá ser feito atualizações no banco de dados atual. Exemplo: <code>binlog-do-db=nome_bd</code> .
<code>binlog-ignore-db=nome_bd</code>	Diz ao mestre que atualizações onde o banco de dados atual é <code>'nome_bd'</code> não devem ser armazenadas no log binário. Perceba que se você usar esta opção você deve ter certeza que você somente atualizará o banco de dados atual. Exemplo: <code>binlog-ignore-db=nome_bd</code>

A tabela seguinte contém as opções que você pode utilizar para **SLAVE**:

Opção	Descrição
<code>master-host=máquina</code>	Nome da máquina mestre ou endereço IP para replicação. Se não for configurada a thread do escravo não será iniciada. Exemplo: <code>master-host=db-master.mycompany.com</code> .

<code>master-user=nome_usuário</code>	<p>O usuário que a thread escrava usará para autenticar ao conectando ao mestre. O usuário deve ter o privilégio FILE. Se <code>master-user</code> não for configurado, assume-se o usuário <code>teste</code>.</p> <p>Exemplo: <code>master-user=scott</code>.</p>
<code>master-password=senha</code>	<p>A senha que a thread escrava usará para autenticar ao conectando ao mestre. Se não for configurada, uma senha vazia é enviada.</p> <p>Exemplo: <code>master-password=tiger</code>.</p>
<code>master-port=número_porta</code>	<p>A porta que o mestre escutará. Se não for configurado, a configuração padrão de compilação <code>MYSQL_PORT</code> é assumida. Se você não tiver alterado opções do <code>configure</code>, este valor deve ser 3306.</p> <p>Exemplo: <code>master-port=3306</code>.</p>
<code>master-connect-retry=segundos</code>	<p>O número de segundos que a thread escrava dormirá antes de tentar novamente a conexão ao mestre no caso do mestre cair ou perder a conexão. o tempo padrão é 60.</p> <p>Exemplo: <code>master-connect-retry=60</code>.</p>
<code>master-info-file=nome_arquivo</code>	<p>A localização do arquivo que mostra em qual parte do mestre estávamos durante o processo de replicação. O padrão é <code>master.info</code> no diretório de dados. Sasha: A única razão que vejo para alterar o padrão é o desejo de se rebelar.</p> <p>Exemplo: <code>master-info-file=master.info</code>.</p>
<code>replicate-do-table=nome_bd.nome_tabela</code>	<p>Diz à thread escrava para restringir a replicação a uma tabela específica. Para especificar mais de uma tabela, utilize a diretiva várias vezes, uma para cada tabela. Isto irá funcionar para atualizações cruzadas em bancos de dados, em contraste com <code>replicate-do-db</code>.</p> <p>Exemplo: <code>replicate-do-table=nome_bd.nome_tabela</code>.</p>
<code>replicate-ignore-table=nome_bd.nome_tabela</code>	<p>Diz à thread escrava não replicar uma tabela específica. Para especificar mais de uma tabela a ser ignorada, utilize a diretiva múltiplas vezes, uma vez para cada tabela. Isto irá funcionar para atualizações cruzadas em bancos de dados, em contraste com <code>replicate-ignore-db</code>.</p> <p>Exemplo: <code>replicate-ignore-table=nome_bd.nome_tabela</code>.</p>

<code>replicate-wild-do-table=nome_bd.nome_tabela</code>	<p>Diz à thread escrava para restringir a replicação às tabelas que combinarem com a máscara especificado. Para especificar mais de uma tabela, utilize a diretiva diversas vezes, uma vez para cada tabela. Isto funcionará para atualizações cruzadas em bancos de dados.</p> <p>Exemplo: <code>replicate-wild-do-table=foo%.bar%</code> irá replicar atualizações em tabelas que iniciem em todos os bancos de dados cujos nomes iniciem com foo e nas tabelas cujos nomes iniciam com bar.</p>
<code>replicate-wild-ignore-table=nome_bd.nome_tabela</code>	<p>Diz à thread escrava para não replicar as tabelas que combinarem com a máscara especificada. Para especificar mais de uma tabela, utilize a diretiva diversas vezes, uma vez para cada tabela. Isto funcionará para atualizações cruzadas em bancos de dados.</p> <p>Exemplo: <code>replicate-wild-ignore-table=foo%.bar%</code> não irá atualizar as tabelas nos bancos de dados cujos nomes iniciam com foo e nas tabelas que iniciam com bar.</p>
<code>replicate-ignore-db=nome_banco_dados</code>	<p>Diz à thread escrava não replicar um banco de dados específico. Para especificar mais que um banco de dados para ignorar, utilize a diretiva várias vezes, uma para cada banco de dados. Esta opção não irá funcionar se você utilizar atualizações cruzadas em bancos de dados. Se você necessitar de atualizações cruzadas, tenha certeza que você possui o MySQL 3.23.28 ou superior, e utilize <code>replicate-wild-ignore-table=db_name.%</code></p> <p>Exemplo: <code>replicate-ignore-db=algum_bd.</code></p>
<code>replicate-do-db=nome_banco_dados</code>	<p>Diz à thread escrava restringir a replicação para um banco de dados especificado. Para especificar mais de um banco de dados, utilize a mesma diretiva diversas vezes, uma para cada banco de dados. Perceba que isto funcionará somente se você não utilizar consultas com bancos de dados cruzados como <code>UPDATE algum_bd.alguma_tabela SET foo='bar'</code> enquanto tiver selecionado um banco de dados diferentei ou nenhum banco de dados. Se você necessita ter atualizações de bancos de dados cruzados funcionando, tenha certeza que sua versão do MySQL é a 3.23.28 ou posterior e utilize <code>replicate-wild-do-table=nome_bd.%</code></p> <p>Example: <code>replicate-do-db=algum_bd.</code></p>
<code>log-slave-updates</code>	<p>Diz ao escravo para registrar as atualizações da thread escrava no log binário. Desligado por padrão. Deve ser ligado se você planeja ter escravos daisy-chain.</p>

<code>replicate-rewrite-db=do_nome->para_nome</code>	<p>Atualiza em um banco de dados com um nome diferente do original</p> <p>Exemplo: <code>replicate-rewrite-db=nome_bd_mestre->nome_bd_escravo</code>.</p>
<code>slave-skip-errors=cód_erro_1,cód_erro_2,..</code>	<p>Disponível somente na versão 3.23.47 e posterior. Diz à thread escrava para continuar a replicação quando uma consulta retornar um erro fornecido na lista. Normalmente, replicação irá para quando um erro for encontrado, dando ao usuário uma chance para resolver a inconsistência nos dados manualmente. Não utilize esta opção a menos que você entenda bem a razão dos erros. Se não existirem bugs na configuração de sua replicação e programas clientes e até mesmo no próprio MySQL, você nunca deve ter um aborto com erro. O uso indiscriminado desta opção irão resultar em escravos não confiáveis e sempre fora de sincronia com o mestre e você não ter idéia de como o problema ocorreu.</p> <p>Para códigos de erros, você deve utilizar os números fornecidos pela mensagem de erro no log de erros do escravo e na saída do comando <code>SHOW SLAVE STATUS</code>. A lista completa das mensagens de erros podem ser encontradas na distribuição fonte em <code>Docs/mysqld_error.txt</code></p> <p>Você também pode (mas não deve) usar um valor não recomendado de <code>all</code> que irá ignorar todas as mensagens de erros e manter a comunicação sem maiores critérios. Não é necessário falar, que se você usá-lo, não poderá ser assegurada a integridade de seus dados. Por favor não reclame se os dados em sua máquina escrava não estiverem nem perto dos que estão no mestre no seu caso - você foi avisado.</p> <p>Exemplo: <code>slave-skip-errors=1062,1053</code> ou <code>slave-skip-errors=all</code></p>
<code>skip-slave-start</code>	<p>Diz ao servidor escravo para não iniciar o escravo na inicialização. O usuário deve chamá-lo posteriormente com <code>SLAVE START</code>.</p>
<code>slave_read_timeout=#</code>	<p>Número de segundos a esperar por mais dados do mestre antes de abortar a leitura.</p>

4.10.6 Comandos SQL Relacionados à Replicação

A replicação pode ser controlada através da interface SQL. Abaixo segue o resumo dos comandos:

Comando	Descrição
<code>SLAVE START</code>	Inicia a thread escrava. (Escravo)
<code>SLAVE STOP</code>	Termina a thread escrava. (Escravo)

SET SQL_LOG_BIN=0	Desativa o registro de atualizações se o usuário possui o privilégio process. De outra forma, é ignorado. (Mestre)
SET SQL_LOG_BIN=1	Reabilita o registro de atualizações se o usuário possui o privilégio process. De outra maneira, é ignorado. (Mestre)
SET SQL_SLAVE_SKIP_COUNTER=n	Salta o(s) próximo(s) n eventos do mestre. Válido somente quando a thread escrava não estiver em execução, de outra forma, retorna um erro. Útil para recuperação de pequenos erros da replicação.
RESET MASTER	Apaga todos os logs binários listados no arquivo de índice, zerando o arquivo de índice. Nas versões pre-3.23.26, FLUSH MASTER (Mestre)
RESET SLAVE	Faz o escravo esquecer da sua posição da replicação nos logs do mestre. Em versões anteriores à 3.23.26 o comando era chamado FLUSH SLAVE (Escravo)
LOAD TABLE nome_tabela FROM MASTER	Descarrega uma cópia da tabela do mestre para o escravo. (Escravo)

CHANGE MASTER TO
lista_def_mestre

Altera os parâmetros do mestre para os valores especificados em `lista_def_mestre` e reinicia a thread escrava. `lista_def_mestre` é uma lista separada por vírgulas de `master_def` onde `master_def` é um dos seguintes: `MASTER_HOST`, `MASTER_USER`, `MASTER_PASSWORD`, `MASTER_PORT`, `MASTER_CONNECT_RETRY`, `MASTER_LOG_FILE`, `MASTER_LOG_POST`. Exemplo:

```
CHANGE MASTER TO
  MASTER_HOST='master2.mycompany.com',
  MASTER_USER='replication',
  MASTER_PASSWORD='big3cret',
  MASTER_PORT=3306,
  MASTER_LOG_FILE='master2-bin.001',
  MASTER_LOG_POS=4;
```

Você só precisa especificar os valores que devem ser alterados. Os valores omitidos permanecerão os mesmos com a exceção de quando você altera a máquina ou a porta. Neste caso, o escravo irá assumir que desde que você esteja conectando a uma máquina diferente ou uma porta diferente, o mestre é diferente. Assim, os valores antigos do log e sua posição não são mais aplicáveis, e serão zerados automaticamente para uma string vazia e 0, respectivamente (os valores iniciais). Perceba que se você reiniciar o escravo, ele lembrará do seu último mestre. Se isto não for desejável, você deve apagar o arquivo `master.info` antes de reiniciar, e o escravo irá ler o seu mestre do arquivo `my.cnf` ou da linha de comando. (Escravo)

SHOW MASTER STATUS

Fornecer informações de estado no log binário do mestre. (Mestre)

SHOW SLAVE STATUS

Fornecer informações de estado nos parâmetros essenciais da thread escrava. (Escravo)

SHOW MASTER LOGS

Disponível somente a partir da versão 3.23.28. Lista os logs binários no mestre. Você deve usar este comando antes de `PURGE MASTER LOGS TO` para saber até aonde deve ir.

PURGE MASTER LOGS TO
'nome_log

Disponível a partir da versão 3.23.28. Apaga todos os logs da replicação que estiverem listados no índice de log anteriores ao log especificado e também os remove do índice de log, portanto o log fornecido agora vem em primeiro. Exemplo:

```
PURGE MASTER LOGS TO 'mysql-bin.010'
```

Este comando não fará nada e irá falhar retornando um erro caso você tenha um escravo ativo que esteja lendo um dos logs que você estiver tentando apagar. Entretanto, se você possui um escravo inativo e limpar um dos logs que ele deseja ler, o escravo não conseguirá replicar no momento que iniciar. O comando é seguro para ser executado enquanto os escravos estiverem replicando - você não precisa pará-los.

Primeiro, você deve conferir todos os escravos com **SHOW SLAVE STATUS** para ver qual log estão sendo usados, então faça uma listagem dos logs no mestre com **SHOW MASTER LOGS**, encontre o log mais novo entre todos os escravos (se todos os escravos estiver atualizados, ele será o último log na lista), faça um backup de todos os logs que você deseja deletar (opcional) e limpe até o log alvo.

4.10.7 FAQ da Replicação

Q: Por quê as vezes eu vejo mais que uma thread **Binlog_Dump** no mestre depois que eu reinicio o escravo ?

R: **Binlog_Dump** é um processo contínuo que é tratado pelo servidor da seguinte maneira:

- Pegue as atualizações.
- Uma vez que não existem mais atualizações, vá para `pthread_cond_wait()`, de onde podemos despertar de uma atualização ou um kill.
- Quando acordado, confira a razão. Se não seremos finalizados, continue o loop **Binlog_dump**.
- Se existir algum erro fatal, tais como detectar um cliente finalizado, termine o loop.

Portanto, se a thread escrava parar no servidor escravo, a thread **Binlog_Dump** correspondente no mestre não irá notificá-lo até ocorrer pelo menos uma atualização no mestre (ou um kill), que é necessário para despertá-lo de `pthread_cond_wait()`. Neste intervalo, o escravo pode ter aberto outra conexão, que resulta em outra thread **Binlog_Dump**.

O problema acima não deve estar presente na versão 3.23.26 ou posterior. Na versão 3.23.26 nós adicionamos `server-id` para cada servidor da replicação, ie agora todas as antigas threads zumbis são finalizadas no mestre quando uma nova thread de replicação conectar do mesmo escravo.

Q: Como rotaciono os logs de replicação?

R: Na versão 3.23.28 você deve usar o comando **PURGE MASTER LOGS TO** depois de determinar quais logs podem ser apagados e opcionalmente fazer backups dos mesmos anteriormente. Em versões anteriores o processo era muito mais doloroso e não pode ser feita com segurança sem parar todos os escravos caso você planeje reutilizar os logs. Você precisará parar as threads escravas, editar o arquivo de índice do log binário, apagar as linhas de logs antigos, reiniciar o mestre, iniciar as threads escravas e então remover os arquivos de log antigos.

Q: Como eu faço uma atualização a sem parar uma replicação ?

R: Se você estiver atualizando versões anteriores a 3.23.26, deve apenas travar as tabelas do mestre, deixar os escravos se atualizarem, e então executar `FLUSH MASTER` no mestre e `FLUSH SLAVE` no escravo para zerar os logs, e então reiniciar novas versões do mestre e do escravo. Perceba que o escravo pode continuar derrubado por algum tempo - desde que o mestre estiver registrando todas as atualizações, o escravo irá estar apto a captá-las uma vez que esteja ligado e possa conectar.

Depois da versão 3.23.26, não mais modificamos o protocolo de replicação, assim você pode atualizar mestres e escravos sem parar a replicação para uma versão 3.23 mais nova e você pode ter divergentes versões do MySQL executando no escravo e no mestre, desde que ambos sejam mais novos que 3.23.26.

Q: Do que devo estar ciente quando for configurar uma replicação de duas vias?

R: A replicação do MySQL atualmente não suporta nenhum protocolo de bloqueio entre o mestre e o escravo para garantir a atomicidade de uma atualização distribuída (cross-server). Em outras palavras, é possível para um cliente A fazer uma atualização para um co-master 1, e no intervalo, antes dele propagar para o co-master 2, o cliente B pode fazer uma atualização para o co-master 2 que irá tornar a atualização do cliente A diferente que foi solicitado em co-master 1. Assim, quando o cliente A atualizar no co-master 2, ela irá produzir tabelas que serão diferentes das que você possui no co-master 1, mesmo depois de todas as atualizações do co-master 2 terem sido propagadas. Portanto, você não deve fazer uma ação em cadeia de dois servidores em uma relação de replicação de duas vias, a menos que você tenha certeza que suas atualizações podem seguramente acontecer em alguma ordem, ou a menos que você tome cuidado com as atualizações fora de ordem no código cliente.

Você também pode perceber que a replicação de duas vias não proporciona um grande aumento de performance, se considerarmos as atualizações. Ambos servidores precisam fazer a mesma quantidade de atualizações cada um, como se existisse apenas um servidor. A única diferença é que haverá um pouco menos de bloqueios, já que as atualizações originadas em outro servidor serão colocadas em série em uma thread escrava. Este benefício, no entanto, pode acarretar atraso na rede.

Q: Como eu posso utilizar a replicação para melhorar a performance em meu sistema?

R: Você deve configurar um servidor como o mestre, e direcionar todas as escritas para ele, e configurar quantos escravos seu dinheiro e seu espaço permitir, distribuir as leituras entre o mestre e os escravos. Você pode também iniciar os escravos com `--skip-bdb`, `--low-priority-updates` e `--delay-key-write-for-all-tables` para obter melhora na velocidade do escravo. Neste caso o escravo irá usar tabelas `MyISAM` não transacionais em vez de tabelas `BDB` para obter mais velocidade.

Q: O que eu devo fazer para preparar o meu código cliente para obter melhor performance na replicação ?

R: Se a parte do seu código que é responsável para acesso no banco de dados foi corretamente abstraído/modularizado, convertê-lo para executar com a configuração replicada deve ser muito suave e fácil - apenas altera a implementação do acesso ao banco de dados para ler de algum escravo ou do mestre e para sempre gravar no mestre. Se o seu código não possui este tipo de abstração, configurar um sistema replicado lhe dará uma oportunidade

e motivação para ajustá-lo. Você deve iniciar criando uma camada de biblioteca/modulo com as seguintes funções:

- `safe_writer_connect()`
- `safe_reader_connect()`
- `safe_reader_query()`
- `safe_writer_query()`

`safe_` significa que a função irá tomar cuidado com o tratamento de todas as condições de erro.

Então você deve converter seu código cliente para usar a camada de biblioteca. Esse processo pode ser doloroso e assustador a princípio, mas ele valerá a pena a longo prazo. Todas as aplicações que seguirem o padrão acima receberão poderão tirar vantagem de uma solução de um mestre/vários escravos. O código será um pouco mais fácil de manter, e adicionar opções para localização de defeitos será trivial. Você só precisará modificar uma ou duas funções, por exemplo, para registrar quanto tempo cada consulta demora para ser executada, ou qual consulta, entre milhares, está errada. Se você já escreveu muito código, pode precisar automatizar a tarefa de conversão utilizando o utilitário `replace` desenvolvido pelo Monty, que acompanha a distribuição padrão do MySQL, ou escreva seu próprio script Perl. Esperançosamente, seu código segue algum padrão reconhecido. Se não, provavelmente será melhor reescrever tudo novamente, ou pelo menos tente começar a seguir algum padrão.

Perceba que, é claro, você pode utilizar nomes diferentes para as funções. O que é importante é ter uma interface unificada para conexões de leitura e conexões de escrita.

Q: Quando e quanto a replicação do MySQL pode melhorar a performance de meu sistema?

A: A replicação do MySQL é mais eficaz para um sistema com leituras frequentes e que não tenha muitas escritas. Em teoria, utilizando uma configuração de um mestre para vários escravos você pode escalar adicionando mais escravos até não ter mais banda de rede, ou até que sua carga de atualizações cresça ao ponto que o mestre não consiga lidar.

Para determinar quantos escravos você pode utilizar antes dos benefícios adicionados começarem a funcionar e o quanto você pode melhorar a performance de seu site, você precisará saber seus estilos de query e de maneira empírica (por benchmarking) determinar a relação entre o ritmo de transferência nas leituras (por segundo ou `max_reads`) e nas escritas `max_writes`) em um mestre e escravo comuns. O exemplo abaixo mostra um cálculo especialmente simplificado de quanto você pode obter com replicação para seu sistema imaginário.

Digamos que a carga de seu sistema consista em 10% de escrita e 90% de leitura, e nós determinamos que `max_reads = 1200 - 2 * max_writes`, ou em outras palavras, nosso sistema pode fazer 1200 leituras por segundo com nenhuma escrita, nossa escrita média é duas vezes mais lenta que a leitura média e a relação é linear. Vamos supor que nosso mestre e escravo não tenham a mesma capacidade, e possuímos N escravos e 1 mestre. Então teremos para cada servidor (mestre ou escravo):

`leituras = 1200 - 2 * escritas` (dos benchmarks)

`leituras = 9* escritas / (N + 1)` (a leitura é dividida, mas a escrita é enviada a todos os servidores)

`9*escritas/(N+1) + 2 * escritas = 1200`

$$\text{escritas} = 1200 / (2 + 9 / (N+1))$$

Portanto se $N = 0$, que significa que não possuímos replicação, nosso sistema pode lidar com 1200/11, cerca de 109 escritas por segundo (que significa que teremos 9 vezes o número de leituras devido à natureza de nossa aplicação).

Se $N = 1$, teremos um ganho de 184 escritas por segundo.

Se $N = 8$, ganharemos 400.

Se $N = 17$, 480 escritas.

Eventualmente como N aproxima do infinito (e nosso orçamento vai infinitamente para o negativo), podemos chegar perto de 600 escritas por segundo, aumentando o rendimento do sistema em cerca de 5.5 vezes. Entretanto com somente 8 servidores já seria quase 4 vezes mais rápido.

Perceba que nossos cálculos assumem uma banda de rede infinita e ignora diversos outros fatores que podem se tornar muito significante em nosso sistema. Em vários casos, você pode não conseguir fazer uma cálculo similar ao acima predizendo de forma acurada o que acontecerá com seu sistema se você adicionar N escravos de replicação. Entretanto, responder as seguintes questões deve ajudar a sua decisão, se com tudo, a replicação melhorará a performance de seu sistema:

- Qual a razão de leitura/escrita em seu sistema?
- Quanta carga de escritas a mais o servidor pode tratar se você reduzir as leituras?
- Quantos servidores sua banda de rede pode suportar?

P: Como eu posso utilizar a replicação para fornecer redundância e alta disponibilidade?

R: Com os recursos atualmente disponíveis, você teria que configurar um mestre e um escravo (ou vários escravos) e escrever um script que irá monitorar o mestre para ver se ele está funcionando, e instruir suas aplicações e os escravos da mudança de mestre em caso de falha. Algumas sugestões:

- Para dizer a um escravo para trocar de mestre utilize o comando `CHANGE MASTER TO`.
- Uma boa forma de manter suas aplicações informadas aonde está o mestre é utilizar uma entrada de DNS dinâmica para o mestre. Com **bind** você pode utilizar `nsupdate` para atualizar dinamicamente seu DNS.
- Você deve executar seus escravos com a opção `log-bin` e sem `log-slave-updates`. Dessa forma o escravo estará pronto para virar um mestre logo que você executar o comando `STOP SLAVE; RESET MASTER` e `CHANGE MASTER TO` nos outros escravos. Isto também pode ajudar a encontrar falsas atualizações que podem acontecer por motivos de configurações erradas do escravo (idealmente, você deve configurar direitos de acesso para que nenhum cliente possa atualizar o escravo, exceto para a thread escrava) combinado com possíveis bugs em seus programas clientes (eles nunca devem atualizar o escravo diretamente).

Atualmente estamos trabalhando em integrar um sistema de eleição automática de mestres no MySQL, mas até que esteja pronto, você deverá criar suas próprias ferramentas de monitoramento.

4.10.8 Corrigindo Problemas de Replicação

Se você tiver seguido as instruções e sua configuração de replicação não estiver funcionando, primeiro elimine o fator erros de usuários verificando o seguinte:

- O mestre está gravando no log binário? Confira com `SHOW MASTER STATUS`. Se estiver, `Position` será um valor diferente de zero. Se não verifique se você forneceu ao mestre a opção `log-bin` e configurou `server-id`.
- O escravo está funcionando? Confira com `SHOW SLAVE STATUS`. A resposta é encontrada na coluna `Slave_running`. Se não, verifique as opções do escravo e verifique as mensagens no log de erros por mensagens.
- Se o escravo estiver funcionando, ele estabeleceu comunicação com o mestre? Execute o `SHOW PROCESSLIST`, encontre a thread com o valor `system user` na coluna `User` e `none` na coluna `Host` e confira a coluna `State`. Se ela conter `connecting to master`, verifique os privilégios para o usuário de replicação no mestre, o nome da máquina mestre, sua configuração DNS, se o mestre está funcionando, se ele é alcançado pelo escravo e se tudo parece ok, leia os logs de erros.
- Se o escravo estava funcionando, mas foi parado, procure na saída de `SHOW SLAVE STATUS` e confira os logs de erros. Isto normalmente acontece quando alguma consulta concluída com sucesso no mestre falha no escravo. Isto nunca deve acontecer se você fizer uma copia dos dados do mester correto e nunca modificar os dados no escravo fora da thread escrava. Se acontecer, isto é um bug, leia abaixo como reportá-lo.
- Se uma consulta que foi concluída no mestre recusar a executar no escravo e um resincronismo completo do banco de dados (o mais adequado a se fazer) não parece resolver, experimente o seguinte:
 - Veja primeiro se existe algum registro estático no caminho. Compreenda como isto ocorreu, entao apague-o e execute `SLAVE START`
 - Se o dito acima não funcionar ou não for aplicável ao seu caso, tente descobrir se seria seguro fazer a atualização manualmente (se for necessário) e então ignore a próxima consulta do mestre.
 - Se você decidiu que deve ignorar a próxima consulta, faça `SET SQL_SLAVE_SKIP_COUNTER=1; SLAVE START;` para saltar uma consulta que não utilize `auto_increment` ou `last_insert_id` ou de outra forma, `SET SQL_SLAVE_SKIP_COUNTER=2; SLAVE START;`. A razão das consultas `auto_increment/last_insert_id` serem diferentes é que elas são utilizam eventos do log binário do mestre.
 - Se você tem certeza que o escravo iniciou perfeitamente em sincronia com o mestre, e que as tabelas envolvidas nao foram atualizadas fora da thread escrava, relate o erro, para que você não precise repetir os truques acima novamente.
- Tenha certeza de que você não está tendo problemas com um erro antigo atualizando para a versão mais recente.
- Se tudo o mais falhar, leia os logs de erro. Se eles estiverem muito grandes, execute `grep -i slave /caminho/para/seu/log.err` no escravo. Não existe um modelo genérico para pesquisar no mestre, já que os únicos erros que ele registra são erros gerais de sistema - se ele puder, ele enviará os erros para o escravo quando alguma coisa estiver errada.

Quando você tiver determinado que não há erro de usuário envolvido, e a replicação ainda não funciona perfeitamente ou está instável, é hora de começar a fazer num relatório de erros. Nós precisamos do máximo de informações que você puder fornecer para conseguirmos rastrear o bug. Por favor gaste algum tempo e esforço preparando um bom relato de erro. O ideal seria nos enviar um relatório no formato encontrado no diretório `mysql-test/t/rpl*` da árvore fonte. Se você submeter um relatório como este, você pode esperar um patch em um dia ou dois na maioria dos casos, embora, é claro, o tempo pode variar dependendo de um número de fatores.

Outra opção é criar um programa com parâmetros de configuração fáceis para o mestre e o escravo que irão demonstrar o problema em nossos sistemas. Você pode escrevê-lo em Perl ou em C, dependendo da linguagem que você tenha mais domínio.

Se você pode demonstrar o problema com alguma das formas descritas acima, utilize `mysqlbug` para preparar um relatório de erros e envie-o para bugs@lists.mysql.com. Se você tem um fantasma - um problema que só pode ser simulado em seu sistema:

- Verifique se não existem erros de usuário envolvidos. Por exemplo, se você atualiza o escravo fora da thread escrava, os dados podem ficar fora de sincronia e podem ocorrer violações de chave única nas atualizações, neste caso a thread escrava irá terminar e esperar que você limpe as tabelas manualmente para entrar em sincronia.
- Execute o escravo com `log-slave-updates` e `log-bin` - isto irá registrar todas as atualizações no escravo.
- Salve todas as evidências antes de restaurar a replicação. Se não tivermos nenhuma informação ou apenas algum esboço, será um pouco mais difícil para rastreamos o problema. As evidências que você deve coletar são:
 - Todos os logs binários no mestre
 - Todos os logs binários no escravo
 - A saída de `SHOW MASTER STATUS` no mestre na hora que você descobriu o problema.
 - A saída de `SHOW SLAVE STATUS` no mestre na hora que você descobriu o problema.
 - Logs de erro no mestre e no escravo
- Utilize `mysqlbinlog` para examinar os logs binários. A informação a seguir pode ser útil para encontrar a consulta problemática, por exemplo:

```
mysqlbinlog -j pos_from_slave_status /caminho/para/log_do_escravo | head
```

Uma vez que você coletou as evidências do problema fantasma, tente isolá-lo em um caso de testes separados inicialmente. Então relate o problema para bugs@lists.mysql.com com a maior quantidade possíveis de informações.

5 Otimização do MySQL

Otimização é uma tarefa complicada porque necessita um entendimento do sistema como um todo. Enquanto for possível fazer algumas otimizações com pequeno conhecimento de seu sistema ou aplicação, quanto mais otimizado você desejar que o seu sistema esteja, mais terá que saber sobre ele.

Este capítulo tentará explicar e fornecer alguns exemplos de diferentes formas de otimizar o MySQL. Lembre-se, no entanto, que sempre existirão (cada vez mais difíceis) formas adicionais de deixar seu sistema mais rápido.

5.1 Visão Geral da Otimização

A parte mais importante para obter um sistema rápido é com certeza o projeto básico. Você também precisa saber quais tipos de coisas seu sistema estará fazendo, e quais são gargalos existentes.

Os gargalos mais comuns são:

- Pesquisa em disco. É necessário tempo para o disco encontrar uma quantidade de dados. Com discos modernos em 1999, o tempo médio para isto era normalmente menor que 10ms, portanto em teoria poderíamos fazer 1000 buscas por segundo. Este tempo melhora moderadamente com discos novos e isso é muito difícil otimizar para uma única tabela. A maneira de otimizar isto é colocando os dados em mais de um disco.
- Leitura de disco/Escrita (I/O). Quando o disco estiver na posição correta precisaremos que os dados sejam lidos. Com discos mais modernos em 1999, um disco retorna algo em torno de 10-20Mb/s. Isto é mais fácil de otimizar que as buscas porque você pode ler vários discos em paralelo.
- Ciclos de CPU. Quando tivermos os dados na memória principal (ou se eles já estiverem lá) precisaremos processá-los para conseguir nosso resultado. O fator de limitação mais comum é ter pequenas tabelas, comparadas com a memória. Mas, com pequenas tabelas, normalmente não teremos problemas com velocidade.
- Largura de banda da memória. Quando a CPU precisa de mais dados que podem caber no cache da CPU a largura de banda da memória principal se torna um gargalo. Isto é um gargalo muito incomum para a maioria dos sistemas, mas é bom estarmos cientes dele.

5.1.1 Limitações do projeto MySQL/Trocas

Como o MySQL utiliza um bloqueio de tabelas extremamente rápido (múltiplas leituras / única escritas) o maior problema restante é um conjunto de um fluxo constante de inserções e selects lentas na mesma tabela.

Nós acreditamos que para um grande número de sistemas, a performance extremamente rápida em outros casos faz desta uma escolha vencedora. Isto é normalmente possível de ser feito tendo várias cópias da tabela, mas necessita de mais esforço e hardware.

Também estamos trabalhando em algumas extensões para resolver este problema para alguns nichos comuns de aplicações.

5.1.2 Portabilidade

Como todos os servidores SQL implementam partes diferentes da linguagem SQL, é trabalhoso escrever aplicações SQL portáveis. Para inserts e selects muito simples é fácil, mas quanto mais recurso você precisa, mais difícil se torna a implementação. Se você deseja uma aplicação que seja rápida com vários bancos de dados a tarefa se tornará mais difícil ainda!

Para fazer uma aplicação complexa portátil você precisará eleger um número fechado de servidores SQL para funcionar com seu sistema.

Você pode utilizar o MySQL programa/web-page crash-me - <http://www.mysql.com/information/crash-me.php> - para encontrar funções, tipos e limites que você pode utilizar com uma seleção de servidores de bancos de dados. O Crash-me agora testa quase tudo possível, mas continua compreensível com aproximadamente 450 itens testados.

Por exemplo, você não deve ter nomes de colunas maior do que 18 caracteres se desejar utilizar o Informix ou DB2.

Os programas de benchmarks e crash-me do MySQL são bastante independentes do bancos de dados. Dando uma olhada em como nós os tratamos, você pode sentir o que é necessário para escrever sua aplicação independente do banco de dados. Os benchmarks podem ser encontrados no diretório 'sql-bench' na distribuição fonte do MySQL. Eles são escritos em Perl com a interface de banco de dados DBI (que resolve a parte do problema de acesso).

Veja <http://www.mysql.com/information/benchmarks.html> para os resultados deste benchmark.

Como pode ser visto nestes resultados, todos os bancos de dados tem alguns pontos fracos. Isto é, eles possuem diferentes compromissos de projeto que levam a comportamentos diferentes.

Se você procura por independência de banco de dados, precisará ter uma boa idéia dos gargalos de cada servidor SQL. O MySQL é muito rápido para recuperação e atualização de dados, mas terá problemas em misturar leituras/escritas lentas na mesma tabela. O Oracle, por outro lado, possui um grande problema quando você tentar acessar registros que foram recentemente atualizados (até eles serem atualizados no disco). Bancos de dados transacionais geralmente não são muito bons gerando tabelas de resumo das tabelas log, nestes casos o travamento de registros é praticamente inútil.

Para fazer sua aplicação *realmente* independente de banco de dados, você precisará definir uma interface que pode ser estendida através da qual você fará a manipulação dos dados. Como o C++ está disponível na maioria dos sistemas, faz sentido utilizar classes C++ para fazer a interface ao banco de dados.

Se você utilizar algum recurso específico para algum banco de dados (como o comando REPLACE no MySQL), você deve codificar um método para os outros servidores SQL para implementar o mesmo recurso (mas mais lento). Com o MySQL você pode utilizar a sintaxe `/*! */` para adicionar palavras chave específicas do MySQL para uma query. O código dentro de `/**/` será tratado como um comentário (ignorado) pela maioria dos servidores SQL.

Se alta performance REAL é mais importante que exatidão, como em algumas aplicações WEB, uma possibilidade é criar uma camada de aplicação que armazena todos os resultados para lhe fornecer uma performance ainda mais alta. Deixando resultados antigos 'expirar'

depois de um tempo, você pode manter o cache razoavelmente atual. Isto é muito bom no caso de uma carga extremamente pesada, pois neste caso você pode aumentar o cache dinamicamente e configurar o tempo de expiração maior até que as coisas voltem ao normal.

Neste caso a informação de criação de tabelas devem conter informações do tamanho inicial do cache e com qual frequência a tabela, normalmente, deve ser renovada.

5.1.3 Para que Utilizamos o MySQL ?

Durante o desenvolvimento inicial do MySQL, os recursos do MySQL foram desenvolvidos para atender nosso maior cliente. Eles lidam com data warehousing para alguns dos maiores varejistas na Suécia.

De todas as lojas, obtemos resumos semanais de todas as transações de cartões de bonus e esperamos fornecer informações úteis para ajudar os donos das lojas a descobrir como suas campanhas publicitárias estão afetando seus clientes.

Os dados são bem grandes (cerca de 7 milhões de transações por mês), e armazenamos dados por cerca de 4-10 anos que precisamos apresentar para os usuários. Recebemos requisições semanais dos clientes que desejam ter acesso 'instantâneo' aos novos relatórios contendo estes dados.

Resolvemos este problema armazenando todas informações mensalmente em tabelas com transações compactadas. Temos um conjunto de macros (script) que geram tabelas resumidas agrupadas por diferentes critérios (grupo de produto, id do cliente, loja...) das tabelas com transações. Os relatórios são páginas Web que são geradas dinamicamente por um pequeno shell script que analisa uma página Web, executa as instruções SQL na mesma e insere os resultados. Nós usariamos PHP ou mod_perl mas eles não estavam disponíveis na época.

Para dados graficos escrevemos um ferramenta simples em C que pode produzir GIFs baseados no resultado de uma consulta SQL (com alguns processamentos do resultado). Isto também é executado dinamicamente a partir do script Perl que analisa os arquivos HTML.

Na maioria dos casos um novo relatório pode simplesmente ser feito copiando um script existente e modificando a consulta SQL no mesmo. Em alguns casos, precisamos adicionar mais campos a uma tabela de resumo existente ou gerar uma nova, mas isto também é bem simples, pois mantemos todas as tabelas com as transações no disco. (Atualmente possuímos pelo menos 50G de tabelas com transações e 200G de outros dados do cliente.)

Nós também deixamos nossos clientes acessarem as tabelas sumárias diretamente com ODBC para que os usuários avançados possam também fazer experimentar com os dados.

Nós não tivemos nenhum problema lidando com isso em um servidor Sun Ultra SPARC-station (2x200 Mhz) bem modesto. Atualmente atualizamos um de nossos servidores para um UltraSPARC com 2 CPUs de 400 Mhz, e planejamos lidar com transações no nível de produto, o que pode significar um aumento de pelo menos dez vezes nosso volume de dados. Acreditamos que podemos lidar com isto apenas adicionando mais disco aos nossos sistemas.

Também estamos experimentando com Intel-Linux para obter mais poder de CPU por um melhor preço. Agora que possuímos o formato binários do bancos de dados portáveis (a partir da versão 3.23), começaremos a utilizá-lo para partes da aplicação.

Nossa sensação inicial é que o Linux irá atuar muito melhor em cargas baixas a médias e o Solaris irá atuar melhor quando você começar a ter uma carga alta pelo uso extremo de IO de disco, mas ainda não temos nada conclusivo sobre isto. Depois de algumas discussões com um desenvolvedor do Kernel do Linux, concluímos que isto pode ser um efeito colateral do Linux; fornecer tantos recursos para uma tarefa batch que a performance interativa se torna muito baixa. Isto deixa a máquina muito lenta e sem resposta enquanto grandes batches estiverem em execução. Esperamos que isto tenha um tratamento melhor em futuras versões do kernel Linux.

5.1.4 O Pacote de Benchmark do MySQL

Esta seção deve conter uma descrição técnica do pacote de benchmarks do MySQL (e **crash-me**), mas a descrição ainda não está pronta. Atualmente, você pode ter uma boa idéia do benchmark verificando os códigos e resultados no diretório 'sql-bench' em qualquer distribuição fonte do MySQL.

Este conjunto de benchmark pretende ser um benchmark que irá dizer a qualquer usuário se uma determinada implementação SQL irá atuar bem ou mal.

Note que este benchmark utiliza uma única thread, portanto ele mede o tempo mínimo para as operações. Planejamos no futuro adicionar vários testes multi-threaded no conjunto de benchmark.

Por exemplo, (executado na mesma máquina NT 4.0):

Lendo 2000000 linhas por índice	Segundos	Segundos
mysql	367	249
mysql_odbc	464	
db2_odbc	1206	
informix_odbc	121126	
ms-sql_odbc	1634	
oracle_odbc	20800	
solid_odbc	877	
sybase_odbc	17614	
Inserindo (350768) linhas	Segundos	Segundos
mysql	381	206
mysql_odbc	619	
db2_odbc	3460	
informix_odbc	2692	
ms-sql_odbc	4012	
oracle_odbc	11291	
solid_odbc	1801	
sybase_odbc	4802	

No teste acima o MySQL foi executado com um cache de índices de 8M.

Temos concentrado alguns resultados de benchmarks em <http://www.mysql.com/information/benchmark>

Perceba que a Oracle não está incluída porque eles solicitaram a remoção. Todos benchmarks Oracle devem ser aprovados pela Oracle! Acreditamos que os benchmarks da Oracle são **MUITO** tendencioso pois os benchmarks acima devem ser executados supostamente para uma instalação padrão para um único cliente.

Para executar a suite de benchmarks, você deve fazer download de uma distribuição fonte do MySQL, instalar o driver DBI do perl, e o driver DBD do perl para o banco de dados que desejar testar e depois fazer:

```
cd sql-bench
perl run-all-tests --server=#
```

onde # é um dos servidores suportados. Você pode obter uma lista de todos parâmetros e servidores suportados executando `run-all-tests --help`.

`crash-me` tenta determinar quais recursos um banco de dados suporta e quais suas capacidades e limitações atuais para a execução de consultas. Por exemplo, ele determina:

- Quais tipos de colunas são suportados
- Quantos índices são suportados
- Quais funções são suportadas
- Qual o tamanho máximo de uma query
- Qual o tamanho máximo de um registro do tipo `VARCHAR`

Podemos encontrar o resultado do `crash-me` para diversos bancos de dados em <http://www.mysql.com/information/crash-me.php>.

5.1.5 Utilizando seus Próprios Benchmarks

Definitivamente você deve fazer benchmarks de sua aplicação e banco de dados para saber quais são os gargalos. Corrigindo (ou substituindo o gargalho com um 'módulo burro') você pode facilmente identificar o próximo gargalo (e continuar). Mesmo se a performance geral para sua aplicação é suficiente, você deve pelo menos criar um plano para cada gargalo e decidir como resolvê-lo se algum dia você precisar de performance extra.

Para um exemplo de programas de benchmarks portáteis, consulte o conjunto de benchmarks do MySQL. See [\[Benchmarks do MySQL\]](#), page [\[Benchmarks do MySQL\]](#). Você pode pegar qualquer programa deste conjunto e modificá-lo para suas necessidades. Fazendo isto você pode tentar soluções diferentes para seu problema e testar qual solução é a mais rápida para você.

É muito comum que alguns problemas somente ocorram quando o sistema estiver muito carregado. Nós tivemos alguns clientes que nos contactaram quando eles testaram um sistema em produção e encontraram problemas de carga. Em cada um dos casos, existiam problemas com projeto básico (busca em tabelas não eram bons com alta carga) ou detalhes de SO/biblioteca. A maioria destes seriam **MUITO** mais fáceis de resolver se os sistemas já não estivessem em uso.

Para evitar problemas deste tipo, você deve colocar algum esforço em testar a performance de toda sua aplicação sobre a pior carga possível! Você pode utilizar o Super Smack para isto, e ele está disponível em: <http://www.mysql.com/Downloads/super-smack/super-smack-1.0.tar.gz>. Como o nome sugere, ele pode derrubar seu sistema se você solicitar, portanto, utilize-o somente em sistemas de desenvolvimento.

5.2 Otimizando SELECTs e Outras Consultas

Primeiramente, uma coisa que afeta todas as consultas: Quanto mais complexo seu sistema de permissões, maior a sobrecarga.

Se você não tiver nenhuma instrução **GRANT** realizada, MySQL otimizará a verificação de permissões de alguma forma. Dessa forma, se você possui um volume muito alto, o tempo pode piorar tentando permitir o acesso. Por outro lado, maior verificação de permissões resulta em uma sobrecarga maior.

Se o seu problema é com alguma função explícita do MySQL, você pode sempre consultar o tempo da mesma com o cliente MySQL:

```
mysql> select benchmark(1000000,1+1);
+-----+
| benchmark(1000000,1+1) |
+-----+
|                          0 |
+-----+
1 row in set (0.32 sec)
```

O exemplo acima demonstra que o MySQL pode executar 1.000.000 expressões + em 0.32 segundos em um PentiumII 400MHz.

Todas funções MySQL devem ser bem otimizadas, mas existem algumas excessões e o `benchmark(loop_count,expression)` é uma ótima ferramenta para saber se existe um problema com sua query.

5.2.1 Sintaxe de EXPLAIN (Obter informações sobre uma SELECT)

```
EXPLAIN nome_tabela
ou EXPLAIN SELECT opções_select
```

`EXPLAIN nome_tabela` é um sinônimo para `DESCRIBE nome_tabela` ou `SHOW COLUMNS FROM nome_tabela`.

Quando uma instrução `SELECT` for precedida da palavra chave `EXPLAIN`, o MySQL explicará como ele deve processar a `SELECT`, fornecendo informação sobre como as tabelas estão sendo unidas e em qual ordem.

Com a ajuda de `EXPLAIN`, você pode ver quando devem ser adicionados índices à tabelas para obter uma `SELECT` mais rápida que utiliza índices para encontrar os registros. Você também pode ver se o otimizador une as tabelas em uma melhor ordem. Para forçar o otimizador a utilizar uma ordem específica de join para uma instrução `SELECT`, adicione uma cláusula `STRAIGHT_JOIN`.

Para ligações mais complexas, `EXPLAIN` retorna uma linha de informação para cada tabela utilizada na instrução `SELECT`. As tabelas são listadas na ordem que seriam lidas. O MySQL soluciona todas as joins utilizando um método multi-join de varedura simples. Isto significa que o MySQL lê uma linha da primeira tabela, depois encontra uma linha que combina na segunda tabela, depois na terceira tabela e continua. Quando todas tabelas são processadas, ele exhibe as colunas selecionadas e recua através da lista de tabelas até uma tabela na qual existem registros coincidentes for encontrada. O próximo registro é lido desta tabela e o processo continua com a próxima tabela.

A saída de `EXPLAIN` inclui as seguintes colunas:

table	A tabela para a qual a linha de saída se refere.
type	O tipo de join. Informações sobre os vários tipos são fornecidas abaixo.
possible_keys	<p>A coluna possible_keys indica quais índices o MySQL pode utilizar para encontrar os registros nesta tabela. Note que esta coluna é totalmente independente da ordem das tabelas. Isto significa que algumas das chaves em possible_keys podem não ser usadas na prática com a ordem de tabela gerada.</p> <p>Se esta coluna estiver vazia, não existem índices relevantes. Neste caso, você poderá melhorar a performance de sua query examinando a cláusula WHERE para ver se ela refere a alguma coluna ou colunas que podem ser indexadas. Se for verdade, crie um índice apropriado e confira a consulta com EXPLAIN novamente. See (undefined) [ALTER TABLE], page (undefined).</p> <p>Para ver os índices existentes em uma tabela, utilize SHOW INDEX FROM nome_tabela.</p>
key	A coluna key indica a chave que o MySQL decidiu usar. A chave será NULL se nenhum índice for escolhido. Se o MySQL escolher o índice errado, você provavelmente deve forçar o MySQL a usar outro índice utilizando myisamchk --analyze , ou See (undefined) [myisamchk syntax] , page (undefined) ou USE INDEX/IGNORE INDEX . See (undefined) [JOIN] , page (undefined) .
key_len	A coluna key_len indica o tamanho da chave que o MySQL decidiu utilizar. O tamanho será NULL se key for NULL . Note que isto nos diz quantas partes de uma chave multi-partes o MySQL realmente está utilizando.
ref	A coluna ref exhibe quais colunas ou constantes são usadas com a key para selecionar registros da tabela.
rows	A coluna rows informa o número de linhas que o MySQL deve examinar para executar a consulta.
Extra	<p>Esta coluna contém informações adicionais de como o MySQL irá resolver a consulta. A seguir uma explicação das diferentes strings de texto que podem ser encontradas nesta coluna:</p> <p>Distinct O MySQL não continuará a procurar por mais registros para a combinação de registro atual depois de ter encontrado o primeiro registro coincidente.</p> <p>Not exists O MySQL estava apto a fazer uma otimização LEFT JOIN na consulta e não examinará mais registros nesta tabela para a combinação do registro anterior depois que encontrar um registro que satisfaça o critério do LEFT JOIN.</p> <p>Exemplo:</p> <pre>SELECT * FROM t1 LEFT JOIN t2 ON t1.id=t2.id WHERE t2.id IS NUL</pre> <p>Assume que t2.id é definido com NOT NULL. Neste caso o MySQL irá percorrer t1 e procurar pelos registros em t2 através de t1.id. Se o MySQL encontrar um registro combinando em t2, ele sabe que</p>

`t2.id` nunca poderá ser `NULL` e não irá percorrer até o resto dos registros em `t2` que possuírem o mesmo `id`. Em outras palavras, para cada registro em `t1` o MySQL só precisa fazer uma única pesquisa em `t2`, independente de quantos registros coincidentes existirem em `t2`.

range checked for each record (index map: #)

O MySQL não encontrou um bom índice para usar. No lugar, ele irá fazer uma verificação sobre qual índice usar (se existir) para cada combinação das tabelas precedentes, e usará este índice para recuperar os registros da tabela. Isto não é muito rápido mas é mais rápido que fazer um `join` sem um índice.

Using filesort

O MySQL precisará fazer uma passada extra para descobrir como recuperar os registros na ordem de classificação. A classificação é feita indo através de todos os registros de acordo com `join type` e armazenar a chave de ordenação mais o ponteiro para o registro para todos os registros que combinarem com o `WHERE`. Então as chaves são classificadas. Finalmente os registros são recuperados na ordem de classificação.

Using index

A informação da coluna é recuperada da tabela utilizando somente informações na árvore de índices sem ter que fazer uma pesquisa adicional para ler o registro atual. Isto pode ser feito quando todas as colunas usadas para a tabela fizerem parte do mesmo índice.

Using temporary

Para resolver a consulta, o MySQL precisará criar uma tabela temporária para armazenar o resultado. Isto acontece normalmente se você fizer um `ORDER BY` em um conjunto de colunas diferentes das quais você fez um `GROUP BY`.

Where used

Uma cláusula `WHERE` será utilizada para restringir quais registros serão combinados com a próxima tabela ou enviar para o cliente. se você não possui esta informação e a tabela é do tipo `ALL` ou `index`, pode existir alguma coisa errada na sua query (Se você não pretender examinar todos os registros da tabela).

Se você desejar deixar suas consultas o mais rápido possível, você deve dar uma olhada em `Using filesort` e `Using temporary`.

Os diferentes tipos de `joins` estão listados abaixo, ordenados do melhor para o pior tipo:

- `system` A tabela tem somente um registro (= tabela de sistema). Este é um caso especial do `join` do tipo `const`.
- `const` A tabela possui pelo menos um registro coincidente, que será lido no início da consulta. Como só há um registro, valores da coluna neste registro podem ser tratados como constantes pelo otimizador até o final. Tabelas `const` são muito rápidas como elas são lidas somente uma vez!

eq_ref	Um registro será lido desta tabela para cada combinação de registros das tabelas anteriores. Este é o melhor tipo possível de join, depois dos tipos const . Ele é utilizado quando todas as partes de um índice são usados pelo join e o índice for UNIQUE ou um PRIMARY KEY .
ref	Todos os registros com valores de índices combinando serão lidos desta tabela para cada combinação de registros das tabelas anteriores. ref é utilizado se o join usar somente o prefixo mais a esquerda da chave, ou se a chave não for UNIQUE ou uma PRIMARY KEY (em outras palavras, se a ligação não pode selecionar um único registro baseado no valor da chave). Se a chave usada coincidir apenas com poucos registros este tipo de join é de boa qualidade.
range	Serão recuperados somente os registros que estiverem dentro de uma faixa dada, utilizando um índice para selecionar os registros. A coluna key indica qual índice é usado. O key_len contém a parte mais longa da chave foi usada. A coluna ref será NULL para este tipo.
index	Isto é o mesmo que ALL , exceto que somente a árvore de índice é lida. Isto normalmente é mais rápido que ALL já que o arquivo de índice é normalmente menor que o arquivo de dados.
ALL	Uma pesquisa completa será feita na tabela para cada combinação de registros das tabelas anteriores. Isto normalmente não é bom se a tabela é a primeira não marcada com const , e normalmente muito ruim em todos os outros casos. Normalmente você pode evitar ALL adicionando mais índices, assim o registro pode ser recuperado baseado em valores contantes ou valores de colunas de tabelas anteriores.

Você pode ter uma boa indicação de quão boa é sua join multiplicando todos os valores na coluna **rows** na saída de **EXPLAIN**. Isto deve dizer a grosso modo quantos registros o MySQL deve examinar para executar a consulta. Este número é também usado quando você restringe consultas com a variável **max_join_size**. See [\[Parâmetros de servidor\]](#), page [\[Parâmetros de servidor\]](#).

O exemplo a seguir mostra como um **JOIN** pode ser otimizado progressivamente utilizando a informação fornecida por **EXPLAIN**.

Suponha que você tem a instrução **SELECT** exibida abaixo, que você está examinando utilizando **EXPLAIN**:

```
EXPLAIN SELECT tt.TicketNumber, tt.TimeIn,
             tt.ProjectReference, tt.EstimatedShipDate,
             tt.ActualShipDate, tt.ClientID,
             tt.ServiceCodes, tt.RepetitiveID,
             tt.CurrentProcess, tt.CurrentDPPerson,
             tt.RecordVolume, tt.DPPrinted, et.COUNTRY,
             et_1.COUNTRY, do.CUSTNAME
FROM tt, et, et AS et_1, do
WHERE tt.SubmitTime IS NULL
      AND tt.ActualPC = et.EMPLOYID
      AND tt.AssignedPC = et_1.EMPLOYID
      AND tt.ClientID = do.CUSTNMBR;
```

Para este exemplo, assumo que:

- As colunas comparadas foram declaradas como a seguir:

Tabela	Coluna	Tipo da coluna
tt	ActualPC	CHAR(10)
tt	AssignedPC	CHAR(10)
tt	ClientID	CHAR(10)
et	EMPLOYID	CHAR(15)
do	CUSTNMBR	CHAR(15)

- As tabelas possuem os índices mostrados abaixo:

Tabela	Índice
tt	ActualPC
tt	AssignedPC
tt	ClientID
et	EMPLOYID (chave primária)
do	CUSTNMBR (chave primária)

- Os valores tt.ActualPC não foram distribuídos de maneira uniforme.

Inicialmente, antes de ser feita qualquer otimização, a instrução EXPLAIN produzia a seguinte informação.

```

table type possible_keys          key  key_len ref  rows  Extra
et   ALL  PRIMARY                    NULL NULL  NULL  74
do   ALL  PRIMARY                    NULL NULL  NULL 2135
et_1 ALL  PRIMARY                    NULL NULL  NULL  74
tt   ALL  AssignedPC,ClientID,ActualPC NULL NULL  NULL 3872
      range checked for each record (key map: 35)

```

Como o tipo é ALL em todas tabelas, esta saída indica que o MySQL está fazendo uma ligação completa em todas as tabelas! Isto levará muito tempo para ser executado, pois o produto do número de registros em cada tabela deve ser examinado! Neste caso, existem $74 * 2135 * 74 * 3872$ registros. Se as tabelas forem maiores, imagine quanto tempo este tipo de consulta pode demorar.

Um dos problemas aqui é que o MySQL não pode (ainda) utilizar índices em colunas de maneira eficiente se elas foram declaradas de forma diferente. Neste contexto, VARCHAR e CHAR são o mesmo a menos que tenham sido declarados com tamanhos diferentes. Como tt.ActualPC é declarado como CHAR(10) e et.EMPLOYID é declarado como CHAR(15), existe aqui uma diferença de tamanho.

Para corrigir esta diferença entre tamanhos de registros, utilize ALTER TABLE para alterar o tamanho de ActualPC de 10 para 15 caracteres:

```
mysql> ALTER TABLE tt MODIFY ActualPC VARCHAR(15);
```

Agora ambos campos tt.ActualPC e et.EMPLOYID são VARCHAR(15). Executando a instrução EXPLAIN novamente produzirá este resultado:

```

table type possible_keys          key  key_len ref  rows  Extra
tt   ALL  AssignedPC,ClientID,ActualPC NULL NULL  NULL 3872  where used
do   ALL  PRIMARY                    NULL NULL  NULL 2135
      range checked for each record (key map: 1)
et_1 ALL  PRIMARY                    NULL NULL  NULL  74
      range checked for each record (key map: 1)
et   eq_ref PRIMARY              PRIMARY 15      tt.ActualPC 1

```

Isto não está perfeito, mas está bem melhor (o produto dos valores de `rows` agora menor por um fator de 74). Esta versão é executada em vários segundos.

Uma segunda alteração pode ser feita para eliminar as diferenças de tamanho das colunas para as comparações `tt.AssignedPC = et_1.EMPLOYID` e `tt.ClientID = do.CUSTNMBR` :

```
mysql> ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),
        MODIFY ClientID VARCHAR(15);
```

Agora `EXPLAIN` produz a saída mostrada abaixo:

table	type	possible_keys	key	key_len	ref	rows	Extra
et	ALL	PRIMARY	NULL	NULL	NULL	74	
tt	ref	AssignedPC,ClientID,ActualPC	ActualPC	15	et.EMPLOYID	52	where used
et_1	eq_ref	PRIMARY	PRIMARY	15	tt.AssignedPC	1	
do	eq_ref	PRIMARY	PRIMARY	15	tt.ClientID	1	

Este resultado é quase o melhor que se pode obter.

O problema restante é que, por padrão, o MySQL assume que valores na coluna `tt.ActualPC` estão distribuídos igualmente, e este não é o caso para a tabela `tt`. Felizmente, é fácil informar ao MySQL sobre isto:

```
shell> myisamchk --analyze caminho_para_banco_dados_MySQL/tt
shell> mysqladmin refresh
```

Agora a join está perfeita, e `EXPLAIN` produz esta saída:

table	type	possible_keys	key	key_len	ref	rows	Extra
tt	ALL	AssignedPC,ClientID,ActualPC	NULL	NULL	NULL	3872	where used
et	eq_ref	PRIMARY	PRIMARY	15	tt.ActualPC	1	
et_1	eq_ref	PRIMARY	PRIMARY	15	tt.AssignedPC	1	
do	eq_ref	PRIMARY	PRIMARY	15	tt.ClientID	1	

Perceba que a coluna `rows` na saída de `EXPLAIN` é uma boa ajuda para otimizador de joins do MySQL. Para otimizar uma consulta, você deve conferir se os números estão perto da realidade. Se não, você pode obter melhor desempenho utilizando `STRAIGHT_JOIN` em sua instrução `SELECT` e tentar listar as tabelas em uma ordem diferente na cláusula `FROM`.

5.2.2 Estimando o Desempenho de uma Consulta

Na maioria dos casos você pode estimar a performance contando buscas em disco. Para tabelas pequenas, normalmente você pode encontrar o registro com 1 pesquisa em disco (uma vez que o índice provavelmente está no cache). Para tabelas maiores, você pode estimar (usando índices de árvores B++) que você precisará de: $\log(\text{row_count}) / \log(\text{index_block_length} / 3 * 2 / (\text{index_length} + \text{data_pointer_length})) + 1$ buscas em disco para encontrar um registro.

No MySQL um bloco de índice tem geralmente 1024 bytes e o ponteiro de dados 4 bytes. Uma tabela de 500.000 registros com um índice com tamanho de 3 (inteiro médio) lhe dá: $\log(500,000) / \log(1024/3*2/(3+4)) + 1 = 4$ pesquisas.

Como o índice acima necessita cerca de $500,000 * 7 * 3/2 = 5.2\text{M}$, (assumindo que os buffers de índices são carregados até 2/3, que é o normal) você provavelmente terá grande parte dos índices em memória e provavelmente precisará somente de 1 ou 2 chamadas para ler dados do SO para encontrar o registro.

Entretanto, para escritas, você precisará utilizar 4 requisições para encontrar onde posicionar o novo índice e normalmente 2 buscas para atualizar o índice e escrever o registro.

Perceba que o que foi dito acima não significa que sua aplicação perderá performance por $N \log N$! Como tudo é armazenado no cache de seu SO ou do servidor SQL as coisas começarão a ficar um pouco mais lentas quando as tabelas começarem a crescer. Quando os dados se tornam muito grandes para o cache, as coisas começarão a ficar bem mais lentas até que suas aplicações estejam limitadas a buscas em disco (o que aumenta em $N \log N$). Para evitar isto, aumente o cache de índice quando os dados crescerem. See [\[Parâmetros do servidor\]](#), page [\[undefined\]](#).

5.2.3 Velocidade das Consultas que Utilizam SELECT

Em geral, quando você desejar tornar uma consulta lenta `SELECT ... WHERE` mais rápida, a primeira coisa que deve ser conferida é se você pode ou não adicionar um índice. See [\[índices MySQL\]](#), page [\[undefined\]](#). Todas as referências entre diferentes tabelas devem ser feitas normalmente com índices. Você pode utilizar o comando `EXPLAIN` para determinar quais índices são usados para uma `SELECT`. See [\[EXPLAIN\]](#), page [\[undefined\]](#).

Algumas dicas gerais:

- Para ajudar o MySQL a otimizar melhor as consultas, execute `myisamchk --analyze` em uma tabela depois dela ter sido carregada com dados relevantes. Isto atualiza um valor para cada parte do índice que indica o número médio de registros que tem o mesmo valor. (Para índices únicos, isto é sempre 1, é claro). O MySQL usará isto para decidir qual índice escolher quando você conectar duas tabelas utilizando uma 'expressão não constante'. Os resultados de `analyze` podem ser conferidos utilizando `SHOW INDEX FROM nome_tabela` e examinando a coluna `Cardinality`.
- Para ordenar um índice e dados de acordo com um índice, utilize `myisamchk --sort-index --sort-records=1` (se você deseja ordenar pelo índice 1). Se você possui um índice unico no qual deseja ler todos registros na ordem do índice, esta é uma boa forma para torná-lo mais rápido. Perceba entretanto, que esta ordenação não foi escrita de maneira otimizada e levará muito tempo em tabelas grandes!

5.2.4 Como o MySQL Otimiza Cláusulas WHERE

As otimizações `WHERE` são colocadas aqui na parte da `SELECT` porque normalmente elas são usadas com `SELECT`, mas as mesmas otimizações aplicam-se para `WHERE` em instruções `DELETE` e `UPDATE`.

Note também que esta seção está incompleta. O MySQL faz várias otimizações e ainda não tivemos tempo para documentarmos todas elas.

Algumas das otimizações feitas pelo MySQL são são listadas abaixo:

- Remoção de parênteses desnecessários:


```
((a AND b) AND c OR (((a AND b) AND (c AND d))))
-> (a AND b AND c) OR (a AND b AND c AND d)
```
- Enlaços de constantes:


```
(a<b AND b=c) AND a=5
-> b>5 AND b=c AND a=5
```

- Remoção de condições contantes (necessário por causa dos enlaços de contantes):
 $(B \geq 5 \text{ AND } B = 5) \text{ OR } (B = 6 \text{ AND } 5 = 5) \text{ OR } (B = 7 \text{ AND } 5 = 6)$
 -> $B = 5 \text{ OR } B = 6$
- Expressões constantes utilizadas por índices são avaliadas somente uma vez.
- `COUNT(*)` em uma única tabela sem um `WHERE` é recuperado diretamente da informação da tabela. Isto também é feito para qualquer expressão `NOT NULL` quando usada somente com uma tabela.
- Pré detecção de expressões contantes inválidas. O MySQL detecta rapidamente que algumas instruções `SELECT` são impossíveis e não retornará registros.
- `HAVING` é fundido com `WHERE` se não for utilizado `GROUP BY` ou funções de agrupamento (`COUNT()`, `MIN()`...).
- Para cada sub-join, um `WHERE` mais simples é construído para obter uma avaliação mais rápida de `WHERE` para cada sub-join e também para saltar registros da maneira mais rápida possível.
- Todas tabelas constantes são lidas primeiro, antes de qualquer tabelas na consulta. Uma tabela constante é:
 - Uma tabela vazia ou uma tabela com 1 registro.
 - Uma tabela que é usada com uma cláusula `WHERE` em um índice `UNIQUE`, ou uma `PRIMARY KEY`, onde todas as partes do índice são usadas com expressões constantes e as partes do índice são definidas como `NOT NULL`.

Todas as tabelas seguintes são usadas como tabelas constantes:

```
mysql> SELECT * FROM t WHERE primary_key=1;
mysql> SELECT * FROM t1,t2
        WHERE t1.primary_key=1 AND t2.primary_key=t1.id;
```

- A melhor combinação de join para unir as tabelas é encontrada tentando todas as possibilidades. Se todas colunas em `ORDER BY` e em `GROUP BY` vierem da mesma tabela, então esta tabela será preferencialmente a primeira na união.
- Se existirem uma cláusula `ORDER BY` e uma `GROUP BY` diferente, ou se a `ORDER BY` ou `GROUP BY` conterem colunas de tabelas diferentes da primeira tabela na fila de join, uma tabela temporária será criada.
- Se você utilizar `SQL_SMALL_RESULT`, o MySQL usará a tabela temporária em memória.
- Cada índice de tabela é consultado e o melhor índice que cobrir menos de 30% dos registros é usado. Se nenhum índice for encontrado, uma varredura rápida é feita pela tabela.
- Em alguns casos, o MySQL pode ler registros do índice mesmo sem consultar o arquivo de dados. Se todas colunas usadas do índice são numéricas, então somente a árvore de índice é usada para resolver a consulta.
- Antes de dar saída em cada registro, aqueles que não combinam com a cláusula `HAVING` são ignorados.

Alguns exemplos de consultas muito rápidas:

```
mysql> SELECT COUNT(*) FROM nome_tabela;
mysql> SELECT MIN(key_part1),MAX(key_part1) FROM nome_tabela;
mysql> SELECT MAX(key_part2) FROM nome_tabela
```

```

WHERE key_part_1=constant;
mysql> SELECT ... FROM nome_tabela
ORDER BY key_part1,key_part2,... LIMIT 10;
mysql> SELECT ... FROM nome_tabela
ORDER BY key_part1 DESC,key_part2 DESC,... LIMIT 10;

```

As seguintes consultas são resolvidas utilizando somente a árvore de índices (assumindo que as colunas indexadas são numéricas):

```

mysql> SELECT key_part1,key_part2 FROM nome_tabela WHERE key_part1=val;
mysql> SELECT COUNT(*) FROM nome_tabela
WHERE key_part1=val1 AND key_part2=val2;
mysql> SELECT key_part2 FROM nome_tabela GROUP BY key_part1;

```

As consultas a seguir utilizam indexação para recuperar os registros na ordem de classificação sem um passo de ordenação separado:

```

mysql> SELECT ... FROM nome_tabela ORDER BY key_part1,key_part2,... ;
mysql> SELECT ... FROM nome_tabela ORDER BY key_part1 DESC,key_part2 DESC,... ;

```

5.2.5 Como o MySQL Otimiza Cláusulas DISTINCT

DISTINCT é convertido em GROUP BY em todas as colunas. DISTINCT combinado com ORDER BY também irá em vários casos criar uma tabela temporária.

Quando combinando LIMIT # com DISTINCT, o MySQL irá parar logo que encontrar # registros únicos.

Se você não utiliza colunas de todas tabelas usadas, o MySQL irá parar a varredura das tabelas não usadas logo que encontrar a primeira coincidência.

```
SELECT DISTINCT t1.a FROM t1,t2 where t1.a=t2.a;
```

Neste caso, assumindo que t1 é usando antes de t2 (confira com EXPLAIN), MySQL irá parar de ler de t2 (para aquele registro particular em t1) quando o primeiro registro em t2 for encontrado.

5.2.6 How MySQL Optimizes LEFT JOIN and RIGHT JOIN

A LEFT JOIN B no MySQL é implementando como a seguir:

- A tabela B é configurada para ser dependente da tabela A e de todas as tabelas das quais A depende.
- A tabela A é configurada para ser dependente de todas as tabelas (exceto B) que são usadas na condição LEFT JOIN.
- Todas as condições LEFT JOIN são movidas para a cláusula WHERE.
- Todas as otimizações padrões de join são feitas, com a exceção que uma tabela é sempre lida depois de todas as tabelas das quais é dependente. Se existir uma dependência circular o MySQL irá emitir um erro.
- Todas as otimizações padrões de WHERE são realizadas.
- Se existir um registro em A que coincida com a cláusula WHERE, mas não existir nenhum registro em B que comincida com a condição LEFT JOIN então um registro extra em B é gerado com todas as colunas com valor NULL.

- Se você utiliza `LEFT JOIN` para encontrar registros que não existem em alguma tabela e está usando o seguinte teste: `nome_coluna IS NULL` na parte `WHERE`, onde `nome_coluna` é um campo que é declarado como `NOT NULL`, então o MySQL para de pesquisar por mais registros (para uma combinação particular de chaves) depois de ter encontrado um registro que combinar com a condição `LEFT JOIN`.

`RIGHT JOIN` é implementado de forma análoga à `LEFT JOIN`.

A ordem de leitura das tabelas forçada por `LEFT JOIN` e `STRAIGHT JOIN` irá ajudar o otimizador de joins (que calcula em qual ordem as tabelas devem ser unidas) a fazer seu trabalho mais rapidamente, já que haverão poucas permutações de tabelas a serem conferidas.

Perceba que o texto acima significa que se você fizer uma consulta do tipo:

```
SELECT * FROM a,b LEFT JOIN c ON (c.key=a.key) LEFT JOIN d (d.key=a.key) WHERE b.ke
```

O MySQL irá fazer uma pesquisa completa em `b` já que o `LEFT JOIN` irá força-lo a ser lido antes de `d`.

A correção neste caso é alterar a consulta para:

```
SELECT * FROM b,a LEFT JOIN c ON (c.key=a.key) LEFT JOIN d (d.key=a.key) WHERE b.ke
```

5.2.7 Como o MySQL Otimiza Cláusulas LIMIT

Em alguns casos o MySQL irá tratar a consulta de maneira diferente quando você estiver utilizando `LIMIT #` e não estiver utilizando `HAVING`:

- Se você estiver selecionando apenas alguns registros com `LIMIT`, o MySQL usará índices em alguns casos quando ele normalmente preferiria fazer uma varredura completa na tabela.
- Se você utilizar `LIMIT #` com `ORDER BY`, O MySQL irá terminar a ordenação logo que ele encontrar os primeiros `#` registros em vez de ordenar a tabela inteira.
- Ao combinar `LIMIT #` com `DISTINCT`, o MySQL irá parar logo que ele encontrar `#` registros únicos.
- Em alguns casos um `GROUP BY` pode ser resolvido lendo a chave em ordem (ou fazer uma classificação na chave) e então calcular resumos até o valor da chave alterar. Neste caso, `LIMIT #` não irá calcular nenhum `GROUP BY` desnecessário.
- Logo que o MySQL enviar os primeiros `#` registros para o cliente, ele irá abortar a consulta.
- `LIMIT 0` irá sempre retornar rapidamente um conjunto vazio. Isto é útil para conferir a consulta e obter os tipos de campos do resultado.
- O tamanho das tabelas temporárias usa `LIMIT #` para calcular quanto de espaço é necessário para resolver a consulta.

5.2.8 Performance das Consultas que Utilizam INSERT

O tempo para inserir um registro consiste aproximadamente de:

- Conexão: (3)
- Enviar a consulta para o servidor: (2)

- Analisar a consulta (2)
- Inserir o registro: (1 x size of record)
- Inserir os índices: (1 x number of indexes)
- Fechar: (1)

onde os números são de certa forma proporcionais ao tempo total. Isto não leva em consideração o sobrecarga inicial para abrir tabelas (que é feita uma vez para cada consulta concorrente em execução).

O tamanho da tabela diminuem a velocidade da inserção de índices em $N \log N$ (Árvores B).

Algumas maneiras de acelerar as inserções:

- Se você estiver inserindo vários registros do mesmo cliente ao mesmo tempo, utilize instruções `INSERT` com listas de múltiplos valores. Isto é muito mais rápido (muitas vezes em alguns casos) do que utilizar instruções `INSERT` separadas.
- Se você inserir vários registros de diferentes clientes, você pode obter velocidades mais altas utilizando a instrução `INSERT DELAYED`. See [\[INSERT\]](#), page [\[undefined\]](#).
- Perceba que com `MyISAM` você pode inserir registros ao mesmo tempo que `SELECTs` estejam executando se não existirem registros apagados nas tabelas.
- Ao carregar uma tabela de um arquivo texto, utilize `LOAD DATA INFILE`. Isto é normalmente 20 vezes mais rápido do que utilizar várias instruções `INSERT` See [\[LOAD DATA\]](#), page [\[undefined\]](#).
- É possível com algum trabalho extra fazer o `LOAD DATA INFILE` executar ainda mais rápido quando a tabela tiver vários índices. Utilize o seguinte procedimento:
 1. Opcionalmente crie a tabela com `CREATE TABLE`. Por exemplo, utilizando `mysql` ou `Perl-DBI`.
 2. Execute a instrução `FLUSH TABLES` ou o comando shell `mysqladmin flush-tables`.
 3. Utilize `myisamchk --keys-used=0 -rq /path/to/db/nome_tabela`. Isto removerá o uso de todos os índices da tabela.
 4. Insira dados na tabela com `LOAD DATA INFILE`. Isto não atualizará índices e será muito mais rápido.
 5. Se no futuro você precisar da tabela somente para leitura, execute `myisampack` na mesma para torná-la menor. See [\[Formato compactado\]](#), page [\[undefined\]](#).
 6. Recrie os índices com `myisamchk -r -q /caminho/para/bd/nome_tabela`. Isto criará a árvore de índices em memória antes de escrevê-la para o disco, que é muito mais rápido porque evita que seja feita muita busca disco. A árvore de índices resultante é também balanceada perfeitamente.
 7. Execute uma instrução `FLUSH TABLES` ou o comando shell `mysqladmin flush-tables`.

Este procedimento irá ser construído dentro de `LOAD DATA INFILE` em alguma versão futura do MySQL.

- Você pode acelerar inserções bloqueando suas tabelas:

```
mysql> LOCK TABLES a WRITE;
mysql> INSERT INTO a VALUES (1,23),(2,34),(4,33);
mysql> INSERT INTO a VALUES (8,26),(6,29);
mysql> UNLOCK TABLES;
```

A principal diferença na velocidade é que o buffer de índices é descarregado no disco somente uma vez, depois de todas instruções `INSERT` terem sido completadas. Normalmente existiria tantas descargas do buffer de índices quanto instruções `INSERT` diferentes. O bloqueio não é necessário se você pode inserir todos registros com uma simples instrução.

O bloqueio irá também diminuir o tempo total de testes de multi-conexões, mas o tempo máximo de espera para algumas threads irá aumentar (porque eles esperam pelos bloqueios). Por exemplo:

```
thread 1 faz 1000 inserções
thread 2, 3 e 4 faz 1 inserção
thread 5 faz 1000 inserções
```

Se você não estiver usando travas, 2, 3 e 4 irão terminar antes de 1 e 5, Se estiver utilizando travas, 2, 3 e 4 provavelmente não irão terminar antes de 1 ou 5, mas o tempo total deve ser cerca de 40% mais rápido.

Como as operações `INSERT`, `UPDATE` e `DELETE` são muito rápidas no MySQL, você obterá melhor performance geral adicionando travas em tudo que fizer mais que cerca de 5 inserções ou atualizações em um registro. Se você fizer várias inserções em um registro, você pode utilizar `LOCK TABLES` seguido de um `UNLOCK TABLES` de vez em quando (em torno de 1000 registro) para permitir que outras threads acessem a tabela. Isto também continua mostrando um bom ganho de performance.

Com certeza, `LOAD DATA INFILE` é muito mais rápido para carregar dados.

Para obter mais velocidade para `LOAD DATA INFILE` e `INSERT`, aumente o tamanho do buffer de chaves. See [\[Parâmetros de servidor\]](#), page [\[Parâmetros de servidor\]](#).

5.2.9 Performance das Consultas que Utilizam `UPDATE`

Consultas de atualização são otimizadas como uma consulta que usa `SELECT` com a sobrecarga adicional de escrita. A velocidade da escrita depende do tamanho dos dados e do número de índices que serão atualizados. Índices que não forem alterados não serão atualizados.

Outra forma para obter atualizações rápidas é atrasar as atualizações e então fazer várias atualizações em um registro posteriormente. Fazer várias atualizações em um registro é muito mais rápido do que fazer uma por vez se você travar a tabela.

Perceba que, com formato de registros dinâmicos, atualizar um registro para um valor maior que o tamanho total pode dividir o registro. Portanto, se você faz isso frequentemente, é muito importante usar `OPTIMIZE TABLE` de vez em quando. See [\[OPTIMIZE TABLE\]](#), page [\[OPTIMIZE TABLE\]](#).

5.2.10 Performance das Consultas que Utilizam `DELETE`

Se você deseja apagar todos os registros em uma tabela, deve usar `TRUNCATE TABLE nome_tabela`. See [\[TRUNCATE\]](#), page [\[TRUNCATE\]](#).

O tempo para apagar um registro é exatamente proporcional ao número de índices. Para apagar registros mais rapidamente, você pode aumentar o tamanho do cache de índices. See [\[Parâmetros do servidor\]](#), page [\[Parâmetros do servidor\]](#).

5.2.11 Mais Dicas sobre Otimizações.

Dicas não ordenadas para sistemas rápidos:

- Utilize conexões persistentes aos banco de dados para evitar a sobrecarga da conexão. Se você não poder utilizar conexões persistentes e for fazer várias novas conexões para o banco de dados, você pode desejar alterar o valor da variável `thread_cache_size`. See [\[Parâmetros do servidor\]](#), page [\[Parâmetros do servidor\]](#).
- Sempre verifique se todas as suas consultas realmente utilizam os índices que foram criados nas tabelas. No MySQL você pode fazer isto com o comando `EXPLAIN`. See [\[Explain\]](#), page [\[Explain\]](#).
- Tente evitar consultas `SELECT` complexas em tabelas que são muito atualizadas. Isto evita problemas com travamento de tabelas.
- As novas tabelas `MyISAM` permitem inserir registros em uma tabela sem registros apagados ao mesmo tempo que outra tabela a estiver lendo. Se este recurso é importante para você, deve considerar métodos onde você não tem que apagar registrou ou executar `OPTIMIZE TABLE` depois de ter apagado vários registros.
- Utilize `ALTER TABLE ... ORDER BY expr1,expr2...` se você na maioria das vezes recupera registros na ordem `expr1,expr2...`. Utilizando esta opção depois de grandes alterações para a tabela, pode lhe dar um ganho de performance.
- Em alguns casos pode fazer sentido introduzir uma coluna 'hash' baseada nas informações das outras colunas. Se esta coluna for curta e razoavelmente única pode ser muito mais rápido do que ter um grande índice em várias colunas. No MySQL é muito fácil usar esta coluna extra: `SELECT * FROM nome_tabela WHERE hash=MD5(concat(col1,col2)) AND col_1='constante' AND col_2='constante'`
- Para tabelas que alteram muito você deve tentar evitar todas colunas `VARCHAR` ou `BLOB`. Você terá tamanho de registro dinâmico assim que usar um simples campo `VARCHAR` ou `BLOB`. See [\[Tipos de tabelas\]](#), page [\[Tipos de tabelas\]](#).
- Normalmente não é muito útil cortar uma tabela em diferentes tabelas apenas porque os registros estão 'grandes'. Para acessar um registro, o maior problema para a performance é a busca em disco para encontra o primeiro byte do registro. Depois de encontrar os dados a maioria dos novos discos podem ler o registro inteiro rápido o bastante para a maioria das aplicações. Os únicos caos onde realmente faz sentido dividir uma tabela é se ela é uma tabela de registros com tamanho dinâmico (veja acima) que você pode alterar para um tamanho fixo, ou se você frequentemente precisa examinar a tabela e não precisa da maioria das colunas. See [\[Tipos de tabela\]](#), page [\[Tipos de tabela\]](#).
- Se frequentemente você precisar calcular alguma coisa baseada em informação de vários registros (ex: contagem de registros), provavlmnte é melhor introduzir uma nova tabela e atualizar o contador em tempo real. Uma atualização do tipo `UPDATE table set count=count+1 where index_column=constante` é muito rapida!

Isto é realmente importante quando você usa bancos de dados como o MySQL que só tem travamento de tabelas (múltiplas leituras/escrita única). Isto também dará melhor performance com a maioria dos bancos de dados, já que o gerenciador de bloqueio de registro terá menos a fazer neste caso.

- Se você precisar colterar estatísticas de tabelas maiores, utilize tabelas resumo em vez de buscar em toda a tabela. Manter os resumos deve ser mais rápido que tentar criar estatísticas instantaneamente. É muito mais rápido criar novas tabelas através dos logs quando as coisas mudam (dependendo das decisões de negócio) que ter que alterar a aplicação em execução.
- Se possível, deve-se classificar relatórios como 'instantâneo' ou 'estatísticos' onde os dados necessários para relatórios estatísticos são gerados apenas com base nas tabelas resumo que são geradas a partir dos dados atuais.
- Tire vantagem do fato de que a coluna tem valores padrões. Insira valores explicitamente apenas quando os valores a serem inseridos diferem do padrão. Isto reduz a análise que o MySQL precisa fazer e aumenta a velocidade de inserção.
- Em alguns casos é conveniente empacotar e armazenar os dados em um campo blob. Neste caso você deve adicionar algum código em sua aplicação para empacotar/desempacotar as coisas no campo blob, mas isto pode poupar vários acessos a algum estágio. Isto é prático quando você possui dados que não conformam com uma estrutura estática de tabela.
- Normalmente, você deve tentar manter todos dados não-redundantes (o que é chamado de 3ª forma normal na teoria de bancos de dados), mas você não deve ter medo de duplicar alguns itens ou criar tabelas de resumo se você precisar delas para ganhar mais velocidade.
- Stored Procedures ou UDF (funções definidas pelo usuário) pode ser uma boa forma para obter mais performance. Neste caso você deve, entretanto, sempre ter uma maneira de fazer isso de outra maneira (mais lenta) se você utilizar algum banco de dados que não suporta isto.
- Você pode sempre ganhar velocidade fazendo cache de perguntas/respostas na sua aplicação e tentando fazer várias inserções/atualizações ao mesmo tempo. Se seu banco de dados suporta travamento de tabelas (como o MySQL e Oracle), isto deve ajudar a garantir que o cache de índices é descarregado somente uma vez depois de todas atualizações.
- Use `INSERT /*! DELAYED */` quando não precisar saber quando os dados são gravados. Isto melhora a velocidade porque vários registros podem ser gravados com uma simples escrita em disco.
- Use `INSERT /*! LOW_PRIORITY */` quando você desejar que suas consultas sejam mais importantes.
- Use `SELECT /*! HIGH_PRIORITY */` para obter consultas que ignoram a fila. Isto é, a consulta é feita mesmo se alguém estiver esperando para fazer uma escrita.
- Use a instrução `INSERT` multi-linhas para armazenar vários registros com um comando SQL (vários servidores SQL suportam isto).
- Use `LOAD DATA INFILE` para carregar volumes maiores de dados. Isto é mais rápido que as inserções normais e mais rápido até quando o `myisamchk` for integrado no `mysqld`.

- Use colunas `AUTO_INCREMENT` para garantir valores únicos.
- Use `OPTIMIZE TABLE` de vez em quando para evitar fragmentação quando estiver usando formatos de tabela dinâmica. See [\[OPTIMIZE TABLE\]](#), page [\[Tipos de tabelas\]](#), page [\[Tipos de tabelas\]](#).
- Use tabelas `HEAP` para obter mais velocidade sempre que possível. See [\[Tipos de tabelas\]](#), page [\[Tipos de tabelas\]](#).
- Quando estiver usando uma configuração de servidor Web normal, imagens devem ser armazenadas como arquivos. Isto é, armazene apenas uma referência para o arquivo no banco de dados. A principal razão para isto é que um servidor Web normal é muito melhor trabalhando com cache de arquivos do que com conteúdo de banco de dados. Portanto será muito mais fácil obter um sistema rápido se você utilizar arquivos.
- Use tabelas em memória para dados não-críticos que são acessados frequentemente (como informações sobre o último banner visto para usuários que não possuem cookies).
- Colunas com informações idênticas em diferentes tabelas devem ser declaradas idênticas e ter nomes idênticos. No entanto, antes da versão 3.23, você pode obter ligações mais lentas.

Tente manter os nomes mais simples (use `nome` em vez de `nome_cliente` na tabela cliente). Para deixar seus nomes portáteis para outros servidores SQL você deve mantê-los menores que 18 caracteres.

- Se você realmente precisa de alta velocidade, você deve verificar as interfaces de baixo nível para armazenagem de dados que os diferentes servidores SQL suportam! Por exemplo, para acessar tabelas MySQL `MyISAM` diretamente, você pode obter um aumento de velocidade de 2-5 vezes comparado ao uso da interface SQL. Para conseguir essa façanha, os dados devem estar no mesmo servidor que sua aplicação, e normalmente devem ser acessados por apenas um processo (porque travamento de arquivos externo são muito lentos). Os problemas acima podem ser eliminados introduzindo comandos `MyISAM` de baixo nível no servidor MySQL (isto pode ser a maneira mais fácil para aumentar a performance). Tenha cuidado em projetar a interface com o banco de dados, ela deve ser bem fácil para suportar estes tipos de otimizações.
- Em vários casos é mais rápido acessar dados de um banco de dados (utilizando uma conexão ativa) do que acessar um arquivo texto, apenas pelo fato do banco de dados ser mais compacto do que o arquivo texto (se você estiver utilizando dados numéricos), e isto irá envolver menos acessos à disco. Você também irá poupar código porque não será necessário analisar seus arquivos texto para encontrar limites de registros e campos.
- Você pode também usar replicação para conseguir ainda mais performance nas suas aplicações. See [\[Replicação\]](#), page [\[Replicação\]](#).
- Declarando uma tabela com `DELAY_KEY_WRITE=1` irá tornar a atualização de índices mais rápida, pois as mesmas não serão escritas em disco até o arquivo ser fechado. O lado ruim é que você deve executar `myisamchk` nestas tabelas antes de iniciar o `mysqld` para garantir que os dados estão corretos se o `mysqld` for finalizado no meio da execução. Como a informação de chave pode sempre ser gerada a partir dos dados, você não deve perder nada usando `DELAY_KEY_WRITE`.

5.3 Detalhes sobre Bloqueios

5.3.1 Como o MySQL Bloqueia as Tabelas

Você pode encontrar uma discussão sobre diferentes métodos de bloqueios no apêndice. See [\[Métodos de bloqueio\]](#), page [\[Métodos de bloqueio\]](#).

Todos os bloqueios no MySQL são livres de deadlock. Isto é gerenciado requisitando todos os bloqueios necessários de uma vez no começo de uma consulta e sempre bloqueando as tabelas na mesma ordem.

O método de bloqueio que o MySQL utiliza para **ESCRITA** funciona da seguinte forma:

- Se não existirem travas na tabela, coloque uma bloqueio de escrita na mesma.
- Caso contrário, coloca a requisição de trava na fila de bloqueios para escrita.

O método de bloqueio que o MySQL utilizado para **LEITURA** funciona da seguinte maneira:

- Se não existirem tarvas na tabela, coloca um bloqueio de leitura na mesma.
- Caso contrário, coloca a requisição de trava na fila de bloqueios para leitura.

Quando um bloqueio é liberado, a trava fica disponível para as threads na fila de bloqueios de escrita, e então para as threads na fila de bloqueios de leitura.

Isto significa que se você possui várias atualizações em uma tabela, instruções **SELECT** irão esperar até que não existam mais atualizações.

Para contornar este problema no caso onde você precisa fazer várias operações de **INSERT** e **SELECT** em uma tabela, você pode inserir registros em uma tabela temporária e atualizar a tabela real com os registros da tabela temporária de uma só vez.

Isto pode ser feito usando o código a seguir:

```
mysql> LOCK TABLES tabela_real WRITE, tabela_insercao WRITE;
mysql> insert into tabela_real select * from tabela_insercao;
mysql> TRUNCATE TABLE tabela_insercao;
mysql> UNLOCK TABLES;
```

Você pode utilizar as opções **LOW_PRIORITY** com **INSERT**, **UPDATE** ou **DELETE** ou **HIGH_PRIORITY** com **SELECT** se você desejar priorizar a recuperação em alguns casos específicos. Também podei-se iniciar o `mysqld` com `--low-priority-updates` para obter o mesmo comportamento.

Utilizar **SQL_BUFFER_RESULT** pode também tornar a criação de locks de tabelas mais curtos. See [\[SELECT\]](#), page [\[SELECT\]](#).

Você também pode alterar o código de bloqueios no `'mysys/thr_lock.c'` para usar uma fila simples. Neste caso, bloqueios de escrita e leitura devem ter a mesma prioridade, o que pode ajudar em algumas aplicações.

5.3.2 Detalhes de Bloqueios de Tabelas

O código de bloqueio de tabelas no MySQL é livre de deadlock.

O MySQL utiliza bloqueio de tabelas (no lugar de bloqueio de registros ou colnas) em todos os tipos de tabelas, exceto tabelas BDB, para obter uma alta velocidade nos bloqueios. Para grandes tabelas, bloqueio de tabelas é **MUITO** melhor que bloqueio de registros para a maioria das aplicações, mas existem, é claro, algumas desvantagens.

Para tabelas BDB e InnoDB, O MySQL só utiliza bloqueio de tabelas se você bloquear explicitamente a tabela com **LOCK TABLES** ou executar um comando quer irá modificar todos

os registros na tabela, como `ALTER TABLE`. Para estes tipos de tabelas nós recomendamos a você não utilizar `LOCK TABLES`.

No MySQL versão 3.23.7 ou superior, você pode inserir registros em tabelas MyISAM ao mesmo tempo que outras threads estão lendo da mesma tabela. Perceba que atualmente isto funciona somente se não existirem buracos depois de registros apagados na tabela no momento que a inserção é feita. Quando todos os buracos forem preenchidos com novos dados, inserções concorrentes irão automaticamente ser habilitadas novamente.

O bloqueio de tabelas habilita várias threads para lerem de uma tabela ao mesmo tempo, mas se uma thread desejar escrever a uma tabela, ela primeiramente deve obter acesso exclusivo. Durante a atualização, todas outras threads que desejarem acessar esta tabela em particular irão esperar até que a atualização acabe.

Como atualizações em tabelas normalmente são consideradas mais importantes que `SELECT`, todas as instruções que atualizam uma tabela tem maior prioridade que instruções que simplesmente recuperam informações. Isto deve garantir que atualizações não fiquem na fila por terem sido passadas várias consultas pesadas em uma tabela específica. (Você pode alterar isto utilizando `LOW_PRIORITY` com a instrução que faz a atualização ou `HIGH_PRIORITY` com a instrução `SELECT`.)

A partir do MySQL versão 3.23.7 pode-se utilizar a variável `max_write_lock_count` para forçar o MySQL a fornecer temporariamente a todas as instruções `SELECT`, que esperam por uma tabela, uma prioridade mais alta depois de um número específico de inserções em uma tabela.

O bloqueio de tabela não é, no entanto, muito bom sobre os seguintes cenários:

- Um cliente emite uma `SELECT` que exige muito tempo para ser executada.
- Outro cliente então executa um `UPDATE` na tabela usada. Este cliente terá que esperar até que a `SELECT` seja terminada.
- Outro cliente executa outra instrução `SELECT` na mesma tabela. Como `UPDATE` tem maior prioridade que `SELECT`, esta `SELECT` irá esperar pelo término da `UPDATE`. Ela também irá esperar pelo término da primeira `SELECT`!
- Uma thread está esperando por algo do tipo `disco cheio`, caso em que todas as threads que desejam acessar a tabela com problema irão ser colocadas em estado de espera até que mais espaço em disco seja disponível.

Algumas soluções possíveis para este problema são:

- Tente deixar suas instruções `SELECT` sempre rápidas. Você pode ter que criar algumas tabelas de resumo para fazer isto.
- Inicie o `mysqld` com `--low-priority-updates`. Isto irá fornecer a todas instruções que atualizam (modificam) uma tabela prioridade menor que uma instrução `SELECT`. Neste caso a última instrução `SELECT` no cenário anterior deveria executar antes da instrução `INSERT`.
- Você pode fornecer a uma instrução `INSERT`, `UPDATE` ou `DELETE` específica menor prioridade com o atributo `LOW_PRIORITY`.
- Inicie o `mysqld` com um valor baixo para `max_write_lock_count` para fornecer bloqueios de `LEITURA` depois de um certo número de bloqueios de `ESCRITA`.

- Você pode especificar que todas as atualizações de uma thread específica deve ser feita utilizando prioridade baixa com o comando SQL: `SET SQL_LOW_PRIORITY_UPDATES=1`. See [\(undefined\) \[SET OPTION\]](#), page [\(undefined\)](#).
- Você pode especificar que uma `SELECT` específica é muito importante com o atributo `HIGH_PRIORITY`. See [\(undefined\) \[SELECT\]](#), page [\(undefined\)](#).
- Se você tiver problemas com `INSERT` combinado com `SELECT`, utilize as novas tabelas `MyISAM`, pois elas suportam `SELECTs` e `INSERTs` concorrentes.
- Se você utiliza principalmente instruções `INSERT` e `SELECT` misturadas, o atributo `DELAYED` no `INSERT` provavelmente irá resolver seus problemas. See [\(undefined\) \[INSERT\]](#), page [\(undefined\)](#).
- Se você tiver problemas com `SELECT` e `DELETE`, a opção `LIMIT` para `DELETE` pode ajudar. See [\(undefined\) \[DELETE\]](#), page [\(undefined\)](#).

5.4 Otimizando a Estrutura do Banco de Dados

5.4.1 Opções do Projeto

O MySQL mantém dados de registros e índices em arquivos separados. Vários (quase todos) bancos de dados misturam dados de registros e índice no mesmo arquivo. Nós acreditamos que a escolha do MySQL é melhor para uma ampla escala de sistemas modernos.

Outra forma de armazenar os dados de registros é manter a informação para cada coluna em uma área separada (exemplos são o `SDBM` e o `Focus`). Isto irá causar um ponto de performance para toda consulta que acessar mais de uma coluna. Como isto degrada rapidamente quando mais de uma coluna é acessada, acreditamos que este modelo não é bom para propósitos gerais de bancos de dados.

O caso mais comum é aquele em que o índice e dados são armazenados juntos (como no `Oracle/Sybase`). Neste caso você irá encontrar a informação do registro na folha da página de índice. A coisa boa com este layout é que ele, em vários casos, dependendo de como o índice é armazenado no cache, salva uma leitura de disco. As desvantagens deste layout são:

- A varredura da tabela é muito mais lenta porque você tem que ler os índices para encontrar os dados.
- Não podem ser usados apenas a tabela de índices para recuperar dados para uma consulta.
- Você perde muito espaço de armazenagem, já que os índices devem ser duplicados nos nós (pois os registros não podem ser armazenados nos nós).
- Deleções irão degenerar a tabela depois de um tempo (já que os índices nos nós normalmente não são atualizados na deleção).
- É mais difícil fazer o cache somente dos dados de índices.

5.4.2 Deixando os Dados com o Menor Tamanho Possível

Uma das otimizações mais básicas é tentar manter seus dados (e índices) utilizando o menor espaço possível no disco (e em memória). Isto pode fornecer grandes melhorias porque a

leitura de disco é mais rápida e normalmente menos memória principal será usada. A indexação também exige menos recursos se for feita em colunas menores.

O MySQL suporta vários diferentes tipos de tabelas e formatos de registros. Você pode ter um ótimo ganho de performance escolhendo o formato certo de tabela a ser usada. See [\[Tipos de tabelas\]](#), page [\[Tipos de tabelas\]](#).

Pode-se obter melhor performance em uma tabela e minimizar espaço de armazenagem utilizando as técnicas listadas abaixo:

- Utilize os tipos mais eficientes (menores) sempre que possível. O MySQL tem vários tipos especializados que economizam espaço em disco e memória.
- Utilize tipos inteiros menores se possível para obter tabelas menores. Por exemplo, MEDIUMINT normalmente é melhor que INT.
- Declare colunas para serem NOT NULL se possível. Isto deixa tudo mais rápido e você economiza um bit por coluna. Perceba que se você realmente precisa de NULL nas suas aplicações, podem ser usados. Tente simplesmente não usá-la em todas as colunas por padrão.
- Se você não possui nenhuma coluna de tamanho variável (VARCHAR, TEXT ou BLOB), um formato de registro de tamanho fixo para é utilizado. Isto é mais rápido mas infelizmente pode ocupar mais espaço. See [\[Formatos de tabelas MyISAM\]](#), page [\[Formatos de tabelas MyISAM\]](#).
- O índice primário de uma tabela deve ser o mais curto possível. Isto torna a identificação de um registro fácil e eficiente.
- Para cada tabela, você deve decidir qual método de armazenamento/índice utilizar. See [\[Tipos de tabelas\]](#), page [\[Tipos de tabelas\]](#).
- Crie somente os índices necessários. Índices são bons para recuperação mas ruins quando você precisa armazenar os dados rapidamente. Se na maioria das vezes você acessa uma tabela pesquisando em uma combinação de colunas, crie um índice para elas. A primeira parte do índice deve ser a coluna mais utilizada. Se você SEMPRE utiliza várias colunas, deve usar a coluna com mais duplicações em primeiro lugar para obter melhor compactação do índice.
- Se for melhor que uma coluna tenha um prefixo único nos primeiros caracteres, é melhor indexar somente este prefixo. O MySQL suporta um índice em uma parte de uma coluna de caracteres. Índices menores são mais rápidos não somente porque eles exigem menos espaço em disco mas também porque eles irão fornecer a você mais acerto no cache de índice e isto diminui acessos a disco. See [\[Parâmetros de servidor\]](#), page [\[Parâmetros de servidor\]](#).
- Em algumas circunstâncias pode ser benéfico dividir uma tabela que é varrida frequentemente em duas. Isto é verdade especificamente se a tabela tiver um formato dinâmico e for possível utilizar um formato de tabela estático que possa ser usada para encontrar os registros relevantes quando se fizer uma varredura da tabela.

5.4.3 Como o MySQL Utiliza Índices

Os índices são utilizados para encontrar registros com um valor específico de uma coluna rapidamente. Sem um índice o MySQL tem de iniciar com o primeiro registro e depois ler através de toda a tabela até que ele encontre os registros relevantes. Quanto maior a tabela,

maior será o custo. Se a tabela possui um índice para as colunas em questão, o MySQL pode rapidamente obter uma posição para procurar no meio do arquivo de dados sem ter que varrer todos os registros. Se uma tabela possui 1000 registros, isto é pelo menos 100 vezes mais rápido do que ler todos os registros sequencialmente. Note que se você precisar acessar quase todos os 1000 registros, seria mais rápido acessá-los sequencialmente porque evitaria acessos ao disco.

Todos os índices do MySQL (PRIMARY, UNIQUE e INDEX) são armazenados em árvores B. Strings são automaticamente compactadas nos espaços finais e prefixados. See [\[CREATE INDEX\]](#), page [\[undefined\]](#).

Índices são utilizados para:

- Encontrar rapidamente os registros que coincidam com uma cláusula WHERE.
- Recuperar registros de outras tabelas ao realizar joins.
- Encontrar o valor MAX() ou MIN() para uma coluna indexada específica. Isto é otimizado por um preprocessador que confere se você está utilizando WHERE key_part_#=constante em todas as partes da chave < N. Neste caso o MySQL irá fazer uma simples procura na chave e trocar a expressão MIN() com uma constante. Se todas as expressões forem trocadas por constantes, a consulta retornará imediatamente:

```
SELECT MIN(key_part2),MAX(key_part2) FROM nome_tabela where key_part1=10
```

- Ordenar ou agrupar uma tabela se a ordenação ou agrupamento for feito em um prefixo mais à esquerda de uma chave útil (por exemplo, ORDER BY key_part_1, key_part_2). A chave é lida na ordem invertida se todas as partes da chave forem seguidas por DESC.

O índice também pode ser utilizado mesmo se ORDER BY não coincidir exatamente com o índice, já que todas as partes não utilizadas do índice e todos os extras que são colunas ORDER BY são constantes na cláusula WHERE A consulta seguinte utilizará o índice para resolver a parte ORDER BY:

```
SELECT * FROM foo ORDER BY key_part1,key_part2,key_part3;
SELECT * FROM foo WHERE coluna=constante ORDER BY coluna, key_part1;
SELECT * FROM foo WHERE key_part1=const GROUP BY key_part2;
```

- Em alguns casos uma consulta pode ser otimizada para recuperar valores sem consultar o arquivo de dados. Se todas colunas utilizadas para alguma tabela são numéricas e formam um prefixo mais à esquerda para alguma chave, os valores podem ser recuperados da árvore de índices para aumentar a velocidade:

```
SELECT key_part3 FROM nome_tabela WHERE key_part1=1
```

Suponha que você utilize a seguinte instrução SELECT:

```
mysql> SELECT * FROM nome_tabela WHERE col1=val1 AND col2=val2;
```

Se um índice de colunas múltiplas existir em col1 e col2, os registros apropriados podem ser recuperados diretamente. Se índices separados de únicas colunas existirem em col1 e col2, o otimizador tentará encontrar o índice mais restritivo decidindo qual índice irá encontrar menos registros e usará este índice para recuperar os registros.

Se a tabela possuir um índice de múltiplas colunas, qualquer prefixo mais à esquerda do índice pode ser usado pelo otimizador para encontrar registros. Por exemplo, se você possui um índice de três colunas em (col1,col2,col3), você tem capacidades de busca indexada em (col1), (col1,col2) e (col1,col2,col3).

O MySQL não pode utilizar um índice parcial se as colunas não formarem um prefixo mais à esquerda do índice. Suponha que você tenha as instruções SELECT mostradas abaixo:

```
mysql> SELECT * FROM nome_tabela WHERE col1=val1;
mysql> SELECT * FROM nome_tabela WHERE col2=val2;
mysql> SELECT * FROM nome_tabela WHERE col2=val2 AND col3=val3;
```

Se um índice existir em (col1,col2,col3), somente a primeira consulta exibida acima utiliza o índice. A segunda e terceira consultas envolvem colunas indexadas, mas (col2) e (col2,col3) não são os prefixos mais à esquerda de (col1,col2,col3).

O MySQL também utiliza índices para comparações do tipo LIKE se o argumento para LIKE for uma string constante que não inicie com um meta caracter. Por exemplo as seguintes instruções SELECT utilizam índices:

```
mysql> select * from nome_tabela where key_col LIKE "Patrick%";
mysql> select * from nome_tabela where key_col LIKE "Pat%ck%";
```

Na primeira instrução, somente os registros com "Patrick" <= key_col < "Patricl" são considerados. Na segunda instrução, somente registros com "Pat" <= key_col < "Pau" são considerados.

As seguintes instruções SELECT não usarão índices:

```
mysql> select * from nome_tabela where key_col LIKE "%Patrick%";
mysql> select * from nome_tabela where key_col LIKE other_col;
```

Na primeira instrução, o valor LIKE inicia com um meta caracter. Na segunda instrução, o valor LIKE não é uma constante.

Pesquisas com nome_coluna IS NULL usarão índices se nome_coluna for um índice.

O MySQL normalmente utiliza o índice que encontra o menor número de registros. Um índice é usado para colunas que você compara com os seguintes operadores: =, >, >=, <, <=, BETWEEN e um LIKE com um prefixo sem meta caracteres como 'algo%'.

Qualquer índice que não cobrem todos os níveis de AND na cláusula WHERE não é utilizado para otimizar a consulta. Em outras palavras: Para poder usar um índice, um prefixo do índice deve ser utilizado em todo agrupamento AND.

A seguinte cláusula WHERE utilizará índices:

```
... WHERE index_part1=1 AND index_part2=2 AND other_column=3
... WHERE index=1 OR A=10 AND index=2      /* index = 1 OR index = 2 */
... WHERE index_part1='hello' AND index_part_3=5
      /* otimizado como "index_part1='hello'" */
... WHERE index1=1 and index2=2 or index1=3 and index3=3;
      /* pode utilizar índice em index1 mas não em index2 ou index 3 */
```

Estas cláusulas WHERE **NÃO** utilizam índices:

```
... WHERE index_part2=1 AND index_part3=2 /* index_part_1 não é usado */
... WHERE index=1 OR A=10                /* O índice não é usado em ambas as partes AND*/
... WHERE index_part1=1 OR index_part2=10 /* Nenhum indice cobre os registros */
```

Perceba que em alguns casos o MySQL não utilizará um índice, mesmo se algum estiver disponível. Alguns dos casos onde isto acontece:

- Se o uso do índice necessita que o MySQL acesse mais de 30% dos registros na tabela. (Neste caso uma varredura da tabela é provavelmente mais rápido, já que isto necessite

de menos pesquisas em discos). Perceba que se uma consulta utiliza `LIMIT` para recuperar somente parte dos registros, o MySQL irá utilizar um índice de qualquer forma, pois assim pode encontrar os poucos registros mais rapidamente e retornar o resultado.

5.4.4 Índices de Colunas

Todos os tipos de colunas do MySQL podem ser indexadas. O uso de índices nas colunas relevantes é a melhor forma de melhorar a performance de operações `SELECT`.

O número máximo de chaves e o tamanho máximo de um índice é definido pelo manipulador de tabelas. See [\[Tipos de tabelas\]](#), page [\[Tipos de tabelas\]](#). Você pode com todos os manipuladores de tabelas ter pelo menos 16 chaves e um índice de tamanho total de pelo menos 256 bytes.

Para colunas `CHAR` e `VARCHAR` você pode indexar um prefixo da coluna. Isto é muito mais rápido e necessita de menos espaço em disco do que indexar a coluna inteira. A sintaxe para utilizar na instrução `CREATE TABLE` para indexar um prefixo de uma coluna se parece com o exemplo a seguir:

```
KEY nome_indice (nome_campo(tamanho))
```

O exemplo abaixo cria um índice para os primeiros 10 caracteres da coluna `nome`:

```
mysql> CREATE TABLE teste (  
    nome CHAR(200) NOT NULL,  
    KEY nome_indice (nome(10)));
```

Para colunas `BLOB` e `TEXT`, você deve indexar um prefixo da coluna. Você não pode indexar a coluna inteira.

No MySQL Versão 3.23.23 ou posterior, você pode também criar índices **FULLTEXT** especiais. Eles são utilizados para pesquisas textuais. Somente o tipo de tabela `MyISAM` suporta índices **FULLTEXT**. Eles podem ser criados somente de colunas `VARCHAR` e `TEXT`. Indexação sempre acontece sobre toda a coluna e indexação parcial não é suportada. Veja [\[Fulltext Search\]](#), page [\[Fulltext Search\]](#) para detalhes.

5.4.5 Índices de Múltiplas Colunas

O MySQL pode criar índices em múltiplas colunas. Um índice pode consistir de até 15 colunas. (Em colunas `CHAR` e `VARCHAR` você também pode utilizar um prefixo da coluna como parte de um índice).

Um índice de múltiplas colunas pode ser considerado um array ordenado contendo valores que são criados concatenando valores de colunas indexadas.

O MySQL utiliza índices de múltiplas colunas de forma que consultas são rápidas quando você especifica uma quantidade conhecida para a primeira coluna do índice em uma cláusula `WHERE`, mesmo se você não especificar valores para as outras colunas.

Suponha que uma tabela é criada com as seguintes especificações:

```
mysql> CREATE TABLE teste (  
    id INT NOT NULL,  
    ultimo_nome CHAR(30) NOT NULL,  
    primeiro_nome CHAR(30) NOT NULL,  
    PRIMARY KEY (id),
```

```
INDEX nome (ultimo_nome,primeiro_nome));
```

Então o índice `nome` é um índice com `ultimo_nome` e `primeiro_nome`. O índice será usado para consultas que especificarem valores em um limite conhecido para `ultimo_nome`, ou para ambos `ultimo_nome` e `primeiro_nome`. Desta forma, o índice `nome` será usado nas seguintes consultas:

```
mysql> SELECT * FROM teste WHERE ultimo_nome="Widenius";
```

```
mysql> SELECT * FROM teste WHERE ultimo_nome="Widenius"
      AND primeiro_nome="Michael";
```

```
mysql> SELECT * FROM teste WHERE ultimo_nome="Widenius"
      AND (primeiro_nome="Michael" OR primeiro_nome="Monty");
```

```
mysql> SELECT * FROM teste WHERE ultimo_nome="Widenius"
      AND primeiro_nome >="M" AND primeiro_nome < "N";
```

Entretanto, o índice `nome` não será usado nas seguintes consultas:

```
mysql> SELECT * FROM teste WHERE primeiro_nome="Michael";
```

```
mysql> SELECT * FROM teste WHERE ultimo_nome="Widenius"
      OR primeiro_nome="Michael";
```

Para maiores informações sobre a maneira que o MySQL utiliza índices para melhorar o desempenho das consultas, veja [\[índices do MySQL\]](#), page [\(undefined\)](#).

5.4.6 Como o MySQL abre e fecha tabelas

`table_cache`, `max_connections` e `max_tmp_tables` afetam o número máximo de arquivos que o servidor mantém abertos. Se você aumentar um ou ambos destes valores, você pode ir contra um limite imposto pelo seu sistema operacional no número de arquivos abertos por processo. Entretanto, o limite em vários sistemas pode ser aumentado. Consulte a documentação de seu Sistema Operacional para saber como fazê-lo, porque o método para alterar o limite varia muito de um sistema para outro.

`table_cache` é relacionado a `max_connections`. Por exemplo, para 200 conexões concorrentes em execução, você deve ter um chace de tabela de pelo menos $200 * n$, onde n é o número máximo de tabelas em um join. Você também precisa reservar alguns descritores de arquivos para tabelas e arquivos temporários.

O cache de tabelas abertas pode crescer até um máximo de `table_cache` (o padrão é 64; isto pode ser alterado com a opção do `mysqld -O table_cache=#`. Uma tabela nunca é fechada, exceto quando o cache está cheio e outra thread tentar abrir uma tabela ou se você utilizar `mysqladmin refresh` ou `mysqladmin flush-tables`.

Quando o cache de tabela encher, o servidor usa o seguinte procedimento para encontrar uma entrada de cache para usar:

- Tabelas que não estiverem em uso são liberadas, na ordem LRU (least-recently-used), ou seja, a tabela que foi usada menos recentemente.
- Se o cache estiver cheio e nenhuma tabelas pode ser liberada, mas uma nova tabela precisar ser aberta, o cache é estendido temporariamente quando necessário.

- Se o cache estiver no estado temporariamente estendido e uma tabela vai do estado em-uso para o fora-de-uso, a tabela é fechada e liberada do cache.

Uma tabela é aberta para cada acesso simultâneo. Isto significa que se você tem duas threads acessando a mesma tabela ou acessando a tabela duas vezes na mesma query (com AS) a tabelas precisa ser aberta duas vezes. A primeira abertura de qualquer tabela exige dois descritores de arquivos; cada uso adicional da tabela exige somente um descritor. O descritor extra para a primeira abertura é para o arquivo de índice: este descritor é compartilhado entre todas as threads.

Você pode conferir se o seu cache de tabela está muito pequeno conferindo a variável do mysqld `opened_tables`. Se este valor for muito grande, mesmo se você não fez vários `FLUSH TABLES`, você deve aumentar seu cache de tabelas. See `<undefined>` [`SHOW STATUS`], page `<undefined>`.

5.4.7 Desvantagem em Criar um Número Grande de Tabelas no Mesmo Banco de Dados

Se você possui muitos arquivos em um diretório, operações de abrir, fechar e criação ficarão lentos. Se você executar instruções `SELECT` em diversas tabelas, existirá uma pequena sobrecarga quando o cache de tabela estiver cheio, porque para toda tabela que teve que ser aberta, outra deve ser fechada. Você pode reduzir esta sobrecarga tornando o cache de tabelas maior.

5.4.8 Por Quê Tantas Tabelas Abertas ?

Ao executar o comando `mysqladmin status`, você verá algo deste tipo:

```
Uptime: 426 Running threads: 1 Questions: 11082 Reloads: 1 Open tables: 12
```

Isto pode ser bastante estranho se você só possui 6 tabelas.

O MySQL é multithreaded, portanto ele pode ter várias consultas abertas na mesma tabela simultaneamente. Para minimizar o problema com duas threads tendo diferentes estados no mesmo arquivo, a tabela é aberta independentemente por cada thread concorrente. Isto exige mais memória e um descritor extra de arquivo para o arquivo de dados. O descritor de arquivo de índice é compartilhado entre todas as threads.

5.5 Otimizando o servidor MySQL

5.5.1 Sintonia dos Parâmetros em Tempo de Sistema/Compilação e na Inicialização

Nós iniciamos com o nível do sistema pois algumas destas decisões devem ser feitas bem cedo. Em outros casos uma rápida olhada para esta parte pode satisfazer porque ela não é tão importante para os grandes ganhos. Entretanto, é sempre bom ter ter noções de como você pode obter melhorias alterando coisas neste nível.

Qual sistema operacional a usar é realmente importante! Para obter o melhor uso de máquinas com múltiplas CPUs você deve utilizar Solaris (porque as threads funcionam muito bem) ou Linux (porque o kernel 2.2 tem suporte SMP muito bom). Em máquinas

Linux 32-bits temos o limite de tamanho de arquivo de 2G por padrão. Esperamos que isto seja corrigido logo quando novos sistemas de arquivos forem liberados (XFS/Reiserfs). Se você precisa desesperadamente de trabalhar com arquivos maiores que 2G em máquinas intel Linux, você deve obter o patch LFS para o sistema de arquivos ext2.

Como ainda não temos o MySQL em produção em muitas outras plataformas, nós aconselhamos que você teste a plataforma pretendida antes de escolhê-la, se possível.

Outras dicas:

- Se você possui RAM suficiente, você pode remover todos os dispositivos de troca. Alguns sistemas operacionais irão utilizar um dispositivo de troca em alguns contextos, mesmo se você possuir memória livre.
- Utilize a opção do MySQL `--skip-locking` para evitar bloqueios externos. Perceba que isto não irá afetar a funcionalidade do MySQL se você estiver executando um único servidor. Apenas lembre-se de desligar o servidor (ou travar as partes relevantes) antes de executar `myisamchk`. Em alguns sistemas esta troca é obrigatório porque o bloqueio externo não funciona em nenhum caso.

A opção `--skip-locking` está ligada por padrão quando compilando com MIT-pthreads, porque `flock()` não é totalmente suportado pelas MIT-pthreads em todas plataformas. É também o padrão para Linux pois o bloqueio de arquivos no Linux não é muito seguro.

O único caso que você não pode utilizar `--skip-locking` é se você precisa de vários servidores MySQL (não clientes) acessando os mesmos dados, ou executar `myisamchk` na tabela sem primeiramente descarregar e travar as tabelas no servidor `mysqld`.

Você pode continuar usando `LOCK TABLES/UNLOCK TABLES` mesmo se você estiver utilizando `--skip-locking`.

5.5.2 Parâmetros de Sintonia do Servidor

Você pode obter o tamanho padrão do buffer usados pelo servidor `mysqld` com este comando:

```
shell> mysqld --help
```

Este comando produz uma lista de todas as opções do `mysqld` e variáveis configuráveis. A saída inclui os valores padrão e se parece com isto:

```
Possible variables for option --set-variable (-O) are:
back_log                current value: 5
bdb_cache_size          current value: 1048540
binlog_cache_size       current value: 32768
connect_timeout         current value: 5
delayed_insert_timeout  current value: 300
delayed_insert_limit    current value: 100
delayed_queue_size      current value: 1000
flush_time              current value: 0
interactive_timeout     current value: 28800
join_buffer_size        current value: 131072
key_buffer_size         current value: 1048540
lower_case_table_names  current value: 0
long_query_time         current value: 10
max_allowed_packet      current value: 1048576
```

```

max_binlog_cache_size current_value: 4294967295
max_connections      current value: 100
max_connect_errors   current value: 10
max_delayed_threads  current value: 20
max_heap_table_size  current value: 16777216
max_join_size        current value: 4294967295
max_sort_length      current value: 1024
max_tmp_tables       current value: 32
max_write_lock_count current value: 4294967295
mysam_sort_buffer_size current value: 8388608
net_buffer_length    current value: 16384
net_retry_count      current value: 10
net_read_timeout     current value: 30
net_write_timeout    current value: 60
query_buffer_size    current value: 0
record_buffer        current value: 131072
record_rnd_buffer    current value: 131072
slow_launch_time     current value: 2
sort_buffer          current value: 2097116
table_cache          current value: 64
thread_concurrency   current value: 10
tmp_table_size       current value: 1048576
thread_stack         current value: 131072
wait_timeout         current value: 28800

```

Se existir um servidor `mysqld` em execução, você pode ver quais valores ele está usando atualmente para as variáveis executando este comando:

```
shell> mysqladmin variables
```

Para encontrar uma descrição completa de todas as variáveis na seção `SHOW VARIABLES` neste manual. See [\[SHOW VARIABLES\]](#), page [\[SHOW VARIABLES\]](#).

O comando `SHOW STATUS` exibe estatísticas de um servidor em funcionamento. See [\[SHOW STATUS\]](#), page [\[SHOW STATUS\]](#).

O MySQL utiliza algoritmos que são muito escaláveis, portanto, normalmente você pode trabalhar com pouca memória. Entretanto, se você fornecer ao MySQL mais memória, obterá um desempenho melhor.

Quando estiver ajustando um servidor MySQL, as duas variáveis mais importantes que devem ser usadas são `key_buffer_size` e `table_cache`. Você deve se sentir confiante que as duas estejam corretas antes de tentar alterar qualquer outra variável.

Se você possui miuita memória ($\geq 256\text{M}$) e várias tabelas e deseja obter o melhor desempenho com um número moderado de clientes, deve utilizar algo como:

```
shell> safe_mysqld -O key_buffer=64M -O table_cache=256 \
-O sort_buffer=4M -O record_buffer=1M &
```

Se possui apenas 128M e possui algumas poucas tabelas, mas ainda deseja realizar várias ordenações, você pode utilizar:

```
shell> safe_mysqld -O key_buffer=16M -O sort_buffer=1M
```

Se você possui pouca memória e tiver muitas conexões, utilize algo como:

```
shell> safe_mysqld -O key_buffer=512k -O sort_buffer=100k \
-O record_buffer=100k &
```

ou mesmo:

```
shell> safe_mysqld -O key_buffer=512k -O sort_buffer=16k \
-O table_cache=32 -O record_buffer=8k -O net_buffer=1K &
```

Se você estiver executando um `GROUP BY` ou `ORDER BY` em arquivos que são muito maiores que sua memória disponível você deve aumentar o valor de `record_rnd_buffer` para acelerar a leitura de registros depois que a ordenação é feita.

Quando você tiver instalado o MySQL, o diretório `'support-files'` irá conter alguns arquivos exemplos do `my.cnf`, `'my-huge.cnf'`, `'my-large.cnf'`, `'my-medium.cnf'` e `'my-small.cnf'`, você pode usá-los como base para otimizar seu sistema.

Se você possui várias conexões, “problemas de trocas” pode ocorrer a menos que o `mysqld` tenha sido configurado para usar muito pouca memória para cada conexão. O `mysqld` tem melhor performance se você tiver memória suficiente para todas as conexões, é claro.

Perceba que se você alterar uma opção para `mysqld`, ele permanece em efeito somente para aquela instância do servidor.

Para ver os efeitos de uma alteração de parâmetro, faça algo como:

```
shell> mysqld -O key_buffer=32m --help
```

Tenha certeza que a opção `--help` seja a última do comando; de outra forma o efeito de qualquer opções listadas depois na linha de comando não serão refletidas na saída.

5.5.3 Como a Compilação e a Ligação Afeta a Velocidade do MySQL

A maioria dos testes seguintes são feitos no Linux com os benchmarks do MySQL, mas eles devem fornecer alguma indicação para outros sistemas operacionais e workloads.

Você obtêm um executável mais veloz quando ligado com `-static`.

No Linux, você irá obter o código mais rápido quando compilando com `pgcc` e `-O3`. Para compilar `'sql_yacc.cc'` com estas opções, você precisa de cerca de 200M de memória porque o `gcc/pgcc` precisa de muita memória para criar todas as funções em linha. Também deve ser configurado o parâmetro `CXX=gcc` para evitar que a biblioteca `libstdc++` seja incluída (não é necessária). Perceba que com algumas versões do `pgcc`, o código resultante irá executar somente em verdadeiros processadores Pentium, mesmo que você utilize a opção do compilador para o código resultante que você quer, funcionando em todos os processadores do tipo x586 (como AMD).

Só pelo fato de utilizar um melhor compilador e/ou melhores opções do compilador você pode obter um aumento de desempenho de 10-30% na sua aplicação. Isto é particularmente importante se você mesmo compila o servidor SQL!

Nós testamos ambos os compiladores Cygnus Codefusion e o Fujitsu, mas quando os testamos, nenhum dos dois era suficientemente livre de erros para que o MySQL compilasse com as otimizações.

Quando você compila o MySQL deve incluir suporte somente para os conjuntos de caracteres que deseja usar. (Opção `--with-charset=xxx`). As distribuições binárias padrão do MySQL são compiladas com suporte para todos os conjuntos de caracteres.

Segue uma lista de algumas medidas que temos feito:

- Se você utiliza o `pgcc` e compila tudo com `-O6`, o servidor `mysqld` é 1% mais rápido do que com o `gcc 2.95.2`.
- Se você liga dinamicamente (sem `-static`), o resultado é 13% mais lento no Linux. Note que você ainda pode utilizar uma biblioteca do MySQL dinamicamente ligada. É só o servidor que é crítico para performance.
- Se você corta seu binário `mysqld` com `strip libexec/mysqld`, o binário gerado pode ficar até 4% mais rápido.
- Se você conecta utilizando TCP/IP em vez de utilizar sockets Unix, o resultado é 7.5% mais lento no mesmo computador. (Se você fizer conexão à `localhost`, o MySQL irá, por padrão, utilizar sockets).
- Se você conectar utilizando TCP/IP de outro computador sobre uma rede Ethernet de 100M as coisas ficarão de 8-11% mais lentas.
- Se você compilar com `--with-debug=full`, estará perdendo cerca de 20% para a maioria das consultas, mas algumas consultas podem demorar muito mais tempo (os benchmarks do MySQL demonstram 35% de perda). Se utilizar `--with-debug`, então irá perder 15%. Ao iniciar uma versão do `mysqld` compilada com `--with-debug=full` com `--skip-safemalloc` o resultado final deve ser perto de quando compilado com `--with-debug`.
- Em uma Sun SPARCstation 20, SunPro C++ 4.2 é 5 % mais rápido que `gcc 2.95.2`.
- Compilando com `gcc 2.95.2` para o ultrasparc com a opção `-mcpu=v8 -Wa,-xarch=v8plusa` melhora a performance em 4%.
- No Solaris 2.5.1, a MIT-pthreads é 8-12% mais lenta do que as threads nativas do Solaris em um único processador. Com mais carga/CPU's a diferença deve aumentar.
- Executando com `--log-bin` deixa o **MySQL** 1 % mais lento.
- Compilando no Linux-x86 com `gcc` sem frame pointers `-fomit-frame-pointer` ou `-fomit-frame-pointer -ffixed-ebp` deixa o `mysqld` 1-4% mais rápido.

A distribuição MySQL-Linux fornecida pela MySQL AB é normalmente compilada com `pgcc`, mas vamos retornar a usar o `gcc` pelo fato de um bug no `pgcc` que gera o código que não executa no AMD. Continuaremos a usar o `gcc` até que o bug seja resolvido. Neste meio tempo, se você possui uma máquina que não seja AMD, você pode ter um binário mais rápido compilando com o `pgcc`. O binário padrão do MySQL para Linux é ligado estaticamente para conseguir mais desempenho e ser mais portátil.

5.5.4 Como o MySQL Utiliza a Memória

A lista abaixo indica algumas das maneiras nas quais o servidor `mysqld` utiliza a memória. Onde aplicável, o nome da variável do servidor relevante ao uso de memória é fornecido:

- O buffer de chave (variável `key_buffer_size`) é compartilhado por todas as threads; Outros buffers usados pelo servidor são alocados quando necessários. See [\[Parâmetros de servidor\]](#), page [\[undefined\]](#).
- Cada conexão utiliza algum espaço específico da thread: Uma de pilha (padrão de 64K, variável `thread_stack`), um buffer de conexão (variável `net_buffer_length`), e um buffer de resultados (variável `net_buffer_length`). Os buffers de conexões e resultados são aumentados dinamicamente para `max_allowed_packet` quando necessário. Quando

uma consulta está sendo executada, uma cópia da string da consulta atual é também alocada.

- Todas as threads compartilhas a mesma memória base.
- Somente as tabelas ISAM / MyISAM compactadas são mapeadas em memória. Isto é porque o espaço de memória de 32-bits de 4GB não é grande o bastante para a maioria das grandes tabelas. Quando sistemas com endereçamento de 64-bits se tornarem comuns poderemos adicionar um suporte gieraal para o mapeamento de memória.
- Cada requisição fazendo uma varredura sequencial em uma tabela aloca um buffer de leitura (variável `record_buffer`).
- Ao ler registros na ordem 'randômica' (por exemplo, depois de uma ordenação) um buffer de leitura randômico é alocado para evitar pesquisas em disco. (variável `record_rnd_buffer`).
- Todas as joins são feitas em um único passo, e a maioria delas podem ser feitas mesmo sem usar uma tabela temporária. A maioria das tabelas temporárias são tabelas baseadas em memória (HEAP). Tabelas temporárias com uma grande extensão de registros (calculada como a soma do tamanho de todas as colunas) ou que contenham colunas BLOB são armazenadas em disco.

Um problema nas versões do MySQL anteriores a versão 3.23.2 é que se uma tabela HEAP excede o tamanho de `tmp_table_size`, você recebe o erro `The table nome_tabela is full`. Em versões mais novas isto é tratado alterando automaticamente a tabela em memória (HEAP) para uma tabela baseada em disco (MyISAM) quando necessário. Para contornar este problema, você pode aumentar o tamanho da tabela temporária configurando a opção `tmp_table_size` do `mysqld`, ou configurando a opção do SQL `SQL_BIG_TABLES` no progrma cliente. See [\[SET OPTION\]](#), page [\[undefined\]](#). Na versão 3.20 do MySQL, o número máximo da tabela temporária era `record_buffer*16`, assim, se você estiver utilizando esta versão, você terá que aumentar o valor `record_buffer`. Você também pode iniciar o `mysqld` com a opção `--big-tables` para sempre armazenar as tabelas temporárias em disco. Entretanto isto afetará a velocidade de várias consultas complicadas.

- A maioria das requisições da ordenação alocam um bufer de ordenação e 0-2 arquivos temporários dependendo do tamanho do resultado. See [\[Arquivos temporário\]](#), page [\[undefined\]](#).
- Quase todas as análises e cálculos são feitos em um armazenamento de memória local. Nenhuma sobrecarga de memória é necessário para ítems pequenos e a alocação e liberação normal de memória lenta é evitada. A memória é alocada somente para grandes strings inesperadas (isto é feito com `malloc()` e `free()`).
- Cada arquivo de índice é aberto uma vez e o arquivo de dados é aberto uma vez para cada thread concorrente. Uma estrutura de tabela, estrutura de coluna para cada coluna e um buffer de tamanho $3 * n$ é alocado para cada thread concorrente. (onde `n` é o maior tamanho do registro, sem levar em consideração colunas BLOB. Um BLOB utiliza de 5 a 8 bytes mais o tamanho dos dados contidos na mesma. O manipulador de tabelas ISAM/MyISAM irão usar um registro extra no buffer para uso interno.
- Para cada tabela com colunas BLOB, um buffer é aumentado dinamicamente para ler grandes valores BLOB. Se você ler uma tabela, um buffer do tamanho do maior registro BLOB é alocado.

- Manipuladores de tabelas para todas tabelas em uso são salvos em um cache e gerenciado como FIFO. Normalmente o cache possui 64 entradas. Se uma tabela foi usada por duas threads ao mesmo tempo, o cache terá duas entradas para a tabela. See [\[Cache de tabela\]](#), page [\[Cache de tabela\]](#).
- Um comando `mysqladmin flush-tables` fecha todas tabelas que não estão em uso e marca todas tabelas em uso para serem fechadas quando a thread atualmente em execução terminar. Isto irá liberar efetivamente a maioria da memória em uso.

`ps` e outros programas de informações do sistema podem relatar que o `mysqld` usa muita memória. Isto pode ser causado pelas pilhas de threads em diferentes endereços de memória. Por exemplo, a versão do `ps` do Solaris conta a memória não usada entre as pilhas como memória usada. Você pode verificar isto conferindo a memória disponível com `swap -s`. Temos testado o `mysqld` com detectores comerciais de perda de memória, portanto tais perdas não devem existir.

5.5.5 Como o MySQL Utiliza o DNS

Quando uma nova thread conecta ao `mysqld`, o `mysqld` irá extender uma nova thread para lidar com o pedido. Esta thread primeiro confere se o nome da máquina está no cache de nomes de máquinas. Se não, a thread irá chamar `gethostbyaddr_r()` e `gethostbyname_r()` para resolver o nome.

Se o sistema operacional não suporta as chamadas acima em threads seguras, a thread irá travar um mutex e chamar `gethostbyaddr()` e `gethostbyname()`. Perceba que neste caso nenhuma outra thread pode resolver outros nomes de máquinas que não existam no cache de nomes de máquina até que a primeira thread esteja pronta.

Você pode desabilitar a procura de nomes de máquinas no DNS iniciando o `mysqld` com a opção `--skip-name-resolve`. Neste caso você só pode usar números IP nas tabelas de privilégio do MySQL.

Se você possuir um DNS muito lento e várias máquinas, pode obter mais desempenho desligando a procura de nomes de máquinas usando a opção `--skip-name-resolve` ou aumentando `HOST_CACHE_SIZE` (default: 128) e recompilar `mysqld`.

Você pode desabilitar o cache de nomes de máquinas com `--skip-host-cache`. Este cache pode ser limpo com os comandos `FLUSH HOSTS` ou `mysqladmin flush-hosts`.

Se você não deseja permitir conexões TCP/IP, pode iniciar o `mysqld` com `--skip-networking`.

5.5.6 SET Syntax

```
SET [OPTION] SQL_VALUE_OPTION= valor, ...
```

`SET OPTION` configura várias opções que afetam a operação do servidor ou seu cliente. Qualquer opção que for configurada continua em efeito até que a seção atual termine ou até que a opção seja configurada para um valor diferente.

```
CHARACTER SET nome_conjunto_caracteres | DEFAULT
```

Mapeia todas as strings do e para o cliente com o mapa especificado. Atualmente a única opção para `character_set_name` é `cp1251_koi8`, mas você pode adicionar novos mapas editando o arquivo `'sql/convert.cc'` na distribuição

fonte do MySQL. O mapeamento padrão pode ser restaurado utilizando o valor `DEFAULT` para `character_set_name`.

Perceba que a sintaxe para configurar a opção `CHARACTER SET` é diferente da sintaxe para configurar as outras opções.

`PASSWORD = PASSWORD('alguma senha')`

Configura a senha para o usuário atual. Qualquer usuário que não seja anônimo pode alterar sua própria senha!

`PASSWORD FOR user = PASSWORD('alguma senha')`

Configura a senha para um usuário específico no servidor atual. Somente um usuário com acesso ao banco de dados `mysql` pode fazer isto. O usuário deve ser fornecido no formato `usuário@home_maquina`, onde `usuário` e `nome_maquina` são exatamente o que estão listados nas colunas `User` e `Host` da tabela `mysql.user`. Por exemplo, se você possui uma entrada com os campos `User` e `Host` com `'bob'` e `'%.loc.gov'`, você escreveria:

```
mysql> SET PASSWORD FOR bob@%.loc.gov" = PASSWORD("nova_senha");
```

or

```
mysql> UPDATE mysql.user SET password=PASSWORD("nova_senha") where user="
```

`SQL_AUTO_IS_NULL = 0 | 1`

Se configurado com 1 (padrão) o último registro inserido em uma tabela com um registro `auto_increment` pode ser encontrado com a seguinte construção: `WHERE auto_increment_column IS NULL`. Isto é usado por alguns programas ODBC como o Access.

`AUTOCOMMIT= 0 | 1`

Se configurado com 1 todas alterações em uma tabela será feita de uma vez. Para iniciar uma transação de vários comandos, deve ser usada a instrução `BEGIN`. See [\[COMMIT\], page \[COMMIT\]](#). Se configurado com 0 deve ser usado `COMMIT/ROLLBACK` para aceitar/recusar aquela transação. See [\[COMMIT\], page \[COMMIT\]](#). Note que quando você altera do modo não-`AUTOCOMMIT` para `AUTOCOMMIT`, o MySQL irá fazer um `COMMIT` automático em quaisquer transações abertas.

`SQL_BIG_TABLES = 0 | 1`

Se configurado com 1, todas tabelas temporárias são armazenadas em disco em vez da memória. Isto será um pouco mais lento, mas você não terá o erro `The table nome_tabela is full` para grandes operações `SELECT` que necessitam de uma tabela temporária grande. O valor padrão para uma nova conexão é 0 (isto é, utiliza tabelas temporárias em memória).

`SQL_BIG_SELECTS = 0 | 1`

Se configurado com 0, o MySQL irá abortar se uma tentativa de se fazer um `SELECT` for, provavelmente, levar muito tempo. Isto é útil quando uma instrução `WHERE` não aconselhada for utilizado. Uma grande consulta é definida como uma `SELECT` que provavelmente terá que examinar mais que `max_join_size` registros. O valor padrão para uma nova conexão é 1 (que permitirá qualquer instrução `SELECT`).

`SQL_BUFFER_RESULT = 0 | 1`

`SQL_BUFFER_RESULT` irá forçar que o resultado de `SELECT`'s seja colocado em tabelas temporárias. Isto irá ajudar o MySQL a liberar mais cedos bloqueios de tabela e ajudarão em casos onde elas ocupam muito tempo para enviar o conjunto de resultados para o cliente.

`SQL_LOW_PRIORITY_UPDATES = 0 | 1`

Se configurado com 1, todas instruções `INSERT`, `UPDATE`, `DELETE` e `LOCK TABLE WRITE` irão esperar até que não existam `SELECT` ou `LOCK TABLE READ` pendentes na tabela afetada.

`SQL_MAX_JOIN_SIZE = valor | DEFAULT`

Não permita que `SELECT`s que provavelmente necessitem examinar mais que `valor` combinações de registros. Configurando este valor, você pode obter `SELECT`s onde chaves não são usadas corretamente e que provavelmente gastarão um bom tempo. Configurando-o para um valor diferente do `DEFAULT` irá definir o atributo `SQL_BIG_SELECTS` com o padrão. Se você configurar o atributo `SQL_BIG_SELECTS` novamente, a variável `SQL_MAX_JOIN_SIZE` será ignorada. Você pode configurar um valor padrão para esta variável iniciando o `mysqld` com `-O max_join_size=#`.

`SQL_SAFE_UPDATES = 0 | 1`

Se configurado com 1, o MySQL irá abortar se tentarmos fazer um `UPDATE` ou `DELETE` sem utilizar uma chave ou `LIMIT` na cláusula `WHERE`. Desta forma é possível capturar atualizações erradas ao criarmos comandos SQL manualmente.

`SQL_SELECT_LIMIT = valor | DEFAULT`

O número máximo de registros para retornar de instruções `SELECT`. Se uma `SELECT` tem uma cláusula `LIMIT`, o `LIMIT` tem precedência sobre o valor de `SQL_SELECT_LIMIT`. O valor padrão para uma nova conexão é "unlimited" (ilimitado). Se você alterou o limite, o valor padrão pode ser restaurado atribuindo o valor `DEFAULT` a `SQL_SELECT_LIMIT`.

`SQL_LOG_OFF = 0 | 1`

Se configurado com 1, nenhum registro será feito no log padrão para este cliente, se o cliente tiver o privilégio **process**. Isto não afeta o log de atualizações!

`SQL_LOG_UPDATE = 0 | 1`

Se configurado com 0, nenhum registro será feito no log de atualizações para o cliente, se o cliente tiver o privilégio **process**. Isto não afeta o log padrão!

`SQL_QUOTE_SHOW_CREATE = 0 | 1`

Se configurado com 1, `SHOW CREATE TABLE` irá colocar os nomes de tabela e colunas entre aspas. Está **ligado** por padrão, para que replicação de tabelas com nomes de colunas estranhos funcione. [\[SHOW CREATE TABLE\]](#), [page](#).

`TIMESTAMP = valor_timestamp | DEFAULT`

Configura a hora/data para este cliente. É usado para obter a hora e data original se você utiliza o log de atualizações para restaurar registros. `valor_timestamp` deve ser um timestamp UNIX Epoch, não um timestamp MySQL.

LAST_INSERT_ID = #

Configura o valor a ser retornado de `LAST_INSERT_ID()`. Ele é armazenado no log de atualizações quando você utiliza `LAST_INSERT_ID()` em um comando que atualiza uma tabela.

INSERT_ID = #

Configura o valor que será usado pelo comando `INSERT` ou `ALTER TABLE` seguinte ao inserir um valor `AUTO_INCREMENT`. Isto é usado principalmente com o log de atualizações.

5.6 Detalhes de Disco

- Como mencionado acima, pesquisas em disco são o maior gargalo de desempenho. Estes problemas ficam cada vez mais aparentes quando os dados começam a crescer tanto que efetivo armazenamento em cache se torna impossível. Para grandes bancos de dados, onde você acessa dados mais ou menos aleatoriamente, você pode ter certeza de que precisará de pelo menos uma busca em disco para ler e várias para gravar os dados. Para minimizar este problema, utilize discos com menor tempo de pesquisa.
- Aumente o número de eixo de discos disponíveis (e então reduza a sobrecarga da pesquisa) ligando arquivos simbolicamente em diferentes discos ou utilizando striping de discos.

Usando links simbólicos

Significa que você liga simbolicamente o índice e/ou arquivos de dados ao diretório de dados normal em outro disco (que pode também ser striped). Isto torna os tempos de pesquisa e leitura melhor (Se os discos não são usados para outras coisas). See [\[Links simbólicos\]](#), page [\[undefined\]](#).

Striping

Striping significa que você possui vários discos e coloca o primeiro bloco no primeiro disco, o segundo bloco no segundo disco, e o N-simo no (N módulo número_de_discos) disco, e assim por diante. Isto significa que se o seu tamanho de dados normais é menos que o tamanho do bloco (ou perfeitamente alinhado) você irá obter um desempenho muito melhor. Perceba que striping é muito dependente do SO e do tamanho do bloco. Portanto meça a performance de sua aplicação com diferentes tamanhos de blocos. See [\[Benchamrks\]](#), page [\[undefined\]](#).

Perceba que a diferença de velocidade para striping é **muito** dependente dos parâmetros. Dependendo de como você configura os parâmetros do striping e do número de discos você pode obter uma diferença de várias ordens de grandeza. Note que você deve escolher a otimização randômica ou pelo acesso sequencial.

- Para confiabilidade você pode desejar utilizar RAID 0+1 (striping + espelhamento) mas neste caso você irá precisar de 2*N discos para armazenar N discos de dados. Isto é provavelmente a melhor opção se você possuir dinheiro! Você pode também, entretanto, ter que investir em algum software gerenciador de volumes para lidar com isto eficientemente.

- Uma boa opção é ter dados de média importância (aqueles que podem ser regenerados) em um armazenamento RAID 0 enquanto os dados realmente importantes (como informações de máquinas e logs) em um sistema RAID 0+1 ou RAID de N discos. RAID N pode ser um problema se você tem várias escritas devido ao tempo para atualizar os bits de paridade.
- Você pode também configurar os parâmetros para o sistema de arquivos que o banco de dados usa. Uma alteração simples é montar o sistema de arquivos com a opção `noatime`. Isto faz com que ele evite a atualização do último tempo de acesso no inode e com isto também evita algumas buscas em disco.
- No Linux, você pode obter um desempenho muito melhor (cerca de 100% sobre carga pode ser comum) utilizando `hdparm` para configurar sua interface de disco! O exemplo a seguir deve ser muito útil para o MySQL (e provavelmente várias outras aplicações):

```
hdparm -m 16 -d 1
```

Perceba que o desempenho/confiança ao utilizar o exemplo acima depende de seu hardware, portanto nós sugerimos que você teste bem seu sistema depois de utilizar `hdparm`! Por favor consulte a página do manual (man) do `hdparm` para maiores informações! Se o `hdparm` não for usado corretamente, poderá resultar em corrupção do sistema de arquivos. Realize backups de tudo antes de experimentar!

- Em vários sistemas operacionais os discos podem ser montados com a opção `'async'` para configurar o sistema de arquivos a ser atualizado de modo assíncrono. Se o seu computador é razoavelmente estável, isto deve fornecer mais desempenho sem sacrificar a segurança. (Esta opção é ligada por padrão no Linux.)
- Se você não precisar saber a última vez que um arquivo foi acessado (o que realmente não é muito útil em um servidor de banco de dados), você pode montar seu sistema de arquivos com a opção `noatime`.

5.6.1 Utilizando Links Simbólicos

Você pode mover tabelas e bancos de dados do diretório de banco de dados para outras localizações e trocá-los por links simbólicos para os novos locais. Você pode fazer isto, por exemplo, para mover um banco de dados para um sistema de arquivos com mais espaço livre ou aumentar a velocidade de seu sistema espalhando suas tabelas para discos diferentes.

A maneira recomendada de se fazer isto é ligar simbolicamente bancos de dados a discos diferentes e só ligar tabelas como último recurso.

5.6.1.1 Utilizando Links Simbólicos para Bancos de Dados

A maneira de ligar simbolicamente um banco de dados é, primeiramente, criar um diretório em algum disco onde você possui espaço livre e então criar uma ligação simbólica para ele a partir do diretório do banco de dados do MySQL.

```
shell> mkdir /dr1/databases/test
shell> ln -s /dr1/databases/test mysql-d-datadir
```

O MySQL não suporta que você ligue um diretório a vários bancos de dados. Trocando um diretório de banco de dados com uma ligação simbólica irá funcionar bem desde que não sejam feitos links simbólicos entre os bancos de dados. Suponha que você tenha um banco

de dados `db1` sob o diretório de dados do MySQL, e então criar uma ligação simbólica `db2` que aponte para `db1`.

```
shell> cd /caminho/para/diretorio/dados
shell> ln -s db1 db2
```

Agora, para qualquer tabela `tbl_a` em `db1`, também aparecerá uma tabela `tbl_a` em `db2`. Se uma thread atualizar `db1.tbl_a` e outra atualizar `db2.tbl_a`, ocorrerão problemas.

Se você realmente precisar disto, você deve alterar o código seguinte em `'mysys/mf_format.c'`:

```
if (flag & 32 || (!lstat(to,&stat_buff) && S_ISLNK(stat_buff.st_mode)))
```

para

```
if (1)
```

No Windows você pode utilizar links simbólicos para diretórios compilando o MySQL com `-DUSE_SYMDIR`. Isto lhe permite colocar diferentes bancos de dados em discos diferentes. See [\[Links simbólicos no Windows\]](#), page [\[undefined\]](#).

5.6.1.2 Utilizando Links Simbólicos para Tabelas

Antes do MySQL 4.0 você não deve utilizar tabelas com ligações simbólicas, se você não tiver muito cuidado com as mesmas. O problema é que se você executar `ALTER TABLE`, `REPAIR TABLE` ou `OPTIMIZE TABLE` em uma tabela ligada simbolicamente, os links simbólicos serão removidas e substituídos pelos arquivos originais. Isto acontece porque o comando acima funciona criando um arquivo temporário no diretório de banco de dados e quando o comando é completo, substitui o arquivo original pelo arquivo temporário.

Você não deve ligar simbolicamente tabelas em um sistema que não possui uma chamada `realpath()` completa. (Pelo menos Linux e Solaris suportam `realpath()`)

No MySQL 4.0 links simbólicos só são suportados completamente por tabelas MyISAM. Para outros tipos de tabelas você provavelmente obterá problemas estranhos ao fazer qualquer um dos comandos mencionados acima.

O tratamento de links simbólicos no MySQL 4.0 funciona da seguinte maneira (isto é mais relevante somente para tabelas MyISAM).

- No diretório de dados você sempre terá o arquivo de definições das tabelas e os arquivos de índice/dados.
- Você pode ligar simbolicamente o arquivo índice e o arquivo de dados para diretórios diferentes, independente do outro arquivo.
- A ligação pode ser feita partir do sistema operacional (se o `mysqld` não estiver em execução) ou com o comando `INDEX/DATA DIRECTORY="caminho-para- diretorio"` em `CREATE TABLE`. See [\[CREATE TABLE\]](#), page [\[undefined\]](#).
- `myisamchk` não irá substituir um link simbólico pelo índice/arquivo mas funciona diretamente nos arquivos apontados pelos links simbólicos. Qualquer arquivo temporário será criado no mesmo diretório que o arquivo de dados/índice está.
- Quando você remove uma tabela que está usando links simbólicos, o link e o arquivo para o qual ela aponta são apagados. Esta é uma boa razão pela qual você NÃO deve executar `mysqld` como root e não deve permitir que pessoas tenham acesso de escrita ao diretórios de bancos de dados do MySQL.

- Se você renomear uma tabela com `ALTER TABLE RENAME` e não deseja alterar o banco de dados, o link simbólico para o diretório de banco de dados será renomeada corretamente.
- Se você utiliza `ALTER TABLE RENAME` para mover uma tabela para outro banco de dados, então a tabela será movida para outro diretório de banco de dados e os links simbólicos antigos e os arquivos para os quais eles apontam serão removidos.
- Se você não utiliza links simbólicos, você deve usar a opção `--skip-symlink` do `mysqld` para garantir que ninguém pode apagar ou renomear um arquivo fora do diretório de dados do `mysqld`.

O que ainda não é suportado:

- `ALTER TABLE` ignora todas as opções `INDEX/DATA DIRECTORY="caminho"`.
- `CREATE TABLE` não relata se a tabela possui links simbólicos.
- O `mysqldump` não inclui a informação de links simbólicos na saída.
- `BACKUP TABLE` e `RESTORE TABLE` não respeitam links simbólicos.

6 Referência de Linguagem do MySQL

O MySQL possui uma interface SQL muito complexa mas intuitiva e fácil de aprender. Este capítulo descreve os vários comandos, tipos e funções que você precisa conhecer para usar o MySQL de maneira eficiente e efetiva. Este capítulo também serve como referência para todas as funcionalidades incluídas no MySQL. Para poder utilizar este capítulo eficientemente, você deve achar útil fazer referência aos vários índices.

6.1 Estrutura da Linguagem

6.1.1 Literais: Como Gravar Strings e Numerais

Esta seção descreve as diversas maneiras para gravar strings e números no MySQL. Ela também cobre as várias nuances e “pegadinhas” pelas quais você pode passar ao lidar com estes tipos básicos no MySQL.

6.1.1.1 Strings

Uma string é uma sequência de caracteres, cercada por caracteres de aspas simples (‘’) ou duplas (‘’’) (Se você utiliza o modo ANSI deve utilizar somente as aspas simples). Exemplos:

```
'uma string'
"outra string"
```

Em uma string, certas sequências tem um significado especial. Cada uma destas sequências começam com uma barra invertida (‘\’), conhecida como *caracter de escape*. O MySQL reconhece a seguinte sequência de escape:

\0	Um caracter ASCII 0 (NUL).
\’	Um caracter de aspas simples (‘’).
\"	Um caracter de aspas duplas (‘’’).
\b	Um caracter de backspace.
\n	Um caracter de nova linha.
\r	Um caracter de retorno de carro.
\t	Um caracter de tabulação.
\z	ASCII(26) (Control-Z). Este caracter pode ser codificado para permitir que você contorne o problema que o ASCII(26) possui como END-OF-FILE ou EOF (Fim do arquivo) no Windows. (ASCII(26) irá causar problemas se você tentar usar <code>mysql banco_dados < nome_arquivo</code>).
\\	O caracter de barra invertida (‘\’) character.
\%	Um caracter ‘%’. Ele pode ser usado para pesquisar por instâncias literais de ‘%’ em contextos onde ‘%’ deve, de outra maneira, ser interpretado como um meta character. See [Funções de comparações de string] , page [undefined] .

`_` Um caracter ‘_’. Ele é usado para pesquisar por instâncias literais de ‘_’ em contextos onde ‘_’ deve, de outra maneira, ser intrerpretado como um meta caracter. See [\(undefined\)](#) [Funções de comparações de string], page [\(undefined\)](#).

Note que se você utilizar ‘\%’ ou ‘_’ em alguns contextos de strings, eles retornarão as strings ‘\%’ e ‘_’ e não ‘%’ e ‘_’.

Estas são as várias maneiras de incluir aspas com uma string:

- Um ‘’ dentro de uma string com ‘’ pode ser escrita como ‘’’’.
- Um ‘’ dentro de uma string com ‘’ pode ser escrita como ‘””’.
- Você pode preceder o caracter de aspas com um caracter de escape (‘\’).
- Um ‘’ dentro de uma string com ‘’ não precisa de tratamento especial e não precisa ser duplicada ou utilizada com caracter de escape. Da mesma maneira, ‘’ dentro de uma string com ‘’ não necessita de tratamento especial.

As instruções `SELECT` exibidas abaixo demonstram como citações e escapes funcionam:

```
mysql> SELECT 'hello', 'hello', 'hello', 'hel'lo', '\hello';
+-----+-----+-----+-----+-----+
| hello | "hello" | "hello" | hel'lo | 'hello |
+-----+-----+-----+-----+-----+
```

```
mysql> SELECT "hello", "'hello'", ''hello'', "hel"lo", "\"hello";
+-----+-----+-----+-----+-----+
| hello | 'hello' | ''hello'' | hel"lo | "hello |
+-----+-----+-----+-----+-----+
```

```
mysql> SELECT "Isto\nsão\nQuatro\nlinhas";
+-----+
| Isto
são
Quatro
linhas |
+-----+
```

Se você deseja inserir dados binários em uma coluna `BLOB`, os caracteres a seguir devem ser representados por sequências de espaço:

`NUL` ASCII 0. Você deve representá-lo como ‘\0’ (uma barra invertida e um caractere ‘0’).

`\` ASCII 92, barra invertida. Representado como ‘\\’.

`'` ASCII 39, aspas simples. Representado como ‘\’’.

`"` ASCII 34, aspas duplas. Representado como ‘\”’.

Se você escreve código C, você pode utilizar a função da API C `mysql_escape_string()` para caracteres de escape para a instrução `INSERT`. See [\(undefined\)](#) [Visão geral da função API C], page [\(undefined\)](#). No Perl, pode ser utilizado o método `quote` do pacote `DBI` para converter caracteres especiais para as sequências de escape corretas. See [\(undefined\)](#) [Classe `DBI` do Perl], page [\(undefined\)](#).

Deve ser utilizada uma função de escape em qualquer string que contém qualquer um dos caracteres especiais listados acima!

6.1.1.2 Números

Inteiros são representados como uma sequência de dígitos. Números de ponto flutuante utilizam ‘.’ como um separador decimal. Ambos os tipos devem ser precedidos por ‘-’ para indicar um valor negativo.

Exemplos de inteiros válidos:

```
1221
0
-32
```

Exemplo de números de ponto flutuante válidos:

```
294.42
-32032.6809e+10
148.00
```

Um inteiro pode ser usado em um contexto de ponto flutuante; ele é interpretado como o de ponto flutuante equivalente.

6.1.1.3 Valores Hexadecimais

O MySQL suporta valores hexadecimais. No contexto numérico estes atuam como um inteiro (precisão de 64-bits). No contexto de strings, atuam como uma string binária onde cada par de dígitos hexadecimais é convertido para um caracter:

```
mysql> SELECT 0xa+0;
-> 10
mysql> select 0x5061756c;
-> Paul
```

Strings hexadecimais são frequentemente usadas pelo ODBC para fornecer valores às colunas BLOB.

6.1.1.4 Valores NULL

O valor NULL significa “sem dados” e é diferente de valores como 0 para tipos numéricos ou strings vazias para tipos string. See [\(undefined\)](#) [Problemas com NULL], page [\(undefined\)](#).

NULL pode ser representado por \N ao usar o formato de arquivo texto para importação ou exportação (LOAD DATA INFILE, SELECT ... INTO OUTFILE). See [\(undefined\)](#) [LOAD DATA], page [\(undefined\)](#).

6.1.2 Nomes de Banco de dados, Tabela, Índice, Coluna e Apelidos

Nomes de banco de dados, tabela, índice, coluna e apelidos seguem todos as mesmas regras no MySQL.

Note que as regras foram alteradas a partir do MySQL versão 3.23.6, quando introduzimos aspas em identificadores (nomes banco de dados, tabela e coluna) com ‘‘’. ‘’’ funcionará também para citar identificadores se você executar no modo ANSI. See [\(undefined\)](#) [Modo ANSI], page [\(undefined\)](#).

Identificador	Tamanho máximo	Caracteres permitidos
Banco de dados	64	Qualquer caractere que é 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.
Alias	255	Todos os caracteres.

Note que em adição ao mostrado acima, você não pode ter ASCII(0) ou ASCII(255) ou o caracter de citação (aspas) em um identificador.

Perceba que se o identificador é uma palavra restrita ou contém caracteres especiais você deve sempre colocá-lo entre ' ao usá-lo:

```
SELECT * from 'select' where 'select'.id > 100;
```

Em versões anteriores do MySQL, as regras se nomes eram as seguintes:

- Um nome pode consistir de caracteres alfanuméricos do conjunto atual de caracteres e também '_' e '\$'. O conjunto de caracteres padrão é o ISO-8859-1 Latin1; e pode ser alterado com a opção `--default-character-set` no `mysqld`. See [\(undefined\)](#) [Conjunto de caracteres], page [\(undefined\)](#).
- Um nome pode iniciar com qualquer caractere que é legal no nome. Em particular, pode iniciar com um número (isto difere de vários outros sistemas de bancos de dados!). Entretanto um nome não pode consistir *somente* de números.
- O caractere '.' não pode ser utilizado em nomes porque ele é usado para extender o formato pelo qual você pode fazer referências a colunas (veja abaixo).

É recomendado que você não utilize nomes como `1e`, porque uma expressão como `1e+1` é ambígua. Ela pode ser interpretada como a expressão `1e + 1` ou como o número `1e+1`.

No MySQL você pode se referir a uma coluna utilizando uma das formas seguintes:

Coluna de referência	Significado
<code>nome_campo</code>	Coluna <code>nome_campo</code> de qualquer tabela usada na consulta contendo uma coluna com aquele nome.
<code>nome_tabela.nome_campo</code>	Coluna <code>nome_campo</code> da tabela <code>nome_tabela</code> do banco de dados atual.
<code>nome_bd.nome_tabela.nome_campo</code>	Coluna <code>nome_campo</code> da tabela <code>nome_tabela</code> do banco de dados <code>nome_bd</code> . Esta forma é disponível no MySQL Versão 3.22 ou posterior.
<code>'nome_coluna'</code>	Uma coluna que é uma palavra chave ou contém caracteres especiais.

Você não precisa especificar um prefixo de `nome_tabela` ou `nome_bd.nome_tabela` para uma referência de coluna em uma instrução, a menos que a referência seja ambígua. Por exemplo, suponha que cada tabela `t1` e `t2` contenham uma coluna `c`, e você deve recuperar `c` em uma instrução `SELECT` que utiliza ambas tabelas `t1` e `t2`. Neste caso, `c` é ambíguo porque ele não é único entre as tabelas usadas na instrução, portanto deve ser indicado qual é a tabela que se deseja escrever, `t1.c` ou `t2.c`. De mesma forma, se você for recuperar de uma tabela `t` em um banco de dados `db1` e uma tabela `t` em um banco de dados `db2`, você deve se referir às colunas nestas tabelas como `db1.t.nome_campo` e `db2.t.nome_campo`.

A sintaxe `.nome_tabela` indica a tabela `nome_tabela` no banco de dados atual. Esta sintaxe é aceita para compatibilidade ODBC, porque alguns programas ODBC prefixam os nomes das tabelas com um caractere `'.'`.

6.1.3 Caso Sensitivo nos Nomes

No MySQL, bancos de dados e tabelas correspondem a diretórios e arquivos em seus diretórios. Conseqüentemente, o caso sensitivo no sistema operacional irá determinar o caso sensitivo nos nomes de bancos de dados e tabelas. Isto significa que nomes de bancos de dados e tabelas são caso sensitivo no Unix e caso insensitivo no Windows. See [\[Extensões ao ANSI\]](#), page [\[undefined\]](#).

NOTA: Apesar de nomes de bancos e tabelas serem caso insensitivo no Windows, você não deve fazer referência a um certo banco de dados ou tabela utilizando casos diferentes na mesma consulta. A consulta a seguir não deve funcionar porque ela chama uma tabela como `minha_tabela` e outra como `MINHA_TABELA`.

```
mysql> SELECT * FROM minha_tabela WHERE MINHA_TABELA.col=1;
```

Nomes de colunas não são caso sensitivo em todas as circunstâncias.

Aliases nas tabelas são caso sensitivo. A consulta seguinte não deve funcionar porque ela faz referência ao alias como `a` e como `A`.

```
mysql> SELECT nome_campo FROM nome_tabela AS a
        WHERE a.nome_campo = 1 OR A.nome_campo = 2;
```

Apelidos (alias) em colunas não caso insensitivo.

Se você tem um problema para lembrar o caso usado para os nomes de tabelas, adote uma convenção consistente, como sempre criar bancos de dados e tabelas utilizando nomes em minúsculas.

Uma maneira para evitar este problema é iniciar o `mysqld` com `-O lower_case_table_names=1`. Por padrão esta opção é 1 no Windows e 0 no Unix.

Se `lower_case_table_names` for 1, o MySQL irá converte todos os nomes de tabelas para minúsculo no armazenamento e pesquisa. Perceba que se você alterar esta opção, será necessário converter primeiramente seus nomes de tabelas antigos para minúsculo antes de iniciar o `mysqld`.

6.1.4 Variáveis de Usuário

O MySQL suporta variáveis específicas de thread com a sintaxe `@nomevariável`. Um nome de variável consiste de caracteres alfanuméricos do conjunto atual de caracteres e também `'_'`, `'$'` e `'.'`. O conjunto de caracteres padrão é ISO-8859-1 Latin1; ele pode ser alterado com a opção `--default-character-set` do `mysqld`. See [\[Conjunto de caracteres\]](#), page [\[undefined\]](#).

As variáveis não precisam ser inicializadas. Elas contém `NULL` por padrão e podem armazenar um valor inteiro, real ou uma string. Todas as variáveis de uma thread são automaticamente liberadas quando uma thread termina.

Você pode configurar uma variavel com a sintaxe `SET`.

```
SET @variável= { expressao inteira | expressiao real | expressao string }
[,@variável= ...].
```

Voce tambm pode configurar uma variável em uma expressão com a sintaxe `@variable:=expr`

```
select @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
+-----+-----+-----+-----+
| @t1:=(@t2:=1)+@t3:=4 | @t1 | @t2 | @t3 |
+-----+-----+-----+-----+
|                    5 |  5 |  1 |  4 |
+-----+-----+-----+-----+
```

(Devemos utilizar a syntax `:=` aqui porque `=` esta reservado para comparações).

Variáveis de usuários devem ser utilizadas em expressões onde são permitidas. Isto não inclui utiliza-las em contextos onde um número é explicitamente necessário, assim como na cláusula `LIMIT` de uma instrução `SELECT` ou a clausula `IGNORE number LINE` de uma instrução `LOAD DATA`.

NOTE: Em uma instrução `SELECT`, cada expressão só é avaliada quando enviada ao cliente. Isto significa que nas cláusula `HAVING`, `GROUP BY`, ou `ORDER BY`, você não pode fazer referência a uma expreesão que envolve variáveis que são configuradas na instrução `SELECT`. Por exemplo, a seguinte instrução **NÃO** funcionará como o esperado:

```
SELECT (@aa:=id) AS a, (@aa+3) AS b FROM nome_tabela HAVING b=5;
```

A razão é que o `@aa` não irá conter o valor da linha atual. , mas o valor da `id` da linha previamente aceita.

6.1.5 Sintaxe de Comentários

O servidor MySQL suporta os estilos de comentário `#` no fim da linha, `--` no fim da linha e `/*` na linha ou em multiplas linhas `*/`

```
mysql> select 1+1;      # Este comentário continua até o fim da linha
mysql> select 1+1;      -- Este comnetário continua até o fim da linha
mysql> select 1 /* Este é um comentário de linha */ + 1;
mysql> select 1+
/*
Este é um comentário
de múltiplas linhas
*/
1;
```

Note que o estilo de comentário `--` requer que pelo menos um espaço após o código `{--!`

Embora o servidor entenda as sintaxes de comentários aqui descritas, existem algumas limitções no modo que o cliente `mysql` analisa o comentário `/* ... */:`

- Caracteres de aspas simples e aspas duplas são utilizados para indicar o início de uma string com aspas, mesmo dentro de um comentário. Se as aspas não coincidirem com uma segunda aspas dentro do comentário, o analisador não percebe que o comentário tem um fim. Se você estiver executando o `mysql` interativamente, você pode perceber a confusão ocorrida por causa da mudança do prompt de `mysql>` para `'>` ou `">`.
- Um ponto e vírgula é utilizado para indicar o fim de uma instrução SQL e qualquer coisa que venha após ele indica o início da próxima instrução.

Estas limitações se aplicam tanto a quando se executa `mysql` interativamente quanto quando se coloca os comandos em um arquivo e pede para que `mysql` leia as entradas deste arquivo com o comando `mysql < some-file`.

MySQL não suporta o estilo de comentário ANSI SQL `--`. See [\(undefined\)](#) [Missing comments], page [\(undefined\)](#).

6.1.6 MySQL é Exigente com Palavra Reservadas?

Um problema comum ocorre quando tentamos criar tabelas com nome de campo que usam nomes de tipos de dados ou funções criadas no MySQL, com `TIMESTAMP` ou `GROUP`, Você poderá fazer isso (por exemplo, `ABS` é um nome de campo permitido), mas espaços não são permitidos entre o nome da função e o `'` quando estiver usando este nome de função também como o nome de uma campo.

As seguintes palavras são explicitamente reservadas em MySQL. Muitas delas são proibidas pelo ANSI SQL92 como nomes de campos e/ou tabelas. (por exemplo, `group`). Algumas poucas são reservadas porque o MySQL precisa delas e está usando (atualmente) um analisador yacc:

<code>action</code>	<code>add</code>	<code>aggregate</code>	<code>all</code>
<code>alter</code>	<code>after</code>	<code>and</code>	<code>as</code>
<code>asc</code>	<code>avg</code>	<code>avg_row_length</code>	<code>auto_increment</code>
<code>between</code>	<code>bigint</code>	<code>bit</code>	<code>binary</code>
<code>blob</code>	<code>bool</code>	<code>both</code>	<code>by</code>
<code>cascade</code>	<code>case</code>	<code>char</code>	<code>character</code>
<code>change</code>	<code>check</code>	<code>checksum</code>	<code>column</code>
<code>columns</code>	<code>comment</code>	<code>constraint</code>	<code>create</code>
<code>cross</code>	<code>current_date</code>	<code>current_time</code>	<code>current_</code>
			<code>timestamp</code>
<code>data</code>	<code>database</code>	<code>databases</code>	<code>date</code>
<code>datetime</code>	<code>day</code>	<code>day_hour</code>	<code>day_minute</code>
<code>day_second</code>	<code>dayofmonth</code>	<code>dayofweek</code>	<code>dayofyear</code>
<code>dec</code>	<code>decimal</code>	<code>default</code>	<code>delayed</code>
<code>delay_key_write</code>	<code>delete</code>	<code>desc</code>	<code>describe</code>
<code>distinct</code>	<code>distinctrow</code>	<code>double</code>	<code>drop</code>
<code>end</code>	<code>else</code>	<code>escape</code>	<code>escaped</code>
<code>enclosed</code>	<code>enum</code>	<code>explain</code>	<code>exists</code>
<code>fields</code>	<code>file</code>	<code>first</code>	<code>float</code>
<code>float4</code>	<code>float8</code>	<code>flush</code>	<code>foreign</code>
<code>from</code>	<code>for</code>	<code>full</code>	<code>function</code>
<code>global</code>	<code>grant</code>	<code>grants</code>	<code>group</code>
<code>having</code>	<code>heap</code>	<code>high_priority</code>	<code>hour</code>
<code>hour_minute</code>	<code>hour_second</code>	<code>hosts</code>	<code>identified</code>
<code>ignore</code>	<code>in</code>	<code>index</code>	<code>infile</code>
<code>inner</code>	<code>insert</code>	<code>insert_id</code>	<code>int</code>
<code>integer</code>	<code>interval</code>	<code>int1</code>	<code>int2</code>
<code>int3</code>	<code>int4</code>	<code>int8</code>	<code>into</code>
<code>if</code>	<code>is</code>	<code>isam</code>	<code>join</code>
<code>key</code>	<code>keys</code>	<code>kill</code>	<code>last_insert_id</code>

leading	left	length	like
lines	limit	load	local
lock	logs	long	longblob
longtext	low_priority	max	max_rows
match	mediumblob	mediumtext	mediumint
middleint	min_rows	minute	minute_second
modify	month	monthname	myisam
natural	numeric	no	not
null	on	optimize	option
optionally	or	order	outer
outfile	pack_keys	partial	password
precision	primary	procedure	process
processlist	privileges	read	real
references	reload	regexp	rename
replace	restrict	returns	revoke
rlike	row	rows	second
select	set	show	shutdown
smallint	soname	sql_big_tables	sql_big_selects
sql_low_priority_	sql_log_off	sql_log_update	sql_select_limit
updates			
sql_small_result	sql_big_result	sql_warnings	straight_join
starting	status	string	table
tables	temporary	terminated	text
then	time	timestamp	tinyblob
tinytext	tinyint	trailing	to
type	use	using	unique
unlock	unsigned	update	usage
values	varchar	variables	varying
varbinary	with	write	when
where	year	year_month	zerofill

Os símbolos seguintes (da tabela acima) não são permitidos pela ANSI SQL mas permitidos pelo MySQL como nome de campos/tabelas. Isto ocorre porque alguns destes nomes são muito naturais e vários pessoas já o utilizaram.

- ACTION
- BIT
- DATE
- ENUM
- NO
- TEXT
- TIME
- TIMESTAMP

6.2 Tipos de Campos

MySQL suporta um certo número de tipos de campos que podem ser agrupados em três categorias: tipos numéricos, tipos de data e hora, e tipos string (caracteres). Esta seção primeiro

lhe dá uma visão geral dos tipos disponíveis e resume as exigências de armazenamento em cada tipo de coluna, também fornece uma descrição mais detalhada da propriedade dos tipos em cada categoria. A visão dada é propositalmente breve. As descrições mais detalhadas devem ser consultadas para informações adicionais sobre tipos de campo particulares como os formatos permitidos nos quais você pode especificar valores.

Os tipos de campos suportados pelo MySQL estão listados abaixo: As seguintes letras são usadas como código nas descrições:

M Indica o tamanho máximo do display. O tamanho máximo oficial do display é 255.

D Aplica aos tipos de ponto flutuante e indica o número de dígitos após o ponto decimal. O maior valor possível é 30, mas não pode ser maior que M-2.

Colchetes (‘[’ and ‘]’) indicam partes de tipos específicos que são opcionais

Note que se você especificar **ZEROFILL** para um campo MySQL automaticamente irá adicionar o atributo **UNSIGNED** ao campo.

TINYINT[(M)] [UNSIGNED] [ZEROFILL]

Um inteiro muito pequeno. A faixa deste inteiro com sinal é de -128 até 127. A faixa sem sinal é de 0 até 255.

SMALLINT[(M)] [UNSIGNED] [ZEROFILL]

Um inteiro pequeno. A faixa do inteiro com sinal é de -32768 até 32767. A faixa sem sinal é de 0 a 65535.

MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]

Um inteiro de tamanho médio. A faixa com sinal é de -8388608 a 8388607. A faixa sem sinal é de 0 to 16777215.

INT[(M)] [UNSIGNED] [ZEROFILL]

Um inteiro de tamanho normal. A faixa com sinal é de -2147483648 a 2147483647. A faixa sem sinal é de 0 a 4294967295.

INTEGER[(M)] [UNSIGNED] [ZEROFILL]

Este é um sinônimo para **INT**.

BIGINT[(M)] [UNSIGNED] [ZEROFILL]

Um inteiro grande. A faixa com sinal é de -9223372036854775808 a 9223372036854775807. A faixa sem sinal é de 0 a 18446744073709551615.

Existem algumas coisas sobre campos **BIGINT** sobre as quais você deve estar ciente:

- Como todas as operações aritméticas são feitas usando valores **BIGINT** ou **DOUBLE** com sinal, não devemos utilizar inteiros sem sinal maiores que 9223372036854775807 (63 bits) exceto com funções de bit! Se você fizer isto, alguns dos últimos dígitos no resultado podem estar errados por causa de erros de arredondamento na conversão de **BIGINT** para **DOUBLE**.
- Você pode armazenar valores inteiros exatos em um campo **BIGINT** armazenando-os como string, como ocorre nestes casos não haverá nenhuma representação intermediária dupla.

- ‘-’, ‘+’, e ‘*’ serão utilizados em cálculos aritméticos **BIGINT** quando ambos os argumentos forem valores do tipo **INTEGER**! Isto significa que se você multilicar dois inteiros grandes (ou obter resultados de funções que retornam inteiros) você pode obter resultados inesperados quando o resultado for maior que 9223372036854775807.

FLOAT(precisão) [ZEROFILL]

Um número de ponto flutuante. Não pode ser sem sinal. **precisão** pode ser ≤ 24 para um número de ponto flutuante de precisão simples e entre 25 e 53 para um número de ponto flutuante de dupla-precisão. Estes tipos são como os tipos **FLOAT** e **DOUBLE** descritos logo abaixo. **FLOAT(X)** tem o mesma faixa que os tipos correspondentes **FLOAT** e **DOUBLE**, mas o tamanho do display e número de casas decimais é indefinido.

Na versão 3.23 do MySQL, este é um verdadeiro valor de ponto flutuante. Em versões anteriores, **FLOAT(precisão)** sempre tem 2 casas decimais.

Note que o uso de **FLOAT** pode trazer alguns problemas inesperados como nos cálculos já que em MySQL todos são feitos com dupla-precisão. See [\(undefined\)](#) [No matching rows], page [\(undefined\)](#).

Esta sintaxe é fornecida para comptibilidade com ODBC.

FLOAT[(M,D)] [ZEROFILL]

Um numero de ponto flutuante pequeno (precisão simples). Não pode ser sem sinal. Os valores permitidos são de $-3.402823466E+38$ a $-1.175494351E-38$, 0 e de $1.175494351E-38$ a $3.402823466E+38$. O M é a largura do display e o D é o número de casas decimais. **FLOAT** sem um argumento ou com um argumento ≤ 24 tende a um numero de ponto flutuante de precisão simples.

DOUBLE[(M,D)] [ZEROFILL]

Um número de ponto flutuante de tamanho normal (dupla-precisão). Não pode ser sem sinal. Valores permitidos entre $-1.7976931348623157E+308$ e $-2.2250738585072014E-308$, 0 e entre $2.2250738585072014E-308$ e $1.7976931348623157E+308$. O M é a largura do display e o D é número de casa decimais. **DOUBLE** sem argumento ou **DOUBLE(X)** onde $25 \leq X \leq 53$ são números de ponto flutuante de dupla-precisão.

DOUBLE PRECISION[(M,D)] [ZEROFILL]**REAL[(M,D)] [ZEROFILL]**

Estes são sinônimos para **DOUBLE**.

DECIMAL[(M[,D])] [ZEROFILL]

Um número de ponto flutuante não empacotado. Não pode ser sem sinal. Se comporta como um campo **CHAR**: “não empacotado” significa que o número é armazenado como uma string, usando um caracter para cada dígito do valor. O ponto decimal e, para números negativos, o sinal de menos (‘-’), não são contados em M (mas é reservado espaço para isto). Se D for 0, os valores não terão ponto decimal ou parte fracionária. A faixa máxima do valor **DECIMAL** é a mesma do **DOUBLE**, mas a faixa atual para um campo **DECIMAL** dado pode ser limitado pela escolha de M e D.

Se D não for definido será considerado como 0. Se M não for definido é considerado como 10.

Note que na versão 3.22 do MySQL o argumento M tem que incluir o espaço necessário para o sinal é o ponto decimal.

NUMERIC(M,D) [ZEROFILL]

Este é um sinônimo para DECIMAL.

DATE

Uma data. A faixa suportada é entre '1000-01-01' e '9999-12-31'. MySQL mostra valores DATE no formato 'AAAA-MM-DD', mas permite a você a atribuir valores a campos DATE utilizando tanto strings quanto números. See [\[DATETIME\]](#), page [\[undefined\]](#).

DATETIME

Um combinação de hora e data. A faixa suportada é entre '1000-01-01 00:00:00' e '9999-12-31 23:59:59'. MySQL mostra valores DATETIME no formato 'AAAA-MM-DD HH:MM:SS', mas permite a você que atribuir valores a campos DATETIME utilizado strings ou números. See [\[undefined\]](#) [\[DATETIME\]](#), page [\[undefined\]](#).

TIMESTAMP [(M)]

Uma estampa de tempo. A faixa é entre '1970-01-01 00:00:00' e algum momento no ano 2037. MySQL mostra valores TIMESTAMP nos formatos YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD, ou YYMMDD, dependendo se M é 14 (ou não definido), 12, 8, ou 6, mas permite a você atribuir valores ao campo TIMESTAMP usando strings ou números. Um campo TIMESTAMP é útil para gravar a data e a hora em uma operação de INSERT or UPDATE porque é automaticamente definido a data e a hora da operação mais recente se você próprio não especificar um valor. Você também pode definir a data e a hora atual atribuindo ao campo um valor NULL. See [\[undefined\]](#) [\[Date and time types\]](#), page [\[undefined\]](#).

Um TIMESTAMP sempre é armazenado em 4 bytes. O argumento M só afeta como a coluna TIMESTAMP é mostrada.

Note que colunas do tipo TIMESTAMP(X) columns onde X é 8 ou 14 são apresentadas como números enquanto as outras colunas TIMESTAMP(X) são strings. Isto é apenas para assegurar que podemos eliminar e restaurar com segurança tabelas com estes tipos! See [\[undefined\]](#) [\[DATETIME\]](#), page [\[undefined\]](#).

TIME

Uma hora. A faixa é entre '-838:59:59' e '838:59:59'. MySQL mostra valores TIME no formato 'HH:MM:SS', mas permite a você atribuir valores para as colunas TIME usando strings ou números. See [\[undefined\]](#) [\[TIME\]](#), page [\[undefined\]](#).

YEAR[(2|4)]

Um ano no formato de 2 ou 4 dígitos (padrão são 4 dígitos). Os valores permitidos estão entre 1901 e 2155, 0000 no formato de 4 dígitos, e 1970-2069 se você estiver usando o formato de 2 dígitos (70-69). MySQL mostra valores

YEAR no formato **YYYY**, mas permie atribuir valores aos campos do tipo **YEAR** usando strings ou números. (O tipo **YEAR** é novo na versão 3.22 do MySQL). See [\(undefined\) \[YEAR\], page \(undefined\)](#).

[NATIONAL] CHAR(M) [BINARY]

Uma string de tamanho fixo que é sempre preenchida a direita com espaços até o tamanho especificado quando armazenado. A faixa de **M** é de 1 a 255 caracteres. Espaços extras são removidos quando o valor é recuperado. Valores **CHAR** são ordenados e comparados no modo caso insensitivo de acordo com o conjunto de caracteres padrão, a menos que a palavra chave **BINARY** seja utilizada.

NATIONAL CHAR (forma reduzida **NCHAR**) é o modo ANSI SQL de definir que um campo **CHAR** deve usar o conjunto de caracteres padrão. Isto é padrão no MySQL.

CHAR é uma simplificação para **CHARACTER**.

MySQL lhe permite criar um campo do tipo **CHAR(0)**. Isto é muito útil quando você precisa de comptibilidade com aplicativos antigos que dependem da existência de uma coluna, mas que, na verdade, não utiliza um valor. Isto também é muito bom quando você precisa de uma coluna que só pode receber 2 valores. Um **CHAR(0)**, que não é definido como um **NOT NULL**, só irá ocupar um bit e pode assumir 2 valores: **NULL** or **"**". See [\(undefined\) \[CHAR\], page \(undefined\)](#).

BIT

BOOL

CHAR Estes três são sinônimos para **CHAR(1)**.

[NATIONAL] VARCHAR(M) [BINARY]

Uma string de tamanho variável. **NOTA:** Espaços extras são removidos quando o caracter é armazenado (o que difere da especificação ANSI SQL). A faixa de **M** é de 1 a 255 characters. Valores **VARCHAR** são ordenados e comparados no modo caso insensitivo a menos que a palavra chave **BINARY** seja utilizada. See [\(undefined\) \[Silent column changes\], page \(undefined\)](#).

VARCHAR é uma simplificação para **CHARACTER VARYING**. See [\(undefined\) \[CHAR\], page \(undefined\)](#).

TINYBLOB

TINYTEXT

Um campo **BLOB** ou **TEXT** com tamanho máximo de 255 ($2^8 - 1$) caracteres. See [\(undefined\) \[Silent column changes\], page \(undefined\)](#). See [\(undefined\) \[BLOB\], page \(undefined\)](#).

BLOB

TEXT

Um campo **BLOB** ou **TEXT** com tamanho máximo de 65535 ($2^{16} - 1$) caracteres. See [\(undefined\) \[Silent column changes\], page \(undefined\)](#). See [\(undefined\) \[BLOB\], page \(undefined\)](#).

MEDIUMBLOB**MEDIUMTEXT**

Um campo BLOB ou TEXT com tamanho máximo de 16777215 ($2^{24} - 1$) caracteres. See [\[Silent column changes\]](#), page [\[BLOB\]](#). See [\[BLOB\]](#), page [\[BLOB\]](#).

LONGBLOB**LONGTEXT**

Um campo BLOB ou TEXT com tamanho máximo de 4294967295 ($2^{32} - 1$) caracteres. See [\[Silent column changes\]](#), page [\[BLOB\]](#). Note que como o protocolo cliente/servidor e tabelas MyISAM tem, atualmente, um limite de 16M por pacote de transmissão/registo de tabela, você ainda não pode utilizar toda a faixa de valores deste tipo. See [\[BLOB\]](#), page [\[BLOB\]](#).

ENUM('valor1', 'valor2', ...)

Uma enumeração. Um objeto string que só pode ter um valor, selecionado da lista de valores 'valor1', 'valor2', ..., NULL ou valor especial de erro "". Um ENUM pode ter um máximo de 65535 valores diferentes. See [\[ENUM\]](#), page [\[ENUM\]](#).

SET('valor1', 'valor2', ...)

Um conjunto. Um objeto string que pode ter zero ou mais valores, cada um deve ser selecionado da lista de valores 'valor1', 'valor2', Um SET pode ter até 64 membros. See [\[SET\]](#), page [\[SET\]](#).

6.2.1 Tipos Numéricos

MySQL suporta todos os tipos numéricos da ANSI/ISO SQL92. Estes tipos incluem o tipos de dados numéricos exatos (NUMERIC, DECIMAL, INTEGER, e SMALLINT), assim como o tipos de dados numéricos aproximados (FLOAT, REAL, e DOUBLE PRECISION). A palavra-chave INT é um sinônimo para INTEGER, e a palavra-chave DEC é um sinônimo para DECIMAL.

Os tipos NUMERIC e DECIMAL são implementados como o mesmo tipo pelo MySQL, como permitido pelo padrão SQL92. Eles são usados por valores para os quais é importante preservar a exatidão como, por exemplo, dados monetários. Quando é declarado um campo de algum desses tipos a precisão e a escala podem ser (e normalmente é) especificadas; por exemplo:

```
salario DECIMAL(9,2)
```

Neste exemplo, 9 (*precisão*) representa o número de dígitos decimais significantes que serão armazenados no valor, e 2 (*escala*) representa o número de dígitos que serão armazenados após o ponto decimal. Neste caso, no entanto, a faixa de valores que podem ser armazenados na coluna `salario` é de -9999999.99 a 9999999.99. (MySQL pode, na verdade, armazenar números acima de 9999999.99 neste campo porque ele não precisa armazenar o sinal para números positivos).

Em ANSI/ISO SQL92, a sintaxe DECIMAL(p) é equivalente a DECIMAL(p,0). Da mesma forma, a sintaxe DECIMAL é equivalente a DECIMAL(p,0), onde a implementação permite decidir o valor de p. MySQL ainda não suporta nenhuma dessas duas formas variantes dos tipos de dados DECIMAL/NUMERIC. Este, geralmente, não é um problema sério, já que

os principais benefícios destes tipos derivam da habilidade de controlar precisão e escala explicitamente.

Valores `DECIMAL` e `NUMERIC` são armazenados como strings, ao invés de um número de ponto-flutuante binário, para preservar a precisão decimal destes valores. Um caractere é usado para cada dígito, para o ponto decimal (se `escala > 0`), e para o sinal ‘-’ (para números negativos). Se `escala` é 0, valores `DECIMAL` e `NUMERIC` não contêm ponto decimal ou parte fracionária.

A faixa máxima dos valores `DECIMAL` e `NUMERIC` é a mesma do `DOUBLE`, mas a faixa real para um campo `DECIMAL` ou `NUMERIC` pode ser limitada pela `precisão` ou pela `escala` para uma dada coluna. Quando é atribuído a uma coluna um valor com mais dígitos após o ponto decimal do que o permitido especificado na `escala`, o valor é arredondado para aquela `escala`. Quando é atribuído um valor a uma coluna `DECIMAL` ou `NUMERIC` o qual excede a faixa determinada pelas `precisão` e `escala` especificada (ou padrão), MySQL armazena o valor correspondente ao final daquela faixa.

Como uma extensão do padrão ANSI/ISO SQL92, MySQL também suporta os tipos integrais `TINYINT`, `MEDIUMINT`, e `BIGINT` como listado nas tabelas abaixo. Outra extensão suportada pelo MySQL é especificar, opcionalmente, o tamanho do display de um valor inteiro entre parênteses seguindo o nome do tipo (por exemplo, `INT(4)`). Esta especificação opcional do tamanho é usada para preenchimento à esquerda do display de valores cujo tamanho é menor que o especificado para a coluna, mas não limita a faixa de valores que podem ser armazenados na coluna, nem o número de dígitos que serão mostrados para valores que excederem o tamanho especificado na coluna. Quando usados em conjunto com o atributo opcional de extensão `ZEROFILL`, o padrão do preenchimento de espaços é a substituição por zeros. Por exemplo, para uma coluna declarada com `INT(5) ZEROFILL`, o valor 4 é retornado como 00004. Note que se você armazenar valores maiores que a largura do display em uma coluna do tipo inteiro, você pode ter problemas quando o MySQL gerar tabelas temporárias para algum join complicado, já que nestes casos o MySQL acredita que os dados cabem na largura original da coluna.

Todos os tipos inteiros podem ter um atributo opcional (não-padrão) `UNSIGNED`. Valores sem sinal podem ser usados quando você permite apenas números positivos em uma coluna e você precisa de uma faixa de valores um pouco maior para a coluna.

O tipo `FLOAT` é usado para representar tipos de dados numéricos aproximados. O padrão ANSI/ISO SQL92 permite especificação opcional da precisão (mas não da faixa do expoente) em bits, após a palavra `FLOAT` e entre parênteses. A implementação MySQL também suporta esta especificação opcional de precisão. Quando `FLOAT` é usada para um tipo de coluna sem especificação de precisão, MySQL utiliza quatro bytes para armazenar os valores. Uma sintaxe variante também é suportada, com dois números entre parênteses após a palavra `FLOAT`. Com esta opção, o primeiro número continua a representar a quantidade de bytes necessária para armazenar o valor, e o segundo número especifica o número de dígitos a serem armazenados e mostrados após o ponto decimal (como com `DECIMAL` e `NUMERIC`). Quando é pedido ao MySQL para armazenar um número em uma coluna com mais dígitos decimais após o ponto decimal que o especificado para esta coluna, o valor é arredondado eliminando os dígitos extras quando armazenado.

Os tipos `REAL` e `DOUBLE PRECISION` não aceitam especificações de precisão. Como uma extensão do padrão ANSI/ISO SQL92, MySQL reconhece `DOUBLE` como um sinônimo para o

tipo `DOUBLE PRECISION`. Em contraste com a exigência do padrão de que a precisão do tipo `REAL` seja menor que aquele usado pelo `DOUBLE PRECISION`, MySQL implementa ambos como valores de ponto flutuante de 8 bits de dupla precisão (quando não estiver executando em “modo ANSI”). Para uma portabilidade máxima, códigos que requerem armazenamento de valores de dados numéricos aproximados usam `FLOAT` ou `DOUBLE PRECISION` sem especificação de precisão ou de números decimais.

Quando solicitado a armazenar um valor em uma coluna numérica que está fora da faixa permitida pelo tipo da coluna, o MySQL ajusta o valor ao limite da faixa permitida mais apropriado e armazena este valor.

Por exemplo, a faixa de uma coluna `INT` é de `-2147483648` a `2147483647`. Se você tentar inserir `-999999999` em uma coluna `INT`, o valor é ajustado para o limite mais baixo da faixa de valores e `-2147483648` é armazenado. Da mesma forma, se você tentar inserir `999999999`, `2147483647` será armazenado.

Se o campo `INT` é `UNSIGNED`, o tamanho da faixa do campo é o mesmo mas o limite passa a ser de 0 a `4294967295`. Se você tentar armazenar `-999999999` e `999999999`, os valores armazenados na coluna serão 0 e `4294967296`.

Conversões que ocorrem devido a ajustes são relatados como “avisos” para `ALTER TABLE`, `LOAD DATA INFILE`, `UPDATE`, e instruções `INSERT` multi-registros.

6.2.2 Tipos de Data e Hora

Os tipos de data e hora são `DATETIME`, `DATE`, `TIMESTAMP`, `TIME`, e `YEAR`. Cada um desses tipos tem uma faixa de valores legais, assim com um valor “zero” que é usado quando você especifica um valor ilegal. Note que o MySQL permite que você armazene certos valores de datas inexistentes, como `1999-11-31`. A razão para isto é que pensamos que é responsabilidade do aplicativo tratar das verificações de data, não do servidor SQL. Para fazer uma verificação ‘rápida’ de data, MySQL só checa se o mês está na faixa de 0-12 e o dia está na faixa de 0-31. As faixas acima são definidas desta forma porque MySQL lhe permite armazenar, em um campo `DATE` ou `DATETIME`, datas onde o dia ou o dia/mês são zero. Isto é extremamente útil para aplicativos que precisam armazenar uma data de nascimento na qual você não sabe a data exata. Nestes casos você simplesmente armazena a data como `1999-00-00` ou `1999-01-00`. (Você não pode esperar obter um valor correto para funções como `DATE_SUB()` ou `DATE_ADD` para datas como estas.)

Aqui estão algumas considerações para ter em mente quando estiver trabalhando com tipos de data e hora.

- MySQL recupera valores para um tipo de data ou hora dado em um formato padrão, mas ele tenta interpretar uma variedade de formatos para os valores fornecidos (por exemplo, quando você especifica um valor a ser atribuído ou comparado a um tipo de data ou hora). No entanto, só os formatos descritos na seção seguinte são suportados. É esperado que você forneça valores permitidos. Resultados imprevisíveis podem ocorrer se você usar outros formatos.
- Embora o MySQL tente interpretar valores em diversos formatos, ele sempre espera que a parte da data referente ao ano esteja mais a esquerda do valor. Datas devem ser dadas na ordem ano-mês-dia (por exemplo, `'98-09-04'`), ao invés das ordens mais usadas mês-dia-ano ou dia-mês-ano (por exemplo: `'09-04-98'`, `'04-09-98'`).

- MySQL converte automaticamente um tipo de data ou hora em um número se o valor é usado em um contexto numérico, e vice-versa.
- Quando o MySQL encontra um valor para um tipo de data ou hora que está fora da faixa permitida ou é ilegal neste tipo (veja o início desta seção), ele converte o valor para “zero”. (A exceção ocorre no campo `TIME`, onde o valor fora da faixa é ajustado para o valor limite apropriado na faixa de valores deste tipo.) A tabela abaixo mostra o formato do valor “zero” para cada tipo:

Tipo de Coluna	Valor “Zero”
<code>DATETIME</code>	'0000-00-00 00:00:00'
<code>DATE</code>	'0000-00-00'
<code>TIMESTAMP</code>	00000000000000 (tamanho depende do tamanho do display)
<code>TIME</code>	'00:00:00'
<code>YEAR</code>	0000

- Os valores “zero” são especiais, mas você pode armazenar ou fazer referência a eles explicitamente usando os valores mostrados na tabela. Você também pode fazer `into` usando '0' ou 0, o que é mais fácil de escrever.
- Valores “zero” para data ou hora usados em **MyODBC** são convertidos automaticamente para `NULL` na versão 2.50.12 **MyODBC** e acima, porque ODBC não pode tratar tais valores.

6.2.2.1 Assuntos referentes ao ano 2000 (Y2K) e Tipos de Data

O MySQL tem sua própria segurança para o ano 2000 (see [\[compatibilidade com o ano 2000\]](#), page [\[compatibilidade com o ano 2000\]](#)), mas os dados entrados no MySQL podem não ter. Qualquer entrada contendo valores de ano de 2 dígitos é ambíguo, porque o século é desconhecido. Tais valores devem ser interpretados na forma de 4 dígitos já que o MySQL armazena anos internamente utilizando 4 dígitos.

Para tipos `DATETIME`, `DATE`, `TIMESTAMP` e `YEAR`, MySQL interpreta datas com valores ambíguos para o ano usando as seguintes regras:

- Valores de ano na faixa 00-69 são convertidos para 2000-2069.
- Valores de anos na faixa 70-99 são convertidos para 1970-1999.

Lembre-se de que essas regras fornecem apenas palpites razoáveis sobre o que a sua data significa. Se a heurística usada pelo MySQL não produz o valor você deve fornecer entre sem ambiguidade contendo valores de ano de 4 dígitos.

`ORDER BY` irá ordenar tipos `YEAR/DATE/DATETIME` de 2 dígitos apropriadamente.

Note também que algumas funções com `MIN()` e `MAX()` irão converter `TIMESTAMP/DATE` para um número. Isto significa que um timestamp com ano de 2 dígitos não irá funcionar corretamente com estas funções. A solução neste caso é converter o `TIMESTAMP/DATE` para um formato de ano de 4 dígitos ou usar algo como `MIN(DATE_ADD(timestamp, INTERVAL 0 DAYS))`.

6.2.2.2 Os Tipos `DATETIME`, `DATE` e `TIMESTAMP`

Os tipos `DATETIME`, `DATE`, e `TIMESTAMP` são relacionados. Esta seção descreve suas características, como eles se assemelham ou como se diferem.

O tipo `DATETIME` é usado quando você precisa de valores que contém informações sobre data e a hora. MySQL recupera e mostra valores `DATETIME` no formato `'YYYY-MM-DD HH:MM:SS'`. A faixa suportada é de `'1000-01-01 00:00:00'` até `'9999-12-31 23:59:59'`. (“Suportada” significa que embora valores anteriores possam funcionar, não há nenhuma garantia de disto.)

O tipo `DATA` é usado quando se necessita apenas do valor da data, sem a parte da hora. MySQL recupera e mostra valores do tipo `DATA` no formato `'YYYY-MM-DD'`. A faixa suportada é de `'1000-01-01'` até `'9999-12-31'`.

O tipo de campo `TIMESTAMP` fornece um tipo que pode ser usado para, automaticamente, marcar operações `INSERT` or `UPDATE` com a data e hora atual. Se você tiver múltiplas colunas `TIMESTAMP`, só a primeira é atualizada automaticamente.

Atualizações automáticas da primeira coluna `TIMESTAMP` ocorrem sob qualquer uma das seguintes condições:

- A coluna não é explicitamente especificada em uma instrução `INSERT` ou `LOAD DATA INFILE`.
- A coluna não é explicitamente especificada em uma instrução `UPDATE` e e alguma outra coluna muda o valor. (Note que um `UPDATE` que coloca em uma coluna o mesmo valor que ele já possui não irá causar a atualização da coluna `TIMESTAMP`, porque se você atribui a uma coluna o seu valor atual, MySQL ignora a atualização para maior eficiência).
- Você define explicitamente a uma coluna `TIMESTAMP` o valor `NULL`.

Outras colunas `TIMESTAMP`, além da primeira podem ser definidas com a data e hora atuais. Basta defini-las com `NULL` ou `NOW()`

Você pode definir colunas `TIMESTAMP` com um valor diferente da data e hora atuais colocando explicitamente o valor desejado. Isto é verdade mesmo para a primeira coluna `TIMESTAMP`. Você pode usar esta propriedade se, por exemplo, você quiser que um `TIMESTAMP` tenha seu valor definido como a data e hora atuais na criação de registros, mas não quer alterá-los quando o registro for atualizado mais tarde:

- Deixe o MySQL definir a coluna quando o registro é criado. Isto irá inicializa-la com a data e hora atuais.
- Quando você realizar subseqüentes atualizações em outras colunas do registro, defina explicitamente a coluna `TIMESTAMP` com o valor atual.

Por outro lado, você pode achar que é mais fácil usar uma coluan `DATETIME` que você inicializa com `NOW()` quando o registro for criado e deixa como está em atualizações subseqüentes. Valores `TIMESTAMP` podem ter valores do incio de 1970 até algum momento do ano 2037, com a resolução de um segundo. Valores são mostrados como números

O formato no qual o MySQL recupera e mostra valores `TIMESTAMP` depende do tamanho do display, como ilustrado pela tabela que se segue: O formato ‘cheio’ `TIMESTAMP` é de 14 dígitos, mas colunas `TIMESTAMP` podem ser criadas com tamanho de display menores:

Tipo da Coluna	Formato do Display
<code>TIMESTAMP(14)</code>	<code>YYYYMMDDHHMMSS</code>
<code>TIMESTAMP(12)</code>	<code>YYMMDDHHMMSS</code>
<code>TIMESTAMP(10)</code>	<code>YYMMDDHHMM</code>

TIMESTAMP(8)	YYYYMMDD
TIMESTAMP(6)	YYMMDD
TIMESTAMP(4)	YYMM
TIMESTAMP(2)	YY

Todas as colunas `TIMESTAMP` tem o mesmo tamanho de armazenamento, independente do tamanho de display. Os tamanhos de display mais comuns são 6, 8, 12, e 14. Você pode especificar um tamanho de display arbitrário na hora da criação da tabela, mas valores de 0 ou maiores que 14 são mudados para 14. Valores ímpares de tamanho na faixa de 1 a 13 são mudados para o maior número par mais próximo.

Nota: Na versão 4.1, `TIMESTAMP` é retornado com uma string com o formato `'YYYY-MM-DD HH:MM:DD'`, e timestamp de diferentes tamanhos não são mais suportados.

Você pode especificar valores `DATETIME`, `DATE` e `TIMESTAMP` usando qualquer conjunto de formatos comum:

- Como uma string nos formatos `'YYYY-MM-DD HH:MM:SS'` ou `'YY-MM-DD HH:MM:SS'`. Uma sintaxe “relaxada” é permitida---nenhum caractere de pontuação pode ser usado como um delimitador entre parte de data ou hora. Por exemplo, `'98-12-31 11:30:45'`, `'98.12.31 11+30+45'`, `'98/12/31 11*30*45'`, e `'98@12@31 11^30^45'` são equivalentes.
- Como uma string nos formatos `'YYYY-MM-DD'` ou `'YY-MM-DD'`. Uma sintaxe “relaxada” é permitida aqui também. Por exemplo, `'98-12-31'`, `'98.12.31'`, `'98/12/31'`, e `'98@12@31'` são equivalentes.
- Como uma string sem delimitadores nos formatos `'YYYYMMDDHHMMSS'` ou `'YYMMDDHHMMSS'`, desde que a string faça sentido como data. Por exemplo, `'19970523091528'` e `'970523091528'` são interpretadas como `'1997-05-23 09:15:28'`, mas `'971122129015'` é ilegal (tem uma parte de minutos sem sentido) e se torna `'0000-00-00 00:00:00'`.
- Como uma string sem delimitadores nos formatos `'YYYYMMDD'` ou `'YYMMDD'`, desde que a string tenha sentido como data. Por exemplo, `'19970523'` e `'970523'` são interpretadas como `'1997-05-23'`, mas `'971332'` é ilegal (tem uma parte de mês sem sentido) e se torna `'0000-00-00'`.
- Como um número nos formatos `YYYYMMDDHHMMSS` ou `YYMMDDHHMMSS`, desde que o número faça sentido como uma data. Por exemplo, `19830905132800` e `830905132800` são interpretados como `'1983-09-05 13:28:00'`.
- Como um número nos formatos `YYYYMMDD` ou `YYMMDD`, desde que o número faça sentido como data. Por exemplo, `19830905` e `830905` são interpretados como `'1983-09-05'`.
- Como o resultado de uma função que retorne um valor aceitável em um contexto `DATETIME`, `DATE` ou `TIMESTAMP`, tal como `NOW()` ou `CURRENT_DATE`.

Valores `DATETIME`, `DATE`, ou `TIMESTAMP` ilegais são convertidos para o valor “zero” do tipo apropriado (`'0000-00-00 00:00:00'`, `'0000-00-00'`, ou `0000000000000000`).

Para valores especificados com strings que incluem delimitadores de data, não é necessário especificar dois dígitos para valores de mês ou dia que são menores que 10. `'1979-6-9'` é o mesmo que `'1979-06-09'`. Similarmente, para valores especificados como strings que incluem delimitadores de hora, não é necessário especificar dois dígitos para valores de

hora, minutos ou segundo que são menores que 10. '1979-10-30 1:2:3' Ré o mesmo que '1979-10-30 01:02:03'.

Valores especificados como números devem ter 6, 8, 12, ou 14 dígitos. Se o número é de 8 ou 14 dígitos, ele assume estar no formato YYYYMMDD ou YYYYMMDDHHMMSS e que o ano é dado pelos 4 primeiros dígitos. Se o é de 6 ou 12 dígitos, ele assume estar no formato YYMMDD or YYMMDDHHMMSS e que o ano é dado pelos 2 primeiros dígitos. Números que não possuam estes tamanhos são interpretados como valores preenchidos com zero até o tamanho mais próximo.

Valores especificados como strings não delimitadas são interpretados usando o seu tamanho como dado. Se a string possui 8 ou 14 caracteres, o ano é assumido como os 4 primeiros caracteres. De outra forma o assume-se que o ano são os 2 primeiros caracteres. A string é interpretada da esquerda para a direita para encontrar os valores do ano, mês, dia, hora, minuto e segundo, para as partes da string. Isto significa que você não deve utilizar strings com menos de 6 caracteres. Por exemplo, se você especificar '9903', pensando em representar Março de 1999, você perceberá que o MySQL insere uma data "zero" em sua tabela. Isto ocorre porque os valores do ano e mês são 99 e 03, mas a parte contendo o dia não existe (zero), então o valor não é uma data legal.

Colunas **TIMESTAMP** armazenam valores legais utilizando precisão total com a qual os valores foram especificados, independente do tamanho do display. Isto tem diversas implicações:

- Sempre especifique o ano, mês e dia, mesmo se seus tipos de coluna são **TIMESTAMP(4)** ou **TIMESTAMP(2)**. De outra forma, os valores não serão datas legais e um 0 será armazenado.
- Se você usa **ALTER TABLE** para aumentar uma coluna **TIMESTAMP**, informações serão mostradas como se antes estivessem "escondidas".
- De forma similar, reduzindo o tamanho de uma coluna **TIMESTAMP** não causa perda de informação, exceto no sentido de que menos informação aparece quando os valores são mostrados.
- Embora os valores **TIMESTAMP** sejam armazenados com precisão total, a única função que opera diretamente com o valor armazenado é **UNIX_TIMESTAMP()**. Outras funções operam com o formato do valor recuperado. Isto significa que não se pode usar funções como **HOUR()** or **SECOND()** a menos que a parte relevante do valor **TIMESTAMP** esteja incluído no valor formatado. Por exemplo, a parte **HH** de uma coluna **TIMESTAMP** não é mostrada a menos que o tamanho do display seja de pelo menos 10, logo tentar usar **HOUR()** em um valor **TIMESTAMP** menor produz um resultado sem significado.

Você pode, algumas vezes, atribuir valores de um tipo de data para um objeto de um diferente tipo de data. No entanto pode haver algumas alterações de valores ou perda de informação

- Se você atribuir um valor de **DATE** a um objeto **DATETIME** ou **TIMESTAMP**, a parte da hora do valor resultante é definido como '00:00:00', porque o valor **DATE** não contém informações de hora.
- Se você atribuir um valor **DATETIME** ou **TIMESTAMP** para um objeto **DATE**, a parte da hora do valor resultante é deletado, pois o tipo **DATE** não armazena informações de hora.
- Lembre-se de que embora todos os valores **DATETIME**, **DATE**, e **TIMESTAMP** possam ser especificados usando o mesmo conjunto de formatos, os tipos não tem a mesma faixa de

valores. Por exemplo, valores `TIMESTAMP` não podem ser anteriores a 1970 ou posteriores a 2037. Isto significa que datas como `'1968-01-01'`, são permitidas como valores `DATETIME` ou `DATE`, mas não são válidas para valores `TIMESTAMP` e serão convertidas para 0 se atribuídas para tais objetos.

Esteja ciente de certas dificuldades quando especificar valores de data:

- A forma “relaxada” permitida em valores especificados com strings podem causar certas confusões. Por exemplo, um valor como `'10:11:12'` pode parecer com um valor de hora devido ao limitador `:`, mas se usado em um contexto de data será interpretado como o ano `'2010-11-12'`. O valor `'10:45:15'` será convertido para `'0000-00-00'` pois `'45'` não é um valor de mês permitido.
- O servidor MySQL funciona basicamente checando a validade da data: dias entre 00-31, mês entre 00-12, anos entre 1000-9999. Qualquer data que não esteja nesta faixa será revertida para `0000-00-00`. Por favor, note que isto ainda lhe permite armazenar datas inválidas tais como `2002-04-31`. Isto permite a aplicações web armazenar dados de um formulário sem verificações adicionais. Para assegurar que a data é válida, faça a checagem em sua aplicação.
- Valores de anos especificados com 2 dígitos são ambíguos, pois o século não é conhecido. MySQL interpreta valores de anos com dois dígitos usando as seguintes regras:
 - Valores de ano na faixa de 00-69 são convertidos para 2000-2069.
 - Valores de ano na faixa de 70-99 são convertidos para 1970-1999.

6.2.2.3 O tipo TIME

O MySQL recupera e mostra valores `TIME` no formato `'HH:MM:SS'` (ou no formato `'HHH:MM:SS'` para valores grandes). Valores `TIME` podem estar na faixa de `'-838:59:59'` até `'838:59:59'`. A razão para a parte da hora ser tão grande é que o tipo `TIME` pode ser usado não apenas para representar a hora do dia (que deve ser menor que 24 horas), mas também para tempo restante ou intervalos de tempo entre dois eventos (que podem ser maior que 24 horas ou mesmo negativo).

Você pode especificar valores `TIME` de variadas formas:

- Como uma string no formato `'D HH:MM:SS.fração'`. (Note que o MySQL não armazena ainda frações para a coluna `time`.) Pode-se também utilizar uma das seguintes sintaxes “relaxadas”:
`HH:MM:SS.fração`, `HH:MM:SS`, `HH:MM`, `D HH:MM:SS`, `D HH:MM`, `D HH` ou `SS`. Aqui `D` é um dia entre 0-33.
- Como uma string sem delimitadores no formato `'HHMMSS'`, desde que ela tenha sentido como uma hora. Por exemplo, `'101112'` é entendido como `'10:11:12'`, mas `'109712'` é ilegal (a parte dos minutos não tem nenhum sentido) e se torna `'00:00:00'`.
- Como um número no formato `HHMMSS`, desde que tenha sentido como uma hora. Por exemplo, `101112` é entendido com `'10:11:12'`. Os formatos alternativos seguintes também são entendidos: `SS`, `MMSS`, `HHMMSS` e `HHMMSS.fração`. Note que o MySQL ainda não armazena frações.
- Como o resultado de uma função que retorne um valor que é aceitável em um contexto do tipo `TIME`, tal como `CURRENT_TIME`.

Para valores TIME especificados como uma string que incluem delimitadores de hora, não é necessário especificar dois dígitos para valores de hora, minutos ou segundos que sejam menores que 10. '8:3:2' é o mesmo que '08:03:02'.

Seja cuidadoso ao atribuir valores TIME “pequenos” para uma coluna TIME. Sem dois pontos, o MySQL interprete valores assumindo que os dígitos mais a direita representam segundos. (MySQL interpreta valores TIME como tempo decorrido ao invés de hora do dia.) Por exemplo, você poderia pensar em '1112' e 1112 significam '11:12:00' (11 horas e 12 minutos), mas o MySQL o interpreta como '00:11:12' (onze minutos e 12 segundos). De forma similar, '12' e 12 são interpretados como '00:00:12'. Valores TIME com dois pontos, em contrapartida, são tratados como hora do dia. Isto é, '11:12' significará '11:12:00', não '00:11:12'.

Valores que são legais mas que estão fora da faixa permitidas são ajustados para o valor limita da faixa mais apropriado. Por exemplo, '-850:00:00' e '850:00:00' são convertidos para '-838:59:59' e '838:59:59', respectivamente.

Valores TIME ilegais são convertidos para '00:00:00'. Note que como '00:00:00' é um valor TIME, não temos com dizer, a partir de um valor '00:00:00' armazenado na tabela, se o valor original armazenado foi especificado como '00:00:00' ou se foi ilegal.

6.2.2.4 The YEAR Type

O tipo YEAR é um tipo de 1 byte usado para representar anos.

O MySQL recupera e mostra valores YEAR no formato YYYY. A faixa de valores é de 1901 até 2155.

Você pode especificar valores YEAR em uma variedade de formatos:

- Como uma string de 4 dígitos na faixa de '1901' até '2155'.
- Como um número de 4 dígitos na faixa de 1901 até 2155.
- Como uma string de dois dígitos na faixa '00' até '99'. Valores na faixa de '00' até '69' e '70' até '99' são convertidas para valores YEAR na faixa de 2000 até 2069 e 1970 até 1999.
- Como um número de 2 dígitos na faixa de 1 até 99. Valores na faixa de 1 até 69 e 70 até 99 são convertidos para valores YEAR na faixa de 2001 até 2069 e 1970 até 1999. Note que a faixa para números de dois dígitos é um pouco diferente da faixa de strings de dois dígitos, pois não se pode especificar zero diretamente como um número e tê-lo interpretado com 2000. Você **deve** especificá-lo como uma string '0' ou '00' ou ele será interpretado com 0000.
- Como o resultado de uma função que retorna um valor que é aceitável em um contexto do tipo YEAR, tal como NOW().

Valores YEAR ilegais são convertidos para 0000.

6.2.3 Tipos String

Os tipos strings são CHAR, VARCHAR, BLOB, TEXT, ENUM, e SET. Esta seção descreve como este tipos funcionam, suas exigências de armazenamento e como usá-los em suas consultas.

Tipo	Tam.maxímo	Bytes
------	------------	-------

TINYTEXT ou TINYBLOB	2 ⁸ -1	255
TEXT ou BLOB	2 ¹⁶ -1 (64K-1)	65535
MEDIUMTEXT ou MEDIUMBLOB	2 ²⁴ -1 (16M-1)	16777215
LOB	2 ³² -1 (4G-1)	4294967295

6.2.3.1 Os tipos CHAR e VARCHAR

Os tipos CHAR e VARCHAR são parecidos, mas diferem no modo como são armazenados e recuperados.

O tamanho de um campo CHAR é fixado pelo tamanho declarado na criação da tabela. O tamanho pode ser qualquer valor entre 1 e 255 (Como na versão 3.23 do MySQL, o tamanho pode ser de 0 a 255). Quando valores CHAR são armazenados, eles são preenchidos a direita com espaços até o tamanho especificado. Quando valores CHAR são recuperados, espaços extras são removidos.

Valores no campo VARCHAR são strings de tamanho variável. Você pode declarar um campo VARCHAR para ter qualquer tamanho entre 1 e 255, assim como para campo CHAR. No entanto, diferente de CHAR, valores VARCHAR são armazenados usando apenas quantos caracteres forem necessários, mais 1 byte para gravar o tamanho. Valores não são preenchidos; ao contrário, espaços extras são removidos quando valores são armazenados. (Esta remoção de espaços difere das especificações do SQL-99).

Se você atribuir um valor para uma coluna CHAR ou VARCHAR que exceda o tamanho máximo da coluna, o valor é truncado para este tamanho.

A seguinte tabela ilustra as diferenças entre os dois tipos de colunas, mostrando o resultado de se armazenar vários valores de strings em campos CHAR(4) e VARCHAR(4):

Valor	CHAR(4)	Exigência p/ armazenamento	VARCHAR(4)	Exigência p/ armazenamento
''	' '	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

Os valores recuperados para as colunas CHAR(4) e VARCHAR(4) serão os mesmos em cada caso, já que espaços extras são removidos das colunas CHAR quando recuperados.

Valores nas colunas CHAR e VARCHAR são ordenados e comparados no modo caso-insensitivo, a menos que o atributo BINARY seja especificado quando a tabela for criada. O atributo BINARY significa que os valores das colunas são ordenados e comparados no modo caso-sensitivo de acordo com a ordem ASCII da máquina onde o servidor MySQL está sendo executado. BINARY não afeta como as colunas são armazenadas e recuperadas.

O atributo BINARY é pegajoso. Isto significa que se uma coluna definida com BINARY é usada na expressão, toda a expressão é comparada como um valor BINARY.

MySQL pode alterar sem aviso o tipo de uma coluna CHAR ou VARCHAR na hora de criar a tabela. See [\[Silent column changes\]](#), page [\[Silent column changes\]](#).

6.2.3.2 Os Tipos BLOB e TEXT

Um BLOB é um objeto binário grande que pode guardar um montante variado de dados. Os quatro tipos BLOB: TINYBLOB, BLOB, MEDIUMBLOB, e LOB diferem apenas

no tamanho máximo dos valores que eles podem guardar. See [\[Storage requirements\]](#), page [\[Storage requirements\]](#).

Os quatro tipos `TEXT`: `TINYTEXT`, `TEXT`, `MEDIUMTEXT`, e `LONGTEXT` correspondem aos quatro tipos `BLOB` e têm o mesmo tamanho máximo e necessidade de tamanho para armazenamento. A única diferença entre os tipos `BLOB` e `TEXT` é que ordenação e comparação são realizadas no modo caso-sensitivo para valores `BLOB` e no modo caso-insensitivo para valores `TEXT`. Em outras palavras, um `TEXT` é um `BLOB` no modo caso-insensitivo.

Se você atribuir um valor a uma coluna `BLOB` ou `TEXT` que exceda o tamanho máximo do tipo da coluna, o valor é truncado para servir ao campo.

Em muitos casos, podemos considerar um campo `TEXT` como um campo `VARCHAR` que pode ser tão grande quando desejamos. Da mesma forma podemos considerar um campo `BLOB` como um campo `VARCHAR BINARY`. As diferenças são:

- Você pode ter índices em um campo `BLOB` e `TEXT` no MySQL Versão 3.23.2 e mais novas. Versões antigas do MySQL não suportam isto.
- Não há remoção de espaços extras para campos `BLOB` e `TEXT` quando os valores são armazenados, como há em campos `VARCHAR`.
- Colunas `BLOB` e `TEXT` não podem ter valores padrões.

MyODBC define valores `BLOB` como `LONGVARBINARY` e valores `TEXT` como `LONGVARCHAR`.

Como valores `BLOB` e `TEXT` podem ser extremamente longos, você pode deparar com alguns problemas quando utilizá-los:

- Se você quiser utilizar `GROUP BY` ou `ORDER BY` em um campo `BLOB` ou `TEXT`, você deve convertê-los em objetos de tamanho fixo. O modo padrão de se fazer isto é com a função `SUBSTRING`. Por exemplo:

```
mysql> SELECT comentario FROM nome_tabela, SUBSTRING(comentario,20) AS substr
-> ORDER BY substr;
```

Se você não fizer isto, só os primeiros `max_sort_length` bytes de uma coluna serão utilizados na ordenação. O valor padrão de `max_sort_length` é 1024; este valor pode ser alterado utilizando-se a opção `-O` quando o servidor é inicializado. Você pode agrupar uma expressão envolvendo valores `BLOB` ou `TEXT` especificando a posição da coluna ou utilizando apelidos (alias):

```
mysql> SELECT id, SUBSTRING(col_blob,1,100) FROM nome_tabela GROUP BY 2;
mysql> SELECT id, SUBSTRING(col_blob,1,100) AS b FROM nome_tabela GROUP BY b;
```

- O tamanho máximo de um objeto `BLOB` ou `TEXT` é determinado pelo seu tipo, mas o maior valor que você pode, atualmente, transmitir entre o cliente e o servidor é determinado pela quantidade de memória disponível e o tamanho dos buffers de comunicação. Você pode mudar o tamanho do buffer de mensagem, mas você deve fazê-lo no servidor e no cliente. See [\[Server parameters\]](#), page [\[Server parameters\]](#).

Note que cada valor `BLOB` ou `TEXT` é representado internamente por um objeto alocado separadamente. Está é uma diferença com todos os outros tipos de colunas, para o qual o armazenamento é alocado um por coluna quando a tabela é aberta.

6.2.3.3 O Tipo ENUM

Um **ENUM** é um objeto string cujo valor normalmente é escolhido de uma lista de valores permitidos que são enumerados explicitamente na especificação da coluna na criação da tabela.

O valor pode ser a string vazia ("") ou **NULL** sob certas circunstâncias:

- Se você inserir um valor inválido em um **ENUM** (isto é, uma string que não está presente na lista de valores permitidos), a string vazia é inserida no lugar como um valor especial de erro. Esta string pode se diferenciar de um string vazia 'norma' pelo fato de que esta string tem o valor numérico 0. Veremos mais sobre este assunto mais tarde.
- Se um **ENUM** é declarado **NULL**, **NULL** é também um valor permitido para a coluna, e o valor padrão é **NULL**. Se um **ENUM** é declarado **NOT NULL**, o valor padrão é o primeiro elemento da lista de valores permitidos.

Cada enumeração tem um índice:

- Valores da lista de elementos permitidos na especificação da coluna são números começados com 1.
- O valor de índice de uma string vazia que indique erro é 0. Isto significa que você pode usar a seguinte instrução **SELECT** para encontrar linhas nas quais valores **ENUM** inválidos foram atribuídos:

```
mysql> SELECT * FROM nome_tabela WHERE col_enum=0;
```

- O índice de um valor **NULL** é **NULL**.

Por exemplo, uma coluna especificada como **ENUM("um", "dois", "três")** pode ter qualquer um dos valores mostrados aqui. O índice de cada valor também é mostrado:

Valor	Índice
NULL	NULL
"	0
"um"	1
"dois"	2
"três"	3

Uma enumeração pode ter um máximo de 65535 elementos.

A partir da versão 3.23.51 espaços extras são automaticamente deletados dos valores **ENUM** quando a tabela é criada.

O caso da letra é irrelevante quando você atribui valores a uma coluna **ENUM**. No entanto, valores recuperados posteriormente da coluna terá o caso de letras de acordo com os valores que foram usados para especificar os valores permitidos na criação da tabela.

Se você recupera um **ENUM** em um contexto numérico, o índice do valor da coluna é retornado. Por exemplo, você pode recuperar valores numéricos de uma coluna **ENUM** desta forma:

```
mysql> SELECT col_enum+0 FROM nome_tabela;
```

Se você armazena um número em um **ENUM**, o número é tratado como um índice, e o valor armazenado é o membro da enumeração com este índice. (No entanto, este não irá funcionar com **LOAD DATA**, o qual trata todas as entradas como strings.) Não é aconselhável armazenar números em uma string **ENUM** pois pode tornar as coisas um pouco confusas.

Valores `ENUM` são armazenados de acordo com a ordem na qual os membros da enumeração foram listados na especificação da coluna. (Em outras palavras, valores `ENUM` são ordenados de acordo com o seus números de índice.) Por exemplo, "a" vem antes de "b" para `ENUM("a", "b")`, mas "b" vem antes de "a" para `ENUM("b", "a")`. A string vazia vem antes de strings não-vazias, e valores `NULL` vem antes de todos os outros valores de enumeração. Para evitar resultados inesperados, especifique a lista `ENUM` em ordem alfabética. Você também pode usar `GROUP BY CONCAT(col)` para ter certeza de que as colunas estão ordenadas alfabeticamente e não pelo índice numérico.

Se você quiser obter todos os valores possíveis para uma coluna `ENUM`, você deve usar: `SHOW COLUMNS FROM nome_tabela LIKE nome_coluna_enum` e analisar a definição de `ENUM` na segunda coluna.

6.2.3.4 O Tipo SET

Um `SET` é um objeto string que pode ter zero ou mais valores, cada um deve ser escolhido de uma lista de valores permitidos especificados quando a tabela é criada. Valores de colunas `SET` que consistem de múltiplos membros são especificados separados por vírgula (','). Uma consequência disto é que valores dos membros de `SET` não podem, eles mesmos, conter vírgula.

Por exemplo, uma coluna especificada como `SET("um", "dois") NOT NULL` pode ter qualquer um destes valores:

```
""
"um"
"dois"
"um, dois"
```

Um `SET` pode ter no máximo 64 membros diferentes.

A partir da versão 3.23.51, espaços extras são automaticamente removidos dos valores de `SET` quando a tabela é criada.

MySQL armazena valores `SET` numericamente, com o bit de baixa-ordem do valor armazenado correspondendo ao primeiro membro do conjunto. Se você recupera um valor `SET` em um contexto numérico, o valor recuperado tem o conjunto de bits correspondente aos membros que aparecem no valor da coluna. Por exemplo, você pode recuperar valores numéricos de uma coluna `SET` assim:

```
mysql> SELECT col_set+0 FROM nome_tabela;
```

Se um número é armazenado em uma coluna `SET`, os bits que estão habilitados (com 1) na representação binária do número determinam o qual o membro no valor da coluna. Suponha uma coluna especificada como `SET("a", "b", "c", "d")`. Então os membros terão os seguintes valores binários:

SET membro	Valor decimal	Valor binário
a	1	0001
b	2	0010
c	4	0100
d	8	1000

Se você atribuir um valor 9 a esta coluna, que é 1001 em binário, o primeiro e o quarto valores membros do `SET` "a" e "d" são selecionados e o valor resultante é "a,d".

Para um valor contendo mais que um elemento de SET, não importa em qual ordem os elementos são listados quando foram inseridos seus valores. Também não importa quantas vezes um dado elemento é listado no valor. Quando o valor é recuperado posteriormente, cada elemento aparecerá uma vez, listados de acordo com a ordem em que eles foram especificados na criação da tabela. Por exemplo, se uma coluna é especificada como SET("a","b","c","d"), então "a,d", "d,a" e "d,a,a,d,d" irão todos aparecer como "a,d" quando recuperados.

Se você define um valor que não é suportado pela coluna SET, o valor será ignorado.

Valores SET são ordenados numericamente. Valores NULL vêm antes de valores SET não NULL.

Normalmente, você realiza um SELECT em uma coluna SET usando o operador LIKE ou a função FIND_IN_SET():

```
mysql> SELECT * FROM nome_tabela WHERE col_set LIKE '%valor%';
mysql> SELECT * FROM nome_tabela WHERE FIND_IN_SET('valor',col_set)>0;
```

Mas o seguinte também funciona:

```
mysql> SELECT * FROM nome_tabela 2 WHERE col_set = 'val1,val2';
mysql> SELECT * FROM nome_tabela 3 WHERE col_set & 1;
```

A primeira desta instruções procura por uma correspondência exata. A segunda por valores contendo o primeiro membro.

Se você quer obter todos os valores possíveis para uma coluna SET, você deve usar: SHOW COLUMNS FROM nome_tabela LIKE nome_coluna_set e analisar a definição do SET na segunda coluna.

6.2.4 Escolhendo o Tipo Certo para um Campo

Para um uso mais eficiente do armazenamento, tente usar o tipo mais adequado em todos os casos. Por exemplo, se um campo de inteiro for usado para valores em uma faixa entre 1 e 99999, MEDIUMINT UNSIGNED é o melhor tipo.

Repretação precisa de valores monetários é um problema comum. No MySQL você deve usar o tipo DECIMAL. Ele armazena uma string, então nenhuma perda de precisão deve ocorrer. Se a precisão não é tão importante, o tipo DOUBLE pode ser satisfatório.

Para uma alta precisão você sempre pode converter para um tipo de ponto fixo armazenado em um BIGINT. Isto permite fazer todos os cálculos com inteiros e converter o resultado para um ponto flutuante somente quando necessário.

6.2.5 Usando Tipos de Campos de Outros Mecanismos de Banco de Dados

Para facilitar o uso de code para implementações SQL de outras empresas, MySQL mapeia os tipos de campos como mostrado na tabela seguinte. Este mapeamento torna fácil mudar definições de tabelas de outros mecanismos de banco de dados para o MySQL:

Tipo de outras empresas	Tipo MySQL
BINARY(NUM)	CHAR(NUM) BINARY
CHAR VARYING(NUM)	VARCHAR(NUM)

FLOAT4	FLOAT
FLOAT8	DOUBLE
INT1	TINYINT
INT2	SMALLINT
INT3	MEDIUMINT
INT4	INT
INT8	BIGINT
LONG VARBINARY	MEDIUMBLOB
LONG VARCHAR	MEDIUMTEXT
MIDDLEINT	MEDIUMINT
VARBINARY(NUM)	VARCHAR(NUM) BINARY

O mapeamento do tipo de campo ocorre na criação da tabela. Se você cria uma tabela com tipos usador por outras empresas e então executa uma instrução `DESCRIBE nome_tabela`, MySQL relaciona a estrutura de tabela utilizando os tipos equivalentes do MySQL.

6.2.6 Exigências de Armazenamento dos Tipos de Coluna

As exigências de armazenamento para cada um dos tipos de colunas suportados pelo MySQL estão listados por categoria.

Exigências de armazenamento para tipos numéricos

Tipo da coluna	Tamanho exigido
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(X)	4 se $X \leq 24$ ou 8 se $25 \leq X \leq 53$
FLOAT	4 bytes
DOUBLE	8 bytes
DOUBLE PRECISION	8 bytes
REAL	8 bytes
DECIMAL(M,D)	M+2 bytes se $D > 0$, M+1 bytes se $D = 0$ (D+2, se $M < D$)
NUMERIC(M,D)	M+2 bytes se $D > 0$, M+1 bytes se $D = 0$ (D+2, se $M < D$)

Exigência de armazenamento para tipos data e hora

Tipo de coluna	Tamanho exigido
DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

Exigência de armazenamento para tipos string

Tipo de coluna	Tamanho exigido
CHAR(M)	M bytes, $1 \leq M \leq 255$
VARCHAR(M)	L+1 bytes, onde $L \leq M$ e $1 \leq M \leq 255$
TINYBLOB, TINYTEXT	L+1 bytes, onde $L < 2^8$
BLOB, TEXT	L+2 bytes, onde $L < 2^{16}$
MEDIUMBLOB, MEDIUMTEXT	L+3 bytes, onde $L < 2^{24}$
LONGBLOB, LONGTEXT	L+4 bytes, onde $L < 2^{32}$
ENUM('valor1', 'valor2', ...)	1 ou 2 bytes, dependendo do número de valores enumerados (65535 valores no máximo)
SET('valor1', 'valor2', ...)	1, 2, 3, 4 or 8 bytes, dependendo do número de membros do conjunto (64 membros no máximo)

Tipos VARCHAR, BLOB e TEXT são de tamanho variáveis, tendo o tamanho exigido para armazenamento dependendo do tamanho atual dos valores da coluna (representado por L na tabela anterior), e não do tamanho máximo do tipo. Por exemplo, uma coluna VARCHAR(10) pode guardar uma string com um tamanho máximo de 10 caracteres. O tamanho exigido para armazenamento atual é o tamanho da string (L), mais 1 byte para gravar o tamanho da string. Por exemplo, para a string 'abcd', L é 4 e o tamanho exigido para armazenamento é 5 bytes.

Os tipos BLOB e TEXT exigem 1, 2, 3 ou 4 bytes para gravar o tamanho do valor da coluna, dependendo do tamanho máximo possível do tipo. See [\[BLOB\]](#), page [\[undefined\]](#).

Se uma tabela inclui qualquer tipo de coluna de tamanho variável, o formato do registro também será de tamanho variável. Note que quando uma tabela é criada, MySQL pode, sob certas condições, mudar uma coluna de um tipo de tamanho variável para um tipo de tamanho fixo, ou vice-versa. See [\[Silent column changes\]](#), page [\[undefined\]](#).

O tamanho de um objeto ENUM é determinado por um número de diferentes valores enumerados. Um byte é usado para enumerações até 255 valores possíveis. Dois bytes são usados para enumerações até 65535 valores. See [\[ENUM\]](#), page [\[undefined\]](#).

O tamanho de uma objeto é determinado pelo número de diferentes membros do conjunto. Se o tamanho do conjunto é N, o objeto ocupa $(N+7)/8$ bytes, arredondados acima para 1, 2, 3, 4, ou 8 bytes. Um SET pode ter no máximo 64 membros. See [\[SET\]](#), page [\[undefined\]](#).

O tamanho máximo de um registro em uma tabela MyISAM é 65534 bytes. Cada coluna BLOB e TEXT ocupa apenas 5-9 bytes deste tamanho.

6.3 Funções para Uso em Cláusulas SELECT e WHERE

Um `select_expression` ou `where_definition` em uma instrução SQL pode consistir de qualquer expressão utilizando as funções descritas abaixo.

Uma expressão que contém NULL sempre produz um valor NULL a menos que esteja indicado na documentação para os operandos e funções envolvidos na expressão.

Nota: Não deve haver nenhum espaço em branco entre um nome de função e os parêntesis que a seguem. Isto ajuda o analisador MySQL a distinguir entre chamadas de funções e

referências a tabelas ou colunas que possuem o mesmo nome de uma função. Espaços entre argumentos são permitidos.

Você pode forçar o MySQL a aceitar espaços depois do nome de funções iniciando o `mysql` com a opção `--ansi` ou usando o `CLIENT_IGNORE_SPACE` no `mysql_connect()`, mas neste caso nome de funções se tornarão palavras reservadas. See [\[ANSI mode\]](#), [page](#).

Para sermos breves, exemplos mostram a saída do programa `mysql` na forma abreviada. Então isto:

```
mysql> SELECT MOD(29,9);
1 rows in set (0.00 sec)
```

```
+-----+
| mod(29,9) |
+-----+
|          2 |
+-----+
```

é mostrado desta forma:

```
mysql> SELECT MOD(29,9);
-> 2
```

6.3.1 Operadores e Funções de Tipos não Especificados

6.3.1.1 Parenteses

(...)

Use parenteses para forçar a ordem em que as expressões serão avaliadas. Por exemplo:

```
mysql> SELECT 1+2*3;
-> 7
mysql> SELECT (1+2)*3;
-> 9
```

6.3.1.2 Operadores de Comparação

Operações de comparação resultam em um valor 1 (VERDADEIRO), 0 (FALSO), ou NULL. Estas funções funcionam tanto para tipos numéricos quanto para tipos strings. Strings são convertidas automaticamente para números e números para strings quando necessário (como em Perl).

MySQL realiza comparações de acordo com as seguintes regras:

- Se um ou ambos os argumentos são NULL, o resultado da comparação é NULL, exceto para o operador `<=>`.
- Se ambos os argumentos em uma comparação são strings, eles são comparados como strings.
- Se ambos os argumentos são inteiros, eles são comparados como inteiros.
- Valores hexadecimais são tratados como strings binárias se não comparadas a um número.

- Se uma dos argumentos é uma coluna `TIMESTAMP` ou `DATETIME` e o outro argumento é uma constante, a constante é convertida para um timestamp antes da comparação ser realizada. Isto ocorre para ser mais amigável ao ODBC.
- Em todos os outros casos, os argumentos são comparados como números de ponto flutuante (real).

Por padrão, comparações de string são feita de modo independente do caso, usando o conjunto de caracteres atual (ISO-8859-1 Latin1 por padrão, o qual também funciona de forma excelente para o Inglês).

Os seguintes exemplos ilustram a conversão de strings para números para operações de comparação:

```
mysql> SELECT 1 > '6x';
-> 0
mysql> SELECT 7 > '6x';
-> 1
mysql> SELECT 0 > 'x6';
-> 0
mysql> SELECT 0 = 'x6';
-> 1
```

= Igual:

```
mysql> SELECT 1 = 0;
-> 0
mysql> SELECT '0' = 0;
-> 1
mysql> SELECT '0.0' = 0;
-> 1
mysql> SELECT '0.01' = 0;
-> 0
mysql> SELECT '.01' = 0.01;
-> 1
```

<>

!= Diferente:

```
mysql> SELECT '.01' <> '0.01';
-> 1
mysql> SELECT .01 <> '0.01';
-> 0
mysql> SELECT 'zapp' <> 'zappp';
-> 1
```

<= Menor que ou igual:

```
mysql> SELECT 0.1 <= 2;
-> 1
```

< Menor que:

```
mysql> SELECT 2 < 2;
-> 0
```

>= Maior que ou igual:

```
mysql> SELECT 2 >= 2;
-> 1
```

> Maior que:

```
mysql> SELECT 2 > 2;
-> 0
```

<=> Igual para NULL:

```
mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
-> 1 1 0
```

IS NULL

IS NOT NULL

Teste para saber se um valor é ou não NULL:

```
mysql> SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;
-> 0 0 1
```

```
mysql> SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
-> 1 1 0
```

Para estar apto a funcionar bem com outros programas, MySQL suporta os seguintes recursos extras quando utiliza-se IS NULL:

- Você pode encontrar o último registro inserido com:

```
SELECT * FROM nome_tabela WHERE auto_col IS NULL
```

Isto pode ser desabilitado configurando SQL_AUTO_IS_NULL=0. See [\(undefined\)](#) [SET OPTION], page [\(undefined\)](#).

- Para colunas DATE e DATETIME NOT NULL você pode encontrar a data especial 0000-00-00 utilizando:

```
SELECT * FROM nome_tabela WHERE coluna_data IS NULL
```

Isto é necessário para que alguns aplicações ODBC funcionem (já que ODBC não tem suporte a data 0000-00-00)

expr BETWEEN min AND max

Se **expr** é maior que ou igual a **min** e **expr** é menor que ou igual a **max**, **BETWEEN** retorna 1, senão é retornado 0. Isto é equivalente a expressão (**min** <= **expr** AND **expr** <= **max**) se todos os argumentos são do mesmo tipo. Senão os tipos são convertidos, conforme as regras acima, e aplicadas a todos os três argumentos. **Note** que antes da versão 4.0.5 argumentos eram convertidos para o tipo da **expr**.

```
mysql> SELECT 1 BETWEEN 2 AND 3;
-> 0
mysql> SELECT 'b' BETWEEN 'a' AND 'c';
-> 1
mysql> SELECT 2 BETWEEN 2 AND '3';
-> 1
mysql> SELECT 2 BETWEEN 2 AND 'x-3';
-> 0
```

expr NOT BETWEEN min AND max

O mesmo que NOT (expr BETWEEN min AND max).

expr IN (valor, ...)

Retorna 1 se **expr** é qualquer dos valores na lista **IN**, senão retorna 0. Se todos os valores são constantes, então os valores são avaliados de acordo com o tipo da **expr** e ordenado. A busca do item é então feita usando pesquisa binária. Isto significa que **IN** é muito rápido se os valores da lista **IN** forem todos contantes. Se **expr** é uma expressão strig em caso-sensitivo, a comparação é realizadas no modo caso-sensitivo:

```
mysql> SELECT 2 IN (0,3,5,'wefwf');
-> 0
mysql> SELECT 'wefwf' IN (0,3,5,'wefwf');
-> 1
```

O número de valores na lista **IN** é limitada apenas pelo valor `max_allowed_packet`.

Na versão 4.1 (para se adequar ao padrão SQL-99), **IN** retorna **NULL** não apeans se a expressão a sua esquerda é **NULL**, mas também se nenhuma correspondência é encontrada na lista e uma de suas expressões é **NULL**.

expr NOT IN (valor, ...)

O mesmo que **NOT (expr IN (valor, ...))**.

ISNULL(expr)

Se **expr** é **NULL**, **ISNULL()** retorna 1, senão retorna 0:

```
mysql> SELECT ISNULL(1+1);
-> 0
mysql> SELECT ISNULL(1/0);
-> 1
```

Note que a compração de valores **NULL** usando **=** sempre será falso!

COALESCE(lista)

Retorna o primeiro elemento não **NULL** na lista:

```
mysql> SELECT COALESCE(NULL,1);
-> 1
mysql> SELECT COALESCE(NULL,NULL,NULL);
-> NULL
```

INTERVAL(N,N1,N2,N3,...)

Retorna 0 se $N < N_1$, 1 se $N < N_2$ e assim por diante. Todos os argumentos são tratados como inteiros. Isto exige que $N_1 < N_2 < N_3 < \dots < N_n$ para que esta função funcione corretamente. Isto ocorre devido a utilização pesquisa binária (muito rápida):

```
mysql> SELECT INTERVAL(23, 1, 15, 17, 30, 44, 200);
-> 3
mysql> SELECT INTERVAL(10, 1, 10, 100, 1000);
-> 2
mysql> SELECT INTERVAL(22, 23, 30, 44, 200);
-> 0
```

Se você está comparando strings em caso-insensitivo com qualquer um dos operadores padrões (**=**, **<>**..., mas não **LIKE**) espaços em branco extras (espaços, tabulações e novas linhas) serão ignorados.

```
mysql> SELECT "a" ="A \n";
-> 1
```

6.3.1.3 Operadores Lógicos

Em SQL, todos os operadores lógicos avaliam TRUE (VERDADEIRO), FALSE (FALSO) ou NULL (DESCONHECIDO). No MySQL, esta implementação é como 1 (TRUE), 0 (FALSE), e NULL. A maioria deles é comum entre diferentes bancos de dados SQL. no entanto alguns podem retonar qualquer valor diferente de zero para TRUE.

NOT

! NOT lógico. Avalia como 1 se o operador é 0, como 0 se o operador é diferente de zero, e NOT NULL retorna NULL.

```
mysql> SELECT NOT 10;
-> 0
mysql> SELECT NOT 0;
-> 1
mysql> SELECT NOT NULL;
-> NULL
mysql> SELECT ! (1+1);
-> 0
mysql> SELECT ! 1+1;
-> 1
```

O último exemplo produz 1 pois a expressão é avaliada como (!1)+1.

AND

&& AND lógico. Avalia como 1 se todos os operandos são diferentes de zero e não é NULL, como 0 se um ou mais operandos são 0, senão retorna NULL.

```
mysql> SELECT 1 && 1;
-> 1
mysql> SELECT 1 && 0;
-> 0
mysql> SELECT 1 && NULL;
-> NULL
mysql> SELECT 0 && NULL;
-> 0
mysql> SELECT NULL && 0;
-> 0
```

Por favor note que as versões do MySQL anteriores a versão 4.0.5 param a avaliação quando um valor NULL é encontrado, e não continua o processo buscando por possíveis 0s. Isto significa que nessa versão, `SELECT (NULL AND 0)` retorna NULL ao invés de 0. Na versão 4.0.5 o código tem sido re-elaborado para que o resultado sempre seja como prescrito pelo padrão SQL utilizando a otimização sempre que possível.

OR

|| OR lógico. Avalia como 1 se algum operando é diferente de zero e como NULL se algum operando for NULL, senão 0 é retornado.

```
mysql> SELECT 1 || 1;
-> 1
mysql> SELECT 1 || 0;
-> 1
mysql> SELECT 0 || 0;
-> 0
mysql> SELECT 0 || NULL;
-> NULL
mysql> SELECT 1 || NULL;
-> 1
```

XOR XOR lógico. Retorna NULL se o operando também é NULL. Para operandos não NULL, avalia como 1 se um número ímpar de operandos é diferente de zero, senão 0 é retornado.

```
mysql> SELECT 1 XOR 1;
-> 0
mysql> SELECT 1 XOR 0;
-> 1
mysql> SELECT 1 XOR NULL;
-> NULL
mysql> SELECT 1 XOR 1 XOR 1;
-> 1
```

a XOR b é matematicamente igual a (a AND (NOT b)) OR ((NOT a) and b). XOR foi adicionado na versão 4.0.2.

6.3.1.4 Funções de Fluxo de Controle

IFNULL(expr1,expr2)

Se *expr1* não é NULL, IFNULL() retorna *expr1*, senão retorna *expr2*. IFNULL() retorna um valor numérico ou string, dependendo do contexto no qual é usado:

```
mysql> SELECT IFNULL(1,0);
-> 1
mysql> SELECT IFNULL(NULL,10);
-> 10
mysql> SELECT IFNULL(1/0,10);
-> 10
mysql> SELECT IFNULL(1/0,'yes');
-> 'yes'
```

Na versão 4.0.6 e acima o valor resultante padrão de IFNULL(*expr1*,*expr2*) é o mais geral das duas expressões, na seguinte ordem: STRING, REAL ou INTEGER. A diferença das versões anteriores é mais notável quando se cria uma tabela baseada em uma expressão ou o MySQL tem que armazenar internamente um valor de IFNULL() em uma tabela temporária.

```
CREATE TABLE foo SELECT IFNULL(1,"teste") as teste;
```

Na versão 4.0.6 do MySQL o tipo da coluna 'teste' é CHAR(4) enquanto nas versões anteriores ela seria do tipo BIGINT.

NULLIF(expr1,expr2)

Se `expr1 = expr2` for verdadeiro, é retornado `NULL` senão é retornado `expr1`. Isto é o mesmo que `CASE WHEN x = y THEN NULL ELSE x END`:

```
mysql> SELECT NULLIF(1,1);
      -> NULL
mysql> SELECT NULLIF(1,2);
      -> 1
```

Note que `expr1` é avaliada duas vezes no MySQL se os argumentos não são iguais.

IF(expr1,expr2,expr3)

Se `expr1` é VERDADEIRA (`expr1 <> 0` e `expr1 <> NULL`) então `IF()` retorna `expr2`, senão ela retorna `expr3`. `IF()` retorna um valor numérico ou string, dependendo do contexto no qual é usado.

```
mysql> SELECT IF(1>2,2,3);
      -> 3
mysql> SELECT IF(1<2,'sim','não');
      -> 'sim'
mysql> SELECT IF(STRCMP('teste','teste1'),'não','sim');
      -> 'não'
```

Se `expr2` ou `expr3` é explicitamente `NULL` então o tipo resultante da função `IF()` é o tipo da coluna não `NULL`. (Este comportamento é novo na versão 4.0.3 do MySQL).

`expr1` é avaliada como um valor inteiro, o qual significa que se você está testando valores de ponto flutuante ou strings, você de fazê-lo usando um operando de comparação:

```
mysql> SELECT IF(0.1,1,0);
      -> 0
mysql> SELECT IF(0.1<>0,1,0);
      -> 1
```

No primeiro caso acima, `IF(0.1)` retorna 0 porque 0.1 é convertido para um valor inteiro, resultando em um teste `IF(0)`. Isto pode não ser o que você esperava. No segundo caso, a comparação testa se o valor de ponto flutuante não é zero. O resultado da comparação converte o termo em um inteiro.

O tipo de retorno padrão de `IF()` (o que pode importar quando ele é armazenado em uma tabela temporária) é calculado na versão 3.23 do MySQL de seguinte forma:

Expressão	Valor de retorno
<code>expr2</code> ou <code>expr3</code> retorna string	string
<code>expr2</code> ou <code>expr3</code> retorna um valor de ponto flutuante	ponto flutuante
<code>expr2</code> ou <code>expr3</code> retorna um inteiro	inteiro

Se `expr2` e `expr3` são strings, então o resultado é caso-insensitivo se ambas strings são caso insensitivo. (A partir da versão 3.23.51)

```
CASE valor WHEN [valor comparado] THEN resultado [WHEN [valor comparado] THEN
resultado ...] [ELSE resultado] END
```

```
CASE WHEN [condição] THEN result [WHEN [condição] THEN resultado ...] [ELSE
resultado] END
```

A primeira expressão retorna o resultado onde valor=valor comparado. A segunda expressão retorna o o resultado da primeira condição, a qual é verdadeira. Se não existe nenhum resultado correspondente, então o resultado depois do ELSE é retornado. Se não existe parte ELSE então é retornado NULL is returned:

```
mysql> SELECT CASE 1 WHEN 1 THEN "um"
           WHEN 2 THEN "dois" ELSE "mais" END;
-> "one"
mysql> SELECT CASE WHEN 1>0 THEN "verdadeiro" ELSE "falso" END;
-> "true"
mysql> SELECT CASE BINARY "B" WHEN "a" THEN 1 WHEN "b" THEN 2 END;
-> NULL
```

O tipo do valor de retorno (INTEGER, DOUBLE ou STRING) é do mesmo tipo do primeiro valor retornado (a expressão depois do primeiro THEN).

6.3.2 Funções String

Funções string retornam NULL se o tamanho do resultado for maior que o parâmetro do servidor `max_allowed_packet`. See [\(undefined\) \[Server parameters\]](#), page [\(undefined\)](#).

Para funções que operam com as posições de uma string, a primeira posição é numerada como 1.

ASCII(str)

Retorna o valor do código ASCII do caracter mais a esquerda da string `str`. Retorna 0 se `str` é uma string vazia. Retorna NULL se `str` é NULL:

```
mysql> SELECT ASCII('2');
-> 50
mysql> SELECT ASCII(2);
-> 50
mysql> SELECT ASCII('dx');
-> 100
```

Veja também a função `ORD()`.

ORD(str) Se o caracter mais a esquerda da string `str` é um caracter multi-byte, é retornado o código para este caracter, calculado a partir dos valores do código ASCII dos seus caracteres contituantes utilizando-se a seguinte fórmula: ((primeiro byte do código ASCII)*256+(segundo byte do código ASCII)) [*256+terceiro byte do código ASCII...]. Se o caracter mais a esquerda não é multi-byte, é retornado o mesmo valor que a função `ASCII()` retorna:

```
mysql> SELECT ORD('2');
-> 50
```

CONV(N,da_base,para_base)

Converte números entre diferentes bases. Retorna uma representação string do número N, convertido da base `da_base` para base `para_base`. Retorna NULL se qualquer argumento é NULL. O argumento N é interpretado como um inteiro, mas pode ser especificado como um inteiro ou uma string. A base mínima é 2 e a máxima é 36. Se `para_base` é um número negativo, N é considerado como um número com sinal. Caso contrário, N é tratado como um número sem sinal. CONV funciona com precisão de 64-bit:

```
mysql> SELECT CONV("a",16,2);
      -> '1010'
mysql> SELECT CONV("6E",18,8);
      -> '172'
mysql> SELECT CONV(-17,10,-18);
      -> '-H'
mysql> SELECT CONV(10+"10"+"10'+0xa,10,10);
      -> '40'
```

BIN(N) Retorna uma representação string do valor binário de N, onde N é um número muito grande (BIGINT). Isto é equivalente a `CONV(N,10,2)`. Retorna NULL se N é NULL:

```
mysql> SELECT BIN(12);
      -> '1100'
```

OCT(N)

Retorna uma representação string do valor octal de N, onde N é um número muito grande. Isto é equivalente a `CONV(N,10,8)`. Retorna NULL se N é NULL:

```
mysql> SELECT OCT(12);
      -> '14'
```

HEX(N_ou_S)

Se `N_OU_S` é um número, é retornado uma representação string do valor hexadecimal de N, onde N é um número muito grande (BIGINT). Isto é equivalente a `CONV(N,10,16)`.

Se `N_OU_S` é uma string, é retornado uma string hexadecimal de `N_OU_S` onde cada caracter de `N_OU_S` é convertido para 2 dígitos hexadecimais. Isto é o inverso da string `0xff`.

```
mysql> SELECT HEX(255);
      -> 'FF'
mysql> SELECT HEX("abc");
      -> 616263
mysql> SELECT 0x616263;
      -> "abc"
```

CHAR(N,...)

`CHAR()` interpreta os argumentos como inteiros e retorna uma string com caracteres dados pelo valor do código ASCII referentes a estes inteiros. Valores NULL são desconsiderados:

```
mysql> SELECT CHAR(77,121,83,81,'76');
      -> 'MySQL'
```

```
mysql> SELECT CHAR(77,77.3,'77.3');
-> 'MMM'
```

CONCAT(str1,str2,...)

Retorna a string resultante da concatenação dos argumentos. Retorna NULL se qualquer dos argumentos for NULL. Pode ter mais de 2 argumentos. Um argumento numérico é convertido para sua forma string equivalente:

```
mysql> SELECT CONCAT('My', 'S', 'QL');
-> 'MySQL'
mysql> SELECT CONCAT('My', NULL, 'QL');
-> NULL
mysql> SELECT CONCAT(14.3);
-> '14.3'
```

CONCAT_WS(separador, str1, str2,...)

CONCAT_WS() significa CONCAT With Separator (CONCAT com separador) e é uma forma especial do CONCAT(). O primeiro argumento é o separador para os outros argumentos. O separador pode ser uma string assim como os outros argumentos. Se o separador é NULL, o resultado será NULL. A função irá desconsiderar qualquer NULL e strings vazias, depois do argumento do separador. O separador será adicionado entre as strings concatenadas:

```
mysql> SELECT CONCAT_WS(",","First name","Second name","Last Name");
-> 'First name,Second name,Last Name'
mysql> SELECT CONCAT_WS(",","First name",NULL,"Last Name");
-> 'First name,Last Name'
```

LENGTH(str)**OCTET_LENGTH(str)****CHAR_LENGTH(str)****CHARACTER_LENGTH(str)**

Retorna o tamanho da string str:

```
mysql> SELECT LENGTH('text');
-> 4
mysql> SELECT OCTET_LENGTH('text');
-> 4
```

Note que para CHAR_LENGTH() e CHARACTER_LENGTH(), caracteres multi-byte são contados apenas uma vez.

BIT_LENGTH(str)

Retorna o tamanho da string str em bits:

```
mysql> SELECT BIT_LENGTH('text');
-> 32
```

LOCATE(substr, str)**POSITION(substr IN str)**

Retorna a posição da primeira ocorrência da substring substr na string str. Retorna 0 se substr não está na str:

```
mysql> SELECT LOCATE('bar', 'foobarbar');
-> 4
```

```
mysql> SELECT LOCATE('xbar', 'foobar');
-> 0
```

Esta função é multi-byte. Na versão 3.23 do MySQL esta função é caso sensitivo, enquanto na versão 4.0 ela só é caso-sensitivo se os argumentos são uma string binária.

LOCATE(substr, str, pos)

Retorna a posição da primeira ocorrência da substring **substr** na string **str**, iniciando na posição **pos**. Retorna 0 se **substr** não está em **str**:

```
mysql> SELECT LOCATE('bar', 'foobarbar',5);
-> 7
```

Esta função é multi-byte. Na versão 3.23 do MySQL esta função é caso sensitivo, enquanto na versão 4.0 ela só é caso-sensitivo se os argumentos são uma string binária.

INSTR(str, substr)

Retorna a posição da primeira ocorrência da substring **substr** na string **str**. É o mesmo que as o LOCATE() com dois argumentos, exceto pelo fato de que os argumentos estão tracados:

```
mysql> SELECT INSTR('foobarbar', 'bar');
-> 4
mysql> SELECT INSTR('xbar', 'foobar');
-> 0
```

Esta função é multi-byte. Na versão 3.23 do MySQL esta função é caso sensitivo, enquanto na versão 4.0 ela só é caso-sensitivo se os argumentos são uma string binária.

LPAD(str, tam, strpreench)

Retorna a string **str**, preenchida a esquerda com a string **strpreench** até que **str** possua **tam** caracteres. Se **str** é maior que **tam** então ela será reduzida para **tam** caracteres.

```
mysql> SELECT LPAD('hi',4,'??');
-> '??hi'
```

RPAD(str, tam, strpreech)

Retorna a string **str**, preenchida a direita com a string **strpreench** até que **str** possua **tam** caracteres. Se **str** é maior que **tam** então ela será reduzida para **tam** caracteres.

```
mysql> SELECT RPAD('hi',5,'?');
-> 'hi???'
```

LEFT(str, tam)

Retorna os **tam** caracteres mais a esquerda da string **str**:

```
mysql> SELECT LEFT('foobarbar', 5);
-> 'fooba'
```

Esta função é multi-byte.

RIGHT(str, tem)

Retorna os **tam** caracteres mais a esquerda da string **str**:

```
mysql> SELECT RIGHT('foobarbar', 4);
-> 'rbar'
```

Esta função é multi-byte.

SUBSTRING(str,pos,tam)

SUBSTRING(str FROM pos FOR tam)

MID(str,pos,tam)

Retorna a substring com **tam** caracteres da string **str**, iniciando da posição **pos**. A forma variante que utiliza **FROM** é a sintaxe SQL-92:

```
mysql> SELECT SUBSTRING('Quadratically',5,6);
-> 'ratica'
```

Esta função é multi-byte.

SUBSTRING(str,pos)

SUBSTRING(str FROM pos)

Retorna uma substring da string **str** iniciando na posição **pos**:

```
mysql> SELECT SUBSTRING('Quadratically',5);
-> 'ratically'
mysql> SELECT SUBSTRING('foobarbar' FROM 4);
-> 'barbar'
```

Esta função é multi-byte.

SUBSTRING_INDEX(str,delim,cont)

Retorna a substring da string **str** antes de **cont** ocorrências do delimitador **delim**. Se **cont** é positivo, tudo a esquerda do delimitador final (contando a partir da esquerda) é retornado. Se **cont** é negativo, tudo a direita do delimitador final (contando a partir da direita) é retornado.

```
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
-> 'www.mysql'
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
-> 'mysql.com'
```

Esta função é multi-byte.

LTRIM(str)

Retorna a string **str** com caracteres de espaços extras iniciais removidos:

```
mysql> SELECT LTRIM('  barbar');
-> 'barbar'
```

RTRIM(str)

Retorna a string **str** com caracteres de espaços extras finais removidos:

```
mysql> SELECT RTRIM('barbar  ');
-> 'barbar'
```

Esta função é multi-byte.

TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)

Retorna a string **str** com todos prefixos e/ou sufixos **remstr** removidos. Se nenhum dos especificadores **BOTH**, **LEADING** ou **TRAILING** são dados, é considerado **BOTH**. Se **remstr** não é especificada, espaços são removidos:

```
mysql> SELECT TRIM(' bar ');
-> 'bar'
mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
-> 'barxxx'
mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
-> 'bar'
mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxyz');
-> 'barx'
```

Esta função é multi-byte.

SOUNDEX(str)

Retorna uma string 'soundex' de **str**. Duas strings que soam de forma parecida devem ter strings 'soundex' iguais. Uma string soundex padrão possui 4 caracteres, mas a função SOUNDEX() retorna uma string de tamanho arbitrário. Você pode usar SUBSTRING() no resultado para obter uma string 'soundex' padrão. Todos os caracteres não alfanuméricos são ignorados na string dada. Todas as caracteres internacionais fora da faixa A-Z são tratados como vogais:

```
mysql> SELECT SOUNDEX('Hello');
-> 'H400'
mysql> SELECT SOUNDEX('Quadratically');
-> 'Q36324'
```

SPACE(N) Retorna uma string contendo N caracteres de espaço:

```
mysql> SELECT SPACE(6);
-> '      '
```

REPLACE(str,da_str,para_str)

Retorna a string **str** com todas as ocorrências da string **da_str** substituída pela string **para_str**:

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
-> 'WwWwWw.mysql.com'
```

Esta função é multi-byte.

REPEAT(str,cont)

Retorna uma string consistindo da string **str** repetida **cont** vezes. Se **cont** <= 0, é retornado uma string vazia. É retornado NULL se **str** ou **cont** são NULL:

```
mysql> SELECT REPEAT('MySQL', 3);
-> 'MySQLMySQLMySQL'
```

REVERSE(str)

Returns the string **str** with the order of the characters reversed:

```
mysql> SELECT REVERSE('abc');
-> 'cba'
```

Esta função é multi-byte.

INSERT(str,pos,tam,novastr)

Retorna a string **str**, com a substring começando na posição **pos** e contendo **tam** caracteres substituída pela string **novastr**:

```
mysql> SELECT INSERT('Quadraticio', 3, 4, 'Onde');
```

```
-> 'Qu0ndetico'
```

Esta função é multi-byte.

ELT(N, str1, str2, str3, ...)

Retorna *str1* se *N* = 1, *str2* se *N* = 2, e assim por diante. Retorna NULL se *N* é menor que 1 ou maior que o número de argumentos. **ELT()** é o complemento de **FIELD()**:

```
mysql> SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');
-> 'ej'
mysql> SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo');
-> 'foo'
```

FIELD(str, str1, str2, str3, ...)

Retorna o índice de *str* na lista *str1, str2, str3, ...*. Retorna 0 se *str* não for encontrada. **FIELD()** é o complemento de **ELT()**:

```
mysql> SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
-> 2
mysql> SELECT FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');
-> 0
```

FIND_IN_SET(str, strlista)

Retorna um valor 1 para *N* se a string *str* está na lista *strlist* contendo *N* substrings. A lista de string é composta de substrings separadas pelo caracter ','. Se o primeiro argumento é uma string constante e o segundo é uma coluna do tipo SET, a função **FIND_IN_SET()** é otimizada para usar aritmética binária! Retorna 0 se *str* não está na *strlista* ou se *strlista* é uma string vazia. Retorna NULL se os argumentos são NULL. Esta função não irá funcionar adequadamente se o primeiro argumento contém uma ',':

```
mysql> SELECT FIND_IN_SET('b', 'a,b,c,d');
-> 2
```

MAKE_SET(bits, str1, str2, ...)

Retorna um conjunto (uma string contendo substrings separadas por ',') contendo as strings que tem o bit correspondente em *bits* definido. *str1* corresponde ao bit 1, *str2* ao bit 2, etc. Strings NULL em *str1, str2, ...* não são adicionadas ao resultado:

```
mysql> SELECT MAKE_SET(1, 'a', 'b', 'c');
-> 'a'
mysql> SELECT MAKE_SET(1 | 4, 'Oi', 'meu', 'mundo');
-> 'Oi,mundo'
mysql> SELECT MAKE_SET(0, 'a', 'b', 'c');
-> ''
```

EXPORT_SET(bits, on, off, [separador, [numero_de_bits]])

Retorna uma string onde para todo bit 1 em 'bit', você obtém uma string 'on' e para cada bit 0 você obtém uma string 'off'. Cada string é separada com 'separador' (padrão, ',') e só 'numero_de_bits' (padrão 64) de 'bits' é usado:

```
mysql> SELECT EXPORT_SET(5, 'S', 'N', ',', ', ', 4)
-> S,N,S,N
```

LCASE(str)

LOWER(str)

Retorna a string `str` com todos caracteres alterados para letra minúsculas de acordo com o conjunto de caracteres atual (o padrão é ISO-8859-1 Latin1):

```
mysql> SELECT LCASE('MYSQL');
      -> 'mysql'
```

Esta é uma função multi-byte.

UCASE(str)

UPPER(str)

Retorna a string `str` com todos caracteres alterados para letra maiúsculas de acordo com o conjunto de caracteres atual (o padrão é ISO-8859-1 Latin1):

```
mysql> SELECT UCASE('Hej');
      -> 'HEJ'
```

Esta é uma função multi-byte.

LOAD_FILE(nome_arquivo)

Lê o arquivo e retorna o conteúdo do arquivo como uma string. O arquivo deve estar no servidor, você deve especificar o caminho completo para o arquivo, e você deve ter o privilégio FILE. O arquivo deve ser legível para todos e ser menor que o especificado em `max_allowed_packet`.

Se o arquivo não existe ou não pode ser lido devido a alguma das razões acima, a função retornará NULL:

```
mysql> UPDATE nome_tabela
      SET coluna_blob=LOAD_FILE("/tmp/picture")
      WHERE id=1;
```

Se você não está usando a versão 3.23 MySQL, você tem que fazer a leitura do arquivo dentro do seu aplicativo e criar uma instrução INSERT para atualizar o banco de dados com a informação do arquivo. Um modo de se fazer isto, se você estiver usando a biblioteca MySQL++, pode ser encontrada em <http://www.mysql.com/documentation/mysql++/mysql++-examples.html>.

QUOTE(str)

Coloca uma string entre aspas para produzir um resultado que possa ser usada em uma instrução SQL como um valor de dados com o caracter de escape correto. A string é retornada entre aspas simples e cada instância de aspas simples (''), barra invertida ('\'), ASCII NUL, e Control-Z é precedida por uma barra invertida. Se o argumento é NULL, o valor retornado é a palavra "NULL" sem aspas simples.

A função QUOTE foi adicionada na versão 4.0.3 do MySQL.

```
mysql> SELECT QUOTE("Don't");
      -> 'Don\'t!'
mysql> SELECT QUOTE(NULL);
      -> NULL
```

6.3.2.1 Funções de Comparação de Strings

MySQL automaticamente converte números para quando necessário, e vice-versa:

```
mysql> SELECT 1+"1";
-> 2
mysql> SELECT CONCAT(2,' test');
-> '2 test'
```

Se você quiser converter um número em uma string de forma explícita, passe-o como um argumento de `CONCAT()`.

Se uma função de string tem uma string binária como argumento, a string resultante é também um string binária. Um número convertido para uma string é tratado como um string binária. Isto afeta apenas a comparação.

Normalmente, se qualquer expressão em uma string é caso-sensitivo, a comparação é realizada no modo caso sensitivo.

expr LIKE pad [ESCAPE 'car-escape']

Correspondência de padrões usando uma simples expressão de comparações SQL. Retorna 1 (VERDADEIRO) ou 0 (FALSO). Com `LIKE` você pode usar os seguintes meta-caracteres no padrão:

Car	Descrição
%	Corresponde a qualquer número de caracteres, até zero caracteres
_	Corresponde a exatamente um caracter

```
mysql> SELECT 'David!' LIKE 'David_';
-> 1
mysql> SELECT 'David!' LIKE '%D%v%';
-> 1
```

Para testar instâncias literais de um meta caracter, preceda o caracter com o carcter de escape. Se você não especificar o caracter de `ESCAPE`, assume-se `'\'`:

String	Description
\%	Correponde a um caracter %
_	Correponde a um caracter _

```
mysql> SELECT 'David!' LIKE 'David\_';
-> 0
mysql> SELECT 'David_' LIKE 'David\_';
-> 1
```

Para especificar um caracter de escape diferente, use a cláusula `ESCAPE`:

```
mysql> SELECT 'David_' LIKE 'David|_' ESCAPE '|';
-> 1
```

As seguintes instruções mostram que a comparação de strings são caso-insensitivo, a menos que um dos operandos seja uma string binária:

```
mysql> SELECT 'abc' LIKE 'ABC';
-> 1
mysql> SELECT 'abc' LIKE BINARY 'ABC';
-> 0
```

`LIKE` é permitido em uma expressão numérica! (Esta é uma extensão MySQL para o `LIKE` do SQL-99.)

```
mysql> SELECT 10 LIKE '1%';
```

-> 1

Nota: Como MySQL usa sintaxe de escape do C em strings (por exemplo, '\n'), você deve dobrar qualquer '\' que você usar em sua string LIKE. Por exemplo, para pesquisar por '\n', especifique-o como '\\n'. Para buscar por '\', especifique-o como '\\\\' (as barras invertidas são eliminadas uma vez pelo analisador e outra vez quando a correspondência de padrões é feita, deixando uma única barra invertida para ser verificada).

Note: O LIKE atual não é um caracter multi-byte. Comparações são feitas caracter por caracter.

expr NOT LIKE pad [ESCAPE 'car-escape']

O mesmo que NOT (expr LIKE pad [ESCAPE 'car-escape']).

expr SOUNDS LIKE expr

O mesmo que SOUNDEX(expr)=SOUNDEX(expr) (disponível apenas na versão 4.1 ou posterior).

expr REGEXP pad

expr RLIKE pad

Realiza a busca de padrões em uma expressão string com base no padrão pad. O padrão pode ser uma expressão regular estendida. See [\(undefined\)](#) [Regexp], page [\(undefined\)](#). Retorna 1 se expr coincide com pad, senão retorna 0. RLIKE é um sinônimo para REGEXP, fornecido para compatibilidade com mSQL. Nota: Como MySQL usa a sintaxe de escape do C em strings (por exemplo, '\n'), você deve dobrar qualquer '\' que você use em sua string REGEXP. Como na versão 3.23.4 do MySQL, REGEXP é caso-insensitivo para strings normais (não binárias).

```
mysql> SELECT 'Monty!' REGEXP 'm%y%';
-> 0
mysql> SELECT 'Monty!' REGEXP '.*';
-> 1
mysql> SELECT 'new*\n*line' REGEXP 'new\\*.*\\*line';
-> 1
mysql> SELECT "a" REGEXP "A", "a" REGEXP BINARY "A";
-> 1 0
mysql> SELECT "a" REGEXP "[a-d]";
-> 1
```

REGEXP e RLIKE usam o conjunto de caracteres atual (ISO-8859-1 Latin1 por padrão) para decidir o tipo de caracter.

expr NOT REGEXP pad

expr NOT RLIKE pad

O mesmo que NOT (expr REGEXP pad).

STRCMP(expr1,expr2)

STRCMP() retorna 0 se as string são a mesma, -1 se o primeiro argumento é menor que o segundo de acordo com a ordenação atual e 1 em caso contrário:

```
mysql> SELECT STRCMP('texto', 'texto2');
-> -1
```

```
mysql> SELECT STRCMP('texto2', 'texto');
-> 1
mysql> SELECT STRCMP('texto', 'texto');
-> 0
```

MATCH (col1,col2,...) AGAINST (expr)

MATCH (col1,col2,...) AGAINST (expr IN BOOLEAN MODE)

MATCH ... AGAINST() é usado para busca de textos completos e retorna a relevância - similaridade medida entre o texto nas colunas (col1,col2,...) e a consulta **expr**. Relevância é um número de ponto flutuante. Relevância zero significa que não houve nenhuma similaridade. **MATCH ... AGAINST()** está disponível na versão 3.23.23 ou posterior do MySQL. A extensão **IN BOOLEAN MODE** foi adicionada na versão 4.0.1. Para detalhes e exemplos de uso, veja [\[Fulltext Search\]](#), page [\[undefined\]](#).

6.3.2.2 Caso-Sensitivo

BINARY O operador **BINARY** transforma uma string em uma string binária. Este é um modo fácil de forçar a comparação para se caso-sensitivo mesmo se a coluna não seja definida como **BINARY** ou **BLOB**:

```
mysql> SELECT "a" = "A";
-> 1
mysql> SELECT BINARY "a" = "A";
-> 0
```

BINARY string é um atalho para **CAST(string AS BINARY)**. See [\[Cast Functions\]](#), page [\[undefined\]](#). **BINARY** foi introduzida na versão 3.23.0 do MySQL.

Note que em alguns contextos MySQL não estará apto a usar o índice de forma eficiente quando se transformar uma coluna índice em **BINARY**.

Se você quiser compara um blob caso-insensitivo você pode sempre convertê-lo para letras maiúsculas antes de faer a comparação:

```
SELECT 'A' LIKE UPPER(col_blob1) FROM nome_tabela;
```

Não planejamos introduzir em breve coerção (casting) entre diferentes conjuntos de caracteres para tornar comparações de strings mais flexível.

6.3.3 Funções Numéricas

6.3.3.1 Operações Aritiméticas

Os operadores aritiméticos usuais estão disponíveis. '-', '+', e '*', o resultado é calculado com precisão de **BIGINT** (64-bit) se ambos os argumentos são inteiros! Se um dos argumentos for um inteiro sem sinal, e o outro argumento é um inteiro também, o resultado será um inteiro sem sinal. See [\[Cast Functions\]](#), page [\[undefined\]](#).

+ Adição:

```
mysql> SELECT 3+5;
-> 8
```

- Subtração:

```
mysql> SELECT 3-5;
-> -2
```

* Multiplicação:

```
mysql> SELECT 3*5;
-> 15
mysql> SELECT 18014398509481984*18014398509481984.0;
-> 324518553658426726783156020576256.0
mysql> SELECT 18014398509481984*18014398509481984;
-> 0
```

O resultado da última expressão é incorreta porque o resultado da multiplicação de inteiros excede a faixa de 64-bits dos cálculos BIGINT.

/ Divisão:

```
mysql> SELECT 3/5;
-> 0.60
```

Divisões por zero produz um resultado NULL:

```
mysql> SELECT 102/(1-1);
-> NULL
```

Uma divisão será calculada com aritmética BIGINT somente se executada em um contexto no qual o resultado é convertido para um inteiro!

6.3.3.2 Funções Matemáticas

Todas as funções matemáticas retornam NULL no caso de um erro.

- Menos unario. Muda o sinal do argumento:

```
mysql> SELECT - 2;
-> -2
```

Note que se este operador é utilizado com um BIGINT, o valor retornado é um BIGINT! Isto significa que você deve evitar usar - em inteiros que pode ter o valor de -2^{63} !

ABS(X) Retorna o valor absoluto de X:

```
mysql> SELECT ABS(2);
-> 2
mysql> SELECT ABS(-32);
-> 32
```

O uso desta função é seguro com valores BIGINT.

SIGN(X) Retorna o sinal do argumento como -1, 0, ou 1, dependendo de quando X é negativo, zero, ou positivo:

```
mysql> SELECT SIGN(-32);
-> -1
mysql> SELECT SIGN(0);
-> 0
mysql> SELECT SIGN(234);
-> 1
```

MOD(N,M)

% Modulo (como o operador % em C). Retorna o resto de N dividido por M:

```
mysql> SELECT MOD(234, 10);
-> 4
mysql> SELECT 253 % 7;
-> 1
mysql> SELECT MOD(29,9);
-> 2
mysql> SELECT 29 MOD 9;
-> 2
```

O uso desta função é seguro com valores **BIGINT**. O último exemplo só funciona no MySQL 4.1

FLOOR(X) Retorna o maior valor inteiro não maior que X:

```
mysql> SELECT FLOOR(1.23);
-> 1
mysql> SELECT FLOOR(-1.23);
-> -2
```

Note que o valor retornado é convertido para um **BIGINT**!

CEILING(X)

Retorna o menor valor inteiro não menor que X:

```
mysql> SELECT CEILING(1.23);
-> 2
mysql> SELECT CEILING(-1.23);
-> -1
```

Note que o valor retornado é convertido para um **BIGINT**!

ROUND(X)**ROUND(X,D)**

Retorna o argumento X, arredondado para o inteiro mais próximo. Com dois argumentos o arredondamento é feito para um número com D decimais.

```
mysql> SELECT ROUND(-1.23);
-> -1
mysql> SELECT ROUND(-1.58);
-> -2
mysql> SELECT ROUND(1.58);
-> 2
mysql> SELECT ROUND(1.298, 1);
-> 1.3
mysql> SELECT ROUND(1.298, 0);
-> 1
```

Note que o comportamento de **ROUND()** quando o argumento está no meio do caminho entre dois inteiros depende da implementação da biblioteca C. Alguns arredondamentos para o número mais próximo, são sempre para baixo, para cima ou são zero. Se você precisa de um tipo de arredondamento, você deve usar uma função bem definida como **TRUNCATE()** ou **FLOOR()**.

DIV

Divisão de inteiros. Similar ao **FLOOR()** mas seguro com valores **BIGINT**.

```
mysql> SELECT 5 DIV 2
-> 2
```

DIV é novo no MySQL 4.1.0.

EXP(X) Retorna o valor de e (the base of natural logarithms) raised to the power of X:

```
mysql> SELECT EXP(2);
-> 7.389056
mysql> SELECT EXP(-2);
-> 0.135335
```

LN(X) Retorna o logaritmo natural de X:

```
mysql> SELECT LN(2);
-> 0.693147
mysql> SELECT LN(-2);
-> NULL
```

Esta função foi adicionada na versão 4.0.3 do MySQL. É sinônimo de LOG(X) no MySQL.

LOG(X)

LOG(B, X) Se chamado com um parâmetro, esta função retorna o logaritmo natural de X:

```
mysql> SELECT LOG(2);
-> 0.693147
mysql> SELECT LOG(-2);
-> NULL
```

Se chamado com dois parâmetros, esta função retorna o logaritmo natural de X para uma base arbitrária B:

```
mysql> SELECT LOG(2,65536);
-> 16.000000
mysql> SELECT LOG(1,100);
-> NULL
```

A opção de base arbitrária foi adicionada na versão 4.0.3 do MySQL. LOG(B, X) é equivalente a LOG(X)/LOG(B).

LOG2(X) Retorna o logaritmo na base 2 de X:

```
mysql> SELECT LOG2(65536);
-> 16.000000
mysql> SELECT LOG2(-100);
-> NULL
```

LOG2() é útil para descobrir quantos bits um número necessitaria para ser armazenado. Esta função foi adicionada na versão 4.0.3 do MySQL. Em versões anteriores, você pode usar LOG(X)/LOG(2).

LOG10(X) Retorna o logaritmo na base 10 de X:

```
mysql> SELECT LOG10(2);
-> 0.301030
mysql> SELECT LOG10(100);
-> 2.000000
mysql> SELECT LOG10(-100);
-> NULL
```

POW(X, Y)

POWER(X, Y)

Retorna o valor de X elevado a potência de Y:

```
mysql> SELECT POW(2,2);
```

```
-> 4.000000
```

```
mysql> SELECT POW(2,-2);
```

```
-> 0.250000
```

SQRT(X)

Retorna o a raiz quadrada não negativa de X:

```
mysql> SELECT SQRT(4);
```

```
-> 2.000000
```

```
mysql> SELECT SQRT(20);
```

```
-> 4.472136
```

PI()

Retorna o valor de PI. A quantidade de números decimais padrão é 5, mas o MySQL usa internamente a precisão dupla completa para PI.

```
mysql> SELECT PI();
```

```
-> 3.141593
```

```
mysql> SELECT PI()+0.00000000000000000000;
```

```
-> 3.141592653589793116
```

COS(X)

Retorna o cosseno de X, onde X é dado em radianos:

```
mysql> SELECT COS(PI());
```

```
-> -1.000000
```

SIN(X)

Retorna o seno de X, onde X é dado em radianos:

```
mysql> SELECT SIN(PI());
```

```
-> 0.000000
```

TAN(X)

Retorna a tangente de X, onde X é dado em radianos:

```
mysql> SELECT TAN(PI()+1);
```

```
-> 1.557408
```

ACOS(X)

Retorna o arco cosseno X, isto é, o valor cujo cosseno é X. Retorna NULL se X não está na faixa de -1 a 1:

```
mysql> SELECT ACOS(1);
```

```
-> 0.000000
```

```
mysql> SELECT ACOS(1.0001);
```

```
-> NULL
```

```
mysql> SELECT ACOS(0);
```

```
-> 1.570796
```

ASIN(X)

Retorna o arco seno X, isto é, o valor cujo seno é X. Retorna NULL se X não está na faixa de -1 a 1:

```
mysql> SELECT ASIN(0.2);
```

```
-> 0.201358
```

```
mysql> SELECT ASIN('foo');
```

```
-> 0.000000
```

ATAN(X)

Retorna o arco tangente X, isto é, o valor cuja tangente é X. X:

```
mysql> SELECT ATAN(2);
      -> 1.107149
mysql> SELECT ATAN(-2);
      -> -1.107149
```

ATAN(Y, X)

ATAN2(Y, X)

Retorna o arco tangente de duas variáveis X e Y. É similar ao cálculo do arco tangente de Y / X, exceto que os sinais de ambos argumentos são usados para determinar o quadrante do resultado:

```
mysql> SELECT ATAN(-2,2);
      -> -0.785398
mysql> SELECT ATAN2(PI(),0);
      -> 1.570796
```

COT(X)

Returns a cotangente de X:

```
mysql> SELECT COT(12);
      -> -1.57267341
mysql> SELECT COT(0);
      -> NULL
```

RAND()

RAND(N)

Retorna um valor de ponto flutuante aleatório na faixa de 0 a 1.0. Se um argumento inteiro N é especificado, ele é usado como uma semente (produzindo uma sequência repetitiva):

```
mysql> SELECT RAND();
      -> 0.9233482386203
mysql> SELECT RAND(20);
      -> 0.15888261251047
mysql> SELECT RAND(20);
      -> 0.15888261251047
mysql> SELECT RAND();
      -> 0.63553050033332
mysql> SELECT RAND();
      -> 0.70100469486881
```

Você não pode usar uma coluna com valores RAND() em uma cláusula ORDER BY, pois ORDER BY avaliaria a coluna múltiplas vezes. Na versão 3.23 você pode fazer: `SELECT * FROM nome_tabela ORDER BY RAND()`

Isto é útil para obter uma amostra aleatória de um conjunto `SELECT * FROM tabela1,tabela2 WHERE a=b AND c<d ORDER BY RAND() LIMIT 1000`.

Note que um RAND() em uma cláusula WHERE será reavaliado toda vez que WHERE é executado.

RAND() não é um gerador de números aleatórios perfeito, mas é um modo rápido de se gerar números aleatórios ad hoc que serão portáveis entre plataformas para a mesma versão do MySQL.

LEAST(X, Y, ...)

Com dois ou mais argumentos, retorna o menor (valor-mínimo) argumento. Os argumentos são comparados usando as seguintes regras:

- Se o valor de retorno é usado em um contexto `INTEGER`, ou todos argumentos são valores inteiro, eles são comparados como inteiros.
- Se o valor de retorno é usado em um contexto `REAL`, ou todos argumentos são valores reais, eles são comparados como inteiros.
- Se qualquer um dos argumento for uma string caso-sensitivo, os argumentos são comparados como strings caso-sensitivo.
- Nos outros casos, os argumentos são comparados como strings caso-insensitivo:

```
mysql> SELECT LEAST(2,0);
-> 0
mysql> SELECT LEAST(34.0,3.0,5.0,767.0);
-> 3.0
mysql> SELECT LEAST("B","A","C");
-> "A"
```

Em versões do MySQL anteriores a versão 3.22.5, você pode usar `MIN()` no lugar de `LEAST`.

`GREATEST(X, Y, ...)`

Retorna o maior (valor máximo) argumento. Os argumentos são comparados usando as mesmas regras do `LEAST`:

```
mysql> SELECT GREATEST(2,0);
-> 2
mysql> SELECT GREATEST(34.0,3.0,5.0,767.0);
-> 767.0
mysql> SELECT GREATEST("B","A","C");
-> "C"
```

Em versões do MySQL anteriores a versão 3.22.5, você pode usar `MAX()` no lugar de `GRATEST`.

`DEGREES(X)`

Retorna o argumento `X`, convertido de radianos para graus:

```
mysql> SELECT DEGREES(PI());
-> 180.000000
```

`RADIANS(X)`

Retorna o argumento `X`, convertido de graus para radianos:

```
mysql> SELECT RADIANS(90);
-> 1.570796
```

`TRUNCATE(X,D)`

Retorna o número `X`, truncado para `D` casas decimais. Se `D` é 0, o resultado não terá ponto decimal ou parte fracionária:

```
mysql> SELECT TRUNCATE(1.223,1);
-> 1.2
mysql> SELECT TRUNCATE(1.999,1);
-> 1.9
mysql> SELECT TRUNCATE(1.999,0);
-> 1
```

```
mysql> SELECT TRUNCATE(-1.999,1);
-> -1.9
```

A partir do MySQL 3.23.51 todos o números são arredondados para zero.

Se D é negativo, então D numeros da parte inteira são zerados:

```
mysql> SELECT TRUNCATE(122,-2);
-> 100
```

Note que como os números decimais não são normalmente armazenados como números exatos, mas como valores double, você pode obter o seguinte resultado:

```
mysql> SELECT TRUNCATE(10.28*100,0);
-> 1027
```

O resultado acima acontece porque 10.28 é, na verdade, armazenado como 10.279999999999999.

6.3.4 Funções de Data e Hora

Veja [\[Tipos de data e hora\]](#), page [\[Tipos de data e hora\]](#) para um descrição da faixa de valores que cada tipo tem e os formatos válidos nos quais valores de de data e hora podese ser especificados.

Aqui está um exemplo que usa funções de data. A consulta seguinte seleciona todos os registros com um valores em uma coluna `col_data` dentro dos últimos 30 dias:

```
mysql> SELECT algo FROM nome_tabela
        WHERE TO_DAYS(NOW()) - TO_DAYS(col_data) <= 30;
```

DAYOFWEEK(data)

Retorna o índice do dia da semana para `data` (1 = Domingo, 2 = Segunda, ... 7 = Sábado). Estes valores de índices correspondem ao padrão ODBC.

```
mysql> SELECT DAYOFWEEK('1998-02-03');
-> 3
```

WEEKDAY(data)

Retorna o índice do dia das semana para `data` (0 = Segunda, 1 = Terça, ... 6 = Domingo):

```
mysql> SELECT WEEKDAY('1998-02-03 22:23:00');
-> 1
mysql> SELECT WEEKDAY('1997-11-05');
-> 2
```

DAYOFMONTH(data)

Retorna o dia do mês para `data`, na faixa de 1 até 31:

```
mysql> SELECT DAYOFMONTH('1998-02-03');
-> 3
```

DAYOFYEAR(data)

Retorna o dia do ano para `data`, na faixa de 1 até 366:

```
mysql> SELECT DAYOFYEAR('1998-02-03');
-> 34
```

MONTH(data)

Retorna o mês para *data*, na faixa de 1 até 12:

```
mysql> SELECT MONTH('1998-02-03');
-> 2
```

DAYNAME(data)

Retorna o nome do dia da semana para *data*:

```
mysql> SELECT DAYNAME("1998-02-05");
-> 'Thursday'
```

MONTHNAME(data)

Retorna o nome do mês para *data*:

```
mysql> SELECT MONTHNAME("1998-02-05");
-> 'February'
```

QUARTER(data)

Retorna o trimestre para *data*, na faixa de 1 até 4:

```
mysql> SELECT QUARTER('98-04-01');
-> 2
```

WEEK(data)**WEEK(data, primeiro)**

Com um único argumento, retorna a semana para *date*, na faixa de 0 a 53 (sim, pode ter o início de uma semana 53), para locais onde Domingo é o primeiro dia da semana. A forma de **WEEK** com dois argumentos permite especificar se a semana começa no Domingo ou na Segunda e se o valor de retorno de estar na faixa 0-53 ou 1-52.

Aqui está uma tabela para como o segundo argumento funciona:

Value	Meaning
0	Semana começa no Domingo e retorna valor na faixa 0-53
1	Semana começa na Segunda e retorna valor na faixa 0-53
2	Semana começa no Domingo e retorna valor na faixa 1-53
3	Semana começa na Segunda e retorna valor na faixa 1-53 (ISO 8601)

```
mysql> SELECT WEEK('1998-02-20');
-> 7
mysql> SELECT WEEK('1998-02-20', 0);
-> 7
mysql> SELECT WEEK('1998-02-20', 1);
-> 8
mysql> SELECT WEEK('1998-12-31', 1);
-> 53
```

Você pode definir o valor padrão do segundo argumento através de uma variável `default_week_format`. A sintaxe de `default_week_format` é:

```
SET [SESSION | GLOBAL] default_week_format = [0|1|2|3];
```

Nota: Na versão 4.0, `WEEK(#, 0)` foi alterado para corresponder ao calendário americano.

Note que se a semana for a última semana do ano anterior, MySQL retornará 0 se você não usar 2 ou 3 como argumento opcional:

```
mysql> SELECT YEAR('2000-01-01'), WEEK('2000-01-01',0);
      -> 2000, 0
mysql> SELECT WEEK('2000-01-01',2);
      -> 52
```

Alguém pode dizer que o MySQL deveria retornar 52 para a função WEEK() já que a data dada é, na verdade, a 52 semana de 1999. Nós decidimos retornar 0 já que queremos que função retorne 'o número da semana do ano dado'. Isto faz com que o uso da função WEEK() seja seguro quando combinado com outras funções que extraíam um parte de uma data.

Se você prefere saber a semana correta do ano, então você deve usar o 2 ou 3 como argumento opcional ou usar a função YEARWEEK():

```
mysql> SELECT YEARWEEK('2000-01-01');
      -> 199952
mysql> SELECT MID(YEARWEEK('2000-01-01'),5,2);
      -> 52
```

YEAR(data)

Retorna o ano para data na faixa de 1000 a 9999:

```
mysql> SELECT YEAR('98-02-03');
      -> 1998
```

YEARWEEK(data)

YEARWEEK(data,primeiro)

Retorna o ano e a semana para a data. O segundo argumento funciona exatamente como o segundo argumento de WEEK(). Note que o ano pode ser diferente do ano no argumento data para a primeira e a última semana do ano:

```
mysql> SELECT YEARWEEK('1987-01-01');
      -> 198653
```

Note que o número da semana é diferente do que seria retornado pela função WEEK() (0) para os argumentos opcionais 0 ou 1, já que WEEK() retorna a semana no contexto de um ano dado.

HOURL(hora)

Retorna a hora para hora, na faixa de 0 a 23:

```
mysql> SELECT HOUR('10:05:03');
      -> 10
```

MINUTE(hora)

Retorna o minuto para hora, na faixa de 0 a 59:

```
mysql> SELECT MINUTE('98-02-03 10:05:03');
      -> 5
```

SECOND(hora)

Retorna o segundo para hora, na faixa de 0 a 59:

```
mysql> SELECT SECOND('10:05:03');
      -> 3
```

PERIOD_ADD(P, N)

Adiciona N meses ao período P (no formato AAMM ou AAAAMM). Retorna um valor no formato AAAAMM.

Note que o argumento de período P **não** é um valor de data:

```
mysql> SELECT PERIOD_ADD(9801,2);
-> 199803
```

PERIOD_DIFF(P1, P2)

Retorna o número de meses entre os períodos P1 e P2. P1 e P2 devem estar no formato AAMM ou AAAAMM.

Note que os argumentos de período P1 e P2 **não** são valores de data:

```
mysql> SELECT PERIOD_DIFF(9802,199703);
-> 11
```

DATE_ADD(data, INTERVAL tipo expr)

DATE_SUB(data, INTERVAL tipo expr)

ADDDATE(data, INTERVAL tipo expr)

SUBDATE(data, INTERVAL tipo expr)

Estas funções realizam operações aritméticas em datas. **ADDDATE()** e **SUBDATE()** são sinônimos para **DATE_ADD()** e **DATE_SUB()**.

Na versão 3.23 do MySQL, você pode usar + e - ao invés de **DATE_ADD()** e **DATE_SUB()** se a expressão do lado direito é um coluna date ou datetime. (Veja exemplo abaixo.)

data é um valor **DATETIME** ou **DATE** especificando a data de início. **expr** is an expressão especificando o intervalo a ser adicionado ou subtraído da data de início. **expr** é uma string; ela pode iniciar com um '-' para intervalos negativos. **type** é uma palavra chave indicando como a expressão deve ser interpretada.

A seguinte tabela mostra como os argumentos **tipo** e **expr** se relacionam:

tipo do valor	Formarto esperado da expr
SECOND	SEGUNDOS
MINUTE	MINUTOS
HOUR	HORAS
DAY	DIAS
MONTH	MESES
YEAR	ANOS
MINUTE_SECOND	"MINUTOS:SEGUNDOS"
HOUR_MINUTE	"HORAS:MINUTOS"
DAY_HOUR	"DIAS HORAS"
YEAR_MONTH	"ANOS-MESES"
HOUR_SECOND	"HORAS:MINUTOS:SEGUNDOS"
DAY_MINUTE	"DIAS HORAS:MINUTOS"
DAY_SECOND	"DIAS HORAS:MINUTOS:SEGUNDOS"

O MySQL permite qualquer delimitador de pontuação no formato de **expr**. Os delimitadores mostrados na tabela são apenas sugeridos. Se o argumento **date** é um valor de **DATA** e seus cálculos envolvem apenas as partes **ANO**, **MS**, e **DIA** (into é, nenhuma parte de hora), o resultado é um valor do tipo **DATE**. Senão, o resultado é um valor do tipo **DATETIME**:

```

mysql> SELECT "1997-12-31 23:59:59" + INTERVAL 1 SECOND;
      -> 1998-01-01 00:00:00
mysql> SELECT INTERVAL 1 DAY + "1997-12-31";
      -> 1998-01-01
mysql> SELECT "1998-01-01" - INTERVAL 1 SECOND;
      -> 1997-12-31 23:59:59
mysql> SELECT DATE_ADD("1997-12-31 23:59:59",
      ->          INTERVAL 1 SECOND);
      -> 1998-01-01 00:00:00
mysql> SELECT DATE_ADD("1997-12-31 23:59:59",
      ->          INTERVAL 1 DAY);
      -> 1998-01-01 23:59:59
mysql> SELECT DATE_ADD("1997-12-31 23:59:59",
      ->          INTERVAL "1:1" MINUTE_SECOND);
      -> 1998-01-01 00:01:00
mysql> SELECT DATE_SUB("1998-01-01 00:00:00",
      ->          INTERVAL "1 1:1:1" DAY_SECOND);
      -> 1997-12-30 22:58:59
mysql> SELECT DATE_ADD("1998-01-01 00:00:00",
      ->          INTERVAL "-1 10" DAY_HOUR);
      -> 1997-12-30 14:00:00
mysql> SELECT DATE_SUB("1998-01-02", INTERVAL 31 DAY);
      -> 1997-12-02

```

Se você especificado um intervalo muito curto (não inclui todas as partes que seriam esperadas pelo intervalo para aquele tipo), MySQL assume que você não especificou a parte mais a esquerda do valor do intervalo. Por exemplo, se você especifica um tipo `DAY_SECOND`, o valor esperado de `expr` deverá ter as partes de dias, horas, minutos e segundos. Se você especifica um valor como `"1:10"`, MySQL assume que as partes do dia e da hora foram esquecidas e o valor representa minutos e segundos. Em outras palavras, `"1:10" DAY_SECOND` é interpretado de forma equivalente a `"1:10" MINUTE_SECOND`. Isto é análogo a forma que o MySQL interpreta valores `TIME` representado tempo decorrido no lugar de hora do dia.

Note que se você adicionar ou subtrair um valor contendo data com um outro contendo uma parte de hora, o valor data será automaticamente convertido para um valor `datetime`:

```

mysql> SELECT DATE_ADD("1999-01-01", INTERVAL 1 DAY);
      -> 1999-01-02
mysql> SELECT DATE_ADD("1999-01-01", INTERVAL 1 HOUR);
      -> 1999-01-01 01:00:00

```

Se você utilizar datas incorretas, o valor retornado `NULL`. Se você adicionar `MONTH`, `YEAR_MONTH`, ou `YEAR` e a data resultante tiver um dia maior que o dia máximo para aquele mês, o dia é ajustado para o dia máximo no mês.

```

mysql> SELECT DATE_ADD('1998-01-30', INTERVAL 1 MONTH);
      -> 1998-02-28

```

Note pelo exemplo anterior que a palavra `INTERVAL` e a palavra chave `type` não são caso sensetivo.

EXTRACT(tipo FROM data)

A função `EXTRACT()` usa o mesmo tipo de intervalo especificado como `DATE_ADD()` ou `DATE_SUB()`, mas extrai partes da data em vez de realizar aritmética de data.

```
mysql> SELECT EXTRACT(YEAR FROM "1999-07-02");
-> 1999
mysql> SELECT EXTRACT(YEAR_MONTH FROM "1999-07-02 01:02:03");
-> 199907
mysql> SELECT EXTRACT(DAY_MINUTE FROM "1999-07-02 01:02:03");
-> 20102
```

TO_DAYS(data)

Dada uma data `data`, retorna o número do dia (o número de dias desde o ano 0);

```
mysql> SELECT TO_DAYS(950501);
-> 728779
mysql> SELECT TO_DAYS('1997-10-07');
-> 729669
```

`TO_DAYS()` não pode ser usado com valores que precedem o advento do calendário Gregoriano (1582), porque ele não leva em conta os dias perdidos quando o calendário foi mudado.

FROM_DAYS(N)

Dado um número de dia `N`, retorna um valor `DATE`:

```
mysql> SELECT FROM_DAYS(729669);
-> '1997-10-07'
```

`FROM_DAYS()` não pode ser usado com valores que precedem o advento do calendário Gregoriano (1582), porque ele não leva em conta os dias perdidos quando o calendário foi mudado.

DATE_FORMAT(data, formato)

Formata o valor de `data` de acordo com a string `formato` string. Os seguintes identificadores podem ser utilizados na string `formato`:

Specifier	Description
%M	Nome do mês (January..December)
%W	Nome da semana (Sunday..Saturday)
%D	Dia do mês com sufixo Inglês (0th, 1st, 2nd, 3rd, etc.)
%Y	Ano, numerico, 4 digitos
%y	Ano, numerico, 2 digitos
%X	Ano para a semana onde o Domingo é o primeiro dia da semana, numerico, 4 digitos, usado com '%V'
%x	Ano para a semana onde a segunda é o primeiro dia da semana, numerico, 4 digitos, usado com '%v'
%a	Nome da semana abreviado (Sun..Sat)
%d	Dia do mês, numerico (00..31)
%e	Dia do mês, numerico (0..31)
%m	Mês, numerico (00..12)
%c	Mês, numerico (0..12)

%b	Nome do mês abreviado (Jan..Dec)
%j	Dia do ano (001..366)
%H	Hora (00..23)
%k	Hora (0..23)
%h	Hora (01..12)
%I	Hora (01..12)
%l	Hora (1..12)
%i	Minutos, numerico (00..59)
%r	Tempo, 12-horas (hh:mm:ss [AP]M)
%T	Tempo, 24-horas (hh:mm:ss)
%S	Segundos (00..59)
%s	Segundos (00..59)
%p	AM ou PM
%w	Dia da semana (0=Domingo..6=Sabado)
%U	Semana(00..53), onde o Domingo é o primeiro dia da semana.
%u	Semana(00..53), onde a Segunda é o primeiro dia da semana.
%V	Semana(01..53), onde o Domingo é o primeiro dia da semana. Usado com '%X'
%v	Semana(01..53), onde a Segunda é o primeiro dia da semana. Usado com '%x'
%%	Um literal '%'

Todos os outros caracteres são apenas copiados para o resultado, sem interpretação:

```
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y');
-> 'Saturday October 1997'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%H:%i:%s');
-> '22:23:00'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
                          '%D %y %a %d %m %b %j');
-> '4th 97 Sat 04 10 Oct 277'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
                          '%H %k %I %r %T %S %w');
-> '22 22 10 10:23:00 PM 22:23:00 00 6'
mysql> SELECT DATE_FORMAT('1999-01-01', '%X %V');
-> '1998 52'
```

Como na versão 3.23 do MySQL, o caracter '%' é exigido antes dos caracteres de especificação de formato. Em versões anteriores do MySQL '%' era opcional.

A razão para a faixa de valores do mês e do dia começarem com zero é que o MySQL permite datas incompletas tais como '2004-00-00' serem armazenadas no MySQL 3.23.

TIME_FORMAT(hora, formato)

É usado como a função DATE_FORMAT() acima, mas a string de formato pode conter apenas os especificadores de formato que tratam de horas, minutos e segundos. Outros especificadores produzem um valor NULL ou 0.

`CURDATE()`

`CURRENT_DATE`

Retorna a data de hoje como um valor no formato 'YYYY-MM-DD' ou YYYYMMDD, dependendo se a função é usada num contexto numérico ou de string.

```
mysql> SELECT CURDATE();
-> '1997-12-15'
mysql> SELECT CURDATE() + 0;
-> 19971215
```

`CURTIME()`

`CURRENT_TIME`

Retorna a hora atual como um valor no formato 'HH:MM:SS' ou HHMMSS,

```
mysql> SELECT CURTIME();
-> '23:50:26'
mysql> SELECT CURTIME() + 0;
-> 235026
```

`NOW()`

`SYSDATE()`

`CURRENT_TIMESTAMP`

Retorna a data e hora atual como um valor no formato 'YYYY-MM-DD HH:MM:SS' ou YYYYMMDDHHMMSS, dependendo se a função é utilizada num contexto numérico ou de string.

```
mysql> SELECT NOW();
-> '1997-12-15 23:50:26'
mysql> SELECT NOW() + 0;
-> 19971215235026
```

Note que `NOW()` só é avaliado uma vez por consulta, definido no início da execução. Isto significa que múltiplas referências a `NOW()` em uma única consulta irá sempre retornar o mesmo valor.

`UNIX_TIMESTAMP()`

`UNIX_TIMESTAMP(data)`

Se chamado sem argumento, retorna um tipo timestamp do Unix (segundos desde '1970-01-01 00:00:00' GMT) como um inteiro sem sinal. Se `UNIX_TIMESTAMP()` é chamada com um argumento `data`, é retornado o valor do argumento como segundo desde '1970-01-01 00:00:00' GMT. `data` pode ser um string `DATE`, uma string `DATETIME`, um `TIMESTAMP`, ou um número no formato `YYMMDD` ou `YYYYMMDD` na hora local:

```
mysql> SELECT UNIX_TIMESTAMP();
-> 882226357
mysql> SELECT UNIX_TIMESTAMP('1997-10-04 22:23:00');
-> 875996580
```

Quando `UNIX_TIMESTAMP` é usado em uma coluna `TIMESTAMP`, a função retornará o valor timestamp interno diretamente, sem nenhuma conversão "string-para-unix-timestamp" implícita. Se você passar uma data fora da faixa para `UNIX_TIMESTAMP()`, a função irá retornar 0, mas por favor note que só verificações básicas são realizadas. (ano 1970-2037, mês 01-12, dia 01-31).

Se você subtrair colunas `UNIX_TIMESTAMP()`, você pode querer mudar o resultado para inteiro com sinal. See [\(undefined\) \[Funções de tipagem\]](#), page [\(undefined\)](#).

`FROM_UNIXTIME(unix_timestamp [,formato])`

Retorna a representação do argumento `unix_timestamp` como um valor no formato `'YYYY-MM-DD HH:MM:SS'` ou `YYYYMMDDHHMMSS`, dependendo de do contexto em que a função é utilizada:

Se o `formato` é dado o resultado é formatado de acordo com a string `formato`. `formato` pode conter os especificadores listados acima para a função `DATE_FORMAT()`

```
mysql> SELECT FROM_UNIXTIME(875996580);
-> '1997-10-04 22:23:00'
mysql> SELECT FROM_UNIXTIME(875996580) + 0;
-> 19971004222300
mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(),
                           '%Y %D %M %h:%i:%s %x');
-> '1997 23rd December 03:43:30 1997'
```

`SEC_TO_TIME(seconds)`

Retorna o argumento `segundos`, convertido em horas, minutos e segundos como um valor no formato `'HH:MM:SS'` ou `HHMMSS`, dependendo do contexto em que a função é utilizada:

```
mysql> SELECT SEC_TO_TIME(2378);
-> '00:39:38'
mysql> SELECT SEC_TO_TIME(2378) + 0;
-> 3938
```

`TIME_TO_SEC(time)`

Retorna o argumento `time`, convertido em segundos:

```
mysql> SELECT TIME_TO_SEC('22:23:00');
-> 80580
mysql> SELECT TIME_TO_SEC('00:39:38');
-> 2378
```

6.3.5 Funções de Conversão

A sintaxe da função `CAST` é:

`CAST(expressão AS tipo)`

ou

`CONVERT(expressão,tipo)`

Onde `tipo` é um dos:

- `BINARY`
- `CHAR` (Novo na versão 4.0.6)
- `DATE`
- `DATETIME`

- SIGNED {INTEGER}
- TIME
- UNSIGNED {INTEGER}

CAST() é da sintaxe SQL-99 syntax e CONVERT() é da sintaxe ODBC.

A função de conversão é principalmente útil quando você deseja criar uma coluna com um tipo específico em uma CREATE ... SELECT:

```
CREATE TABLE nova_tabela SELECT CAST('2000-01-01' AS DATE);
```

CAST(string AS BINARY é a mesma coisa que BINARY string. CAST(expr AS CHAR trata a expressão como sendo uma string com o conjunto de caracteres padrão.

Para converter uma string para um valor numérico, normalmente não é necessário se fazer nada; apenas use a string como se fosse um número:

```
mysql> SELECT 1+'1';
-> 2
```

Se você usar um número em um contexto string, o número será convertido automaticamente para uma string BINARY.

```
mysql> SELECT CONCAT("hello you ",2);
-> "hello you 2"
```

O MySQL suporta aritmético com valores de 64 bits com sinal e sem sinal. Se você está usando um operação numérica (como +) e um dos operandos é **unsigned integer** (inteiro sem sinal), o resultado também será sem sinal (unsigned). Você pode forçar o tipo usando os operadores de conversão SIGNED e UNSIGNED, os quais irão converter a operação para um inteiro de 65 bits com sinal e sem sinal, respectivamente.

```
mysql> SELECT CAST(1-2 AS UNSIGNED)
-> 18446744073709551615
mysql> SELECT CAST(CAST(1-2 AS UNSIGNED) AS SIGNED);
-> -1
```

Note que se um dos operadores for um valor de ponto flutuante (neste contexto DECIMAL()) é considerado um valor de ponto flutuante) o resultado será um valor de ponto flutuante e não é afetado pela regra acima.

```
mysql> SELECT CAST(1 AS UNSIGNED) -2.0
-> -1.0
```

Se você estiver utilizando uma string em uma operação aritmética, ela é convertida para um número de ponto flutuante.

As funções CAST() e CONVERT() foram adicionadas no MySQL 4.0.2.

O tratamento de valores sem sinais foi mudado no MySQL 4.0 para suportar valores BIGINT apropriadamente. Se você tiver algum código que deseja executar no MySQL 4.0 e 3.23 (casos em que você provavelmente não poderá usar a função CAST), você pode utilizar o seguinte truque para conseguir um resultado com sinal quando subtraindo duas colunas do tipo unsigned integer (inteiro sem sinal):

```
SELECT (coluna_sem_sinal_1+0.0)-(coluna_sem_sinal_2+0.0);
```

A idéia é que as colunas sejam convertidas para ponto flutuante antes de se fazer a subtração.

Se você tiver algum problema com colunas UNSIGNED no seu aplicação MySQL antiga ao portar para o MySQL 4.0, você pode usar a opção `--sql-mode=NO_UNSIGNED_SUBTRACTION`

ao iniciar `mysqld`. Note, no entanto, que enquanto você utilizar esta opção, não será possível conseguir um uso efetivo do tipo de coluna `UNSIGNED BIGINT`.

6.3.6 Outras Funções

6.3.6.1 Funções Binárias

O MySQL utiliza aritmética `BIGINT` (64bits) para operações binárias, assim estes operadores possuem uma faixa máxima de 64 bits.

| Operador binário OR

```
mysql> SELECT 29 | 15;
-> 31
```

O resultado é um inteiro sem sinal de 64 bits.

& Operado binário AND

```
mysql> SELECT 29 & 15;
-> 13
```

O resultado é um inteiro sem sinal de 64 bits.

^ Operado binário XOR

```
mysql> SELECT 1 ^ 1;
-> 0
mysql> SELECT 1 ^ 0;
-> 1
mysql> SELECT 11 ^ 3;
-> 8
```

O resultado é um inteiro sem sinal de 64 bits.

XOR foi adicionado na versão 4.0.2.

<<

Desloca um numero `BIGINT` (muito grande) a esquerda:

```
mysql> SELECT 1 << 2;
-> 4
```

O resultado é um inteiro sem sinal de 64 bits.

>> Desloca um numero `BIGINT` (muito grande) a direita:

```
mysql> SELECT 4 >> 2;
-> 1
```

O resultado é um inteiro sem sinal de 64 bits.

~ Inverte todos os bits:

```
mysql> SELECT 5 & ~1;
-> 4
```

O resultado é um inteiro sem sinal de 64 bits.

`BIT_COUNT(N)`

Retorna o número de bits que são passados no argumento `N`:

```
mysql> SELECT BIT_COUNT(29);
-> 4
```

6.3.6.2 Funções Diversas

DATABASE()

Retorna o nome do banco de dados atual:

```
mysql> SELECT DATABASE();
-> 'test'
```

Se nenhum banco de dados estiver selecionado, DATABASE() retorna uma string vazia.

USER()

SYSTEM_USER()

SESSION_USER()

retorna o nome do usuário MySQL atual:

```
mysql> SELECT USER();
-> 'davida@localhost'
```

Na versão 3.22.11 do MySQL ou posterior, são incluídos o nome de máquina do cliente e o nome do usuário. Você pode extrair apenas a parte do nome da seguinte forma (o qual funciona também com a parte do nome de máquina):

```
mysql> SELECT SUBSTRING_INDEX(USER(), "@", 1);
-> 'davida'
```

CURRENT_USER()

Retorna o nome do usuário, como foi autenticado na sessão atual

```
mysql> SELECT USER();
-> 'davida@localhost'
mysql> SELECT * FROM mysql.user;
-> ERROR 1044: Access denied for user: '@localhost' to database 'mysql'
mysql> SELECT CURRENT_USER();
-> '@localhost'
```

PASSWORD(str)

OLD_PASSWORD(str)

Calcula a senha a partir de senha **str** em texto puro. Está é a função que é utilizada para criptografar a senha do MySQL para armazenamento na coluna Password da tabela de permissões user

```
mysql> SELECT PASSWORD('badpwd');
-> '7f84554057dd964b'
```

A criptografia de PASSWORD() não é reversível.

PASSWORD() não realiza a criptografia da senha da mesma maneira que as senhas Unix são criptografadas. Veja ENCRYPT().

Note: A função PASSWORD() é usada pelo sistema de autenticação no servidor MySQL, você NÃO deve utilizá-las em suas próprias aplicações. Para este propósito utilize MD5() ou SHA1(). Veja também RFC-2195 para maiores informações sobre o tratamento de senha e autenticação segura em suas aplicações.

ENCRYPT(str[,salt])

Criptografa **str** utilizando a chamada de sistema crypt() do Unix. O argumento **salt** deve ser uma string com dois caracteres. (Na versão 3.22.16 do MySQL, salt deve ser maior que dois caracteres.):

```
mysql> SELECT ENCRYPT("hello");
      -> 'VxuFAJXVARROc'
```

Se `crypt()` não estiver disponível no seu sistema, `ENCRYPT()` sempre retorna `NULL`.

`ENCRYPT()` ignora tudo depois dos primeiros 8 caracteres de `str`, pelo menos em alguns sistemas. Isto é determinado pelo comportamento da chamada de sistema `crypt()`.

`ENCODE(str, senha_str)`

Criptografa `str` usando `senha_str` como a senha. Para descriptografar o resultado, utilize `DECODE()`.

O resultado é uma string binária do mesmo tamanho de `str`. Se você deseja salvá-la em uma coluna, use uma coluna do tipo `BLOB`.

`DECODE(cript_str, senha_str)`

Descriptografa o string criptografada `cript_str` usando `senha_str` como a senha. `cript_str` deve ser uma string retornada de `ENCODE()`.

`MD5(string)`

Calcula um checksum MD5 de 128 bits para a string. O valor é retornado como um número hexadecimal de 32 dígitos que pode, por exemplo, ser usado como uma chave hash:

```
mysql> SELECT MD5("testing");
      -> 'ae2b1fca515949e5d54fb22b8ed95575'
```

Este é o "RSA Data Security, Inc. MD5 Message-Digest Algorithm".

`SHA1(string)`

`SHA(string)`

Calcula um checksum SHA1 de 160 bit para a string, como descrito no RFC 3174 (Algoritmo Hash de Segurança). O valor é retornado como um número hexadecimal de 40 dígitos, ou `NULL` no caso do argumento ser `NULL`. Uma das possibilidades para o uso desta função é a chave hash. Você também pode usá-la como uma função segura de criptografia para armazenar senhas.

```
mysql> SELECT SHA1("abc");
      -> 'a9993e364706816aba3e25717850c26c9cd0d89d'
```

`SHA1()` foi adicionado na versão 4.0.2, e pode ser considerada um equivalente ao `MD5()` com criptografia mais segura. `SHA()` é um sinônimo para `SHA1()`.

`AES_ENCRYPT(string, string_chave)`

`AES_DECRYPT(string, string_chave)`

Estas funções permitem criptografia/descriptografia de dados usando o algoritmo oficial AES (Padrão Avançado de Criptografia), antes conhecido como Rijndael. Criptografia com uma chave de 128 bits podem ser usadas, mas você pode estendê-la para 256 bits através da fonte. Nós escolhemos 128 bits porque é muito mais rápido e é bastante seguro.

Os argumentos de entrada podem ser de qualquer tamanho. Se ambos argumentos são `NULL`, o resultado desta função também será `NULL`.

Como o AES é um algoritmo de nível de bloco, padding é usado para codificar strings de tamanho ímpares e então a string resultante pode ser calculada como $16 * (\text{trunc}(\text{tamanho_string}/16) + 1)$.

Se `AES_DECRYPT()` detectar dados inválidos ou padding incorreto, ela retornará `NULL`. No entanto, é possível para o `AES_DECRYPT()` retornar um valor não-`NULL` (possivelmente lixo) se os dados de entrada ou a chave forem inválidos

Você pode usar as funções AES para armazenar dados de forma criptografada modificando as suas consultas:

```
INSERT INTO t VALUES (1,AES_ENCRYPT("texto","senha"));
```

Você pode obter mais segurança evitando transferir a chave em suas conexões a cada consulta, o que pode ser conseguido armazenando-o em variáveis do lado do servidor na hora das conexões.

```
SELECT @senha:="minha senha";
INSERT INTO t VALUES (1,AES_ENCRYPT("texto",@senha));
```

`AES_ENCRYPT()` e `AES_DECRYPT()` foram adicionados na versão 4.0.2, e podem ser considerados a função de criptografia mais segura atualmente disponível no MySQL.

`DES_ENCRYPT(string_para_criptografar [, (numero_chave | chave_string)])`

Criptografa a string com a chave dada utilizando o algoritmo Triplo-DES.

Note que esta função só funciona se você tiver configurado o MySQL com suporte a SSL. See [\(undefined\)](#) [Conexões seguras], page [\(undefined\)](#).

A chave de criptografia utilizada é escolhida da seguinte forma:

Argumento	Descrição
Somente argumento	um A primeira chave de <code>des-key-file</code> é utilizada.
Número da chave	A chave dada (0-9) de <code>des-key-file</code> é utilizada.
string	A <code>chave_string</code> dada será utilizada para criptografar <code>string_para_criptografar</code> .

O string retornada será uma string binária onde o primeiro caracter será `CHAR(128 | número_chave)`.

O 128 é adicionado para facilitar o reconhecimento da chave de criptografia. Se você usar uma chave string, `numéro_chave` será 127.

Havendo erro, esta função retorna `NULL`.

O tamanho da string para o resultado será `novo_tamanho= tamanho_orig + (8-(tamanho_orig % 8))+1`.

O `des-key-file` terá o seguinte formato:

```
numero_chave chave_string_des
numero_chave chave_string_des
```

Cada `numero_chave` deve ser um número na faixa de 0 a 9. As linhas do arquivo podem estar em qualquer ordem. `chave_string_des` é a string que será usada para criptografar a mensagem. Entre o número e a chave deve haver pelo menos um espaço. A primeira chave é a chave padrão que será utilizada se não for especificada nenhuma chave como argumento para `DES_ENCRYPT()`

Você pode dizer ao MySQL para ler novos valores de arquivos de chave com o comando `FLUSH DES_KEY_FILE`. Isto exige o privilégio `Reload_priv`.

Um benefício de ter um conjunto de chaves padrões é que ele dá a aplicação um modo de verificar a existência de valores criptografados em colunas, sem dar ao usuário final o direito de descriptografar estes valores.

```
mysql> SELECT endereco_clientes FROM tabela_clientes WHERE
        cartao_credito_criptografado = DES_ENCRYPT("numero_cartao_
        credito");
```

`DES_DECRYPT(string_para_descriptografar [, chave_string])`

Descriptografa uma string criptografada com `DES_ENCRYPT()`.

Note que esta função só funciona se você tiver configurado o MySQL com suporte SSL. See [\[Conexões seguras\]](#), page [\[Conexões seguras\]](#).

Se nenhum argumento `chave_string` for dado, `DES_DECRYPT()` examina o primeiro byte da string criptografada para determinar o número de chave DES que foi usado para criptografar a string original, e então lê a chave de `des-key-file` para descriptografar a mensagem. Para isto funcionar o usuário deve ter o privilégio `SUPER`.

Se você passar para esta função um argumento `chave_string`, aquela string é usada como a chave para descriptografar a mensagem.

Se a `string_para_descriptografar` não se parecer com uma string criptografada, o MySQL retornará a `string_para_descriptografar` dada.

Havendo erro, esta função retorna `NULL`.

`LAST_INSERT_ID([expr])`

Retorna o último valor gerado automaticamente que tenha sido inserido em um coluna `AUTO_INCREMENT`. See [\[mysql_insert_id\(\)\]](#), page [\[mysql_insert_id\(\)\]](#).

```
mysql> SELECT LAST_INSERT_ID();
-> 195
```

O último ID que foi gerado e mantido no servidor em uma base por conexão. Ele não será alterado por outro cliente. Ele nem mesmo será atualizado se você atualizar outra coluna `AUTO_INCREMENT` com um valor não-mágico (Isto é, um valor que não seja `NULL` e nem 0).

Se você inserir muitos registros ao mesmo tempo com uma instrução `insert`, `LAST_INSERT_ID()` retorna o valor da primeira linha inserida. A razão para isto é tornar possível reproduzir facilmente a mesma instrução `INSERT` em algum outro servidor.

Se `expr` é dado com um argumento para `LAST_INSERT_ID()`, então o valor do argumento é retornado pela função e é configurado como o próximo valor para ser retornado pela `LAST_INSERT_ID()`. Isto pode ser útil para simular sequências:

Primeiro crie a tabela:

```
mysql> CREATE TABLE sequencia (id INT NOT NULL);
mysql> INSERT INTO sequencia VALUES (0);
```

Então a tabela pode ser usada para gerar sequência de números como estes:

```
mysql> UPDATE sequencia SET id=LAST_INSERT_ID(id+1);
```

Você pode gerar sequências sem chamar `LAST_INSERT_ID()`, mas a utilidade de se usar a função deste modo é que o valor ID é mantido no servidor como o último valor gerado automaticamente (seguro para multi-usurário). Você pode recuperar a nova ID como você leria qualquer valor `AUTO_INCREMENT` normal no MySQL. Por exemplo, `LAST_INSERT_ID()` (sem um argumento) retornará a nova ID. A função `mysql_insert_id()` da API C também pode ser usada para obter o valor.

Note que como `mysql_insert_id()` só é atualizado depois de instruções `INSERT` e `UPDATE`, você não pode utilizar a função da API C para recuperar o valor para `LAST_INSERT_ID(expr)` depois de executar outra instrução SQL como `SELECT` ou `SET`.

FORMAT(X,D)

Formata o número X com um format como '`#,###,###.##`', arredondado para D casas decimais. Se D é 0, o resultado não terá nenhum ponto decimal ou parte fracionária:

```
mysql> SELECT FORMAT(12332.123456, 4);
-> '12,332.1235'
mysql> SELECT FORMAT(12332.1,4);
-> '12,332.1000'
mysql> SELECT FORMAT(12332.2,0);
-> '12,332'
```

VERSION()

Retorna uma string indicando a versão do servidor MySQL:

```
mysql> SELECT VERSION();
-> '3.23.13-log'
```

Note que se seu versão finalizar com `-log`, significa que o log está habilitado.

CONNECTION_ID()

Retorna a identificação (`thread_id`) desta conexão. Cada conexão tem seu próprio id único:

```
mysql> SELECT CONNECTION_ID();
-> 1
```

GET_LOCK(str, tempo_limite)

Tenta conseguir uma trava com o nome dado pela string `str`, com um tempo limite de `timeout` segundos. Retorna 1 se o bloqueio foi obtido com sucesso, 0 se o tempo esgotou ou NULL se uma erro ocorreu (tal como estouro de memória ou a threado tiver sido finalizada com `mysqladmin kill`). Uma trava é liberada quando você executa `RELEASE_LOCK()`, executa uma nova `GET_LOCK()`, ou a thread termina. Esta função pode ser usada para implementar bloqueio de aplicação ou para simular registros travados. Ela bloqueia pedido de outros clientes para travas com o mesmo nome; clientes que concordam com um dado nome da trava podem usar a string para realizar travamento de consultas cooperativos:

```
mysql> SELECT GET_LOCK("lock1",10);
```

```

-> 1
mysql> SELECT IS_FREE_LOCK("lock2");
-> 1
mysql> SELECT GET_LOCK("lock2",10);
-> 1
mysql> SELECT RELEASE_LOCK("lock2");
-> 1
mysql> SELECT RELEASE_LOCK("lock1");
-> NULL

```

Note que a segunda chamada de `RELEASE_LOCK()` retorna `NULL` porque a trava "lock1" foi liberada automaticamente pela segunda. `GET_LOCK()` call.

`RELEASE_LOCK(str)`

Libera a trava nomeada pela string `str` que foi obtida com `GET_LOCK()`. Retorna 1 se a trava foi liberada, 0 se a trava não foi bloqueada pela thread (caso onde a trava não é liberada), e `NULL` se o nome da trava não existe. A trava nunca existirá se ela nunca for obtida pela chamada de `GET_LOCK()` ou se ela já tiver sido liberada.

A instrução `DO` é conveniente para ser utilizada com `RELEASE_LOCK()`. See [<undefined> \[DO\], page <undefined>](#).

`IS_FREE_LOCK(str)`

Verifica se a trava chamada `str` está livre para ser utilizada (ex. não está bloqueada). Retorna 1 se a trava está livre (ninguém a esta usando), 0 se a trava está em uso, e `NULL` caso ocorra erro (como argumentos incorretos).

`BENCHMARK(cont, expr)`

A função `BENCHMARK()` executa a expressão `expr` repetidamente `cont` vezes. Ela pode ser usada para medir a velocidade em que o MySQL processa a expressão. O valor resultante é sempre 0. A intenção é usá-la no cliente `mysql`, relatando o tempo de execução da consulta:

```

mysql> SELECT BENCHMARK(1000000,ENCODE("hello","goodbye"));
+-----+
| BENCHMARK(1000000,ENCODE("hello","goodbye")) |
+-----+
|                                     0 |
+-----+
1 row in set (4.74 sec)

```

O tempo relatado é o tempo decorrido no cliente, não o tempo de CPU no servidor. Pode ser aconselhável executar `BENCHMARK()` diversas vezes e interpretar o resultado considerado o peso da carga da máquina servidora.

`INET_NTOA(expr)`

Dado um endereço numérico de rede (4 ou 8 bytes), retorna a representação no formato com pontos do endereço como uma string:

```

mysql> SELECT INET_NTOA(3520061480);
-> "209.207.224.40"

```

INET_ATON(expr)

Dada a representação com pontos de um endereço de rede como uma string, retorna um inteiro que representa o valor numérico deste endereço. Endereços podem ter 4 ou 8 bytes de endereçamento:

```
mysql> SELECT INET_ATON("209.207.224.40");
-> 3520061480
```

O número gerado é sempre na ordem de bytes da rede; por exemplo o número acima é calculado como $209*256^3 + 207*256^2 + 224*256 + 40$.

MASTER_POS_WAIT(nome_log, log_pos [, tempo_limite])

Envia blocos o escravo alcançar (ex.: ter lido e aplicado todas as atualizações) a posição específica no log mestre. Se a informação mestre não está inicializada, ou se os argumentos estão incorretos, retorna NULL. Se o escravo não está em execução, enviará blocos e irá esperar até que ele seja iniciado e vá para (ou passe por) a posição especificada. Se o escravo já passou pela posição especificada, retorna imediatamente.

Se **tempo_limite** (novo na versão 4.0.10) é especificado, irá esperar até que **tempo_limite** segundos tenham se passado. **tempo_limite** deve ser maior que 0; zero ou um **tempo_limite** negativo significa sem tempo_limite. O valor de retorno é o número de eventos de log que ele tem que esperar para obter a posição especificada, NULL no caso de erro, ou -1 se o tempo_limite tiver sido excedido.

O comando é útil para controle de sincronização no mestre-escravo.

FOUND_ROWS()

Retorna o número de linhas que o comando **SELECT SQL_CALC_FOUND_ROWS ...** anterior teria retornado, se ele não tiver sido restringido com **LIMIT**.

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM nome_tabela
      WHERE id > 100 LIMIT 10;
mysql> SELECT FOUND_ROWS();
```

O segundo **SELECT** irá retornar um número indicando quantas linhas o primeiro **SELECT** teria retornado se ele fosse escrito sem a cláusula **LIMIT**.

Note que se você estiver usando **SELECT SQL_CALC_FOUND_ROWS ...**, o MySQL tem que calcular todos os registros no conjunto de resultados. No entanto, isto é mais rápido que se você não utilizar **LIMIT**, já que o resultado precisa ser enviado ao cliente.

O valor de **FOUND_ROWS()** só é relevante logo após um comando **SELECT SQL_CALC_FOUND_ROWS**.

SQL_CALC_FOUND_ROWS está disponível a partir da versão 4.0.0 do MySQL.

6.3.7 Funções para Usar com Cláusulas GROUP BY

Se você utiliza um função de agrupamento em uma instrução que não contenha um cláusula **GROUP BY**, equivale a fazer um agrupamento com todos os registros.

COUNT(expr)

Retorna a quantidade de valores não-NULL nos registros recuperados por uma instrução **SELECT**:

```
mysql> SELECT estudante.nome_estudante,COUNT(*)
->      FROM estudante,curso
->      WHERE estudante.id_estudante=curso.id_estudante
->      GROUP BY nome_estudante;
```

COUNT(*) é um pouco diferente de retornar o número de registros recuperados, quando eles possuírem valores NULL.

COUNT(*) é otimizado para retornar muito rápido se SELECT recuperar registros de uma tabela, nenhuma outra coluna for retornada, e não houver nenhuma cláusula WHERE. Por exemplo:

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

COUNT(DISTINCT expr, [expr...])

Retorna a quantidade de registros com valores não-NULL diferentes:

```
mysql> SELECT COUNT(DISTINCT resultados) FROM estudante;
```

No MySQL você pode obter o número de combinação de expressões distintas que não contém NULL fornecendo uma lista de expressões. No SQL-99 você teria que concatenar todas as expressões utilizando COUNT(DISTINCT ...).

AVG(expr)

Retorna o valor médio de expr:

```
mysql> SELECT nome_estudante, AVG(nota_teste)
->      FROM estudante
->      GROUP BY nome_estudante;
```

MIN(expr)

MAX(expr)

Retorna o valor mínimo ou o máximo de expr. MIN() e MAX() podem usar uma string como argumento; nestes casos eles retornam a string de valor mínimo ou máximo. See [\[Índices do MySQL\], page \[Índices do MySQL\]](#).

```
mysql> SELECT nome_estudante, MIN(nota_teste), MAX(nota_teste)
->      FROM estudante
->      GROUP BY nome_estudante;
```

Em MIN(), MAX() e outras funções de agrupamento o MySQL, atualmente, compara colunas ENUM e SET pelo seu valor string em vez de fazê-lo pela sua posição relativa de string no conjunto. Isto será retificado.

SUM(expr)

Retorna a soma de expr. Note que se o conjunto de retorno não possuir registros ele retornará NULL!

GROUP_CONCAT(expr)

Sintaxe completa:

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
              [ORDER BY {inteiro_sem_sinal | nome_coluna | formula} [ASC |
              [SEPARATOR valor_str]])
```

Esta função foi adicionada na versão 4.1 do MySQL. Ele retorna a string resultante contendo valores de um grupo:

```
mysql> SELECT nome_estudante,
->         GROUP_CONCAT(note_teste)
->         FROM estudante
->         GROUP BY nome_estudante;

ou

mysql> SELECT nome_estudante,
->         GROUP_CONCAT(DISTINCT nota_teste
->                        ORDER BY nota_teste DESC SEPARATOR " ")
->         FROM estudante
->         GROUP BY nome_estudante;
```

No MySQL você pode obter valores de combinações de expressões concatenados. Você pode eliminar valores duplicados utilizando `DISTINCT`. Se você quiser ordenar valores no resultado você deve utilizar a cláusula `ORDER BY`. Para ordenar inversamente, adicione a palavra chave `DESC` (descendente) ao nome da coluna que você está ordenando na cláusula `ORDER BY`. O padrão é a ordem crescente; pode-se também especificá-la explicitamente usando a palavra chave `ASC`. `SEPARATOR` é o valor string que deve ser inserido entre os valores no resultado. O padrão é um virgula (',''). Você pode remover o separador especificando `SEPARATOR ""`.

Você pode definir um tamanho máximo permitido com a variável `group_concat_max_len` em sua configuração. A sintaxe para se fazer isto em tempo de execução é:

```
SET [SESSION | GLOBAL] group_concat_max_len = unsigned_integer;
```

Se um tamanho máximo tiver sido atribuído, o resultado é truncado no seu tamanho máximo.

A função `GROUP_CONCAT()` é uma implementação aprimorada da função básica `LIST()` suportada pelo Sybase SQL Anywhere. `GROUP_CONCAT()` é compatível com a funcionalidade extremamente limitada de `LIST()`, se utilizada em apenas uma coluna e nenhuma outra opção é especificada. `LIST()` não tem uma ordem de classificação padrão.

VARIANCE(expr)

Retorna a variância padrão de `expr`. Esta é uma extensão do SQL-99 (disponível somente a partir da versão 4.1).

STD(expr)

STDDEV(expr)

Retorna o desvio padrão de `expr`. Esta é uma extensão do SQL-99. O formato `STDDEV()` desta função é fornecida para compatibilidade com Oracle.

BIT_OR(expr)

Retorna o resultado da operação binária `OR` de todos os bits em `expr`. O cálculo é realizado com precisão de 64 bits (`BIGINT`).

A função retorna 0 se não houver registros coincidentes.

BIT_AND(expr)

Retorna o resultado da operação binária `AND` de todos os bits em `expr`. O cálculo é realizado com precisão de 64 bits (`BIGINT`).

A função retorna 1 se não houver registros coincidentes.

O MySQL tem estendido o uso de GROUP BY. Você pode utilizar colunas ou cálculos na expressão SELECT que não aparecem na parte GROUP BY. Ele espera por *qualquer valor possível para este grupo*. Você pode utilizar isto para conseguir um melhor desempenho evitando ordenação e agrupamento em itens desnecessários. Por exemplo, você não precisa fazer um agrupamento em `cliente.nome` na consulta seguinte:

```
mysql> SELECT pedido.idcliente,cliente.nome,MAX(pagamento)
->      FROM pedido, cliente
->      WHERE pedido.idcliente = cliente.idcliente
->      GROUP BY pedido.idcliente;
```

No padrão SQL, você teria que adicionar `cliente.nome` a cláusula GROUP BY. No MySQL, o nome é redundante se você não o executa em modo ANSI.

Não utilize este recurso se as colunas omitidas na parte GROUP BY não são únicas no grupo! Você obterá resultados inesperados.

Em alguns casos, você pode utilizar MIN e MAX para obter o valor de uma coluna específica, mesmo que ele não seja único. O exemplo seguinte fornece o valor de coluna do registro contendo o menor valor na coluna ordem:

```
SUBSTR(MIN(CONCAT(RPAD(ordem,6,' '),coluna)),7)
```

See [Maximum-column-group-row-snt](#) [example-Maximum-column-group-row], page [Maximum-column-group-row-pg](#).

NOTE que se você estiver usando a versão 3.22 do MySQL (ou anterior) ou se estiver tentando seguir o SQL-99, você não pode utilizar expressões nas cláusulas GROUP BY or ORDER BY. Você pode contornar esta limitação utilizando um alias para a expressão:

```
mysql> SELECT id,FLOOR(value/100) AS val FROM nome_tabela
->      GROUP BY id,val ORDER BY val;
```

Na versão 3.23 do MySQL você pode fazer:

```
mysql> SELECT id,FLOOR(value/100) FROM nome_tabela ORDER BY RAND();
```

6.4 Manipulação de Dados: SELECT, INSERT, UPDATE, DELETE

6.4.1 Sintaxe SELECT

```
SELECT [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS] [HIGH_PRIORITY]
      [DISTINCT | DISTINCTROW | ALL]
expressão_select,...
[INTO {OUTFILE | DUMPFILE} 'nome_arquivo' opções_exportação]
[FROM tabelas_ref
  [WHERE definição_where]
  [GROUP BY {inteiro_sem_sinal | nome_coluna | formula} [ASC | DESC], ...]
  [HAVING where_definition]
  [ORDER BY {inteiro_sem_sinal | nome_coluna | formula} [ASC | DESC], ...]
  [LIMIT [offset,] linhas | linhas OFFSET offset]
  [PROCEDURE nome_procedimento(lista_argumentos)]
  [FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT é utilizado para retornar registros selecionados de uma ou mais tabelas. `expressão_select` indica as colunas que você deseja recuperar. **SELECT** também pode ser utilizado para retornar registros calculados sem referência a nenhuma tabela. Por exemplo:

```
mysql> SELECT 1 + 1;
-> 2
```

Todas as palavras chaves devem ser fornecidas exatamente na ordem mostrada acima. Por exemplo, uma cláusula **HAVING** deve vir depois de qualquer cláusula **GROUP BY** e antes de qualquer cláusula **ORDER BY**.

- Uma expressão **SELECT** pode utilizar um alias usando **AS**. O alias é usado como o nome da coluna da expressão e pode ser usado com cláusulas **ORDER BY** ou **HAVING**. Por exemplo:

```
mysql> SELECT CONCAT(primeiro_nome,' ',ultimo_nome) AS nome_completo
FROM minha_tabela ORDER BY nome_completo;
```

- Não é permitido utilizar um alias de coluna em uma cláusula **WHERE**, pois o valor da coluna pode ainda não ter sido determinado quando a cláusula **WHERE** for executada. See [\(undefined\)](#) [Problemas com alias], page [\(undefined\)](#).
- A cláusula **FROM table_references** indica a tabela de onde os registros serão retornados. Se você indicar mais de uma tabela, você estará realizando uma join. Para informações sobre a sintaxe de join, veja [\(undefined\)](#) [JOIN], page [\(undefined\)](#). Para cada tabela especificada, você pode, opcionalmente, especificar um alias.

```
nome_tabela [[AS] alias] [[USE INDEX (lista_indice)] | [IGNORE INDEX (lista_
indice)] | FORCE INDEX (lista_indice)]]
```

Como na versão 3.23.12 do MySQL, você pode dar sugestões sobre qual índice o MySQL deve usar ao recuperar informações de uma tabela. Isto é útil se **EXPLAIN** mostrar que o MySQL esta utilizando o índice errado da lista de índices possíveis. Especificando **USE INDEX (lista_indice)** você pode dizer ao MySQL para usar somente um dos índices possíveis para encontrar registros em uma tabela. A sintaxe alternativa **IGNORE INDEX (lista_indice)** pode ser usada para dizer ao MySQL para não utilizar alguns índices particulares.

Na versão 4.0.9 do MySQL você também pode usar **FORCE INDEX**. Ele funciona como **USE INDEX (lista_indice)** mas ele assume que uma varredura em uma tabelas é MUITO cara. Em outras palavras, uma varredura só será usada se não houver nenhum modo de utilizar um dos índices dados para encontrar registros nas tabelas.

USE/IGNORE/FORCE KEY é sinônimo de **USE/IGNORE/FORCE INDEX**.

- Você pode se referir a uma tabela como `nome_tabela` (dentro do banco de dados atual) ou como `nomebd.nome_tabela` para especificar um banco de dados. Você pode se referir a um coluna como `nome_coluna`, `nome_tabela.nome_coluna` ou `nomebd.nome_tabela.nome_coluna`. Você não precisa especificar um prefixo `nome_tabla` ou `nomebd.nome_tabela` para referência a uma coluna em uma instrução **SELECT** a menos a referência seja ambígua. Veja [\(undefined\)](#) [Legal names], page [\(undefined\)](#), para exemplos de ambiguidade que exigem a forma mais explicita de referência a coluna.
- Pode se definir um alias fazendo referência a uma tabela utilizando `nome_tabela [AS] nome_alias`:

```
mysql> SELECT t1.nome, t2.salario FROM funcionarios AS t1, info AS t2
->      WHERE t1.nome = t2.nome;
mysql> SELECT t1.nome, t2.salario FROM funcionarios t1, info t2
->      WHERE t1.nome = t2.nome;
```

- Colunas selecionadas para saída podem ser referidas em cláusulas `ORDER BY` e `GROUP BY` usando nomes de colunas, alias de colunas ou posições de colunas. As posições de colunas começam com 1:

```
mysql> SELECT college, region, seed FROM tournament
->      ORDER BY region, seed;
mysql> SELECT college, region AS r, seed AS s FROM tournament
->      ORDER BY r, s;
mysql> SELECT college, region, seed FROM tournament
->      ORDER BY 2, 3;
```

Para ordenar inversamente, adicione a palavra-chave `DESC` (descendente) ao nome da coluna na cláusula `ORDER BY` na qual você está ordenando. A ordem padrão é ascendente; ela pode ser especificada explicitamente usando a palavra-chave `ASC`.

- Você pode usar qualquer uma das funções suportadas pelo MySQL na cláusula `WHERE`. See [\(undefined\) \[Funções\], page \(undefined\)](#).
- A cláusula `HAVING` pode se referir a qualquer coluna ou alias definido na expressão `select`. Ele é aplicado por último, pouco antes dos itens serem enviados ao cliente, sem otimização. Não utilize `HAVING` para itens que devem estar na cláusula `WHERE`. Por exemplo, não escreva isto:

```
mysql> SELECT nome_col FROM nome_tabela HAVING nome_col > 0;
```

Escreva assim:

```
mysql> SELECT nome_col FROM nome_tabela WHERE nome_col > 0;
```

Na versão 3.22.5 ou posterior, você também pode escrever consultar desta forma:

```
mysql> SELECT usuario,MAX(salario) FROM usuarios
->      GROUP BY usuario HAVING MAX(salario)>10;
```

Em versões mais antigas, você pode escrever desta forma:

```
mysql> SELECT usuario,MAX(salario) AS soma FROM usuarios
->      group by usuario HAVING soma>10;
```

- As opções `DISTINCT`, `DISTINCTROW` e `ALL` especificam quando registros duplicados devem ser retornados. O padrão é (`ALL`), todos os registros coincidentes são retornados. `DISTINCT` e `DISTINCTROW` são sinônimos e especificam que registros duplicados no conjunto de resultados devem ser removidos.
- Todas as opções iniciando com `SQL_`, `STRAIGHT_JOIN`, e `HIGH_PRIORITY` são extensões do MySQL para SQL-99.
- `HIGH_PRIORITY` dará uma prioridade maior ao `SELECT` do que para uma instrução que atualizam uma tabela. Você só deve isto para consultas que sejam rápidas e devam ser feitas imediatamente. Uma consulta `SELECT HIGH_PRIORITY` retornará se a tabela está bloqueada para leitura mesmo se houver uma instrução de atualização que estiver esperando a liberação da tabela.
- `SQL_BIG_RESULT` pode ser usado com `GROUP BY` ou `DISTINCT` para dizer ao otimizador que o conjunto de resultados terá muitas linhas. Neste caso, o MySQL usará diretamente tabelas temporárias em disco se necessário. O MySQL também irá, neste caso,

preferir ordenar fazendo uma tabela temporária com um chave nos elementos **GROUP BY**.

- **SQL_BUFFER_RESULT** forçará que o resultado seja colocado em uma tabela temporária. Isto ajudará o MySQL a liberar as travas de tabelas mais cedo e ajudará nos casos onde ele levá muito tempo para enviar o conjunto de resultado ao cliente.
- **SQL_SMALL_RESULT**, uma opção específica do MySQL, pode ser usada com **GROUP BY** ou **DISTINCT** para dizer ao otimizador que o conjunto de resultados será pequeno. Neste caso, o MySQL usará tabelas temporárias rápidas para armazenar a tabela resultante em vez de usar ordenação. Na versão 3.23 do MySQL isto não é necessário normalmente.
- **SQL_CALC_FOUND_ROWS** (versão 4.0.0 e acima) diz ao MySQL para calcular quantas linhas haveriam no conjunto de resultados, desconsiderando qualquer cláusula **LIMIT**. O número de linhas pode ser recuperado com **SELECT FOUND_ROWS()**. See [\[Funções diversas\]](#), page [\[undefined\]](#).

Por favor, note que em versões anteriores a 4.1.0 isto não funciona com **LIMIT 0**, o qual é otimizado para retornar instantaneamente (resultando em 0 registros). See [\[Otimização LIMIT\]](#), page [\[undefined\]](#).

- **SQL_CACHE** diz ao MySQL para armazenar o resultado da consulta em um cache de consultas se você estiver utilizando **QUERY_CACHE_TYPE=2 (DEMAND)**. See [\[Cache de consultas\]](#), page [\[undefined\]](#).
- **SQL_NO_CACHE** diz ao MySQL para não permitir que o resultado da consulta seja armazenado nesta cache de consultas. See [\[Cache de consultas\]](#), page [\[undefined\]](#).
- Se você utiliza **GROUP BY**, os registros de saída serão ordenados de acordo com o **GROUP BY** como se você tivesse tido um **ORDER BY** sobre todos os campos no **GROUP BY**. O MySQL tem estendido o **GROUP BY** para que você também possa especificar **ASC** e **DESC** ao **GROUP BY**:

```
SELECT a,COUNT(b) FROM tabela_teste GROUP BY a DESC
```

- O MySQL tem estendido o uso do **GROUP BY** para lhe permitir selecionar campos que não estão mencionados na cláusula **GROUP BY**. Se você não está conseguindo os resultados esperados para a sua consulta, leia a descrição de **GROUP BY**. See [\[Funções Group By\]](#), page [\[undefined\]](#).
- **STRAIGHT_JOIN** força o otimizador a unir as tabelas em ordem em que elas são listadas na cláusula **FROM**. Você pode usá-lo para aumentar a velocidade de uma consulta e o otimizador de unir as tabelas em uma ordem não otimizada. See [\[EXPLAIN\]](#), page [\[undefined\]](#).
- A cláusula **LIMIT** pode ser usada para restringir o número de linhas retornadas pela instrução **SELECT**. **LIMIT** utiliza um ou dois argumentos numéricos. Os argumentos devem ser constantes inteiras.

Se forem fornecidos dois argumentos, o primeiro especifica a posição do primeiro registro a ser retornado e o segundo especifica o número máximo de linhas a retornar. A posição do registro inicial é 0 (não 1):

Para ser compatível com o PostgreSQL, o MySQL suporta a sintaxe: **LIMIT # OFFSET #**.

```
mysql> SELECT * FROM tabela LIMIT 5,10; # Recupera linhas 6-15
```

To retrieve all rows from a certain offset up to the end of the result set, you can use -1 for the second parameter:

```
mysql> SELECT * FROM tabela LIMIT 95,-1; # Recupera linhas 96-ultima.
```

Se um dos argumentos é dado, ele indica o número máximo de linhas a retornar:

```
mysql> SELECT * FROM tabela LIMIT 5;      # Recupera as primeiras 5 linhas
```

Em outras palavras, LIMIT n é equivalente a LIMIT 0,n.

- iA forma SELECT ... INTO OUTFILE 'nome_arquivo' do SELECT grava os registros selecionados em um arquivo. O arquivo é criado na máquina servidora e não pode já existir (entre outras coisas, isto previne tabelas de banco de dados e arquivos tais como '/etc/passwd' de serem destruídos). Você deve ter o privilégio FILE na máquina servidora para utilizar esta forma de SELECT.

SELECT ... INTO OUTFILE tem como intenção principal deixar que você descarregue rapidamente um tabela de uma máquina servidora. Se você quiser criar o arquivo resultante em outra máquina, diferente do servidor, você não deve usar SELECT ... INTO OUTFILE. Neste caso você deve usar algum programa cliente como mysqldump --tab ou mysql -e "SELECT..." > outfile para gerar o arquivo.

SELECT ... INTO OUTFILE é o complemento de LOAD DATA INFILE; a sintaxe para a parte opções_exportação de uma instrução consiste das mesmas cláusulas CAMPOS e LINHAS que são usadas com a instrução LOAD DATA INFILE. See [\[LOAD DATA\]](#), page [\[undefined\]](#).

No arquivo texto resultante, somente os seguintes caracteres são escritos com o caracter ESCAPE BY:

- O caracter ESCAPE BY
- O primeiro caracter em FIELDS TERMINATED BY
- O primeiro caracter em LINES TERMINATED BY

Adicionalmente, ASCII 0 é convertido para ESCAPE BY seguido por 0 (ASCII 48).

A razão para o mostrado acima é que você deve escapar qualquer caracter FIELDS TERMINATED BY, ESCAPE BY, or LINES TERMINATED BY para termos a segurança que o arquivo poderá ser lido de volta. É feito escape de ASCII 0 para facilitar a visualização com alguns paginadores.

Como o arquivo resultante não tem que estar em conformidade com a sintaxe SQL, nada mais precisa ser seguido de caracteres de escape.

Aqui segue um exemplo de como se obter um arquivo no formato usado por muitos programas antigos.

```
SELECT a,b,a+b INTO OUTFILE "/tmp/result.text"
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY "\n"
FROM tabela_teste;
```

- Se você utilizar INTO DUMPFILE em vez de INTO OUTFILE, o MySQL só irá escrever um linha no arquivo, sem nenhum terminador de linha ou colunas e sem nenhum escape. Ele é útil se você quiser armazenar um blob em um arquivo.
- Note que qualquer arquivo criado por INTO OUTFILE e INTO DUMPFILE poderão ser escritos por todos os usuários! A razão é que o servidor MySQL não pode criar um

arquivo que pertence a qualquer um além do usuário que o está executando (você nunca deve executar `mysqld` como `root`) o arquivo tem que ser gravável para todos para que você possa manipulá-lo.

- Se você estiver utilizando `FOR UPDATE` em um mecanismo de armazenamento com travamento de páginas/registros, as linhas examinadas serão travadas para escrita.

6.4.1.1 Sintaxe JOIN

O MySQL suporta as seguintes sintaxes JOIN para uso em instruções `SELECT`:

```
tabela_ref, tabela_ref
tabela_ref [CROSS] JOIN tabela_ref
tabela_ref INNER JOIN tabela_ref condição_join
tabela_ref STRAIGHT_JOIN tabela_ref
tabela_ref LEFT [OUTER] JOIN tabela_ref condição_join
tabela_ref LEFT [OUTER] JOIN tabela_ref
tabela_ref NATURAL [LEFT [OUTER]] JOIN tabela_ref
{ OJ tabela_ref LEFT OUTER JOIN tabela_ref ON expr_condicional }
tabela_ref RIGHT [OUTER] JOIN tabela_ref condição_join
tabela_ref RIGHT [OUTER] JOIN tabela_ref
tabela_ref NATURAL [RIGHT [OUTER]] JOIN tabela_ref
```

Onde `tabela_ref` é definido como:

```
nome_tabela [[AS] alias] [[USE INDEX (lista_indice)] | [IGNORE INDEX (lista_
indice)] | [FORCE INDEX (lista_indice)]]
```

a `condição_join` é definido como:

```
ON expr_condicional |
USING (lista_colunas)
```

Geralmente você não deverá ter nenhuma condição na parte `ON` que é usada para restringir quais registros você terá no seu resultado (existem exceções a esta regra). Se quiser restringir quais registros devem estar no resultado, você terá que fazê-lo na cláusula `WHERE`.

Note que em versões anteriores a 3.23.17, o `INNER JOIN` não utilizava uma `condição_join`!

A última sintaxe `LEFT OUTER JOIN` mostrada acima só existe para compatibilidade com ODBC:

- Pode se usar um alias para referência a tabelas com `nome_tabela AS nome_alias` ou `nome_tabela nome_alias`:

```
mysql> SELECT t1.nome, t2.salario FROM funcionarios AS t1, info AS t2
->          WHERE t1.nome = t2.nome;
```

- A condicional `ON` é qualquer condição da forma que pode ser usada em uma cláusula `WHERE`.
- Se não houver registros coincidentes para a tabela a direita da parte `ON` ou `USING` em um `LEFT JOIN`, uma linha com `NULL` atribuído a todas as colunas é usada para a tabela a direita. Você pode usar este fato para encontrar registro em uma tabela que não houver contrapartes em outra tabela

```
mysql> SELECT tabela1.* FROM tabela1
->          LEFT JOIN tabela2 ON tabela1.id=tabela2.id
->          WHERE tabela2.id IS NULL;
```

Este exemplo encontra todas as linhas em `tabela1` com um valor `id` que não está presente em `tabela2` (isto é, toda as linhas em `tabela1` sem linha correspondente em `tabela2`). Assume-se que `tabela2.id` é declarada NOT NULL. See [\[Otimização LEFT JOIN\]](#), page [\[Otimização LEFT JOIN\]](#).

- A cláusula `USING (lista_colunas)` nomeia uma lista de colunas que devem existir em ambas as tabelas. Uma cláusula `USING` como:

```
A LEFT JOIN B USING (C1,C2,C3,...)
```

é definida para ser semanticamente idêntica a uma expressão `ON` como esta:

```
A.C1=B.C1 AND A.C2=B.C2 AND A.C3=B.C3,...
```

- Um `NATURAL [LEFT] JOIN` de duas tabelas é definido para ser semanticamente equivalente a um `INNER JOIN` ou um `LEFT JOIN` com uma cláusula `USING` que nomeia todas as colunas que existem em ambas as tabelas.
- `INNER JOIN` e `,` (vírgula) são semanticamente equivalentes. Ambos fazem um join completo entre as tabelas usadas. Normalmente você especifica como as tabelas devem ser ligadas em uma condição `WHERE`.
- `RIGHT JOIN` funciona de forma análoga a um `LEFT JOIN`. Para manter o código portátil entre banco de dados, é recomendado usar `LEFT JOIN` em vez de `RIGHT JOIN`.
- `STRAIGHT_JOIN` é idêntico a `JOIN`, exceto pelo fato de que a tabela de esquerda sempre é lida antes da tabela da direita. Ele pode ser usado para aqueles casos (poucos) onde o otimizador join coloca as tabelas na ordem errada.
- Como na versão 3.23.12, você pode dar sugestões sobre qual índice o MySQL deve usar quando retornar informações de uma tabela. Isto é útil se `EXPLAIN` mostrar que o MySQL está utilizando o índice errado da lista de índices possíveis. Especificando `USE INDEX (lista_indice)`, você pode dizer ao MySQL para usar somente um dos índices possíveis para encontrar registros em uma tabela. A sintaxe alternativa `IGNORE INDEX (lista_indice)` pode ser usado para dizer ao MySQL para não utilizar índices particulares.

Na versão 4.0.9 do MySQL você também pode utilizar `FORCE INDEX`. Ele funciona como `USE INDEX (key_list)` mas com assume que uma varredura na tabela é MUITO cara. Em outras palavras, uma varredura na tabela só será feita se não houver modo de utilizar um dos índices fornecidos para se encontrar registros no tabela.

`USE/IGNORE KEY` são sinônimos de `USE/IGNORE INDEX`.

Alguns exemplos:

```
mysql> SELECT * FROM tabela1,tabela2 WHERE tabela1.id=tabela2.id;
mysql> SELECT * FROM tabela1 LEFT JOIN tabela2 ON tabela1.id=tabela2.id;
mysql> SELECT * FROM tabela1 LEFT JOIN tabela2 USING (id);
mysql> SELECT * FROM tabela1 LEFT JOIN tabela2 ON tabela1.id=tabela2.id
->     LEFT JOIN tabela3 ON tabela2.id=tabela3.id;
mysql> SELECT * FROM tabela1 USE INDEX (chave1,chave2)
->     WHERE chave1=1 AND chave2=2 AND chave3=3;
mysql> SELECT * FROM tabela1 IGNORE INDEX (chave3)
->     WHERE chave1=1 AND chave2=2 AND chave3=3;
```

See [\[Otimização LEFT JOIN\]](#), page [\[Otimização LEFT JOIN\]](#).

6.4.1.2 Sintaxe UNION

```
SELECT ...
UNION [ALL]
SELECT ...
[UNION
SELECT ...]
```

UNION foi implementado no MySQL 4.0.0.

UNION é usado para combinar o resultado de muitas instruções **SELECT** em um único conjunto de resultados.

As colunas listadas na porção expressão_select de **SELECT** devem ter o mesmo tipo. Os nomes das colunas usadas na primeira consulta **SELECT** serão usadas como nomes de colunas para o resultado retornado.

Os comandos **SELECT** são comandos selects normais, mas com a seguinte restrição:

- Somente o último comando **SELECT** pode ter **INTO OUTFILE**.

Se você não utilizar a palavra-chave **ALL** para o **UNION**, todas as linhas retornadas serão únicas, como se você tivesse utilizado um **DISTINCT** para o resultado final. Se você especificar **ALL**, você obterá todos os registros encontrados em todas as instruções **SELECT**.

Se você quiser usar um **ORDER BY** para o resultado **UNION** final, você deve utilizar parênteses:

```
(SELECT a FROM nome_tabela WHERE a=10 AND B=1 ORDER BY a LIMIT 10)
UNION
(SELECT a FROM nome_tabela WHERE a=11 AND B=2 ORDER BY a LIMIT 10)
ORDER BY a;
```

6.4.2 Sintaxe HANDLER

```
HANDLER nome_tabela OPEN [ AS alias ]
HANDLER nome_tabela READ nome_indice { = | >= | <= | < } (valor1,valor2,...)
[ WHERE ... ] [LIMIT ... ]
HANDLER nome_tabela READ nome_indice { FIRST | NEXT | PREV | LAST }
[ WHERE ... ] [LIMIT ... ]
HANDLER nome_tabela READ { FIRST | NEXT }
[ WHERE ... ] [LIMIT ... ]
HANDLER nome_tabela CLOSE
```

A instrução **HANDLER** fornece acesso direto a interface do mecanismo de armazenamento de tabelas MyISAM.

A primeira forma da instrução **HANDLER** abre uma tabela, tornando a acessível através de subsequentes instruções **HANDLER ... READ**. Este objeto de tabela não é compartilhada com outras threads e não serão fechadas até que as chamadas de thread **HANDLER nome_tabela CLOSE** ou a thread termine.

A segunda forma busca um registro (ou mais, especificado pela cláusula **LIMIT**) onde o índice especificado é utilizado na condição e a condição **WHERE** é encontrada. Se o índice consiste de várias partes (se divide em várias colunas) os valores são especificados em uma lista separadas por vírgulas, fornecendo valores somente para algumas poucas primeiras colunas possíveis.

A terceira forma busca uma linha (ou mais, especificado pela cláusula LIMIT) da tabela na ordem do índice, correspondendo a condição WHERE.

A quarta forma (sem especificação de índice) busca um registro (ou mais, especificado pela cláusula LIMIT) da tabela na ordem natural da linhas (como armazenado no arquivo de dados) de acordo com a condição WHERE é mais rápido que HANDLER nome_tabela READ nome_indice quando é necessária uma varredura completa da tabela.

HANDLER ... CLOSE fecha uma tabela que foi aberta com HANDLER ... OPEN.

Nota: Se você estiver utilizando a interface HANDLER para PRIMARY KEY você deve se lembrar de colocar o nome entre aspas: HANDLER tbl READ 'PRIMARY' > (...)

HANDLER é uma instrução de baixo nível. Por exemplo, ela não fornece consistência. Isto é, HANDLER ... OPEN **NÃO** pega uma imagem instântanea da tabela, e **NÃO** trava a tabela. Isto significa que depois que um HANDLER ... OPEN é feito, os dados da tabela podem ser modificados (por esta ou outra thread) e estas modificações podem aparecer apenas parcialmente nas buscas HANDLER ... NEXT ou HANDLER ... PREV.

As razões para se utilizar esta interface em vez do SQL normal são:

- Ela é mais rápida que SELECT porque:
 - Um mecanismo de armazenamento designado é alocado pela thread em HANDLER OPEN.
 - Existe menos análise envolvida.
 - Não existe sobrecarga de otimização e verificação de consultas.
 - A tabela utilizada não precisa estar travada em pedidos de dois handlers.
 - A interface handler não precisa fornecer uma aprência consistente dos dados (por exemplo, dirty-reads são permitidas), assim o mecanismo de armazenamento pode fazer otimizações que o SQL normalmente não permite.
- É muito mais fácil portar aplicações que usam interface como ISAM para o MySQL.
- Ele permite se fazer uma travessia em um banco de dados de uma maneira que não é facil (em alguns casos impossível) de fazer com SQL. A interface handler é um modo mais natural de mostrar dados ao trabalhar com aplicações que fornecem uma interface interativa com o usuário para o banco de dados.

6.4.3 Sintaxe INSERT

```

INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
      [INTO] nome_tabela [(nome_coluna,...)]
      VALUES ((expressão | DEFAULT),...),(...),...
      [ ON DUPLICATE KEY UPDATE nome_coluna=expressão, ... ]
or INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
      [INTO] nome_tabela [(nome_coluna,...)]
      SELECT ...
or INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
      [INTO] nome_tabela
      SET nome_coluna=(expressão | DEFAULT), ...
      [ ON DUPLICATE KEY UPDATE nome_coluna=expressão, ... ]

```

`INSERT` insere novos registros em uma tabela existente. A forma `INSERT ... VALUES` da instrução insere registros baseado em valores especificados explicitamente. A forma `INSERT ... SELECT` insere linhas selecionadas de outra(s) tabela(s). A forma `INSERT ... VALUES` com listas de múltiplos valores é suportado a partir da versão 3.22.5. A sintaxe `nome_coluna=expressão` é suportada a partir da versão 3.22.10 do MySQL.

`nome_tabela` é a tabela na qual as linhas serão inseridas. A lista de nome das colunas ou a cláusula `SET` indica para quais colunas a instrução especifica valor:

- Se você não especificar a lista de colunas para `INSERT ... VALUES` ou `INSERT ... SELECT`, deverão ser fornecidos valores para todas as colunas na lista `VALUES()` ou pelo `SELECT`. Se você não souber a ordem das colunas nas tabelas, use `DESCRIBE nome_tabela` para descobrir.
- Qualquer coluna que não tiver o valor fornecido explicitamente assumirá o seu valor padrão. Por exemplo, se você especificar uma lista de colunas que não definem todas as colunas na tabela, às colunas não definidas serão atribuídos o seu valor padrão. Atribuição de valor padrão é definido em [〈undefined〉 \[CREATE TABLE\], page 〈undefined〉](#). Você também pode utilizar a palavra-chave `DEFAULT` para atribuir o valor padrão a uma coluna (Novo na versão 4.0.3. do MySQL). Fica mais fácil de se escrever instruções `INSERT` que atribuem valor a apenas algumas colunas porque ele permite que você evite escrever uma lista `VALUES()` incompleta (uma lista que não inclu um valor para cada coluna da tabela). De outra forma, você teria que escrever a lista de nomes de colunas correspondentes a cada valor na lista `VALUES()`.

MySQL sempre tem um valor padrão para todos os campos. Isto é algo imposto pelo MySQL para estar apto a funcionar com tabelas transacionais e não transacionais.

Nossa visão é que a verificação do conteúdo dos campos deve ser feita pela application and not in the database server.

- Uma `expressão` pode se referir a qualquer coluna que tenha sido definida anteriormente na lista de valores. Por exemplo, você pode dizer:

```
mysql> INSERT INTO nome_tabela (col1,col2) VALUES(15,col1*2);
```

Mas não:

```
mysql> INSERT INTO nome_tabela (col1,col2) VALUES(col2*2,15);
```

- Se você especificar a palavra-chave `LOW_PRIORITY`, a execução do `INSERT` é atrasada até que nenhum outro cliente esteja lendo a tabela. Neste caso o cliente tem que esperar até que a instrução `insert` esteja completa, a qual pode levar bastante tempo se a tabela estiver em uso constante. É diferente de `INSERT DELAYED`, que deixa o cliente continuar de uma vez. See [〈undefined〉 \[INSERT DELAYED\], page 〈undefined〉](#). Note que `LOW_PRIORITY` não deve normalmente ser usado com tabelas `MyISAM` já que elas desabilitam inserções concorrentes. See [〈undefined〉 \[MyISAM\], page 〈undefined〉](#).
- If you specify the keyword `IGNORE` in an `INSERT` with many value rows, any rows that duplicate an existing `PRIMARY` or `UNIQUE` key in the table are ignored and are not inserted. If you do not specify `IGNORE`, the insert is aborted if there is any row that duplicates an existing key value. You can determine with the C API function `mysql_info()` how many rows were inserted into the table.
- If you specify `ON DUPLICATE KEY UPDATE` clause (new in MySQL 4.1.0), and a row is inserted that would cause a duplicate value in a `PRIMARY` or `UNIQUE` key, an `UPDATE` of the old row is performed. For example, the command:

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3)
--> ON DUPLICATE KEY UPDATE c=c+1;
```

in case of column `a` is declared as `UNIQUE` and already holds 1 once, would be identical to the

```
mysql> UPDATE table SET c=c+1 WHERE a=1;
```

Note: that if column `b` is unique too, the `UPDATE` command would be written as

```
mysql> UPDATE table SET c=c+1 WHERE a=1 OR b=2 LIMIT 1;
```

and if `a=1 OR b=2` matches several rows, only **one** row will be updated! In general, one should try to avoid using `ON DUPLICATE KEY` clause on tables with multiple `UNIQUE` keys.

Since MySQL 4.1.1 one can use function `VALUES(nome_coluna)` to refer to the column value in the `INSERT` part of the `INSERT ... UPDATE` command - that is the value that would be inserted if there would be no duplicate key conflict. This function especially useful in multiple-row inserts. Naturally `VALUES()` function is only meaningful in `INSERT ... UPDATE` command and returns `NULL` otherwise.

Example:

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
--> ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

The command above is identical to

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3)
--> ON DUPLICATE KEY UPDATE c=3;
mysql> INSERT INTO table (a,b,c) VALUES (4,5,6)
--> ON DUPLICATE KEY UPDATE c=9;
```

When one uses `ON DUPLICATE KEY UPDATE`, the `DELAYED` option is ignored.

- If MySQL was configured using the `DONT_USE_DEFAULT_FIELDS` option, `INSERT` statements generate an error unless you explicitly specify values for all columns that require a non-`NULL` value. See [\[configure options\]](#), page [\[undefined\]](#).
- You can find the value used for an `AUTO_INCREMENT` column with the `mysql_insert_id` function. See [\[mysql_insert_id\(\)](#), page [\[undefined\]](#).

If you use `INSERT ... SELECT` or an `INSERT ... VALUES` statement with multiple value lists, you can use the C API function `mysql_info()` to get information about the query. The format of the information string is shown here:

```
Records: 100 Duplicates: 0 Warnings: 0
```

`Duplicates` indicates the number of rows that couldn't be inserted because they would duplicate some existing unique index value. `Warnings` indicates the number of attempts to insert column values that were problematic in some way. Warnings can occur under any of the following conditions:

- Inserting `NULL` into a column that has been declared `NOT NULL`. The column is set to its default value.
- Setting a numeric column to a value that lies outside the column's range. The value is clipped to the appropriate endpoint of the range.
- Setting a numeric column to a value such as `'10.34 a'`. The trailing garbage is stripped and the remaining numeric part is inserted. If the value doesn't make sense as a number at all, the column is set to 0.

- Inserting a string into a `CHAR`, `VARCHAR`, `TEXT`, or `BLOB` column that exceeds the column's maximum length. The value is truncated to the column's maximum length.
- Inserting a value into a date or time column that is illegal for the column type. The column is set to the appropriate zero value for the type.

6.4.3.1 INSERT ... SELECT Syntax

```
INSERT [LOW_PRIORITY] [IGNORE] [INTO] nome_tabela [(column list)] SELECT ...
```

With `INSERT ... SELECT` statement you can quickly insert many rows into a table from one or many tables.

```
INSERT INTO tblTemp2 (fldID) SELECT tblTemp1.fldOrder_ID FROM tblTemp1 WHERE
tblTemp1.fldOrder_ID > 100;
```

The following conditions hold for an `INSERT ... SELECT` statement:

- The target table of the `INSERT` statement cannot appear in the `FROM` clause of the `SELECT` part of the query because it's forbidden in standard SQL to `SELECT` from the same table into which you are inserting. (The problem is that the `SELECT` possibly would find records that were inserted earlier during the same run. When using subquery clauses, the situation could easily be very confusing!)
- `AUTO_INCREMENT` columns work as usual.
- You can use the C API function `mysql_info()` to get information about the query. See [\[INSERT\]](#), page [\[INSERT\]](#).
- To ensure that the binary log can be used to re-create the original tables, MySQL will not allow concurrent inserts during `INSERT ... SELECT`.

You can of course also use `REPLACE` instead of `INSERT` to overwrite old rows.

6.4.4 INSERT DELAYED Syntax

```
INSERT DELAYED ...
```

The `DELAYED` option for the `INSERT` statement is a MySQL-specific option that is very useful if you have clients that can't wait for the `INSERT` to complete. This is a common problem when you use MySQL for logging and you also periodically run `SELECT` and `UPDATE` statements that take a long time to complete. `DELAYED` was introduced in MySQL Version 3.22.15. It is a MySQL extension to SQL-92.

`INSERT DELAYED` only works with `ISAM` and `MyISAM` tables. Note that as `MyISAM` tables supports concurrent `SELECT` and `INSERT`, if there is no free blocks in the middle of the datafile, you very seldom need to use `INSERT DELAYED` with `MyISAM`. See [\[MyISAM\]](#), page [\[MyISAM\]](#).

When you use `INSERT DELAYED`, the client will get an `OK` at once and the row will be inserted when the table is not in use by any other thread.

Another major benefit of using `INSERT DELAYED` is that inserts from many clients are bundled together and written in one block. This is much faster than doing many separate inserts.

Note that currently the queued rows are only stored in memory until they are inserted into the table. This means that if you kill `mysqld` the hard way (`kill -9`) or if `mysqld` dies unexpectedly, any queued rows that weren't written to disk are lost!

The following describes in detail what happens when you use the `DELAYED` option to `INSERT` or `REPLACE`. In this description, the “thread” is the thread that received an `INSERT DELAYED` command and “handler” is the thread that handles all `INSERT DELAYED` statements for a particular table.

- When a thread executes a `DELAYED` statement for a table, a handler thread is created to process all `DELAYED` statements for the table, if no such handler already exists.
- The thread checks whether the handler has acquired a `DELAYED` lock already; if not, it tells the handler thread to do so. The `DELAYED` lock can be obtained even if other threads have a `READ` or `WRITE` lock on the table. However, the handler will wait for all `ALTER TABLE` locks or `FLUSH TABLES` to ensure that the table structure is up to date.
- The thread executes the `INSERT` statement, but instead of writing the row to the table, it puts a copy of the final row into a queue that is managed by the handler thread. Any syntax errors are noticed by the thread and reported to the client program.
- The client can't report the number of duplicates or the `AUTO_INCREMENT` value for the resulting row; it can't obtain them from the server, because the `INSERT` returns before the insert operation has been completed. If you use the C API, the `mysql_info()` function doesn't return anything meaningful, for the same reason.
- The binary log is updated by the handler thread when the row is inserted into the table. In case of multiple-row inserts, the binary log is updated when the first row is inserted.
- After every `delayed_insert_limit` rows are written, the handler checks whether any `SELECT` statements are still pending. If so, it allows these to execute before continuing.
- When the handler has no more rows in its queue, the table is unlocked. If no new `INSERT DELAYED` commands are received within `delayed_insert_timeout` seconds, the handler terminates.
- If more than `delayed_queue_size` rows are pending already in a specific handler queue, the thread requesting `INSERT DELAYED` waits until there is room in the queue. This is done to ensure that the `mysqld` server doesn't use all memory for the delayed memory queue.
- The handler thread will show up in the MySQL process list with `delayed_insert` in the `Command` column. It will be killed if you execute a `FLUSH TABLES` command or kill it with `KILL thread_id`. However, it will first store all queued rows into the table before exiting. During this time it will not accept any new `INSERT` commands from another thread. If you execute an `INSERT DELAYED` command after this, a new handler thread will be created.

Note that the above means that `INSERT DELAYED` commands have higher priority than normal `INSERT` commands if there is an `INSERT DELAYED` handler already running! Other update commands will have to wait until the `INSERT DELAYED` queue is empty, someone kills the handler thread (with `KILL thread_id`), or someone executes `FLUSH TABLES`.

- The following status variables provide information about `INSERT DELAYED` commands:

Variable	Meaning
<code>Delayed_insert_threads</code>	Number of handler threads
<code>Delayed_writes</code>	Number of rows written with <code>INSERT DELAYED</code>

`Not_flushed_delayed_` Number of rows waiting to be written
rows

You can view these variables by issuing a `SHOW STATUS` statement or by executing a `mysqladmin extended-status` command.

Note that `INSERT DELAYED` is slower than a normal `INSERT` if the table is not in use. There is also the additional overhead for the server to handle a separate thread for each table on which you use `INSERT DELAYED`. This means that you should only use `INSERT DELAYED` when you are really sure you need it!

6.4.5 UPDATE Syntax

```
UPDATE [LOW_PRIORITY] [IGNORE] nome_tabela
      SET nome_coluna1=expr1 [, nome_coluna2=expr2 ...]
      [WHERE where_definition]
      [ORDER BY ...]
      [LIMIT rows]
```

or

```
UPDATE [LOW_PRIORITY] [IGNORE] nome_tabela [, nome_tabela ...]
      SET nome_coluna1=expr1 [, nome_coluna2=expr2 ...]
      [WHERE where_definition]
```

`UPDATE` updates columns in existing table rows with new values. The `SET` clause indicates which columns to modify and the values they should be given. The `WHERE` clause, if given, specifies which rows should be updated. Otherwise, all rows are updated. If the `ORDER BY` clause is specified, the rows will be updated in the order that is specified.

If you specify the keyword `LOW_PRIORITY`, execution of the `UPDATE` is delayed until no other clients are reading from the table.

If you specify the keyword `IGNORE`, the update statement will not abort even if we get duplicate key errors during the update. Rows that would cause conflicts will not be updated.

If you access a column from `nome_tabela` in an expression, `UPDATE` uses the current value of the column. For example, the following statement sets the `age` column to one more than its current value:

```
mysql> UPDATE persondata SET age=age+1;
```

`UPDATE` assignments are evaluated from left to right. For example, the following statement doubles the `age` column, then increments it:

```
mysql> UPDATE persondata SET age=age*2, age=age+1;
```

If you set a column to the value it currently has, MySQL notices this and doesn't update it.

`UPDATE` returns the number of rows that were actually changed. In MySQL Version 3.22 or later, the C API function `mysql_info()` returns the number of rows that were matched and updated and the number of warnings that occurred during the `UPDATE`.

Starting from MySQL version 3.23, you can use `LIMIT #` to ensure that only a given number of rows are changed. MySQL will stop the update as soon as it has found `LIMIT` rows that satisfies the `WHERE` clause, independent of the rows changed content or not.

If an `ORDER BY` clause is used (available from MySQL 4.0.0), the rows will be updated in that order. This is really only useful in conjunction with `LIMIT`.

Starting with MySQL Version 4.0.4, you can also perform `UPDATE` operations that cover multiple tables:

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

Note: you can not use `ORDER BY` or `LIMIT` with multi-table `UPDATE`.

6.4.6 DELETE Syntax

```
DELETE [LOW_PRIORITY] [QUICK] FROM table_name
      [WHERE where_definition]
      [ORDER BY ...]
      [LIMIT rows]
```

or

```
DELETE [LOW_PRIORITY] [QUICK] table_name[*] [, table_name[*] ...]
      FROM table-references
      [WHERE where_definition]
```

or

```
DELETE [LOW_PRIORITY] [QUICK]
      FROM table_name[*] [, table_name[*] ...]
      USING table-references
      [WHERE where_definition]
```

`DELETE` deletes rows from `table_name` that satisfy the condition given by `where_definition`, and returns the number of records deleted.

If you issue a `DELETE` with no `WHERE` clause, all rows are deleted. If you do this in `AUTOCOMMIT` mode, this works as `TRUNCATE`. See [\[TRUNCATE\]](#), page [\[undefined\]](#). In MySQL 3.23, `DELETE` without a `WHERE` clause will return zero as the number of affected records.

If you really want to know how many records are deleted when you are deleting all rows, and are willing to suffer a speed penalty, you can use a `DELETE` statement of this form:

```
mysql> DELETE FROM table_name WHERE 1>0;
```

Note that this is much slower than `DELETE FROM table_name` with no `WHERE` clause, because it deletes rows one at a time.

If you specify the keyword `LOW_PRIORITY`, execution of the `DELETE` is delayed until no other clients are reading from the table.

If you specify the word `QUICK` then the storage engine will not merge index leaves during delete, which may speed up certain kind of deletes.

In `MyISAM` tables, deleted records are maintained in a linked list and subsequent `INSERT` operations reuse old record positions. To reclaim unused space and reduce file-sizes, use the `OPTIMIZE TABLE` statement or the `myisamchk` utility to reorganise tables. `OPTIMIZE TABLE` is easier, but `myisamchk` is faster. See [\[OPTIMIZE TABLE\]](#), page [\[undefined\]](#) and [\[Optimisation\]](#), page [\[undefined\]](#).

The first multi-table delete format is supported starting from MySQL 4.0.0. The second multi-table delete format is supported starting from MySQL 4.0.2.

The idea is that only matching rows from the tables listed **before** the **FROM** or before the **USING** clause are deleted. The effect is that you can delete rows from many tables at the same time and also have additional tables that are used for searching.

The `.*` after the table names is there just to be compatible with Access:

```
DELETE t1,t2 FROM t1,t2,t3 WHERE t1.id=t2.id AND t2.id=t3.id
```

or

```
DELETE FROM t1,t2 USING t1,t2,t3 WHERE t1.id=t2.id AND t2.id=t3.id
```

In the above case we delete matching rows just from tables `t1` and `t2`.

If an **ORDER BY** clause is used (available from MySQL 4.0.0), the rows will be deleted in that order. This is really only useful in conjunction with **LIMIT**. For example:

```
DELETE FROM somelog
WHERE user = 'jcole'
ORDER BY timestamp
LIMIT 1
```

This will delete the oldest entry (by `timestamp`) where the row matches the **WHERE** clause.

The MySQL-specific **LIMIT rows** option to **DELETE** tells the server the maximum number of rows to be deleted before control is returned to the client. This can be used to ensure that a specific **DELETE** command doesn't take too much time. You can simply repeat the **DELETE** command until the number of affected rows is less than the **LIMIT** value.

From MySQL 4.0, you can specify multiple tables in the **DELETE** statement to delete rows from one table depending on a particular condition in multiple tables. However, you can not use **ORDER BY** or **LIMIT** in a multi-table **DELETE**.

6.4.7 TRUNCATE Syntax

```
TRUNCATE TABLE table_name
```

In 3.23 **TRUNCATE TABLE** is mapped to **COMMIT ; DELETE FROM table_name**. See [\(undefined\)](#) [**DELETE**], page [\(undefined\)](#).

TRUNCATE TABLE differs from **DELETE FROM . . .** in the following ways:

- Truncate operations drop and re-create the table, which is much faster than deleting rows one by one.
- Not transaction-safe; you will get an error if you have an active transaction or an active table lock.
- Doesn't return the number of deleted rows.
- As long as the table definition file '`table_name.frm`' is valid, the table can be re-created this way, even if the data or index files have become corrupted.

TRUNCATE is an Oracle SQL extension.

6.4.8 REPLACE Syntax

```

REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] nome_tabela [(nome_coluna,...)]
    VALUES (expression,...),(...),...
or REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] nome_tabela [(nome_coluna,...)]
    SELECT ...
or REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] nome_tabela
    SET nome_coluna=expression, nome_coluna=expression,...

```

REPLACE works exactly like INSERT, except that if an old record in the table has the same value as a new record on a UNIQUE index or PRIMARY KEY, the old record is deleted before the new record is inserted. See [\[INSERT\]](#), page [\[INSERT\]](#).

In other words, you can't access the values of the old row from a REPLACE statement. In some old MySQL versions it appeared that you could do this, but that was a bug that has been corrected.

To be able to use REPLACE you must have INSERT and DELETE privileges for the table.

When you use a REPLACE command, `mysql_affected_rows()` will return 2 if the new row replaced an old row. This is because one row was inserted after the duplicate was deleted.

This fact makes it easy to determine whether REPLACE added or replaced a row: check whether the affected-rows value is 1 (added) or 2 (replaced).

Note that unless you use a UNIQUE index or PRIMARY KEY, using a REPLACE command makes no sense, since it would just do an INSERT.

6.4.9 LOAD DATA INFILE Syntax

```

LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name.txt'
    [REPLACE | IGNORE]
    INTO TABLE nome_tabela
    [FIELDS
        [TERMINATED BY '\t']
        [[OPTIONALLY] ENCLOSED BY ''']
        [ESCAPED BY '\\']
    ]
    [LINES TERMINATED BY '\n']
    [IGNORE number LINES]
    [(nome_coluna,...)]

```

The LOAD DATA INFILE statement reads rows from a text file into a table at a very high speed. If the LOCAL keyword is specified, the file is read from the client host. If LOCAL is not specified, the file must be located on the server. (LOCAL is available in MySQL Version 3.22.6 or later.)

For security reasons, when reading text files located on the server, the files must either reside in the database directory or be readable by all. Also, to use LOAD DATA INFILE on server files, you must have the FILE privilege on the server host. See [\[Privileges provided\]](#), page [\[Privileges provided\]](#).

In MySQL 3.23.49 and MySQL 4.0.2 `LOCAL` will only work if you have not started `mysqld` with `--local-infile=0` or if you have not enabled your client to support `LOCAL`. See [\[LOAD DATA LOCAL\]](#), page [\[LOAD DATA LOCAL\]](#).

If you specify the keyword `LOW_PRIORITY`, execution of the `LOAD DATA` statement is delayed until no other clients are reading from the table.

If you specify the keyword `CONCURRENT` with a `MyISAM` table, then other threads can retrieve data from the table while `LOAD DATA` is executing. Using this option will of course affect the performance of `LOAD DATA` a bit even if no other thread is using the table at the same time. Using `LOCAL` will be a bit slower than letting the server access the files directly, because the contents of the file must travel from the client host to the server host. On the other hand, you do not need the `FILE` privilege to load local files.

If you are using MySQL before Version 3.23.24 you can't read from a FIFO with `LOAD DATA INFILE`. If you need to read from a FIFO (for example the output from `gunzip`), use `LOAD DATA LOCAL INFILE` instead.

You can also load datafiles by using the `mysqlimport` utility; it operates by sending a `LOAD DATA INFILE` command to the server. The `--local` option causes `mysqlimport` to read datafiles from the client host. You can specify the `--compress` option to get better performance over slow networks if the client and server support the compressed protocol.

When locating files on the server host, the server uses the following rules:

- If an absolute pathname is given, the server uses the pathname as is.
- If a relative pathname with one or more leading components is given, the server searches for the file relative to the server's data directory.
- If a filename with no leading components is given, the server looks for the file in the database directory of the current database.

Note that these rules mean a file given as `./myfile.txt` is read from the server's data directory, whereas a file given as `myfile.txt` is read from the database directory of the current database. For example, the following `LOAD DATA` statement reads the file `data.txt` from the database directory for `db1` because `db1` is the current database, even though the statement explicitly loads the file into a table in the `db2` database:

```
mysql> USE db1;
mysql> LOAD DATA INFILE "data.txt" INTO TABLE db2.my_table;
```

The `REPLACE` and `IGNORE` keywords control handling of input records that duplicate existing records on unique key values. If you specify `REPLACE`, new rows replace existing rows that have the same unique key value. If you specify `IGNORE`, input rows that duplicate an existing row on a unique key value are skipped. If you don't specify either option, an error occurs when a duplicate key value is found, and the rest of the text file is ignored.

If you load data from a local file using the `LOCAL` keyword, the server has no way to stop transmission of the file in the middle of the operation, so the default behaviour is the same as if `IGNORE` is specified.

If you use `LOAD DATA INFILE` on an empty `MyISAM` table, all non-unique indexes are created in a separate batch (like in `REPAIR`). This normally makes `LOAD DATA INFILE` much faster when you have many indexes.

`LOAD DATA INFILE` is the complement of `SELECT ... INTO OUTFILE`. See [\[SELECT\]](#), page [\[SELECT\]](#). To write data from a database to a file, use `SELECT ... INTO`

OUTFILE. To read the file back into the database, use `LOAD DATA INFILE`. The syntax of the `FIELDS` and `LINES` clauses is the same for both commands. Both clauses are optional, but `FIELDS` must precede `LINES` if both are specified.

If you specify a `FIELDS` clause, each of its subclauses (`TERMINATED BY`, [OPTIONALLY] `ENCLOSED BY`, and `ESCAPED BY`) is also optional, except that you must specify at least one of them.

If you don't specify a `FIELDS` clause, the defaults are the same as if you had written this:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
```

If you don't specify a `LINES` clause, the default is the same as if you had written this:

```
LINES TERMINATED BY '\n'
```

In other words, the defaults cause `LOAD DATA INFILE` to act as follows when reading input:

- Look for line boundaries at newlines.
- Break lines into fields at tabs.
- Do not expect fields to be enclosed within any quoting characters.
- Interpret occurrences of tab, newline, or `'\'` preceded by `'\'` as literal characters that are part of field values.

Conversely, the defaults cause `SELECT ... INTO OUTFILE` to act as follows when writing output:

- Write tabs between fields.
- Do not enclose fields within any quoting characters.
- Use `'\'` to escape instances of tab, newline or `'\'` that occur within field values.
- Write newlines at the ends of lines.

Note that to write `FIELDS ESCAPED BY '\\'`, you must specify two backslashes for the value to be read as a single backslash.

The `IGNORE number LINES` option can be used to ignore a header of column names at the start of the file:

```
mysql> LOAD DATA INFILE "/tmp/file_name" INTO TABLE test IGNORE 1 LINES;
```

When you use `SELECT ... INTO OUTFILE` in tandem with `LOAD DATA INFILE` to write data from a database into a file and then read the file back into the database later, the field and line handling options for both commands must match. Otherwise, `LOAD DATA INFILE` will not interpret the contents of the file properly. Suppose you use `SELECT ... INTO OUTFILE` to write a file with fields delimited by commas:

```
mysql> SELECT * INTO OUTFILE 'data.txt'
->      FIELDS TERMINATED BY ','
->      FROM ...;
```

To read the comma-delimited file back in, the correct statement would be:

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE table2
->      FIELDS TERMINATED BY ',';
```

If instead you tried to read in the file with the statement shown here, it wouldn't work because it instructs `LOAD DATA INFILE` to look for tabs between fields:

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE table2
->          FIELDS TERMINATED BY '\t';
```

The likely result is that each input line would be interpreted as a single field.

LOAD DATA INFILE can be used to read files obtained from external sources, too. For example, a file in dBASE format will have fields separated by commas and enclosed in double quotes. If lines in the file are terminated by newlines, the command shown here illustrates the field and line handling options you would use to load the file:

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE nome_tabela
->          FIELDS TERMINATED BY ',' ENCLOSED BY '"'
->          LINES TERMINATED BY '\n';
```

Any of the field or line handling options may specify an empty string (''). If not empty, the FIELDS [OPTIONALLY] ENCLOSED BY and FIELDS ESCAPED BY values must be a single character. The FIELDS TERMINATED BY and LINES TERMINATED BY values may be more than one character. For example, to write lines that are terminated by carriage return-linefeed pairs, or to read a file containing such lines, specify a LINES TERMINATED BY '\r\n' clause. For example, to read a file of jokes, that are separated with a line of %, into a SQL table you can do:

```
CREATE TABLE jokes (a INT NOT NULL AUTO_INCREMENT PRIMARY KEY, joke TEXT NOT NULL);
LOAD DATA INFILE "/tmp/jokes.txt" INTO TABLE jokes FIELDS TERMINATED BY ""
LINES TERMINATED BY "\n%\n" (joke);
```

FIELDS [OPTIONALLY] ENCLOSED BY controls quoting of fields. For output (SELECT ... INTO OUTFILE), if you omit the word OPTIONALLY, all fields are enclosed by the ENCLOSED BY character. An example of such output (using a comma as the field delimiter) is shown here:

```
"1","a string","100.20"
"2","a string containing a , comma","102.20"
"3","a string containing a \" quote","102.20"
"4","a string containing a \", quote and comma","102.20"
```

If you specify OPTIONALLY, the ENCLOSED BY character is used only to enclose CHAR and VARCHAR fields:

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a \" quote",102.20
4,"a string containing a \", quote and comma",102.20
```

Note that occurrences of the ENCLOSED BY character within a field value are escaped by prefixing them with the ESCAPED BY character. Also note that if you specify an empty ESCAPED BY value, it is possible to generate output that cannot be read properly by LOAD DATA INFILE. For example, the output just shown above would appear as shown here if the escape character is empty. Observe that the second field in the fourth line contains a comma following the quote, which (erroneously) appears to terminate the field:

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a " quote",102.20
4,"a string containing a ", quote and comma",102.20
```

For input, the ENCLOSED BY character, if present, is stripped from the ends of field values. (This is true whether OPTIONALLY is specified; OPTIONALLY has no effect on input interpretation.) Occurrences of the ENCLOSED BY character preceded by the ESCAPED BY character are interpreted as part of the current field value. In addition, duplicated ENCLOSED BY characters occurring within fields are interpreted as single ENCLOSED BY characters if the field itself starts with that character. For example, if ENCLOSED BY ''' is specified, quotes are handled as shown here:

```
"The ""BIG"" boss"  -> The "BIG" boss
The "BIG" boss      -> The "BIG" boss
The ""BIG"" boss    -> The ""BIG"" boss
```

FIELDS ESCAPED BY controls how to write or read special characters. If the FIELDS ESCAPED BY character is not empty, it is used to prefix the following characters on output:

- The FIELDS ESCAPED BY character
- The FIELDS [OPTIONALLY] ENCLOSED BY character
- The first character of the FIELDS TERMINATED BY and LINES TERMINATED BY values
- ASCII 0 (what is actually written following the escape character is ASCII '0', not a zero-valued byte)

If the FIELDS ESCAPED BY character is empty, no characters are escaped. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

For input, if the FIELDS ESCAPED BY character is not empty, occurrences of that character are stripped and the following character is taken literally as part of a field value. The exceptions are an escaped '0' or 'N' (for example, \0 or \N if the escape character is '\'). These sequences are interpreted as ASCII 0 (a zero-valued byte) and NULL. See below for the rules on NULL handling.

For more information about '\'-escape syntax, see [\(undefined\) \[Literals\], page \(undefined\)](#).

In certain cases, field and line handling options interact:

- If LINES TERMINATED BY is an empty string and FIELDS TERMINATED BY is non-empty, lines are also terminated with FIELDS TERMINATED BY.
- If the FIELDS TERMINATED BY and FIELDS ENCLOSED BY values are both empty (''), a fixed-row (non-delimited) format is used. With fixed-row format, no delimiters are used between fields. Instead, column values are written and read using the “display” widths of the columns. For example, if a column is declared as INT(7), values for the column are written using 7-character fields. On input, values for the column are obtained by reading 7 characters. Fixed-row format also affects handling of NULL values; see below. Note that fixed-size format will not work if you are using a multi-byte character set.

Handling of NULL values varies, depending on the FIELDS and LINES options you use:

- For the default FIELDS and LINES values, NULL is written as \N for output and \N is read as NULL for input (assuming the ESCAPED BY character is '\').
- If FIELDS ENCLOSED BY is not empty, a field containing the literal word NULL as its value is read as a NULL value (this differs from the word NULL enclosed within FIELDS ENCLOSED BY characters, which is read as the string 'NULL').
- If FIELDS ESCAPED BY is empty, NULL is written as the word NULL.

- With fixed-row format (which happens when `FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` are both empty), `NULL` is written as an empty string. Note that this causes both `NULL` values and empty strings in the table to be indistinguishable when written to the file because they are both written as empty strings. If you need to be able to tell the two apart when reading the file back in, you should not use fixed-row format.

Some cases are not supported by `LOAD DATA INFILE`:

- Fixed-size rows (`FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` both empty) and `BLOB` or `TEXT` columns.
- If you specify one separator that is the same as or a prefix of another, `LOAD DATA INFILE` won't be able to interpret the input properly. For example, the following `FIELDS` clause would cause problems:

```
FIELDS TERMINATED BY ''' ENCLOSED BY '''
```

- If `FIELDS ESCAPED BY` is empty, a field value that contains an occurrence of `FIELDS ENCLOSED BY` or `LINES TERMINATED BY` followed by the `FIELDS TERMINATED BY` value will cause `LOAD DATA INFILE` to stop reading a field or line too early. This happens because `LOAD DATA INFILE` cannot properly determine where the field or line value ends.

The following example loads all columns of the `persondata` table:

```
mysql> LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

No field list is specified, so `LOAD DATA INFILE` expects input rows to contain a field for each table column. The default `FIELDS` and `LINES` values are used.

If you wish to load only some of a table's columns, specify a field list:

```
mysql> LOAD DATA INFILE 'persondata.txt'
-> INTO TABLE persondata (col1,col2,...);
```

You must also specify a field list if the order of the fields in the input file differs from the order of the columns in the table. Otherwise, MySQL cannot tell how to match up input fields with table columns.

If a row has too few fields, the columns for which no input field is present are set to default values. Default value assignment is described in [\(undefined\) \[CREATE TABLE\], page \(undefined\)](#).

An empty field value is interpreted differently than if the field value is missing:

- For string types, the column is set to the empty string.
- For numeric types, the column is set to 0.
- For date and time types, the column is set to the appropriate “zero” value for the type. See [\(undefined\) \[Date and time types\], page \(undefined\)](#).

Note that these are the same values that result if you assign an empty string explicitly to a string, numeric, or date or time type explicitly in an `INSERT` or `UPDATE` statement.

`TIMESTAMP` columns are only set to the current date and time if there is a `NULL` value for the column, or (for the first `TIMESTAMP` column only) if the `TIMESTAMP` column is left out from the field list when a field list is specified.

If an input row has too many fields, the extra fields are ignored and the number of warnings is incremented.

LOAD DATA INFILE regards all input as strings, so you can't use numeric values for ENUM or SET columns the way you can with INSERT statements. All ENUM and SET values must be specified as strings!

If you are using the C API, you can get information about the query by calling the API function `mysql_info()` when the LOAD DATA INFILE query finishes. The format of the information string is shown here:

```
Records: 1 Deleted: 0 Skipped: 0 Warnings: 0
```

Warnings occur under the same circumstances as when values are inserted via the INSERT statement (see [\[INSERT\]](#), page [\[undefined\]](#)), except that LOAD DATA INFILE also generates warnings when there are too few or too many fields in the input row. The warnings are not stored anywhere; the number of warnings can only be used as an indication if everything went well. If you get warnings and want to know exactly why you got them, one way to do this is to use `SELECT ... INTO OUTFILE` into another file and compare this to your original input file.

If you need LOAD DATA to read from a pipe, you can use the following trick:

```
mkfifo /mysql/db/x/x
chmod 666 /mysql/db/x/x
cat < /dev/tcp/10.1.1.12/4711 > /nt/mysql/db/x/x
mysql -e "LOAD DATA INFILE 'x' INTO TABLE x" x
```

If you are using a version of MySQL older than 3.23.25 you can only do the above with LOAD DATA LOCAL INFILE.

For more information about the efficiency of INSERT versus LOAD DATA INFILE and speeding up LOAD DATA INFILE, See [\[Insert speed\]](#), page [\[undefined\]](#).

6.4.10 DO Syntax

```
DO expression, [expression, ...]
```

Execute the expression but don't return any results. This is a shorthand of `SELECT expression, expression`, but has the advantage that it's slightly faster when you don't care about the result.

This is mainly useful with functions that has side effects, like `RELEASE_LOCK`.

6.5 Data Definition: CREATE, DROP, ALTER

6.5.1 CREATE DATABASE Syntax

```
CREATE DATABASE [IF NOT EXISTS] db_name
```

CREATE DATABASE creates a database with the given name. Rules for allowable database names are given in [\[Legal names\]](#), page [\[undefined\]](#). An error occurs if the database already exists and you didn't specify `IF NOT EXISTS`.

Databases in MySQL are implemented as directories containing files that correspond to tables in the database. Because there are no tables in a database when it is initially created, the CREATE DATABASE statement only creates a directory under the MySQL data directory.

You can also create databases with `mysqladmin`. See [Side Scripts-snt \[Client-Side Scripts\]](#), page [Side Scripts-pg](#).

6.5.2 DROP DATABASE Syntax

```
DROP DATABASE [IF EXISTS] db_name
```

DROP DATABASE drops all tables in the database and deletes the database. If you do a DROP DATABASE on a symbolic linked database, both the link and the original database is deleted.

Be VERY careful with this command!

DROP DATABASE returns the number of files that were removed from the database directory. Normally, this is three times the number of tables, because normally each table corresponds to a '.MYD' file, a '.MYI' file, and a '.frm' file.

The DROP DATABASE command removes from the given database directory all files with the following extensions:

Ext	Ext	Ext	Ext
.BAK	.DAT	.HSH	.ISD
.ISM	.ISM	.MRG	.MYD
.MYI	.db	.frm	

All subdirectories that consists of 2 digits (RAID directories) are also removed.

In MySQL Version 3.22 or later, you can use the keywords IF EXISTS to prevent an error from occurring if the database doesn't exist.

You can also drop databases with `mysqladmin`. See [Side Scripts-snt \[Client-Side Scripts\]](#), page [Side Scripts-pg](#).

6.5.3 CREATE TABLE Syntax

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nome_tabela [(create_definition,...)]
[table_options] [select_statement]
```

or

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nome_tabela LIKE old_table_name;
```

create_definition:

```
nome_coluna type [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT]
[PRIMARY KEY] [reference_definition]
or PRIMARY KEY (index_nome_coluna,...)
or KEY [index_name] (index_nome_coluna,...)
or INDEX [index_name] (index_nome_coluna,...)
or UNIQUE [INDEX] [index_name] (index_nome_coluna,...)
or FULLTEXT [INDEX] [index_name] (index_nome_coluna,...)
or [CONSTRAINT symbol] FOREIGN KEY [index_name] (index_nome_coluna,...)
[reference_definition]
or CHECK (expr)
```

type:

```
TINYINT[(length)] [UNSIGNED] [ZEROFILL]
or SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
or MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
or INT[(length)] [UNSIGNED] [ZEROFILL]
or INTEGER[(length)] [UNSIGNED] [ZEROFILL]
```

```

or      BIGINT[(length)] [UNSIGNED] [ZEROFILL]
or      REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
or      DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
or      FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
or      DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]
or      NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]
or      CHAR(length) [BINARY]
or      VARCHAR(length) [BINARY]
or      DATE
or      TIME
or      TIMESTAMP
or      DATETIME
or      TINYBLOB
or      BLOB
or      MEDIUMBLOB
or      LONGBLOB
or      TINYTEXT
or      TEXT
or      MEDIUMTEXT
or      LONGTEXT
or      ENUM(value1,value2,value3,...)
or      SET(value1,value2,value3,...)

```

```

index_nome_coluna:
    nome_coluna [(length)]

```

```

reference_definition:
    REFERENCES nome_tabela [(index_nome_coluna,...)]
        [MATCH FULL | MATCH PARTIAL]
        [ON DELETE reference_option]
        [ON UPDATE reference_option]

```

```

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

```

```

table_options:
TYPE = {BDB | HEAP | ISAM | InnoDB | MERGE | MRG_MYISAM | MYISAM }
or AUTO_INCREMENT = #
or AVG_ROW_LENGTH = #
or CHECKSUM = {0 | 1}
or COMMENT = "string"
or MAX_ROWS = #
or MIN_ROWS = #
or PACK_KEYS = {0 | 1 | DEFAULT}
or PASSWORD = "string"
or DELAY_KEY_WRITE = {0 | 1}
or      ROW_FORMAT= { default | dynamic | fixed | compressed }
or RAID_TYPE= {1 | STRIPED | RAID0 } RAID_CHUNKS=# RAID_CHUNKSIZE=#
or UNION = (table_name,[table_name...])

```

```

or INSERT_METHOD= {NO | FIRST | LAST }
or      DATA DIRECTORY="absolute path to directory"
or      INDEX DIRECTORY="absolute path to directory"

```

```

select_statement:
[IGNORE | REPLACE] SELECT ... (Some legal select statement)

```

CREATE TABLE creates a table with the given name in the current database. Rules for allowable table names are given in [\(undefined\) \[Legal names\], page \(undefined\)](#). An error occurs if there is no current database or if the table already exists.

In MySQL Version 3.22 or later, the table name can be specified as `db_name.nome_tabela`. This works regardless of whether there is a current database.

From MySQL Version 3.23, you can use the **TEMPORARY** keyword when you create a table. The name is restricted to the current connection, and the temporary table will automatically be deleted when the connection is closed. This means that two different connections can both use the same temporary table name without conflicting with each other or with an existing table of the same name. (The existing table is hidden until the temporary table is deleted.). From MySQL 4.0.2 one must have the **CREATE TEMPORARY TABLES** privilege to be able to create temporary tables.

In MySQL Version 3.23 or later, you can use the keywords **IF NOT EXISTS** so that an error does not occur if the table already exists. Note that there is no verification that the table structures are identical.

In MySQL 4.1 you can use **LIKE** to create a table based on a table definition in another table. In MySQL 4.1 you can also specify the type for a generated column:

```
CREATE TABLE foo (a tinyint not null) SELECT b+1 AS 'a' FROM bar;
```

Each table `nome_tabela` is represented by some files in the database directory. In the case of MyISAM-type tables you will get:

File	Purpose
<code>nome_tabela.frm</code>	Table definition (form file)
<code>nome_tabela.MYD</code>	Datafile
<code>nome_tabela.MYI</code>	Index file

For more information on the properties of the various column types, see [\(undefined\) \[Column types\], page \(undefined\)](#):

- If neither **NULL** nor **NOT NULL** is specified, the column is treated as though **NULL** had been specified.
- An integer column may have the additional attribute **AUTO_INCREMENT**. When you insert a value of **NULL** (recommended) or 0 into an **AUTO_INCREMENT** column, the column is set to `value+1`, where `value` is the largest value for the column currently in the table. **AUTO_INCREMENT** sequences begin with 1. See [\(undefined\) \[mysql_insert_id\(\)\], page \(undefined\)](#).

If you delete the row containing the maximum value for an **AUTO_INCREMENT** column, the value will be reused with an ISAM, or BDB table but not with a MyISAM or InnoDB table. If you delete all rows in the table with **DELETE FROM table_name** (without a **WHERE**) in **AUTOCOMMIT** mode, the sequence starts over for all table types.

Note: there can be only one `AUTO_INCREMENT` column per table, and it must be indexed. MySQL Version 3.23 will also only work properly if the `AUTO_INCREMENT` column only has positive values. Inserting a negative number is regarded as inserting a very large positive number. This is done to avoid precision problems when numbers 'wrap' over from positive to negative and also to ensure that one doesn't accidentally get an `AUTO_INCREMENT` column that contains 0.

In MyISAM and BDB tables you can specify `AUTO_INCREMENT` secondary column in a multi-column key. See [AUTO_INCREMENT-snt \[example-AUTO_INCREMENT\]](#), page [AUTO_INCREMENT-pg](#).

To make MySQL compatible with some ODBC applications, you can find the last inserted row with the following query:

```
SELECT * FROM nome_tabela WHERE auto_col IS NULL
```

- `CREATE TABLE` automatically commits the current InnoDB transaction if MySQL binary logging is used.
- `NULL` values are handled differently for `TIMESTAMP` columns than for other column types. You cannot store a literal `NULL` in a `TIMESTAMP` column; setting the column to `NULL` sets it to the current date and time. Because `TIMESTAMP` columns behave this way, the `NULL` and `NOT NULL` attributes do not apply in the normal way and are ignored if you specify them.

On the other hand, to make it easier for MySQL clients to use `TIMESTAMP` columns, the server reports that such columns may be assigned `NULL` values (which is true), even though `TIMESTAMP` never actually will contain a `NULL` value. You can see this when you use `DESCRIBE nome_tabela` to get a description of your table.

Note that setting a `TIMESTAMP` column to 0 is not the same as setting it to `NULL`, because 0 is a valid `TIMESTAMP` value.

- A `DEFAULT` value has to be a constant, it cannot be a function or an expression. If no `DEFAULT` value is specified for a column, MySQL automatically assigns one, as follows.

If the column may take `NULL` as a value, the default value is `NULL`.

If the column is declared as `NOT NULL`, the default value depends on the column type:

- For numeric types other than those declared with the `AUTO_INCREMENT` attribute, the default is 0. For an `AUTO_INCREMENT` column, the default value is the next value in the sequence.
- For date and time types other than `TIMESTAMP`, the default is the appropriate zero value for the type. For the first `TIMESTAMP` column in a table, the default value is the current date and time. See [\(undefined\) \[Date and time types\]](#), page [\(undefined\)](#).
- For string types other than `ENUM`, the default value is the empty string. For `ENUM`, the default is the first enumeration value.

Default values must be constants. This means, for example, that you cannot set the default for a date column to be the value of a function such as `NOW()` or `CURRENT_DATE`.

- `KEY` is a synonym for `INDEX`.
- In MySQL, a `UNIQUE` key can have only distinct values. An error occurs if you try to add a new row with a key that matches an existing row.

- A **PRIMARY KEY** is a unique **KEY** with the extra constraint that all key columns must be defined as **NOT NULL**. In MySQL the key is named **PRIMARY**. A table can have only one **PRIMARY KEY**. If you don't have a **PRIMARY KEY** and some applications ask for the **PRIMARY KEY** in your tables, MySQL will return the first **UNIQUE** key, which doesn't have any **NULL** columns, as the **PRIMARY KEY**.
- A **PRIMARY KEY** can be a multiple-column index. However, you cannot create a multiple-column index using the **PRIMARY KEY** key attribute in a column specification. Doing so will mark only that single column as primary. You must use the **PRIMARY KEY(index_nome_coluna, ...)** syntax.
- If the **PRIMARY** or **UNIQUE** key consists of only one column and this is of type integer, you can also refer to it as **_rowid** (new in Version 3.23.11).
- If you don't assign a name to an index, the index will be assigned the same name as the first **index_nome_coluna**, with an optional suffix (**_2**, **_3**, ...) to make it unique. You can see index names for a table using **SHOW INDEX FROM nome_tabela**. See [\[SHOW DATABASE INFO\]](#), page [\[undefined\]](#).
- Only the **MyISAM**, **InnoDB**, and **BDB** table types support indexes on columns that can have **NULL** values. In other cases you must declare such columns **NOT NULL** or an error results.
- With **nome_coluna(length)** syntax, you can specify an index that uses only a part of a **CHAR** or **VARCHAR** column. This can make the index file much smaller. See [\[Indexes\]](#), page [\[undefined\]](#).
- Only the **MyISAM** table type supports indexing on **BLOB** and **TEXT** columns. When putting an index on a **BLOB** or **TEXT** column you **MUST** always specify the length of the index:


```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```
- When you use **ORDER BY** or **GROUP BY** with a **TEXT** or **BLOB** column, only the first **max_sort_length** bytes are used. See [\[BLOB\]](#), page [\[undefined\]](#).
- In MySQL Version 3.23.23 or later, you can also create special **FULLTEXT** indexes. They are used for full-text search. Only the **MyISAM** table type supports **FULLTEXT** indexes. They can be created only from **CHAR**, **VARCHAR**, and **TEXT** columns. Indexing always happens over the entire column; partial indexing is not supported. See [\[Fulltext Search\]](#), page [\[undefined\]](#) for details of operation.
- In MySQL Version 3.23.44 or later, **InnoDB** tables support checking of foreign key constraints. See [\[InnoDB\]](#), page [\[undefined\]](#). Note that the **FOREIGN KEY** syntax in **InnoDB** is more restricted than the syntax presented above. **InnoDB** does not allow **index_name** to be specified, and the columns of the referenced table always have to be explicitly named. Starting from 4.0.8 **InnoDB** supports both **ON DELETE** and **ON UPDATE** actions on foreign keys. See the **InnoDB** manual section for the precise syntax. See [\[InnoDB\]](#), page [\[undefined\]](#). For other table types, MySQL Server does parse the **FOREIGN KEY**, **CHECK**, and **REFERENCES** syntax in **CREATE TABLE** commands, but without further action being taken. See [\[ANSI diff Foreign Keys\]](#), page [\[undefined\]](#).
- Each **NULL** column takes one bit extra, rounded up to the nearest byte.
- The maximum record length in bytes can be calculated as follows:

```

row length = 1
              + (sum of column lengths)
              + (number of NULL columns + 7)/8
              + (number of variable-length columns)

```

- The `table_options` and `SELECT` options are only implemented in MySQL Version 3.23 and above.

The different table types are:

Table type	Description
BDB or BerkeleyDB	Transaction-safe tables with page locking. See [BDB] , page [BDB] .
HEAP	The data for this table is only stored in memory. See [HEAP] , page [HEAP] .
ISAM	The original storage engine. See [ISAM] , page [ISAM] .
InnoDB	Transaction-safe tables with row locking. See [InnoDB] , page [InnoDB] .
MERGE	A collection of MyISAM tables used as one table. See [MERGE] , page [MERGE] .
MRG_MyISAM	An alias for MERGE tables
MyISAM	The new binary portable storage engine that is replacing ISAM. See [MyISAM] , page [MyISAM] .

See [\[Table types\]](#), page [\[Table types\]](#).

If a table type is specified, and that particular type is not available, MySQL will choose the closest table type to the one that you have specified. For example, if `TYPE=BDB` is specified, and that distribution of MySQL does not support BDB tables, the table will be created as `MyISAM` instead.

The other table options are used to optimise the behaviour of the table. In most cases, you don't have to specify any of them. The options work for all table types, if not otherwise indicated:

Option	Description
<code>AUTO_INCREMENT</code>	The next <code>AUTO_INCREMENT</code> value you want to set for your table (MyISAM).
<code>AVG_ROW_LENGTH</code>	An approximation of the average row length for your table. You only need to set this for large tables with variable size records.
<code>CHECKSUM</code>	Set this to 1 if you want MySQL to maintain a checksum for all rows (makes the table a little slower to update but makes it easier to find corrupted tables) (MyISAM).
<code>COMMENT</code>	A 60-character comment for your table.
<code>MAX_ROWS</code>	Max number of rows you plan to store in the table.
<code>MIN_ROWS</code>	Minimum number of rows you plan to store in the table.
<code>PACK_KEYS</code>	Set this to 1 if you want to have a smaller index. This usually makes updates slower and reads faster (MyISAM, ISAM). Setting this to 0 will disable all packing of keys. Setting this to <code>DEFAULT</code> (MySQL 4.0) will tell the storage engine to only pack long <code>CHAR/VARCHAR</code> columns.

PASSWORD	Encrypt the <code>.frm</code> file with a password. This option doesn't do anything in the standard MySQL version.
DELAY_KEY_WRITE	Set this to 1 if want to delay key table updates until the table is closed (MyISAM).
ROW_FORMAT	Defines how the rows should be stored. Currently this option only works with MyISAM tables, which supports the DYNAMIC and FIXED row formats. See [MyISAM table formats] , page [undefined] .

When you use a MyISAM table, MySQL uses the product of `max_rows * avg_row_length` to decide how big the resulting table will be. If you don't specify any of the above options, the maximum size for a table will be 4G (or 2G if your operating systems only supports 2G tables). The reason for this is just to keep down the pointer sizes to make the index smaller and faster if you don't really need big files.

If you don't use `PACK_KEYS`, the default is to only pack strings, not numbers. If you use `PACK_KEYS=1`, numbers will be packed as well.

When packing binary number keys, MySQL will use prefix compression. This means that you will only get a big benefit of this if you have many numbers that are the same. Prefix compression means that every key needs one extra byte to indicate how many bytes of the previous key are the same for the next key (note that the pointer to the row is stored in high-byte-first-order directly after the key, to improve compression). This means that if you have many equal keys on two rows in a row, all following 'same' keys will usually only take 2 bytes (including the pointer to the row). Compare this to the ordinary case where the following keys will take `storage_size_for_key + pointer_size` (usually 4). On the other hand, if all keys are totally different, you will lose 1 byte per key, if the key isn't a key that can have NULL values. (In this case the packed key length will be stored in the same byte that is used to mark if a key is NULL.)

- If you specify a `SELECT` after the `CREATE` statement, MySQL will create new fields for all elements in the `SELECT`. For example:

```
mysql> CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,
->      PRIMARY KEY (a), KEY(b))
->      TYPE=MyISAM SELECT b,c FROM test2;
```

This will create a MyISAM table with three columns, a, b, and c. Notice that the columns from the `SELECT` statement are appended to the right side of the table, not overlapped onto it. Take the following example:

```
mysql> SELECT * FROM foo;
+----+
| n |
+----+
| 1 |
+----+
```

```
mysql> CREATE TABLE bar (m INT) SELECT n FROM foo;
Query OK, 1 row affected (0.02 sec)
Records: 1 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT * FROM bar;
+-----+-----+
```

```

| m      | n |
+-----+----+
| NULL   | 1 |
+-----+----+
1 row in set (0.00 sec)

```

For each row in table `foo`, a row is inserted in `bar` with the values from `foo` and default values for the new columns.

`CREATE TABLE ... SELECT` will not automatically create any indexes for you. This is done intentionally to make the command as flexible as possible. If you want to have indexes in the created table, you should specify these before the `SELECT` statement:

```
mysql> CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

If any errors occur while copying the data to the table, it will automatically be deleted.

To ensure that the update log/binary log can be used to re-create the original tables, MySQL will not allow concurrent inserts during `CREATE TABLE ... SELECT`.

- The `RAID_TYPE` option will help you to break the 2G/4G limit for the MyISAM datafile (not the index file) on operating systems that don't support big files. Note that this option is not recommended for filesystem that supports big files!

You can get more speed from the I/O bottleneck by putting RAID directories on different physical disks. `RAID_TYPE` will work on any OS, as long as you have configured MySQL with `--with-raid`. For now the only allowed `RAID_TYPE` is `STRIPED` (1 and `RAID0` are aliases for this).

If you specify `RAID_TYPE=STRIPED` for a MyISAM table, MyISAM will create `RAID_CHUNKS` subdirectories named 00, 01, 02 in the database directory. In each of these directories MyISAM will create a `table_name.MYD`. When writing data to the datafile, the RAID handler will map the first `RAID_CHUNKSIZE * 1024` bytes to the first file, the next `RAID_CHUNKSIZE * 1024` bytes to the next file and so on.

- `UNION` is used when you want to use a collection of identical tables as one. This only works with `MERGE` tables. See [\[MERGE\]](#), page [\[MERGE\]](#).

For the moment you need to have `SELECT`, `UPDATE`, and `DELETE` privileges on the tables you map to a `MERGE` table. All mapped tables must be in the same database as the `MERGE` table.

- If you want to insert data in a `MERGE` table, you have to specify with `INSERT_METHOD` into which table the row should be inserted. See [\[MERGE\]](#), page [\[MERGE\]](#). This option was introduced in MySQL 4.0.0.
- In the created table the `PRIMARY` key will be placed first, followed by all `UNIQUE` keys and then the normal keys. This helps the MySQL optimiser to prioritise which key to use and also more quickly detect duplicated `UNIQUE` keys.
- By using `DATA DIRECTORY="directory"` or `INDEX DIRECTORY="directory"` you can specify where the storage engine should put its table and index files. Note that the directory should be a full path to the directory (not relative path).

This only works for MyISAM tables in MySQL 4.0, when you are not using the `--skip-symlink` option. See [\[Symbolic links to tables\]](#), page [\[Symbolic links to tables\]](#).

6.5.3.1 Silent Column Specification Changes

In some cases, MySQL silently changes a column specification from that given in a `CREATE TABLE` statement. (This may also occur with `ALTER TABLE`.):

- `VARCHAR` columns with a length less than four are changed to `CHAR`.
- If any column in a table has a variable length, the entire row is variable-length as a result. Therefore, if a table contains any variable-length columns (`VARCHAR`, `TEXT`, or `BLOB`), all `CHAR` columns longer than three characters are changed to `VARCHAR` columns. This doesn't affect how you use the columns in any way; in MySQL, `VARCHAR` is just a different way to store characters. MySQL performs this conversion because it saves space and makes table operations faster. See [\(undefined\) \[Table types\], page \(undefined\)](#).
- `TIMESTAMP` display sizes must be even and in the range from 2 to 14. If you specify a display size of 0 or greater than 14, the size is coerced to 14. Odd-valued sizes in the range from 1 to 13 are coerced to the next higher even number.
- You cannot store a literal `NULL` in a `TIMESTAMP` column; setting it to `NULL` sets it to the current date and time. Because `TIMESTAMP` columns behave this way, the `NULL` and `NOT NULL` attributes do not apply in the normal way and are ignored if you specify them. `DESCRIBE nome_tabela` always reports that a `TIMESTAMP` column may be assigned `NULL` values.
- MySQL maps certain column types used by other SQL database vendors to MySQL types. See [vendor column types-snt \[Other-vendor column types\], page vendor column types-pg](#).

If you want to see whether MySQL used a column type other than the one you specified, issue a `DESCRIBE nome_tabela` statement after creating or altering your table.

Certain other column type changes may occur if you compress a table using `myisampack`. See [\(undefined\) \[Compressed format\], page \(undefined\)](#).

6.5.4 ALTER TABLE Syntax

```
ALTER [IGNORE] TABLE nome_tabela alter_spec [, alter_spec ...]
```

alter_specification:

```

    ADD [COLUMN] create_definition [FIRST | AFTER column_name ]
  or  ADD [COLUMN] (create_definition, create_definition,...)
  or  ADD INDEX [index_name] (index_nome_coluna,...)
  or  ADD PRIMARY KEY (index_nome_coluna,...)
  or  ADD UNIQUE [index_name] (index_nome_coluna,...)
  or  ADD FULLTEXT [index_name] (index_nome_coluna,...)
  or  ADD [CONSTRAINT symbol] FOREIGN KEY [index_name] (index_nome_coluna,...)
      [reference_definition]
  or  ALTER [COLUMN] nome_coluna {SET DEFAULT literal | DROP DEFAULT}
  or  CHANGE [COLUMN] old_nome_coluna create_definition
      [FIRST | AFTER column_name]
  or  MODIFY [COLUMN] create_definition [FIRST | AFTER column_name]
  or  DROP [COLUMN] nome_coluna
  or  DROP PRIMARY KEY

```

```

or    DROP INDEX index_name
or    DISABLE KEYS
or    ENABLE KEYS
or    RENAME [TO] new_nome_tabela
or    ORDER BY col
or    table_options

```

`ALTER TABLE` allows you to change the structure of an existing table. For example, you can add or delete columns, create or destroy indexes, change the type of existing columns, or rename columns or the table itself. You can also change the comment for the table and type of the table. See [\[CREATE TABLE\]](#), page [\[CREATE TABLE\]](#).

If you use `ALTER TABLE` to change a column specification but `DESCRIBE nome_tabela` indicates that your column was not changed, it is possible that MySQL ignored your modification for one of the reasons described in [\[Silent column changes\]](#), page [\[Silent column changes\]](#). For example, if you try to change a `VARCHAR` column to `CHAR`, MySQL will still use `VARCHAR` if the table contains other variable-length columns.

`ALTER TABLE` works by making a temporary copy of the original table. The alteration is performed on the copy, then the original table is deleted and the new one is renamed. This is done in such a way that all updates are automatically redirected to the new table without any failed updates. While `ALTER TABLE` is executing, the original table is readable by other clients. Updates and writes to the table are stalled until the new table is ready.

Note that if you use any other option to `ALTER TABLE` than `RENAME`, MySQL will always create a temporary table, even if the data wouldn't strictly need to be copied (like when you change the name of a column). We plan to fix this in the future, but as one doesn't normally do `ALTER TABLE` that often this isn't that high on our TODO. For MyISAM tables, you can speed up the index recreation part (which is the slowest part of the recreation process) by setting the `myisam_sort_buffer_size` variable to a high value.

- To use `ALTER TABLE`, you need `ALTER`, `INSERT`, and `CREATE` privileges on the table.
- `IGNORE` is a MySQL extension to SQL-92. It controls how `ALTER TABLE` works if there are duplicates on unique keys in the new table. If `IGNORE` isn't specified, the copy is aborted and rolled back. If `IGNORE` is specified, then for rows with duplicates on a unique key, only the first row is used; the others are deleted.
- You can issue multiple `ADD`, `ALTER`, `DROP`, and `CHANGE` clauses in a single `ALTER TABLE` statement. This is a MySQL extension to SQL-92, which allows only one of each clause per `ALTER TABLE` statement.
- `CHANGE nome_coluna`, `DROP nome_coluna`, and `DROP INDEX` are MySQL extensions to SQL-92.
- `MODIFY` is an Oracle extension to `ALTER TABLE`.
- The optional word `COLUMN` is a pure noise word and can be omitted.
- If you use `ALTER TABLE nome_tabela RENAME TO new_name` without any other options, MySQL simply renames the files that correspond to the table `nome_tabela`. There is no need to create the temporary table. See [\[RENAME TABLE\]](#), page [\[RENAME TABLE\]](#).
- `create_definition` clauses use the same syntax for `ADD` and `CHANGE` as for `CREATE TABLE`. Note that this syntax includes the column name, not just the column type. See [\[CREATE TABLE\]](#), page [\[CREATE TABLE\]](#).

- You can rename a column using a `CHANGE old_nome_coluna create_definition` clause. To do so, specify the old and new column names and the type that the column currently has. For example, to rename an `INTEGER` column from `a` to `b`, you can do this:

```
mysql> ALTER TABLE t1 CHANGE a b INTEGER;
```

If you want to change a column's type but not the name, `CHANGE` syntax still requires two column names even if they are the same. For example:

```
mysql> ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
```

However, as of MySQL Version 3.22.16a, you can also use `MODIFY` to change a column's type without renaming it:

```
mysql> ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
```

- If you use `CHANGE` or `MODIFY` to shorten a column for which an index exists on part of the column (for instance, if you have an index on the first 10 characters of a `VARCHAR` column), you cannot make the column shorter than the number of characters that are indexed.
- When you change a column type using `CHANGE` or `MODIFY`, MySQL tries to convert data to the new type as well as possible.
- In MySQL Version 3.22 or later, you can use `FIRST` or `ADD ... AFTER nome_coluna` to add a column at a specific position within a table row. The default is to add the column last. From MySQL Version 4.0.1, you can also use the `FIRST` and `AFTER` keywords in `CHANGE` or `MODIFY`.
- `ALTER COLUMN` specifies a new default value for a column or removes the old default value. If the old default is removed and the column can be `NULL`, the new default is `NULL`. If the column cannot be `NULL`, MySQL assigns a default value, as described in [\[CREATE TABLE\]](#), page [\[CREATE TABLE\]](#).
- `DROP INDEX` removes an index. This is a MySQL extension to SQL-92. See [\[DROP INDEX\]](#), page [\[DROP INDEX\]](#).
- If columns are dropped from a table, the columns are also removed from any index of which they are a part. If all columns that make up an index are dropped, the index is dropped as well.
- If a table contains only one column, the column cannot be dropped. If what you intend is to remove the table, use `DROP TABLE` instead.
- `DROP PRIMARY KEY` drops the primary index. If no such index exists, it drops the first `UNIQUE` index in the table. (MySQL marks the first `UNIQUE` key as the `PRIMARY KEY` if no `PRIMARY KEY` was specified explicitly.)

If you add a `UNIQUE INDEX` or `PRIMARY KEY` to a table, this is stored before any not `UNIQUE` index so that MySQL can detect duplicate keys as early as possible.

- `ORDER BY` allows you to create the new table with the rows in a specific order. Note that the table will not remain in this order after inserts and deletes. In some cases, it may make sorting easier for MySQL if the table is in order by the column that you wish to order it by later. This option is mainly useful when you know that you are mostly going to query the rows in a certain order; by using this option after big changes to the table, you may be able to get higher performance.

- If you use `ALTER TABLE` on a MyISAM table, all non-unique indexes are created in a separate batch (like in `REPAIR`). This should make `ALTER TABLE` much faster when you have many indexes.
- Since **MySQL 4.0** the above feature can be activated explicitly. `ALTER TABLE ... DISABLE KEYS` makes MySQL to stop updating non-unique indexes for MyISAM table. `ALTER TABLE ... ENABLE KEYS` then should be used to recreate missing indexes. As MySQL does it with special algorithm which is much faster than inserting keys one by one, disabling keys could give a considerable speedup on bulk inserts.
- With the C API function `mysql_info()`, you can find out how many records were copied, and (when `IGNORE` is used) how many records were deleted due to duplication of unique key values.
- The `FOREIGN KEY`, `CHECK`, and `REFERENCES` clauses don't actually do anything, except for InnoDB type tables which support `ADD CONSTRAINT FOREIGN KEY (...)` `REFERENCES ... (...)`. Note that InnoDB does not allow an `index_name` to be specified. See [\[InnoDB\], page <undefined>](#). The syntax for other table types is provided only for compatibility, to make it easier to port code from other SQL servers and to run applications that create tables with references. See [\[Differences from ANSI\], page <undefined>](#).

Here is an example that shows some of the uses of `ALTER TABLE`. We begin with a table `t1` that is created as shown here:

```
mysql> CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

To rename the table from `t1` to `t2`:

```
mysql> ALTER TABLE t1 RENAME t2;
```

To change column `a` from `INTEGER` to `TINYINT NOT NULL` (leaving the name the same), and to change column `b` from `CHAR(10)` to `CHAR(20)` as well as renaming it from `b` to `c`:

```
mysql> ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

To add a new `TIMESTAMP` column named `d`:

```
mysql> ALTER TABLE t2 ADD d TIMESTAMP;
```

To add an index on column `d`, and make column `a` the primary key:

```
mysql> ALTER TABLE t2 ADD INDEX (d), ADD PRIMARY KEY (a);
```

To remove column `c`:

```
mysql> ALTER TABLE t2 DROP COLUMN c;
```

To add a new `AUTO_INCREMENT` integer column named `c`:

```
mysql> ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
      ADD INDEX (c);
```

Note that we indexed `c`, because `AUTO_INCREMENT` columns must be indexed, and also that we declare `c` as `NOT NULL`, because indexed columns cannot be `NULL`.

When you add an `AUTO_INCREMENT` column, column values are filled in with sequence numbers for you automatically. You can set the first sequence number by executing `SET INSERT_ID=#` before `ALTER TABLE` or using the `AUTO_INCREMENT = #` table option. See [\[SET OPTION\], page <undefined>](#).

With MyISAM tables, if you don't change the `AUTO_INCREMENT` column, the sequence number will not be affected. If you drop an `AUTO_INCREMENT` column and then add another `AUTO_INCREMENT` column, the numbers will start from 1 again.

See [\(undefined\) \[ALTER TABLE problems\]](#), page [\(undefined\)](#).

6.5.5 RENAME TABLE Syntax

```
RENAME TABLE nome_tabela TO new_nome_tabela[, nome_tabela2 TO new_tbl_name2,...]
```

The rename is done atomically, which means that no other thread can access any of the tables while the rename is running. This makes it possible to replace a table with an empty one:

```
CREATE TABLE new_table (...);
RENAME TABLE old_table TO backup_table, new_table TO old_table;
```

The rename is done from left to right, which means that if you want to swap two tables names, you have to:

```
RENAME TABLE old_table    TO backup_table,
              new_table    TO old_table,
              backup_table TO new_table;
```

As long as two databases are on the same disk you can also rename from one database to another:

```
RENAME TABLE current_db.nome_tabela TO other_db.nome_tabela;
```

When you execute `RENAME`, you can't have any locked tables or active transactions. You must also have the `ALTER` and `DROP` privileges on the original table, and the `CREATE` and `INSERT` privileges on the new table.

If MySQL encounters any errors in a multiple-table rename, it will do a reverse rename for all renamed tables to get everything back to the original state.

`RENAME TABLE` was added in MySQL 3.23.23.

6.5.6 DROP TABLE Syntax

```
DROP [TEMPORARY] TABLE [IF EXISTS] nome_tabela [, nome_tabela,...] [RESTRICT | CASCADE]
```

`DROP TABLE` removes one or more tables. All table data and the table definition are *removed*, so **be careful** with this command!

In MySQL Version 3.22 or later, you can use the keywords `IF EXISTS` to prevent an error from occurring for tables that don't exist. In 4.1 one gets a `NOTE` for all not existing tables when using `IF EXISTS`. See [\(undefined\) \[SHOW WARNINGS\]](#), page [\(undefined\)](#).

`RESTRICT` and `CASCADE` are allowed to make porting easier. For the moment they don't do anything.

Note: `DROP TABLE` will automatically commit current active transaction (except if you are using 4.1 and the `TEMPORARY` key word).

Option `TEMPORARY` is ignored in 4.0. In 4.1 this option works as follows:

- Only drops temporary tables.
- It doesn't end a running transactions.
- No access rights is checked.

Using `TEMPORARY` is a good way to ensure that you don't accidently drop a real table.

6.5.7 CREATE INDEX Syntax

```
CREATE [UNIQUE|FULLTEXT] INDEX index_name
      ON nome_tabela (nome_coluna[(length)],... )
```

The `CREATE INDEX` statement doesn't do anything in MySQL prior to Version 3.22. In Version 3.22 or later, `CREATE INDEX` is mapped to an `ALTER TABLE` statement to create indexes. See [\[ALTER TABLE\]](#), page [\[undefined\]](#).

Normally, you create all indexes on a table at the time the table itself is created with `CREATE TABLE`. See [\[CREATE TABLE\]](#), page [\[undefined\]](#). `CREATE INDEX` allows you to add indexes to existing tables.

A column list of the form `(col1,col2,...)` creates a multiple-column index. Index values are formed by concatenating the values of the given columns.

For `CHAR` and `VARCHAR` columns, indexes can be created that use only part of a column, using `nome_coluna(length)` syntax. (On `BLOB` and `TEXT` columns the length is required.) The statement shown here creates an index using the first 10 characters of the `name` column:

```
mysql> CREATE INDEX part_of_name ON customer (name(10));
```

Because most names usually differ in the first 10 characters, this index should not be much slower than an index created from the entire `name` column. Also, using partial columns for indexes can make the index file much smaller, which could save a lot of disk space and might also speed up `INSERT` operations!

Note that you can only add an index on a column that can have `NULL` values or on a `BLOB/TEXT` column if you are using MySQL Version 3.23.2 or newer and are using the `MyISAM` table type.

For more information about how MySQL uses indexes, see [\[MySQL indexes\]](#), page [\[undefined\]](#).

`FULLTEXT` indexes can index only `VARCHAR` and `TEXT` columns, and only in `MyISAM` tables. `FULLTEXT` indexes are available in MySQL Version 3.23.23 and later. [\[Fulltext Search\]](#), page [\[undefined\]](#).

6.5.8 DROP INDEX Syntax

```
DROP INDEX index_name ON nome_tabela
```

`DROP INDEX` drops the index named `index_name` from the table `nome_tabela`. `DROP INDEX` doesn't do anything in MySQL prior to Version 3.22. In Version 3.22 or later, `DROP INDEX` is mapped to an `ALTER TABLE` statement to drop the index. See [\[ALTER TABLE\]](#), page [\[undefined\]](#).

6.6 Basic MySQL User Utility Commands

6.6.1 USE Syntax

```
USE db_name
```

The `USE db_name` statement tells MySQL to use the `db_name` database as the default database for subsequent queries. The database remains current until the end of the session or until another `USE` statement is issued:

```
mysql> USE db1;
mysql> SELECT COUNT(*) FROM mytable;      # selects from db1.mytable
mysql> USE db2;
mysql> SELECT COUNT(*) FROM mytable;      # selects from db2.mytable
```

Making a particular database current by means of the `USE` statement does not preclude you from accessing tables in other databases. The following example accesses the `author` table from the `db1` database and the `editor` table from the `db2` database:

```
mysql> USE db1;
mysql> SELECT author_name,editor_name FROM author,db2.editor
->      WHERE author.editor_id = db2.editor.editor_id;
```

The `USE` statement is provided for Sybase compatibility.

6.6.2 DESCRIBE Syntax (Get Information About Columns)

```
{DESCRIBE | DESC} nome_tabela [nome_coluna | wild]
```

`DESCRIBE` is a shortcut for `SHOW COLUMNS FROM`. See [\(undefined\)](#) [`SHOW DATABASE INFO`], page [\(undefined\)](#).

`DESCRIBE` provides information about a table's columns. `nome_coluna` may be a column name or a string containing the SQL `'%'` and `'_'` wildcard characters. There is no need to enclose the string in quotes.

If the column types are different from what you expect them to be based on a `CREATE TABLE` statement, note that MySQL sometimes changes column types. See [\(undefined\)](#) [`Silent column changes`], page [\(undefined\)](#).

This statement is provided for Oracle compatibility.

The `SHOW` statement provides similar information. See [\(undefined\)](#) [`SHOW`], page [\(undefined\)](#).

6.7 MySQL Transactional and Locking Commands

6.7.1 BEGIN/COMMIT/ROLLBACK Syntax

By default, MySQL runs in `autocommit` mode. This means that as soon as you execute an update, MySQL will store the update on disk.

If you are using transactions safe tables (like InnoDB, BDB, you can put MySQL into non-`autocommit` mode with the following command:

```
SET AUTOCOMMIT=0
```

After this you must use `COMMIT` to store your changes to disk or `ROLLBACK` if you want to ignore the changes you have made since the beginning of your transaction.

If you want to switch from `AUTOCOMMIT` mode for one series of statements, you can use the `START TRANSACTION` or `BEGIN` or `BEGIN WORK` statement:

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summmmary=@A WHERE type=1;
COMMIT;
```

`START TRANSACTION` was added to MySQL 4.0.11; This is the recommended way to start an ad-hoc transaction as this is SQL-99 syntax,

Note that if you are using non-transaction-safe tables, the changes will be stored at once, independent of the status of the `autocommit` mode.

If you do a `ROLLBACK` when you have updated a non-transactional table you will get an error (`ER_WARNING_NOT_COMPLETE_ROLLBACK`) as a warning. All transaction-safe tables will be restored but any non-transaction-safe table will not change.

If you are using `START TRANSACTION` or `SET AUTOCOMMIT=0`, you should use the MySQL binary log for backups instead of the older update log. Transactions are stored in the binary log in one chunk, upon `COMMIT`, to ensure that transactions which are rolled back are not stored. See [\[Binary log\]](#), page [\[undefined\]](#).

The following commands automatically end a transaction (as if you had done a `COMMIT` before executing the command):

Command	Command	Command
<code>ALTER TABLE</code>	<code>BEGIN</code>	<code>CREATE INDEX</code>
<code>DROP DATABASE</code>	<code>DROP TABLE</code>	<code>RENAME TABLE</code>
<code>TRUNCATE</code>		

You can change the isolation level for transactions with `SET TRANSACTION ISOLATION LEVEL` See [\[SET TRANSACTION\]](#), page [\[undefined\]](#).

6.7.2 LOCK TABLES/UNLOCK TABLES Syntax

```
LOCK TABLES nome_tabela [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE}
    [, nome_tabela [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE} ...]
...
UNLOCK TABLES
```

`LOCK TABLES` locks tables for the current thread. `UNLOCK TABLES` releases any locks held by the current thread. All tables that are locked by the current thread are automatically unlocked when the thread issues another `LOCK TABLES`, or when the connection to the server is closed.

To use `LOCK TABLES` in MySQL 4.0.2 you need the global `LOCK TABLES` privilege and a `SELECT` privilege on the involved tables. In MySQL 3.23 you need to have `SELECT`, `insert`, `DELETE` and `UPDATE` privileges for the tables.

The main reasons to use `LOCK TABLES` are for emulating transactions or getting more speed when updating tables. This is explained in more detail later.

If a thread obtains a `READ` lock on a table, that thread (and all other threads) can only read from the table. If a thread obtains a `WRITE` lock on a table, then only the thread holding the lock can read from or write to the table. Other threads are blocked.

The difference between `READ LOCAL` and `READ` is that `READ LOCAL` allows non-conflicting `INSERT` statements to execute while the lock is held. This can't however be used if you are going to manipulate the database files outside MySQL while you hold the lock.

When you use `LOCK TABLES`, you must lock all tables that you are going to use and you must use the same alias that you are going to use in your queries! If you are using a table multiple times in a query (with aliases), you must get a lock for each alias!

`WRITE` locks normally have higher priority than `READ` locks, to ensure that updates are processed as soon as possible. This means that if one thread obtains a `READ` lock and then another thread requests a `WRITE` lock, subsequent `READ` lock requests will wait until the

WRITE thread has gotten the lock and released it. You can use `LOW_PRIORITY WRITE` locks to allow other threads to obtain READ locks while the thread is waiting for the WRITE lock. You should only use `LOW_PRIORITY WRITE` locks if you are sure that there will eventually be a time when no threads will have a READ lock.

`LOCK TABLES` works as follows:

1. Sort all tables to be locked in a internally defined order (from the user standpoint the order is undefined).
2. If a table is locked with a read and a write lock, put the write lock before the read lock.
3. Lock one table at a time until the thread gets all locks.

This policy ensures that table locking is deadlock free. There is however other things one needs to be aware of with this schema:

If you are using a `LOW_PRIORITY WRITE` lock for a table, this means only that MySQL will wait for this particular lock until there is no threads that wants a READ lock. When the thread has got the WRITE lock and is waiting to get the lock for the next table in the lock table list, all other threads will wait for the WRITE lock to be released. If this becomes a serious problem with your application, you should consider converting some of your tables to transactions safe tables.

You can safely kill a thread that is waiting for a table lock with `KILL`. See [\(undefined\) \[KILL\], page \(undefined\)](#).

Note that you should **not** lock any tables that you are using with `INSERT DELAYED`. This is because that in this case the `INSERT` is done by a separate thread.

Normally, you don't have to lock tables, as all single `UPDATE` statements are atomic; no other thread can interfere with any other currently executing SQL statement. There are a few cases when you would like to lock tables anyway:

- If you are going to run many operations on a bunch of tables, it's much faster to lock the tables you are going to use. The downside is, of course, that no other thread can update a READ-locked table and no other thread can read a WRITE-locked table.

The reason some things are faster under `LOCK TABLES` is that MySQL will not flush the key cache for the locked tables until `UNLOCK TABLES` is called (normally the key cache is flushed after each SQL statement). This speeds up inserting/updating/deletes on MyISAM tables.

- If you are using a storage engine in MySQL that doesn't support transactions, you must use `LOCK TABLES` if you want to ensure that no other thread comes between a `SELECT` and an `UPDATE`. The example shown here requires `LOCK TABLES` in order to execute safely:

```
mysql> LOCK TABLES trans READ, customer WRITE;
mysql> SELECT SUM(value) FROM trans WHERE customer_id=some_id;
mysql> UPDATE customer SET total_value=sum_from_previous_statement
->      WHERE customer_id=some_id;
mysql> UNLOCK TABLES;
```

Without `LOCK TABLES`, there is a chance that another thread might insert a new row in the `trans` table between execution of the `SELECT` and `UPDATE` statements.

By using incremental updates (`UPDATE customer SET value=value+new_value`) or the `LAST_INSERT_ID()` function, you can avoid using `LOCK TABLES` in many cases.

You can also solve some cases by using the user-level lock functions `GET_LOCK()` and `RELEASE_LOCK()`. These locks are saved in a hash table in the server and implemented with `pthread_mutex_lock()` and `pthread_mutex_unlock()` for high speed. See [\(undefined\)](#) [Miscellaneous functions], page [\(undefined\)](#).

See [\(undefined\)](#) [Internal locking], page [\(undefined\)](#), for more information on locking policy. You can lock all tables in all databases with read locks with the `FLUSH TABLES WITH READ LOCK` command. See [\(undefined\)](#) [FLUSH], page [\(undefined\)](#). This is very convenient way to get backups if you have a filesystem, like Veritas, that can take snapshots in time.

NOTE: `LOCK TABLES` is not transaction-safe and will automatically commit any active transactions before attempting to lock the tables.

6.7.3 SET TRANSACTION Syntax

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }
```

Sets the transaction isolation level for the global, whole session or the next transaction.

The default behaviour is to set the isolation level for the next (not started) transaction. If you use the `GLOBAL` keyword, the statement sets the default transaction level globally for all new connections created from that point on. You will need the `SUPER` privilege to do this. Using the `SESSION` keyword sets the default transaction level for all future transactions performed on the current connection.

You can set the default global isolation level for `mysqld` with `--transaction-isolation=...`. See [line options-snt](#) [Command-line options], page [line options-pg](#).

6.8 MySQL Full-text Search

As of Version 3.23.23, MySQL has support for full-text indexing and searching. Full-text indexes in MySQL are an index of type `FULLTEXT`. `FULLTEXT` indexes are used with `MyISAM` tables and can be created from `CHAR`, `VARCHAR`, or `TEXT` columns at `CREATE TABLE` time or added later with `ALTER TABLE` or `CREATE INDEX`. For large datasets, it will be much faster to load your data into a table that has no `FULLTEXT` index, then create the index with `ALTER TABLE` (or `CREATE INDEX`). Loading data into a table that already has a `FULLTEXT` index will be slower.

Full-text searching is performed with the `MATCH()` function.

```
mysql> CREATE TABLE articles (
->   id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
->   title VARCHAR(200),
->   body TEXT,
->   FULLTEXT (title,body)
-> );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO articles VALUES
-> (NULL,'MySQL Tutorial', 'DBMS stands for DataBase ...'),
-> (NULL,'How To Use MySQL Efficiently', 'After you went through a ...'),
-> (NULL,'Optimising MySQL','In this tutorial we will show ...'),
```

```

-> (NULL,'1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
-> (NULL,'MySQL vs. YourSQL', 'In the following database comparison ...'),
-> (NULL,'MySQL Security', 'When configured properly, MySQL ...');
Query OK, 6 rows affected (0.00 sec)
Records: 6 Duplicates: 0 Warnings: 0

```

```

mysql> SELECT * FROM articles
->          WHERE MATCH (title,body) AGAINST ('database');
+-----+-----+-----+
| id | title                | body                |
+-----+-----+-----+
| 5 | MySQL vs. YourSQL    | In the following database comparison ... |
| 1 | MySQL Tutorial      | DBMS stands for DataBase ...           |
+-----+-----+-----+
2 rows in set (0.00 sec)

```

The `MATCH()` function performs a natural language search for a string against a text collection (a set of one or more columns included in a `FULLTEXT` index). The search string is given as the argument to `AGAINST()`. The search is performed in case-insensitive fashion. For every row in the table, `MATCH()` returns a relevance value, that is, a similarity measure between the search string and the text in that row in the columns named in the `MATCH()` list.

When `MATCH()` is used in a `WHERE` clause (see example above) the rows returned are automatically sorted with highest relevance first. Relevance values are non-negative floating-point numbers. Zero relevance means no similarity. Relevance is computed based on the number of words in the row, the number of unique words in that row, the total number of words in the collection, and the number of documents (rows) that contain a particular word.

It is also possible to perform a boolean mode search. This is explained later in the section. The preceding example is a basic illustration showing how to use the `MATCH()` function. Rows are returned in order of decreasing relevance.

The next example shows how to retrieve the relevance values explicitly. As neither `WHERE` nor `ORDER BY` clauses are present, returned rows are not ordered.

```

mysql> SELECT id,MATCH (title,body) AGAINST ('Tutorial') FROM articles;
+-----+-----+-----+
| id | MATCH (title,body) AGAINST ('Tutorial') |
+-----+-----+-----+
| 1 | 0.64840710366884 |
| 2 | 0 |
| 3 | 0.66266459031789 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
+-----+-----+-----+
6 rows in set (0.00 sec)

```

The following example is more complex. The query returns the relevance and still sorts the rows in order of decreasing relevance. To achieve this result, you should specify `MATCH()` twice. This will cause no additional overhead, because the MySQL optimiser will notice that the two `MATCH()` calls are identical and invoke the full-text search code only once.

```
mysql> SELECT id, body, MATCH (title,body) AGAINST
-> ('Security implications of running MySQL as root') AS score
-> FROM articles WHERE MATCH (title,body) AGAINST
-> ('Security implications of running MySQL as root');
+----+-----+-----+-----+
| id | body                                | score          |
+----+-----+-----+-----+
| 4  | 1. Never run mysqld as root. 2. ... | 1.5055546709332 |
| 6  | When configured properly, MySQL ... | 1.31140957288  |
+----+-----+-----+-----+
2 rows in set (0.00 sec)
```

MySQL uses a very simple parser to split text into words. A “word” is any sequence of characters consisting of letters, digits, ‘ ’, and ‘_’. Any “word” that is present in the stopword list or is just too short (3 characters or less) is ignored.

Every correct word in the collection and in the query is weighted according to its significance in the query or collection. This way, a word that is present in many documents will have lower weight (and may even have a zero weight), because it has lower semantic value in this particular collection. Otherwise, if the word is rare, it will receive a higher weight. The weights of the words are then combined to compute the relevance of the row.

Such a technique works best with large collections (in fact, it was carefully tuned this way). For very small tables, word distribution does not reflect adequately their semantic value, and this model may sometimes produce bizarre results.

```
mysql> SELECT * FROM articles WHERE MATCH (title,body) AGAINST ('MySQL');
Empty set (0.00 sec)
```

The search for the word MySQL produces no results in the above example, because that word is present in more than half the rows. As such, it is effectively treated as a stopword (that is, a word with zero semantic value). This is the most desirable behaviour -- a natural language query should not return every second row from a 1 GB table.

A word that matches half of rows in a table is less likely to locate relevant documents. In fact, it will most likely find plenty of irrelevant documents. We all know this happens far too often when we are trying to find something on the Internet with a search engine. It is with this reasoning that such rows have been assigned a low semantic value in **this particular dataset**.

As of Version 4.0.1, MySQL can also perform boolean full-text searches using the IN BOOLEAN MODE modifier.

```
mysql> SELECT * FROM articles WHERE MATCH (title,body)
-> AGAINST ('+MySQL -YourSQL' IN BOOLEAN MODE);
+----+-----+-----+-----+
----+
| id | title                                | body                                |
+----+-----+-----+-----+
----+
| 1  | MySQL Tutorial                        | DBMS stands for DataBase ...      |
| 2  | How To Use MySQL Efficiently          | After you went through a ...      |
| 3  | Optimising MySQL                      | In this tutorial we will show ...  |
| 4  | 1001 MySQL Tricks                     | 1. Never run mysqld as root. 2. ... |
+----+-----+-----+-----+
|
```

```
| 6 | MySQL Security | When configured properly, MySQL ... |
+---+-----+-----+-----+-----+-----+-----+-----+
----+
```

This query retrieved all the rows that contain the word `MySQL` (note: the 50% threshold is not used), but that do **not** contain the word `YourSQL`. Note that a boolean mode search does not automatically sort rows in order of decreasing relevance. You can see this from result of the preceding query, where the row with the highest relevance (the one that contains `MySQL` twice) is listed last, not first. A boolean full-text search can also work even without a `FULLTEXT` index, although it would be **slow**.

The boolean full-text search capability supports the following operators:

- + A leading plus sign indicates that this word **must be** present in every row returned.
- A leading minus sign indicates that this word **must not be** present in any row returned.

By default (when neither plus nor minus is specified) the word is optional, but the rows that contain it will be rated higher. This mimicks the behaviour of `MATCH() ... AGAINST()` without the `IN BOOLEAN MODE` modifier.
- < > These two operators are used to change a word's contribution to the relevance value that is assigned to a row. The < operator decreases the contribution and the > operator increases it. See the example below.
- () Parentheses are used to group words into subexpressions.
- ~ A leading tilde acts as a negation operator, causing the word's contribution to the row relevance to be negative. It's useful for marking noise words. A row that contains such a word will be rated lower than others, but will not be excluded altogether, as it would be with the - operator.
- * An asterisk is the truncation operator. Unlike the other operators, it should be **appended** to the word, not prepended.
- " The phrase, that is enclosed in double quotes "", matches only rows that contain this phrase **literally, as it was typed**.

And here are some examples:

```
apple banana
```

find rows that contain at least one of these words.

```
+apple +juice
```

... both words.

```
+apple macintosh
```

... word "apple", but rank it higher if it also contain "macintosh".

```
+apple -macintosh
```

... word "apple" but not "macintosh".

```
+apple +(>pie <strudel)
```

... "apple" and "pie", or "apple" and "strudel" (in any order), but rank "apple pie" higher than "apple strudel".

`apple*` ... “apple”, “apples”, “applesauce”, and “applet”.

`"some words"`

... “some words of wisdom”, but not “some noise words”.

6.8.1 Full-text Restrictions

- All parameters to the `MATCH()` function must be columns from the same table that is part of the same `FULLTEXT` index, unless the `MATCH()` is `IN BOOLEAN MODE`.
- The `MATCH()` column list must exactly match the column list in some `FULLTEXT` index definition for the table, unless this `MATCH()` is `IN BOOLEAN MODE`.
- The argument to `AGAINST()` must be a constant string.

6.8.2 Fine-tuning MySQL Full-text Search

Unfortunately, full-text search has few user-tunable parameters yet, although adding some is very high on the TODO. If you have a MySQL source distribution (see [\[Installing source\]](#), page [\[undefined\]](#)), you can exert more control over full-text searching behaviour.

Note that full-text search was carefully tuned for the best searching effectiveness. Modifying the default behaviour will, in most cases, only make the search results worse. Do not alter the MySQL sources unless you know what you are doing!

- The minimum length of words to be indexed is defined by the MySQL variable `ft_min_word_len`. See [\[ft_min_word_len\]](#), page [\[undefined\]](#). Change it to the value you prefer, and rebuild your `FULLTEXT` indexes. (This variable is only available from MySQL version 4.0.)
- The stopword list can be loaded from file specified by `ft_stopword_file` variable. See [\[ft_stopword_file\]](#), page [\[undefined\]](#). Rebuild your `FULLTEXT` indexes after modifying the stopword list. (This variable is only available from MySQL version 4.0.10 and onwards)
- The 50% threshold is determined by the particular weighting scheme chosen. To disable it, change the following line in `'myisam/ftdefs.h'`:

```
#define GWS_IN_USE GWS_PROB
```

To:

```
#define GWS_IN_USE GWS_FREQ
```

Then recompile MySQL. There is no need to rebuild the indexes in this case. **Note:** by doing this you **severely** decrease MySQL’s ability to provide adequate relevance values for the `MATCH()` function. If you really need to search for such common words, it would be better to search using `IN BOOLEAN MODE` instead, which does not observe the 50% threshold.

- Sometimes the search engine maintainer would like to change the operators used for boolean full-text searches. These are defined by the `ft_boolean_syntax` variable. See [\[ft_boolean_syntax\]](#), page [\[undefined\]](#). Still, this variable is read-only, its value is set in `'myisam/ft_static.c'`.

For those changes that require you to rebuild your `FULLTEXT` indexes, the easiest way to do so for a MyISAM table is to use the following statement, which rebuilds the index file:

```
mysql> REPAIR TABLE nome_tabela QUICK;
```

6.8.3 Full-text Search TODO

- Make all operations with FULLTEXT index **faster**.
- Proximity operators
- Support for "always-index words". They could be any strings the user wants to treat as words, examples are "C++", "AS/400", "TCP/IP", etc.
- Support for full-text search in MERGE tables.
- Support for multi-byte charsets.
- Make stopword list to depend of the language of the data.
- Stemming (dependent of the language of the data, of course).
- Generic user-suppliable UDF parser.
- Make the model more flexible (by adding some adjustable parameters to FULLTEXT in CREATE/ALTER TABLE).

6.9 MySQL Query Cache

From version 4.0.1, MySQL server features a **Query Cache**. When in use, the query cache stores the text of a **SELECT** query together with the corresponding result that was sent to the client. If an identical query is later received, the server will retrieve the results from the query cache rather than parsing and executing the same query again.

NOTE: The query cache does not return stale data. When data is modified, any relevant entries in the query cache are flushed.

The query cache is extremely useful in an environment where (some) tables don't change very often and you have a lot of identical queries. This is a typical situation for many web servers that use a lot of dynamic content.

Below is some performance data for the query cache. (These results were generated by running the MySQL benchmark suite on a Linux Alpha 2 x 500 MHz with 2 GB RAM and a 64 MB query cache):

- If all of the queries you're performing are simple (such as selecting a row from a table with one row); but still differ so that the queries can not be cached, the overhead for having the query cache active is 13%. This could be regarded as the worst case scenario. However, in real life, queries are much more complicated than our simple example so the overhead is normally significantly lower.
- Searches after one row in a one row table is 238% faster. This can be regarded as close to the minimum speedup to be expected for a query that is cached.
- If you want to disable the query cache code set `query_cache_size=0`. By disabling the query cache code there is no noticeable overhead. (query cache can be excluded from code with help of configure option `--without-query-cache`)

6.9.1 How The Query Cache Operates

Queries are compared before parsing, thus

```
SELECT * FROM nome_tabela
```

and

```
Select * from nome_tabela
```

are regarded as different queries for query cache, so queries need to be exactly the same (byte for byte) to be seen as identical. In addition, a query may be seen as different if for instance one client is using a new communication protocol format or another character set than another client.

Queries that uses different databases, uses different protocol versions or the uses different default character sets are considered different queries and cached separately.

The cache does work for `SELECT CALC_ROWS ...` and `SELECT FOUND_ROWS() ...` type queries because the number of found rows is also stored in the cache.

If query result was returned from query cache then status variable `Com_select` will not be increased, but `Qcache_hits` will be. See [\(undefined\) \[Query Cache Status and Maintenance\]](#), page [\(undefined\)](#).

If a table changes (`INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, `ALTER` or `DROP TABLE|DATABASE`), then all cached queries that used this table (possibly through a `MRG_MyISAM` table!) become invalid and are removed from the cache.

Transactional InnoDB tables that have been changed will be invalidated when a `COMMIT` is performed.

A query cannot be cached if it contains one of the functions:

Function	Function	Function
User-Defined Functions	<code>CONNECTION_ID</code>	<code>FOUND_ROWS</code>
<code>GET_LOCK</code>	<code>RELEASE_LOCK</code>	<code>LOAD_FILE</code>
<code>MASTER_POS_WAIT</code>	<code>NOW</code>	<code>SYSDATE</code>
<code>CURRENT_TIMESTAMP</code>	<code>CURDATE</code>	<code>CURRENT_DATE</code>
<code>CURTIME</code>	<code>CURRENT_TIME</code>	<code>DATABASE</code>
<code>ENCRYPT</code> (with one parameter)	<code>LAST_INSERT_ID</code>	<code>RAND</code>
<code>UNIX_TIMESTAMP</code> (without parameters)	<code>USER</code>	<code>BENCHMARK</code>

Nor can a query be cached if it contains user variables, references the `mysql` system database, is of the form `SELECT ... IN SHARE MODE`, `SELECT ... INTO OUTFILE ...`, `SELECT ... INTO DUMPFILE ...` or of the form `SELECT * FROM AUTOINCREMENT_FIELD IS NULL` (to retrieve last insert id - ODBC work around).

However, `FOUND_ROWS()` will return the correct value, even if the preceding query was fetched from the cache.

In case a query does not use any tables, or uses temporary tables, or if the user has a column privilege for any of the involved tables, that query will not be cached.

Before a query is fetched from the query cache, MySQL will check that the user has `SELECT` privilege to all the involved databases and tables. If this is not the case, the cached result will not be used.

6.9.2 Query Cache Configuration

The query cache adds a few MySQL system variables for `mysqld` which may be set in a configuration file, on the command-line when starting `mysqld`.

`query_cache_limit` Don't cache results that are bigger than this. (Default 1M).

`query_cache_min_res_unit`

This variable is present from version 4.1.

The result of a query (the data that is also sent to the client) is stored in the query cache during result retrieval. Therefore the data is usually not handled in one big chunk. The query cache allocates blocks for storing this data on demand, so when one block is filled, a new block is allocated. Because memory allocation operation is costly (time wise), the query cache allocates blocks with a minimum size of `query_cache_min_res_unit`. When a query is executed, the last result block is trimmed to the actual data size, so that unused memory is freed.

- The default value of `query_cache_min_res_unit` is 4 KB which should be adequate for most cases.
- If you have a lot of queries with small results, the default block size may lead to memory fragmentation (indicated by a large number of free blocks (`Qcache_free_blocks`), which can cause the query cache to have to delete queries from the cache due to lack of memory (`Qcache_lowmem_prunes`)). In this case you should decrease `query_cache_min_res_unit`.
- If you mostly have queries with big results (see `Qcache_total_blocks` and `Qcache_queries_in_cache`), you can increase performance by increasing `query_cache_min_res_unit`. However, be careful to not make it too large (see the previous point).

`query_cache_size` The amount of memory (specified in bytes) allocated to store results from old queries. If this is 0, the query cache is disabled (default).

`query_cache_type` This may be set (only numeric) to

Option	Description
0	(OFF, don't cache or retrieve results)
1	(ON, cache all results except <code>SELECT SQL_NO_CACHE ...</code> queries)
2	(DEMAND, cache only <code>SELECT SQL_CACHE ...</code> queries)

Inside a thread (connection), the behaviour of the query cache can be changed from the default. The syntax is as follows:

`QUERY_CACHE_TYPE = OFF | ON | DEMAND` `QUERY_CACHE_TYPE = 0 | 1 | 2`

Option	Description
0 or OFF	Don't cache or retrieve results.
1 or ON	Cache all results except <code>SELECT SQL_NO_CACHE ...</code> queries.
2 or DEMAND	Cache only <code>SELECT SQL_CACHE ...</code> queries.

6.9.3 Query Cache Options in SELECT

There are two possible query cache related parameters that may be specified in a `SELECT` query:

Option	Description
<code>SQL_CACHE</code>	If <code>QUERY_CACHE_TYPE</code> is <code>DEMAND</code> , allow the query to be cached. If <code>QUERY_CACHE_TYPE</code> is <code>ON</code> , this is the default. If <code>QUERY_CACHE_TYPE</code> is <code>OFF</code> , do nothing.

SQL_NO_CACHE Make this query non-cachable, don't allow this query to be stored in the cache.

6.9.4 Query Cache Status and Maintenance

With the **FLUSH QUERY CACHE** command you can defragment the query cache to better utilise its memory. This command will not remove any queries from the cache. **FLUSH TABLES** also flushes the query cache.

The **RESET QUERY CACHE** command removes all query results from the query cache.

You can check whether the query cache is present in your MySQL version:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_query_cache | YES |
+-----+-----+
1 row in set (0.00 sec)
```

You can monitor query cache performance in **SHOW STATUS**:

Variable	Description
Qcache_queries_in_cache	Number of queries registered in the cache.
Qcache_inserts	Number of queries added to the cache.
Qcache_hits	Number of cache hits.
Qcache_lowmem_prunes	Number of queries that were deleted from cache because of low memory.
Qcache_not_cached	Number of non-cached queries (not cachable, or due to QUERY_CACHE_TYPE).
Qcache_free_memory	Amount of free memory for query cache.
Qcache_free_blocks	Number of free memory blocks in query cache.
Qcache_total_blocks	Total number of blocks in query cache.

Total number of queries = **Qcache_inserts** + **Qcache_hits** + **Qcache_not_cached**.

The query cache uses variable length blocks, so **Qcache_total_blocks** and **Qcache_free_blocks** may indicate query cache memory fragmentation. After **FLUSH QUERY CACHE** only a single (big) free block remains.

Note: Every query needs a minimum of two blocks (one for the query text and one or more for the query results). Also, every table that is used by a query needs one block, but if two or more queries use same table only one block needs to be allocated.

You can use the **Qcache_lowmem_prunes** status variable to tune the query cache size. It counts the number of queries that have been removed from the cache to free up memory for caching new queries. The query cache uses a **least recently used (LRU)** strategy to decide which queries to remove from the cache.

7 MySQL Table Types

As of MySQL Version 3.23.6, you can choose between three basic table formats (**ISAM**, **HEAP** and **MyISAM**). Newer versions of MySQL support additional table types (**InnoDB**, or **BDB**), depending on how you compile it.

When you create a new table, you can tell MySQL what type of table to create. The default table type is usually **MyISAM**.

MySQL will always create a `.frm` file to hold the table and column definitions. The table's index and data will be stored in one or more other files, depending on the table type.

If you try to use a table type that is not compiled-in or activated, MySQL will instead create a table of type **MyISAM**. This behaviour is convenient when you want to copy tables between MySQL servers that support different table types. (Perhaps your master server supports transactional storage engines for increased safety, while the slave servers use only non-transactional storage engines for greater speed.)

This automatic change of table types can be confusing for new MySQL users. We plan to fix this by introducing warnings in the new client-server protocol in version 4.1 and generating a warning when a table type is automatically changed.

You can convert tables between different types with the **ALTER TABLE** statement. See [\(undefined\)](#) [**ALTER TABLE**], page [\(undefined\)](#).

Note that MySQL supports two different kinds of tables: transaction-safe tables (**InnoDB** and **BDB**) and not transaction-safe tables (**HEAP**, **ISAM**, **MERGE**, and **MyISAM**).

Advantages of transaction-safe tables (TST):

- Safer. Even if MySQL crashes or you get hardware problems, you can get your data back, either by automatic recovery or from a backup + the transaction log.
- You can combine many statements and accept these all in one go with the **COMMIT** command.
- You can execute **ROLLBACK** to ignore your changes (if you are not running in auto-commit mode).
- If an update fails, all your changes will be restored. (With **NTST** tables all changes that have taken place are permanent)
- Can provide better concurrency if the table gets many updates concurrently with reads.

Note that to use **InnoDB** tables you have to use at least the `innodb_data_file_path` startup option. See [\(undefined\)](#) [**InnoDB start**], page [\(undefined\)](#).

Advantages of not transaction-safe tables (NTST):

- Much faster as there is no transaction overhead.
- Will use less disk space as there is no overhead of transactions.
- Will use less memory to do updates.

You can combine **TST** and **NTST** tables in the same statements to get the best of both worlds.

7.1 MyISAM Tables

MyISAM is the default table type in MySQL Version 3.23. It's based on the ISAM code and has a lot of useful extensions.

The index is stored in a file with the '.MYI' (MYIndex) extension, and the data is stored in a file with the '.MYD' (MYData) extension. You can check/repair MyISAM tables with the `myisamchk` utility. See [\[Crash recovery\]](#), page [\[undefined\]](#). You can compress MyISAM tables with `myisampack` to take up much less space. See [\[myisampack\]](#), page [\[undefined\]](#).

The following is new in MyISAM:

- There is a flag in the MyISAM file that indicates whether the table was closed correctly. If `mysqld` is started with `--myisam-recover`, MyISAM tables will automatically be checked and/or repaired on open if the table wasn't closed properly.
- You can INSERT new rows in a table that doesn't have free blocks in the middle of the datafile, at the same time other threads are reading from the table (concurrent insert). A free block can come from an update of a dynamic length row with much data to a row with less data or when deleting rows. When all free blocks are used up, all future inserts will be concurrent again.
- Support for big files (63-bit) on filesystems/operating systems that support big files.
- All data is stored with the low byte first. This makes the data machine and OS independent. The only requirement is that the machine uses two's-complement signed integers (as every machine for the last 20 years has) and IEEE floating-point format (also totally dominant among mainstream machines). The only area of machines that may not support binary compatibility are embedded systems (because they sometimes have peculiar processors).

There is no big speed penalty in storing data low byte first; the bytes in a table row is normally unaligned and it doesn't take that much more power to read an unaligned byte in order than in reverse order. The actual fetch-column-value code is also not time critical compared to other code.

- All number keys are stored with high byte first to give better index compression.
- Internal handling of one `AUTO_INCREMENT` column. MyISAM will automatically update this on `INSERT/UPDATE`. The `AUTO_INCREMENT` value can be reset with `myisamchk`. This will make `AUTO_INCREMENT` columns faster (at least 10%) and old numbers will not be reused as with the old ISAM. Note that when an `AUTO_INCREMENT` is defined on the end of a multi-part-key the old behaviour is still present.
- When inserted in sorted order (as when you are using an `AUTO_INCREMENT` column) the key tree will be split so that the high node only contains one key. This will improve the space utilisation in the key tree.
- BLOB and TEXT columns can be indexed.
- NULL values are allowed in indexed columns. This takes 0-1 bytes/key.
- Maximum key length is 500 bytes by default (can be changed by recompiling). In cases of keys longer than 250 bytes, a bigger key block size than the default of 1024 bytes is used for this key.
- Maximum number of keys/table is 32 as default. This can be enlarged to 64 without having to recompile `myisamchk`.

- `myisamchk` will mark tables as checked if one runs it with `--update-state`. `myisamchk --fast` will only check those tables that don't have this mark.
- `myisamchk -a` stores statistics for key parts (and not only for whole keys as in ISAM).
- Dynamic size rows will now be much less fragmented when mixing deletes with updates and inserts. This is done by automatically combining adjacent deleted blocks and by extending blocks if the next block is deleted.
- `myisampack` can pack BLOB and VARCHAR columns.
- You can use put the datafile and index file on different directories to get more speed (with the `DATA/INDEX DIRECTORY="path"` option to `CREATE TABLE`). See [\[CREATE TABLE\]](#), page [\[undefined\]](#).

MyISAM also supports the following things, which MySQL will be able to use in the near future:

- Support for a true VARCHAR type; a VARCHAR column starts with a length stored in 2 bytes.
- Tables with VARCHAR may have fixed or dynamic record length.
- VARCHAR and CHAR may be up to 64K. All key segments have their own language definition. This will enable MySQL to have different language definitions per column.
- A hashed computed index can be used for UNIQUE. This will allow you to have UNIQUE on any combination of columns in a table. (You can't search on a UNIQUE computed index, however.)

Note that index files are usually much smaller with MyISAM than with ISAM. This means that MyISAM will normally use less system resources than ISAM, but will need more CPU time when inserting data into a compressed index.

The following options to `mysqld` can be used to change the behaviour of MyISAM tables. See [\[SHOW VARIABLES\]](#), page [\[undefined\]](#).

Option	Description
<code>--myisam-recover=#</code>	Automatic recovery of crashed tables.
<code>-O myisam_sort_buffer_size=#</code>	Buffer used when recovering tables.
<code>--delay-key-write=ALL</code>	Don't flush key buffers between writes for any MyISAM table
<code>-O myisam_max_extra_sort_file_size=#</code>	Used to help MySQL to decide when to use the slow but safe key cache index create method. Note that this parameter is given in megabytes before 4.0.3 and in bytes beginning with this version.
<code>-O myisam_max_sort_file_size=#</code>	Don't use the fast sort index method to created index if the temporary file would get bigger than this. Note that this parameter is given in megabytes before 4.0.3 and in bytes beginning with this version.
<code>-O bulk_insert_buffer_size=#</code>	Size of tree cache used in bulk insert optimisation. Note that this is a limit per thread!

The automatic recovery is activated if you start `mysqld` with `--myisam-recover=#`. See [line options-snt \[Command-line options\]](#), page [line options-pg](#). On open, the table is checked if it's marked as crashed or if the open count variable for the table is not 0 and you are running with `--skip-external-locking`. If either of the above is true the following happens.

- The table is checked for errors.
- If we found an error, try to do a fast repair (with sorting and without re-creating the datafile) of the table.
- If the repair fails because of an error in the datafile (for example a duplicate key error), we try again, but this time we re-create the datafile.
- If the repair fails, retry once more with the old repair option method (write row by row without sorting) which should be able to repair any type of error with little disk requirements..

If the recover wouldn't be able to recover all rows from a previous completed statement and you didn't specify **FORCE** as an option to `myisam-recover`, then the automatic repair will abort with an error message in the error file:

```
Error: Couldn't repair table: test.g00pages
```

If you in this case had used the **FORCE** option you would instead have got a warning in the error file:

```
Warning: Found 344 of 354 rows when repairing ./test/g00pages
```

Note that if you run automatic recover with the **BACKUP** option, you should have a cron script that automatically moves file with names like 'tablename-datetime.BAK' from the database directories to a backup media.

See [line options-snt \[Command-line options\]](#), page [line options-pg](#).

7.1.1 Space Needed for Keys

MySQL can support different index types, but the normal type is ISAM or MyISAM. These use a B-tree index, and you can roughly calculate the size for the index file as $(key_length+4)/0.67$, summed over all keys. (This is for the worst case when all keys are inserted in sorted order and we don't have any compressed keys.)

String indexes are space compressed. If the first index part is a string, it will also be prefix compressed. Space compression makes the index file smaller than the above figures if the string column has a lot of trailing space or is a **VARCHAR** column that is not always used to the full length. Prefix compression is used on keys that start with a string. Prefix compression helps if there are many strings with an identical prefix.

In **MyISAM** tables, you can also prefix compress numbers by specifying **PACK_KEYS=1** when you create the table. This helps when you have many integer keys that have an identical prefix when the numbers are stored high-byte first.

7.1.2 MyISAM Table Formats

MyISAM supports 3 different table types. Two of them are chosen automatically depending on the type of columns you are using. The third, compressed tables, can only be created with the `myisampack` tool.

When you **CREATE** or **ALTER** a table you can for tables that doesn't have **BLOBs** force the table format to **DYNAMIC** or **FIXED** with the `ROW_FORMAT=#` table option. In the future you will be able to compress/decompress tables by specifying `ROW_FORMAT=compressed | default` to **ALTER TABLE**. See [\(undefined\) \[CREATE TABLE\]](#), page [\(undefined\)](#).

7.1.2.1 Static (Fixed-length) Table Characteristics

This is the default format. It's used when the table contains no `VARCHAR`, `BLOB`, or `TEXT` columns.

This format is the simplest and most secure format. It is also the fastest of the on-disk formats. The speed comes from the easy way data can be found on disk. When looking up something with an index and static format it is very simple. Just multiply the row number by the row length.

Also, when scanning a table it is very easy to read a constant number of records with each disk read.

The security is evidenced if your computer crashes when writing to a fixed-size MyISAM file, in which case `myisamchk` can easily figure out where each row starts and ends. So it can usually reclaim all records except the partially written one. Note that in MySQL all indexes can always be reconstructed:

- All `CHAR`, `NUMERIC`, and `DECIMAL` columns are space-padded to the column width.
- Very quick.
- Easy to cache.
- Easy to reconstruct after a crash, because records are located in fixed positions.
- Doesn't have to be reorganised (with `myisamchk`) unless a huge number of records are deleted and you want to return free disk space to the operating system.
- Usually requires more disk space than dynamic tables.

7.1.2.2 Dynamic Table Characteristics

This format is used if the table contains any `VARCHAR`, `BLOB`, or `TEXT` columns or if the table was created with `ROW_FORMAT=dynamic`.

This format is a little more complex because each row has to have a header that says how long it is. One record can also end up at more than one location when it is made longer at an update.

You can use `OPTIMIZE table` or `myisamchk` to defragment a table. If you have static data that you access/change a lot in the same table as some `VARCHAR` or `BLOB` columns, it might be a good idea to move the dynamic columns to other tables just to avoid fragmentation:

- All string columns are dynamic (except those with a length less than 4).
- Each record is preceded by a bitmap indicating which columns are empty (') for string columns, or zero for numeric columns. (This isn't the same as columns containing `NULL` values.) If a string column has a length of zero after removal of trailing spaces, or a numeric column has a value of zero, it is marked in the bit map and not saved to disk. Non-empty strings are saved as a length byte plus the string contents.
- Usually takes much less disk space than fixed-length tables.
- Each record uses only as much space as is required. If a record becomes larger, it is split into as many pieces as are required. This results in record fragmentation.
- If you update a row with information that extends the row length, the row will be fragmented. In this case, you may have to run `myisamchk -r` from time to time to get better performance. Use `myisamchk -ei nome_tabela` for some statistics.

- Not as easy to reconstruct after a crash, because a record may be fragmented into many pieces and a link (fragment) may be missing.
- The expected row length for dynamic sized records is:

```

3
+ (number of columns + 7) / 8
+ (number of char columns)
+ packed size of numeric columns
+ length of strings
+ (number of NULL columns + 7) / 8

```

There is a penalty of 6 bytes for each link. A dynamic record is linked whenever an update causes an enlargement of the record. Each new link will be at least 20 bytes, so the next enlargement will probably go in the same link. If not, there will be another link. You may check how many links there are with `myisamchk -ed`. All links may be removed with `myisamchk -r`.

7.1.2.3 Compressed Table Characteristics

This is a read-only type that is generated with the optional `myisampack` tool (`pack_isam` for ISAM tables):

- All MySQL distributions, even those that existed before MySQL went GPL, can read tables that were compressed with `myisampack`.
- Compressed tables take very little disk space. This minimises disk usage, which is very nice when using slow disks (like CD-ROMs).
- Each record is compressed separately (very little access overhead). The header for a record is fixed (1-3 bytes) depending on the biggest record in the table. Each column is compressed differently. Some of the compression types are:
 - There is usually a different Huffman table for each column.
 - Suffix space compression.
 - Prefix space compression.
 - Numbers with value 0 are stored using 1 bit.
 - If values in an integer column have a small range, the column is stored using the smallest possible type. For example, a `BIGINT` column (8 bytes) may be stored as a `TINYINT` column (1 byte) if all values are in the range 0 to 255.
 - If a column has only a small set of possible values, the column type is converted to `ENUM`.
 - A column may use a combination of the above compressions.
- Can handle fixed- or dynamic-length records.
- Can be uncompressed with `myisamchk`.

7.1.3 MyISAM Table Problems

The file format that MySQL uses to store data has been extensively tested, but there are always circumstances that may cause database tables to become corrupted.

7.1.3.1 Corrupted MyISAM Tables

Even if the MyISAM table format is very reliable (all changes to a table is written before the SQL statements returns) , you can still get corrupted tables if some of the following things happens:

- The `mysqld` process being killed in the middle of a write.
- Unexpected shutdown of the computer (for example, if the computer is turned off).
- A hardware error.
- You are using an external program (like `myisamchk`) on a live table.
- A software bug in the MySQL or MyISAM code.

Typical typical symptoms for a corrupt table is:

- You get the error `Incorrect key file for table: '...'`. Try to repair it while selecting data from the table.
- Queries doesn't find rows in the table or returns incomplete data.

You can check if a table is ok with the command `CHECK TABLE`. See [\[CHECK TABLE\]](#), page [\[undefined\]](#).

You can repair a corrupted table with `REPAIR TABLE`. See [\[REPAIR TABLE\]](#), page [\[undefined\]](#). You can also repair a table, when `mysqld` is not running with the `myisamchk` command. `myisamchk` syntax.

If your tables get corrupted a lot you should try to find the reason for this! See [\[Crashing\]](#), page [\[undefined\]](#).

In this case the most important thing to know is if the table got corrupted if the `mysqld` died (one can easily verify this by checking if there is a recent row `restarted mysqld` in the `mysqld` error file). If this isn't the case, then you should try to make a test case of this. See [\[Reproducible test case\]](#), page [\[undefined\]](#).

7.1.3.2 Clients is using or hasn't closed the table properly

Each MyISAM `.MYI` file has in the header a counter that can be used to check if a table has been closed properly.

If you get the following warning from `CHECK TABLE` or `myisamchk`:

```
# clients is using or hasn't closed the table properly
```

this means that this counter has come out of sync. This doesn't mean that the table is corrupted, but means that you should at least do a check on the table to verify that it's okay.

The counter works as follows:

- The first time a table is updated in MySQL, a counter in the header of the index files is incremented.
- The counter is not changed during further updates.
- When the last instance of a table is closed (because of a `FLUSH` or because there isn't room in the table cache) the counter is decremented if the table has been updated at any point.
- When you repair the table or check the table and it was okay, the counter is reset to 0.

- To avoid problems with interaction with other processes that may do a check on the table, the counter is not decremented on close if it was 0.

In other words, the only ways this can go out of sync are:

- The MyISAM tables are copied without a LOCK and FLUSH TABLES.
- MySQL has crashed between an update and the final close. (Note that the table may still be okay, as MySQL always issues writes for everything between each statement.)
- Someone has done a `myisamchk --recover` or `myisamchk --update-state` on a table that was in use by `mysqld`.
- Many `mysqld` servers are using the table and one has done a REPAIR or CHECK of the table while it was in use by another server. In this setup the CHECK is safe to do (even if you will get the warning from other servers), but REPAIR should be avoided as it currently replaces the datafile with a new one, which is not signaled to the other servers.

7.2 MERGE Tables

MERGE tables are new in MySQL Version 3.23.25. The code is still in gamma, but should be reasonable stable.

A MERGE table (also known as a MRG_MyISAM table) is a collection of identical MyISAM tables that can be used as one. You can only SELECT, DELETE, and UPDATE from the collection of tables. If you DROP the MERGE table, you are only dropping the MERGE specification.

Note that DELETE FROM `merge_table` used without a WHERE will only clear the mapping for the table, not delete everything in the mapped tables. (We plan to fix this in 4.1).

With identical tables we mean that all tables are created with identical column and key information. You can't merge tables in which the columns are packed differently, doesn't have exactly the same columns, or have the keys in different order. However, some of the tables can be compressed with `myisampack`. See [\(undefined\) \[myisampack\], page \(undefined\)](#).

When you create a MERGE table, you will get a `.frm` table definition file and a `.MRG` table list file. The `.MRG` just contains a list of the index files (`.MYI` files) that should be used as one. All used tables must be in the same database as the MERGE table itself.

For the moment, you need to have SELECT, UPDATE, and DELETE privileges on the tables you map to a MERGE table.

MERGE tables can help you solve the following problems:

- Easily manage a set of log tables. For example, you can put data from different months into separate files, compress some of them with `myisampack`, and then create a MERGE to use these as one.
- Give you more speed. You can split a big read-only table based on some criteria and then put the different table part on different disks. A MERGE table on this could be much faster than using the big table. (You can, of course, also use a RAID to get the same kind of benefits.)
- Do more efficient searches. If you know exactly what you are looking after, you can search in just one of the split tables for some queries and use a MERGE table for others. You can even have many different MERGE tables active, with possible overlapping files.

- More efficient repairs. It's easier to repair the individual files that are mapped to a `MERGE` file than trying to repair a really big file.
- Instant mapping of many files as one. A `MERGE` table uses the index of the individual tables. It doesn't need to maintain an index of its own. This makes `MERGE` table collections **VERY** fast to make or remap. Note that you must specify the key definitions when you create a `MERGE` table!
- If you have a set of tables that you join to a big table on demand or batch, you should instead create a `MERGE` table on them on demand. This is much faster and will save a lot of disk space.
- Go around the file-size limit for the operating system.
- You can create an alias/synonym for a table by just using `MERGE` over one table. There shouldn't be any really notable performance impacts of doing this (only a couple of indirect calls and `memcpy()` calls for each read).

The disadvantages with `MERGE` tables are:

- You can only use identical `MyISAM` tables for a `MERGE` table.
- `REPLACE` doesn't work.
- `MERGE` tables use more file descriptors. If you are using a `MERGE` table that maps over 10 tables and 10 users are using this, you are using $10 \times 10 + 10$ file descriptors. (10 datafiles for 10 users and 10 shared index files.)
- Key reads are slower. When you do a read on a key, the `MERGE` storage engine will need to issue a read on all underlying tables to check which one most closely matches the given key. If you then do a "read-next" then the `MERGE` storage engine will need to search the read buffers to find the next key. Only when one key buffer is used up, the storage engine will need to read the next key block. This makes `MERGE` keys much slower on `eq_ref` searches, but not much slower on `ref` searches. See [\[EXPLAIN\]](#), page [\[undefined\]](#).
- You can't do `DROP TABLE`, `ALTER TABLE`, `DELETE FROM table_name` without a `WHERE` clause, `REPAIR TABLE`, `TRUNCATE TABLE`, `OPTIMIZE TABLE`, or `ANALYZE TABLE` on any of the table that is mapped by a `MERGE` table that is "open". If you do this, the `MERGE` table may still refer to the original table and you will get unexpected results. The easiest way to get around this deficiency is to issue the `FLUSH TABLES` command, ensuring no `MERGE` tables remain "open".

When you create a `MERGE` table you have to specify with `UNION(list-of-tables)` which tables you want to use as one. Optionally you can specify with `INSERT_METHOD` if you want insert for the `MERGE` table to happen in the first or last table in the `UNION` list. If you don't specify `INSERT_METHOD` or specify `NO`, then all `INSERT` commands on the `MERGE` table will return an error.

The following example shows you how to use `MERGE` tables:

```
CREATE TABLE t1 (a INT AUTO_INCREMENT PRIMARY KEY, message CHAR(20));
CREATE TABLE t2 (a INT AUTO_INCREMENT PRIMARY KEY, message CHAR(20));
INSERT INTO t1 (message) VALUES ("Testing"),("table"),("t1");
INSERT INTO t2 (message) VALUES ("Testing"),("table"),("t2");
CREATE TABLE total (a INT AUTO_INCREMENT PRIMARY KEY, message CHAR(20))
    TYPE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;
```

Note that you can also manipulate the '.MRG' file directly from the outside of the MySQL server:

```
shell> cd /mysql-data-directory/current-database
shell> ls -l t1.MYI t2.MYI > total.MRG
shell> mysqladmin flush-tables
```

Now you can do things like:

```
mysql> SELECT * FROM total;
+----+-----+
| a | message |
+----+-----+
| 1 | Testing |
| 2 | table   |
| 3 | t1      |
| 1 | Testing |
| 2 | table   |
| 3 | t2      |
+----+-----+
```

Note that the a column, though declared as PRIMARY KEY, is not really unique, as MERGE table cannot enforce uniqueness over a set of underlying MyISAM tables.

To remap a MERGE table you can do one of the following:

- DROP the table and re-create it
- Use ALTER TABLE table_name UNION(...)
- Change the '.MRG' file and issue a FLUSH TABLE on the MERGE table and all underlying tables to force the storage engine to read the new definition file.

7.2.1 MERGE Table Problems

The following are the known problems with MERGE tables:

- A MERGE table cannot maintain UNIQUE constraints over the whole table. When you do INSERT, the data goes into the first or last table (according to INSERT_METHOD=xxx) and this MyISAM table ensures that the data are unique, but it knows nothing about others MyISAM tables.
- DELETE FROM merge_table used without a WHERE will only clear the mapping for the table, not delete everything in the mapped tables.
- RENAME TABLE on a table used in an active MERGE table may corrupt the table. This will be fixed in MySQL 4.0.x.
- Creation of a table of type MERGE doesn't check if the underlying tables are of compatible types. If you use MERGE tables in this fashion, you are very likely to run into strange problems.
- If you use ALTER TABLE to first add an UNIQUE index to a table used in a MERGE table and then use ALTER TABLE to add a normal index on the MERGE table, the key order will be different for the tables if there was an old non-unique key in the table. This is because ALTER TABLE puts UNIQUE keys before normal keys to be able to detect duplicate keys as early as possible.

- The range optimiser can't yet use `MERGE` table efficiently and may sometimes produce non-optimal joins. This will be fixed in MySQL 4.0.x.
- `DROP TABLE` on a table that is in use by a `MERGE` table will not work on Windows because the `MERGE` storage engine does the table mapping hidden from the upper layer of MySQL. Because Windows doesn't allow you to drop files that are open, you first must flush all `MERGE` tables (with `FLUSH TABLES`) or drop the `MERGE` table before dropping the table. We will fix this at the same time we introduce `VIEWS`.

7.3 ISAM Tables

The deprecated `ISAM` table type will disappear in MySQL version 5.0. `MyISAM` is a better implementation of the same thing.

`ISAM` uses a `B-tree` index. The index is stored in a file with the `'.ISM'` extension, and the data is stored in a file with the `'.ISD'` extension. You can check/repair `ISAM` tables with the `isamchk` utility. See [\[Crash recovery\]](#), page [\(undefined\)](#).

`ISAM` has the following features/properties:

- Compressed and fixed-length keys
- Fixed and dynamic record length
- 16 keys with 16 key parts/key
- Max key length 256 (default)
- Data is stored in machine format; this is fast, but is machine/OS dependent.

Most of the things true for `MyISAM` tables are also true for `ISAM` tables. See [\[MyISAM tables\]](#), page [\(undefined\)](#). The major differences compared to `MyISAM` tables are:

- `ISAM` tables are not binary portable across OS/Platforms.
- Can't handle tables > 4G.
- Only support prefix compression on strings.
- Smaller key limits.
- Dynamic tables get more fragmented.
- Tables are compressed with `pack_isam` rather than with `myisampack`.

If you want to convert an `ISAM` table to a `MyISAM` table so that you can use utilities such as `mysqlcheck`, use an `ALTER TABLE` statement:

```
mysql> ALTER TABLE nome_tabela TYPE = MYISAM;
```

The embedded MySQL versions doesn't support `ISAM` tables.

7.4 HEAP Tables

`HEAP` tables use hashed indexes and are stored in memory. This makes them very fast, but if MySQL crashes you will lose all data stored in them. `HEAP` is very useful for temporary tables!

The MySQL internal `HEAP` tables use 100% dynamic hashing without overflow areas. There is no extra space needed for free lists. `HEAP` tables also don't have problems with delete + inserts, which normally is common with hashed tables:

```
mysql> CREATE TABLE test TYPE=HEAP SELECT ip,SUM(downloads) AS down
->                                FROM log_table GROUP BY ip;
mysql> SELECT COUNT(ip),AVG(down) FROM test;
mysql> DROP TABLE test;
```

Here are some things you should consider when you use HEAP tables:

- You should always use specify `MAX_ROWS` in the `CREATE` statement to ensure that you accidentally do not use all memory.
- Indexes will only be used with `=` and `<=>` (but are VERY fast).
- HEAP tables can only use whole keys to search for a row; compare this to MyISAM tables where any prefix of the key can be used to find rows.
- HEAP tables use a fixed record length format.
- HEAP doesn't support BLOB/TEXT columns.
- HEAP doesn't support AUTO_INCREMENT columns.
- Prior to MySQL 4.0.2, HEAP doesn't support an index on a NULL column.
- You can have non-unique keys in a HEAP table (this isn't common for hashed tables).
- HEAP tables are shared between all clients (just like any other table).
- You can't search for the next entry in order (that is, to use the index to do an ORDER BY).
- Data for HEAP tables are allocated in small blocks. The tables are 100% dynamic (on inserting). No overflow areas and no extra key space are needed. Deleted rows are put in a linked list and are reused when you insert new data into the table.
- You need enough extra memory for all HEAP tables that you want to use at the same time.
- To free memory, you should execute `DELETE FROM heap_table`, `TRUNCATE heap_table` or `DROP TABLE heap_table`.
- MySQL cannot find out approximately how many rows there are between two values (this is used by the range optimiser to decide which index to use). This may affect some queries if you change a MyISAM table to a HEAP table.
- To ensure that you accidentally don't do anything foolish, you can't create HEAP tables bigger than `max_heap_table_size`.

The memory needed for one row in a HEAP table is:

```
SUM_OVER_ALL_KEYS(max_length_of_key + sizeof(char*) * 2)
+ ALIGN(length_of_row+1, sizeof(char*))
```

`sizeof(char*)` is 4 on 32-bit machines and 8 on 64-bit machines.

7.5 InnoDB Tables

7.5.1 InnoDB Tables Overview

InnoDB provides MySQL with a transaction-safe (ACID compliant) storage engine with commit, rollback, and crash recovery capabilities. InnoDB does locking on row level and also provides an Oracle-style consistent non-locking read in SELECTs. These features increase

multiuser concurrency and performance. There is no need for lock escalation in InnoDB, because row level locks in InnoDB fit in very small space. InnoDB tables support FOREIGN KEY constraints as the first table type in MySQL.

InnoDB has been designed for maximum performance when processing large data volumes. Its CPU efficiency is probably not matched by any other disk-based relational database engine.

InnoDB is used in production at numerous large database sites requiring high performance. The famous Internet news site Slashdot.org runs on InnoDB. Mytrix, Inc. stores over 1 TB of data in InnoDB, and another site handles an average load of 800 inserts/updates per second in InnoDB.

Technically, InnoDB is a complete database backend placed under MySQL. InnoDB has its own buffer pool for caching data and indexes in main memory. InnoDB stores its tables and indexes in a tablespace, which may consist of several files (or raw disk partitions). This is different from, for example, MyISAM tables where each table is stored as a separate file. InnoDB tables can be of any size also on those operating systems where file-size is limited to 2 GB.

You can find the latest information about InnoDB at <http://www.innodb.com/>. The most up-to-date version of the InnoDB manual is always placed there, and you can also order commercial licenses and support for InnoDB.

In the source distribution of MySQL, InnoDB appears as a subdirectory. InnoDB is distributed under the GNU GPL License Version 2 (of June 1991).

7.5.2 InnoDB in MySQL Version 3.23

From MySQL version 4.0, InnoDB is enabled by default. The following information only applies to the 3.23 series.

InnoDB tables are included in the MySQL source distribution starting from 3.23.34a and are activated in the MySQL -Max binary of the 3.23 series. For Windows the -Max binaries are contained in the standard distribution.

If you have downloaded a binary version of MySQL that includes support for InnoDB, simply follow the instructions of the MySQL manual for installing a binary version of MySQL. If you already have MySQL-3.23 installed, then the simplest way to install MySQL -Max is to replace the server executable 'mysqld' with the corresponding executable in the -Max distribution. MySQL and MySQL -Max differ only in the server executable. See [\[Installing binary\]](#), page [\[undefined\]](#). See [max-snt \[mysqld-max\]](#), page [max-pg](#).

To compile MySQL with InnoDB support, download MySQL-3.23.34a or newer version from <http://www.mysql.com/> and configure MySQL with the `--with-innodb` option. See the MySQL manual about installing a MySQL source distribution. See [\[Installing source\]](#), page [\[undefined\]](#).

```
cd /path/to/source/of/mysql-3.23.37
./configure --with-innodb
```

To use InnoDB tables in MySQL-Max-3.23 you **must** specify configuration parameters in the `[mysqld]` section of the configuration file 'my.cnf', or on Windows optionally in 'my.ini'.

At the minimum, in 3.23 you must specify `innodb_data_file_path` where you specify the names and the sizes of datafiles. If you do not mention `innodb_data_home_dir` in 'my.cnf'

the default is to create these files to the `datadir` of MySQL. If you specify `innodb_data_home_dir` as an empty string, then you can give absolute paths to your data files in `innodb_data_file_path`.

The minimal way to modify it is to add to the `[mysqld]` section the line

```
innodb_data_file_path=ibdata:30M
```

but to get good performance it is best that you specify options as recommended. See [\[InnoDB start\]](#), page [\[undefined\]](#).

7.5.3 InnoDB Startup Options

To enable InnoDB tables in MySQL version 3.23, see [\[InnoDB in MySQL 3.23\]](#), page [\[undefined\]](#).

In MySQL-4.0 you do are not required to do anything specific to enable InnoDB tables.

The default behaviour is to create an auto-extending 10 MB file ‘`ibdata1`’ in the `datadir` of MySQL. (In MySQL-4.0.0 and 4.0.1 the datafile is 64 MB and not auto-extending.)

Note: To get good performance you **should** explicitly set the InnoDB parameters listed in the following examples.

If you don’t want to use InnoDB tables, you can add the `skip-innodb` option to your MySQL option file.

Starting from versions 3.23.50 and 4.0.2 InnoDB allows the last datafile on the `innodb_data_file_path` line to be specified as **auto-extending**. The syntax for `innodb_data_file_path` is then the following:

```
pathtodatafile:sizespecification;pathtodatafile:sizespecification;...
... ;pathtodatafile:sizespecification[:autoextend[:max:sizespecification]]
```

If you specify the last datafile with the `autoextend` option, InnoDB will extend the last datafile if it runs out of free space in the tablespace. The increment is 8 MB at a time. An example:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:100M:autoextend
```

instructs InnoDB to create just a single datafile whose initial size is 100 MB and which is extended in 8 MB blocks when space runs out. If the disk becomes full you may want to add another data file to another disk, for example. Then you have to look the size of ‘`ibdata1`’, round the size downward to the closest multiple of $1024 * 1024$ bytes (= 1 MB), and specify the rounded size of ‘`ibdata1`’ explicitly in `innodb_data_file_path`. After that you can add another datafile:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:988M;/disk2/ibdata2:50M:autoextend
```

Be cautious on filesystems where the maximum file-size is 2 GB. InnoDB is not aware of the OS maximum file-size. On those filesystems you might want to specify the max size for the datafile:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:100M:autoextend:max:2000M
```

A simple ‘my.cnf’ example. Suppose you have a computer with 128 MB RAM and one hard disk. Below is an example of possible configuration parameters in ‘`my.cnf`’ or ‘`my.ini`’ for

InnoDB. We assume you are running MySQL-Max-3.23.50 or later, or MySQL-4.0.2 or later. This example suits most users, both on Unix and Windows, who do not want to distribute InnoDB datafiles and log files on several disks. This creates an auto-extending data file 'ibdata1' and two InnoDB log files 'ib_logfile0' and 'ib_logfile1' to the `datadir` of MySQL (typically `/mysql/data`). Also the small archived InnoDB log file 'ib_arch_log_0000000000' ends up in the `datadir`.

```
[mysqld]
# You can write your other MySQL server options here
# ...
#                               Data file(s) must be able to
#                               hold your data and indexes.
#                               Make sure you have enough
#                               free disk space.
innodb_data_file_path = ibdata1:10M:autoextend
#                               Set buffer pool size to
#                               50 - 80 % of your computer's
#                               memory
set-variable = innodb_buffer_pool_size=70M
set-variable = innodb_additional_mem_pool_size=10M
#                               Set the log file size to about
#                               25 % of the buffer pool size
set-variable = innodb_log_file_size=20M
set-variable = innodb_log_buffer_size=8M
#                               Set ..flush_log_at_trx_commit
#                               to 0 if you can afford losing
#                               some last transactions
innodb_flush_log_at_trx_commit=1
```

Check that the MySQL server has the rights to create files in `datadir`.

Note that datafiles must be < 2 GB in some file systems! The combined size of the log files must be < 4 GB. The combined size of datafiles must be >= 10 MB.

When you for the first time create an InnoDB database, it is best that you start the MySQL server from the command prompt. Then InnoDB will print the information about the database creation to the screen, and you see what is happening. See below next section what the printout should look like. For example, in Windows you can start 'mysqld-max.exe' with:

```
your-path-to-mysqld>mysqld-max --console
```

Where to put 'my.cnf' or 'my.ini' in Windows? The rules for Windows are the following:

- Only one of 'my.cnf' or 'my.ini' should be created.
- The 'my.cnf' file should be placed in the root directory of the drive 'C:'.
- The 'my.ini' file should be placed in the WINDIR directory, e.g. 'C:\WINDOWS' or 'C:\WINNT'. You can use the SET command of MS-DOS to print the value of WINDIR.
- If your PC uses a boot loader where the 'C:' drive is not the boot drive, then your only option is to use the 'my.ini' file.

Where to specify options in Unix? On Unix 'mysqld' reads options from the following files, if they exist, in the following order:

- ‘/etc/my.cnf’ Global options.
- ‘COMPILATION_DATADIR/my.cnf’ Server-specific options.
- ‘defaults-extra-file’ The file specified with `--defaults-extra-file=...`
- ‘~/my.cnf’ User-specific options.

‘COMPILATION_DATADIR’ is the MySQL data directory which was specified as a `./configure` option when ‘mysqld’ was compiled (typically ‘/usr/local/mysql/data’ for a binary installation or ‘/usr/local/var’ for a source installation).

If you are not sure from where ‘mysqld’ reads its ‘my.cnf’ or ‘my.ini’, you can give the path as the first command-line option to the server: `mysqld --defaults-file=your_path_to_my.cnf`.

InnoDB forms the directory path to a datafile by textually concatenating `innodb_data_home_dir` to a datafile name or path in `innodb_data_file_path`, adding a possible slash or backslash in between if needed. If the keyword `innodb_data_home_dir` is not mentioned in ‘my.cnf’ at all, the default for it is the ‘dot’ directory ‘.’ which means the `datadir` of MySQL.

An advanced ‘my.cnf’ example. Suppose you have a Linux computer with 2 GB RAM and three 60 GB hard disks (at directory paths ‘/’, ‘/dr2’ and ‘/dr3’). Below is an example of possible configuration parameters in ‘my.cnf’ for InnoDB.

Note that InnoDB does not create directories: you have to create them yourself. Use the Unix or MS-DOS `mkdir` command to create the data and log group home directories.

```
[mysqld]
# You can write your other MySQL server options here
# ...
innodb_data_home_dir =
#
#           Data files must be able to
#           hold your data and indexes
innodb_data_file_path = /ibdata/ibdata1:2000M;/dr2/ibdata/ibdata2:2000M:autoextend
#
#           Set buffer pool size to
#           50 - 80 % of your computer's
#           memory, but make sure on Linux
#           x86 total memory usage is
#           < 2 GB
set-variable = innodb_buffer_pool_size=1G
set-variable = innodb_additional_mem_pool_size=20M
innodb_log_group_home_dir = /dr3/iblogs
#
#           .._log_arch_dir must be the same
#           as .._log_group_home_dir
innodb_log_arch_dir = /dr3/iblogs
set-variable = innodb_log_files_in_group=3
#
#           Set the log file size to about
#           15 % of the buffer pool size
set-variable = innodb_log_file_size=150M
set-variable = innodb_log_buffer_size=8M
#
#           Set ..flush_log_at_trx_commit to
#           0 if you can afford losing
#           some last transactions
```

```
innodb_flush_log_at_trx_commit=1
set-variable = innodb_lock_wait_timeout=50
#innodb_flush_method=fdatasync
#set-variable = innodb_thread_concurrency=5
```

Note that we have placed the two datafiles on different disks. InnoDB will fill the tablespace formed by the datafiles from bottom up. In some cases it will improve the performance of the database if all data is not placed on the same physical disk. Putting log files on a different disk from data is very often beneficial for performance. You can also use **raw disk partitions** (raw devices) as datafiles. In some Unixes they speed up I/O. See the manual section on InnoDB file space management about how to specify them in ‘my.cnf’.

Warning: on Linux x86 you must be careful you **do not set memory usage too high**. glibc will allow the process heap to grow over thread stacks, which will crash your server. It is a risk if the value of

```
innodb_buffer_pool_size + key_buffer +
max_connections * (sort_buffer + read_buffer_size) + max_connections * 2 MB
```

is close to 2 GB or exceeds 2 GB. Each thread will use a stack (often 2 MB, but in MySQL AB binaries only 256 KB) and in the worst case also `sort_buffer + read_buffer_size` additional memory.

How to tune other ‘mysqld’ server parameters? Typical values which suit most users are:

```
skip-locking
set-variable = max_connections=200
set-variable = read_buffer_size=1M
set-variable = sort_buffer=1M
#                               Set key_buffer to 5 - 50%
#                               of your RAM depending on how
#                               much you use MyISAM tables, but
#                               keep key_buffer + InnoDB
#                               buffer pool size < 80% of
#                               your RAM
set-variable = key_buffer=...
```

Note that some parameters are given using the numeric ‘my.cnf’ parameter format: `set-variable = innodb... = 123`, others (string and boolean parameters) with another format: `innodb... = ...`.

The meanings of the configuration parameters are the following:

Option	Description
<code>innodb_data_home_dir</code>	The common part of the directory path for all InnoDB datafiles. If you do not mention this option in ‘my.cnf’ the default is the <code>datadir</code> of MySQL. You can specify this also as an empty string, in which case you can use absolute file paths in <code>innodb_data_file_path</code> .

<code>innodb_data_file_path</code>	Paths to individual datafiles and their sizes. The full directory path to each datafile is acquired by concatenating <code>innodb_data_home_dir</code> to the paths specified here. The file sizes are specified in megabytes, hence the 'M' after the size specification above. InnoDB also understands the abbreviation 'G', 1 G meaning 1024 MB. Starting from 3.23.44 you can set the file-size bigger than 4 GB on those operating systems which support big files. On some operating systems files must be < 2 GB. The sum of the sizes of the files must be at least 10 MB.
<code>innodb_mirrored_log_groups</code>	Number of identical copies of log groups we keep for the database. Currently this should be set to 1.
<code>innodb_log_group_home_dir</code>	Directory path to InnoDB log files.
<code>innodb_log_files_in_group</code>	Number of log files in the log group. InnoDB writes to the files in a circular fashion. Value 3 is recommended here.
<code>innodb_log_file_size</code>	Size of each log file in a log group in megabytes. Sensible values range from 1M to 1/nth of the size of the buffer pool specified below, where n is the number of log files in the group. The bigger the value, the less checkpoint flush activity is needed in the buffer pool, saving disk I/O. But bigger log files also mean that recovery will be slower in case of a crash. The combined size of log files must be < 4 GB on 32-bit computers.
<code>innodb_log_buffer_size</code>	The size of the buffer which InnoDB uses to write log to the log files on disk. Sensible values range from 1M to 8M. A big log buffer allows large transactions to run without a need to write the log to disk until the transaction commit. Thus, if you have big transactions, making the log buffer big will save disk I/O.
<code>innodb_flush_log_at_trx_commit</code>	Normally you set this to 1, meaning that at a transaction commit the log is flushed to disk, and the modifications made by the transaction become permanent, and survive a database crash. If you are willing to compromise this safety, and you are running small transactions, you may set this to 0 or 2 to reduce disk I/O to the logs. Value 0 means that the log is only written to the log file and the log file flushed to disk approximately once per second. Value 2 means the log is written to the log file at each commit, but the log file is only flushed to disk approximately once per second. The default value is 1 starting from MySQL-4.0.13, previously it was 0.
<code>innodb_log_arch_dir</code>	The directory where fully written log files would be archived if we used log archiving. The value of this parameter should currently be set the same as <code>innodb_log_group_home_dir</code> .

<code>innodb_log_archive</code>	This value should currently be set to 0. As recovery from a backup is done by MySQL using its own log files, there is currently no need to archive InnoDB log files.
<code>innodb_buffer_pool_size</code>	The size of the memory buffer InnoDB uses to cache data and indexes of its tables. The bigger you set this the less disk I/O is needed to access data in tables. On a dedicated database server you may set this parameter up to 80% of the machine physical memory size. Do not set it too large, though, because competition of the physical memory may cause paging in the operating system.
<code>innodb_additional_mem_pool_size</code>	Size of a memory pool InnoDB uses to store data dictionary information and other internal data structures. A sensible value for this might be 2M, but the more tables you have in your application the more you will need to allocate here. If InnoDB runs out of memory in this pool, it will start to allocate memory from the operating system, and write warning messages to the MySQL error log.
<code>innodb_file_io_threads</code>	Number of file I/O threads in InnoDB. Normally, this should be 4, but on Windows disk I/O may benefit from a larger number.
<code>innodb_lock_wait_timeout</code>	Timeout in seconds an InnoDB transaction may wait for a lock before being rolled back. InnoDB automatically detects transaction deadlocks in its own lock table and rolls back the transaction. If you use <code>LOCK TABLES</code> command, or other transaction-safe storage engines than InnoDB in the same transaction, then a deadlock may arise which InnoDB cannot notice. In cases like this the timeout is useful to resolve the situation.
<code>innodb_flush_method</code>	(Available from 3.23.40 up.) The default value for this is <code>fdatasync</code> . Another option is <code>O_DSYNC</code> .

7.5.4 Creating InnoDB Tablespace

Suppose you have installed MySQL and have edited `my.cnf` so that it contains the necessary InnoDB configuration parameters. Before starting MySQL you should check that the directories you have specified for InnoDB datafiles and log files exist and that you have access rights to those directories. InnoDB cannot create directories, only files. Check also you have enough disk space for the data and log files.

When you now start MySQL, InnoDB will start creating your datafiles and log files. InnoDB will print something like the following:

```
~/mysqlm/sql > mysqld
InnoDB: The first specified datafile /home/heikki/data/ibdata1
did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file /home/heikki/data/ibdata1 size to 134217728
InnoDB: Database physically writes the file full: wait...
InnoDB: datafile /home/heikki/data/ibdata2 did not exist:
```

```

new to be created
InnoDB: Setting file /home/heikki/data/ibdata2 size to 262144000
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file /home/heikki/data/logs/ib_logfile0 did not exist:
new to be created
InnoDB: Setting log file /home/heikki/data/logs/ib_logfile0 size to 5242880
InnoDB: Log file /home/heikki/data/logs/ib_logfile1 did not exist:
new to be created
InnoDB: Setting log file /home/heikki/data/logs/ib_logfile1 size to 5242880
InnoDB: Log file /home/heikki/data/logs/ib_logfile2 did not exist:
new to be created
InnoDB: Setting log file /home/heikki/data/logs/ib_logfile2 size to 5242880
InnoDB: Started
mysqld: ready for connections

```

A new InnoDB database has now been created. You can connect to the MySQL server with the usual MySQL client programs like `mysql`. When you shut down the MySQL server with `'mysqladmin shutdown'`, InnoDB output will be like the following:

```

010321 18:33:34 mysqld: Normal shutdown
010321 18:33:34 mysqld: Shutdown Complete
InnoDB: Starting shutdown...
InnoDB: Shutdown completed

```

You can now look at the datafiles and logs directories and you will see the files created. The log directory will also contain a small file named `'ib_arch_log_0000000000'`. That file resulted from the database creation, after which InnoDB switched off log archiving. When MySQL is again started, the output will be like the following:

```

~/mysqlm/sql > mysqld
InnoDB: Started
mysqld: ready for connections

```

7.5.4.1 If Something Goes Wrong in Database Creation

If InnoDB prints an operating system error in a file operation, usually the problem is one of the following:

- You did not create InnoDB data or log directories.
- `'mysqld'` does not have the rights to create files in those directories.
- `'mysqld'` does not read the right `'my.cnf'` or `'my.ini'` file, and consequently does not see the options you specified.
- The disk is full or a disk quota is exceeded.
- You have created a subdirectory whose name is equal to a datafile you specified.
- There is a syntax error in `innodb_data_home_dir` or `innodb_data_file_path`.

If something goes wrong in an InnoDB database creation, you should delete all files created by InnoDB. This means all datafiles, all log files, the small archived log file, and in the case you already did create some InnoDB tables, delete also the corresponding `'frm'` files for these tables from the MySQL database directories. Then you can try the InnoDB database creation again.

7.5.5 Creating InnoDB Tables

Suppose you have started the MySQL client with the command `mysql test`. To create a table in the InnoDB format you must specify `TYPE = InnoDB` in the table creation SQL command:

```
CREATE TABLE CUSTOMER (A INT, B CHAR (20), INDEX (A)) TYPE = InnoDB;
```

This SQL command will create a table and an index on column `A` into the InnoDB tablespace consisting of the datafiles you specified in `'my.cnf'`. In addition MySQL will create a file `'CUSTOMER.frm'` to the MySQL database directory `'test'`. Internally, InnoDB will add to its own data dictionary an entry for table `'test/CUSTOMER'`. Thus you can create a table of the same name `CUSTOMER` in another database of MySQL, and the table names will not collide inside InnoDB.

You can query the amount of free space in the InnoDB tablespace by issuing the table status command of MySQL for any table you have created with `TYPE = InnoDB`. Then the amount of free space in the tablespace appears in the table comment section in the output of `SHOW`. An example:

```
SHOW TABLE STATUS FROM test LIKE 'CUSTOMER'
```

Note that the statistics `SHOW` gives about InnoDB tables are only approximate: they are used in SQL optimisation. Table and index reserved sizes in bytes are accurate, though.

7.5.5.1 Converting MyISAM Tables to InnoDB

InnoDB does not have a special optimisation for separate index creation. Therefore it does not pay to export and import the table and create indexes afterwards. The fastest way to alter a table to InnoDB is to do the inserts directly to an InnoDB table, that is, use `ALTER TABLE ... TYPE=INNODB`, or create an empty InnoDB table with identical definitions and insert the rows with `INSERT INTO ... SELECT * FROM ...`

To get better control over the insertion process, it may be good to insert big tables in pieces:

```
INSERT INTO newtable SELECT * FROM oldtable
WHERE yourkey > something AND yourkey <= somethingelse;
```

After all data has been inserted you can rename the tables.

During the conversion of big tables you should set the InnoDB buffer pool size big to reduce disk I/O. Not bigger than 80% of the physical memory, though. You should set InnoDB log files big, and also the log buffer large.

Make sure you do not run out of tablespace: InnoDB tables take a lot more space than MyISAM tables. If an `ALTER TABLE` runs out of space, it will start a rollback, and that can take hours if it is disk-bound. In inserts InnoDB uses the insert buffer to merge secondary index records to indexes in batches. That saves a lot of disk I/O. In rollback no such mechanism is used, and the rollback can take 30 times longer than the insertion.

In the case of a runaway rollback, if you do not have valuable data in your database, it is better that you kill the database process and delete all InnoDB data and log files and all InnoDB table `'frm'` files, and start your job again, rather than wait for millions of disk I/Os to complete.

7.5.5.2 Foreign Key Constraints

Starting from version 3.23.43b InnoDB features foreign key constraints. InnoDB is the first MySQL table type which allows you to define foreign key constraints to guard the integrity of your data.

The syntax of a foreign key constraint definition in InnoDB:

```
[CONSTRAINT symbol] FOREIGN KEY (index_nome_coluna, ...)
    REFERENCES table_name (index_nome_coluna, ...)
    [ON DELETE {CASCADE | SET NULL | NO ACTION
              | RESTRICT}]
    [ON UPDATE {CASCADE | SET NULL | NO ACTION
              | RESTRICT}]
```

Both tables have to be InnoDB type and **there must be an index where the foreign key and the referenced key are listed as the FIRST columns**. InnoDB does not auto-create indexes on foreign keys or referenced keys: you have to create them explicitly.

Corresponding columns in the foreign key and the referenced key must have similar internal data types inside InnoDB so that they can be compared without a type conversion. The **size and the signedness of integer types has to be the same**. The length of string types need not be the same. If you specify a SET NULL action, make sure you have **not declared the columns in the child table NOT NULL**.

If MySQL gives the error number 1005 from a CREATE TABLE statement, and the error message string refers to errno 150, then the table creation failed because a foreign key constraint was not correctly formed. Similarly, if an ALTER TABLE fails and it refers to errno 150, that means a foreign key definition would be incorrectly formed for the altered table.

Starting from version 3.23.50 you can also associate the ON DELETE CASCADE or ON DELETE SET NULL clause with the foreign key constraint. Starting from version 4.0.8 you can use also similar ON UPDATE actions.

If ON DELETE CASCADE is specified, and a row in the parent table is deleted, then InnoDB automatically deletes also all those rows in the child table whose foreign key values are equal to the referenced key value in the parent row. If ON DELETE SET NULL is specified, the child rows are automatically updated so that the columns in the foreign key are set to the SQL NULL value.

Starting from version 3.23.50, InnoDB does not check foreign key constraints on those foreign key or referenced key values which contain a NULL column.

Starting from version 3.23.50 the InnoDB parser allows you to use backquotes (‘) around table and column names in the FOREIGN KEY ... REFERENCES ... clause but the InnoDB parser is not yet aware of the option lower_case_table_names you can specify in ‘my.cnf’.

An example:

```
CREATE TABLE parent(id INT NOT NULL, PRIMARY KEY (id)) TYPE=INNODB;
CREATE TABLE child(id INT, parent_id INT, INDEX par_ind (parent_id),
    FOREIGN KEY (parent_id) REFERENCES parent(id)
    ON DELETE SET NULL
) TYPE=INNODB;
```

Starting from version 3.23.50 InnoDB allows you to add a new foreign key constraint to a table through

```
ALTER TABLE yourtablename
ADD [CONSTRAINT symbol] FOREIGN KEY (...) REFERENCES anothertablename(...)
[on_delete_and_on_update_actions]
```

Remember to create the required indexes first, though.

In InnoDB versions < 3.23.50 `ALTER TABLE` or `CREATE INDEX` should not be used in connection with tables which have foreign key constraints or which are referenced in foreign key constraints: Any `ALTER TABLE` removes all foreign key constraints defined for the table. You should not use `ALTER TABLE` to the referenced table either, but use `DROP TABLE` and `CREATE TABLE` to modify the schema. When MySQL does an `ALTER TABLE` it may internally use `RENAME TABLE`, and that will confuse the foreign key constraints which refer to the table. A `CREATE INDEX` statement is in MySQL processed as an `ALTER TABLE`, and these restrictions apply also to it.

When doing foreign key checks InnoDB sets shared row level locks on child or parent records it has to look at. InnoDB checks foreign key constraints immediately: the check is not deferred to transaction commit.

InnoDB allows you to drop any table even though that would break the foreign key constraints which reference the table. When you drop a table the constraints which were defined in its create statement are also dropped.

If you re-create a table which was dropped, it has to have a definition which conforms to the foreign key constraints referencing it. It must have the right column names and types, and it must have indexes on the referenced keys, as stated above. If these are not satisfied, MySQL returns error number 1005 and refers to `errno 150` in the error message string.

Starting from version 3.23.50 InnoDB returns the foreign key definitions of a table when you call

```
SHOW CREATE TABLE yourtablename
```

Then also `'mysqldump'` produces correct definitions of tables to the dump file, and does not forget about the foreign keys.

You can also list the foreign key constraints for a table T with

```
SHOW TABLE STATUS FROM yourdatabasename LIKE 'T'
```

The foreign key constraints are listed in the table comment of the output.

7.5.6 Adding and Removing InnoDB Data and Log Files

From version 3.23.50 and 4.0.2 you can specify the last InnoDB datafile to `autoextend`. Alternatively, you can increase to your tablespace by specifying an additional datafile. To do this you have to shut down the MySQL server, edit the `'my.cnf'` file adding a new datafile to `innodb_data_file_path`, and then start the MySQL server again.

Currently you cannot remove a datafile from InnoDB. To decrease the size of your database you have to use `'mysqldump'` to dump all your tables, create a new database, and import your tables to the new database.

If you want to change the number or the size of your InnoDB log files, you have to shut down MySQL and make sure that it shuts down without errors. Then copy the old log files into a safe place just in case something went wrong in the shutdown and you will need them to recover the database. Delete then the old log files from the log file directory, edit

'my.cnf', and start MySQL again. InnoDB will tell you at the startup that it is creating new log files.

7.5.7 Backing up and Recovering an InnoDB Database

The key to safe database management is taking regular backups.

InnoDB Hot Backup is an online backup tool you can use to backup your InnoDB database while it is running. InnoDB Hot Backup does not require you to shut down your database and it does not set any locks or disturb your normal database processing. InnoDB Hot Backup is a non-free additional tool which is not included in the standard MySQL distribution. See the InnoDB Hot Backup homepage <http://www.innodb.com/hotbackup.html> for detailed information and screenshots.

If you are able to shut down your MySQL server, then to take a 'binary' backup of your database you have to do the following:

- Shut down your MySQL database and make sure it shuts down without errors.
- Copy all your datafiles into a safe place.
- Copy all your InnoDB log files to a safe place.
- Copy your 'my.cnf' configuration file(s) to a safe place.
- Copy all the '.frm' files for your InnoDB tables into a safe place.

In addition to taking the binary backups described above, you should also regularly take dumps of your tables with 'mysqldump'. The reason to this is that a binary file may be corrupted without you noticing it. Dumped tables are stored into text files which are human-readable and much simpler than database binary files. Seeing table corruption from dumped files is easier, and since their format is simpler, the chance for serious data corruption in them is smaller.

A good idea is to take the dumps at the same time you take a binary backup of your database. You have to shut out all clients from your database to get a consistent snapshot of all your tables into your dumps. Then you can take the binary backup, and you will then have a consistent snapshot of your database in two formats.

To be able to recover your InnoDB database to the present from the binary backup described above, you have to run your MySQL database with the general logging and log archiving of MySQL switched on. Here by the general logging we mean the logging mechanism of the MySQL server which is independent of InnoDB logs.

To recover from a crash of your MySQL server process, the only thing you have to do is to restart it. InnoDB will automatically check the logs and perform a roll-forward of the database to the present. InnoDB will automatically roll back uncommitted transactions which were present at the time of the crash. During recovery, InnoDB will print out something like the following:

```
~/mysqlm/sql > mysqld
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
```

```

InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections

```

If your database gets corrupted or your disk fails, you have to do the recovery from a backup. In the case of corruption, you should first find a backup which is not corrupted. From a backup do the recovery from the general log files of MySQL according to instructions in the MySQL manual.

7.5.7.1 Checkpoints

InnoDB implements a checkpoint mechanism called a fuzzy checkpoint. InnoDB will flush modified database pages from the buffer pool in small batches, there is no need to flush the buffer pool in one single batch, which would in practice stop processing of user SQL statements for a while.

In crash recovery InnoDB looks for a checkpoint label written to the log files. It knows that all modifications to the database before the label are already present on the disk image of the database. Then InnoDB scans the log files forward from the place of the checkpoint applying the logged modifications to the database.

InnoDB writes to the log files in a circular fashion. All committed modifications which make the database pages in the buffer pool different from the images on disk must be available in the log files in case InnoDB has to do a recovery. This means that when InnoDB starts to reuse a log file in the circular fashion, it has to make sure that the database page images on disk already contain the modifications logged in the log file InnoDB is going to reuse. In other words, InnoDB has to make a checkpoint and often this involves flushing of modified database pages to disk.

The above explains why making your log files very big may save disk I/O in checkpointing. It can make sense to set the total size of the log files as big as the buffer pool or even bigger. The drawback in big log files is that crash recovery can last longer because there will be more log to apply to the database.

7.5.8 Moving an InnoDB Database to Another Machine

InnoDB data and log files are binary-compatible on all platforms if the floating-point number format on the machines is the same. You can move an InnoDB database simply by

copying all the relevant files, which we already listed in the previous section on backing up a database. If the floating-point formats on the machines are different but you have not used `FLOAT` or `DOUBLE` data types in your tables then the procedure is the same: just copy the relevant files. If the formats are different and your tables contain floating-point data, you have to use `'mysqldump'` and `'mysqlimport'` to move those tables.

A performance tip is to switch off auto-commit mode when you import data into your database, assuming your tablespace has enough space for the big rollback segment the big import transaction will generate. Do the commit only after importing a whole table or a segment of a table.

7.5.9 InnoDB Transaction Model

In the InnoDB transaction model the goal has been to combine the best properties of a multi-versioning database to traditional two-phase locking. InnoDB does locking on row level and runs queries by default as non-locking consistent reads, in the style of Oracle. The lock table in InnoDB is stored so space-efficiently that lock escalation is not needed: typically several users are allowed to lock every row in the database, or any random subset of the rows, without InnoDB running out of memory.

In InnoDB all user activity happens inside transactions. If the autocommit mode is used in MySQL, then each SQL statement will form a single transaction. MySQL always starts a new connection with the autocommit mode switched on.

If the autocommit mode is switched off with `SET AUTOCOMMIT = 0`, then we can think that a user always has a transaction open. If he issues the `SQL COMMIT` or `ROLLBACK` statement, it ends the current transaction, and a new starts. Both statements will release all InnoDB locks that were set during the current transaction. A `COMMIT` means that the changes made in the current transaction are made permanent and become visible to other users. A `ROLLBACK`, on the other hand, cancels all modifications made by the current transaction.

If the connection has `AUTOCOMMIT = 1`, then the user can still perform a multi-statement transaction by starting it with `BEGIN` and ending it with `COMMIT` or `ROLLBACK`.

In terms of the SQL-92 transaction isolation levels, the InnoDB default is `REPEATABLE READ`. Starting from version 4.0.5, InnoDB offers all 4 different transaction isolation levels described by the SQL-92 standard. You can set the default isolation level for all connections in the `[mysqld]` section of `'my.cnf'`:

```
transaction-isolation = {READ-UNCOMMITTED | READ-COMMITTED
                        | REPEATABLE-READ | SERIALIZABLE}
```

A user can change the isolation level of a single session or all new incoming connections with the

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL
    {READ UNCOMMITTED | READ COMMITTED
     | REPEATABLE READ | SERIALIZABLE}
```

SQL statement. Note that there are no hyphens in level names in the SQL syntax. If you specify the keyword `GLOBAL` in the above statement, it will determine the initial isolation level of new incoming connections, but will not change the isolation level of old connections.

Any user is free to change the isolation level of his session, even in the middle of a transaction. In versions < 3.23.50 `SET TRANSACTION` had no effect on InnoDB tables. In versions < 4.0.5 only `REPEATABLE READ` and `SERIALIZABLE` were available.

You can query the global and session transaction isolation levels with:

```
SELECT @@global.tx_isolation;
SELECT @@tx_isolation;
```

In row level locking InnoDB uses so-called next-key locking. That means that besides index records, InnoDB can also lock the 'gap' before an index record to block insertions by other users immediately before the index record. A next-key lock means a lock which locks an index record and the gap before it. A gap lock means a lock which only locks a gap before some index record.

A detailed description of each isolation level in InnoDB:

- **READ UNCOMMITTED** This is also called 'dirty read': non-locking `SELECT`s are performed so that we do not look at a possible earlier version of a record; thus they are not 'consistent' reads under this isolation level; otherwise this level works like `READ COMMITTED`.
- **READ COMMITTED** Somewhat Oracle-like isolation level. All `SELECT ... FOR UPDATE` and `SELECT ... LOCK IN SHARE MODE` statements only lock the index records, NOT the gaps before them, and thus allow free inserting of new records next to locked records. `UPDATE` and `DELETE` which use a unique index with a unique search condition, only lock the index record found, not the gap before it. But still in range type `UPDATE` and `DELETE` InnoDB must set next-key or gap locks and block insertions by other users to the gaps covered by the range. This is necessary since 'phantom rows' have to be blocked for MySQL replication and recovery to work. **Consistent reads** behave like in Oracle: each consistent read, even within the same transaction, sets and reads its own fresh snapshot.
- **REPEATABLE READ** This is the default isolation level of InnoDB. `SELECT ... FOR UPDATE`, `SELECT ... LOCK IN SHARE MODE`, `UPDATE`, and `DELETE` which use a unique index with a unique search condition, only lock the index record found, not the gap before it. Otherwise these operations employ next-key locking, locking the index range scanned with next-key or gap locks, and block new insertions by other users. In **consistent reads** there is an important difference from the previous isolation level: in this level all consistent reads within the same transaction read the same snapshot established by the first read. This convention means that if you issue several plain `SELECT`s within the same transaction, these `SELECT`s are consistent also with respect to each other.
- **SERIALIZABLE** This level is like the previous one, but all plain `SELECT`s are implicitly converted to `SELECT ... LOCK IN SHARE MODE`.

7.5.9.1 Consistent Read

A consistent read means that InnoDB uses its multi-versioning to present to a query a snapshot of the database at a point in time. The query will see the changes made by exactly those transactions that committed before that point of time, and no changes made by later or uncommitted transactions. The exception to this rule is that the query will see the changes made by the transaction itself which issues the query.

If you are running with the default `REPEATABLE READ` isolation level, then all consistent reads within the same transaction read the snapshot established by the first such read in

that transaction. You can get a fresher snapshot for your queries by committing the current transaction and after that issuing new queries.

Consistent read is the default mode in which InnoDB processes `SELECT` statements in `READ COMMITTED` and `REPEATABLE READ` isolation levels. A consistent read does not set any locks on the tables it accesses, and therefore other users are free to modify those tables at the same time a consistent read is being performed on the table.

7.5.9.2 Locking Reads

A consistent read is not convenient in some circumstances. Suppose you want to add a new row into your table `CHILD`, and make sure that the child already has a parent in table `PARENT`.

Suppose you use a consistent read to read the table `PARENT` and indeed see the parent of the child in the table. Can you now safely add the child row to table `CHILD`? No, because it may happen that meanwhile some other user has deleted the parent row from the table `PARENT`, and you are not aware of that.

The solution is to perform the `SELECT` in a locking mode, `LOCK IN SHARE MODE`.

```
SELECT * FROM PARENT WHERE NAME = 'Jones' LOCK IN SHARE MODE;
```

Performing a read in share mode means that we read the latest available data, and set a shared mode lock on the rows we read. If the latest data belongs to a yet uncommitted transaction of another user, we will wait until that transaction commits. A shared mode lock prevents others from updating or deleting the row we have read. After we see that the above query returns the parent `'Jones'`, we can safely add his child to table `CHILD`, and commit our transaction. This example shows how to implement referential integrity in your application code.

Let us look at another example: we have an integer counter field in a table `CHILD_CODES` which we use to assign a unique identifier to each child we add to table `CHILD`. Obviously, using a consistent read or a shared mode read to read the present value of the counter is not a good idea, since then two users of the database may see the same value for the counter, and we will get a duplicate key error when we add the two children with the same identifier to the table.

In this case there are two good ways to implement the reading and incrementing of the counter: (1) update the counter first by incrementing it by 1 and only after that read it, or (2) read the counter first with a lock mode `FOR UPDATE`, and increment after that:

```
SELECT COUNTER_FIELD FROM CHILD_CODES FOR UPDATE;
UPDATE CHILD_CODES SET COUNTER_FIELD = COUNTER_FIELD + 1;
```

A `SELECT . . . FOR UPDATE` will read the latest available data setting exclusive locks on each row it reads. Thus it sets the same locks a searched SQL `UPDATE` would set on the rows.

7.5.9.3 Next-key Locking: Avoiding the Phantom Problem

In row level locking InnoDB uses an algorithm called next-key locking. InnoDB does the row level locking so that when it searches or scans an index of a table, it sets shared or exclusive locks on the index records in encounters. Thus the row level locks are more precisely called index record locks.

The locks InnoDB sets on index records also affect the 'gap' before that index record. If a user has a shared or exclusive lock on record R in an index, then another user cannot insert a new index record immediately before R in the index order. This locking of gaps is done to prevent the so-called phantom problem. Suppose I want to read and lock all children with identifier bigger than 100 from table CHILD, and update some field in the selected rows.

```
SELECT * FROM CHILD WHERE ID > 100 FOR UPDATE;
```

Suppose there is an index on table CHILD on column ID. Our query will scan that index starting from the first record where ID is bigger than 100. Now, if the locks set on the index records would not lock out inserts made in the gaps, a new child might meanwhile be inserted to the table. If now I in my transaction execute

```
SELECT * FROM CHILD WHERE ID > 100 FOR UPDATE;
```

again, I will see a new child in the result set the query returns. This is against the isolation principle of transactions: a transaction should be able to run so that the data it has read does not change during the transaction. If we regard a set of rows as a data item, then the new 'phantom' child would break this isolation principle.

When InnoDB scans an index it can also lock the gap after the last record in the index. Just that happens in the previous example: the locks set by InnoDB will prevent any insert to the table where ID would be bigger than 100.

You can use next-key locking to implement a uniqueness check in your application: if you read your data in share mode and do not see a duplicate for a row you are going to insert, then you can safely insert your row and know that the next-key lock set on the successor of your row during the read will prevent anyone meanwhile inserting a duplicate for your row. Thus the next-key locking allows you to 'lock' the non-existence of something in your table.

7.5.9.4 Locks Set by Different SQL Statements in InnoDB

- `SELECT ... FROM ...` : this is a consistent read, reading a snapshot of the database and setting no locks.
- `SELECT ... FROM ... LOCK IN SHARE MODE` : sets shared next-key locks on all index records the read encounters.
- `SELECT ... FROM ... FOR UPDATE` : sets exclusive next-key locks on all index records the read encounters.
- `INSERT INTO ... VALUES (...)` : sets an exclusive lock on the inserted row; note that this lock is not a next-key lock and does not prevent other users from inserting to the gap before the inserted row. If a duplicate key error occurs, sets a shared lock on the duplicate index record.
- `INSERT INTO T SELECT ... FROM S WHERE ...` sets an exclusive (non-next-key) lock on each row inserted into T. Does the search on S as a consistent read, but sets shared next-key locks on S if the MySQL logging is on. InnoDB has to set locks in the latter case because in roll-forward recovery from a backup every SQL statement has to be executed in exactly the same way as it was done originally.
- `CREATE TABLE ... SELECT ...` performs the `SELECT` as a consistent read or with shared locks, like in the previous item.

- **REPLACE** is done like an insert if there is no collision on a unique key. Otherwise, an exclusive next-key lock is placed on the row which has to be updated.
- **UPDATE ... SET ... WHERE ...** : sets an exclusive next-key lock on every record the search encounters.
- **DELETE FROM ... WHERE ...** : sets an exclusive next-key lock on every record the search encounters.
- If a **FOREIGN KEY** constraint is defined on a table, any insert, update, or delete which requires checking of the constraint condition sets shared record level locks on the records it looks at to check the constraint. Also in the case where the constraint fails, InnoDB sets these locks.
- **LOCK TABLES ...** : sets table locks. In the implementation the MySQL layer of code sets these locks. The automatic deadlock detection of InnoDB cannot detect deadlocks where such table locks are involved: see the following section. Also, since MySQL does know about row level locks, it is possible that you get a table lock on a table where another user currently has row level locks. But that does not put transaction integrity into danger. See [\(undefined\)](#) [InnoDB restrictions], page [\(undefined\)](#).

7.5.9.5 Deadlock Detection and Rollback

InnoDB automatically detects a deadlock of transactions and rolls back a transaction or transactions to prevent the deadlock. Starting from version 4.0.5, InnoDB will try to pick small transactions to roll back. The size of a transaction is determined by the number of rows it has inserted, updated, or deleted. Previous to 4.0.5, InnoDB always rolled back the transaction whose lock request was the last one to build a deadlock, that is, a cycle in the waits-for graph of transactions.

InnoDB cannot detect deadlocks where a lock set by a MySQL **LOCK TABLES** statement is involved, or if a lock set in another storage engine than InnoDB is involved. You have to resolve these situations using `innodb_lock_wait_timeout` set in `'my.cnf'`.

When InnoDB performs a complete rollback of a transaction, all the locks of the transaction are released. However, if just a single SQL statement is rolled back as a result of an error, some of the locks set by the SQL statement may be preserved. This is because InnoDB stores row locks in a format where it cannot afterwards know which was set by which SQL statement.

7.5.9.6 An Example of How the Consistent Read Works in InnoDB

Suppose you are running on the default **REPEATABLE READ** isolation level. When you issue a consistent read, that is, an ordinary **SELECT** statement, InnoDB will give your transaction a timepoint according to which your query sees the database. Thus, if transaction B deletes a row and commits after your timepoint was assigned, then you will not see the row deleted. Similarly with inserts and updates.

You can advance your timepoint by committing your transaction and then doing another **SELECT**.

This is called multi-versioned concurrency control.

	User A	User B
	SET AUTOCOMMIT=0;	SET AUTOCOMMIT=0;
time		
	SELECT * FROM t;	
	empty set	
		INSERT INTO t VALUES (1, 2);
v	SELECT * FROM t;	
	empty set	COMMIT;
	SELECT * FROM t;	
	empty set;	
	COMMIT;	
	SELECT * FROM t;	

	1 2	

Thus user A sees the row inserted by B only when B has committed the insert, and A has committed his own transaction so that the timepoint is advanced past the commit of B.

If you want to see the “freshest” state of the database, you should use a locking read:

```
SELECT * FROM t LOCK IN SHARE MODE;
```

7.5.9.7 How to cope with deadlocks?

Deadlocks are a classic problem in transactional databases, but they are not dangerous, unless they are so frequent that you cannot run certain transactions at all. Normally you have to write your applications so that they are always prepared to re-issue a transaction if it gets rolled back because of a deadlock.

InnoDB uses automatic row level locking. You can get deadlocks even in the case of transactions which just insert or delete a single row. That is because these operations are not really ‘atomic’: they automatically set locks on the (possibly several) index records of the row inserted/deleted.

You can cope with deadlocks and reduce the number of them with the following tricks:

- Use `SHOW INNODB STATUS` in MySQL versions `>= 3.23.52` and `>= 4.0.3` to determine the cause of the latest deadlock. That can help you to tune your application to avoid deadlocks.
- Always be prepared to re-issue a transaction if it fails in a deadlock. Deadlocks are not dangerous. Just try again.
- Commit your transactions often. Small transactions are less prone to collide.
- If you are using locking reads `SELECT ... FOR UPDATE` or `... LOCK IN SHARE MODE`, try using a lower isolation level `READ COMMITTED`.

- Access your tables and rows in a fixed order. Then transactions will form nice queues, and do not deadlock.
- Add good indexes to your tables. Then your queries need to scan less index records and consequently set less locks. Use `EXPLAIN SELECT` to determine that MySQL picks appropriate indexes for your queries.
- Use less locking: if you can afford a `SELECT` to return data from an old snapshot, do not add the clause `FOR UPDATE` or `LOCK IN SHARE MODE` to it. Using `READ COMMITTED` isolation level is good here, because each consistent read within the same transaction reads from its own fresh snapshot.
- If nothing helps, serialize your transactions with table level locks: `LOCK TABLES t1 WRITE, t2 READ, ... ; [do something with tables t1 and t2 here]; UNLOCK TABLES`. Table level locks make you transactions to queue nicely, and deadlocks are avoided. Note that `LOCK TABLES` implicitly starts a transaction, just like the command `BEGIN`, and `UNLOCK TABLES` implicitly ends the transaction in a `COMMIT`.
- Another solution to serialize transactions is to create an auxiliary 'semaphore' table where there is just a single row. Each transaction updates that row before accessing other tables. In that way all transactions happen in a serial fashion. Note that then also the InnoDB instant deadlock detection algorithm works, because the serializing lock is a row level lock. In MySQL table level locks we have to resort to the timeout method to resolve a deadlock.

7.5.9.8 Performance Tuning Tips

1. If the Unix 'top' or the Windows 'Task Manager' shows that the CPU usage percentage with your workload is less than 70%, your workload is probably disk-bound. Maybe you are making too many transaction commits, or the buffer pool is too small. Making the buffer pool bigger can help, but do not set it bigger than 80% of physical memory.
2. Wrap several modifications into one transaction. InnoDB must flush the log to disk at each transaction commit, if that transaction made modifications to the database. Since the rotation speed of a disk is typically at most 167 revolutions/second, that constrains the number of commits to the same 167/second if the disk does not fool the operating system.
3. If you can afford the loss of some latest committed transactions, you can set the 'my.cnf' parameter `innodb_flush_log_at_trx_commit` to 0. InnoDB tries to flush the log once per second anyway, though the flush is not guaranteed.
4. Make your log files big, even as big as the buffer pool. When InnoDB has written the log files full, it has to write the modified contents of the buffer pool to disk in a checkpoint. Small log files will cause many unnecessary disk writes. The drawback in big log files is that recovery time will be longer.
5. Also the log buffer should be quite big, say 8 MB.
6. (Relevant from 3.23.39 up.) In some versions of Linux and Unix, flushing files to disk with the Unix `fdatasync` and other similar methods is surprisingly slow. The default method InnoDB uses is the `fdatasync` function. If you are not satisfied with the database write performance, you may try setting `innodb_flush_method` in 'my.cnf' to `O_DSYNC`, though `O_DSYNC` seems to be slower on most systems.

7. In importing data to InnoDB, make sure that MySQL does not have `autocommit=1` on. Then every insert requires a log flush to disk. Put before your plain SQL import file line

```
SET AUTOCOMMIT=0;
```

and after it

```
COMMIT;
```

If you use the `'mysqldump'` option `--opt`, you will get dump files which are fast to import also to an InnoDB table, even without wrapping them to the above `SET AUTOCOMMIT=0; ... COMMIT;` wrappers.

8. Beware of big rollbacks of mass inserts: InnoDB uses the insert buffer to save disk I/O in inserts, but in a corresponding rollback no such mechanism is used. A disk-bound rollback can take 30 times the time of the corresponding insert. Killing the database process will not help because the rollback will start again at the database startup. The only way to get rid of a runaway rollback is to increase the buffer pool so that the rollback becomes CPU-bound and runs fast, or delete the whole InnoDB database.

9. Beware also of other big disk-bound operations. Use `DROP TABLE` or `TRUNCATE` (from MySQL-4.0 up) to empty a table, not `DELETE FROM yourtable`.

10. Use the multi-line `INSERT` to reduce communication overhead between the client and the server if you need to insert many rows:

```
INSERT INTO yourtable VALUES (1, 2), (5, 5);
```

This tip is of course valid for inserts into any table type, not just InnoDB.

7.5.9.9 The InnoDB Monitor

Starting from version 3.23.41 InnoDB includes the InnoDB Monitor which prints information on the InnoDB internal state. When switched on, InnoDB Monitor will make the MySQL server `'mysqld'` to print data (note: the MySQL client will not print anything) to the standard output about once every 15 seconds. This data is useful in performance tuning. On Windows you must start `mysqld-max` from a MS-DOS prompt with the `--standalone --console` options to direct the output to the MS-DOS prompt window.

There is a separate `innodb_lock_monitor` which prints the same information as `innodb_monitor` plus information on locks set by each transaction.

The printed information includes data on:

- lock waits of a transactions,
- semaphore waits of threads,
- pending file I/O requests,
- buffer pool statistics, and
- purge and insert buffer merge activity of the main thread of InnoDB.

You can start InnoDB Monitor through the following SQL command:

```
CREATE TABLE innodb_monitor(a INT) type = innodb;
```

and stop it by

```
DROP TABLE innodb_monitor;
```

The `CREATE TABLE` syntax is just a way to pass a command to the InnoDB engine through the MySQL SQL parser: the created table is not relevant at all for InnoDB Monitor. If you

shut down the database when the monitor is running, and you want to start the monitor again, you have to drop the table before you can issue a new CREATE TABLE to start the monitor. This syntax may change in a future release.

A sample output of the InnoDB Monitor:

```

=====
010809 18:45:06 INNODB MONITOR OUTPUT
=====
-----
LOCKS HELD BY TRANSACTIONS
-----
LOCK INFO:
Number of locks in the record hash table 1294
LOCKS FOR TRANSACTION ID 0 579342744
TABLE LOCK table test/mytable trx id 0 582333343 lock_mode IX

RECORD LOCKS space id 0 page no 12758 n bits 104 table test/mytable index
PRIMARY trx id 0 582333343 lock_mode X
Record lock, heap no 2 PHYSICAL RECORD: n_fields 74; 1-byte offs FALSE;
info bits 0
  0: len 4; hex 0001a801; asc ;; 1: len 6; hex 000022b5b39f; asc ";;
  2: len 7; hex 000002001e03ec; asc ;; 3: len 4; hex 00000001;
...
-----
CURRENT SEMAPHORES RESERVED AND SEMAPHORE WAITS
-----
SYNC INFO:
Sorry, cannot give mutex list info in non-debug version!
Sorry, cannot give rw-lock list info in non-debug version!
-----
SYNC ARRAY INFO: reservation count 6041054, signal count 2913432
4a239430 waited for by thread 49627477 op. S-LOCK file NOT KNOWN line 0
Mut ex 0 sp 5530989 r 62038708 sys 2155035;
rws 0 8257574 8025336; rwx 0 1121090 1848344
-----
CURRENT PENDING FILE I/O'S
-----
Pending normal aio reads:
Reserved slot, messages 40157658 4a4a40b8
Reserved slot, messages 40157658 4a477e28
...
Reserved slot, messages 40157658 4a4424a8
Reserved slot, messages 40157658 4a39ea38
Total of 36 reserved aio slots
Pending aio writes:
Total of 0 reserved aio slots
Pending insert buffer aio reads:
Total of 0 reserved aio slots
Pending log writes or reads:
Reserved slot, messages 40158c98 40157f98

```

```

Total of 1 reserved aio slots
Pending synchronous reads or writes:
Total of 0 reserved aio slots
-----
BUFFER POOL
-----
LRU list length 8034
Free list length 0
Flush list length 999
Buffer pool size in pages 8192
Pending reads 39
Pending writes: LRU 0, flush list 0, single page 0
Pages read 31383918, created 51310, written 2985115
-----
END OF INNODB MONITOR OUTPUT
=====
010809 18:45:22 InnoDB starts purge
010809 18:45:22 InnoDB purged 0 pages

```

Some notes on the output:

- If the section `LOCKS HELD BY TRANSACTIONS` reports lock waits, then your application may have lock contention. The output can also help to trace reasons for transaction deadlocks.
- Section `SYNC INFO` will report reserved semaphores if you compile InnoDB with `UNIV_SYNC_DEBUG` defined in `'univ.i'`.
- Section `SYNC ARRAY INFO` reports threads waiting for a semaphore and statistics on how many times threads have needed a spin or a wait on a mutex or a rw-lock semaphore. A big number of threads waiting for semaphores may be a result of disk I/O, or contention problems inside InnoDB. Contention can be due to heavy parallelism of queries, or problems in operating system thread scheduling.
- Section `CURRENT PENDING FILE I/O'S` lists pending file I/O requests. A large number of these indicates that the workload is disk I/O-bound.
- Section `BUFFER POOL` gives you statistics on pages read and written. You can calculate from these numbers how many datafile I/Os your queries are currently doing.

7.5.10 Implementation of Multi-versioning

Since InnoDB is a multi-versioned database, it must keep information of old versions of rows in the tablespace. This information is stored in a data structure we call a rollback segment after an analogous data structure in Oracle.

InnoDB internally adds two fields to each row stored in the database. A 6-byte field tells the transaction identifier for the last transaction which inserted or updated the row. Also a deletion is internally treated as an update where a special bit in the row is set to mark it as deleted. Each row also contains a 7-byte field called the roll pointer. The roll pointer points to an undo log record written to the rollback segment. If the row was updated, then the undo log record contains the information necessary to rebuild the content of the row before it was updated.

InnoDB uses the information in the rollback segment to perform the undo operations needed in a transaction rollback. It also uses the information to build earlier versions of a row for a consistent read.

Undo logs in the rollback segment are divided into insert and update undo logs. Insert undo logs are only needed in transaction rollback and can be discarded as soon as the transaction commits. Update undo logs are used also in consistent reads, and they can be discarded only after there is no transaction present for which InnoDB has assigned a snapshot that in a consistent read could need the information in the update undo log to build an earlier version of a database row.

You must remember to commit your transactions regularly, also those transactions which only issue consistent reads. Otherwise InnoDB cannot discard data from the update undo logs, and the rollback segment may grow too big, filling up your tablespace.

The physical size of an undo log record in the rollback segment is typically smaller than the corresponding inserted or updated row. You can use this information to calculate the space need for your rollback segment.

In our multi-versioning scheme a row is not physically removed from the database immediately when you delete it with an SQL statement. Only when InnoDB can discard the update undo log record written for the deletion, it can also physically remove the corresponding row and its index records from the database. This removal operation is called a purge, and it is quite fast, usually taking the same order of time as the SQL statement which did the deletion.

7.5.11 Table and Index Structures

MySQL stores its data dictionary information of tables in `.frm` files in database directories. But every InnoDB type table also has its own entry in InnoDB internal data dictionaries inside the tablespace. When MySQL drops a table or a database, it has to delete both a `.frm` file or files, and the corresponding entries inside the InnoDB data dictionary. This is the reason why you cannot move InnoDB tables between databases simply by moving the `.frm` files, and why `DROP DATABASE` did not work for InnoDB type tables in MySQL versions $\leq 3.23.43$.

Every InnoDB table has a special index called the clustered index where the data of the rows is stored. If you define a `PRIMARY KEY` on your table, then the index of the primary key will be the clustered index.

If you do not define a primary key for your table, InnoDB will internally generate a clustered index where the rows are ordered by the row id InnoDB assigns to the rows in such a table. The row id is a 6-byte field which monotonically increases as new rows are inserted. Thus the rows ordered by the row id will be physically in the insertion order.

Accessing a row through the clustered index is fast, because the row data will be on the same page where the index search leads us. In many databases the data is traditionally stored on a different page from the index record. If a table is large, the clustered index architecture often saves a disk I/O when compared to the traditional solution.

The records in non-clustered indexes (we also call them secondary indexes), in InnoDB contain the primary key value for the row. InnoDB uses this primary key value to search for the row from the clustered index. Note that if the primary key is long, the secondary indexes will use more space.

7.5.11.1 Physical Structure of an Index

All indexes in InnoDB are B-trees where the index records are stored in the leaf pages of the tree. The default size of an index page is 16 KB. When new records are inserted, InnoDB tries to leave 1 / 16 of the page free for future insertions and updates of the index records.

If index records are inserted in a sequential (ascending or descending) order, the resulting index pages will be about 15/16 full. If records are inserted in a random order, then the pages will be 1/2 - 15/16 full. If the fillfactor of an index page drops below 1/2, InnoDB will try to contract the index tree to free the page.

7.5.11.2 Insert Buffering

It is a common situation in a database application that the primary key is a unique identifier and new rows are inserted in the ascending order of the primary key. Thus the insertions to the clustered index do not require random reads from a disk.

On the other hand, secondary indexes are usually non-unique and insertions happen in a relatively random order into secondary indexes. This would cause a lot of random disk I/Os without a special mechanism used in InnoDB.

If an index record should be inserted to a non-unique secondary index, InnoDB checks if the secondary index page is already in the buffer pool. If that is the case, InnoDB will do the insertion directly to the index page. But, if the index page is not found from the buffer pool, InnoDB inserts the record to a special insert buffer structure. The insert buffer is kept so small that it entirely fits in the buffer pool, and insertions can be made to it very fast.

The insert buffer is periodically merged to the secondary index trees in the database. Often we can merge several insertions on the same page in of the index tree, and hence save disk I/Os. It has been measured that the insert buffer can speed up insertions to a table up to 15 times.

7.5.11.3 Adaptive Hash Indexes

If a database fits almost entirely in main memory, then the fastest way to perform queries on it is to use hash indexes. InnoDB has an automatic mechanism which monitors index searches made to the indexes defined for a table, and if InnoDB notices that queries could benefit from building of a hash index, such an index is automatically built.

But note that the hash index is always built based on an existing B-tree index on the table. InnoDB can build a hash index on a prefix of any length of the key defined for the B-tree, depending on what search pattern InnoDB observes on the B-tree index. A hash index can be partial: it is not required that the whole B-tree index is cached in the buffer pool. InnoDB will build hash indexes on demand to those pages of the index which are often accessed.

In a sense, through the adaptive hash index mechanism InnoDB adapts itself to ample main memory, coming closer to the architecture of main memory databases.

7.5.11.4 Physical Record Structure

- Each index record in InnoDB contains a header of 6 bytes. The header is used to link consecutive records together, and also in the row level locking.
- Records in the clustered index contain fields for all user-defined columns. In addition, there is a 6-byte field for the transaction id and a 7-byte field for the roll pointer.
- If the user has not defined a primary key for a table, then each clustered index record contains also a 6-byte row id field.
- Each secondary index record contains also all the fields defined for the clustered index key.
- A record contains also a pointer to each field of the record. If the total length of the fields in a record is < 128 bytes, then the pointer is 1 byte, else 2 bytes.

7.5.11.5 How an Auto-increment Column Works in InnoDB

After a database startup, when a user first does an insert to a table T where an auto-increment column has been defined, and the user does not provide an explicit value for the column, then InnoDB executes `SELECT MAX(auto-inc-column) FROM T`, and assigns that value incremented by one to the column and the auto-increment counter of the table. We say that the auto-increment counter for table T has been initialised.

InnoDB follows the same procedure in initializing the auto-increment counter for a freshly created table.

Note that if the user specifies in an insert the value 0 to the auto-increment column, then InnoDB treats the row like the value would not have been specified.

After the auto-increment counter has been initialised, if a user inserts a row where he explicitly specifies the column value, and the value is bigger than the current counter value, then the counter is set to the specified column value. If the user does not explicitly specify a value, then InnoDB increments the counter by one and assigns its new value to the column.

The auto-increment mechanism, when assigning values from the counter, bypasses locking and transaction handling. Therefore you may also get gaps in the number sequence if you roll back transactions which have got numbers from the counter.

The behaviour of auto-increment is not defined if a user gives a negative value to the column or if the value becomes bigger than the maximum integer that can be stored in the specified integer type.

7.5.12 File Space Management and Disk I/O

7.5.12.1 Disk I/O

In disk I/O InnoDB uses asynchronous I/O. On Windows NT it uses the native asynchronous I/O provided by the operating system. On Unix, InnoDB uses simulated asynchronous I/O built into InnoDB: InnoDB creates a number of I/O threads to take care of I/O operations, such as read-ahead. In a future version we will add support for simulated aio on Windows NT and native aio on those versions of Unix which have one.

On Windows NT InnoDB uses non-buffered I/O. That means that the disk pages InnoDB reads or writes are not buffered in the operating system file cache. This saves some memory bandwidth.

Starting from 3.23.41 InnoDB uses a novel file flush technique called doublewrite. It adds safety to crash recovery after an operating system crash or a power outage, and improves performance on most Unix flavors by reducing the need for fsync operations.

Doublewrite means that InnoDB before writing pages to a datafile first writes them to a contiguous tablespace area called the doublewrite buffer. Only after the write and the flush to the doublewrite buffer has completed, InnoDB writes the pages to their proper positions in the datafile. If the operating system crashes in the middle of a page write, InnoDB will in recovery find a good copy of the page from the doublewrite buffer.

Starting from 3.23.41 you can also use a raw disk partition as a datafile, though this has not been tested yet. When you create a new datafile you have to put the keyword `newraw` immediately after the data file-size in `innodb_data_file_path`. The partition must be \geq than you specify as the size. Note that 1M in InnoDB is 1024 x 1024 bytes, while in disk specifications 1 MB usually means 1000 000 bytes.

```
innodb_data_file_path=hdd1:5Gnewraw;hdd2:2Gnewraw
```

When you start the database again you **must** change the keyword to `raw`. Otherwise, InnoDB will write over your partition!

```
innodb_data_file_path=hdd1:5Graw;hdd2:2Graw
```

By using a raw disk you can on some Unixes perform unbuffered I/O.

There are two read-ahead heuristics in InnoDB: sequential read-ahead and random read-ahead. In sequential read-ahead InnoDB notices that the access pattern to a segment in the tablespace is sequential. Then InnoDB will post in advance a batch of reads of database pages to the I/O system. In random read-ahead InnoDB notices that some area in a tablespace seems to be in the process of being fully read into the buffer pool. Then InnoDB posts the remaining reads to the I/O system.

7.5.12.2 File Space Management

The datafiles you define in the configuration file form the tablespace of InnoDB. The files are simply catenated to form the tablespace, there is no striping in use. Currently you cannot directly instruct where the space is allocated for your tables, except by using the following fact: from a newly created tablespace InnoDB will allocate space starting from the low end.

The tablespace consists of database pages whose default size is 16 KB. The pages are grouped into extents of 64 consecutive pages. The 'files' inside a tablespace are called segments in InnoDB. The name of the rollback segment is somewhat misleading because it actually contains many segments in the tablespace.

For each index in InnoDB we allocate two segments: one is for non-leaf nodes of the B-tree, the other is for the leaf nodes. The idea here is to achieve better sequentiality for the leaf nodes, which contain the data.

When a segment grows inside the tablespace, InnoDB allocates the first 32 pages to it individually. After that InnoDB starts to allocate whole extents to the segment. InnoDB can add to a large segment up to 4 extents at a time to ensure good sequentiality of data.

Some pages in the tablespace contain bitmaps of other pages, and therefore a few extents in an InnoDB tablespace cannot be allocated to segments as a whole, but only as individual pages.

When you issue a query `SHOW TABLE STATUS FROM ... LIKE ...` to ask for available free space in the tablespace, InnoDB will report the extents which are definitely free in the tablespace. InnoDB always reserves some extents for clean-up and other internal purposes; these reserved extents are not included in the free space.

When you delete data from a table, InnoDB will contract the corresponding B-tree indexes. It depends on the pattern of deletes if that frees individual pages or extents to the tablespace, so that the freed space is available for other users. Dropping a table or deleting all rows from it is guaranteed to release the space to other users, but remember that deleted rows can be physically removed only in a purge operation after they are no longer needed in transaction rollback or consistent read.

7.5.12.3 Defragmenting a Table

If there are random insertions or deletions in the indexes of a table, the indexes may become fragmented. By fragmentation we mean that the physical ordering of the index pages on the disk is not close to the alphabetical ordering of the records on the pages, or that there are many unused pages in the 64-page blocks which were allocated to the index.

It can speed up index scans if you periodically use `mysqldump` to dump the table to a text file, drop the table, and reload it from the dump. Another way to do the defragmenting is to `ALTER` the table type to `MyISAM` and back to `InnoDB` again. Note that a `MyISAM` table must fit in a single file on your operating system.

If the insertions to and index are always ascending and records are deleted only from the end, then the file space management algorithm of InnoDB guarantees that fragmentation in the index will not occur.

7.5.13 Error Handling

The error handling in InnoDB is not always the same as specified in the SQL standard. According to SQL-99, any error during an SQL statement should cause the rollback of that statement. InnoDB sometimes rolls back only part of the statement, or the whole transaction. The following list specifies the error handling of InnoDB.

- If you run out of file space in the tablespace, you will get the MySQL 'Table is full' error and InnoDB rolls back the SQL statement.
- A transaction deadlock or a timeout in a lock wait make InnoDB to roll back the whole transaction.
- A duplicate key error only rolls back the insert of that particular row, even in a statement like `INSERT INTO ... SELECT ...`. This will probably change so that the SQL statement will be rolled back if you have not specified the `IGNORE` option in your statement.
- A 'row too long' error rolls back the SQL statement.
- Other errors are mostly detected by the MySQL layer of code, and they roll back the corresponding SQL statement.

7.5.14 Restrictions on InnoDB Tables

- **Warning:** do **NOT** convert MySQL system tables from MyISAM TO InnoDB tables! This is not supported; if you do this MySQL will not restart until you restore the old system tables from a backup or re-generate them with the `mysql_install_db` script.
- `SHOW TABLE STATUS` does not give accurate statistics on InnoDB tables, except for the physical size reserved by the table. The row count is only a rough estimate used in SQL optimisation.
- If you try to create a unique index on a prefix of a column you will get an error:

```
CREATE TABLE T (A CHAR(20), B INT, UNIQUE (A(5))) TYPE = InnoDB;
```

If you create a non-unique index on a prefix of a column, InnoDB will create an index over the whole column.
- `INSERT DELAYED` is not supported for InnoDB tables.
- The MySQL `LOCK TABLES` operation does not know of InnoDB row level locks set in already completed SQL statements: this means that you can get a table lock on a table even if there still exist transactions of other users which have row level locks on the same table. Thus your operations on the table may have to wait if they collide with these locks of other users. Also a deadlock is possible. However, this does not endanger transaction integrity, because the row level locks set by InnoDB will always take care of the integrity. Also, a table lock prevents other transactions from acquiring more row level locks (in a conflicting lock mode) on the table.
- You cannot have a key on a `BLOB` or `TEXT` column.
- A table cannot contain more than 1000 columns.
- `DELETE FROM TABLE` does not regenerate the table but instead deletes all rows, one by one, which is not that fast. In future versions of MySQL you can use `TRUNCATE` which is fast.
- The default database page size in InnoDB is 16 KB. By recompiling the code one can set it from 8 KB to 64 KB. The maximum row length is slightly less than half of a database page in versions $\leq 3.23.40$ of InnoDB. Starting from source release 3.23.41 `BLOB` and `TEXT` columns are allowed to be < 4 GB, the total row length must also be < 4 GB. InnoDB does not store fields whose size is ≤ 128 bytes on separate pages. After InnoDB has modified the row by storing long fields on separate pages, the remaining length of the row must be less than half a database page. The maximum key length is 7000 bytes.
- On some operating systems datafiles must be < 2 GB. The combined size of log files must be < 4 GB.
- The maximum tablespace size is 4 billion database pages. This is also the maximum size for a table. The minimum tablespace size is 10 MB.
- When you restart the MySQL server, InnoDB may reuse an old value for an `AUTO_INCREMENT` column.

7.5.15 InnoDB Change History

7.5.15.1 MySQL/InnoDB-4.1.0, April, 2003

-
- InnoDB now supports up to 64 GB of buffer pool memory in a Windows 32-bit Intel computer. This is possible because InnoDB can use the AWE extension of Windows to address memory over the 4 GB limit of a 32-bit process. A new startup variable `innodb_buffer_pool_ave_mem_mb` enables AWE and sets the size of the buffer pool in megabytes.
- Reduced the size of buffer headers and the lock table. InnoDB uses 2 % less memory.

7.5.15.2 MySQL/InnoDB-3.23.56, March, 2003

- Fixed a major bug in InnoDB query optimisation: queries of type `SELECT ... WHERE indexcolumn < x` and `SELECT ... WHERE indexcolumn > x` could cause a table scan even if the selectivity would have been very good.

7.5.15.3 MySQL/InnoDB-4.0.12, March, 2003

- In crash recovery InnoDB now prints the progress in percents of a transaction rollback.

7.5.15.4 MySQL/InnoDB-4.0.11, February 25, 2003

- Fixed a bug introduced in 4.0.10: `SELECT ... FROM ... ORDER BY ... DESC` could hang in an infinite loop.
- An outstanding bug: `SET FOREIGN_KEY_CHECKS=0` is not replicated properly in the MySQL replication.

7.5.15.5 MySQL/InnoDB-4.0.10, February 4, 2003

- In `INSERT INTO t1 SELECT ... FROM t2 WHERE ...` MySQL previously set a table level read lock on t2. This lock is now removed.
- Increased `SHOW INNODB STATUS` max printed length to 200 KB.
- Fixed a major bug in InnoDB query optimisation: queries of type `SELECT ... WHERE indexcolumn < x` and `SELECT ... WHERE indexcolumn > x` could cause a table scan even if the selectivity would have been very good.
- Fixed a bug: purge could cause a hang in a BLOB table where the primary key index tree was of height 1. Symptom: semaphore waits caused by an X-latch set in `btr_free_externally_stored_field()`.
- Fixed a bug: using InnoDB `HANDLER` commands on a fresh handle crashed `mysqld` in `ha_innobase::change_active_index()`.
- Fixed a bug: if MySQL estimated a query in the middle of a `SELECT` statement, InnoDB could hang on the adaptive hash index latch in `btr0sea.c`.
- Fixed a bug: InnoDB could report table corruption and assert in `page_dir_find_owner_slot()` if an adaptive hash index search coincided with purge or an insert.
- Fixed a bug: some file system snapshot tool in Windows 2000 could cause an InnoDB file write to fail with error 33 `ERROR_LOCK_VIOLATION`. In synchronous writes InnoDB now retries the write 100 times at 1 second intervals.

- Fixed a bug: REPLACE INTO t1 SELECT ... did not work if t1 has an auto-inc column.
- An outstanding bug: SET FOREIGN_KEY_CHECKS=0 is not replicated properly in the MySQL replication.

7.5.15.6 MySQL/InnoDB-3.23.55, January 24, 2003

- In INSERT INTO t1 SELECT ... FROM t2 WHERE ... MySQL previously set a table level read lock on t2. This lock is now removed.
- Fixed a bug: if the combined size of InnoDB log files was ≥ 2 GB in a 32-bit computer, InnoDB would write log in a wrong position. That could make crash recovery and InnoDB Hot Backup to fail in log scan.
- Fixed a bug: index cursor restoration could theoretically fail.
- Fixed a bug: an assertion in btr0sea.c, in function btr_search_info_update_slow could theoretically fail in a race of 3 threads.
- Fixed a bug: purge could cause a hang in a BLOB table where the primary key index tree was of height 1. Symptom: semaphore waits caused by an X-latch set in btr_free_externally_stored_field().
- Fixed a bug: if MySQL estimated a query in the middle of a SELECT statement, InnoDB could hang on the adaptive hash index latch in btr0sea.c.
- Fixed a bug: InnoDB could report table corruption and assert in page_dir_find_owner_slot() if an adaptive hash index search coincided with purge or an insert.
- Fixed a bug: some file system snapshot tool in Windows 2000 could cause an InnoDB file write to fail with error 33 ERROR_LOCK_VIOLATION. In synchronous writes InnoDB now retries the write 100 times at 1 second intervals.
- An outstanding bug: SET FOREIGN_KEY_CHECKS=0 is not replicated properly in the MySQL replication. The fix will appear in 4.0.11 and will probably not be backported to 3.23.
- Fixed bug in InnoDB page0cur.c in function page_cur_search_with_match which caused InnoDB to remain on the same page forever. This bug is evident only in tables with more than one page.

7.5.15.7 MySQL/InnoDB-4.0.9, January 14, 2003

- Removed the warning message: 'InnoDB: Out of memory in additional memory pool.'
- Fixed a bug: if the combined size of InnoDB log files was ≥ 2 GB in a 32-bit computer, InnoDB would write log in a wrong position. That could make crash recovery and InnoDB Hot Backup to fail.
- Fixed a bug: index cursor restoration could theoretically fail.

7.5.15.8 MySQL/InnoDB-4.0.8, January 7, 2003

- InnoDB now supports also FOREIGN KEY (...) REFERENCES ...(...) [ON UPDATE CASCADE | ON UPDATE SET NULL | ON UPDATE RESTRICT | ON UPDATE NO ACTION].

- Tables and indexes now reserve 4 % less space in the tablespace. Also existing tables reserve less space. By upgrading to 4.0.8 you will see more free space in "InnoDB free" in SHOW TABLE STATUS.
- Fixed bugs: updating the PRIMARY KEY of a row would generate a foreign key error on all FOREIGN KEYS which referenced secondary keys of the row to be updated. Also, if a referencing FOREIGN KEY constraint only referenced the first columns in an index, and there were more columns in that index, updating the additional columns generated a foreign key error.
- Fixed a bug: if an index contains some column twice, and that column is updated, the table will become corrupt. From now on InnoDB prevents creation of such indexes.
- Fixed a bug: removed superfluous error 149 and 150 printouts from the .err log when a locking SELECT caused a deadlock or a lock wait timeout.
- Fixed a bug: an assertion in btr0sea.c, in function btr_search_info_update_slow could theoretically fail in a race of 3 threads.
- Fixed a bug: one could not switch a session transaction isolation level back to REPEATABLE READ after setting it to something else.

7.5.15.9 MySQL/InnoDB-4.0.7, December 26, 2002

- InnoDB in 4.0.7 is essentially the same as in 4.0.6.

7.5.15.10 MySQL/InnoDB-4.0.6, December 19, 2002

- Since innodb_log_arch_dir has no relevance under MySQL, there is no need to specify it any more in my.cnf.
- LOAD DATA INFILE in AUTOCOMMIT=1 mode no longer does implicit commits for each 1 MB of written binlog.
- Fixed a bug introduced in 4.0.4: LOCK TABLES ... READ LOCAL should not set row locks on the rows read. This caused deadlocks and lock wait timeouts in mysqldump.
- Fixed two bugs introduced in 4.0.4: in AUTO_INCREMENT, REPLACE could cause the counter to be left 1 too low. A deadlock or a lock wait timeout could cause the same problem.
- Fixed a bug: TRUNCATE on a TEMPORARY table crashed InnoDB.
- Fixed a bug introduced in 4.0.5: if binlogging was not switched on, INSERT INTO ... SELECT ... or CREATE TABLE ... SELECT ... could cause InnoDB to hang on a semaphore created in btr0sea.c, line 128. Workaround: switch binlogging on.
- Fixed a bug: in replication issuing SLAVE STOP in the middle of a multi-statement transaction could cause that SLAVE START would only perform a part of the transaction. A similar error could occur if the slave crashed and was restarted.

7.5.15.11 MySQL/InnoDB-3.23.54, December 12, 2002

- Fixed a bug: the InnoDB range estimator greatly exaggerated the size of a short index range if the paths to the endpoints of the range in the index tree happened to branch already in the root. This could cause unnecessary table scans in SQL queries.

- Fixed a bug: ORDER BY could fail if you had not created a primary key to a table, but had defined several indexes of which at least one was a UNIQUE index with all its columns declared as NOT NULL.
- Fixed a bug: a lock wait timeout in connection with ON DELETE CASCADE could cause corruption in indexes.
- Fixed a bug: if a SELECT was done with a unique key from a primary index, and the search matched to a delete-marked record, InnoDB could erroneously return the NEXT record.
- Fixed a bug introduced in 3.23.53: LOCK TABLES ... READ LOCAL should not set row locks on the rows read. This caused deadlocks and lock wait timeouts in mysqldump.
- Fixed a bug: if an index contains some column twice, and that column is updated, the table will become corrupt. From now on InnoDB prevents creation of such indexes.

7.5.15.12 MySQL/InnoDB-4.0.5, November 18, 2002

- InnoDB now supports also transaction isolation levels READ COMMITTED and READ UNCOMMITTED. READ COMMITTED more closely emulates Oracle and makes porting of applications from Oracle to MySQL easier.
- Deadlock resolution is now selective: we try to pick as victims transactions with less modified or inserted rows.
- FOREIGN KEY definitions are now aware of the lower_case_table_names setting in my.cnf.
- SHOW CREATE TABLE does not output the database name to a FOREIGN KEY definition if the referred table is in the same database as the table.
- InnoDB does a consistency check to most index pages before writing them to a data file.
- If you set innodb_force_recovery > 0, InnoDB tries to jump over corrupt index records and pages when doing SELECT * FROM table. This helps in dumping.
- InnoDB now again uses asynchronous unbuffered i/o in Windows 2000 and XP; only unbuffered simulated async i/o in NT, 95/98/ME.
- Fixed a bug: the InnoDB range estimator greatly exaggerated the size of a short index range if the paths to the endpoints of the range in the index tree happened to branch already in the root. This could cause unnecessary table scans in SQL queries. The fix will also be backported to 3.23.54.
- Fixed a bug present in 3.23.52, 4.0.3, 4.0.4: InnoDB startup could take very long or even crash on some Win 95/98/ME computers.
- Fixed a bug: the AUTO-INC lock was held to the end of the transaction if it was granted after a lock wait. This could cause unnecessary deadlocks.
- Fixed a bug: if SHOW INNODB STATUS, innodb_monitor, or innodb_lock_monitor had to print several hundred transactions in one report, and the output became truncated, InnoDB would hang, printing to the error log many waits for a mutex created at srv0srv.c, line 1621.
- Fixed a bug: SHOW INNODB STATUS on Unix always reported average file read size as 0 bytes.

- Fixed a potential bug in 4.0.4: InnoDB now does ORDER BY ... DESC like MyISAM.
- Fixed a bug: DROP TABLE could cause crash or a hang if there was a rollback concurrently running on the table. The fix will only be backported to 3.23 if this appears a real problem for users.
- Fixed a bug: ORDER BY could fail if you had not created a primary key to a table, but had defined several indexes of which at least one was a UNIQUE index with all its columns declared as NOT NULL.
- Fixed a bug: a lock wait timeout in connection with ON DELETE CASCADE could cause corruption in indexes.
- Fixed a bug: if a SELECT was done with a unique key from a primary index, and the search matched to a delete-marked record, InnoDB could return the NEXT record.
- Outstanding bugs: in 4.0.4 two bugs were introduced to AUTO_INCREMENT. REPLACE can cause the counter to be left 1 too low. A deadlock or a lock wait timeout can cause the same problem. These will be fixed in 4.0.6.

7.5.15.13 MySQL/InnoDB-3.23.53, October 9, 2002

- We again use unbuffered disk i/o to data files in Windows. Win XP and Win 2000 read performance seems to be very poor with normal i/o.
- Tuned range estimator so that index range scans are preferred over full index scans.
- Allow dropping and creating a table even if innodb_force_recovery is set. One can use this to drop a table which would cause a crash in rollback or purge, or if a failed table import causes a runaway rollback in recovery.
- Fixed a bug present in 3.23.52, 4.0.3, 4.0.4: InnoDB startup could take very long or even crash on some Win 95/98/ME computers.
- Fixed a bug: fast shutdown (which is the default) sometimes was slowed down by purge and insert buffer merge.
- Fixed a bug: doing a big SELECT from a table where no rows were visible in a consistent read could cause a very long (> 600 seconds) semaphore wait in btr0cur.c line 310.
- Fixed a bug: the AUTO-INC lock was held to the end of the transaction if it was granted after a lock wait. This could cause unnecessary deadlocks.
- Fixed a bug: if you created a temporary table inside LOCK TABLES, and used that temporary table, that caused an assertion failure in ha_innobase.cc.
- Fixed a bug: if SHOW INNODB STATUS, innodb_monitor, or innodb_lock_monitor had to print several hundred transactions in one report, and the output became truncated, InnoDB would hang, printing to the error log many waits for a mutex created at srv0srv.c, line 1621.
- Fixed a bug: SHOW INNODB STATUS on Unix always reported average file read size as 0 bytes.

7.5.15.14 MySQL/InnoDB-4.0.4, October 2, 2002

- We again use unbuffered disk i/o in Windows. Win XP and Win 2000 read performance seems to be very poor with normal i/o.

- Increased the max key length of InnoDB tables from 500 to 1024 bytes.
- Increased the table comment field in SHOW TABLE STATUS so that up to 16000 characters of foreign key definitions can be printed there.
- The auto-increment counter is no longer incremented if an insert of a row immediately fails in an error.
- Allow dropping and creating a table even if innodb_force_recovery is set. One can use this to drop a table which would cause a crash in rollback or purge, or if a failed table import causes a runaway rollback in recovery.
- Fixed a bug: Using ORDER BY primarykey DESC in 4.0.3 causes an assertion failure in btr0pcur.c, line 203.
- Fixed a bug: fast shutdown (which is the default) sometimes was slowed down by purge and insert buffer merge.
- Fixed a bug: doing a big SELECT from a table where no rows were visible in a consistent read could cause a very long (> 600 seconds) semaphore wait in btr0cur.c line 310.
- Fixed a bug: if the MySQL query cache was used, it did not get invalidated by a modification done by ON DELETE CASCADE or ...SET NULL.
- Fixed a bug: if you created a temporary table inside LOCK TABLES, and used that temporary table, that caused an assertion failure in ha_innodb.cc.
- Fixed a bug: if you set innodb_flush_log_at_trx_commit to 1, SHOW VARIABLES would show its value as 16 million.

7.5.15.15 MySQL/InnoDB-4.0.3, August 28, 2002

- Removed unnecessary deadlocks when inserts have to wait for a locking read, update, or delete to release its next-key lock.
- The MySQL HANDLER SQL commands now work also for InnoDB type tables. InnoDB does the HANDLER reads always as consistent reads. HANDLER is a direct access path to read individual indexes of tables. In some cases HANDLER can be used as a substitute of server-side cursors.
- Fixed a bug in 4.0.2: even a simple insert could crash the AIX version.
- Fixed a bug: if you used in a table name characters whose code is > 127, in DROP TABLE InnoDB could assert on line 155 of pars0sym.c.
- Compilation from source now provides a working version both on HP-UX-11 and HP-UX-10.20. The source of 4.0.2 worked only on 11, and the source of 3.23.52 only on 10.20.
- Fixed a bug: if compiled on 64-bit Solaris, InnoDB produced a bus error at startup.

7.5.15.16 MySQL/InnoDB-3.23.52, August 16, 2002

- The feature set of 3.23 will be frozen from this version on. New features will go the 4.0 branch, and only bug fixes will be made to the 3.23 branch.
- Many CPU-bound join queries now run faster. On Windows also many other CPU-bound queries run faster.

- A new SQL command `SHOW INNODB STATUS` returns the output of the InnoDB Monitor to the client. The InnoDB Monitor now prints detailed info on the latest detected deadlock.
- InnoDB made the SQL query optimiser to avoid too much index-only range scans and choose full table scans instead. This is now fixed.
- "BEGIN" and "COMMIT" are now added in the binlog around transactions. The MySQL replication now respects transaction borders: a user will no longer see half transactions in replication slaves.
- A replication slave now prints in crash recovery the last master binlog position it was able to recover to.
- A new setting `innodb_flush_log_at_trx_commit=2` makes InnoDB to write the log to the operating system file cache at each commit. This is almost as fast as the setting `innodb_flush_log_at_trx_commit=0`, and the setting 2 also has the nice feature that in a crash where the operating system does not crash, no committed transaction is lost. If the operating system crashes or there is a power outage, then the setting 2 is no safer than the setting 0.
- Added checksum fields to log blocks.
- `SET FOREIGN_KEY_CHECKS=0` helps in importing tables in an arbitrary order which does not respect the foreign key rules.
- `SET UNIQUE_CHECKS=0` speeds up table imports into InnoDB if you have UNIQUE constraints on secondary indexes.
- `SHOW TABLE STATUS` now lists also possible `ON DELETE CASCADE` or `ON DELETE SET NULL` in the comment field of the table.
- When `CHECK TABLE` is run on any InnoDB type table, it now checks also the adaptive hash index for all tables.
- If you defined `ON DELETE CASCADE` or `SET NULL` and updated the referenced key in the parent row, InnoDB deleted or updated the child row. This is now changed to conform to SQL-92: you get the error 'Cannot delete parent row'.
- Improved the auto-increment algorithm: now the first insert or `SHOW TABLE STATUS` initializes the auto-inc counter for the table. This removes almost all surprising deadlocks caused by `SHOW TABLE STATUS`.
- Aligned some buffers used in reading and writing to data files. This allows using unbuffered raw devices as data files in Linux.
- Fixed a bug: If you updated the primary key of a table so that only the case of characters changed, that could cause assertion failures, mostly in `page0page.ic` line 515.
- Fixed a bug: If you delete or update a row referenced in a foreign key constraint and the foreign key check has to wait for a lock, then the check may report an erroneous result. This affects also the `ON DELETE...` operation.
- Fixed a bug: A deadlock or a lock wait timeout error in InnoDB causes InnoDB to roll back the whole transaction, but MySQL could still write the earlier SQL statements to the binlog, even though InnoDB rolled them back. This could, for example, cause replicated databases to get out-of-sync.

- Fixed a bug: If the database happened to crash in the middle of a commit, then the recovery might leak tablespace pages.
- Fixed a bug: If you specified a non-latin1 character set in my.cnf, then, in contrary to what is stated in the manual, in a foreign key constraint a string type column had to have the same length specification in the referencing table and the referenced table.
- Fixed a bug: DROP TABLE or DROP DATABASE could fail if there simultaneously was a CREATE TABLE running.
- Fixed a bug: If you configured the buffer pool bigger than 2 GB in a 32-bit computer, InnoDB would assert in buf0buf.ic line 214.
- Fixed a bug: on 64-bit computers updating rows which contained the SQL NULL in some column could cause the undo log and the ordinary log to become corrupt.
- Fixed a bug: innodb_log_monitor caused a hang if it suppressed lock prints for a page.
- Fixed a bug: in the HP-UX-10.20 version mutexes would leak and cause race conditions and crashes in any part of InnoDB code.
- Fixed a bug: if you ran in the AUTOCOMMIT mode, executed a SELECT, and immediately after that a RENAME TABLE, then RENAME would fail and MySQL would complain about error 1192.
- Fixed a bug: if compiled on 64-bit Solaris, InnoDB produced a bus error at startup.

7.5.15.17 MySQL/InnoDB-4.0.2, July 10, 2002

- InnoDB is essentially the same as InnoDB-3.23.51.
- If no innodb_data_file_path is specified, InnoDB at the database creation now creates a 10 MB auto-extending data file ibdata1 to the datadir of MySQL. In 4.0.1 the file was 64 MB and not auto-extending.

7.5.15.18 MySQL/InnoDB-3.23.51, June 12, 2002

- Fixed a bug: a join could result in a seg fault in copying of a BLOB or TEXT column if some of the BLOB or TEXT columns in the table contained SQL NULL values.
- Fixed a bug: if you added self-referential foreign key constraints with ON DELETE CASCADE to tables and a row deletion caused InnoDB to attempt the deletion of the same row twice because of a cascading delete, then you got an assertion failure.
- Fixed a bug: if you use MySQL 'user level locks' and close a connection, then InnoDB may assert in ha_innobase.cc, line 302.

7.5.15.19 MySQL/InnoDB-3.23.50, April 23, 2002

- InnoDB now supports an auto-extending last data file. You do not need to preallocate the whole data file at the database startup.
- Made several changes to facilitate the use of the InnoDB Hot Backup tool. It is a separate non-free tool you can use to take online backups of your database without shutting down the server or setting any locks.
- If you want to run the InnoDB Hot Backup tool on an auto-extending data file you have to upgrade it to version ibbackup-0.35.

- The log scan phase in crash recovery will now run much faster.
- Starting from this server version, the hot backup tool truncates unused ends in the backup InnoDB data files.
- To allow the hot backup tool to work, on Windows we no longer use unbuffered i/o or native async i/o; instead we use the same simulated async i/o as on Unix.
- You can now define the ON DELETE CASCADE or ON DELETE SET NULL clause on foreign keys.
- FOREIGN KEY constraints now survive ALTER TABLE and CREATE INDEX.
- We suppress the FOREIGN KEY check if any of the column values in the foreign key or referenced key to be checked is the SQL NULL. This is compatible with Oracle, for example.
- SHOW CREATE TABLE now lists also foreign key constraints. Also mysqldump no longer forgets about foreign keys in table definitions.
- You can now add a new foreign key constraint with ALTER TABLE ... ADD CONSTRAINT FOREIGN KEY (...) REFERENCES ... (...).
- FOREIGN KEY definitions now allow backquotes around table and column names.
- MySQL command SET TRANSACTION ISOLATION LEVEL ... has now the following effect on InnoDB tables: if a transaction is defined as SERIALIZABLE then InnoDB conceptually adds LOCK IN SHARE MODE to all consistent reads. If a transaction is defined to have any other isolation level, then InnoDB obeys its default locking strategy which is REPEATABLE READ.
- SHOW TABLE STATUS no longer sets an x-lock at the end of an auto-increment index if the auto-increment counter has already been initialized. This removes in almost all cases the surprising deadlocks caused by SHOW TABLE STATUS.
- Fixed a bug: in a CREATE TABLE statement the string 'foreign' followed by a non-space character confused the FOREIGN KEY parser and caused table creation to fail with errno 150.

7.5.15.20 MySQL/InnoDB-3.23.49, February 17, 2002

- Fixed a bug: if you called DROP DATABASE for a database on which there simultaneously were running queries, the MySQL server could crash or hang. Crashes fixed, but a full fix has to wait some changes in the MySQL layer of code.
- Fixed a bug: on Windows one had to put the database name in lower case for DROP DATABASE to work. Fixed in 3.23.49: case no longer matters on Windows. On Unix the database name remains case-sensitive.
- Fixed a bug: if one defined a non-latin1 character set as the default character set, then definition of foreign key constraints could fail in an assertion failure in dict0crea.c, reporting an internal error 17.

7.5.15.21 MySQL/InnoDB-3.23.48, February 9, 2002

- Tuned the SQL optimiser to favor more often index searches over table scans.
- Fixed a performance problem when several large SELECT queries are run concurrently on a multiprocessor Linux computer. Large CPU-bound SELECT queries will now also generally run faster on all platforms.

- If MySQL binlogging is used, InnoDB now prints after crash recovery the latest MySQL binlog file name and the position in that file (= byte offset) InnoDB was able to recover to. This is useful, for example, when resynchronizing a master and a slave database in replication.
- Added better error messages to help in installation problems.
- One can now recover also MySQL temporary tables which have become orphaned inside the InnoDB tablespace.
- InnoDB now prevents a FOREIGN KEY declaration where the signedness is not the same in the referencing and referenced integer columns.
- Fixed a bug: calling SHOW CREATE TABLE or SHOW TABLE STATUS could cause memory corruption and make mysqld to crash. Especially at risk was mysqldump, because it calls frequently SHOW CREATE TABLE.
- Fixed a bug: if on Unix you did an ALTER TABLE to an InnoDB table and simultaneously did queries to it, mysqld could crash with an assertion failure in row0row.c, line 474.
- Fixed a bug: if inserts to several tables containing an auto-inc column were wrapped inside one LOCK TABLES, InnoDB asserted in lock0lock.c.
- In 3.23.47 we allowed several NULLS in a UNIQUE secondary index. But CHECK TABLE was not relaxed: it reports the table as corrupt. CHECK TABLE no longer complains in this situation.
- Fixed a bug: on Sparc and other high-endian processors SHOW VARIABLES showed innodb_flush_log_at_trx_commit and other boolean-valued startup parameters always OFF even if they were switched on.
- Fixed a bug: if you ran mysqld-max-nt as a service on Windows NT/2000, the service shutdown did not always wait long enough for the InnoDB shutdown to finish.

7.5.15.22 MySQL/InnoDB-3.23.47, December 28, 2001

- Recovery happens now faster, especially in a lightly loaded system, because background checkpointing has been made more frequent.
- InnoDB allows now several similar key values in a UNIQUE secondary index if those values contain SQL NULLS. Thus the convention is now the same as in MyISAM tables.
- InnoDB gives a better row count estimate for a table which contains BLOBs.
- In a FOREIGN KEY constraint InnoDB is now case-insensitive to column names, and in Windows also to table names.
- InnoDB allows a FOREIGN KEY column of CHAR type to refer to a column of VARCHAR type, and vice versa. MySQL silently changes the type of some columns between CHAR and VARCHAR, and these silent changes do not hinder FOREIGN KEY declaration any more.
- Recovery has been made more resilient to corruption of log files.
- Unnecessary statistics calculation has been removed from queries which generate a temporary table. Some ORDER BY and DISTINCT queries will now run much faster.

- MySQL now knows that the table scan of an InnoDB table is done through the primary key. This will save a sort in some ORDER BY queries.
- The maximum key length of InnoDB tables is again restricted to 500 bytes. The MySQL interpreter is not able to handle longer keys.
- The default value of `innodb_lock_wait_timeout` was changed from infinite to 50 seconds, the default value of `innodb_file_io_threads` from 9 to 4.

7.5.15.23 MySQL/InnoDB-4.0.1, December 23, 2001

- InnoDB is the same as in 3.23.47.
- In 4.0.0 the MySQL interpreter did not know the syntax LOCK IN SHARE MODE. This has been fixed.
- In 4.0.0 multi-table delete did not work for transactional tables. This has been fixed.

7.5.15.24 MySQL/InnoDB-3.23.46, November 30, 2001

- This is the same as 3.23.45.

7.5.15.25 MySQL/InnoDB-3.23.45, November 23, 2001

- This is a bugfix release.
- In versions 3.23.42-.44 when creating a table on Windows you have to use lower case letters in the database name to be able to access the table. Fixed in 3.23.45.
- InnoDB now flushes stdout and stderr every 10 seconds: if these are redirected to files, the file contents can be better viewed with an editor.
- Fixed an assertion failure in .44, in `trx0trx.c`, line 178 when you drop a table which has the `.frm` file but does not exist inside InnoDB.
- Fixed a bug in the insert buffer. The insert buffer tree could get into an inconsistent state, causing a crash, and also crashing the recovery. This bug could appear especially in large table imports or alterations.
- Fixed a bug in recovery: InnoDB could go into an infinite loop constantly printing a warning message that it cannot find free blocks from the buffer pool.
- Fixed a bug: when you created a temporary table of the InnoDB type, and then used ALTER TABLE to it, the MySQL server could crash.
- Prevented creation of MySQL system tables 'mysql.user', 'mysql.host', or 'mysql.db', in the InnoDB type.
- Fixed a bug which can cause an assertion failure in 3.23.44 in `srv0srv.c`, line 1728.

7.5.15.26 MySQL/InnoDB-3.23.44, November 2, 2001

- You can define foreign key constraints on InnoDB tables. An example: FOREIGN KEY (col1) REFERENCES table2(col2).
- You can create > 4 GB data files in those file systems that allow it.
- Improved InnoDB monitors, including a new `innodb_table_monitor` which allows you to print the contents of the InnoDB internal data dictionary.

- DROP DATABASE will now work also for InnoDB tables.
- Accent characters in the default character set latin1 will be ordered according to the MySQL ordering.
NOTE: if you are using latin1 and have inserted characters whose code is > 127 to an indexed CHAR column, you should run CHECK TABLE on your table when you upgrade to 3.23.43, and drop and reimport the table if CHECK TABLE reports an error!
- InnoDB will calculate better table cardinality estimates.
- Change in deadlock resolution: in .43 a deadlock rolls back only the SQL statement, in .44 it will roll back the whole transaction.
- Deadlock, lock wait timeout, and foreign key constraint violations (no parent row, child rows exist) now return native MySQL error codes 1213, 1205, 1216, 1217, respectively.
- A new my.cnf parameter innodb_thread_concurrency helps in performance tuning in high concurrency environments.
- A new my.cnf option innodb_force_recovery will help you in dumping tables from a corrupted database.
- A new my.cnf option innodb_fast_shutdown will speed up shutdown. Normally InnoDB does a full purge and an insert buffer merge at shutdown.
- Raised maximum key length to 7000 bytes from a previous limit of 500 bytes.
- Fixed a bug in replication of auto-inc columns with multiline inserts.
- Fixed a bug when the case of letters changes in an update of an indexed secondary column.
- Fixed a hang when there are > 24 data files.
- Fixed a crash when MAX(col) is selected from an empty table, and col is a not the first column in a multi-column index.
- Fixed a bug in purge which could cause crashes.

7.5.15.27 MySQL/InnoDB-3.23.43, October 4, 2001

- This is essentially the same as InnoDB-3.23.42.

7.5.15.28 MySQL/InnoDB-3.23.42, September 9, 2001

- Fixed a bug which corrupted the table if the primary key of a > 8000-byte row was updated.
- There are now 3 types of InnoDB Monitors: innodb_monitor, innodb_lock_monitor, and innodb_tablespace_monitor. innodb_monitor now prints also buffer pool hit rate and the total number of rows inserted, updated, deleted, read.
- Fixed a bug in RENAME TABLE.
- Fixed a bug in replication with an auto-increment column.

7.5.15.29 MySQL/InnoDB-3.23.41, August 13, 2001

- Support for < 4 GB rows. The previous limit was 8000 bytes.
- Use the doublewrite file flush method.

- Raw disk partitions supported as data files.
- InnoDB Monitor.
- Several hang bugs fixed and an ORDER BY bug ('Sort aborted') fixed.

7.5.15.30 MySQL/InnoDB-3.23.40, July 16, 2001

- Only a few rare bugs fixed.

7.5.15.31 MySQL/InnoDB-3.23.39, June 13, 2001

- CHECK TABLE now works for InnoDB tables.
- A new my.cnf parameter innodb_unix_file_flush_method introduced. It can be used to tune disk write performance.
- An auto-increment column now gets new values past the transaction mechanism. This saves CPU time and eliminates transaction deadlocks in new value assignment.
- Several bug fixes, most notably the rollback bug in 3.23.38.

7.5.15.32 MySQL/InnoDB-3.23.38, May 12, 2001

- The new syntax SELECT ... LOCK IN SHARE MODE is introduced.
- InnoDB now calls fsync after every disk write and calculates a checksum for every database page it writes or reads, which will reveal disk defects.
- Several bug fixes.

7.5.16 InnoDB Contact Information

Contact information of Innobase Oy, producer of the InnoDB engine. Web site: <http://www.innodb.com/>. E-mail: Heikki.Tuuri@innodb.com

phone: 358-9-6969 3250 (office) 358-40-5617367 (mobile)
Innobase Oy Inc.
World Trade Center Helsinki
Aleksanterinkatu 17
P.O.Box 800
00101 Helsinki
Finland

7.6 BDB or BerkeleyDB Tables

7.6.1 Overview of BDB Tables

BerkeleyDB, available at <http://www.sleepycat.com/> has provided MySQL with a transactional storage engine. Support for this storage engine is included in the MySQL source distribution starting from version 3.23.34 and is activated in the MySQL-Max binary. This storage engine is typically called BDB for short.

BDB tables may have a greater chance of surviving crashes and are also capable of COMMIT and ROLLBACK operations on transactions. The MySQL source distribution comes with a

BDB distribution that has a couple of small patches to make it work more smoothly with MySQL. You can't use a non-patched BDB version with MySQL.

We at MySQL AB are working in close cooperation with Sleepycat to keep the quality of the MySQL/BDB interface high.

When it comes to supporting BDB tables, we are committed to help our users to locate the problem and help creating a reproducible test case for any problems involving BDB tables. Any such test case will be forwarded to Sleepycat who in turn will help us find and fix the problem. As this is a two-stage operation, any problems with BDB tables may take a little longer for us to fix than for other storage engines. However, as the BerkeleyDB code itself has been used by many other applications than MySQL, we don't envision any big problems with this. See [\[Support\]](#), page [\[Support\]](#).

7.6.2 Installing BDB

If you have downloaded a binary version of MySQL that includes support for BerkeleyDB, simply follow the instructions for installing a binary version of MySQL. See [\[Installing binary\]](#), page [\[Installing binary\]](#). See `max-snt` [`mysqld-max`], page `max-pg`.

To compile MySQL with Berkeley DB support, download MySQL Version 3.23.34 or newer and configure MySQL with the `--with-berkeley-db` option. See [\[Installing source\]](#), page [\[Installing source\]](#).

```
cd /path/to/source/of/mysql-3.23.34
./configure --with-berkeley-db
```

Please refer to the manual provided with the BDB distribution for more updated information. Even though Berkeley DB is in itself very tested and reliable, the MySQL interface is still considered gamma quality. We are actively improving and optimising it to get it stable very soon.

7.6.3 BDB startup options

If you are running with `AUTOCOMMIT=0` then your changes in BDB tables will not be updated until you execute `COMMIT`. Instead of commit you can execute `ROLLBACK` to forget your changes. See [\[COMMIT\]](#), page [\[COMMIT\]](#).

If you are running with `AUTOCOMMIT=1` (the default), your changes will be committed immediately. You can start an extended transaction with the `BEGIN WORK SQL` command, after which your changes will not be committed until you execute `COMMIT` (or decide to `ROLLBACK` the changes).

The following options to `mysqld` can be used to change the behaviour of BDB tables:

Option	Description
<code>--bdb-home=directory</code>	Base directory for BDB tables. This should be the same directory you use for <code>--datadir</code> .
<code>--bdb-lock-detect=#</code>	Berkeley lock detect. One of (DEFAULT, OLDEST, RANDOM, or YOUNGEST).
<code>--bdb-logdir=directory</code>	Berkeley DB log file directory.
<code>--bdb-no-sync</code>	Don't synchronously flush logs.
<code>--bdb-no-recover</code>	Don't start Berkeley DB in recover mode.

```

--bdb-shared-data    Start Berkeley DB in multi-process mode (Don't
                    use DB_PRIVATE when initialising Berkeley DB)
--bdb-
tmpdir=directory    Berkeley DB temporary file directory.
--skip-bdb           Disable usage of BDB tables.
-O bdb_max_
lock=1000           Set the maximum number of locks possible. See
                    <undefined> [bdb_max_lock], page <undefined>.

```

If you use `--skip-bdb`, MySQL will not initialise the Berkeley DB library and this will save a lot of memory. Of course, you cannot use BDB tables if you are using this option. If you try to create a BDB table, MySQL will instead create a MyISAM table.

Normally you should start `mysqld` without `--bdb-no-recover` if you intend to use BDB tables. This may, however, give you problems when you try to start `mysqld` if the BDB log files are corrupted. See <undefined> [Starting server], page <undefined>.

With `bdb_max_lock` you can specify the maximum number of locks (10000 by default) you can have active on a BDB table. You should increase this if you get errors of type `bdb: Lock table is out of available locks` or `Got error 12 from ...` when you have do long transactions or when `mysqld` has to examine a lot of rows to calculate the query.

You may also want to change `binlog_cache_size` and `max_binlog_cache_size` if you are using big multi-line transactions. See <undefined> [COMMIT], page <undefined>.

7.6.4 Characteristics of BDB tables:

- To be able to rollback transactions, the BDB storage engine maintains log files. For maximum performance you should place these on another disk than your databases by using the `--bdb-logdir` option.
- MySQL performs a checkpoint each time a new BDB log file is started, and removes any log files that are not needed for current transactions. One can also run `FLUSH LOGS` at any time to checkpoint the Berkeley DB tables.

For disaster recovery, one should use table backups plus MySQL's binary log. See <undefined> [Backup], page <undefined>.

Warning: If you delete old log files that are in use, BDB will not be able to do recovery at all and you may lose data if something goes wrong.

- MySQL requires a `PRIMARY KEY` in each BDB table to be able to refer to previously read rows. If you don't create one, MySQL will create and maintain a hidden `PRIMARY KEY` for you. The hidden key has a length of 5 bytes and is incremented for each insert attempt.
- If all columns you access in a BDB table are part of the same index or part of the primary key, then MySQL can execute the query without having to access the actual row. In a MyISAM table the above holds only if the columns are part of the same index.
- The `PRIMARY KEY` will be faster than any other key, as the `PRIMARY KEY` is stored together with the row data. As the other keys are stored as the key data + the `PRIMARY KEY`, it's important to keep the `PRIMARY KEY` as short as possible to save disk and get better speed.
- `LOCK TABLES` works on BDB tables as with other tables. If you don't use `LOCK TABLE`, MySQL will issue an internal multiple-write lock on the table to ensure that the table will be properly locked if another thread issues a table lock.

- Internal locking in BDB tables is done on page level.
- `SELECT COUNT(*) FROM table_name` is slow as BDB tables doesn't maintain a count of the number of rows in the table.
- Sequential scanning is slower than with MyISAM tables as the data in BDB tables stored in B-trees and not in a separate datafile.
- The application must always be prepared to handle cases where any change of a BDB table may make an automatic rollback and any read may fail with a deadlock error.
- Keys are not prefix or suffix-compressed like keys in MyISAM tables. In other words, the key information will take a little more space in BDB tables compared to MyISAM tables.
- There are often holes in the BDB table to allow you to insert new rows in the middle of the key tree. This makes BDB tables somewhat larger than MyISAM tables.
- The optimiser needs to know an approximation of the number of rows in the table. MySQL solves this by counting inserts and maintaining this in a separate segment in each BDB table. If you don't issue a lot of `DELETE` or `ROLLBACK` statements, this number should be accurate enough for the MySQL optimiser, but as MySQL only stores the number on close, it may be incorrect if MySQL dies unexpectedly. It should not be fatal even if this number is not 100% correct. One can update the number of rows by executing `ANALYZE TABLE` or `OPTIMIZE TABLE`. See [\[ANALYZE TABLE\]](#), page [\[ANALYZE TABLE\]](#). See [\[OPTIMIZE TABLE\]](#), page [\[OPTIMIZE TABLE\]](#).
- If you get full disk with a BDB table, you will get an error (probably error 28) and the transaction should roll back. This is in contrast with MyISAM and ISAM tables where `mysqld` will wait for enough free disk before continuing.

7.6.5 Things we need to fix for BDB in the near future:

- It's very slow to open many BDB tables at the same time. If you are going to use BDB tables, you should not have a very big table cache (like >256) and you should use `--no-auto-rehash` with the `mysql` client. We plan to partly fix this in 4.0.
- `SHOW TABLE STATUS` doesn't yet provide that much information for BDB tables.
- Optimise performance.
- Change to not use page locks at all when we are scanning tables.

7.6.6 Operating systems supported by BDB

Currently we know that the BDB storage engine works with the following operating systems:

- Linux 2.x Intel
- Solaris SPARC
- Caldera (SCO) OpenServer
- Caldera (SCO) UnixWare 7.0.1

It doesn't work with the following operating systems:

- Linux 2.x Alpha
- Max OS X

Note: The above list is not complete; we will update it as we receive more information.

If you build MySQL with support for BDB tables and get the following error in the log file when you start `mysqld`:

```
bdb: architecture lacks fast mutexes: applications cannot be threaded
Can't init dtabases
```

This means that BDB tables are not supported for your architecture. In this case you must rebuild MySQL without BDB table support.

7.6.7 Restrictions on BDB Tables

Here follows the restrictions you have when using BDB tables:

- BDB tables store in the `.db` file the path to the file as it was created. (This was done to be able to detect locks in a multi-user environment that supports symlinks).
The effect of this is that BDB tables are not movable between directories!
- When taking backups of BDB tables, you have to either use `mysqldump` or take a backup of all `table_name.db` files and the BDB log files. The BDB log files are the files in the base data directory named `log.XXXXXXXXXX` (ten digits); The BDB storage engine stores unfinished transactions in the log files and requires these logs to be present when `mysqld` starts.

7.6.8 Errors That May Occur When Using BDB Tables

- If you get the following error in the `hostname.err` log when starting `mysqld`:

```
bdb: Ignoring log file: ../log.XXXXXXXXXX: unsupported log version #
```

it means that the new BDB version doesn't support the old log file format. In this case you have to delete all BDB logs from your database directory (the files with names that have the format `log.XXXXXXXXXX`) and restart `mysqld`. We would also recommend you to do a `mysqldump --opt` of your old BDB tables, delete the old tables, and restore the dump.
- If you are not running in auto-commit mode and delete a table that is referenced in another transaction, you may get the following error messages in your MySQL error log:

```
001119 23:43:56 bdb: Missing log fileid entry
001119 23:43:56 bdb: txn_abort: Log undo failed for LSN:
1 3644744: Invalid
```

This is not fatal but we don't recommend that you delete tables if you are not in auto-commit mode, until this problem is fixed (the fix is not trivial).

8 MySQL APIs

Este capítulo descreve as APIs disponíveis para o MySQL, onde consegui-las e como utilizá-las. A API C é a coberta mais estensamente, já que ela foi desenvolvida pela equipe do MySQL e é a base para a maioria das outras APIs.

8.1 API C do MySQL

O código da API C é distribuído com o MySQL. Ele está incluído na biblioteca `mysqlclient` e permite programas em C a fazer acesso em banco de dados.

Muitos dos clientes na distribuição fonte do MySQL está escrito em C. Se você estiver procurando por exemplos que demonstrem como utilizar a API C, dê uma olhada neste clientes. Você pode encontrá-los no diretório `clients` na distribuição fonte do MySQL.

A maioria das outras clientes API (todos exceto Connector/J) usam a biblioteca `mysqlclient` para se comunicar com o servidor MySQL. Isto significa que, por exemplo, você pode tirar vantagem das mesmas variáveis de ambientes que são utilizados por outros programas clientes, pois eles referenciados pela biblioteca. Veja [Side Scripts-snt \[Client-Side Scripts\]](#), [page Side Scripts-pg](#), para uma lista destas variáveis.

O cliente tem um tamanho máximo de buffer de comunicação. O tamanho do buffer que é alocado inicialmente (16K bytes) é automaticamente aumentado para o tamanho máximo (o máximo é 16M). Como o tamanho do buffer é aumentado somente como autorização de demanda, o simples aumento do limite máximo padrão não faz, por si só, que mais recursos sejam usado. Esta verificação de tamanho é na maioria verificações por consultas erradas e pacotes de comunicações.

O buffer de comunicação deve ser grande o suficiente para conter uma única instrução SQL (para tráfego cliente-servidor) e uma linha de dado retornado (para trafico servidor-cliente). Cada buffer de comunicação de thread é dinamicamente aumentado para manipular qualquer consulta ou linha até o limite máximo. Por exemplo, se você tiver valores BLOB que contenham até 16M de dados, você deve ter um limite de buffer de comunicação de pelo menos 16M (no servidor e no cliente). A máximo padrão do cliente '16M. mas o máximo padrão no servidor é 1M. Você pode aumentar iso alterando o valor do parâmetro `max_allowed_packet` quando o servidor é iniciado. See [\(undefined\) \[Par6ametros de servidor\]](#), [page \(undefined\)](#).

O servidor MySQL encolhe cada buffer de comunicação para `net_buffer_length` bytes depois de cada consulta. Para clientes, o tamanho do buffer associado com um conexão não é reduzido até que a conexão seja fechada, quando a memória de tempo do cliente é recuperada.

Para programação com threads, veja [\(undefined\) \[Threaded clients\]](#), [page \(undefined\)](#). Para criar uma aplicação stand-alone que inclua o "servidor" e o "cliente" no mesmo programa (e que não comunica com um servidor MySQL externo), veja [\(undefined\) \[libmysqld\]](#), [page \(undefined\)](#).

8.1.1 Tipos de Dados da API C

MYSQL Esta estrutura representa um manipulador para uma conexão ao banco de dados. É usada para quase todas as funções MySQL.

MYSQL_RES

Esta estrutura representa o resultado de uma consulta que retorna linhas (SELECT, SHOW, DESCRIBE, EXPLAIN). A informação retornada de uma consulta é chamada *conjunto de resultado* no resto desta seção.

MYSQL_ROW

Esta é uma representação segura de tipo de uma linha de dados. Ela é implementada atualmente como um vetor de strings de tamanho fixo (Você não pode tratá-los como strings terminadas com null se os valores do campo podem conter dados binários, porque tais valores podem conter um byte null internamente.). Linhas são obtidas pela chamada de `mysql_fetch_row()`.

MYSQL_FIELD

Esta estrutura contém informação sobre um campo, tais como nome, tipo e tamanho do campo. Seus membros são descritos em mais detalhes aqui. Você pode obter a estrutura `MYSQL_FIELD` para cada campo chamando `mysql_fetch_field()` repetidamente. Valores de campos não são parte desta estrutura; eles estão contidos na estrutura `MYSQL_ROW`.

MYSQL_FIELD_OFFSET

Esta é uma representação segura de um offset em uma lista de campos MySQL. (Usado por `mysql_field_seek()`.) Offsets são números de campos em um registro, começando com zero.

my_ulonglong

O tipo usado pelo número de linhas e para `mysql_affected_rows()`, `mysql_num_rows()`, e `mysql_insert_id()`. Este tipo fornece uma faixa de 0 a 1.84e19.

Em alguns sistemas, tentar imprimir um valor do tipo `my_ulonglong` não funcionará. Para imprimir tais valores, converta-os para `unsigned long` e use o formato de impressão `%lu`. Exemplo:

```
printf ("Número de linhas: %lu\n", (unsigned long) mysql_num_rows(resultado));
```

A estrutura `MYSQL_FIELD` contém os membros listados aqui:

char * name

O nome do campo, como um string terminada com null.

char * table

O nome da tabela contendo este campo, se não for um campo calculado. Para campos calculador, o valor `table` é uma string vazia.

char * def

O valor padrão para este campo, como um string terminada em null. Ele é atribuído apenas se você utilizar `mysql_list_fields()`.

enum enum_field_types tipo

O tipo do campo. O valor `tipo` pode ser um dos seguintes:

Valou tipo	Descrição do tipo
FIELD_TYPE_TINY	campo TINYINT

FIELD_TYPE_SHORT	campo SMALLINT
FIELD_TYPE_LONG	campo INTEGER
FIELD_TYPE_INT24	campo MEDIUMINT
FIELD_TYPE_LONGLONG	campo BIGINT
FIELD_TYPE_DECIMAL	campo DECIMAL ou NUMERIC
FIELD_TYPE_FLOAT	campo FLOAT
FIELD_TYPE_DOUBLE	campo DOUBLE ou REAL
FIELD_TYPE_TIMESTAMP	campo TIMESTAMP
FIELD_TYPE_DATE	campo DATE
FIELD_TYPE_TIME	campo TIME
FIELD_TYPE_DATETIME	campo DATETIME
FIELD_TYPE_YEAR	campo YEAR
FIELD_TYPE_STRING	campo String (CHAR ou VARCHAR)
FIELD_TYPE_BLOB	campo BLOB ou TEXT (usa <code>max_length</code> para determinar o tamanho máximo)
FIELD_TYPE_SET	campo SET
FIELD_TYPE_ENUM	campo ENUM
FIELD_TYPE_NULL	campo tipo-NULL
FIELD_TYPE_CHAR	Deprecado; use FIELD_TYPE_TINY

Você pode utilizar a macro `IS_NUM()` para testar se uma campo tem um tipo numérico. Passe o valor `tipo` para `IS_NUM()` e ele irá avaliar como VERDADEIRO (TRUE) se o campo for numérico:

```
if (IS_NUM(campo->tipo))
    printf("Campo é numérico\n");
```

`unsigned int length`

A largura de um campo, como especificado nas definições da tabela.

`unsigned int max_length`

A largura máxima do campo no conjunto de resultados (O tamanho do maior valor do campo para os registro no resultado atual). Se você utilizar `mysql_store_result()` ou `mysql_list_fields()`, ele contém o tamanho máximo para o campo. Se você utiliza `mysql_use_result()`, o valor desta variável é zero.

`unsigned int param`

Diferentes parâmetros binários para o campo. O valor de `param` pode ter zero ou mais dos seguintes conjunto de bits:

Valor param	Descrição param
NOT_NULL_FLAG	Campo não pode ser NULL
PRI_KEY_FLAG	Campo é parte de uma chave primária
UNIQUE_KEY_FLAG	Campo é parte de uma chave única
MULTIPLE_KEY_FLAG	Campo é parte de uma chave não única
UNSIGNED_FLAG	Campo tem o atributo UNSIGNED
ZEROFILL_FLAG	Campo tem o atributo ZEROFILL
BINARY_FLAG	Campo tem o atributo BINARY
AUTO_INCREMENT_FLAG	Campo tem o atributo AUTO_INCREMENT
ENUM_FLAG	Campo é um ENUM (obsoleto)

SET_FLAG	Campo é um SET (obsoleto)
BLOB_FLAG	Campo é um BLOB ou TEXT (obsoleto)
TIMESTAMP_FLAG	Campo é um TIMESTAMP (obsoleto)

Uso dos parâmetros BLOB_FLAG, ENUM_FLAG, SET_FLAG, e TIMESTAMP_FLAG foram obsoletos porque eles indicavam o tipo de um campo e não um atributo do tipo. É preferível testar `campo->tipo` para FIELD_TYPE_BLOB, FIELD_TYPE_ENUM, FIELD_TYPE_SET, ou FIELD_TYPE_TIMESTAMP.

O seguinte exemplo ilustra o uso típico do valor `param`:

```
if (campo->param & NOT_NULL_FLAG)
    printf("Campo não pode ser nulo\n");
```

Você pode usar as seguintes macros para determinar o status dos valores `param`:

Status param	Descrição
IS_NOT_NULL(param)	Verdadeiro se este campo é definido como NOT NULL
IS_PRI_KEY(param)	Verdadeiro de este campo é uma chave primária
IS_BLOB(param)	Verdadeiro se este campo é um BLOB ou TEXT (obsoleto; teste <code>campo->tipo</code>)

`unsigned int decimals`

O número de decimais para um campo numérico.

8.1.2 Visão Geral das Função da API C

As funções disponíveis na API C são listadas aqui e descritas em maiores detalhes em uma seção posterior. See [\[Funções API C\]](#), page [\[undefined\]](#).

Função	Descrição
<code>mysql_affected_rows()</code>	Retorna o número de linhas alteradas/deletadas/insweridas pela última consulta \ UPDATE, DELETE, ou INSERT.
<code>mysql_change_user()</code>	Muda o usuario em um banco de dados em uma conexão aberta.
<code>mysql_character_set_name()</code>	Retorna o nome do conjunto de carcters padrão para a conexão.
<code>mysql_close()</code>	Fecha ua conexão com o servidor
<code>mysql_connect()</code>	Se conecta ao servidro MySQL. Esta função está deprecad; utilize <code>mysql_real_connect()</code> .
<code>mysql_create_db()</code>	Cria um banco de dados. Esta função está obsoleta; utiliza o comando SQL CREATE DATABASE.
<code>mysql_data_seek()</code>	Busca por uma linha arbitrária em um conjunto de resultados de uma consulta.
<code>mysql_debug()</code>	Faz um DEBUG_PUSH com a string dada.

mysql_drop_db()	Apaga um banco de dados; Esta função esta obsoleta; utiliza o comando SQL DROP DATABASE.
mysql_dump_debug_info()	Faz o servidor escrever informações de depuração no log.
mysql_eof()	Determina quando a ultima linha de um conjunto de resultados foi lida. Esta função foi obsoleta; Utilize <code>mysql_errno()</code> ou <code>mysql_error()</code>
mysql_errno()	Retorna o número de erro para a função MySQL chamada mais recentemente.
mysql_error()	Retorna a mensagem de erro para função MySQL chamada mais recentemente.
mysql_escape_string()	Escapa caracteres especiais em uma string para ser usada em uma instrução SQL.
mysql_fetch_field()	Retorna o tipo do próximo campo na tabela.
mysql_fetch_field_direct()	Retorna o tipo de um campo da tabela, dado um número do campo.
mysql_fetch_fields()	Retorna um vetor de todas as estruturas do campo.
mysql_fetch_lengths()	Retorna o tamanho de todas as colunas na linha atual.
mysql_fetch_row()	Busca o próximo registro no conjunto de resultados.
mysql_field_seek()	Coloca o cursor da coluna em uma coluna específica.
mysql_field_count()	Retorna o número de colunas resultantes da consulta mais recente.
mysql_field_tell()	Retorna a posição do cursos de campos usado pelo último <code>mysql_fetch_field()</code> .
mysql_free_result()	Libera a memória usada por um conjunto de resultados.
mysql_get_client_info()	Retorna a versão do cliente.
mysql_get_host_info()	Retorna uma string descrevendo a conexão.
mysql_get_server_version()	Retorna o número da versão do servidor como um inteiro (Novo na versão 4.1)
mysql_get_proto_info()	Retorna a versão do protocolo usado para a conexão.
mysql_get_server_info()	Retorna o número da versão do servidor.
mysql_info()	Retorna informação sobre a consulta executada mais recentemente.
mysql_init()	Obtem ou inicializa uma estrutura MYSQL.

mysql_insert_id()	Retorna o ID gerado para uma coluna <code>AUTO_INCREMENT</code> pela consulta anterior.
mysql_kill()	Mata uma thread dada.
mysql_list_dbs()	Retorna o nome do banco de dados correspondente a uma expressão regular.
mysql_list_fields()	retorna nome de campos coincidindo com uma expressão regular.
mysql_list_processes()	Retorna uma lista das threads atuais do servidor.
mysql_list_tables()	Retorna os nomes de tabelas correspondente a uma expressão regular.
mysql_num_fields()	Retorna o número de colunas em um conjunto de resultados.
mysql_num_rows()	Retorna o número de linhas em um conjunto de resultados.
mysql_options()	Define opções de conexão para <code>mysql_connect()</code> .
mysql_ping()	Verifica se a conexão ao servidor está funcionando, reconectando se necessário.
mysql_query()	Executa uma consulta SQL especificada com uma string terminada com <code>null</code> .
mysql_real_connect()	Conecta ao servidor MySQL.
mysql_real_escape_string()	Escapa caracteres especiais em uma string para ser utilizada em uma instrução SQL, olhando na conta o conjunto de caracteres atual da conexão
mysql_real_query()	Executa uma consulta SQL especificada como uma string fixa.
mysql_reload()	Diz ao servidor pra recarregar a tabela de permissões
mysql_row_seek()	Busca por uma linha no resultado, usando o valor retornado de <code>mysql_row_tell()</code> .
mysql_row_tell()	Retorna a posição do cursor de linhas.
mysql_select_db()	Seleciona um banco de dados.
mysql_shutdown()	Desliga o servidor de banco de dados.
mysql_stat()	Retorna o status do servidor como uma string.
mysql_store_result()	Recupera um resultado completo para o cliente.
mysql_thread_id()	Retorna a identificação da thread atual.
mysql_thread_safe()	Retorna 1 se o cliente foi compilado como thread-safe.

mysql_use_result()	Inicia uma resultado recuperado registro por registro.
mysql_commit()	Faz um commits na transação (novo na versão 4.1).
mysql_rollback()	Faz um roll back na transação (novo na versão 4.1).
mysql_autocommit()	Muda o modo autocommit em ligado/desligado (novo na versão 4.1).
mysql_more_results()	Verifica se não existem mais resultados (novo na versão 4.1).
mysql_next_result()	Retorna/Inicia o próximo resultado em execuções consultas múltiplas (novo na versão 4.1).

Para se conectar ao servidor, chame `mysql_init()` para iniciar um manipulador de conexão, então chame `mysql_real_connect()` com este manipulador (com informações de nome de máquina, usuários e senha). Conectado, `mysql_real_connect()` define o parâmetro `reconnect` (parte da estrutura `MYSQL`) para um valor de 1. Este parâmetro indica, no evento que uma consulta não pode ser realizada por perda de conexão, para tentar reconectar ao servidor ao antes de desistir. Quando não precisar mais da conexão, chame `mysql_close()` para terminá-la.

Enquanto a conexão estiver ativa, o cliente pode enviar consultas SQL para o servidor usando `mysql_query()` ou `mysql_real_query()`. A diferença entre os dois é que `mysql_query()` espera que a consulta seja especificada como uma string terminada em null, enquanto `mysql_real_query()` espera um string de tamanho fixa. Se a string conter dados binários (a qual pode incluir bytes null), você deve usar `mysql_real_query()`.

Para cada consulta não-`SELECT` (por exemplo, `INSERT`, `UPDATE`, `DELETE`), você pode descobrir quantas linhas foram alteradas (afetadas) chamando `mysql_affected_rows()`.

Para consultas `SELECT`, você retorna os registros selecionados como um resultado. (Note que algumas intruções são como a `SELECT` ao retornar registros. Elas incluem `SHOW`, `DESCRIBE` e `EXPLAIN`. elas devem ser tratadas da mesma maneira que instruções `SELECT`.)

Existem dois modos para um cliente processar o resultado. Um modo é recuperar todo o resultado de uma vez chamando `mysql_store_result()`. Esta função busca no servidor todas as linhas retornadas pela consulta e as armazena no cliente. O segundo modo é o cliente iniciar um retorno do resultado registro por registro chamando `mysql_use_result()`. Esta função inicia o retorno, mas não busca realmente nenhuma linha do servidor.

Em ambos os casos, acesse registros chamando `mysql_fetch_row()`. Com `mysql_store_result()`, `mysql_fetch_row()` acessa registros que já tenham sido buscado do servidor. Com `mysql_use_result()`, `mysql_fetch_row()` recupera, na verdade, o registro do servidor. Informações sobre o tamanho dos dados em cada registro é disponível pela chamada `mysql_fetch_lengths()`.

Depois de finalizar o uso do resultado, chame `mysql_free_result()` para liberar a memória usada por ele.

Os dois mecanismos de recuperação são complementares. Programas clientes devem escolher a abordagem mais apropriada para suas necessidades. Na prática, clientes tendem a utilizar `mysql_store_result()`.

Uma vantagem de `mysql_store_result()` é que pelo fato de todos os registros serem trazidos para o cliente, você não só pode acessar registros sequencialmente, mas também

pode mover para `tarz` e para frente no resultado utilizando `mysql_data_seek()` ou `mysql_row_seek()` para alterar a posição atual do registro no resultado. Você também pode saber quantas linhas existem chamando `mysql_num_rows()`. Por outro lado, a necessidade de memória para `mysql_store_result()` pode ser muito alta para resultados grandes e você encontrará como mais facilidade condições de estouro de memória.

Uma vantagem de `mysql_use_result()` é que o cliente exige menos memória para o resultado porque ele mantém apenas um registro por vez (por haver menor sobrecarga de alocação, `mysql_use_result()` pode ser mais rápido). As desvantagens são que você deve processar cada registro rapidamente para evitar prender o servidor, você não tem acesso aleatório aos registros no resultado (você só pode acessá-los sequencialmente) e você não sabe quantos registros existem no resultado até que você recupere todos eles. Além disso, você **deve** recuperar todos os registros mesmo que você já tenha encontrado a informação que procura antes de finalizar o conjunto de resultados.

A API torna possível para os clientes responder apropriadamente as consultas (recuperando somente os registros necessários) sem saber se a consulta é uma instrução `SELECT` ou não. Você pode fazer isto chamando `mysql_store_result()` depois de cada `mysql_query()` (ou `mysql_real_query()`). Se o resultado for obtido com sucesso, a consulta foi um `SELECT` e você pode ler os registros. Se a obtenção do resultado falhar, chame `mysql_field_count()` para determinar se o resultado era o esperado. Se `mysql_field_count()` retornar zero, a consulta não retornou nenhum dado (indicando que ela era um `INSERT`, `UPDATE`, `DELETE`, etc.), e não era esperado que retornasse registros. Se `mysql_field_count()` é diferente de zero, a consulta deveria retornar registros, mas não o fez. Isto indica que a consulta foi um `SELECT` que falhou. Veja a descrição de `mysql_field_count()` para um exemplo de como deve ser feito.

`mysql_store_result()` e `mysql_use_result()` permitem que você obtenha informação sobre os campos que montam o resultado (o número de campos, os seus nomes e tipos, etc.) Você pode acessar informações de campo sequencialmente dentro dos registros chamando `mysql_fetch_field()` repetidamente, ou pelo número do campo dentro do registro chamando `mysql_fetch_field_direct()`. A posição atual do cursor de campos pode ser alterada chamando `mysql_field_seek()`. Definir o cursor de campo afeta chamadas subsequentes de `mysql_fetch_field()`. Você também pode conseguir informações de todos os campos de uma só vez chamando `mysql_fetch_fields()`.

Para detectar e relatar problemas, o MySQL fornece acesso a informações de erro através das funções `mysql_errno()` e `mysql_error()`. Elas retornam o código de erro ou a mensagem de erro para a função chamada mais recentemente que tenha tido sucesso ou que tenha falhado, permitindo a você determinar quando um erro ocorreu e qual foi ele.

8.1.3 Descrição das Funções da API C

Nas descrições a seguir, um parâmetro ou valor retornado `NULL` significa `NULL` no sentido da linguagem de programação C, não um valor `NULL` do MySQL.

Funções que retornam um valor geralmente retornam um ponteiro ou um inteiro. A menos que seja especificado, funções que retornam um ponteiro, retornam um valor diferente de `NULL` para indicar sucesso ou um valor `NULL` para indicar um erro, e funções que retornam um inteiro, retornam zero para indicar sucesso ou um valor diferente de zero para indicar

um erro. A menos que a descrição da função diga algo diferente, não faça teste com outro valor além do zero.

```

if (result)                /* correct */
    ... error ...

if (result < 0)            /* incorrect */
    ... error ...

if (result == -1)         /* incorrect */
    ... error ...

```

Quando uma função retornar um erro, a subsecao **Erros** de descrição de funções lista os possíveis tipos de erro. Você pode descobrir quais deles ocorreu chamando `mysql_errno()`. Uma representação string do erro pode ser obtida chamando `mysql_error()`.

8.1.3.1 `mysql_affected_rows()`

```
my_ulonglong mysql_affected_rows(MYSQL *mysql)
```

Descrição

Retorna o número de registros alterados pelo último UPDATE, deletados pelo último DELETE ou inseridos pelo último INSERT. Pode ser chamado imediatamente após `mysql_query()` para instruções UPDATE, DELETE, ou INSERT. Para instruções SELECT, `mysql_affected_rows()` funciona como `mysql_num_rows()`.

Valor Retornado

Um inteiro maior que zero indica o número de registros afetados ou recuperados. Zero indica que nenhum registro foi atualizado por uma instrução UPDATE, nenhuma linha foi encontrada pela cláusula WHERE na consulta ou a consulta ainda não foi executada. -1 indica que a consulta retornou um erro ou que, para uma consulta SELECT, `mysql_affected_rows()` foi chamado antes da chamada `mysql_store_result()`.

Erros

Nenhum.

Exemplo

```
mysql_query(&mysql,"UPDATE products SET cost=cost*1.25 WHERE group=10");
printf("%ld products updated",(long) mysql_affected_rows(&mysql));
```

Se se for especificado o parâmetro CLIENT_FOUND_ROWS ao conectar no `mysqld`, `mysql_affected_rows()` retornará o número de linhas encontradas pela cláusula WHERE para a instrução UPDATE.

Note que quando for utilizado um comando REPLACE, `mysql_affected_rows()` retornará 2 se o novo registro substituir um mais antigo. Isto é porque neste caso um registro foi inserido e depois os registros duplicados foram deletados.

8.1.3.2 `mysql_change_user()`

```
my_bool mysql_change_user(MYSQL *mysql, const char *user, const char
*password, const char *db)
```

Descrição

Altera o usuário e faz com que o banco de dados especificado por `db` se torne o banco de dados padrão (atual) na conexão especificada por `mysql`. Em consultas subsequentes este banco de dados é o padrão para referências a tabelas que não especificam o banco de dados explicitamente.

Esta função foi introduzida na versão do MySQL.

`mysql_change_user()` falha a menos que o usuário conectado possa ser autenticado ou se ele não tiver permissão para utilizar o banco de dados. Neste caso o usuário e o banco de dados não são alterados.

O parâmetro `db` pode ser definido como `NULL` se você não deseja ter um banco de dados padrão.

A partir da versão 4.0.6 do MySQL este comando sempre fará `ROLLBACK` de qualquer transação ativa, fecha todas as tabelas temporárias, destrói todas as tabelas bloqueadas e volta a um estado como se tivesse feito uma nova conexão. Isto irá acontecer mesmo se o usuário não foi alterado.

Valor Retornado

Zero se obteve sucesso. Diferente de zero se ocorreu um erro.

Erros

O mesmo que pode ser obtido com `mysql_real_connect()`.

`CR_COMMANDS_OUT_OF_SYNC`

Comandos foram executados em ordem inapropriada.

`CR_SERVER_GONE_ERROR`

O servidor MySQL finalizou.

`CR_SERVER_LOST`

A conexão ao servidor foi perdida durante a consulta.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

`ER_UNKNOWN_COM_ERROR`

O servidor MySQL não possui este comando (provavelmente um versão mais antiga)

`ER_ACCESS_DENIED_ERROR`

O usuário ou a senha estavam errados.

`ER_BAD_DB_ERROR`

O banco de dados não existe.

ER_DBACCESS_DENIED_ERROR

O usuário não tem direitos de acesso a este banco de dados.

ER_WRONG_DB_NAME

O nome de banco de dados é muito grande.

Exemplo

```
if (mysql_change_user(&mysql, "user", "password", "new_database"))
{
    fprintf(stderr, "Failed to change user. Error: %s\n",
            mysql_error(&mysql));
}
```

8.1.3.3 mysql_character_set_name()

```
const char *mysql_character_set_name(MYSQL *mysql)
```

Descrição

Retorna o conjunto de caracteres padrão para a conexão atual.

Valor Retornado

O conjunto de caracteres padrão

Erros

Nenhum.

8.1.3.4 mysql_close()

```
void mysql_close(MYSQL *mysql)
```

Descrição

Fecha uma conexão aberta anteriormente. `mysql_close()` também desaloca o ponteiro do manipulador da conexão para o `mysql` se ele tiver sido alocado automaticamente por `mysql_init()` ou `mysql_connect()`.

Valor Retornado

Nenhum.

Erros

Nenhum.

8.1.3.5 mysql_connect()

```
MYSQL *mysql_connect(MYSQL *mysql, const char *host, const char *user, const
char *passwd)
```

Descrição

A função está obsoleta. É melhor utilizar `mysql_real_connect()`.

`mysql_connect()` tenta estabelecer uma conexão a um banco de dados MySQL executando em `host`. `mysql_connect()` deve completar com sucesso antes que você possa executar qualquer uma das função da API, com a exceção de `mysql_get_client_info()`.

O significado dos parâmetros são os mesmos que os parâmetros correspondentes para `mysql_real_connect()` com a diferença que o parâmetro de conexão pode ser NULL. Neste caso a API C aloca memória para a estrutura de conexão automaticamente e a libera quando você chamar `mysql_close()`. A desvantagem desta abordagem é que você não pode retornar uma mensagem de erro se a conexão falhar. (Para obter informações de erro de `mysql_errno()` ou `mysql_error()`, você deve fornecer um ponteiro MYSQL válido.)

Valor Retornado

O mesmo de `mysql_real_connect()`.

Erros

O mesmo de `mysql_real_connect()`.

8.1.3.6 `mysql_create_db()`

```
int mysql_create_db(MYSQL *mysql, const char *db)
```

Descrição

Cria o banco de dados nomeado pelo parâmetro `db`.

Esta função está obsoleta. É melhor utilizar `mysql_query()` para comandar uma instrução SQL `CREATE DATABASE`.

Valor Retornado

Zero se o banco de dados foi criado com sucesso. Diferente de zero se ocorreu um erro.

Erros

`CR_COMMANDS_OUT_OF_SYNC`

Os comando foram executados em uma ordem inapropriada.

`CR_SERVER_GONE_ERROR`

O servidor MySQL foi finalizado.

`CR_SERVER_LOST`

A conexão ao servidor MySQL foi perdida durante a consulta.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

Exemplo

```

if(mysql_create_db(&mysql, "my_database"))
{
    fprintf(stderr, "Failed to create new database.  Error: %s\n",
            mysql_error(&mysql));
}

```

8.1.3.7 mysql_data_seek()

```
void mysql_data_seek(MYSQL_RES *result, my_ulonglong offset)
```

Descrição

Busca um registro arbitrário em um resultado de uma consulta. Ela exige que a estrutura do resultado contenha todo o resultado da consulta, assim `mysql_data_seek()` só pode ser usado em conjunto com `mysql_store_result()`, não com `mysql_use_result()`.

O offset deve der um valor na faixa de 0 a `mysql_num_rows(result)-1`.

Valor Retornado

Nenhum.

Erros

Nenhum.

8.1.3.8 mysql_debug()

```
void mysql_debug(const char *debug)
```

Descrição

Faz um `DEBUG_PUSH` com a string dada. `mysql_debug()` usa a biblioteca de depuração Fred Fish. Para utilizar esta função você deve compilar a biblioteca cliente para suportar depuração. See [\(undefined\) \[Depurando o servidor\]](#), page [\(undefined\)](#). See [\(undefined\) \[Depurando o cliente\]](#), page [\(undefined\)](#).

Valor Retornado

Nenhum.

Erros

Nenhum.

Exemplo

A chamada mostrada aqui faz com que a biblioteca cliente gere um arquivo de rastreamento `‘/tmp/client.trace’` na máquina cliente:

```
mysql_debug("d:t:0,/tmp/client.trace");
```

8.1.3.9 `mysql_drop_db()`

```
int mysql_drop_db(MYSQL *mysql, const char *db)
```

Descrição

Apaga o banco de dados nomeado pelo parâmetro `db`.

Esta função está obsoleta. É melhor utilizar `mysql_query()` para realizar uma instrução SQL `DROP DATABASE`.

Valor Retornado

Zero se o banco de dados foi apagado com sucesso. Diferente de zero ocorreu um erro.

Erros

`CR_COMMANDS_OUT_OF_SYNC`

Os comando foram executados em uma ordem inapropriada.

`CR_SERVER_GONE_ERROR`

O servidor MySQL foi finalizado.

`CR_SERVER_LOST`

A conexão ao servidor MySQL foi perdida durante a consulta.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

Exemplo

```
if(mysql_drop_db(&mysql, "my_database"))
    fprintf(stderr, "Failed to drop the database: Error: %s\n",
            mysql_error(&mysql));
```

8.1.3.10 `mysql_dump_debug_info()`

```
int mysql_dump_debug_info(MYSQL *mysql)
```

Descrição

Instrui o servidor a gravar algumas informações de depuração no log. Para funcionar, o usuário conectado deve ter privilégio `SUPER`.

Valor Retornado

Zero se o comando obteve sucesso. Diferete de zero se ocorreu um erro.

Erros

CR_COMMANDS_OUT_OF_SYNC

Os comando foram executados em uma ordem inapropriada.

CR_SERVER_GONE_ERROR

O servidor MySQL foi finalizado.

CR_SERVER_LOST

A conexão ao servidor MySQL foi perdida durante a consulta.

CR_UNKNOWN_ERROR

Um erro desconhecido ocorreu.

8.1.3.11 `mysql_eof()`

```
my_bool mysql_eof(MYSQL_RES *result)
```

Descrição

Esta função está obsoleta. `mysql_errno()` ou `mysql_error()` podem ser usados em seu lugar.

`mysql_eof()` determina se o último registro de um resultado foi lido.

Se você buscar um resultado com um chamada `mysql_store_result()` bem sucedida, o cliente recebe todo o resultado em uma operação. Neste caso, é um valor NULL retornado de `mysql_fetch_row()` sempre significa que o fim do resultado foi atingido e não é necessário chamar `mysql_eof()`. Quando usado com `mysql_store_result()`, `mysql_eof()` sempre retornará verdadeiro.

Por outro lado, se você utilizar `mysql_use_result()` para iniciar um resultado recuperado, as linhas do conjunto são obtido do servidor uma a uma, chamando `mysql_fetch_row()` repetidamente. Como pode ocorrer um erro na conexão durante este processo, um valor NULL retornado de `mysql_fetch_row()` não significa, necessariamente, que o fim do resultado fo atingido normalmente. Neste caso, você pode utilizar `mysql_eof()` para determinar o que aconteceu. `mysql_eof()` retorna um valor diferente de zero se o fim do resultaod foi atingido e zero se ocorreu um erro.

Historicamente, `mysql_eof()` é preterido pelas funções de erro padrão do MySQL `mysql_errno()` e `mysql_error()`. Como estas funções de erro fornecem a mesma informação, o uso das duas últimas é preferido sobre `mysql_eof()`, a qual está obsoleta. (De fato, elas fornecem mais informações, porque `mysql_eof()` retorna apenas um valor booleano enquanto as funções de erro indicam uma razão para a ocorrência do erro quando ele ocorre).

Valor Retornado

Zero se nenhum erro ocorreu. Diferente de zero o fim do resultado foi atingido.

Erros

Nenhum.

Exemplo

Os exemplos seguintes mostram como você deve usar `mysql_eof()`:

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // faz algo com os dados
}
if(!mysql_eof(result)) // mysql_fetch_row() falha devido a um erro
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

No entanto, você pode conseguir o mesmo efeito com as funções de erro padrões do MySQL:

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // faz algo com os dados
}
if(mysql_errno(&mysql)) // mysql_fetch_row() falha devido a um erro
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

8.1.3.12 `mysql_errno()`

```
unsigned int mysql_errno(MYSQL *mysql)
```

Descrição

Para a conexão especificada pelo `mysql`, `mysql_errno()` retorna o código de erro para a função API chamada mais recentemente que tenha obtido sucesso ou falhado. Um valor de retorno de zero significa que um erro ocorreu. Números de mensagens de erro de clientes são listados no arquivo de cabeçalho `'errmsg.h'` do MySQL. Números de mensagem de erros do servidor são listados no arquivo `'mysqld_error.h'`. Na distribuição fonte do MySQL você pode encontrar uma lista completa de números de mensagens de erro no arquivo `'Docs/mysqld_error.txt'`.

Valor Retornado

Um valor com código de erro. Zero se nenhum erro ocorrer.

Erros

Nenhum.

8.1.3.13 `mysql_error()`

```
char *mysql_error(MYSQL *mysql)
```

Descrição

Para a conexão especificada por `mysql`, `mysql_error()` retorna a mensagem de erro para a função de API chamada mais recentemente API que tenha falhado ou não. Uma string vazia ("") é retornada se nenhum erro ocorrer. OIsto significa que os dois testes seguintes são equivalentes:

```
if(mysql_errno(&mysql))
{
    // ocorreu um erro
}

if(mysql_error(&mysql)[0] != '\0')
{
    // ocorreu um erro
}
```

A língua da mensagem de erro do cliente pode ser alterada recompilando a biblioteca do cliente MySQL. Atualmente você pode escolher mensagens de erro em várias línguas diferentes. See [\[Languages\]](#), page [\[Languages\]](#).

Valor Retornado

Uma string que descreve um erro. Uma string vazia se nenhum erro ocorrer.

Erros

Nenhum.

8.1.3.14 `mysql_escape_string()`

Você deve usar `mysql_real_escape_string()` em seu lugar!

Esta função é idêntica a `mysql_real_escape_string()` exceto que `mysql_real_escape_string()` pega um manipulador de conexão como seu primeiro argumento e escapa a string de acordo com a conjunto de caracteres padrão. `mysql_escape_string()` não utiliza um argumento de conexão e não respeita o conjunto de caracteres atual.

8.1.3.15 `mysql_fetch_field()`

```
MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result)
```

Descrição

Retorna a definição de uma coluna de um resultado como uma estrutura `MYSQL_FIELD`. Chame esta função repetidamente para retornar informações sobre todas as colunas no resultado. `mysql_fetch_field()` retorna `NULL` quando não existirem mais campos.

`mysql_fetch_field()` é definido para retornar a informação do primeiro campo cada vez que você executar uma nova consulta `SELECT`. O campo retornado por `mysql_fetch_field()` também é afetado pela chamadas `mysql_field_seek()`.

Se você tiver chamado `mysql_query()` para realizar um `SELECT` em uma tabela mas não tiver chamado `mysql_store_result()`, MySQL retorna o tamanho padrão do blob (8K bytes) quando chamar `mysql_fetch_field()` para saber o tamanho de um campo BLOB. (O tamanho de 8 k é escolhido porque o MySQL não sabe o tamanho máximo do BLOB. Ele pode ser configurado algumas vezes.) Uma vez retornado o resultado, `campo->tamanho_max` contém o tamanho da maior valor para esta coluna em uma consulta específica.

Valor Retornado

A estrutura `MYSQL_FIELD` para a coluna atual. NULL não houver mais colunas.

Erros

Nenhum.

Exemplo

```
MYSQL_FIELD *field;

while((field = mysql_fetch_field(result)))
{
    printf("field name %s\n", field->name);
}
```

8.1.3.16 mysql_fetch_fields()

```
MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)
```

Descrição

Retorna um vetor de todas as estruturas `MYSQL_FIELD` no resultado. Cada estrutura fornece a definição do campo para uma coluna do resultado.

Valor Retornado

Um vetor da estrutura `MYSQL_FIELD` para todas as colunas no resultado.

Erros

Nenhum.

Exemplo

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *fields;

num_fields = mysql_num_fields(result);
fields = mysql_fetch_fields(result);
for(i = 0; i < num_fields; i++)
```

```

{
    printf("Field %u is %s\n", i, fields[i].name);
}

```

8.1.3.17 `mysql_fetch_field_direct()`

```

MYSQL_FIELD *mysql_fetch_field_direct(MYSQL_RES *result, unsigned int
fieldnr)

```

Descrição

Dado um número de campo `fieldnr` para uma coluna em resultado, retorna a informação de campo daquela coluna como uma estrutura `MYSQL_FIELD`. Você pode utilizar esta função para retornar a definição para uma coluna arbitrária. O valor de `fieldnr` deve estar na faixa de 0 a `mysql_num_fields(result)-1`.

Valor Retornado

A estrutura `MYSQL_FIELD` para uma coluna específica.

Erros

Nenhum.

Exemplo

```

unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *field;

num_fields = mysql_num_fields(result);
for(i = 0; i < num_fields; i++)
{
    field = mysql_fetch_field_direct(result, i);
    printf("Field %u is %s\n", i, field->name);
}

```

8.1.3.18 `mysql_fetch_lengths()`

```

unsigned long *mysql_fetch_lengths(MYSQL_RES *result)

```

Descrição

Retorna o tamanho da coluna do registro atual em um resultado. Se você planeja copiar valores dos campos, esta informação de tamanho é útil também para a otimização, porque você pode evitar a chamada `strlen()`. Se o resultado contém dados binários, você **deve** utilizar esta função para determinar o tamanho dos dados, pois `strlen()` retorna um valor incorreto para qualquer campo contendo caracteres nulos.

O tamanho para colunas vazias e para colunas contendo valores `NULL` é zero. Para ver como distinguir estes dois casos, veja a descrição de `mysql_fetch_row()`.

Valor Retornado

Um vetor de unsigned long integers (inteiros longos sem sinal) representando o tamanho de cada coluna (não incluindo nenhuma caracter nulo). NULL se ocorrer um erro.

Erros

`mysql_fetch_lengths()` só é válido para o registro atual no resultado. Ele retorna NULL se você chamá-lo antes de `mysql_fetch_row()` ou depois de retornar todos os registros em um resultado.

Exemplo

```
MYSQL_ROW row;
unsigned long *lengths;
unsigned int num_fields;
unsigned int i;

row = mysql_fetch_row(result);
if (row)
{
    num_fields = mysql_num_fields(result);
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i < num_fields; i++)
    {
        printf("Column %u is %lu bytes in length.\n", i, lengths[i]);
    }
}
```

8.1.3.19 mysql_fetch_row()

MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)

Descrição

Recuera o próximo registro do resultado. Quando usado depois de `mysql_store_result()`, `mysql_fetch_row()` retorna NULL quando não houver mais registros para retornar. Quando usado depois de `mysql_use_result()`, `mysql_fetch_row()` retorna NULL quando não houver mais registros para retornar ou ocorrer um erro.

O número de valores no registro é dado por `mysql_num_fields(result)`. Se `row` guarda o valor retornado de uma chamada `mysql_fetch_row()`, apontadores para os valores são acessados como `row[0]` a `row[mysql_num_fields(result)-1]`. Valores NULL no registro são indicados por apontadores NULL.

Os tamanhos dos valores do campo no registro poden ser obtidos chamando `mysql_fetch_lengths()`. Campos vazios e campos contendo NULL tem tamanho 0; você pode distingui-los verificando o apontador para o valor do campo. Se o apontador é NULL, o campo é NULL; senão o campo está vazio.

Valor Retornado

Uma estrutura `MYSQL_ROW` para o próximo registro. `NULL` se não houver mais linhas para retornar ou ocorrer um erro.

Erros

`CR_SERVER_LOST`

A conexão com o servidor foi perdida durante a consulta.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

Exemplo

```
MYSQL_ROW row;
unsigned int num_fields;
unsigned int i;

num_fields = mysql_num_fields(result);
while ((row = mysql_fetch_row(result)))
{
    unsigned long *lengths;
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i < num_fields; i++)
    {
        printf("[%.*s] ", (int) lengths[i], row[i] ? row[i] : "NULL");
    }
    printf("\n");
}
```

8.1.3.20 `mysql_field_count()`

```
unsigned int mysql_field_count(MYSQL *mysql)
```

Se você estiver utilizando uma versão anterior a versão 3.22.24 do MySQL, você deve utilizar `unsigned int mysql_num_fields(MYSQL *mysql)`.

Descrição

Retorna o número de colunas para a consulta mais recente na conexão.

Normalmente esta função é utilizada quando `mysql_store_result()` retorna `NULL` (então você não possui um apontador para o resultado). Neste caso, você pode chamar `mysql_field_count()` para determinar se `mysql_store_result()` não produziu um resultado vazio. Isto permite que o programa cliente tome a ação apropriada sem saber se a consulta foi uma instrução `SELECT` (ou do mesmo tipo). O exemplo mostrado aqui ilustra como isto pode ser feito.

See [\[NULL mysql_store_result\(\)\]](#), page [\[undefined\]](#).

Valor Retornado

Um unsigned integer (inteiro sem sinal) representando o número de campo em um resultado.

Erros

Nenhum.

Exemplo

```

MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql,query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result) // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else // mysql_store_result() returned nothing; should it have?
    {
        if(mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
        else // mysql_store_result() should have returned data
        {
            fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
    }
}

```

Uma alternativa é substituir a chamada `mysql_field_count(&mysql)` com `mysql_errno(&mysql)`. Neste caso, você está verificando diretamente um erro de `mysql_store_result()` em vez de conferir o valor de `mysql_field_count()` se a instrução foi uma `SELECT`.

8.1.3.21 `mysql_field_seek()`

```

MYSQL_FIELD_OFFSET mysql_field_seek(MYSQL_RES *result, MYSQL_FIELD_OFFSET
offset)

```

Descrição

Define o cursor campo com o offset dado. A próxima chamada para `mysql_fetch_field()` irá recuperar a definição de campo da coluna associada com o offset.

Para buscar o início de um registro, passe zero como valor do `offset`.

Valor Retornado

O valor anterior do cursor de campo.

Erros

Nenhum.

8.1.3.22 `mysql_field_tell()`

```
MYSQL_FIELD_OFFSET mysql_field_tell(MYSQL_RES *result)
```

Descrição

Retorna a posição do cursos do campo usado pelo último `mysql_fetch_field()`. Este valor pode ser usado como um argumento para `mysql_field_seek()`.

Valor Retornado

O offset atual do cursor de campo.

Erros

Nenhum.

8.1.3.23 `mysql_free_result()`

```
void mysql_free_result(MYSQL_RES *result)
```

Descrição

Libera a memória alocada para o resultado por `mysql_store_result()`, `mysql_use_result()`, `mysql_list_dbs()`, etc. Quando você finalizar o uso do resultado, você deve liberar a memória utilizada chamando `mysql_free_result()`.

Valor Retornado

Nenhum.

Erros

Nenhum.

8.1.3.24 `mysql_get_client_info()`

```
char *mysql_get_client_info(void)
```

Descrição

Retorna uma string que representa a versão da biblioteca cliente.

Valor Retornado

Uma string representando a versão da biblioteca cliente do MySQL.

Erros

Nenhum.

8.1.3.25 `mysql_get_server_version()`

```
unsigned long mysql_get_server_version(MYSQL *mysql)
```

Descrição

Retorna o número de versão do servidor como um inteiro (novo na versão 4.1)

Valor Retornado

Um número que representa a versão do servidor MySQL no formato:

`versão_principal*10000 + versão_menor*100 + sub_versão`

Por exemplo, 4.1.0 é retornado como 40100.

Ela é útil para determinar a versão do servidor rapidamente em um programa cliente para saber se algumas capacidades existem.

Erros

Nenhum.

8.1.3.26 `mysql_get_host_info()`

```
char *mysql_get_host_info(MYSQL *mysql)
```

Descrição

Retorna uma string descrevendo o tipo da conexão em uso, incluindo o nome da máquina servidora.

Valor Retornado

Uma string representando o nome da máquina servidora e o tipo de conexão.

Erros

Nenhum.

8.1.3.27 `mysql_get_proto_info()`

```
unsigned int mysql_get_proto_info(MYSQL *mysql)
```

Descrição

Retorna a versão do protocolo usado pela conexão atual.

Valor Retornado

Um unsigned integer (inteiro sem sinal) representando a versão do protocolo usado pela conexão atual.

Erros

Nenhum.

8.1.3.28 `mysql_get_server_info()`

```
char *mysql_get_server_info(MYSQL *mysql)
```

Descrição

Retorna um string que representa o número da versão do servidor.

Valor Retornado

Um string representando o número da versão do servidor.

Erros

Nenhum.

8.1.3.29 `mysql_info()`

```
char *mysql_info(MYSQL *mysql)
```

Descrição

Retorna um string fornecendo informação sobre a consulta executada mais recentemente, mas apenas para as instruções listadas aqui. Para outras instruções, `mysql_info()` retorna NULL. O formato da string varia dependendo do tipo de consulta, como descrito aqui. Os números são apenas ilustrativos; a string irá conter valores apropriados para a consulta.

```
INSERT INTO ... SELECT ...
```

Formato da string: Records: 100 Duplicates: 0 Warnings: 0

INSERT INTO ... VALUES (...),(...),(...)...

Formato da string: Records: 3 Duplicates: 0 Warnings: 0

LOAD DATA INFILE ...

Formato da string: Records: 1 Deleted: 0 Skipped: 0 Warnings: 0

ALTER TABLE

Formato da string: Records: 3 Duplicates: 0 Warnings: 0

UPDATE Formato da string: Rows matched: 40 Changed: 40 Warnings: 0

Note que `mysql_info()` retorna um valor não-NULL para a instrução `INSERT ... VALUES` somente se um lista de múltiplos valores é especificada na instrução.

Valor Retornado

Uma string representando informação adicional sobre a consulta executada mais recentemente. NULL se não houver nenhuma informação disponível para a consulta.

Erros

Nenhum.

8.1.3.30 `mysql_init()`

MYSQL *mysql_init(MYSQL *mysql)

Descrição

Aloca ou inicializa um objeto MYSQL apropriado para `mysql_real_connect()`. Se `mysql` é um ponteiro NULL, a função aloca, inicializa e retorna um novo objeto. Senão o objeto é inicializado e o endereço do objeto é retornado. Se `mysql_init()` aloca um novo objeto, ele será liberado quando `mysql_close()` for chamado para fechar a conexão.

Valor Retornado

Um handle MYSQL* inicializado. NULL se não houver memória suficiente para alocar o novo objeto.

Erros

Em caso de memória insuficiente, NULL é retornado.

8.1.3.31 `mysql_insert_id()`

my_ulonglong mysql_insert_id(MYSQL *mysql)

Descrição

Retorna o ID gerado para uma coluna `AUTO_INCREMENT` pela consulta anterior. Use esta função depois de ter realizado um consulta `INSERT` em uma tabela que contenha um campo `AUTO_INCREMENT`.

Note que `mysql_insert_id()` retorna 0 se a consulta anterior não gerar um valor `AUTO_INCREMENT`. Se você desejar salvar o valor para uso posterior, chame `mysql_insert_id()` imediatamente depois da consulta que gerou o valor.

`mysql_insert_id()` é atualizado depois de instruções `INSERT` e `UPDATE` que geram um valor `AUTO_INCREMENT` ou que definem um valor de coluna com `LAST_INSERT_ID(expr)`. See [\[Funções diversas\]](#), page [\[Funções diversas\]](#).

Note também que o valor da função SQL `LAST_INSERT_ID()` sempre contém o o valor `AUTO_INCREMENT` gerado mais recentemente e não é zerado entre as consultas porque o valor desta função é mantido no servidor.

Valor Retornado

O valor do campo `AUTO_INCREMENT` que foi atualizado pela consulta anterior. Retorna zero se não houve consultas anteriores na conexão ou se a consulta não atualizou o valor `AUTO_INCREMENT`.

Erros

Nenhum.

8.1.3.32 `mysql_kill()`

```
int mysql_kill(MYSQL *mysql, unsigned long pid)
```

Descrição

Diz para o servidor matar um thread especificada pelo `pid`.

Valor Retornado

Zero em caso de sucesso. Diferente de zero se ocorrer um erro.

Erros

`CR_COMMANDS_OUT_OF_SYNC`

Os comando foram executados em uma ordem inapropriada.

`CR_SERVER_GONE_ERROR`

O servidor MySQL foi finalizado.

`CR_SERVER_LOST`

A conexão ao servidor MySQL foi perdida durante a consulta.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

8.1.3.33 `mysql_list_dbs()`

`MYSQL_RES *mysql_list_dbs(MYSQL *mysql, const char *wild)`

Descrição

Retorna um resultado com nome de banco de dados no servidor que correspondem a uma expressão regular especificada pelo parâmetro `wild`. `wild` pode conter o meta caracteres ‘%’ ou ‘_’, ou pode ser um ponteiro NULL para corresponder a todos os banco de dados. Chamar `mysql_list_dbs()` é o mesmo que executar a consulta `SHOW databases [LIKE wild]`.

Você deve liberar o resultado com `mysql_free_result()`.

Valor Retornado

Um conjunto de resultados `MYSQL_RES` no caso de sucesso. NULL se ocorrer um erro.

Erros

`CR_COMMANDS_OUT_OF_SYNC`

Os comando foram executados em uma ordem inapropriada.

`CR_SERVER_GONE_ERROR`

O servidor MySQL foi finalizado.

`CR_SERVER_LOST`

A conexão ao servidor MySQL foi perdida durante a consulta.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

8.1.3.34 `mysql_list_fields()`

`MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const char *wild)`

Descrição

Retorna um resultado contendo nomes de campos de uma tabela dada que correspondam a expressão regular especificada pelo parâmetro `wild`. `wild` pode conter os metacaracteres ‘%’ ou ‘_’, ou pode ser um ponteiro NULL para corresponder a todos os campos. Chamar `mysql_list_fields()` é o mesmo que executar a consulta `SHOW COLUMNS FROM nome_tabela [LIKE wild]`.

Note que é recomendado que você use `SHOW COLUMNS FROM nome_tabela` em vez de `mysql_list_fields()`.

Você deve liberar o resultado com `mysql_free_result()`.

Valor Retornado

Um conjunto de resultados `MYSQL_RES` em caso de sucesso. NULL se ocorrer um erro.

Erros

`CR_COMMANDS_OUT_OF_SYNC`

Os comando foram executados em uma ordem inapropriada.

`CR_SERVER_GONE_ERROR`

O servidor MySQL foi finalizado.

`CR_SERVER_LOST`

A conexão ao servidor MySQL foi perdida durante a consulta.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

8.1.3.35 `mysql_list_processes()`

`MYSQL_RES *mysql_list_processes(MYSQL *mysql)`

Descrição

Retorna um resultado descrevendo a thread atual do servidor. É o mesmo tipo de informação relatado por `mysqladmin processlist` ou uma consulta `SHOW PROCESSLIST`.

Você deve liberar o resultado com `mysql_free_result()`.

Valor Retornado

Um conjunto de resultados `MYSQL_RES` em caso de sucesso. `NULL` se ocorrer um erro.

Erros

`CR_COMMANDS_OUT_OF_SYNC`

Os comando foram executados em uma ordem inapropriada.

`CR_SERVER_GONE_ERROR`

O servidor MySQL foi finalizado.

`CR_SERVER_LOST`

A conexão ao servidor MySQL foi perdida durante a consulta.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

8.1.3.36 `mysql_list_tables()`

`MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)`

Descrição

Retorna um resultado contendo nomes de tabelas no banco de dados atual que correspondam a expressão regular especificada pelo parâmetro `wild`. `wild` pode conter os mets caracteres `'%'` or `'_'`, ou pode ser uma ponteiro `NULL` para corresponde a todas as tabelas. Chamar `mysql_list_tables()` é o mesmo que executar a consulta `SHOW tables [LIKE wild]`.

Você deve liberar o resultado com `mysql_free_result()`.

Valor Retornado

Um conjunto de resultados `MYSQL_RES` em caso de sucesso. `NULL` se ocorrer um erro.

Erros

`CR_COMMANDS_OUT_OF_SYNC`

Os comando foram executados em uma ordem inapropriada.

`CR_SERVER_GONE_ERROR`

O servidor MySQL foi finalizado.

`CR_SERVER_LOST`

A conexão ao servidor MySQL foi perdida durante a consulta.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

8.1.3.37 `mysql_num_fields()`

```
unsigned int mysql_num_fields(MYSQL_RES *result)
```

ou

```
unsigned int mysql_num_fields(MYSQL *mysql)
```

A segunda forma não funciona na versão 3.22.24 ou mais novas do MySQL. Para passar um argumento `MYSQL*` você de utilizar `unsigned int mysql_field_count(MYSQL *mysql)` em seu lugar.

Descrição

Retorna o número de colunas em um resultado.

Note que você pode obter o número de colunas com um ponteiro para o conjunto de resultados ou para um manipulador (`handle`) de conexão. Você usaria o manipular de conexão se `mysql_store_result()` ou `mysql_use_result()` retorna `NULL` (então você não tem um ponteiro para o resultado). Neste caso, você pode chamar `mysql_field_count()` para determinar se `mysql_store_result()` não produziu um resultado vazio. Isto permite que o programa cliente tome a ação apropriada sem saber se a consulta foi uma instrução `SELECT` (ou do tipo `SELECT`). O exemplo mostrado abaixo ilustra como isto pode ser feito.

See [\[NULL mysql_store_result\(\)\]](#), page [\[undefined\]](#).

Valor Retornado

Um `unsigned integer` (inteiro sem sinal) representando o número de campos no conjunto de resultados.

Erros

Nenhum.

Exemplo

```

MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql,query_string))
{
    // erro
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result) // existem registros
    {
        num_fields = mysql_num_fields(result);
        // retorna registros e chama mysql_free_result(result)
    }
    else // mysql_store_result() retorna vazio; era esperado?
    {
        if (mysql_errno(&mysql))
        {
            fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
        else if (mysql_field_count(&mysql) == 0)
        {
            // consulta não retorna dados
            // (ela não era um SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
    }
}
}

```

Uma alternativa (se você souber que a sua consulta retornou um resultado) é substituir a chamada `mysql_errno(&mysql)` pela verificação de se `mysql_field_count(&mysql)` é = 0. Isto só acontece se alguma coisa der errado.

8.1.3.38 `mysql_num_rows()`

```
my_ulonglong mysql_num_rows(MYSQL_RES *result)
```

Descrição

Retorna o número de linhas em um resultado.

O uso de `mysql_num_rows()` depende de se você utiliza `mysql_store_result()` ou `mysql_use_result()` para retornar o resultado. Se você usa `mysql_store_result()`, `mysql_num_rows()` pode ser chamado imediatamente. Se você usa `mysql_use_result()`, `mysql_num_rows()` não retornará o valor correto até que todas as linhas no resultado tenham sido recuperadas.

Valor Retornado

O numero de linhas no resultado.

Erros

Nenhum.

8.1.3.39 `mysql_options()`

```
int mysql_options(MYSQL *mysql, enum mysql_option option, const char *arg)
```

Descrição

Pode ser usado para definir opções extras de conexão e afetar o comportamento de uma conexão. Esta função pode ser chamada várias vezes para definir diversas opções.

`mysql_options()` deve ser chamado depois de `mysql_init()` e antes de `mysql_connect()` ou `mysql_real_connect()`.

O argumento `option` é a opção que você que definir; o argumento `arg` é o valor para a opção. Se a opção é um inteiro, então `arg` deve apontar para o valor do inteiro.

Valores possíveis para as opções:

Opção	Tipo de Função argumento	de Função
<code>MYSQL_OPT_CONNECT_TIMEOUT</code>	<code>unsigned int *</code>	Tempo limite de conexão em segundos.
<code>MYSQL_OPT_COMPRESS</code>	Não usado	Usa o protocolo cliente/servidor compactado.
<code>MYSQL_OPT_LOCAL_INFILE</code>	ponteiro para <code>unsigned integer</code> opcional	Se nenhum ponteiro for dado ou se apontar para um <code>unsigned int != 0</code> o comando <code>LOAD LOCAL INFILE</code> está habilitado.
<code>MYSQL_OPT_NAMED_PIPE</code>	Não usado	Usa named pipes para conectar ao servidor MySQL no NT.
<code>MYSQL_INIT_COMMAND</code>	<code>char *</code>	Comando para executar ao conectar ao servidor MySQL. Será automaticamente executado ao se reconectar.
<code>MYSQL_READ_DEFAULT_FILE</code>	<code>char *</code>	Lê opções do arquivo de opções definido no lugar de <code>'my.cnf'</code> .
<code>MYSQL_READ_DEFAULT_GROUP</code>	<code>char *</code>	Lê opções do grupo indicado no arquivo <code>'my.cnf'</code> ou no arquivo especificado com <code>MYSQL_READ_DEFAULT_FILE</code> .

Note que o grupo `client` é sempre lido se você utiliza `MYSQL_READ_DEFAULT_FILE` ou `MYSQL_READ_DEFAULT_GROUP`.

O grupo especificado no arquivo de opções pode conter as seguintes opções:

Opção	Descrição
-------	-----------

<code>connect-timeout</code>	Tempo limite de conexão em segundos. No Linux este tempo limite também é utilizado para esperar pela primeira resposta do servidor
<code>compress</code>	Utiliza o protocolo cliente/servidor compactado.
<code>database</code>	Conecta a este banco de dados se nenhum banco de dados for especificado no comando de conexão.
<code>debug</code>	Opções de depuração.
<code>disable-local-infile</code>	Desabilita o uso de LOAD DATA LOCAL.
<code>host</code>	Nome de máquina padrão.
<code>init-command</code>	Comando para executar ao conectar ao servidor MySQL. Será executado automaticamente ao reconectar.
<code>interactive-timeout</code>	O mesmo que o especificado em CLIENT_INTERACTIVE para <code>mysql_real_connect()</code> . See [mysql_real_connect] , page [mysql_real_connect] .
<code>local-infile[=(0 1)]</code>	Se não houver argumento ou o argumento for diferente de 0 habilita o uso de LOAD DATA LOCAL.
<code>max_allowed_packet</code>	Tamanho máximo dos pacotes que o cliente pode ler do servidor.
<code>password</code>	Senha padrão.
<code>pipe</code>	Usa named pipes para conectar ao servidor MySQL no NT.
<code>protocol=(TCP SOCKET PIPE MEMORY)</code>	Qual protocolo usar ao conectar no servidor (Novo na versão 4.1)
<code>port</code>	Número padrão da porta.
<code>return-found-rows</code>	Diz ao <code>mysql_info()</code> para retornar registros encontrados no lugar de registros atualizados ao usar UPDATE.
<code>shared-memory-base-name=name</code>	Nome da memória compartilhada utilizada para conectar ao servidor (o padrão é "MySQL"). Novo na versão 4.1.
<code>socket</code>	Número padrão do socket.
<code>user</code>	Usuário padrão.

Note que `timeout` foi substituído por `connect-timeout`, mas `timeout` ainda funcionará por enquanto.

Para maiores informações sobre arquivos de opções, veja [\[Arquivo de opções\]](#), page [\[Arquivo de opções\]](#).

Valor Retornado

Zero em caso de sucesso. Diferente de zero se você utilizar uma opção desconhecida.

Exemplo

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql, MYSQL_OPT_COMPRESS, 0);
```

```
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"odbc");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
        mysql_error(&mysql));
}
```

O exemplo acima diz ao cliente para usar o protocolo cliente/servidor compactado e ler a opção adicional da seção `odbc` no arquivo de opções `my.cnf`.

8.1.3.40 `mysql_ping()`

```
int mysql_ping(MYSQL *mysql)
```

Descrição

Verifica se a conexão ao servidor está funcionando. Se ela tiver caído é feita uma tentativa de conexão automaticamente.

Esta função pode ser usada pelos clientes que se ficam inativo por um longo tempo para verificar se o servidor fechou a conexão e reconectar se necessário.

Valor Retornado

Zero se o servidor estiver funcionando. Diferente de zero se ocorrer um erro.

Erros

`CR_COMMANDS_OUT_OF_SYNC`

Os comando foram executados em uma ordem inapropriada.

`CR_SERVER_GONE_ERROR`

O servidor MySQL foi finalizado.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

8.1.3.41 `mysql_query()`

```
int mysql_query(MYSQL *mysql, const char *query)
```

Descrição

Executa uma consulta SQL apontada pela string terminada em null `query`. A consulta deve consistir de uma única instrução SQL. Você não deve adicionar ponto e vírgula (`;`) ou `\g` ao fim da instrução.

`mysql_query()` não pode ser usadas por consultas que contenham dados binários; você deve utilizar `mysql_real_query()` em seu lugar. (Dados binários podem conter o caracter `'\0'`, que `mysql_query()` interpreta como o fim a string de consulta.)

Se você quiser saber se a consulta deve retornar um resultado ou não, você pode utilizar `mysql_field_count()` para verificar isto. See [\(undefined\) \[mysql_field_count\(\)\], page \(undefined\)](#).

Valor Retornado

Zero se a consulta obteve sucesso. Diferente de zero se ocorreu um erro.

Erros

`CR_COMMANDS_OUT_OF_SYNC`

Os comando foram executados em uma ordem inapropriada.

`CR_SERVER_GONE_ERROR`

O servidor MySQL foi finalizado.

`CR_SERVER_LOST`

A conexão ao servidor MySQL foi perdida durante a consulta.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

8.1.3.42 `mysql_real_connect()`

```
MYSQL *mysql_real_connect(MYSQL *mysql, const char *host, const char *user,
const char *passwd, const char *db, unsigned int port, const char *unix_socket,
unsigned int client_flag)
```

Descrição

`mysql_real_connect()` tenta estabelecer uma conexão mecanismo MySQL de banco de dados executando em `host`. `mysql_real_connect()` deve completar com sucesso antes que você possa executar qualquer um das outars funções da API, com a excessão de `mysql_get_client_info()`.

Os parâmetros são especificados da seguinte forma:

- O primeiro parâmetro deve ser o endereço de uma estrutura `MYSQL` existente. Antes de chamar `mysql_real_connect()` você deve chamar `mysql_init()` para inicializar a estrutura `MYSQL`. Você pode alterar várias opções de conexão com a chamada `mysql_options()`. See [\[mysql_options\]](#), page [\[mysql_options\]](#).
- O valor de `host` pode ser tanto um nome de máquina quanto um endereço IP. Se `host` é `NULL` ou a string `"localhost"`, a conexão é feita na máquina local. Se o SO suporta sockets (Unix) ou named pipes (Windows), eles são utilizados em vez de TCP/IP para a conexão ao servidor.
- O parâmetro `user` contém a identificação do usuário MySQL. Se `user` é `NULL`, considera-se o usuário padrão. Sob Unix, ele é o login atual. Sob ODBC no Windows, o usuário atual deve ser especificado explicitamente. See [\[Administrador ODBC\]](#), page [\[Administrador ODBC\]](#).
- O parâmetro `passwd` contém a senha para `user`. Se `passwd` é `NULL`, somente entradas na tabela `user` para usuários que tenham campo de senha em branco (vazia) serão verificados ipor um padrão coincidentete. Isto permite que o administrador do banco de dados configure o sistema de privilégios do MySQL de tal maneira que usuários os usuários conseguirão privileios diferentes, dependendo se ele especificou ou não uma senha.

Nota: Não tente criptografar a senha antes de chamar `mysql_real_connect()`; senhas criptografadas são tratadas automaticamente pela API cliente.

- `db` é o nome de banco de dados. Se `db` não é `NULL`, a conexão definirá o banco de dados padrão com este valor.
- Se `port` não é 0, o valor será usado como o número da porta para as conexões TCP/IP. Note que o parâmetro `host` determina o tipo da conexão.
- Se `unix_socket` não é `NULL`, a string especifica o socket ou named pipe que deve ser usado. Note que o parâmetro `host` determina o tipo de conexão.
- O valor de `client_flag` é normalmente 0, mas pode ser definido como uma combinação dos parâmetro seguintes em circunstâncias especiais:

Nome do parâmetro	Descrição do parâmetro
<code>CLIENT_COMPRESS</code>	Usa protocolo compactado.
<code>CLIENT_FOUND_ROWS</code>	Retorna o número de linhas encontradas (correspondentes a um padrão), não o número de linha efetivo.
<code>CLIENT_IGNORE_SPACE</code>	Permite espaço depois do nome de funções. torna todos os nomes de funções palavras reservadas.
<code>CLIENT_INTERACTIVE</code>	Permite <code>interactive_timeout</code> segundos (no lugar de <code>wait_timeout</code> segundos) de inatividade antes de fechar a conexão.
<code>CLIENT_NO_SCHEMA</code>	Não permite a sintaxe <code>db_name.nome_tabela.nome_coluna</code> . Isto é para o ODBC. Ele faz com que o analisador gere um erro se você utilizar aquela sintaxe. É útil para achar erros em alguns programas ODBC.
<code>CLIENT_ODBC</code>	O cliente é um cliente ODBC. Torna o <code>mysqld</code> mais amigável ao ODBC.
<code>CLIENT_SSL</code>	Usa SSL (protocolo criptografado).

Valor Retornado

Um handle de conexão `MYSQL*` se a conexão foi obtida com sucesso, `NULL` se a conexão falhou. Para um conexão estabelecida o valor de retorn é o mesmo que o valor do primeiro parâmetro.

Erros

`CR_CONN_HOST_ERROR`

Falhou ao conectar ao servidor MySQL.

`CR_CONNECTION_ERROR`

Falhou ao conectar ao servidor MySQL local.

`CR_IPSOCK_ERROR`

Falhou au criar um socket IP.

`CR_OUT_OF_MEMORY`

Sem memória.

`CR_SOCKET_CREATE_ERROR`

Falhou ao criar um socket Unix.

CR_UNKNOWN_HOST

Falhou ao procurar o endereço IP para o nome de maquina.

CR_VERSION_ERROR

Um erro de protocolo resultou da tentativa de conexao a um servidor com uma biblioteca cliente que utiliza uma versão de protocolo diferente. Isto pode acontecer se você utiliza uma biblioteca cliente muito antiga para se conectar a um novo servidor qua não foi iniciado com a opção `--old-protocol`.

CR_NAMEDPIPEOPEN_ERROR

Falhou ao criar um named pipe no Windows.

CR_NAMEDPIPEWAIT_ERROR

Falhou ao esperar por um named pipe no Windows.

CR_NAMEDPIPESETSTATE_ERROR

Falhou ao conseguir manipulador do pipe no Windows.

CR_SERVER_LOST

Se `connect_timeout > 0` e leva mais que `connect_timeout` segundos para conectar ao servidor ou se o servidro foi finalizado ao executar o `init-command`.

Exemplo

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"seu_programa");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
            mysql_error(&mysql));
}
```

Usando `mysql_options()` a biblioteca MySQL irá ler as seções `[client]` e `[seu_programa]` no arquivo `my.cnf` o qual irá assegurar que seu programa irá funcionar, mesmo se alguém tiver configurado o MySQL de um modo fora do padrão.

Note que sob a conexão, `mysql_real_connect()` define o parâmetro `reconnect` (parte da estrutura `MYSQL`) para um valor de 1. Este parâmetro indica, no evento em que uma consulta não pode ser realizada devido a perda de conexão, para tentar se reconectar ao servidor antes de esgotar as tentativas.

8.1.3.43 mysql_real_escape_string()

```
unsigned long mysql_real_escape_string(MYSQL *mysql, char *to, const char
*from, unsigned long length)
```

Descrição

A função é usada para criar um string SQL válida que você pode usar em uma instrução SQL. See [\(undefined\)](#) [Sintaxe de string], page [\(undefined\)](#).

A string em `from` é codificada para uma string SQL com escape, levando em conta o conjunto de caracteres atual da conexão. O resultado é colocada em `to` e uma byte nulo de terminação é adicionado. Caracteres codificados são NUL (ASCII 0), `'\n'`, `'\r'`, `'\'`, `'\''`, `'\"'`, e Control-Z (See [\[Literais\]](#), page [\(undefined\)](#)). (O MySQL precisa que apenas a barra invertida e as aspas utilizadas para citar a consulta sejam escapadas. Esta função coloca os outros caracteres entre aspas para torná-lo mais fácil de ser lido em arquivos log.)

A string apontada por `from` deve ter o tamanho de `length` bytes. Você deve alocar o buffer `to` para o tamanho de pelo menos `length*2+1` bytes. (No pior caso, cada caractere pode precisar de ser codificado como se utilizasse dois bytes, e você precisaria de espaço para o byte null de terminação.) Quando `mysql_real_escape_string()` retornar, o conteúdo de `to` será uma string terminada em null. O valor é o tamanho da string codificada, não incluindo o caractere nulo usado para terminar a string.

Exemplo

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
*end++ = '\'';
end += mysql_real_escape_string(&mysql, end,"What's this",11);
*end++ = '\'';
*end++ = ',';
*end++ = '\'';
end += mysql_real_escape_string(&mysql, end,"binary data: \0\r\n",16);
*end++ = '\'';
*end++ = ')';

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
    fprintf(stderr, "Failed to insert row, Error: %s\n",
            mysql_error(&mysql));
}
```

A função `strmov()` usada no exemplo está incluída na biblioteca `mysqlclient` e funciona como `strcpy()` mas retorna um ponteiro para null de terminação do primeiro parâmetro.

Valor Retornado

O tamanho do valor colocado em `to`, não incluindo o caractere null de terminação.

Erros

Nenhum.

8.1.3.44 `mysql_real_query()`

```
int mysql_real_query(MYSQL *mysql, const char *query, unsigned long length)
```

Descrição

Executa a consulta SQL apontada por `query`, que deve ser uma string de `length` bytes. A consulta deve consistir de uma instrução SQL simples. Você não deve adicionar um ponto e vírgula (;) ou \g no fim da instrução.

Você **deve** utilizar `mysql_real_query()` em lugar de `mysql_query()` para consultas que contenham dados binários, pois eles podem conter o caracter '\0'. Além disso, `mysql_real_query()` é mais rápido que `mysql_query()` pois ele não faz chamadas `strlen()` na string de consulta.

Se você quiser saber se a consulta retornou um resultado ou não, você pode usar `mysql_field_count()`. See [\[mysql_field_count\]](#), page [\[mysql_field_count\]](#).

Valor Retornado

Zero se a consulta obteve sucesso. Diferente de zero se ocorrer um erro.

Erros

`CR_COMMANDS_OUT_OF_SYNC`

Os comando foram executados em uma ordem inapropriada.

`CR_SERVER_GONE_ERROR`

O servidor MySQL foi finalizado.

`CR_SERVER_LOST`

A conexão ao servidor MySQL foi perdida durante a consulta.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

8.1.3.45 `mysql_reload()`

```
int mysql_reload(MYSQL *mysql)
```

Descrição

Diz ao servidor MySQL para recarregar a tabela de ables. The connected user must have the `RELOAD` privilege.

This function is deprecated. It is preferable to use `mysql_query()` to issue a SQL `FLUSH PRIVILEGES` statement instead.

Valor Retornado

Zero for success. Non-zero if an error occurred.

Erros

CR_COMMANDS_OUT_OF_SYNC

Os comando foram executados em uma ordem inapropriada.

CR_SERVER_GONE_ERROR

O servidor MySQL foi finalizado.

CR_SERVER_LOST

A conexão ao servidor MySQL foi perdida durante a consulta.

CR_UNKNOWN_ERROR

Um erro desconhecido ocorreu.

8.1.3.46 `mysql_row_seek()`

`MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET offset)`

Descrição

Atribui ao cursor de linha um registro arbitrário em resultado de uma consulta. Isto exige que a estrutura do resultado contenha todo o resultado da consulta, assim `mysql_row_seek()` pode ser usado em conjunto apenas com `mysql_store_result()`, e não com `mysql_use_result()`.

O offset deve ser um valor retornado de uma chamada a `mysql_row_tell()` ou a `mysql_row_seek()`. Este valor não simplesmente um número de linha; se você quiser buscar um registro em um resultado usando o número de linha utilize `mysql_data_seek()`.

Valor Retornado

O valor anterior do cursor de linha. Este valor pode ser passado a uma chamada subsequente `mysql_row_seek()`.

Erros

Nenhum.

8.1.3.47 `mysql_row_tell()`

`MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result)`

Descrição

Retorna a posição atual do cursor de linha para a última `mysql_fetch_row()`. Este valor pode ser utilizado como argumento para `mysql_row_seek()`.

Você deve utilizar `mysql_row_tell()` somente depois de `mysql_store_result()`, e não depois de `mysql_use_result()`.

Valor Retornado

O offset atual do cursos de linha.

Erros

Nenhum.

8.1.3.48 `mysql_select_db()`

```
int mysql_select_db(MYSQL *mysql, const char *db)
```

Descrição

Faz com que o banco de dados especificado por `db` se torne o padrão (atual) na conexão especificada por `mysql`. Nas consultas seguintes este banco de dados é o padrão para tabelas que não incluem uma especificação explícita para o banco de dados.

`mysql_select_db()` falha a menos que o usuário conectado possa ser autenticado com permissão para utilizar o banco de dados.

Valor Retornado

Zero em caso de sucesso. Diferente de zero se ocorrer um erro.

Erros

`CR_COMMANDS_OUT_OF_SYNC`

Os comandos foram executados em uma ordem inapropriada.

`CR_SERVER_GONE_ERROR`

O servidor MySQL foi finalizado.

`CR_SERVER_LOST`

A conexão ao servidor MySQL foi perdida durante a consulta.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

8.1.3.49 `mysql_shutdown()`

```
int mysql_shutdown(MYSQL *mysql)
```

Descrição

Diz ao servidor de banco de dados para finalizar. O usuário conectado deve ter privilégio `SHUTDOWN`.

Valor Retornado

Zero em caso de sucesso. Diferente de zero se ocorrer um erro.

Erros

`CR_COMMANDS_OUT_OF_SYNC`

Os comando foram executados em uma ordem inapropriada.

`CR_SERVER_GONE_ERROR`

O servidor MySQL foi finalizado.

`CR_SERVER_LOST`

A conexão ao servidor MySQL foi perdida durante a consulta.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

8.1.3.50 `mysql_stat()`

`char *mysql_stat(MYSQL *mysql)`

Descrição

Retorna uma string contendo informações sinmiliares a aquelas fornecidas pelo comando `mysqladmin status`. Isto inclui o tempo de conexão em segundos e o número de threads em execução, recargas e tabelas abertas.

Valor Retornado

Uma string descrevendo o status do servidor. `NULL` se um erro ocorrer.

Erros

`CR_COMMANDS_OUT_OF_SYNC`

Os comando foram executados em uma ordem inapropriada.

`CR_SERVER_GONE_ERROR`

O servidor MySQL foi finalizado.

`CR_SERVER_LOST`

A conexão ao servidor MySQL foi perdida durante a consulta.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

8.1.3.51 `mysql_store_result()`

`MYSQL_RES *mysql_store_result(MYSQL *mysql)`

Descrição

Você deve chamar `mysql_store_result()` ou `mysql_use_result()` para cada consulta que retorne dados com sucesso (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`).

Você não precisa chamar `mysql_store_result()` ou `mysql_use_result()` para outras consultas, mas ele não causará nenhum dano ou nenhuma queda notel de desempenho se você

chamar `mysql_store_result()` em todos os casos. Você pode detectar se a consulta não obteve resultado verificando se `mysql_store_result()` retornou 0.

Se você quiser saber se a consulta devia retornar algum resultado, você pode utilizar `mysql_field_count()` para fazer a verificação. See [\[mysql_field_count\]](#), page [\[undefined\]](#).

`mysql_store_result()` lê todo o resultado de uma consulta para um cliente, aloca uma estrutura `MYSQL_RES` e coloca o resultado nesta estrutura.

`mysql_store_result()` retorna um ponteiro para null se a consulta não retornar um resultado (se a consulta foi, por exemplo, uma instrução `INSERT`).

`mysql_store_result()` também retorna um ponteiro para null se a leitura do resultado falhar. Você pode verificar se você obteve um erro verificando se `mysql_error()` não retornou um ponteiro para null, se `mysql_errno()` retorna $<> 0$, ou se `mysql_field_count()` retorna $<> 0$.

Um resultado vazio é retornado se não houver registros a retornar. (Um resultado vazio é diferente de um ponteiro para null em um valor de retorno).

Uma vez que você tenha chamado `mysql_store_result()` e tenha retornado um resultado que não é uma apontador para null, você pode chamar `mysql_num_rows()` para descobrir quantas linhas existem no resultado.

Você pode chamar `mysql_fetch_row()` para buscar registros no resultado ou `mysql_row_seek()` e `mysql_row_tell()` para obter ou definir a posição atual do registro dentro do resultado.

Você deve chamar `mysql_free_result()` quando tiver terminado com o resultado.

See [\[NULL mysql_store_result\(\)\]](#), page [\[undefined\]](#).

Valor Retornado

Uma estrutura de resultado `MYSQL_RES` com o resultado. `NULL` se um erro ocorreu.

Erros

`CR_COMMANDS_OUT_OF_SYNC`

Os comando foram executados em uma ordem inapropriada.

`CR_OUT_OF_MEMORY`

Sem memória.

`CR_SERVER_GONE_ERROR`

O servidor MySQL foi finalizado.

`CR_SERVER_LOST`

A conexão ao servidor MySQL foi perdida durante a consulta.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

8.1.3.52 `mysql_thread_id()`

`unsigned long mysql_thread_id(MYSQL *mysql)`

Descrição

Retorna a ID da thread da conexão atual. Este valor pode ser usado como um argumento para `mysql_kill()` para finalizar a thread.

Se a conexão for perdida e você reconectar com `mysql_ping()`, a ID da thread irá alterar. Isto significa que você deve obter a ID da thread e guardá-la para uso posterior. Você deve obtê-la quando precisar dela.

Valor Retornado

A ID da thread da conexão atual.

Erros

Nenhum.

8.1.3.53 `mysql_use_result()`

`MYSQL_RES *mysql_use_result(MYSQL *mysql)`

Descrição

Você deve chamar `mysql_store_result()` ou `mysql_use_result()` para cada consulta que retornar data com sucesso (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`).

`mysql_use_result()` inicia a recuperação de um resultado mas não lê realmente o resultado no cliente como `mysql_store_result()` faz. Cada registro deve ser recuperado individualmente fazendo chamadas a `mysql_fetch_row()`. Ele lê o resultado de uma consulta diretamente do servidor sem armazenar em uma tabela temporária ou em um buffer local, o que é mais rápido e utiliza menos memória que `mysql_store_result()`. O cliente só irá alocar memória para o registro atual para o buffer de comunicação que pode crescer para `max_allowed_packet` bytes.

Por outro lado, você não deve utilizar `mysql_use_result()` se você estiver fazendo vários processamentos para cada registros no lado do cliente, ou se a saída é enviada para a tela, na qual o usuário de digitar um `^S` (parada de tela). Isto irá prender o servidor e impedir outras threads de atualizar qualquer tabela na qual o dados esteja sendo buascado.

Ao usar `mysql_use_result()`, você deve executar `mysql_fetch_row()` até um valor `NULL` ser retornado, senão, os registros não buscados retornarão como part do resultado de sua próxima consulta. A API C fornecerá o erro `Commands out of sync; you can't run this command now` se você esquecer de fazê-lo.

Você não pode utilizar `mysql_data_seek()`, `mysql_row_seek()`, `mysql_row_tell()`, `mysql_num_rows()`, ou `mysql_affected_rows()` com m resultado retornado de `mysql_use_result()`, nem pode executar outras consultas até que `mysql_use_result()` tenha finalizado. (No entanto, depois de buscar todos os registros, `mysql_num_rows()` retornará corretamente o número de registros buscados).

Você deve chamar `mysql_free_result()` após terminar de utilizar o resultado.

Valor Retornado

Uma estrutura de resultado `MYSQL_RES`. `NULL` se ocorrer um erro.

Erros

`CR_COMMANDS_OUT_OF_SYNC`

Os comando foram executados em uma ordem inapropriada.

`CR_OUT_OF_MEMORY`

Sem memória.

`CR_SERVER_GONE_ERROR`

O servidor MySQL foi finalizado.

`CR_SERVER_LOST`

A conexão ao servidor MySQL foi perdida durante a consulta.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

8.1.3.54 `mysql_commit()`

```
my_bool mysql_commit(MYSQL *mysql)
```

Descrição

Faz um commits na transação atual. Disponível no MySQL 4.1

Valor Retornado

Zero em caso de sucesso. Deiferente de zero se ocorrer um erro.

Erros

Nenhum.

8.1.3.55 `mysql_rollback()`

```
my_bool mysql_rollback(MYSQL *mysql)
```

Descrição

Faz um rollback na transação atual. Disponível no MySQL 4.1

Valor Retornado

Zero em caso de sucesso. Deiferente de zero se ocorrer um erro.

Erros

Nenhum.

8.1.3.56 `mysql_autocommit()`

`my_bool mysql_autocommit(MYSQL *mysql, my_bool mode)`

Descrição

Define o modo autocommit como ligado/desligado. Se o `mode` for 1, defien o modo autocommit como ligado; Se for 0, como desligado. Disponível no MySQL 4.1

Valor Retornado

Zero em caso de sucesso. Deiferente de zero se ocorrer um erro.

Erros

Nenhum.

8.1.3.57 `mysql_more_results()`

`my_bool mysql_more_results(MYSQL *mysql)`

Descrição

Retorna verdade se mais resultados da consulta atualmente em execução existem, e a aplicação deve chamar `mysql_next_result()` para buscar os resultados. Disponível no MySQL 4.1

Valor Retornado

TRUE se existem mais resultados. FALSE se não existem mais resultados.

Erros

Nenhum.

8.1.3.58 `mysql_next_result()`

`int mysql_next_result(MYSQL *mysql)`

Descrição

Se existem mais resultados da consulta, `mysql_next_result()` lê o próximo resultado da consulta e retorna o status a aplicação. Disponível no MySQL 4.1

Valor Retornado

Zero em caso de sucesso. Deiferente de zero se ocorrer um erro.

Erros

Nenhum.

8.1.4 Instruções Preparadas da API C

A partir da versão 4.1 do MySQL, você também pode fazer uso de instruções preparadas utilizando o manipulador de instruções `MYSQL_STMT`, que suporta a execução de múltiplas consultas e ligação de saídas.

Execução preparada é um modo eficiente de executar uma instrução mais de uma vez. A instrução é primeiramente analisada (preparada). Então é executada uma ou mais vezes posteriormente, utilizando o manipulador de instruções retornado pela função preparada.

Outra vantagem de instruções preparadas é que ela utiliza um protocolo binário, o qual faz a transferência de dados entre cliente e servidor de forma mais eficiente.

Execução preparada é mais rápida que a execução direta para instruções executadas mais que uma vez, pois a consulta é analisada apenas uma vez; No caso de execução direta, a consulta é analisada todas as vezes. Execução preparada também pode fornecer uma redução de tráfego de rede pois durante a execução da chamada, só são enviados os dados para os parâmetros.

8.1.5 Tipo de Dados de Instruções Prepaadas da API C

Instrução preparadas utilizam principalmente as duas estruturas `MYSQL_STMT` e `MYSQL_BIND` seguintes:

`MYSQL_STMT`

Esta estrutura representa um manipuladaor de instrução para instruções preparadas. Ele é usada por todas as funções relacionadas a instruções.

A instrução é inicializada quando a consulta é preparada utilizando `mysql_prepare()`.

Uma conexão pode ter 'n' maniouladores de instruções, e o limite depende dos recursos do sistema.

`MYSQL_BIND`

Esta estrutura é utilizada para ligar buffer de parâmetros (`mysql_bind_param()`) utilizando os dados dos parâmetros para chamar `mysql_execute()`; assim como ligar buffer de registros (`mysql_bind_result()`) para buscar o resultado com os dados utilizando `mysql_fetch()`.

A estrutura `MYSQL_BIND` contém os membros listados aqui:

```
enum enum_field_types buffer_type [input]
```

O tipo do buffer. O valor `type` deve ser um dos seguintes:

- `MYSQL_TYPE_TINY`
- `MYSQL_TYPE_SHORT`
- `MYSQL_TYPE_LONG`
- `MYSQL_TYPE_LONGLONG`
- `MYSQL_TYPE_FLOAT`
- `MYSQL_TYPE_DOUBLE`
- `MYSQL_TYPE_TIME`
- `MYSQL_TYPE_DATE`

- MYSQL_TYPE_DATETIME
- MYSQL_TYPE_TIMESTAMP
- MYSQL_TYPE_STRING
- MYSQL_TYPE_VAR_STRING
- MYSQL_TYPE_TINY_BLOB
- MYSQL_TYPE_MEDIUM_BLOB
- MYSQL_TYPE_LONG_BLOB
- MYSQL_TYPE_BLOB

`void *buffer` [input/output]

Um ponteiro para um buffer de dados de parâmetros no caso dele ser utilizado para fornecer dados de parâmetros ou ponteiro para um buffer no qual se deve retornar os dados quando a estrutura para ligação do resultado.

`unsigned long buffer_length` [input]

Tamanho do `*buffer` em bytes. Para caracteres e dados C binários, `buffer_length` especifica o tamanho do `*buffer` a ser utilizado como um dado de parâmetro no caso dele ser usado com `mysql_bind_param()` ou para retornar vários bytes ao buscar o resultado quando ele é usado com `mysql_bind_result()`.

`long *length` [input/output]

Ponteiro para o buffer para o tamanho do parâmetro. Quando a estrutura é usada como uma entrada para a ligação dos dados dos parâmetros, este argumento aponta para um buffer que, quando `mysql_execute()` é chamado, contém o tamanho do valor do parâmetro armazenado em `*buffer`. Isto é ignorado exceto para caracteres ou dados C binários.

Se o tamanho é um ponteiro nulo, o protocolo assume que todos os caracteres e dados binários são terminados com null.

When this structure is used in output binding, then `mysql_fetch()` return the the length of the data that is returned.

`bool *is_null` [input/output]

Indica se o dado do parâmetro é NULL ou o dado buscado é NULL.

MYSQL_TIME

Esta estrutura é utilizada para enviar e receber dados DATE, TIME e TIMESTAMP diretamente de/para o servidor.

A estrutura `MYSQL_TIME` contém os membros listados abaixo:

Membro	Tipo	Descrição
<code>year</code>	unsigned int	Ano.
<code>month</code>	unsigned int	Mês.
<code>day</code>	unsigned int	Dia do mês.
<code>hour</code>	unsigned int	Hora do dia(TIME).
<code>minute</code>	unsigned int	Minutos.

<code>second</code>	<code>unsigned int</code>	Segundos.
<code>neg</code>	<code>my_bool</code>	Um parâmetro booleano para indicar se o tempo é negativo.
<code>second_part</code>	<code>unsigned long</code>	Parte fracionada do segundo(ainda não usado)

8.1.6 Visão Geral das Funções de Instruções Preparadas da API C

The functions available in the prepared statements are listed here and are described in greater detail in the later section. See [\[C API Prepared statement functions\]](#), page [\[undefined\]](#).

Função	Descrição
<code>mysql_prepare()</code>	Prepara uma string SQL para execução.
<code>mysql_param_count()</code>	Retorna o número de parâmetros em uma instrução SQL preparada.
<code>mysql_prepare_result()</code>	Retorna meta informação de instruções preparadas em forma de um conjunto de resultados.
<code>mysql_bind_param()</code>	Liga um buffer para marcador de parâmetro em um instrução SQL preparada.
<code>mysql_execute()</code>	Executa a instrução preparada.
<code>mysql_stmt_affected_rows()</code>	Retorna o número de registros alterados/deletados/inseridos pelo última consulta UPDATE, DELETE, ou INSERT.
<code>mysql_bind_result()</code>	Liga o buffers de dados da aplicação em colunas no resultado da consulta.
<code>mysql_stmt_store_result()</code>	Retorna o resultado completo para o cliente.
<code>mysql_stmt_data_seek()</code>	Busca um registro arbitrário no resultado de uma consulta.
<code>mysql_stmt_row_seek()</code>	Busca por um registro no resultado de uma busca, utilizando o valor reotornado de <code>mysql_stmt_row_tell()</code> .
<code>mysql_stmt_row_tell()</code>	Retorna a posição do cursor de registro.
<code>mysql_stmt_num_rows()</code>	Retorna o total de registros do resultado de uma instrução armazenada.
<code>mysql_fetch()</code>	Busca o próximo conjunto de dados do resultado e retorna os dados para todas as colunas limites.
<code>mysql_stmt_close()</code>	Libera a memória usada pela instrução preparada.
<code>mysql_stmt_errno()</code>	Retorna o número de erro para a última instrução executada.
<code>mysql_stmt_error()</code>	Retorna a mensagem de erro para a última instrução executada.

mysql_send_long_data() Envia dados longos em blocos para o servidor.

Chama `mysql_prepare()` para preparar e iniciar o manipulador de instruções, chama `mysql_bind_param()` para fornecer os dados do parâmetro e chama `mysql_execute()` para executar a consulta. Você pode repetir o `mysql_execute()` alterando o valor do parâmetro do buffer respectivo fornecido por `mysql_bind_param()`.

No caso do consulta ser uma instrução `SELECT` ou qualquer outra consulta que retorne um resultado, `mysql_prepare()` também retornará a informação dos meta dados do resultado na forma de um resultado `MYSQL_RES` através de `mysql_prepare_result()`.

Você pode fornecer o buffer de resultado usando `mysql_bind_result()`, assim `mysql_fetch()` retornará automaticamente os dados para este buffer. Esta busca é feita registro a registro.

Você também pode enviar o texto ou dado binário em blocos para o servidor utilizando `mysql_send_long_data()`, especificando a opção `is_long_data=1` ou `length=MYSQL_LONG_DATA` ou `-2` na estrutura `MYSQL_BIND` fornecida com `mysql_bind_param()`.

Assim que a instrução executada acabar, ela deve ser liberada usando `mysql_stmt_close` para liberar todos os recursos alocados para o manipulador de instrução.

Execution Steps:

Para prepara e executar uma instrução, a aplicação:

- Chama **mysql_prepare()** e passa uma string contendo uma instrução SQL. Em um preparo com sucesso, `mysql_prepare` retorna o manipulador de instrução válido para a aplicação.
- Se a consulta retorna um resultado, **mysql_prepare_result** retorna a informação de meta dados do resultado.
- Define o valor de qualquer parâmetro usando **mysql_bind_param**. Todos os parâmetros devem ser definidos. Senão será retornado um erro ou produzido um resultado inesperado.
- Chama **mysql_execute()** para executar a instrução.
- Repete os passos 2 e 3 se necessário, alterando o valor do parâmetro e re-executando a instrução.
- Liga o buffer de dados para retornar o valor do registro, se for um resultado de uma consulta; usando **mysql_bind_result()**.
- Busca os dados para o buffer registro a registro chamando **mysql_fetch()** repetidas vezes até não haver mais registros.
- Quando **mysql_prepare()** é chamado, no protocolo cliente/servidor do MySQL:
 - O servidor analisa a consulta e envia o status de ok de volta para o cliente atribuindo uma identificação a instrução. Ele também envia um número total de parâmetros, contagem de colunas e sua meta informação se for um resultado orientado a consulta. Toda a sintaxe e semântica da consulta é verificada esta chamada pelo servidor.
 - O cliente utiliza esta identificação da instrução para as demais execuções, assim o servidor identifica a instrução dentre outras existentes. Agora o cliente aloca um manipulador de instruções com esta identificação e a retorna para a aplicação.

- Quando `mysql_execute()` é chamado, no protocolo cliente/servidor do MySQL:
 - O cliente utiliza o manipulador de instruções e envia o dado do parâmetro para o servidor.
 - O servidor identifica a instrução usando a identificação fornecida pelo cliente, e substitui o marcador do parâmetro com o dado fornecido mais recente e executa a consulta. Se é retornado um resultado, envia o dado de volta para o cliente, senão envia o status de OK como número total de registros alterados/deletados/inseridos.
- Quando `mysql_fetch()` é chamado, no protocolo cliente/servidor do MySQL:
 - O cliente lê os dados do pacote registro por registro e o coloca no buffer de dados da aplicação fazendo as conversões necessárias. Se o o tipo do buffer de aplicação é o mesmo do tipo do campo, as conversões seguem em frente.

Você pode obter o código e mensagem de erro da instrução utilizando `mysql_stmt_errno()` e `mysql_stmt_error()` respectivamente.

8.1.7 Descrição das Funções de Instrução Preparada da API C

Você precisa das seguintes funções quando quiser preparar e executar as consultas.

8.1.7.1 `mysql_prepare()`

```
MYSQL_STMT * mysql_prepare(MYSQL *mysql, const char *query, unsigned long
length)
```

Descrição

Prepara a consulta SQL apontada pela string com terminação em nulo 'query'. A consultadeve consistir de uma única instrução SQL. Você não deve adicionar ponto e virgula (;) ou \g a instrução.

A aplicação pode incluir um ou mais marcadores de parâmetro na instrução SQL. Para incluir um marcador de parâmetro, a aplicação embute uma exclamação (?) na string SQL na posição aprpriada.

Os marcadores só são válidos em certos lugares na instrução SQL. Por exemplo, eles não são permitidos em lista select (a lista de colunas retornadas por uma instrução SELECT), nem são permitidas como ambos operadores de operação binária assim como o sinal de igual (=), porque seria impossível determinar o tipo do parâmetro. Em geral, parâmetros são válidos somente em instrução de Linguagem de Manipulação de Dados (Data Manipulation Language-DML), e não em instruções de Linguagem de Definição de Dados (Data Defination Language-DDL).

Os marcadores de parâmetro limitam variáveis de aplicações utilizando `mysql_bind_param()`.

Valor Retornado

MYSQL_STMT se o preparo obteve sucesso. NULL se ocorreu um erro.

Erros

CR_COMMANDS_OUT_OF_SYNC

Os comando foram executados em uma ordem inapropriada.

CR_OUT_OF_MEMORY

Falta de memória

CR_SERVER_GONE_ERROR

O servidor MySQL foi finalizado.

CR_SERVER_LOST

A conexão ao servidor MySQL foi perdida durante a consulta.

CR_UNKNOWN_ERROR

Um erro desconhecido ocorreu.

Se o preparo não obteve sucesso, ex. quando `mysql_prepare()` retorna uma instrução NULL, os erros podem ser obtidos chamando `mysql_error()`.

Exemplo

Para o uso de `mysql_prepare()` consulte o exemplo de [\[mysql_execute\]](#), [page \[mysql_execute\]](#).

8.1.7.2 `mysql_param_count()`

```
unsigned int mysql_param_count(MYSQL_STMT *stmt)
```

Descrição

Retorna o número de marcadores de parâmetros presentes na consulta preparada.

Valor Retornado

Um unsigned integer (inteiro sem sinal) representando o número de parâmetros em uma instrução.

Erros

Nenhum.

Exemplo

Para utilizar `mysql_param_count()` consulte o exemplo de [\[mysql_execute\]](#), [page \[mysql_execute\]](#).

8.1.7.3 `mysql_prepare_result()`

```
MYSQL_RES *mysql_prepare_result(MYSQL_STMT *stmt)
```

Descrição

Se `mysql_prepare()` retornou um resultado de uma consulta, `mysql_prepare_result()` retorna o resultado dos meta dados na forma de uma estrutura `MYSQL_RES`; que também pode ser usada para processar a meta informação como o número total de campos e informação de campos individuais. O resultado retornado pode ser passado como um argumento para qualquer um dos campos com base na API para processar informação do resultado dos meta dados com:

- `mysql_num_fields()`
- `mysql_fetch_field()`
- `mysql_fetch_field_direct()`
- `mysql_fetch_fields()`
- `mysql_field_count()`
- `mysql_field_seek()`
- `mysql_field_tell()` and
- `mysql_free_result()`

Valor Retornado

Uma estrutura de resultado `MYSQL_RES`. `NULL` se nenhuma meta informação existe na consulta preparada.

Erros

`CR_OUT_OF_MEMORY`

Falta de memória

`CR_UNKNOWN_ERROR`

Ocorreu um erro desconhecido

Exemplo

Para utilizar `mysql_prepare_result()` consulte o exemplo de [\[mysql_fetch\]](#), [page](#)

8.1.7.4 `mysql_bind_param()`

```
int mysql_bind_param(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

Descrição

`mysql_bind_param` é utilizado para ligar dados para os marcadores de parâmetros na instrução SQL com `mysql_prepare`. Ele utiliza a estrutura `MYSQL_BIND` para fornecer os dados.

Os tipos de buffers suportados são:

- `MYSQL_TYPE_TINY`

- MYSQL_TYPE_SHORT
- MYSQL_TYPE_LONG
- MYSQL_TYPE_LONGLONG
- MYSQL_TYPE_FLOAT
- MYSQL_TYPE_DOUBLE
- MYSQL_TYPE_TIME
- MYSQL_TYPE_DATE
- MYSQL_TYPE_DATETIME
- MYSQL_TYPE_TIMESTAMP
- MYSQL_TYPE_STRING
- MYSQL_TYPE_VAR_STRING
- MYSQL_TYPE_TINY_BLOB
- MYSQL_TYPE_MEDIUM_BLOB
- MYSQL_TYPE_LONG_BLOB

Valor Retornado

Zeros se a ligação foi obtida com sucesso. Diferente de zero se ocorrer um erro.

Erros

CR_NO_PREPARE_STMT

Não existem instruções preparadas

CR_NO_PARAMETERS_EXISTS

Não existem parâmetros para ligar

CR_INVALID_BUFFER_USE

Indica se a ligação fornecerá dados longos em blocos e se o tipo de buffer é binário ou não é uma string.

CR_UNSUPPORTED_PARAM_TYPE

A conversão não é suportada, possivelmente o `buffer_type` é inválido ou não é um dos tipos suportados listados acima.

CR_OUT_OF_MEMORY

Falta de memória

CR_UNKNOWN_ERROR

Um erro desconhecido ocorreu.

Exemplo

Para utilizar `mysql_bind_param()` consulte o exemplo de [mysql_execute](#), [page](#).

8.1.7.5 `mysql_execute()`

```
int mysql_execute(MYSQL_STMT *stmt.
```

Descrição

`mysql_execute()` executa a consulta preparada associada ao manipulador de instruções. O valor do marcador de parâmetros será enviado para o servidor durante esta chamada, assim que o servidor substituir marcadores com os novos dados fornecidos.

Se a instrução é um `UPDATE`, `DELETE`, ou `INSERT`, o número total de `changed/deleted/inserted` pode ser encontrado chamando `mysql_stmt_affected_rows`. Se este é um resultado de uma consulta, deve se chamar `mysql_fetch()` para buscar dados previamente para fazer qualquer outra chamada que resulte em um processamento de consulta. Para mais informações sobre como buscar dados binários, consulte [\[mysql_fetch\]](#), [page \[mysql_fetch\]](#).

Valor Retornado

`mysql_execute()` retorna os seguintes valores:

Valor de Retorno	Descrição
0	Sucesso
1	Erro ocorrido. Código e mensagem de erro podem ser obtidos chamando <code>mysql_stmt_errno()</code> e <code>mysql_stmt_error()</code> .

Erros

`CR_NO_PREPARE_QUERY`

Nenhuma consulta preparada previamente para execução

`CR_ALL_PARAMS_NOT_BOUND`

Não foram fornecidos todos os dados de parâmetros.

`CR_COMMANDS_OUT_OF_SYNC`

Os comandos foram executados em uma ordem inapropriada.

`CR_OUT_OF_MEMORY`

Falta de memória

`CR_SERVER_GONE_ERROR`

O servidor MySQL foi finalizado.

`CR_SERVER_LOST`

A conexão ao servidor MySQL foi perdida durante a consulta.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

Exemplo

O seguinte exemplo explica o uso de `mysql_prepare`, `mysql_param_count`, `mysql_bind_param`, `mysql_execute` e `mysql_stmt_affected_rows()`.

```
MYSQL_BIND  bind[3];
MYSQL_STMT  *stmt;
```

```

ulonglong    affected_rows;
long         length;
unsigned int  param_count;
int          int_data;
short        small_data;
char         str_data[50], query[255];
my_bool      is_null;

/* Define o modo autocommit como true (verdadeiro) */
mysql_autocommit(mysql, 1);

if (mysql_query(mysql, "DROP TABLE IF EXISTS test_table"))
{
    fprintf(stderr, "\n drop table failed");
    fprintf(stderr, "\n %s", mysql_error(mysql));
    exit(0);
}
if (mysql_query(mysql, "CREATE TABLE test_table(col1 INT, col2 varchar(50), \
                                                col3 smallint, \
                                                col4 timestamp(14))"))
{
    fprintf(stderr, "\n create table failed");
    fprintf(stderr, "\n %s", mysql_error(mysql));
    exit(0);
}

/* Prepara uma consulta insert com 3 parâmetros */
strmov(query, "INSERT INTO test_table(col1,col2,col3) values(?,?,?)");
if (!(stmt = mysql_prepare(mysql, query, strlen(query))))
{
    fprintf(stderr, "\n prepare, insert failed");
    fprintf(stderr, "\n %s", mysql_error(mysql));
    exit(0);
}
fprintf(stdout, "\n prepare, insert successful");

/* Obtém a contagem de parâmetros para a instrução */
param_count = mysql_param_count(stmt);

fprintf(stdout, "\n total parameters in insert: %d", param_count);
if (param_count != 3) /* valida a contagem de parâmetros */
{
    fprintf(stderr, "\n invalid parameter count returned by MySQL");
    exit(0);
}

/* Liga os dados aos parâmetros */

/* PARTE INTEGER */

```

```

bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= 0;
bind[0].length= 0;

/* PARTE STRING */
bind[1].buffer_type= MYSQL_TYPE_VAR_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= sizeof(str_data);
bind[1].is_null= 0;
bind[1].length= 0;

/* PARTE SMALLINT */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null;
bind[2].length= 0;
is_null= 0;

/* Liga os buffers */
if (mysql_bind_param(stmt, bind))
{
    fprintf(stderr, "\n param bind failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* Especifica os dados */
int_data= 10;          /* integer */
strcpy(str_data,"MySQL"); /* string */

/* INSERE dados SMALLINT como NULL */
is_null= 1;

/* Executa a instrução insert - 1*/
if (mysql_execute(stmt))
{
    fprintf(stderr, "\n execute 1 failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    fprintf(stderr, "\n send a bug report to bugs@lists.mysql.com, by asking why th
    exit(0);
}

/* Obtem o números total de registros afetados */
affected_rows= mysql_stmt_affected_rows(stmt);

fprintf(stdout, "\n total affected rows: %lld", affected_rows);
if (affected_rows != 1) /* valida linha afetadas */

```

```

{
    fprintf(stderr, "\n invalid affected rows by MySQL");
    exit(0);
}

/* Re-executa o insert, alterando os valores */
int_data= 1000;
strcpy(str_data,"The most popular open source database");
small_data= 1000;          /* smallint */
is_null= 0;                /* reset NULL */

/* Executa a instrução insert - 2*/
if (mysql_execute(stmt))
{
    fprintf(stderr, "\n execute 2 failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* Obtem o números total de registros afetados */
affected_rows= mysql_stmt_affected_rows(stmt);

fprintf(stdout, "\n total affected rows: %lld", affected_rows);
if (affected_rows != 1) /* validate affected rows */
{
    fprintf(stderr, "\n invalid affected rows by MySQL");
    exit(0);
}

/* Fecha a instrução */
if (mysql_stmt_close(stmt))
{
    fprintf(stderr, "\n failed while closing the statement");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* APAGA A TABELA */
if (mysql_query(mysql,"DROP TABLE test_table"))
{
    fprintf(stderr, "\n drop table failed");
    fprintf(stderr, "\n %s", mysql_error(mysql));
    exit(0);
}
fprintf(stdout, "Success, MySQL prepared statements are working!!!");

```

8.1.7.6 mysql_stmt_affected_rows()

```

ulonglong mysql_stmt_affected_rows(MYSQL_STMT *stmt)

```

Descrição

Retorna o número total de registros alterados pela última instrução executada. Pode ser chamada imediatamente depois de `mysql_execute()` para `UPDATE`, `DELETE`, ou instruções `INSERT`. Para instruções `SELECT`, `mysql_stmt_affected_rows` funciona como `mysql_num_rows()`.

Valor Retornado

Um integer (inteiro) maior que zero indica o número de registros afetados ou retornados. Zero indica que nenhum registro foi atualizado em uma instrução `UPDATE`, nenhum registro coincidiu com a cláusula `WHERE` na consulta ou que nenhuma consulta foi executada ainda. -1 indica que a consulta retornou um erro ou que, para uma consulta `SELECT`, `mysql_stmt_affected_rows()` foi chamado antes de chamar `mysql_fetch()`.

Erros

Nenhum.

Exemplo

Para utilizar `mysql_stmt_affected_rows()` consulte o exemplo de [\[mysql_execute\]](#), page [\[mysql_execute\]](#).

8.1.7.7 `mysql_bind_result()`

```
my_bool mysql_bind_result(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

Descrição

`mysql_bind_result()` é usado para associar, ou ligar, colunas no resultados ao buffer de dados e buffer de tamanho. Quando `mysql_fetch()` é chamado para buscar dados, o protocolo cliente MySQL retorna os para as colunas limitadas no buffer especificado.

Note que todas as colunas devem ser limitadas antes de chamar `mysql_fetch()` no caso de buscar os dados para o buffers; senão `mysql_fetch()` simplesmente ignorado os dados trazidos; os buffers devem se suficientes para guardar os dados já que o protocolo não retorna dados em blocos.

Uma coluna pode ser limitada a qualquer hora, mesmo depois do dados ter sido buscado do resultado. A nova ligação tem efeito na próxima vez em que `mysql_fetch()` é chamado. Por exemplo, suponha que uma aplicação liga a coluna em um resultado e chama `mysql_fetch()`. O protocolo MySQL retorna dados em buffers limitados. Agora suponha que a aplicação ligue a coluna a um diferente conjunto de buffers, então o protocolo não coloca os dados do registro recém buscados em um novo buffer limitado. Ele o faz que o próximo `mysql_fetch()` for chamado.

Para ligar uma coluna, uma aplicação chama `mysql_bind_result()` e passa o tipo, o endereço e o endereço do buffer do tamanho.

Os tipos de buffers suportados são:

- MYSQL_TYPE_TINY
- MYSQL_TYPE_SHORT
- MYSQL_TYPE_LONG
- MYSQL_TYPE_LONGLONG
- MYSQL_TYPE_FLOAT
- MYSQL_TYPE_DOUBLE
- MYSQL_TYPE_TIME
- MYSQL_TYPE_DATE
- MYSQL_TYPE_DATETIME
- MYSQL_TYPE_TIMESTAMP
- MYSQL_TYPE_STRING
- MYSQL_TYPE_VAR_STRING
- MYSQL_TYPE_BLOB
- MYSQL_TYPE_TINY_BLOB
- MYSQL_TYPE_MEDIUM_BLOB
- MYSQL_TYPE_LONG_BLOB

Valor Retornado

Zero se a ligação obteve sucesso. Diferente de zero se ocorreu um erro.

Erros

CR_NO_PREPARE_STMT

Não existe instruções preparadas

CR_UNSUPPORTED_PARAM_TYPE

A conversão não é suportada, possivelmente o `buffer.type` é inválido ou não nanlista dos tipos de buffers suportados

CR_OUT_OF_MEMORY

Falta de memória

CR_UNKNOWN_ERROR

Ocorreu um erro desconhecido

Exemplo

Para utilizar `mysql_bind_result()` consulta o exemplo de [\[mysql_fetch\]](#), [page](#)

8.1.7.8 `mysql_stmt_store_result()`

```
int mysql_stmt_store_result(MYSQL_STMT *stmt)
```

Descrição

Você deve chamar `mysql_stmt_store_result()` para cada query que retornar dados com sucesso (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`), e só se você quiser armazenar todo o resultado no buffer no cliente, assim que a chamada `mysql_fetch()` subsequente retornar os dados em buffers.

Você não precisa chamar `mysql_stmt_store_result()` para outras consultas, mas não causará nenhum dano ou queda de performance em todo caso. Você pode detectar se a consulta não gera um resultado verificado se `mysql_prepare_result()` retorna 0. Para mais informações consulte [\[mysql_prepare_result\]](#), page [\[mysql_prepare_result\]](#).

Valor Retornado

Zero se o resultado foi armazenado em buffer com sucesso ou Diferente de zero em caso de erro.

Erros

`CR_COMMANDS_OUT_OF_SYNC`

Os comando foram executados em uma ordem inapropriada.

`CR_OUT_OF_MEMORY`

Falta de memoria.

`CR_SERVER_GONE_ERROR`

O servidor MySQL foi finalizado.

`CR_SERVER_LOST`

A conexão ao servidor MySQL foi perdida durante a consulta.

`CR_UNKNOWN_ERROR`

Um erro desconhecido ocorreu.

8.1.7.9 `mysql_stmt_data_seek()`

```
void mysql_stmt_data_seek(MYSQL_STMT *stmt, my_ulonglong offset)
```

Descrição

Busca um registro arbitrário no resultado de uma instrução. Isto exige que a estrutura do resultado da instrução contenha todo o resultado da última consulta executada, assim `mysql_stmt_data_seek()` pode ser usada em conjunto apenas com `mysql_stmt_store_result()`.

O offset deve ser um valor na faixa de 0 a `mysql_stmt_num_rows(stmt)-1`.

Valor Retornado

Nenhum.

Erros

Nenhum.

8.1.7.10 `mysql_stmt_row_seek()`

`MYSQL_ROW_OFFSET mysql_stmt_row_seek(MYSQL_STMT *stmt, MYSQL_ROW_OFFSET offset)`

Descrição

Define o cursor de linha com um registro arbitrário em um resultado de instrução. Isto exige que a estrutura do resultado contenha todo o resultado da consulta, assim `mysql_stmt_row_seek()` pode ser usado em conjunto apenas com `mysql_stmt_store_result()`. O offset deve ser um valor retornado de uma chamada a `mysql_stmt_row_tell()` ou a `mysql_stmt_row_seek()`. Este valor não é simplesmente um número de linha; se você quiser buscar um registro em um resultado usando um número de linha utilize `mysql_stmt_data_seek()`.

Valor Retornado

O valor anterior do cursor de linha. Este valor pode ser passado a uma chamada subsequente de `mysql_stmt_row_seek()`.

Erros

Nenhum.

8.1.7.11 `mysql_stmt_row_tell()`

`MYSQL_ROW_OFFSET mysql_stmt_row_tell(MYSQL_STMT *stmt)`

Descrição

Retorna a posição corrente do cursor de linha para o último `mysql_fetch()`. Este valor pode ser usado como um argumento para `mysql_stmt_row_seek()`.

Você deve usar `mysql_stmt_row_tell()` somente depois de `mysql_stmt_store_result()`.

Valor Retornado

O offset atual do cursor de linha.

Erros

Nenhum.

8.1.7.12 `mysql_stmt_num_rows()`

`my_ulonglong mysql_stmt_num_rows(MYSQL_STMT *stmt)`

Descrição

Retorna o número de registros no resultado.

O uso de `mysql_stmt_num_rows()` depende de se você utilizou `mysql_stmt_store_result()` para armazenar todo o resultado no manipulador de instruções ou não.

Se você utilizou `mysql_stmt_store_result()`, `mysql_stmt_num_rows()` pode ser chamado imediatamente.

Valor Retornado

O número de linhas no resultado.

Erros

Nenhum.

8.1.7.13 `mysql_fetch()`

```
int mysql_fetch(MYSQL_STMT *stmt)
```

Descrição

`mysql_fetch()` retorna o próximo conjunto de registros no resultado. Ele pode ser chamado apenas enquanto existir o conjunto de resultados. Per exemplo, depois de uma chamada de `mysql_execute()` que cria o resultado ou depois de `mysql_stmt_store_result()`, que é chamado depois de `mysql_execute()` para armazenar todo o resultado.

Se buffers de linhas são limitados usando `mysql_bind_result()`, ele retorna os dados neste buffer para todas as colunas no registro atual e os tamanhos são retornados para o apontador do tamanho.

Note que, todas as colunas devem ser limitadas pela aplicação.

Se o dado buscado é um dado NULL, então o valor `is_null` de `MYSQL_BIND` contém TRUE (VERDASDEIRO), 1, senão o dado e seu tamanho é retornado para as variáveis `*buffer` e `*length` baseada no tipo de buffer especificado pela aplicação, Todos os tipos numéricos, float e double tem o tamanho fixo (em bytes) como mostrado abaixo:

Tipo	Tamanho
<code>MYSQL_TYPE_TINY</code>	1
<code>MYSQL_TYPE_SHORT</code>	2
<code>MYSQL_TYPE_LONG</code>	4
<code>MYSQL_TYPE_FLOAT</code>	4
<code>MYSQL_TYPE_LONGLONG</code>	8
<code>MYSQL_TYPE_DOUBLE</code>	8
<code>MYSQL_TYPE_TIME</code>	<code>sizeof(MYSQL_TIME)</code>
<code>MYSQL_TYPE_DATE</code>	<code>sizeof(MYSQL_TIME)</code>
<code>MYSQL_TYPE_DATETIME</code>	<code>sizeof(MYSQL_TIME)</code>
<code>MYSQL_TYPE_TIMESTAMP</code>	<code>sizeof(MYSQL_TIME)</code>
<code>MYSQL_TYPE_STRING</code>	<code>tam_dados</code>

MYSQL_TYPE_VAR_STRING	tam_dados
MYSQL_TYPE_BLOB	tam_dados
MYSQL_TYPE_TINY_BLOB	tam_dados
MYSQL_TYPE_MEDIUM_BLOB	tam_dados
MYSQL_TYPE_LONG_BLOB	tam_dados

onde `*tam_dados` não é nada mais que 'Tamanho atual do dado'.

Valor Retornado

Valor retornado	Descrição
0	Sucesso, o dado foi buscado para o buffers de dados da aplicação.
1	Ocorreu um erro. O código e a mensagem de erro podem ser obtidos chamando <code>mysql_stmt_errno()</code> e <code>mysql_stmt_error()</code> .
100, MYSQL_NO_DATA	Não existem mais registros/dados

Erros

CR_COMMANDS_OUT_OF_SYNC	Os comando foram executados em uma ordem inapropriada.
CR_OUT_OF_MEMORY	Falta de memoria.
CR_SERVER_GONE_ERROR	O servidor MySQL foi finalizado.
CR_SERVER_LOST	A conexão ao servidor MySQL foi perdida durante a consulta.
CR_UNKNOWN_ERROR	Um erro desconhecido ocorreu.
CR_UNSUPPORTED_PARAM_TYPE	Se o tipo de buffer é <code>MYSQL_TYPE_DATE</code> , <code>DATETIME</code> , <code>TIME</code> , ou <code>TIMESTAMP</code> ; e se o tipo de dado não é <code>DATE</code> , <code>TIME</code> , <code>DATETIME</code> ou <code>TIMESTAMP</code> . Todos os outros erros de conversão não suportada são retornados de <code>mysql_bind_result()</code> .

Exemplo

O seguinte exemplo explica o uso de `mysql_prepare_result`, `mysql_bind_result()`, e `mysql_fetch()`

```
MYSQL_STMT *stmt;
MYSQL_BIND bind[2];
MYSQL_RES *result;
int int_data;
```

```

long          int_length, str_length;
char          str_data[50];
my_bool      is_null[2];

query= "SELECT col1, col2 FROM test_table WHERE col1= 10");
if (!(stmt= mysql_prepare(mysql, query, strlen(query)))
{
    fprintf(stderr, "\n prepare failed");
    fprintf(stderr, "\n %s", mysql_error(mysql));
    exit(0);
}

/* Obtém meta informação dos campos */
if (!(result= mysql_prepare_result(stmt)))
{
    fprintf(stderr, "\n prepare_result failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

fprintf(stdout, "Total fields: %ld", mysql_num_fields(result));

if (mysql_num_fields(result) != 2)
{
    fprintf(stderr, "\n prepare returned invalid field count");
    exit(0);
}

/* Executa a consulta SELECT */
if (mysql_execute(stmt))
{
    fprintf(stderr, "\n execute failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* Liga o buffer de dados do resultado */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= &is_null[0];
bind[0].length= &int_length;

bind[1].buffer_type= MYSQL_TYPE_VAR_STRING;
bind[1].buffer= (void *)str_data;
bind[1].buffer_length= 20;
bind[1].is_null= &is_null[1];
bind[1].length= &str_length;

if (mysql_bind_result(stmt, bind))

```

```

{
    fprintf(stderr, "\n bind_result failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* Agora busca os dados para o buffer*/
if (mysql_fetch(stmt))
{
    fprintf(stderr, "\n fetch failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

if (is_null[0])
    fprintf(stdout, "\n Col1 data is NULL");
else
    fprintf(stdout, "\n Col1: %d, length: %ld", int_data, int_length);

if (is_null[1])
    fprintf(stdout, "\n Col2 data is NULL");
else
    fprintf(stdout, "\n Col2: %s, length: %ld", str_data, str_length);

/* chama mysql_fetch de novo */
if (mysql_fetch(stmt) != MYSQL_NO_DATA)
{
    fprintf(stderr, "\n fetch return more than one row);
    exit(0);
}

/* Libera a meta informação do resultado preparado */
mysql_free_result(result);

/* Libera o manipulador de instrução */
if (mysql_stmt_free(stmt))
{
    fprintf(stderr, "\n failed to free the statement handle);
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

```

8.1.7.14 mysql_send_long_data()

```

int mysql_send_long_data(MYSQL_STMT *stmt, unsigned int parameter_number,
const char *data, ulong length)

```

Descrição

Permite que um aplicação envie os dados em blocos para o servidor. Esta função pode ser usada para enviar valores de dados binários e caracteres em partes para uma coluna (deve ser text ou blob) com um tipo de dado binário ou de caracter.

`data` é um ponteiro para o buffer contendo os dados atuais para o parâmetro representado por `parameter_number`. `length` indica a quantidade de dados a ser enviado em bytes.

Valor Retornado

Zero se os dados são enviados com sucesso para o servidor. Diferente de zero se ocorrer um erro.

Erros

<code>CR_INVALID_PARAMETER_NO</code>	Número de parâmetro inválido
<code>CR_COMMANDS_OUT_OF_SYNC</code>	Os comando foram executados em uma ordem inapropriada.
<code>CR_OUT_OF_MEMORY</code>	Falta de memoria.
<code>CR_SERVER_GONE_ERROR</code>	O servidor MySQL foi finalizado.
<code>CR_UNKNOWN_ERROR</code>	Um erro desconhecido ocorreu.

Example

O exemplo seguinte explica como enviar os dados em blocos para um coluna do tipo text:

```

MYSQL_BIND bind[1];
long         length;

query= "INSERT INTO test_long_data(text_column) VALUES(?)";
if (!mysql_prepare(mysql, query, strlen(query))
{
    fprintf(stderr, "\n prepare failed");
    fprintf(stderr, "\n %s", mysql_error(mysql));
    exit(0);
}
memset(bind, 0, sizeof(bind));
bind[0].buffer_type= MYSQL_TYPE_STRING;
bind[0].length= &length;
bind[0].is_null= 0;

/* Liga os buffers */

```

```

if (mysql_bind_param(stmt, bind))
{
    fprintf(stderr, "\n param bind failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* Envia os dados em blocos para o servidor */
if (!mysql_send_long_data(stmt,1,"MySQL",5))
{
    fprintf(stderr, "\n send_long_data failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* Envia o próximo pedaço de dados */
if (mysql_send_long_data(stmt,1," - The most popular open source database",40))
{
    fprintf(stderr, "\n send_long_data failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* Agora executa a consulta */
if (mysql_execute(stmt))
{
    fprintf(stderr, "\n mysql_execute failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

```

This inserts the data, "MySQL - The most popular open source database" to the field 'text_column'.

8.1.7.15 mysql_stmt_close()

```
my_bool mysql_stmt_close(MYSQL_STMT *)
```

Descrição

Fecha a instrução preparada. `mysql_stmt_close()` também desaloca o manipulador de instruções apontado por `stmt`.

Se os resultados do consulta atual estão pendentes ou não foram lidos; o resultado da consulta é cancelado para que a próxima chamada possa ser executada.

Valor Retornado

Zero se a instrução for liberada com sucesso. Diferente de zero se ocorrer um erro.

Erros

CR_SERVER_GONE_ERROR

O servidor MySQL foi finalizado.

CR_UNKNOWN_ERROR

Ocorreu um erro desconhecido.

Exemplo

Para utilizar `mysql_stmt_close()` consulte o exemplo de [\[mysql_execute\]](#), [page](#).

8.1.7.16 `mysql_stmt_errno()`

```
unsigned int mysql_stmt_errno(MYSQL_STMT *stmt)
```

Descrição

Para a instrução especificada por `stmt`, `mysql_stmt_errno()` retorna o código de erro para a função de instruções da API chamada mais recentemente. Um valor de retorno de zero significa que não ocorreu nenhum erro. Números de mensagens de erro do cliente estão listadas no arquivo cabeçalho `errmsg.h` do MySQL. `lient error message numbers are listed in the MySQL errmsg.h header file.` Números de mensagens de erro do servidor estão listado no arquivo `mysqld_error.h`. Na distribuição fonte do MySQL você pode encontra uma lista completa com mensagem de erros e números de erros no arquivo `Docs/mysqld_error.txt`

Valor Retornado

Um valor de código de erro. Zero se não ocorreu erro.

Erros

Nenhum

8.1.7.17 `mysql_stmt_error()`

```
char *mysql_stmt_error(MYSQL_STMT *stmt)
```

Descrição

Para a instrução especificada por `stmt`, `mysql_stmt_error()` retorna a mensagem de erro para a função de instrução da API chamada mais recentemente. Um string vazia ("") é retornado se não ocorreu nenhum erro. Isto significa que os seguintes comandos são equivalentes:

```
if (mysql_stmt_errno(stmt))
{
    // an error occurred
```

```

}

if (mysql_stmt_error(stmt))
{
    // an error occurred
}

```

A linguagem da mensagem de erro do cliente pode ser alterada recompilando a biblioteca cliente do MySQL. Atualmente você pode escolher mensagem de erros em diversas linguagens.

Valor Retornado

Um string contendo a descrição do erro. Uma string vazia se não ocorrer erros.

Erros

Nenhum

8.1.8 Manipulando a execução de múltiplas consultas na API C

A partir da versão 4.1, o MySQL suporta a execução de multi consultas em um único comando. Para utilizá-lo, você deve definir o parâmetro do cliente CLIENT_MULTI_QUERIES ao abrir a conexão.

Por padrão `mysql_query()` ou `mysql_real_query()` retornam apenas o status da primeira consulta e o status das consultas subsequentes podem ser processados usando `mysql_more_results()` e `mysql_next_result()`.

```

/* Conecta ao servidor com a opção CLIENT_MULTI_QUERIES */
mysql_real_query(..., CLIENT_MULTI_QUERIES);

/* Executa múltiplas consultas */
mysql_query(mysql, "DROP TABLE IF EXISTS test_table;\
CREATE TABLE test_table(id INT);\
INSERT INTO test_table VALUES(10);\
UPDATE test_table SET id=20 WHERE id=10;\
SELECT * FROM test_table;\
DROP TABLE test_table";
while (mysql_more_results(mysql))
{
    /* Processa todos os resultados */
    mysql_next_result(mysql);
    ...
    printf("total affected rows: %lld", mysql_affected_rows(mysql));
    ...
    if ((result= mysql_store_result(mysql))
    {
        /* Retorna um resultado e o processa */
    }
}

```

```
}

```

8.1.9 Manipulando DATE, TIME e TIMESTAMP na API C

Utilizando o novo protocolo binário do MySQL 4.1 e acima, pode se enviar e receber dados DATE, TIME e TIMESTAMP utilizando a estrutura `MYSQL_TIME`.

A estrutura `MYSQL_TIME` consiste dos seguintes membros:

- year
- month
- day
- hour
- minute
- second
- second_part

Para enviar dados, deve se usar as instruções preparadas com `mysql_prepare()` e `mysql_execute()`; e deve se ligar os parâmetros usando tipos como `MYSQL_TYPE_DATE` para processar valores de data (date), `MYSQL_TYPE_TIME` para hora (time) e `MYSQL_TYPE_DATETIME` ou `MYSQL_TYPE_TIMESTAMP` para datetime/timestamp usando `mysql_bind_param()` ao enviar e `mysql_bind_results()` enquanto recebe os dados.

Aqui está um exemplo simples; que insere dados DATE, TIME e TIMESTAMP.

```
MYSQL_TIME  ts;
MYSQL_BIND  bind[3];
MYSQL_STMT  *stmt;

    strmov(query, "INSERT INTO test_table(date_field, time_field,
                                                timestamp_field) VALUES(?,?,?");

    stmt= mysql_prepare(mysql, query, strlen(query));

    /* define a entrada do buffer com 3 parâmetros */
    bind[0].buffer_type= MYSQL_TYPE_DATE;
    bind[0].buffer= (char *)&ts;
    bind[0].is_null= 0;
    bind[0].length= 0;
    ..
    bind[1]= bind[2]= bind[0];
    ..

    mysql_bind_param(stmt, bind);

    /* fornece os dados a serme enviados na estrutura ts */
    ts.year= 2002;
    ts.month= 02;
    ts.day= 03;
```

```
ts.hour= 10;
ts.minute= 45;
ts.second= 20;

mysql_execute(stmt);
..
```

8.1.10 Descrição das Funções de Threaded da API C

Você precisa utilizar as seguintes funções quando quiser criar um cliente em uma thread. See [\[Clientes em threads\]](#), page [\[Clientes em threads\]](#).

8.1.10.1 my_init()

```
void my_init(void)
```

Descrição

Esta função precisa ser chamada uma vez pelo programa antes de se chamar qualquer função do MySQL. Ela inicializa algumas variáveis globais que o MySQL precisa. se você está usando uma biblioteca cliente de thread segura, também será feita uma chamada a `mysql_thread_init()` para esta thread.

Ela é chamada automaticamente por `mysql_init()`, `mysql_server_init()` e `mysql_connect()`.

Valor Retornado

Nenhum

8.1.10.2 mysql_thread_init()

```
my_bool mysql_thread_init(void)
```

Descrição

Esta função precisa ser chamada para cada thread criada para inicializar variáveis específicas de threads.

Ela é automaticamente chamada por `my_init()` e `mysql_connect()`.

Valor Retornado

Nenhum.

8.1.10.3 mysql_thread_end()

```
void mysql_thread_end(void)
```

Descrição

Esta função precisa ser chamada antes da chamada de `pthread_exit()` para liberar a memória alocada por `mysql_thread_init()`.

Note que a função **não é chamada automaticamente** pela biblioteca cliente. Deve ser chamada explicitamente para evitar perda de memória.

Valor Retornado

Nenhum.

8.1.10.4 `mysql_thread_safe()`

```
unsigned int mysql_thread_safe(void)
```

Descrição

Esta função indica se o cliente é compilado como uma thread segura.

Valor Retornado

1 se o cliente possui thread segura, 0 em outro caso.

8.1.11 Descrição de Funções do Servidor Embutido da API C

Você deve utilizar as seguintes funções se você quiser permitir que a sua aplicação seja ligada a biblioteca de servidor MySQL embutido. See [\[libmysqld\]](#), page [\[libmysqld\]](#). Se o programa é ligado com `-lmysqlclient` em vez de `-lmysqld`, estas funções não farão nada. Isto torna possível escolher entre usar o servidor MySQL embutido e um servidor stand-alone sem modificar nenhum código.

8.1.11.1 `mysql_server_init()`

```
int mysql_server_init(int argc, char **argv, char **groups)
```

Descrição

Esta função **deve** ser chamada uma vez no program usando o servidor embutido antes de se chamar qualquer outra função do MySQL. Ela inicia o servidor e inicializa qualquer subsistema (mysys, InnoDB, etc.) que o servidor utilize. Se esta função não for chamada, o programa irá falhar. Se você estiver usando o pacote DEBUG que vem com o MySQL, você deve chamar esta função depois de ter chamado `MY_INIT()`.

Os argumentos `argc` e `argv` são análogos aos argumentos para o `main()`. O primeiro elemento de `argv` é ignorado (ele contém normalmente, o nome do programa). por conveniência, `argc` pode ser 0 (zero) se não houver argumentos de linha de comando para o servidor. `mysql_server_init()` faz uma cópia dos argumentos, assim é seguro destruir `argv` ou `groups` depois da chamada.

A lista de strings terminadas em NULL em `groups` seleciona qual grupo no arquivo de opções será ativado. See [\[Arquivos de opções\]](#), page [\[Arquivos de opções\]](#). Por conveniência, `groups` deve ser NULL, caso no qual os grupos `[server]` e `[emdedd]` estarão ativos.

Exemplo

```
#include <mysql.h>
#include <stdlib.h>

static char *server_args[] = {
    "this_program",      /* this string is not used */
    "--datadir=.",
    "--key_buffer_size=32M"
};
static char *server_groups[] = {
    "embedded",
    "server",
    "this_program_SERVER",
    (char *)NULL
};

int main(void) {
    mysql_server_init(sizeof(server_args) / sizeof(char *),
                    server_args, server_groups);

    /* Use any MySQL API functions here */

    mysql_server_end();

    return EXIT_SUCCESS;
}
```

Valor Retornado

0 se okay, 1 se ocorrer um erro.

8.1.11.2 mysql_server_end()

```
void mysql_server_end(void)
```

Descrição

Esta função **deve** ser chamada no programa depois de todas as outras funções MySQL. Ela finaliza o servidor embutido.

Valor Retornado

Nenhum.

8.1.12 Questões e problemas comuns ao utilizar a API C

8.1.12.1 Porque Algumas Vezes `mysql_store_result()` Retorna NULL Após `mysql_query()` Retornar com Sucesso?

É possível para `mysql_store_result()` retornar NULL seguida de uma chamada com sucesso ao `mysql_query()`. Quando isto acontece, significa que uma das seguintes condições ocorreu:

- Existe uma falha no `malloc()` (por exemplo, se o resultado for muito grande).
- Os dados não podem ser lidos (ocorreu um erro na conexão).
- A consulta não retornou dados (por exemplo, ela era um `INSERT`, `UPDATE`, ou `DELETE`).

Você sempre pode verificar se a instrução devia produzir um resultado não vazio chamando `mysql_field_count()`. Se `mysql_field_count()` retornar zero, o resultado está vazio e a última consulta era uma instrução que não devia retornar valor (por exemplo, um `INSERT` ou um `DELETE`). Se `mysql_field_count()` retorna um valor diferente de zero, a instrução devia ter produzido um resultado não vazio. Veja a descrição da função `mysql_field_count()` para um exemplo.

Você pode testar um erro chamando `mysql_error()` ou `mysql_errno()`.

8.1.12.2 Que Resultados Posso Obter de uma Consulta?

Sobre o resultado retornado de uma consulta, você pode obter as seguintes informações:

- `mysql_affected_rows()` retorna o número de registros afetados pela última consulta ao se fazer uma `INSERT`, `UPDATE`, ou `DELETE`. Uma exceção é que se for utilizado `DELETE` sem uma cláusula `WHERE`, a tabela é recriada vazia, o que é mais rápido! Neste caso, `mysql_affected_rows()` retorna zero para o número de registros afetados.
- `mysql_num_rows()` retorna o número de registros em um resultado. Com `mysql_store_result()`, `mysql_num_rows()` pode ser chamado assim que `mysql_store_result()` retornar. Com `mysql_use_result()`, `mysql_num_rows()` só pode ser chamado depois de ter buscado todos os registros com `mysql_fetch_row()`.
- `mysql_insert_id()` retorna o ID gerado pela última consulta que inseriu um registro em uma tabela com índice `AUTO_INCREMENT`. See [\[mysql_insert_id\(\)\]](#), page [\[mysql_insert_id\(\)\]](#).
- Algumas consultas (`LOAD DATA INFILE ...`, `INSERT INTO ... SELECT ...`, `UPDATE`) retornam informações adicionais. O resultado é retornado por `mysql_info()`. Veja a descrição de `mysql_info()` para o formato da string que ela retornou. `mysql_info()` retorna um ponteiro NULL se não houver informações adicionais.

8.1.12.3 Como Posso Obter a ID Única para a Última Linha Inserida?

Se você inserir um registro em uma tabela contendo uma coluna que tiver o atributo `AUTO_INCREMENT`, você pode obter o ID gerado mais recentemente chamando a função `mysql_insert_id()`.

Você também pode recuperar o ID utilizando a função `LAST_INSERT_ID()` em uma string de consulta que foi passada a `mysql_query()`.

Você pode verificar se um índice `AUTO_INCREMENT` é usado executando o seguinte código. Ele também verifica se a consulta era um `INSERT` com um índice `AUTO_INCREMENT`:

```

if (mysql_error(&mysql)[0] == 0 &&
    mysql_num_fields(result) == 0 &&
    mysql_insert_id(&mysql) != 0)
{
    used_id = mysql_insert_id(&mysql);
}

```

O ID gerado mais recentemente é mantido no servidor em uma base por conexão. Ele não será alterado por outro cliente. Ele não será alterado mesmo se você atualizar outra coluna AUTO_INCREMENT com um valor não mágico (isto é, um valor que não é NULL e nem 0).

Se você quiser utilizar o ID que foi gerado por uma tabela e inserido em uma segunda tabela, você ode utilizar instruções SQL como esta:

```

INSERT INTO foo (auto,text)
VALUES(NULL,'text');           # gera ID inserindo NULL
INSERT INTO foo2 (id,text)
VALUES(LAST_INSERT_ID(),'text'); # usa ID na segunda tabela

```

8.1.12.4 Problemas com Ligação na API C

Ar ligar com a API C, os seguintes erros podem ocorrer em alguns sistemas:

```

gcc -g -o client test.o -L/usr/local/lib/mysql -lmysqlclient -lsocket -
lnsl

```

```

Undefined          first referenced
symbol             in file
floor              /usr/local/lib/mysql/libmysqlclient.a(password.o)
ld: fatal: Symbol referencing errors. No output written to client

```

Se isto acontecer em seu sistema, você deve incluir a biblioteca math adicionando `-lm` ao fim da linha de compilação/ligação.

8.1.13 Construindo Programas Clientes

Se você compilar clientes MySQL escritos por você mesmo ou obtido de terceiros, eles devem ser ligados utilizando a opção `-lmysqlclient -lz` no comando de ligação. Você também pode precisar de especificar uma opção `-L` para dizer ao ligado onde encontrar a biblioteca. Por exemplo, se a biblioteca é instalada em `'/usr/local/mysql/lib'`, use `-L/usr/local/mysql/lib -lmysqlclient -lz` no comando de ligação.

Para clientes que utilizam arquivos de cabeçalho do MySQL, pode ser necessário especificar a opção `-I` ao compilá-los, (por exemplo, `-I/usr/local/mysql/include`), assim o compilador pode encontrar o arquivo de cabeçalho.

Para o mostrado acima de forma simples no Unix, fornecemos o script `mysql_config` para você. See [\[mysql_config\]](#), page [\[mysql_config\]](#).

Você pode utilizá-lo para compilar o cliente MySQL como a seguir:

```

CFG=/usr/local/mysql/bin/mysql_config
sh -c "gcc -o progname '$CFG --cflags' progname.c '$CFG --libs'"

```

`sh -c` é necessário para fazer com que a shell não trate a saída de `mysql_config` como uma palavra.

8.1.14 Como Fazer um Cliente em Threads

A biblioteca cliente é quase segura com threads. O maior problema é que as subrotinas em ‘net.c’ que leem dos sockets não são seguras a interrupções. Isto foi feito pensando que você pudesse desejar ter o seu próprio alarme que possa quebrar uma longa leitura no servidor. Se você instalar manipuladores de interrupção para a interrupção SIGPIPE, o manipulador socket deve ser segura com threads.

Nos binários antigos que distribuimos em nosso web site (<http://www.mysql.com/>), as bibliotecas clientes não estão normalmente compiladas com a opção de segurança com thread (os binários são compilados com segurança com thread por padrão). Distribuições binárias mais novas devem ter uma biblioteca normal e uma segura com threads.

Para termos um cliente em threads onde você pode interromper o cliente a partir de outras threads a definir tempo limites ao falar com o servidor MySQL, você deve utilizar as bibliotecas `-mysys`, `-mystrings`, e `-ldbug` e o código `net_serv.o` que o servidor utiliza. Se você não precisar de interrupções ou de tempos limites, você pode apenas compilar um biblioteca cliente (`mysqlclient_r`) segura com threads e utilizá-las. See [\[API C MySQL\]](#), page [\[undefined\]](#). Neste caso você não precisa se preocupar com o arquivo objeto `net_serv.o` ou outras bibliotecas MySQL.

Quando usar um cliente em thread e você quiser utilizar tempos limite e interrupções, você pode ter um grande uso das rotinas no arquivo ‘thr_alarm.c’. Se você estiver utilizando rotinas da biblioteca `mysys`, a única coisa que você deve lembrar é de chamar primeiro `my_init()`! See [\[Funções Threads do C\]](#), page [\[undefined\]](#).

Todas as funções com exceção de `mysql_real_connect()` são seguras com thread por padrão. As anotações seguintes descrevem como compilar uma biblioteca cliente segura com thread e utilizá-la de maneira segura. (As anotações abaixo para `mysql_real_connect()` na verdade se aplicam também a `mysql_connect()`, mas como `mysql_connect()` está obsoleto, você deve utilizar `mysql_real_connect()`.)

Para tornar `mysql_real_connect()` seguro com thread, você deve recompilar a biblioteca cliente com este comando:

```
shell> ./configure --enable-thread-safe-client
```

Isto irá criar uma biblioteca cliente `libmysqlclient_r`. (Assumindo que o seu SO tenha a função `gethostbyname_r()` segura com thread). Esta biblioteca é segura com thread por conexão. Você pode deixar duas threads compartilharem a mesma conexão com os seguintes cuidados:

- Duas threads não podem enviar uma consulta ao servidor MySQL ao mesmo tempo na mesma conexão. Em particular, você deve assegurar que entre um `mysql_query()` e `mysql_store_result()` nenhuma outra thread está usando a mesma conexão.
- Várias threads podem acessar resultados diferentes que são recuperados com `mysql_store_result()`.
- Se você utilizar `mysql_use_result`, você terá que assegurar que nenhuma outra thread está usando a mesma conexão até que o resultado seja fechado. No entanto, é melhor para clientes em threads que compartilham a mesma conexão utilizar `mysql_store_result()`.
- Se você quiser utilizar múltiplas threads na mesma conexão, você deve ter uma trava mutex na combinação das chamadas `mysql_query()` e `mysql_store_result()`. Uma

vez que `mysql_store_result()` esteja pronto, a trva pode ser liberada e outras threads podem utilizar a mesma conexão.

- Se você programa com threads POSIX, você pode utilizar `pthread_mutex_lock()` e `pthread_mutex_unlock()` para estabelecer e liberar uma trava mutex.

Você precisa saber o seguinte se você tiver uma thread que estiver chamando funções MySQL que não criaram a conexão ao banco de dados MySQL:

Quando você chamar `mysql_init()` ou `mysql_connect()`, MySQL irá criar um variável específica da thread para a thread que é utilizada pela biblioteca de depuração (entre outras coisas).

Se você chamar uma função MySQL, antes da thread chamar `mysql_init()` ou `mysql_connect()`, a thread não terá as variáveis específicas de thread necessárias alocadas e você acabará finalizando com uma descarga de memória mais cedo ou mais tarde.

Para fazer que as coisas funcionem suavemente você tem que fazer o seguinte:

1. Chama `my_init()` no início do seu programa se for chamar qualquer outra função MySQL antes de chamar `mysql_real_connect()`.
2. Chame `mysql_thread_init()` no manipulador de thread antes de chamar qualquer outra função MySQL.
3. Na thread, chame `mysql_thread_end()` antes de chamar `pthread_exit()`. Isto irá liberar a memória usada pelas variáveis específicas da thread do MySQL.

Você pode obter alguns erros devido a símbolos indefinidos ao ligar seu cliente com `libmysqlclient_r`. Na maioria dos casos isto ocorre por não estar incluída a biblioteca de threads na linha de ligação/compilação.

8.1.15 libmysqld, a Biblioteca do Servidor Embutido MySQL

8.1.15.1 Visão Geral da Biblioteca do Servidor MySQL Embutido

A biblioteca do servidor MySQL embutido torna possível executar um servidor MySQL com todos os recursos dentro da aplicação cliente. Os principais benefícios são o aumento de velocidade e o gerenciamento mais simples de aplicações embutidas.

A API é idêntica para a versão embutida do MySQL e a versão cliente/servidor. Para alterar uma aplicação em thread antiga para utilizar a biblioteca embutida, você normalmente só precisa adicionar chamadas as seguintes funções:

Função	Quando chamar
<code>mysql_server_init()</code>	Deve ser chamada antes de qualquer outra função MySQL, de preferência no início da função <code>main()</code> .
<code>mysql_server_end()</code>	Deve ser chamada antes da saída do programa.
<code>mysql_thread_init()</code>	Deve ser chamada em cada thread que você criar que acessará o MySQL.
<code>mysql_thread_end()</code>	Deve ser chamada antes de se chamar <code>pthread_exit()</code>

Você deve ligar seu código com `'libmysqld.a'` em vez de `'libmysqlclient.a'`.

As funções acima `mysql_server_XXX` também estão incluídas em `'libmysqlclient.a'` para permitir a troca entre a versão embutida e a cliente/servidor apenas ligando sua aplicação na biblioteca certa. See [\(undefined\)](#) [`mysql_server_init`], page [\(undefined\)](#).

8.1.15.2 Compilando Programas com libmysqld

Para obter uma biblioteca `libmysqld` você deve configurar o MySQL com a opção `--with-embedded-server`.

Quando você liga o seu programa com `libmysqld`, você também deve incluir a biblioteca específica do sistema `pthread` e algumas bibliotecas que o servidor MySQL utiliza. Você pode conseguir a lista completa de bibliotecas executando `mysql_config --libmysqld-libs`.

Os parâmetros corretos para compilar e ligar um programa em thread devem ser usados, mesmo se você não chamar nenhuma função thread diretamente em seu código.

8.1.15.3 Restrições usando um Servidor MySQL Embutido

O servidor embutido tem as seguintes limitações:

- Não tem suporte a tabelas ISAM. (Isto é feito para tornar a biblioteca menor)
- Não possui funçs UDF.
- Não ratreia pilha em caso de descarga de memória.
- Sem suporte a RAID interno. (Normalmente não é necessário já que a maioria dos SO possui suporte a arquivos grandes).
- Você pode configurá-lo como servidor ou master (sem replicação).
- Você não pode conectar ao servidor embutido de um processo externo com sockets ou TCP/IP.

Algumas desta limitações podem ser alteradas editando o arquivo `'mysql_embed.h'` e recompilando o MySQL.

8.1.15.4 Usando Arquivo de Opções com o Servidor Embutido

O descrito abaixo é o modo recomendado de utilizar arquivos de opções para facilitar a troca entre uma aplicação cliente/servidor é uma onde o MySQL está embutido. See [\[Arquivos de opção\], page](#) [\[Arquivos de opção\], page](#).

- Coloque as seções comuns na seção `[server]`. Ela será lida por ambas as versões do MySQL.
- Coloque a opções específicas do cliente/servidor na seção `[mysqld]`.
- Coloque as opções específicas do MySQL embutido na seção `[embedded]`.
- Coloque as opções específicas da aplicação na seção `[ApplicationName_SERVER]`.

8.1.15.5 Itens a Fazer no Servidor Embutido (TODO)

- Estamos fornecendo opções para deixar de fora algumas partes do MySQL para tornar a biblioteca menor.
- Ainda há muita otimização de velocidade a se fazer.
- O erros são escritos no `stderr`. Adicionaremos uma opção para especificar um nome de arquivo para eles.
- Temos que alterar o InnoDB para não ser tão descritivo quando usado em um servidor embutido.

8.1.15.6 Um Exemplo Simples de Seridor Embutido

Este programa e makefile exemplo devem funcionar sem nenhuma alteração em um sistema Linux ou FreeBSD. Para outros sistemas operacionais, pequenas mudanças serão necessárias. Este exemplo é feito para dar detalhes suficientes para entender o problema, sem a desordem que é uma parte necessária de uma aplicação real.

Para experimentar o exemplo, crie um diretório 'test_libmysqld' no mesmo nível que o diretório fonte do mysql-4.0. Salve o fonte 'test_libmysqld.c' e o 'GNUmakefile' no diretório e execute GNU 'make' de dentro do diretório 'test_libmysqld'.

```
'test_libmysqld.c'
/*
 * A simple example client, using the embedded MySQL server library
 */

#include <mysql.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>

MYSQL *db_connect(const char *dbname);
void db_disconnect(MYSQL *db);
void db_do_query(MYSQL *db, const char *query);

const char *server_groups[] = {
    "test_libmysqld_SERVER", "embedded", "server", NULL
};

int
main(int argc, char **argv)
{
    MYSQL *one, *two;

    /* mysql_server_init() devve ser chamado antes de qualquer
     * função mysql.
     *
     * Você pode usar mysql_server_init(0, NULL, NULL), e iniciar
     * o servidor usando os grupos = {
     *   "server", "embedded", NULL
     * }.
     *
     * Em seu arquivo $HOME/.my.cnf file, você provavelmente deseja colocar:

[test_libmysqld_SERVER]
language = /path/to/source/of/mysql/sql/share/english

     * É claro que você poderia modificar argc e argv antes de passá-los
     * a esta função. Ou poderá criar novos do modo que preferir. Mas
     * todos os argumentos em argv (exceto argv[0], que é o nome do
```

```

    * programa) devem ser opções válidas para o servidor MySQL.
    *
    * Se você ligar este cliente em um biblioteca mysqlclient
    * normal, esta função não fará nada.
    */
mysql_server_init(argc, argv, (char **)server_groups);

one = db_connect("test");
two = db_connect(NULL);

db_do_query(one, "SHOW TABLE STATUS");
db_do_query(two, "SHOW DATABASES");

mysql_close(two);
mysql_close(one);

/* Isto deve ser chamado depois de todas outras funções mysql*/
mysql_server_end();

exit(EXIT_SUCCESS);
}

static void
die(MYSQL *db, char *fmt, ...)
{
    va_list ap;
    va_start(ap, fmt);
    vfprintf(stderr, fmt, ap);
    va_end(ap);
    (void)putc('\n', stderr);
    if (db)
        db_disconnect(db);
    exit(EXIT_FAILURE);
}

MYSQL *
db_connect(const char *dbname)
{
    MYSQL *db = mysql_init(NULL);
    if (!db)
        die(db, "mysql_init failed: no memory");
    /*
     * Certifique-se que o cliente e o servidor utilizam grupos diferentes.
     * Isto é crítico pois o servidor não aceitará as opções do
     * cliente e vice versa.
     */
    mysql_options(db, MYSQL_READ_DEFAULT_GROUP, "test_libmysqld_CLIENT");
    if (!mysql_real_connect(db, NULL, NULL, NULL, dbname, 0, NULL, 0))
        die(db, "mysql_real_connect failed: %s", mysql_error(db));
}

```

```

    return db;
}

void
db_disconnect(MYSQL *db)
{
    mysql_close(db);
}

void
db_do_query(MYSQL *db, const char *query)
{
    if (mysql_query(db, query) != 0)
        goto err;

    if (mysql_field_count(db) > 0)
    {
        MYSQL_RES *res;
        MYSQL_ROW row, end_row;
        int num_fields;

        if (!(res = mysql_store_result(db)))
            goto err;
        num_fields = mysql_num_fields(res);
        while ((row = mysql_fetch_row(res)))
        {
            (void)fputs(">> ", stdout);
            for (end_row = row + num_fields; row < end_row; ++row)
                (void)printf("%s\t", row ? (char*)*row : "NULL");
            (void)fputc('\n', stdout);
        }
        (void)fputc('\n', stdout);
    }
    else
        (void)printf("Affected rows: %lld\n", mysql_affected_rows(db));

    mysql_free_result(res);

    return;

err:
    die(db, "db_do_query failed: %s [%s]", mysql_error(db), query);
}

'GNUmakefile'

# This assumes the MySQL software is installed in /usr/local/mysql
inc      := /usr/local/mysql/include/mysql
lib      := /usr/local/mysql/lib

```

```

# If you have not installed the MySQL software yet, try this instead
#inc      := $(HOME)/mysql-4.0/include
#lib      := $(HOME)/mysql-4.0/libmysqld

CC        := gcc
CPPFLAGS  := -I$(inc) -D_THREAD_SAFE -D_REENTRANT
CFLAGS    := -g -W -Wall
LDFLAGS   := -static
# You can change -lmysqld to -mysqlclient to use the
# client/server library
LDLIBS    = -L$(lib) -lmysqld -lz -lm -lcrypt

ifneq (,$(shell grep FreeBSD /COPYRIGHT 2>/dev/null))
# FreeBSD
LDFLAGS += -pthread
else
# Assume Linux
LDLIBS += -lpthread
endif

# This works for simple one-file test programs
sources := $(wildcard *.c)
objects := $(patsubst %c,%o,$(sources))
targets := $(basename $(sources))

all: $(targets)

clean:
rm -f $(targets) $(objects) *.core

```

8.1.15.7 Licenciando o Servidor Embutido

O código fonte do MySQL é coberto pela licença GNU GPL (see [\[Licença GPL\]](#), page [\[licença GPL\]](#)). Um resultado disto é que qualquer programa que inclua, na ligação com `libmysqld`, o código fonte do MySQL deve ser distribuído como software livre. (sob uma licença compatível com a GPL).

Nós encorajamos a todos a promover o software livre distribuindo o código sob a GPL ou uma licença compatível. Para aqueles que não puderem fazê-lo, outra opção é comprar um licença comercial para o código MySQL da MySQL AB. Para maiores detalhes, consulte [\[MySQL licenses\]](#), page [\[MySQL licenses\]](#).

8.2 Suporte ODBC ao MySQL

O MySQL fornece suporte para ODBC através do programa MyODBC. Este capítulo lhe ensinará como instalar o MyODBC, e como usá-lo. Aqui, você também encontrará uma lista de programas comuns que são conhecidos por funcionar com MyODBC.

8.2.1 Como Instalar o MyODBC

MyODBC 2.50 é um driver de nível 0 (com recursos de nível 1 e 2) e especificação de ODBC 2.50 de 32 bits para conectar um programa ODBC ao MySQL. MyODBC funciona nos Windows 9x/Me/NT/2000/XP e na maioria das plataformas Unix. MyODBC 3.51 é uma versão melhorada com nível 1 (core API completo + recursos de nível 2) e especificação de ODBC 3.5x.

MyODBC é Open Source, e você pode encontrar a versão mais nova em <http://www.mysql.com/downloads/api-myodbc.html>. Note que a versão 2.50.x utiliza a licença LGPL, enquanto a versão 3.51.x utiliza a licença GPL.

Se você tiver problemas com o MyODBC e seu programa também funciona com OLEDB, você deve experimentar o driver OLEDB.

Normalmente você só precisa instalar o MyODBC em máquinas Windows. Você só precisará de MyODBC para Unix se tiver um programa como ColdFusion que roda em máquinas Unix e utilizam ODBC para conectar ao banco de dados.

Se você quiser instalar MyODBC em um Unix, você precisará de um gerenciador ODBC. MyODBC funciona com a maioria dos gerenciadores ODBC para Unix.

Para instalar MyODBC no Windows, você deve baixar o arquivo MyODBC ‘.zip’ apropriado, descompactá-lo com WinZIP ou algum programa parecido e executar o arquivo ‘SETUP.EXE’.

No Windows/NT/XP você pode obter o seguinte erro ao tentar instalar o MyODBC:

```
An error occurred while copying C:\WINDOWS\SYSTEM\MFC30.DLL. Restart
Windows and try installing again (before running any applications which
use ODBC)
```

O problema neste caso é que algum outro programa está utilizando ODBC e pela forma que como o Windows é feito, você pode, neste caso, não estar apto a instalar um novo driver ODBC com programa de instalação do ODBC da Microsoft. Neste caso você pode continuar selecionando **Ignore** para copiar o resto dos arquivos ODBC e a instalação final deve funcionar. Se não funcionar, a solução é reinicializar seu computador em “modo seguro” (Escolhendo-o ao pressionar F8 assim que seu computador iniciar o Windows durante a reinicialização), instalar MyODBC, e reiniciar em modo normal.

- Para conectar a uma máquina Unix de uma máquina Windows, com uma aplicação ODBC (uma que não tenha suporte nativo ao MySQL), você deve primeiro instalar MyODBC em uma máquina Windows.
- O usuário em uma máquina Windows deve ter privilégios para acessar o servidor MySQL na máquina Unix. Isto pode ser feito com o comando **GRANT**. See [\[GRANT\]](#), page [\[undefined\]](#).
- Você deve criar uma entrada ODBC DSN como a seguir:
 - Abra o painel de controle na máquina Windows.
 - Dê um duplo clique no ícone Fonte de Dados ODBC 32-bit.
 - Clique na seção Usuário DSN.
 - Clique no botão Adicionar.
 - Selecione MySQL na tela Criar Nova Fonte de Dados e clique no botão Finalizar.
 - A tela de configuração padrão do Driver MySQL é mostrada. See [\[undefined\]](#) [\[Administrador ODBC\]](#), page [\[undefined\]](#).

- Agora inicie a sua aplicação e selecione o driver ODBC com o DSN que você especificou no administrador ODBC.

Verifique se há outra opção de configuração na tela do MySQL (trace, não pergunta ao conectar, etc) que você possa tentar se ocorrerem problemas.

8.2.2 Como Preencher os Vários Campos no Programa de Administração do ODBC

Existem três maneiras possíveis de especificar o nome de servidor no Windows95:

- Use o endereço IP no servidor.
- Adicione um arquivo 'windows\lmhosts' com a seguinte informação:

```
ip nome_maquina
```

Por exemplo:

```
194.216.84.21 meu_nome_maquina
```

- Configure o PC para utilizar DNS.

Exemplo de como preencher a configuração do ODBC.

```
Windows DSN name:  test
Description:       This is my test database
MySQL Database:   test
Server:           194.216.84.21
User:             monty
Password:         my_password
Port:
```

O valor para o campo **Windows DSN name** é qualquer nome que seja único em sua configuração ODBC Windows.

You don't have to specify values for the **Server**, **User**, **Password**, or **Port** fields in the ODBC setup screen. However, if you do, the values will be used as the defaults later when you attempt to make a connection. You have the option of changing the values at that time.

If the port number is not given, the default port (3306) is used.

If you specify the option **Read options from C:\my.cnf**, the groups **client** and **odbc** will be read from the 'C:\my.cnf' file. You can use all options that are usable by `mysql_options()`. See [mysql_options\(\)](#), page [mysql_options\(\)](#).

8.2.3 Connect parameters for MyODBC

One can specify the following parameters for MyODBC on the **[Servername]** section of an 'ODBC.INI' file or through the `InConnectionString` argument in the `SQLDriverConnect()` call.

Parameter	Default value	Comment
user	ODBC (on Windows)	The username used to connect to MySQL.
server	localhost	The hostname of the MySQL server.
database		The default database.

option	0	A integer by which you can specify how MyODBC should work. See below.
port	3306	The TCP/IP port to use if <code>server</code> is not <code>localhost</code> .
stmt		A statement that will be executed when connecting to MySQL.
password		The password for the <code>server user</code> combination.
socket		The socket or Windows pipe to connect to.

The option argument is used to tell MyODBC that the client isn't 100% ODBC compliant. On Windows, one normally sets the option flag by toggling the different options on the connection screen but one can also set this in the option argument. The following options are listed in the same order as they appear in the MyODBC connect screen:

Bit	Description
1	The client can't handle that MyODBC returns the real width of a column.
2	The client can't handle that MySQL returns the true value of affected rows. If this flag is set then MySQL returns 'found rows' instead. One must have MySQL 3.21.14 or newer to get this to work.
4	Make a debug log in <code>c:\myodbc.log</code> . This is the same as putting <code>MYSQL_DEBUG=d:t:0,c::\myodbc.log</code> in 'AUTOEXEC.BAT'
8	Don't set any packet limit for results and parameters.
16	Don't prompt for questions even if driver would like to prompt
32	Simulate a ODBC 1.0 driver in some context.
64	Ignore use of database name in 'database.table.column'.
128	Force use of ODBC manager cursors (experimental).
256	Disable the use of extended fetch (experimental).
512	Pad CHAR fields to full column length.
1024	SQLDescribeCol() will return fully qualified column names
2048	Use the compressed server/client protocol
4096	Tell server to ignore space after function name and before ' (' (needed by PowerBuilder). This will make all function names keywords!
8192	Connect with named pipes to a <code>mysqld</code> server running on NT.
16384	Change LONGLONG columns to INT columns (some applications can't handle LONGLONG).
32768	Return 'user' as Table_qualifier and Table_owner from SQLTables (experimental)
65536	Read parameters from the <code>client</code> and <code>odbc</code> groups from 'my.cnf'
131072	Add some extra safety checks (should not be needed but...)

If you want to have many options, you should add the above flags! For example setting option to 12 (4+8) gives you debugging without package limits!

The default 'MYODBC.DLL' is compiled for optimal performance. If you want to debug MyODBC (for example to enable tracing), you should instead use 'MYODBCD.DLL'. To install this file, copy 'MYODBCD.DLL' over the installed 'MYODBC.DLL' file.

8.2.4 How to Report Problems with MyODBC

MyODBC has been tested with Access, Admndemo.exe, C++-Builder, Borland Builder 4, Centura Team Developer (formerly Gupta SQL/Windows), ColdFusion (on Solaris and NT with svc pack 5), Crystal Reports, DataJunction, Delphi, ERwin, Excel, iHTML, FileMaker Pro,

FoxPro, Notes 4.5/4.6, SBSS, Perl DBD-ODBC, Paradox, Powerbuilder, Powerdesigner 32 bit, VC++, and Visual Basic.

If you know of any other applications that work with MyODBC, please send mail to myodbc@lists.mysql.com about this!

With some programs you may get an error like: `Another user has modifies the record that you have modified`. In most cases this can be solved by doing one of the following things:

- Add a primary key for the table if there isn't one already.
- Add a timestamp column if there isn't one already.
- Only use double float fields. Some programs may fail when they compare single floats.

If the above doesn't help, you should do a MyODBC trace file and try to figure out why things go wrong.

8.2.5 Programs Known to Work with MyODBC

Most programs should work with MyODBC, but for each of those listed here, we have tested it ourselves or received confirmation from some user that it works:

Program	Comment
---------	---------

Access	To make Access work:
--------	----------------------

- If you are using Access 2000, you should get and install the newest (version 2.6 or above) Microsoft MDAC (Microsoft Data Access Components) from <http://www.microsoft.com/data/>. This will fix the following bug in Access: when you export data to MySQL, the table and column names aren't specified. Another way to around this bug is to upgrade to MyODBC Version 2.50.33 and MySQL Version 3.23.x, which together provide a workaround for this bug!

You should also get and apply the Microsoft Jet 4.0 Service Pack 5 (SP5) which can be found here <http://support.microsoft.com/support/kb/articles/Q239/1/14.ASP>. This will fix some cases where columns are marked as `#deleted#` in Access.

Note that if you are using MySQL Version 3.22, you must to apply the MDAC patch and use MyODBC 2.50.32 or 2.50.34 and above to go around this problem.

- For all Access versions, you should enable the MyODBC option flag `Return matching rows`. For Access 2.0, you should additionally enable `Simulate ODBC 1.0`.
- You should have a timestamp in all tables you want to be able to update. For maximum portability `TIMESTAMP(14)` or simple `TIMESTAMP` is recommended instead of other `TIMESTAMP(X)` variations.
- You should have a primary key in the table. If not, new or updated rows may show up as `#DELETED#`.
- Only use `DOUBLE` float fields. Access fails when comparing with single floats. The symptom usually is that new or updated rows may show up as `#DELETED#` or that you can't find or update rows.

- If you are linking a table through MyODBC, which has BIGINT as one of the column, then the results will be displayed as #DELETED. The work around solution is:
 - Have one more dummy column with TIMESTAMP as the data type, preferably TIMESTAMP(14).
 - Check the 'Change BIGINT columns to INT' in connection options dialog in ODBC DSN Administrator
 - Delete the table link from access and re-create it.

It still displays the previous records as #DELETED#, but newly added/updated records will be displayed properly.

- If you still get the error **Another user has changed your data** after adding a TIMESTAMP column, the following trick may help you:
Don't use **table** data sheet view. Create instead a form with the fields you want, and use that **form** data sheet view. You should set the **DefaultValue** property for the TIMESTAMP column to NOW(). It may be a good idea to hide the TIMESTAMP column from view so your users are not confused.
- In some cases, Access may generate illegal SQL queries that MySQL can't understand. You can fix this by selecting "Query|SQLspecific|Pass-Through" from the Access menu.
- Access on NT will report BLOB columns as OLE OBJECTS. If you want to have MEMO columns instead, you should change the column to TEXT with ALTER TABLE.
- Access can't always handle DATE columns properly. If you have a problem with these, change the columns to DATETIME.
- If you have in Access a column defined as BYTE, Access will try to export this as TINYINT instead of TINYINT UNSIGNED. This will give you problems if you have values > 127 in the column!

ADO

When you are coding with the ADO API and MyODBC you need to put attention in some default properties that aren't supported by the MySQL server. For example, using the **CursorLocation** Property as **adUseServer** will return for the **RecordCount** Property a result of -1. To have the right value, you need to set this property to **adUseClient**, like is showing in the VB code here:

```
Dim myconn As New ADODB.Connection
Dim myrs As New Recordset
Dim mySQL As String
Dim myrows As Long

myconn.Open "DSN=MyODBCsample"
mySQL = "SELECT * from user"
myrs.Source = mySQL
Set myrs.ActiveConnection = myconn
myrs.CursorLocation = adUseClient
myrs.Open
myrows = myrs.RecordCount
```

```
myrs.Close
myconn.Close
```

Another workaround is to use a `SELECT COUNT(*)` statement for a similar query to get the correct row count.

Active server pages (ASP)

You should use the option flag `Return matching rows`.

BDE applications

To get these to work, you should set the option flags `Don't optimize column widths` and `Return matching rows`.

Borland Builder 4

When you start a query you can use the property `Active` or use the method `Open`. Note that `Active` will start by automatically issuing a `SELECT * FROM ...` query that may not be a good thing if your tables are big!

ColdFusion (On Unix)

The following information is taken from the ColdFusion documentation:

Use the following information to configure ColdFusion Server for Linux to use the unixODBC driver with MyODBC for MySQL data sources. Allaire has verified that MyODBC Version 2.50.26 works with MySQL Version 3.22.27 and ColdFusion for Linux. (Any newer version should also work.) You can download MyODBC at <http://www.mysql.com/downloads/api-myodbc.html>

ColdFusion Version 4.5.1 allows you to use the ColdFusion Administrator to add the MySQL data source. However, the driver is not included with ColdFusion Version 4.5.1. Before the MySQL driver will appear in the ODBC datasources drop-down list, you must build and copy the MyODBC driver to `'/opt/coldfusion/lib/libmyodbc.so'`.

The Contrib directory contains the program `'mydsn-xxx.zip'` which allows you to build and remove the DSN registry file for the MyODBC driver on Coldfusion applications.

DataJunction

You have to change it to output `VARCHAR` rather than `ENUM`, as it exports the latter in a manner that causes MySQL grief.

Excel

Works. A few tips:

- If you have problems with dates, try to select them as strings using the `CONCAT()` function. For example:

```
select CONCAT(rise_time), CONCAT(set_time)
      from sunrise_sunset;
```

Values retrieved as strings this way should be correctly recognised as time values by Excel97.

The purpose of `CONCAT()` in this example is to fool ODBC into thinking the column is of "string type". Without the `CONCAT()`, ODBC knows the column is of time type, and Excel does not understand that.

Note that this is a bug in Excel, because it automatically converts a string to a time. This would be great if the source was a text file, but is plain stupid when the source is an ODBC connection that reports exact types for each column.

Word

To retrieve data from MySQL to Word/Excel documents, you need to use the MyODBC driver and the Add-in Microsoft Query help.

For example, create a db with a table containing 2 columns of text:

- Insert rows using the `mysql` client command-line tool.
- Create a DSN file using the ODBC manager, for example, 'my' for the db above.
- Open the Word application.
- Create a blank new documentation.
- Using the tool bar called Database, press the button insert database.
- Press the button Get Data.
- At the right hand of the screen Get Data, press the button Ms Query.
- In the Ms Query create a New Data Source using the DSN file my.
- Select the new query.
- Select the columns that you want.
- Make a filter if you want.
- Make a Sort if you want.
- Select Return Data to Microsoft Word.
- Click Finish.
- Click Insert data and select the records.
- Click OK and you see the rows in your Word document.

odbcadmin

Test program for ODBC.

Delphi

You must use BDE Version 3.2 or newer. Set the Don't optimize column width option field when connecting to MySQL.

Also, here is some potentially useful Delphi code that sets up both an ODBC entry and a BDE entry for MyODBC (the BDE entry requires a BDE Alias Editor that is free at a Delphi Super Page near you. (Thanks to Bryan Brunton bryan@flesherfab.com for this):

```
fReg:= TRegistry.Create;
  fReg.OpenKey('\Software\ODBC\ODBC.INI\DocumentsFab', True);
  fReg.WriteString('Database', 'Documents');
  fReg.WriteString('Description', ' ');
  fReg.WriteString('Driver', 'C:\WINNT\System32\myodbc.dll');
  fReg.WriteString('Flag', '1');
  fReg.WriteString('Password', '');
  fReg.WriteString('Port', '');
```

```

fReg.WriteString('Server', 'xmark');
fReg.WriteString('User', 'winuser');
fReg.OpenKey('\Software\ODBC\ODBC.INI\ODBC Data Sources', True);
fReg.WriteString('DocumentsFab', 'MySQL');
fReg.CloseKey;
fReg.Free;

Memo1.Lines.Add('DATABASE NAME=');
Memo1.Lines.Add('USER NAME=');
Memo1.Lines.Add('ODBC DSN=DocumentsFab');
Memo1.Lines.Add('OPEN MODE=READ/WRITE');
Memo1.Lines.Add('BATCH COUNT=200');
Memo1.Lines.Add('LANGDRIVER=');
Memo1.Lines.Add('MAX ROWS=-1');
Memo1.Lines.Add('SCHEMA CACHE DIR=');
Memo1.Lines.Add('SCHEMA CACHE SIZE=8');
Memo1.Lines.Add('SCHEMA CACHE TIME=-1');
Memo1.Lines.Add('SQLPASSTHRU MODE=SHARED AUTOCOMMIT');
Memo1.Lines.Add('SQLQRYMODE=');
Memo1.Lines.Add('ENABLE SCHEMA CACHE=FALSE');
Memo1.Lines.Add('ENABLE BCD=FALSE');
Memo1.Lines.Add('ROWSET SIZE=20');
Memo1.Lines.Add('BLOBS TO CACHE=64');
Memo1.Lines.Add('BLOB SIZE=32');

AliasEditor.Add('DocumentsFab', 'MySQL', Memo1.Lines);

```

C++ Builder

Tested with BDE Version 3.0. The only known problem is that when the table schema changes, query fields are not updated. BDE, however, does not seem to recognise primary keys, only the index PRIMARY, though this has not been a problem.

Vision You should use the option flag `Return matching rows`.

Visual Basic

To be able to update a table, you must define a primary key for the table.

Visual Basic with ADO can't handle big integers. This means that some queries like `SHOW PROCESSLIST` will not work properly. The fix is to set the option `OPTION=16384` in the ODBC connect string or to set the `Change BIGINT columns to INT` option in the MyODBC connect screen. You may also want to set the `Return matching rows` option.

VisualInterDev

If you get the error `[Microsoft] [ODBC Driver Manager] Driver does not support this parameter` the reason may be that you have a `BIGINT` in your result. Try setting the `Change BIGINT columns to INT` option in the MyODBC connect screen.

Visual Objects

You should use the option flag `Don't optimize column widths`.

8.2.6 How to Get the Value of an AUTO_INCREMENT Column in ODBC

A common problem is how to get the value of an automatically generated ID from an INSERT. With ODBC, you can do something like this (assuming that `auto` is an `AUTO_INCREMENT` field):

```
INSERT INTO foo (auto,text) VALUES(NULL,'text');
SELECT LAST_INSERT_ID();
```

Or, if you are just going to insert the ID into another table, you can do this:

```
INSERT INTO foo (auto,text) VALUES(NULL,'text');
INSERT INTO foo2 (id,text) VALUES(LAST_INSERT_ID(),'text');
```

See [\[Getting unique ID\]](#), page [\[Getting unique ID\]](#).

For the benefit of some ODBC applications (at least Delphi and Access), the following query can be used to find a newly inserted row:

```
SELECT * FROM nome_tabela WHERE auto IS NULL;
```

8.2.7 Reporting Problems with MyODBC

If you encounter difficulties with MyODBC, you should start by making a log file from the ODBC manager (the log you get when requesting logs from ODBCADMIN) and a MyODBC log.

To get a MyODBC log, you need to do the following:

1. Ensure that you are using `'myodbcd.dll'` and not `'myodbc.dll'`. The easiest way to do this is to get `'myodbcd.dll'` from the MyODBC distribution and copy it over the `'myodbc.dll'`, which is probably in your `'C:\windows\system32'` or `'C:\winnt\system32'` directory.

Note that you probably want to restore the old `myodbc.dll` file when you have finished testing, as this is a lot faster than `'myodbcd.dll'`.

2. Tag the `'Trace MyODBC'` option flag in the MyODBC connect/configure screen. The log will be written to file `'C:\myodbc.log'`.

If the trace option is not remembered when you are going back to the above screen, it means that you are not using the `myodbcd.dll` driver (see the item above).

3. Start your application and try to get it to fail.

Check the MyODBC trace file, to find out what could be wrong. You should be able to find out the issued queries by searching after the string `>mysql_real_query` in the `'myodbc.log'` file.

You should also try duplicating the queries in the `mysql` monitor or `admdemo` to find out if the error is MyODBC or MySQL.

If you find out something is wrong, please only send the relevant rows (max 40 rows) to myodbc@lists.mysql.com. Please never send the whole MyODBC or ODBC log file!

If you are unable to find out what's wrong, the last option is to make an archive (tar or zip) that contains a MyODBC trace file, the ODBC log file, and a README file that explains the problem. You can send this to <ftp://support.mysql.com/pub/mysql/secret/>. Only we at MySQL AB will have access to the files you upload, and we will be very discrete with the data!

If you can create a program that also shows this problem, please upload this too!

If the program works with some other SQL server, you should make an ODBC log file where you do exactly the same thing in the other SQL server.

Remember that the more information you can supply to us, the more likely it is that we can fix the problem!

8.3 MySQL Java Connectivity (JDBC)

There are 2 supported JDBC drivers for MySQL:

- MySQL Connector/J from MySQL AB, implemented in 100% native Java. This product was formerly known as the `mm.mysql` driver. You can download MySQL Connector/J from <http://www.mysql.com/products/connector-j/>.
- The Resin JDBC driver, which can be found at <http://www.caucho.com/projects/jdbc-mysql/index.xtp>.

For documentation, consult any JDBC documentation, plus each driver's own documentation for MySQL-specific features.

8.4 MySQL PHP API

PHP is a server-side, HTML-embedded scripting language that may be used to create dynamic web pages. It contains support for accessing several databases, including MySQL. PHP may be run as a separate program or compiled as a module for use with the Apache web server.

The distribution and documentation are available at the PHP web site (<http://www.php.net/>).

8.4.1 Common Problems with MySQL and PHP

- Error: "Maximum Execution Time Exceeded" This is a PHP limit; go into the `'php3.ini'` file and set the maximum execution time up from 30 seconds to something higher, as needed. It is also not a bad idea to double the ram allowed per script to 16 MB instead of 8 MB.
- Error: "Fatal error: Call to unsupported or undefined function `mysql_connect()` in .." This means that your PHP version isn't compiled with MySQL support. You can either compile a dynamic MySQL module and load it into PHP or recompile PHP with built-in MySQL support. This is described in detail in the PHP manual.
- Error: "undefined reference to 'uncompress'" This means that the client library is compiled with support for a compressed client/server protocol. The fix is to add `-lz` last when linking with `-lmysqlclient`.

8.5 MySQL Perl API

This section documents the Perl DBI interface. The former interface was called `mysqlperl`. DBI/DBD now is the recommended Perl interface, so `mysqlperl` is obsolete and is not documented here.

8.5.1 DBI with DBD::mysql

DBI is a generic interface for many databases. That means that you can write a script that works with many different database engines without change. You need a DataBase Driver (DBD) defined for each database type. For MySQL, this driver is called DBD::mysql.

For more information on the Perl5 DBI, please visit the DBI web page and read the documentation:

<http://dbi.perl.org/>

For more information on Object Oriented Programming (OOP) as defined in Perl5, see the Perl OOP page:

<http://language.perl.com/info/documentation.html>

Note that if you want to use transactions with Perl, you need to have DBD-mysql version 1.2216 or newer. Version 2.1022 or newer is recommended.

Installation instructions for MySQL Perl support are given in [\[Perl support\]](#), page [\[undefined\]](#).

If you have the MySQL module installed, you can find information about specific MySQL functionality with one of the following command

```
shell> perldoc DBD/mysql
shell> perldoc mysql
```

8.5.2 The DBI Interface

Portable DBI Methods

Method	Description
connect	Establishes a connection to a database server.
disconnect	Disconnects from the database server.
prepare	Prepares a SQL statement for execution.
execute	Executes prepared statements.
do	Prepares and executes a SQL statement.
quote	Quotes string or BLOB values to be inserted.
fetchrow_array	Fetches the next row as an array of fields.
fetchrow_arrayref	Fetches next row as a reference array of fields.
fetchrow_hashref	Fetches next row as a reference to a hashtable.
fetchall_arrayref	Fetches all data as an array of arrays.
finish	Finishes a statement and lets the system free resources.
rows	Returns the number of rows affected.
data_sources	Returns an array of databases available on localhost.
ChopBlanks	Controls whether <code>fetchrow_*</code> methods trim spaces.
NUM_OF_PARAMS	The number of placeholders in the prepared statement.
NULLABLE	Which columns can be NULL.
trace	Perform tracing for debugging.

MySQL-specific Methods

Method	Description
insertid	The latest AUTO_INCREMENT value.

<code>is_blob</code>	Which columns are BLOB values.
<code>is_key</code>	Which columns are keys.
<code>is_num</code>	Which columns are numeric.
<code>is_pri_key</code>	Which columns are primary keys.
<code>is_not_null</code>	Which columns CANNOT be NULL. See NULLABLE.
<code>length</code>	Maximum possible column sizes.
<code>max_length</code>	Maximum column sizes actually present in result.
<code>NAME</code>	Column names.
<code>NUM_OF_FIELDS</code>	Number of fields returned.
<code>table</code>	Table names in returned set.
<code>type</code>	All column types.

The Perl methods are described in more detail in the following sections. Variables used for method return values have these meanings:

<code>\$dbh</code>	Database handle
<code>\$sth</code>	Statement handle
<code>\$rc</code>	Return code (often a status)
<code>\$rv</code>	Return value (often a row count)

Portable DBI Methods

`connect($data_source, $username, $password)`

Use the `connect` method to make a database connection to the data source. The `$data_source` value should begin with `DBI:driver_name::`. Example uses of `connect` with the `DBD::mysql` driver:

```
$dbh = DBI->connect("DBI:mysql:$database", $user, $password);
$dbh = DBI->connect("DBI:mysql:$database:$hostname",
                  $user, $password);
$dbh = DBI->connect("DBI:mysql:$database:$hostname:$port",
                  $user, $password);
```

If the user name and/or password are undefined, DBI uses the values of the `DBI_USER` and `DBI_PASS` environment variables, respectively. If you don't specify a hostname, it defaults to `'localhost'`. If you don't specify a port number, it defaults to the default MySQL port (3306).

As of `Mysql-Mysql-modules` Version 1.2009, the `$data_source` value allows certain modifiers:

`mysql_read_default_file=file_name`

Read `'file_name'` as an option file. For information on option files, see [<undefined> \[Option files\], page <undefined>](#).

`mysql_read_default_group=group_name`

The default group when reading an option file is normally the `[client]` group. By specifying the `mysql_read_default_group` option, the default group becomes the `[group_name]` group.

`mysql_compression=1`

Use compressed communication between the client and server (MySQL Version 3.22.3 or later).

```
mysql_socket=/path/to/socket
```

Specify the pathname of the Unix socket that is used to connect to the server (MySQL Version 3.21.15 or later).

Multiple modifiers may be given; each must be preceded by a semicolon.

For example, if you want to avoid hardcoding the user name and password into a DBI script, you can take them from the user's `~/my.cnf` option file instead by writing your `connect` call like this:

```
$dbh = DBI->connect("DBI:mysql:$database"
    . ";mysql_read_default_file=$ENV{HOME}/.my.cnf",
    $user, $password);
```

This call will read options defined for the `[client]` group in the option file. If you wanted to do the same thing but use options specified for the `[perl]` group as well, you could use this:

```
$dbh = DBI->connect("DBI:mysql:$database"
    . ";mysql_read_default_file=$ENV{HOME}/.my.cnf"
    . ";mysql_read_default_group=perl",
    $user, $password);
```

disconnect

The `disconnect` method disconnects the database handle from the database. This is typically called right before you exit from the program. Example:

```
$rc = $dbh->disconnect;
```

prepare(\$statement)

Prepares a SQL statement for execution by the database engine and returns a statement handle (`$sth`), which you can use to invoke the `execute` method.

Typically you handle `SELECT` statements (and `SELECT`-like statements such as `SHOW`, `DESCRIBE`, and `EXPLAIN`) by means of `prepare` and `execute`. Example:

```
$sth = $dbh->prepare($statement)
    or die "Can't prepare $statement: $dbh->errstr\n";
```

If you want to read big results to your client you can tell Perl to use `mysql_use_result()` with:

```
my $sth = $dbh->prepare($statement { "mysql_use_result" => 1});
```

execute

The `execute` method executes a prepared statement. For non-`SELECT` statements, `execute` returns the number of rows affected. If no rows are affected, `execute` returns `"0E0"`, which Perl treats as zero but regards as true. If an error occurs, `execute` returns `undef`. For `SELECT` statements, `execute` only starts the SQL query in the database; you need to use one of the `fetch_*` methods described here to retrieve the data. Example:

```
$rv = $sth->execute
    or die "can't execute the query: $sth->errstr;
```

do(\$statement)

The `do` method prepares and executes a SQL statement and returns the number of rows affected. If no rows are affected, `do` returns `"0E0"`, which Perl treats as

zero but regards as true. This method is generally used for non-SELECT statements that cannot be prepared in advance (due to driver limitations) or that do not need to be executed more than once (inserts, deletes, etc.). Example:

```
$rv = $dbh->do($statement)
    or die "Can't execute $statement: $dbh->errstr\n";
```

Generally the 'do' statement is much faster (and is preferable) than prepare/execute for statements that don't contain parameters.

quote(\$string)

The quote method is used to "escape" any special characters contained in the string and to add the required outer quotation marks. Example:

```
$sql = $dbh->quote($string)
```

fetchrow_array

This method fetches the next row of data and returns it as an array of field values. Example:

```
while(@row = $sth->fetchrow_array) {
    print qw($row[0]\t$row[1]\t$row[2]\n);
}
```

fetchrow_arrayref

This method fetches the next row of data and returns it as a reference to an array of field values. Example:

```
while($row_ref = $sth->fetchrow_arrayref) {
    print qw($row_ref->[0]\t$row_ref->[1]\t$row_ref->[2]\n);
}
```

fetchrow_hashref

This method fetches a row of data and returns a reference to a hash table containing field name/value pairs. This method is not nearly as efficient as using array references as demonstrated above. Example:

```
while($hash_ref = $sth->fetchrow_hashref) {
    print qw($hash_ref->{firstname}\t$hash_ref->{lastname}\t\
        $hash_ref->{title}\n);
}
```

fetchall_arrayref

This method is used to get all the data (rows) to be returned from the SQL statement. It returns a reference to an array of references to arrays for each row. You access or print the data by using a nested loop. Example:

```
my $table = $sth->fetchall_arrayref
    or die "$sth->errstr\n";

my($i, $j);
for $i ( 0 .. $#{$table} ) {
    for $j ( 0 .. ${$table->[$i]} ) {
        print "$table->[$i][$j]\t";
    }
    print "\n";
}
```

finish Indicates that no more data will be fetched from this statement handle. You call this method to free up the statement handle and any system resources associated with it. Example:

```
$rc = $sth->finish;
```

rows Returns the number of rows changed (updated, deleted, etc.) by the last command. This is usually used after a non-SELECT execute statement. Example:

```
$rv = $sth->rows;
```

NULLABLE Returns a reference to an array of values that indicate whether columns may contain NULL values. The possible values for each array element are 0 or the empty string if the column cannot be NULL, 1 if it can, and 2 if the column's NULL status is unknown. Example:

```
$null_possible = $sth->{NULLABLE};
```

NUM_OF_FIELDS

This attribute indicates the number of fields returned by a SELECT or SHOW FIELDS statement. You may use this for checking whether a statement returned a result: A zero value indicates a non-SELECT statement like INSERT, DELETE, or UPDATE. Example:

```
$nr_of_fields = $sth->{NUM_OF_FIELDS};
```

data_sources(\$driver_name)

This method returns an array containing names of databases available to the MySQL server on the host 'localhost'. Example:

```
@dbs = DBI->data_sources("mysql");
```

ChopBlanks

This attribute determines whether the `fetchrow_*` methods will chop leading and trailing blanks from the returned values. Example:

```
$sth->{'ChopBlanks'} = 1;
```

trace(\$trace_level)

trace(\$trace_level, \$trace_filename)

The `trace` method enables or disables tracing. When invoked as a DBI class method, it affects tracing for all handles. When invoked as a database or statement handle method, it affects tracing for the given handle (and any future children of the handle). Setting `$trace_level` to 2 provides detailed trace information. Setting `$trace_level` to 0 disables tracing. Trace output goes to the standard error output by default. If `$trace_filename` is specified, the file is opened in append mode and output for **all** traced handles is written to that file. Example:

```
DBI->trace(2);                # trace everything
DBI->trace(2, "/tmp/dbi.out"); # trace everything to
                              # /tmp/dbi.out
$dth->trace(2);               # trace this database handle
$sth->trace(2);               # trace this statement handle
```

You can also enable DBI tracing by setting the `DBI_TRACE` environment variable. Setting it to a numeric value is equivalent to calling `DBI->(value)`. Setting it to a pathname is equivalent to calling `DBI->(2, value)`.

MySQL-specific Methods

The methods shown here are MySQL-specific and not part of the DBI standard. Several of them are now deprecated: `is_blob`, `is_key`, `is_num`, `is_pri_key`, `is_not_null`, `length`, `max_length`, and `table`. Where DBI-standard alternatives exist, they are noted here:

`insertid` If you use the `AUTO_INCREMENT` feature of MySQL, the new auto-incremented values will be stored here. Example:

```
$new_id = $sth->{insertid};
```

As an alternative, you can use `$dbh->{'mysql_insertid'}`.

`is_blob` Returns a reference to an array of boolean values; for each element of the array, a value of `TRUE` indicates that the respective column is a BLOB. Example:

```
$keys = $sth->{is_blob};
```

`is_key` Returns a reference to an array of boolean values; for each element of the array, a value of `TRUE` indicates that the respective column is a key. Example:

```
$keys = $sth->{is_key};
```

`is_num` Returns a reference to an array of boolean values; for each element of the array, a value of `TRUE` indicates that the respective column contains numeric values. Example:

```
$nums = $sth->{is_num};
```

`is_pri_key`

Returns a reference to an array of boolean values; for each element of the array, a value of `TRUE` indicates that the respective column is a primary key. Example:

```
$pri_keys = $sth->{is_pri_key};
```

`is_not_null`

Returns a reference to an array of boolean values; for each element of the array, a value of `FALSE` indicates that this column may contain `NULL` values. Example:

```
$not_nulls = $sth->{is_not_null};
```

`is_not_null` is deprecated; it is preferable to use the `NULLABLE` attribute (described above), because that is a DBI standard.

`length`

`max_length`

Each of these methods returns a reference to an array of column sizes. The `length` array indicates the maximum possible sizes that each column may be (as declared in the table description). The `max_length` array indicates the maximum sizes actually present in the result table. Example:

```
$lengths = $sth->{length};
$max_lengths = $sth->{max_length};
```

`NAME` Returns a reference to an array of column names. Example:

```
$names = $sth->{NAME};
```

`table` Returns a reference to an array of table names. Example:

```
$tables = $sth->{table};
```

`type` Returns a reference to an array of column types. Example:

```
$types = $sth->{type};
```

8.5.3 More DBI/DBD Information

You can use the `perldoc` command to get more information about DBI.

```
perldoc DBI
perldoc DBI::FAQ
perldoc DBD::mysql
```

You can also use the `pod2man`, `pod2html`, etc., tools to translate to other formats.

You can find the latest DBI information at the DBI web page: <http://dbi.perl.org/>.

8.6 MySQL C++ API

MySQL Connector/C++ (or MySQL++) is the official MySQL API for C++. More information can be found at <http://www.mysql.com/products/mysql++/>.

8.6.1 Borland C++

You can compile the MySQL Windows source with Borland C++ 5.02. (The Windows source includes only projects for Microsoft VC++, for Borland C++ you have to do the project files yourself.)

One known problem with Borland C++ is that it uses a different structure alignment than VC++. This means that you will run into problems if you try to use the default `libmysql.dll` libraries (that was compiled with VC++) with Borland C++. You can do one of the following to avoid this problem.

- You can use the static MySQL libraries for Borland C++ that you can find on <http://www.mysql.com/downloads/os-win32.html>.
- Only call `mysql_init()` with `NULL` as an argument, not a pre-allocated `MYSQL` struct.

8.7 MySQL Python API

MySQLdb provides MySQL support for Python, compliant with the Python DB API version 2.0. It can be found at <http://sourceforge.net/projects/mysql-python/>.

8.8 MySQL Tcl API

MySQLtcl is a simple API for accessing a MySQL database server from the Tcl programming language. It can be found at <http://www.xdobry.de/mysqltcl/>.

8.9 MySQL Eiffel wrapper

Eiffel MySQL is an interface to the MySQL database server using the Eiffel programming language, written by Michael Ravits. It can be found at <http://efsa.sourceforge.net/archive/ravits/mysql.htm>.

9 Spatial Extensions in MySQL

9.1 Introduction

In release 4.1 MySQL introduces spatial extensions, which allow generating, storing and analysing of geographic features.

9.1.1 Geographic Features

A **geographic feature** is anything in the world that has a location.

A feature can be:

An entity. For example, a mountain, a pond, a city.

A space. For example, a postcode area, the tropics.

A definable location. For example, a crossroad, as a particular place where two streets intersect.

You can also find documents which use term **geospatial feature** to refer to geographic features.

Geometry is another word that denotes a geographic feature. The original meaning of the word **geometry** denotes a branch of mathematics. Another meaning that comes from cartography, referring to the geometric features that cartographers use to map the world.

We will mean the same thing using all these terms, a **geographic feature**, or a **geospatial feature**, or a **feature**, or a **geometry**, with **geometry** as the most used in this documentation.

Let's define a **geometry** as *a point or an aggregate of points representing anything in the world that has a location*.

9.1.2 Approach

MySQL implements spatial extensions following OpenGIS specifications.

The **OpenGIS Consortium** (OGC), is an international consortium of more than 250 companies, agencies, universities participating in the development of publicly available conceptual solutions that can be useful with all kinds of applications that manage spatial data. See <http://www.opengis.org/>.

In 1997, the OpenGIS Consortium published the **OpenGIS (r) Simple Features Specifications For SQL**, which proposes several conceptual ways for extending an SQL RDBMS to support spatial data. MySQL implements a subset of the **SQL with Geometry Types** environment proposed by OGC. This term refers to an SQL environment that has been extended with a set of geometry types. A geometry-valued SQL column is implemented as a column of a geometry type. The specifications describe a set of SQL geometry types, as well as functions on those types to create and analyse geometry values.

9.2 OpenGIS Geometry Model

The set of geometry types, proposed by OGC's **SQL with Geometry Types** environment, is based of **OpenGIS Geometry Model**. In this model, each geometric object:

is associated with a Spatial Reference System, which describes the coordinate space in which the object is defined.

belongs to some geometry class.

9.2.1 Geometry Class Hierarchy

```

Geometry
  Point
  Curve
    LineString
      Line
      LinearRing
  Surface
    Polygon
  GeometryCollection
    MultiPoint
    MultiCurve
      MultiLineString
    MultiSurface
      MultiPolygon
  
```

Geometry is the base class. It's an abstract (non-instantiable) class. The instantiable subclasses of **Geometry** are restricted to zero, one, and two-dimensional geometric objects that exist in two-dimensional coordinate space. All instantiable geometry classes are defined so that valid instances of a geometry class are topologically closed (i.e. all defined geometries include their boundary).

The base **Geometry** class has subclasses for **Point**, **Curve**, **Surface** and **GeometryCollection**.

Curve stands for 1-dimensional objects, and has subclass **LineString**, with sub-subclasses **Line** and **LinearRing**.

Surface is designed for two-dimensional objects and has subclass **Polygon**.

GeometryCollection has specialised 0, 1 and two-dimensional collection classes named **MultiPoint**, **MultiLineString** and **MultiPolygon** for modelling geometries corresponding to collections of **Points**, **LineStrings** and **Polygons** respectively. **MultiCurve** and **MultiSurface** are introduced as abstract superclasses that generalise the collection interfaces to handle **Curves** and **Surfaces**.

Geometry, **Curve**, **Surface**, **MultiCurve** and **MultiSurface** are defined as non-instantiable classes, it is not possible to create an object of these classes. They define a common set of methods for its subclasses and included for the reason of extensibility.

Point, **LineString**, **Polygon**, **GeometryCollection**, **MultiPoint**, **MultiLineString**, **MultiPolygon** are instantiable classes (marked bold in the hierarchy tree).

9.2.2 Class Geometry

Geometry is the root class of the hierarchy. Each geometry is described by a number of its properties. Particular subclasses of the root class Geometry have their own specific properties. Properties, which are common for all geometry subclasses, are described in the list below. Geometry is a non-instantiable class.

9.2.3 Geometry properties

The **type** that a geometry belongs to. Each geometry belongs to one of instantiable classes in the hierarchy.

Its **SRID**, the identifier of a geometry's associated Spatial Reference System which describes the coordinate space in which the geometry object is defined.

Geometry's **coordinates** in its Spatial Reference System, represented as double precision (8 byte) numbers. All non-empty geometries include at least one pair of (X,Y) coordinates. Empty geometries contain no coordinates.

Its **interior, boundary and exterior**. All geometries occupy some position in space. The exterior of a geometry is all space not occupied by the geometry. The interior is the space occupied by the geometry. The boundary is the interface between geometry's interior and exterior.

Its **MBR**, or Envelope, the geometry's Minimum Bounding Rectangle. This is the bounding geometry, formed by the minimum and maximum (X,Y) coordinates:

((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))

The quality of being **simple or non-simple**. Geometry values of some types (LineString, MultyPoint, MultiLineString) are either simple or non-simple. Each type determines its own assertions for being simple or non-simple.

The quality of being **closed or not closed**. Geometry values of some types (LineString, MultiString) are either closed or not closed. Each type determines its own assertions for being closed or not closed.

The quality of being **empty or not empty** A geometry is empty if it does not have any points. Exterior, interior and boundary of an empty geometry are not defined, i.e., they are represented by a NULL value. An empty geometry is defined to be always simple. An empty geometry has an area of 0.

Its **dimension** A geometry can have a dimension of -1, 0, 1 or 2.

-1 stands for empty geometries.

0 stands for geometries with no length and no area.

1 stands for geometries with non-zero length and zero area.

2 stands for geometries with non-zero area.

Points have a dimension of zero. LineStrings have a dimension of 1. Polygons have a dimension of 2. Dimensions of MultiPoints, MultiLineStrings and MultiPolygons are the same as the dimensions of the elements they consist of.

For example, in different coordinate systems, a distance between two objects may differ even when objects have the same coordinates, because the distance on the **planar** coordinate system and the distance on the **geocentric** system (coordinates on the Earth's surface) are different things.

9.2.4 Class Point

A `Point` is a geometry that represents a single location in coordinate space.

Point examples

Imagine a large-scale map of the world with a lot of cities. A point could represent each city.

On a city map, a `Point` could represent a bus stop.

Point properties

X-coordinate value.

Y-coordinate value.

The Boundary of a `Point` is an empty set.

A `Point` is defined as zero-dimensional geometry.

9.2.5 Class Curve

A `Curve` is a one-dimensional geometry, usually represented by a sequence of points. Particular subclasses of `Curve` specify the form of the interpolation between points. `Curve` is a non-instantiable class.

Curve properties

Coordinates of its points.

`Curve` is defined as one-dimensional geometry.

A `Curve` is simple if it does not pass through the same point twice.

A `Curve` is closed if its start point is equal to its end point.

The boundary of a closed `Curve` is empty.

The boundary of a non-closed `Curve` consists of its two end points.

A `Curve` that is simple and closed is `Ring`.

9.2.6 Class LineString

A `LineString` is a `Curve` with linear interpolation between points.

LineString examples

On a world map a `LineStrings` could represent rivers.

In a city map a `LineStrings` could represent streets.

LineString properties

Coordinates of `LineString` segments, defined by each consecutive pair of points.

A `LineString` is a `Line`, if it consists of exactly two points.

A `LineString` is a `LinearRing`, if it's both closed and simple.

9.2.7 Class Surface

A `Surface` is a two-dimensional geometric object.

Surface properties

A `Surface` is defined as a two-dimensional geometry.

The OpenGIS specification defines a simple `Surface` as consisting of a single 'patch' that is associated with one 'exterior boundary' and zero or more 'interior' boundaries.

The boundary of a simple `Surface` is the set of closed curves corresponding to its exterior and interior boundaries.

The only instantiable subclass of `Surface` defined in OpenGIS specification, is `Polygon`.

9.2.8 Class Polygon

A `Polygon` is a planar `Surface` representing a multisided geometry, defined by one exterior boundary and zero or more interior boundaries. Each interior boundary defines a hole in the `Polygon`.

Polygon examples

On a region map: forests, districts, etc.

The assertions for polygons (the rules that define valid polygons) are:

1. The boundary of a `Polygon` consists of a set of `LinearRings` (i.e. `LineStrings` that are both simple and closed) that make up its exterior and interior boundaries.
2. No two rings in the boundary cross, the rings in the boundary of a `Polygon` may intersect at a `Point` but only as a tangent.
3. A `Polygon` may not have cut lines, spikes or punctures.
4. The Interior of every `Polygon` is a connected point set.
5. The Exterior of a `Polygon` with one or more holes is not connected. Each hole defines a connected component of the Exterior.

In the above assertions, polygons are simple geometries.

9.2.9 Class GeometryCollection

A `GeometryCollection` is a geometry that is a collection of one or more geometries of any class.

All the elements in a `GeometryCollection` must be in the same `Spatial Reference` (i.e. in the same coordinate system). `GeometryCollection` places no other constraints on its elements.

Subclasses of `GeometryCollection` described below may restrict membership based on:

Element types

Dimension.

Other constraints on the degree of spatial overlap between elements.

9.2.10 Class MultiPoint

A `MultiPoint` is a collection whose elements are restricted to `Points`. The points are not connected or ordered in any way.

MultiPoint examples

On a world map, a `Multipoint` could represent a chain of small islands.

MultiPoint properties

`MultiPoint` is defined as a zero-dimensional geometry.

A `MultiPoint` is simple if no two `Points` in the `MultiPoint` are equal (have identical coordinate values).

The boundary of a `MultiPoint` is empty set.

9.2.11 Class MultiCurve

A `MultiCurve` is a geometry collection whose elements are `Curves`. `MultiCurve` is a non-instantiable class.

MultiCurve properties

A `MultiCurve` is simple if and only if all of its elements are simple, the only intersections between any two elements occur at points that are on the boundaries of both elements.

The boundary of a `MultiCurve` is obtained by applying the "mod 2 union rule": A point is in the boundary of a `MultiCurve` if it is in the boundaries of an odd number of elements of the `MultiCurve`.

A `MultiCurve` is defined as a one-dimensional geometry.

A `MultiCurve` is closed if all of its elements are closed.

The boundary of a closed `MultiCurve` is always empty.

9.2.12 Class MultiLineString

A `MultiLineString` is a `MultiCurve` whose elements are `LineStrings`.

MultiLineString examples

On a region map, a `MultiLineString` could represent a river system or a highway system.

9.2.13 Class MultiSurface

A `MultiSurface` is a geometric collection whose elements are surfaces. `MultiSurface` is a non-instantiable class.

MultiSurface assertions

1. The interiors of any two surfaces in a MultiSurface may not intersect.
2. The boundaries of any two elements in a MultiSurface may intersect at most at a finite number of points.

The only instantiable subclass of MultiSurface is MultiPolygon.

9.2.14 Class MultiPolygon

A MultiPolygon is a MultiSurface whose elements are Polygons.

MultiPolygon examples

On a region map, a MultiPolygon could represent a system of lakes.

The assertions for MultiPolygons are:

1. The interiors of two Polygons that are elements of a MultiPolygon may not intersect.
2. The Boundaries of any two Polygons that are elements of a MultiPolygon may not cross and may touch at only a finite number of points. (Note that crossing is already forbidden by the first assertion.)
3. A MultiPolygon may not have cut lines, spikes or punctures; a MultiPolygon is a Regular, Closed point set.
4. The interior of a MultiPolygon with more than one Polygon is not connected, the number of connected components of the interior of a MultiPolygon is equal to the number of Polygons in the MultiPolygon.

MultiPolygon properties

MultiPolygon is defined as two-dimensional geometry.

The boundary of a MultiPolygon is a set of closed curves (LineStrings) corresponding to the boundaries of its element Polygons.

Each Curve in the boundary of the MultiPolygon is in the boundary of exactly one element Polygon, and every Curve in the boundary of an element Polygon is in the boundary of the MultiPolygon.

9.3 Supported Spatial Data Formats

This section describes the standard spatial data formats that are used to store geometry objects.

They are:

Well-Known Text (WKT) representation.

Well-Known Binary (WKB) representation.

9.3.1 Well-Known Text (WKT) Representation

The Well-Known Text (WKT) representation of Geometry is designed to exchange geometry data in ASCII form.

9.3.1.1 WKT Examples

Examples of WKT representations of geometry objects are:

POINT(10 10)

A Point.

LINESTRING(10 10, 20 20, 30 40)

A LineString with three points.

POLYGON((10 10, 10 20, 20 20, 20 15, 10 10))

A Polygon with one exterior ring and zero interior rings.

MULTIPOINT(10 10, 20 20)

A MultiPoint with two Points.

MULTILINESTRING((10 10, 20 20), (15 15, 30 15))

A MultiLineString with two LineStrings.

MULTIPOLYGON(((10 10, 10 20, 20 20, 20 15, 10 10)), ((60 60, 70 7, 80 60, 60 60)))

A MultiPolygon with two Polygons.

GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30), LINESTRING(15 15, 20 20))

A GeometryCollection consisting of two Points and one LineString.

9.3.1.2 Bachus-Naur Forms of WKT

The text representation of the implemented instantiable geometric types conforms to this grammar:

The notation {}* denotes zero or more repetitions of the tokens within the braces;

The braces do not appear in the output token list.

```

<Geometry Tagged Text> :=
    <Point Tagged Text>
  | <LineString Tagged Text>
  | <Polygon Tagged Text>
  | <MultiPoint Tagged Text>
  | <MultiLineString Tagged Text>
  | <MultiPolygon Tagged Text>
  | <GeometryCollection Tagged Text>
<Point Tagged Text> := POINT <Point Text>
<LineString Tagged Text> := LINESTRING <LineString Text>
<Polygon Tagged Text> := POLYGON <Polygon Text>
<MultiPoint Tagged Text> := MULTIPOINT <Multipoint Text>
<MultiLineString Tagged Text> := MULTILINESTRING <MultiLineString Text>
<MultiPolygon Tagged Text> := MULTIPOLYGON <MultiPolygon Text>
<GeometryCollection Tagged Text> := GEOMETRYCOLLECTION <GeometryCollection Text>
<Point Text> := EMPTY | ( <Point> )
<Point> := <x> <y>
<x> := double precision literal
<y> := double precision literal
<LineString Text> :=          EMPTY | ( <Point> {, <Point> }* )

```

```

<Polygon Text> :=          EMPTY | ( <LineString Text > {, < LineString Text > } )
<Multipoint Text> :=      EMPTY | ( <Point Text > {, <Point Text > }* )
<MultiLineString Text> := EMPTY | ( <LineString Text > {, < LineString Text > } )
<MultiPolygon Text> :=    EMPTY | ( < Polygon Text > {, < Polygon Text > }* )
<GeometryCollection Text> := EMPTY | ( <Geometry Tagged Text> {, <Geometry Tagged T

```

9.3.2 Well-Known Binary (WKB) Representation

Well-Known Binary (WKB) representation is defined by the OpenGIS specifications. It's also defined in the ISO "SQL/MM Part 3: Spatial" standard.

WKB is used to exchange geometry data as binary streams represented by BLOB values containing geometric information, according to the structures described below.

WKB uses the following basic type definitions:

9.3.2.1 WKB Basic Types

```

// byte   : 8-bit unsigned integer (1 byte)
// uint32  : 32-bit unsigned integer (4 bytes)
// double  : double precision number (8 bytes)
enum wkbGeometryType
{
wkbPoint           = 1,
wkbLineString     = 2,
wkbPolygon        = 3,
wkbMultiPoint     = 4,
wkbMultiLineString = 5,
wkbMultiPolygon   = 6,
wkbGeometryCollection = 7
}
enum wkbByteOrder
{
wkbXDR = 0,    // Big Endian
wkbNDR = 1    // Little Endian
}

```

9.3.2.2 WKB Building Blocks

```

// Building Blocks : Point, LinearRing
Point
{
double x;
double y;
}
LinearRing
{
uint32 numPoints;
Point points[numPoints];
}

```

9.3.2.3 WKB Representation of Geometry Values

```

WKBPoint
{
byte    byteOrder;
uint32  wkbType;    // 1
Point   point;
}
WKBLineString
{
byte    byteOrder;
uint32  wkbType;    // 2
uint32  numPoints;
Point   points[numPoints];
}
WKBPolygon
{
byte    byteOrder;
uint32  wkbType;    // 3
uint32  numRings;
LinearRing rings[numRings];
}
WKBMultiPoint
{
byte    byteOrder;
uint32  wkbType;    // 4
uint32  num_wkbPoints;
WKBPoint  WKBPoints[num_wkbPoints];
}
WKBMultiLineString
{
byte    byteOrder;
uint32  wkbType;    // 5
uint32  num_wkbLineStrings;
WKBLineString WKBLineStrings[num_wkbLineStrings];
}
wkbMultiPolygon
{
byte    byteOrder;
uint32  wkbType;    // 6
uint32  num_wkbPolygons;
WKBPolygon wkbPolygons[num_wkbPolygons];
}
WKBGeometry
{
union
{
WKBPoint point;
WKBLineString linestring;
WKBPolygon polygon;
}
}

```

```

WKBGeometryCollection collection;
WKBMultiPoint mpoint;
WKBMultiLineString mlinestring;
WKBMultiPolygon mpolygon;
}
}
WKBGeometryCollection
{
byte byte_order;
uint32 wkbType; // 7
uint32 num_wkbGeometries;
WKBGeometry wkbGeometries[num_wkbGeometries];
}

```

9.3.2.4 WKB Examples

A WKB which corresponds to POINT(1,1) looks like this sequence of 21 bytes:

```
010100000000000000000000F03F000000000000F03F
```

Where, consequently,

```

Byte order : 01
WKB type   : 01000000
X          : 000000000000F03F
Y          : 000000000000F03F

```

9.4 Creating a Spatially Enabled MySQL Database

9.4.1 MySQL Spatial Data Types

MySQL provides a hierarchy of datatypes, corresponding to the OpenGIS Geometry Model.

```

GEOMETRY
POINT
LINESTRING
POLYGON
MULTIPOINT
MULTILINESTRING
MULTIPOLYGON
GEOMETRYCOLLECTION

```

The GEOMETRY type can store geometries of any type, other types restrict their values to a particular geometry type. GEOMETRYCOLLECTION can store a collection of objects of any type, other collection types restrict the type of collection members to a particular geometry type.

9.4.2 Creating Spatial Values

9.4.2.1 Creating a Geometry Using Its WKT

MySQL provides a number of function which take a Well-Known Text representation and, optionally, a spatial reference system identifier as input parameters, and return the corresponding geometry.

`GeomFromText()` accepts a WKT of any geometry type as its first argument.

For construction of geometry values restricted to a particular type, an implementation also provides a type-specific construction function for each geometry type.

9.4.2.2 Functions To Create a Geometry Using Its WKT

`GeomFromText(wkt,srid)`

`GeometryFromText(wkt,srid)`

Constructs a geometry of any type using its WKT representation and SRID.

`PointFromText(wkt,srid)`

Constructs a POINT using its WKT representation and SRID.

`LineFromText(wkt,srid)`

`LineStringFromText(wkt,srid)`

Constructs a LINESTRING using its WKT representation and SRID.

`PolyFromText(wkt,srid)`

`PolygonFromText(wkt,srid)`

Constructs a POLYGON using its WKT representation and SRID.

`MPointFromText(wkt,srid)`

`MultiPointFromText(wkt,srid)`

Constructs a MULTIPOINT using its WKT representation and SRID.

`MLineFromText(wkt,srid)`

`MultiLineStringFromText(wkt,srid)`

Constructs a MULTILINESTRING using its WKT representation and SRID.

`MPolyFromText(wkt,srid)`

`MultiPolygonFromText(wkt,srid)`

Constructs a MULTIPOLYGON using its WKT representation and SRID.

`GeomCollFromText(wkt,srid)`

Constructs a GEOMETRYCOLLECTION using its WKT representation and SRID.

As an optional feature, an implementation may also support building of Polygon or MultiPolygon values, given an arbitrary collection of possibly intersecting rings or closed LineString values. Implementations that support this feature should include the following functions (**Note:** MySQL does not yet implement these):

`BdPolyFromText(multiLineStringTaggedText String, SRID Integer):Polygon`

Constructs a Polygon given an arbitrary collection of closed linestrings as a MultiLineString text representation.

`BdMPolyFromText(multiLineStringTaggedText String, SRID Integer):MultiPolygon`

Constructs a MultiPolygon given an arbitrary collection of closed Linestrings as a MultiLineString text representation.

9.4.2.3 Creating a Geometry Using Its WKB

MySQL provides a set of functions which take a BLOB containing Well-Known Binary representation and, optionally, a spatial reference system identifier (SRID) as their input parameters, and return the corresponding geometry.

`GeomFromWKB` can accept a WKB of any geometry type as its first argument. For construction of geometry values restricted to a particular type, an implementation also provides a specific construction function for each type of geometry as described in the list above.

9.4.2.4 Functions To Create a Geometry Using Its WKB

`GeomFromWKB(wkt,srid)`

`GeometryFromWKB(wkt,srid)`

Constructs a geometry of any type using its WKB representation and SRID.

`PointFromWKB(wkt,srid)`

Constructs a POINT using its WKB representation and SRID.

`LineFromWKB(wkt,srid)`

`LineStringFromWKB(wkt,srid)`

Constructs a LINESTRING using its WKB representation and SRID.

`PolyFromWKB(wkt,srid)`

`PolygonFromWKB(wkt,srid)`

Constructs a POLYGON using its WKB representation and SRID.

`MPointFromWKB(wkt,srid)`

`MultiPointFromWKB(wkt,srid)`

Constructs a MULTIPOINT using its WKB representation and SRID.

`MLineFromWKB(wkt,srid)`

`MultiLineStringFromWKB(wkt,srid)`

Constructs a MULTILINESTRING using its WKB representation and SRID.

`MPolyFromWKB(wkt,srid)`

`MultiPolygonFromWKB(wkt,srid)`

Constructs a MULTIPOLYGON using its WKB representation and SRID.

`GeomCollFromWKB(wkt,srid)`

Constructs a GEOMETRYCOLLECTION using its WKB representation and SRID.

As an optional feature, an implementation may also support the ‘*Building*’ of Polygon or MultiPolygon values given an arbitrary collection of possibly intersecting rings or closed LineString values. Implementations that support this feature should include the following functions (**Note:** MySQL does not yet implement these):

`BdPolyFromWKB(WKBMultiLineString Binary,SRID Integer): Polygon`

Constructs a Polygon given an arbitrary collection of closed linestrings as a MultiLineString binary representation.

`BdMPolyFromWKB(WKBMultiLineString Binary, SRID Integer):MultiPolygon`

Constructs a MultiPolygon given an arbitrary collection of closed linestrings as a MultiLineString binary representation.

9.4.2.5 Creating a Geometry Value Using MySQL-Specific Functions

Note: the functions listed in this section are not yet implemented in the current version.

MySQL provides a set of useful functions for creating geometry WKB representations. The functions described in this section are MySQL extensions to the OpenGIS specifications. The results of these functions are BLOBs containing geometry WKB representations. The results of these functions can be substituted as first argument for `GeomFromWKB()` function family.

`Point(x,y)`

Constructs a `WKBPoint` using its coordinates.

`MultiPoint(WKBPoint,WKBPoint,...,WKBPoint)`

Constructs a `WKBMultiPoint` using `WKBPoints`. When any argument is not `WKBPoint`, the return value is `NULL`.

`LineString(WKBPoint,WKBPoint,...,WKBPoint)`

Constructs a `WKBLineString` from a number of `WKBPoints`. When any argument is not `WKBPoint`, the return value is `NULL`. When the number of `WKBPoints` is less than two the return value is `NULL`.

`MultiLineString(WKBLineString,WKBLineString,...,WKBLineString)`

Constructs a `WKBMultiLineString` using using `WKBLineStrings`. When any argument is not `WKBLineString`, the return value is `NULL`.

`Polygon(WKBLineString,WKBLineString,...,WKBLineString)`

Constructs a `Polygon` from a number of `WKBLineStrings`. When any argument is not representing WKB of a `LinearRing` (i.e. not closed and simple `LineString`) the return value is `NULL`.

`MultiPolygon(WKBPolygon,WKBPolygon,...,WKBPolygon)`

Constructs a `WKBMultiPolygon` from a set of `WKBPolygons`. When any argument is not a `WKBPolygon`, the return value is `NULL`.

`GeometryCollection(WKBGeometry,WKBGeometry,...,WKBGeometry)`

Constructs a `GeometryCollection`. When any argument is not a well-formed WKB representation of a geometry, the return value is `NULL`.

9.4.3 Creating Spatial Columns

MySQL provides a standard way of creating spatial columns for geometry types.

CREATE TABLE

Use the `CREATE TABLE` statement to create a table with a spatial column:

```
mysql> CREATE TABLE g1 (p1 GEOMETRY);
Query OK, 0 rows affected (0.02 sec)
mysql>
```

ALTER TABLE

Use the `ALTER TABLE` statement to add a spatial column to an existing table:

```
mysql> ALTER TABLE g1 ADD p2 POINT;
Query OK, 0 rows affected (0.00 sec)
```


9.4.5.2 Fetching Spatial Data Using WKB Representation

The `AsBinary()` function provides binary access to geometry values by converting them into a BLOB containing WKB.

```
SELECT AsBinary(g) FROM geom;
```

`AsBinary()` returns a BLOB with a geometry in its WKB representation.

9.5 Analysing Spatial Information

After populating spatial columns with values, you are ready to query and analyse them. Spatial analysis can be performed using spatial functions in:

Any interactive SQL program, like `mysql` or `MySQLCC`.

Application programs in any languages supporting a MySQL client API.

MySQL provides a set of functions to perform various operations on spatial data. These functions can be labeled into four major groups according to the type of operation they perform:

Functions that convert geometries between various formats.

Functions that describe qualitative or quantitative properties of a geometry.

Functions that describe relations between two geometries.

Functions that create new geometries from existing ones.

9.5.1 Functions To Convert Geometries Between Different Formats

As discussed (see [\[GIS MySQL spatial data types\]](#), page [\[undefined\]](#), see [\[undefined\]](#) [\[GIS populating spatial columns\]](#), page [\[undefined\]](#)), MySQL understands Well-Known Text (WKT) and Well-Known Binary (WKB) geometry representations through support of these functions:

`GeomFromWKT(string wkt [,integer srid]): geometry`

Converts WKT representation into internal geometry format. A number of type-specific functions are also supported.

`GeomFromWKB(binary wkb [,integer srid]): geometry`

Converts WKB representation into internal geometry format A number of type-specific functions are also supported.

`AsWKT(geometry g): string`

Converts internal geometry format into WKT representation.

`AsWKB(geometry g): binary`

Converts internal geometry format into WKB representation.

```
mysql> SELECT AsText(GeomFromText('LineString(1 1,2 2,3 3)'));
+-----+
| AsText(GeomFromText('LineString(1 1,2 2,3 3)')) |
+-----+
| LINESTRING(1 1,2 2,3 3) |
+-----+
```

9.5.2 Functions To Analyse Geometry Properties

Functions that belong to this group take a geometry value as their argument and return some quantitative or qualitative property of this geometry. Some functions restrict their argument type.

9.5.2.1 Basic Functions To Analyse Geometry Properties

These functions don't restrict their argument and accept a geometry of any type.

`GeometryType(geometry g):string`

Returns as string the name of the geometry subtype of which this geometry instance is a member.

```
mysql> SELECT GeometryType(GeomFromText('POINT(1 1)'));
+-----+
| GeometryType(GeomFromText('POINT(1 1)')) |
+-----+
| POINT                                     |
+-----+
```

`Dimension(geometry g):integer`

The inherent dimension of this Geometry object, which can be -1, 0, 1 or 2.

```
mysql> SELECT Dimension(GeomFromText('LineString(1 1,2 2)'));
+-----+
| Dimension(GeomFromText('LineString(1 1,2 2)')) |
+-----+
|                                               1 |
+-----+
```

`SRID(geometry g):integer`

Returns the Spatial Reference System ID for this geometry.

```
mysql> SELECT SRID(GeomFromText('LineString(1 1,2 2)',101));
+-----+
| SRID(GeomFromText('LineString(1 1,2 2)',101)) |
+-----+
|                                               101 |
+-----+
```

`Envelope(geometry g):geometry`

The Minimum Bounding Rectangle (MBR) for this geometry, returned as a polygon. The polygon is defined by the corner points of the bounding box:

```
POLYGON((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
mysql> SELECT AsText(Envelope(GeomFromText('LineString(1 1,2 2)',101)));
+-----+
| AsText(Envelope(GeomFromText('LineString(1 1,2 2)',101))) |
+-----+
| POLYGON((1 1,2 1,2 2,1 2,1 1))                               |
+-----+
```

Note: MySQL does not yet implement the following functions:

`Boundary(g:Geometry):Geometry`

Returns the closure of the combinatorial boundary of this Geometry.

`IsEmpty(geometry g):Integer`

Returns 1 (TRUE) if this Geometry is the empty geometry. If true, then this Geometry represents the empty point set.

`IsSimple(geometry g):Integer`

Returns 1 (TRUE) if this Geometry has no anomalous geometric points, such as self intersection or self tangency. The description of each instantiable geometric class includes the specific conditions that cause an instance of that class to be classified as not simple.

9.5.2.2 Functions To Analyse Point Properties

`X(point p):Double`

The x-coordinate value for this point.

```
mysql> SELECT X(GeomFromText('Point(56.7 53.34)',101));
+-----+
| X(GeomFromText('Point(56.7 53.34)',101)) |
+-----+
|                                     56.7 |
+-----+
```

`Y(point p):Double`

The y-coordinate value for this point

```
mysql> SELECT Y(GeomFromText('Point(56.7 53.34)',101));
+-----+
| Y(GeomFromText('Point(56.7 53.34)',101)) |
+-----+
|                                     53.34 |
+-----+
```

9.5.2.3 Functions To Analyse LineString Properties

`StartPoint(LineString l):Point`

The start point of this LineString.

```
mysql> SELECT AsText(StartPoint(GeomFromText('LineString(1 1,2 2,3 3)')))
+-----+
--+
| AsText(StartPoint(GeomFromText('LineString(1 1,2 2,3 3)'))) |
+-----+
--+
| POINT(1 1) |
+-----+
--+
```

`EndPoint(LineString l):Point`

The end point of this LineString.

```
mysql> SELECT AsText(EndPoint(GeomFromText('LineString(1 1,2 2,3 3)')));
+-----+
| AsText(EndPoint(GeomFromText('LineString(1 1,2 2,3 3)'))) |
+-----+
| POINT(3 3) |
+-----+
+
```

PointN(LineString l, integer n):Point

Returns the specified point N in this LineString.

```
mysql> SELECT AsText(PointN(GeomFromText('LineString(1 1,2 2,3 3)'),2));
+-----+
| AsText(PointN(GeomFromText('LineString(1 1,2 2,3 3)'),2)) |
+-----+
| POINT(2 2) |
+-----+
+
```

GLength(LineString l):Double

The length of this LineString in its associated spatial reference.

```
mysql> SELECT GLength(GeomFromText('LineString(1 1,2 2,3 3)'));
+-----+
| GLength(GeomFromText('LineString(1 1,2 2,3 3)')) |
+-----+
| 2.8284271247462 |
+-----+
+
```

NumPoints(LineString l):Integer

The number of points in this LineString.

```
mysql> SELECT NumPoints(GeomFromText('LineString(1 1,2 2,3 3)'));
+-----+
| NumPoints(GeomFromText('LineString(1 1,2 2,3 3)')) |
+-----+
| 3 |
+-----+
+
```

Note: MySQL does not yet implement the following functions:

IsRing(LineString l):Integer

Returns 1 (TRUE) if this LineString is closed (StartPoint () = EndPoint ()) and this LineString is simple (does not pass through the same point more than once).

IsClosed(LineString l):Integer

Returns 1 (TRUE) if this LineString is closed (StartPoint() == EndPoint()).

```
mysql> SELECT IsClosed(GeomFromText('LineString(1 1,2 2,3 3)'));
+-----+
| IsClosed(GeomFromText('LineString(1 1,2 2,3 3)')) |
+-----+
| 0 |
+-----+
+
```



```

+-----+
-----+
|
+-----+
-----+
1 row in set (0.00 sec)

```

ExteriorRing(Polygon p):LineString

Returns the exterior ring of this Polygon as a LineString.

```

mysql> SELECT AsText(ExteriorRing(GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0)
+-----+
-----+
| AsText(ExteriorRing(GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1
+-----+
-----+
| LINESTRING(0 0,0 3,3 3,3 0,0 0)
+-----+
-----+

```

InteriorRingN(Polygon p, Integer N):LineString

Returns the N-th interior ring for this Polygon as a LineString.

```

mysql> SELECT AsText(InteriorRingN(GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0)
+-----+
-----+
| AsText(InteriorRingN(GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1
+-----+
-----+
| LINESTRING(1 1,1 2,2 2,2 1,1 1)
+-----+
-----+

```

Note: MySQL does not yet implement the following functions:

Centroid(Polygon p):Point

The mathematical centroid for this Polygon as a Point. The result is not guaranteed to be on this Polygon.

PointOnSurface(p:Polygon):Point

A point guaranteed to be on this Polygon.

9.5.2.6 Functions To Analyse MultiPolygon Properties

Area(MultiPolygon m):Double

The area of this MultiPolygon, as measured in the spatial reference system of this MultiPolygon.

```

mysql> SELECT Area(GeomFromText('MultiPolygon(((0 0,0 3,3 3,3 0,0 0),(1 1
+-----+
-----+
| Area(GeomFromText('MultiPolygon(((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1
+-----+
-----+

```

```

|
+-----+
-----+

```

Note: MySQL does not yet implement the following functions:

`Centroid(MultiPolygon p):Point`

The mathematical centroid for this MultiPolygon as a Point. The result is not guaranteed to be on this MultiPolygon.

`PointOnSurface(MultiPolygon m):Point`

A Point guaranteed to be on this MultiPolygon.

9.5.2.7 Functions To Analyse GeometryCollection Properties

`NumGeometries(GeometryCollection g):Integer`

Returns the number of geometries in this GeometryCollection.

```

mysql> SELECT NumGeometries(GeomFromText('GeometryCollection(Point(1 1),L
+-----+
-----+
| NumGeometries(GeomFromText('GeometryCollection(Point(1 1),LineString(2
+-----+
-----+
|
+-----+
-----+

```

`GeometryN(GeometryCollection g,integer N):Geometry`

Returns the N-th geometry in this GeometryCollection.

```

mysql> SELECT AsText(GeometryN(GeomFromText('GeometryCollection(Point(1 1
+-----+
-----+
| AsText(GeometryN(GeomFromText('GeometryCollection(Point(1 1),LineString
+-----+
-----+
| POINT(1 1)
+-----+
-----+

```

Note: Functions for specific geometry type return NULL if the passed geometry is of wrong geometry type. For example `Area()` returns NULL if object type is neither Polygon nor MultiPolygon.

9.5.3 Functions That Create New Geometries From Existing Ones

9.5.3.1 Geometry Properties That Produce New Geometries

In the section (see [\[GIS functions to analyse geometry properties\]](#), page [\[undefined\]](#)), we've already discussed some functions that can construct new geometries from the existing ones:

```

Envelope(geometry g):geometry
StartPoint(LineString l):Point
EndPoint(LineString l):Point
PointN(LineString l,integer n):Point
ExteriorRing(Polygon p):LineString
InteriorRingN(Polygon p, Integer N):LineString
GeometryN(GeometryCollection g,integer N):Geometry

```

9.5.3.2 Spatial Operators

OpenGIS proposes a number of other functions that can produce geometries. They are designed to implement Spatial Operators.

Note: These functions are not yet implemented. They should appear in the future releases.

```
Intersection(Geometry g1,g2):Geometry
```

A geometry that represents the point set intersection of g1 with g2.

```
Union(Geometry g1,g2):Geometry
```

A geometry that represents the point set union of g1 with g2.

```
Difference(Geometry g1,g2):Geometry
```

A geometry that represents the point set difference of g1 with g2.

```
SymDifference(Geometry g1,g2):Geometry
```

A geometry that represents the point set symmetric difference of g1 with g2.

```
Buffer(Geometry g, double d):Geometry
```

A geometry that represents all points whose distance from g is less than or equal to distance of d.

```
ConvexHull(Geometry g):Geometry
```

A geometry that represents the convex hull of g.

9.5.4 Functions For Testing Spatial Relations Between Geometric Objects

The functions described in this sections take two geometries as input parameters and return a qualitative or quantitative relation between them.

9.5.5 Relations On Geometry Minimal Bounding Rectangles (MBR)

The current release provides some functions that can test relations between minimal bounding rectangles of two geometries. They include:

```
MBRContains(geom1,geom2)
```

Returns 1 if the Minimum Bounding Rectangle of geom1 contains the Minimum Bounding Rectangle of geom2. Otherwise, 0 is returned.

```
mysql> SELECT MBRContains(GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))'),
+-----+
+-----+
| MBRContains(GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))'),GeomFromText
+-----+
+-----+
|
+-----+
+-----+
```

MBRWithin(geom1,geom2)

Returns 1 if the Minimum Bounding Rectangle of **geom1** is within the Minimum Bounding Rectangle of **geom2**. Otherwise, 0 is returned.

```
mysql> SELECT MBRWithin(GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))'),Ge
+-----+
+-----+
| MBRWithin(GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))'),GeomFromText('
+-----+
+-----+
|
+-----+
+-----+
```

MBRDisjoint(geom1,geom2)

Returns 1 if the Minimum Bounding Rectangles of the two geometries do not intersect. Otherwise, 0 is returned.

MBREqual(geom1,geom2)

Returns 1 if the Minimum Bounding Rectangles of the two geometries are the same. Otherwise, 0 is returned.

MBRIntersects(geom1,geom2)

Returns 1 if the Minimum Bounding Rectangles of the two geometries intersect. Otherwise, 0 is returned.

MBROverlaps(geom1,geom2)

Returns 1 if the Minimum Bounding Rectangles of the two geometries overlaps. Otherwise, 0 is returned.

MBRTouches(geom1,geom2)

Returns 1 if the Minimum Bounding Rectangles of the two geometries overlaps. Otherwise, 0 is returned.

9.5.6 Functions That Test Spatial Relationships Between Geometries

Note: The functions given in the list below are not yet implemented. These functions will provide full (not MBR-based only) support for spatial analysis.

Contains(geom1,geom2)

Returns 1 if **geom1** completely contains **geom2**, otherwise 0 is returned.

Crosses(geom1,geom2)

Returns 1 if **geom1** spatially crosses **geom2**. If **geom1** is a polygon or a multipolygon, NULL is returned. If **geom2** is a point or a multipoint, NULL is returned. Otherwise 0 is returned.

The term **spatially crosses** denotes a spatial relation when two given geometries intersect, and their intersection results in a geometry that has a dimension that is one less than the maximum dimension of the two given geometries, and the intersection is not equal to any of the two given geometries.

Disjoint(geom1,geom2)

Returns 1 if **geom1** is spatially disjoint from **geom2**, i.e. if given geometries do not intersect. Otherwise, 0 is returned.

Equal(geom1,geom2)

Returns 1 if **geom1** is spatially equal to **geom2**, otherwise 0 is returned.

Intersects(geom1,geom2)

Returns 1 if **geom1** spatially intersects **geom2**, otherwise 0 is returned.

Overlaps(geom1,geom2)

Returns 1 if **geom1** spatially overlaps **geom2**, otherwise, 0 is returned. Term **spatially overlaps** is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

Touches(geom1,geom2)

Returns 1 if **geom1** spatially touches **geom2**, otherwise, 0 is returned. Two geometries spatially touch if the interior of both geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the geometries.

Within(geom1,geom2)

Returns 1 if **geom1** is spatially within **geom2**, otherwise, 0 is returned.

Distance(geom1:Geometry,geom2:Geometry):Double

The shortest distance between any two points in the two geometries.

9.6 Optimising Spatial Analysis

It is known that search operations in usual databases can be optimised using indexes. This is still true for spatial databases. With help of great variety of multi-dimensional indexing methods which have already been designed in the world, it's possible to optimise spatial searches, the most typical of which are:

A point query. Searching for all objects that contain a given point.

A region query. Searching for all objects that overlap a given region.

MySQL utilises **R-Trees with quadratic splitting** to index spatial columns. A spatial index is built using the MBR of a geometry. For most geometries, the MBR is a minimum rectangle that surrounds the geometries. For a horizontal or a vertical linestring as well as for a point, the MBR is a degenerated rectangle, into the linestring and the point respectively.

9.6.1 Creating Spatial Indexes

MySQL can create spatial indexes in the same way it can create regular indexes. The normal syntax for creating indexes is extended with the `SPATIAL` keyword:

With `CREATE TABLE`:

```
CREATE TABLE g (g GEOMETRY NOT NULL, SPATIAL INDEX(g));
```

With `CREATE INDEX`:

```
CREATE SPATIAL INDEX sp_index ON g (g);
```

With `ALTER TABLE`:

```
ALTER TABLE g (ADD SPATIAL KEY(g));
```

Let's say we have a database with more than 32000 geometries. Geometries are stored in the field `g` of type `GEOMETRY`. The table also has a field `fid`, storing object IDs, with the `AUTO_INCREMENT` attribute.

```
mysql> SHOW FIELDS FROM g;
```

```
+-----+-----+-----+-----+-----+-----+-----+
+
| Field | Type      | Collation | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+-----+
+
| fid   | int(11)   | binary    |      | PRI | NULL     | auto_increment |
| g     | geometry  | binary    |      |     |          |                |
+-----+-----+-----+-----+-----+-----+-----+
+
2 rows in set (0.00 sec)
```

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

```
+-----+
| count(*) |
+-----+
|    32376 |
+-----+
1 row in set (0.00 sec)
```

Let's add a spatial index:

```
mysql> ALTER TABLE g ADD SPATIAL KEY(g);
Query OK, 32376 rows affected (4.05 sec)
Records: 32376 Duplicates: 0 Warnings: 0
```

9.6.2 Using a Spatial Index

The optimiser investigates if available spatial indexes can be involved in the search whenever a query with a function like `MBRContains()` or `MBRWithin()` in the `WHERE` clause is executed.

For example, let's say we want to find all objects that are in the given rectangle:

```
mysql> SELECT fid,AsText(g) FROM g WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000 16000,30000
+-----+-----+
-----+
```

```

| fid | AsText(g)
+-----+-----+
-----+
| 21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30333.8 15828.8)
| 22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8,30334 15871.4)
| 23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4,30334 15914.2)
| 24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4,30273.4 15823)
| 25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882.4,30274.8 15866.2)
| 26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4,30275 15918.2)
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946.8,30320.4 15938.4)
| 1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136.4,30240 15127.2)
| 2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136,30210.4 15121)
| 3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,30169 15113)
| 4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30157 15111.6)
| 5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4,30194.2 15075.2)
| 6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,30244.6 15077)
| 7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8,30201.2 15049.4)
| 10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6,30189.6 15019)
| 11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2,30151.2 15009.8)
| 13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,30114.6 15067.8)
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30278 15134)
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30259 15083.4)
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4,30128.8 15001)
+-----+-----+
-----+
20 rows in set (0.00 sec)

```

Now let's check the way this query is executed, using EXPLAIN:

```

mysql> EXPLAIN SELECT fid,AsText(g) FROM g WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000 16000,30000
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | g | range | g | g | 32 | NULL | 50 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+
1 row in set (0.00 sec)

```

Now let's check what would happen if we didn't have a spatial index:

```

mysql> EXPLAIN SELECT fid,AsText(g) FROM g IGNORE INDEX (g) WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000 16000,30000
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | g | ALL | NULL | NULL | NULL | NULL | 32376 |

```

```

+-----+-----+-----+-----+-----+-----+-----+-----+
---+-----+-----+
1 row in set (0.00 sec)

```

Lets execute the above query, ignoring the spatial key we have:

```

mysql> SELECT fid,AsText(g) FROM g IGNORE INDEX (g) WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000 16000,30000
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+
| fid | AsText(g)
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+
| 1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136.4,30240 15127.2)
| 2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136,30210.4 15121)
| 3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,30169 15113)
| 4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30157 15111.6)
| 5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4,30194.2 15075.2)
| 6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,30244.6 15077)
| 7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8,30201.2 15049.4)
| 10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6,30189.6 15019)
| 11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2,30151.2 15009.8)
| 13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,30114.6 15067.8)
| 21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30333.8 15828.8)
| 22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8,30334 15871.4)
| 23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4,30334 15914.2)
| 24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4,30273.4 15823)
| 25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882.4,30274.8 15866.2)
| 26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4,30275 15918.2)
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30278 15134)
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30259 15083.4)
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4,30128.8 15001)
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946.8,30320.4 15938.4)
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+
20 rows in set (0.46 sec)

```

The execution time for this query rises from 0.00 seconds to 0.46 seconds, when the index is not used.

In the future releases, spatial indexes will also be used for optimising other functions. See [\[GIS functions for testing spatial relations between geometric objects\]](#), page [\[undefined\]](#).

9.7 MySQL Conformance And Compatibility

9.7.1 GIS Features That Are Not Yet Implemented

Additional Metadata Views

OpenGIS specifications propose several additional metadata views. For example, a system view named `GEOMETRY_COLUMNS` contains a description of geometry columns, one row for each geometry column in the database.

Functions to add/drop spatial columns

OpenGIS assumes that columns can be added/dropped using `AddGeometryColumn()` and `DropGeometryColumn()` functions correspondently. In MySQL this is to be done using the regular `ALTER TABLE` command instead.

Spatial Reference Systems and their IDs (SRIDs) related things:

Functions like `Length()` and `Area()` assume planar coordinate system.

All objects are currently considered to be in the same planar coordinate system.

Function `Length()` on `LineString` and `MultiLineString` currently should be called as `GLength()`.

The problem is that it conflicts with a usual function `Length()` and sometimes it's not possible to distinguish if the function was called in the textual or spatial context. We need either to solve this somehow, or decide on another function name.

10 Extending MySQL

10.1 MySQL Internals

This chapter describes a lot of things that you need to know when working on the MySQL code. If you plan to contribute to MySQL development, want to have access to the bleeding-edge in-between versions code, or just want to keep track of development, follow the instructions in [\[Installing source tree\]](#), page [\[undefined\]](#). If you are interested in MySQL internals, you should also subscribe to our `internals` mailing list. This list is relatively low traffic. For details on how to subscribe, please see [list-snt \[Mailing-list\]](#), page [list-pg](#). All developers at MySQL AB are on the `internals` list and we help other people who are working on the MySQL code. Feel free to use this list both to ask questions about the code and to send patches that you would like to contribute to the MySQL project!

10.1.1 MySQL Threads

The MySQL server creates the following threads:

- The TCP/IP connection thread handles all connection requests and creates a new dedicated thread to handle the authentication and SQL query processing for each connection.
- On Windows NT there is a named pipe handler thread that does the same work as the TCP/IP connection thread on named pipe connect requests.
- The signal thread handles all signals. This thread also normally handles alarms and calls `process_alarm()` to force timeouts on connections that have been idle too long.
- If `mysqld` is compiled with `-DUSE_ALARM_THREAD`, a dedicated thread that handles alarms is created. This is only used on some systems where there are problems with `sigwait()` or if one wants to use the `thr_alarm()` code in ones application without a dedicated signal handling thread.
- If one uses the `--flush_time=#` option, a dedicated thread is created to flush all tables at the given interval.
- Every connection has its own thread.
- Every different table on which one uses `INSERT DELAYED` gets its own thread.
- If you use `--master-host`, a slave replication thread will be started to read and apply updates from the master.

`mysqladmin processlist` only shows the connection, `INSERT DELAYED`, and replication threads.

10.1.2 MySQL Test Suite

Until recently, our main full-coverage test suite was based on proprietary customer data and for that reason has not been publicly available. The only publicly available part of our testing process consisted of the `crash-me` test, a Perl DBI/DBD benchmark found in the `sql-bench` directory, and miscellaneous tests located in `tests` directory. The lack of a standardised publicly available test suite has made it difficult for our users, as well

developers, to do regression tests on the MySQL code. To address this problem, we have created a new test system that is included in the source and binary distributions starting in Version 3.23.29.

The current set of test cases doesn't test everything in MySQL, but it should catch most obvious bugs in the SQL processing code, OS/library issues, and is quite thorough in testing replication. Our eventual goal is to have the tests cover 100% of the code. We welcome contributions to our test suite. You may especially want to contribute tests that examine the functionality critical to your system, as this will ensure that all future MySQL releases will work well with your applications.

10.1.2.1 Running the MySQL Test Suite

The test system consist of a test language interpreter (`mysqltest`), a shell script to run all tests(`mysql-test-run`), the actual test cases written in a special test language, and their expected results. To run the test suite on your system after a build, type `make test` or `mysql-test/mysql-test-run` from the source root. If you have installed a binary distribution, `cd` to the install root (eg. `/usr/local/mysql`), and do `scripts/mysql-test-run`. All tests should succeed. If not, you should try to find out why and report the problem if this is a bug in MySQL. See [\[Reporting mysqltest bugs\]](#), page [\[undefined\]](#).

If you have a copy of `mysqld` running on the machine where you want to run the test suite you do not have to stop it, as long as it is not using ports 9306 and 9307. If one of those ports is taken, you should edit `mysql-test-run` and change the values of the master and/or slave port to one that is available.

You can run one individual test case with `mysql-test/mysql-test-run test_name`.

If one test fails, you should test running `mysql-test-run` with the `--force` option to check if any other tests fails.

10.1.2.2 Extending the MySQL Test Suite

You can use the `mysqltest` language to write your own test cases. Unfortunately, we have not yet written full documentation for it. You can, however, look at our current test cases and use them as an example. The following points should help you get started:

- The tests are located in `mysql-test/t/*.test`
- A test case consists of `;` terminated statements and is similar to the input of `mysql` command-line client. A statement by default is a query to be sent to MySQL server, unless it is recognised as internal command (eg. `sleep`).
- All queries that produce resultse.g., `SELECT`, `SHOW`, `EXPLAIN`, etc., must be preceded with `@/path/to/result/file`. The file must contain the expected results. An easy way to generate the result file is to run `mysqltest -r < t/test-case-name.test` from `mysql-test` directory, and then edit the generated result files, if needed, to adjust them to the expected output. In that case, be very careful about not adding or deleting any invisible characters - make sure to only change the text and/or delete lines. If you have to insert a line, make sure the fields are separated with a hard tab, and there is a hard tab at the end. You may want to use `od -c` to make sure your text editor has not messed anything up during edit. We, of course, hope that you will never have to edit the output of `mysqltest -r` as you only have to do it when you find a bug.

- To be consistent with our setup, you should put your result files in `mysql-test/r` directory and name them `test_name.result`. If the test produces more than one result, you should use `test_name.a.result`, `test_name.b.result`, etc.
- If a statement returns an error, you should on the line before the statement specify with the `--error error-number`. The error number can be a list of possible error numbers separated with `' , '`.
- If you are writing a replication test case, you should on the first line of the test file, put `source include/master-slave.inc;`. To switch between master and slave, use `connection master;` and `connection slave;`. If you need to do something on an alternate connection, you can do `connection master1;` for the master, and `connection slave1;` for the slave.
- If you need to do something in a loop, you can use something like this:


```
let $1=1000;
while ($1)
{
    # do your queries here
    dec $1;
}
```
- To sleep between queries, use the `sleep` command. It supports fractions of a second, so you can do `sleep 1.3;`, for example, to sleep 1.3 seconds.
- To run the slave with additional options for your test case, put them in the command-line format in `mysql-test/t/test_name-slave.opt`. For the master, put them in `mysql-test/t/test_name-master.opt`.
- If you have a question about the test suite, or have a test case to contribute, e-mail to internals@lists.mysql.com. As the list does not accept attachments, you should ftp all the relevant files to: <ftp://support.mysql.com/pub/mysql/Incoming/>

10.1.2.3 Reporting Bugs in the MySQL Test Suite

If your MySQL version doesn't pass the test suite you should do the following:

- Don't send a bug report before you have found out as much as possible of what when wrong! When you do it, please use the `mysqlbug` script so that we can get information about your system and MySQL version. See [\(undefined\) \[Bug reports\], page \(undefined\)](#).
- Make sure to include the output of `mysql-test-run`, as well as contents of all `.reject` files in `mysql-test/r` directory.
- If a test in the test suite fails, check if the test fails also when run by its own:

```
cd mysql-test
mysql-test-run --local test-name
```

If this fails, then you should configure MySQL with `--with-debug` and run `mysql-test-run` with the `--debug` option. If this also fails send the trace file `'var/tmp/master.trace'` to <ftp://support.mysql.com/pub/mysql/secret> so that we can examine it. Please remember to also include a full description of your system, the version of the `mysqld` binary and how you compiled it.

- Try also to run `mysql-test-run` with the `--force` option to see if there is any other test that fails.

- If you have compiled MySQL yourself, check our manual for how to compile MySQL on your platform or, preferable, use one of the binaries we have compiled for you at <http://www.mysql.com/downloads/>. All our standard binaries should pass the test suite !
- If you get an error, like `Result length mismatch` or `Result content mismatch` it means that the output of the test didn't match exactly the expected output. This could be a bug in MySQL or that your `mysqld` version produces slight different results under some circumstances.

Failed test results are put in a file with the same base name as the result file with the `.reject` extension. If your test case is failing, you should do a diff on the two files. If you cannot see how they are different, examine both with `od -c` and also check their lengths.

- If a test fails totally, you should check the logs file in the `mysql-test/var/log` directory for hints of what went wrong.
- If you have compiled MySQL with debugging you can try to debug this by running `mysql-test-run` with the `--gdb` and/or `--debug` options. See [\[Making trace files\]](#), page [\[undefined\]](#).

If you have not compiled MySQL for debugging you should probably do that. Just specify the `--with-debug` options to `configure`! See [\[Installing source\]](#), page [\[undefined\]](#).

10.2 Adding New Functions to MySQL

There are two ways to add new functions to MySQL:

- You can add the function through the user-definable function (UDF) interface. User-definable functions are added and removed dynamically using the `CREATE FUNCTION` and `DROP FUNCTION` statements. See [\[CREATE FUNCTION\]](#), page [\[undefined\]](#).
- You can add the function as a native (built in) MySQL function. Native functions are compiled into the `mysqld` server and become available on a permanent basis.

Each method has advantages and disadvantages:

- If you write a user-definable function, you must install the object file in addition to the server itself. If you compile your function into the server, you don't need to do that.
- You can add UDFs to a binary MySQL distribution. Native functions require you to modify a source distribution.
- If you upgrade your MySQL distribution, you can continue to use your previously installed UDFs. For native functions, you must repeat your modifications each time you upgrade.

Whichever method you use to add new functions, they may be used just like native functions such as `ABS()` or `SOUNDEX()`.

10.2.1 CREATE FUNCTION/DROP FUNCTION Syntax

```
CREATE [AGGREGATE] FUNCTION function_name RETURNS {STRING|REAL|INTEGER}
SONAME shared_library_name
```

DROP FUNCTION *function_name*

A user-definable function (UDF) is a way to extend MySQL with a new function that works like native (built in) MySQL functions such as `ABS()` and `CONCAT()`.

AGGREGATE is a new option for MySQL Version 3.23. An **AGGREGATE** function works exactly like a native MySQL **GROUP** function like `SUM` or `COUNT()`.

CREATE FUNCTION saves the function's name, type, and shared library name in the `mysql.func` system table. You must have the **INSERT** and **DELETE** privileges for the `mysql` database to create and drop functions.

All active functions are reloaded each time the server starts, unless you start `mysqld` with the `--skip-grant-tables` option. In this case, UDF initialisation is skipped and UDFs are unavailable. (An active function is one that has been loaded with **CREATE FUNCTION** and not removed with **DROP FUNCTION**.)

For instructions on writing user-definable functions, see [\[Adding functions\]](#), page [\[undefined\]](#). For the UDF mechanism to work, functions must be written in C or C++, your operating system must support dynamic loading and you must have compiled `mysqld` dynamically (not statically).

Note that to make **AGGREGATE** work, you must have a `mysql.func` table that contains the column `type`. If this is not the case, you should run the script `mysql_fix_privilege_tables` to get this fixed.

10.2.2 Adding a New User-definable Function

For the UDF mechanism to work, functions must be written in C or C++ and your operating system must support dynamic loading. The MySQL source distribution includes a file `'sql/udf_example.cc'` that defines 5 new functions. Consult this file to see how UDF calling conventions work.

For `mysqld` to be able to use UDF functions, you should configure MySQL with `--with-mysqld-ldflags=-rdynamic`. The reason is that to on many platforms (including Linux) you can load a dynamic library (with `dlopen()`) from a static linked program, which you would get if you are using `--with-mysqld-ldflags=-all-static`. If you want to use an UDF that needs to access symbols from `mysqld` (like the `methaphone` example in `'sql/udf_example.cc'` that uses `default_charset_info`), you must link the program with `-rdynamic` (see `man dlopen`).

For each function that you want to use in SQL statements, you should define corresponding C (or C++) functions. In the discussion below, the name "xxx" is used for an example function name. To distinguish between SQL and C/C++ usage, `XXX()` (uppercase) indicates a SQL function call, and `xxx()` (lowercase) indicates a C/C++ function call.

The C/C++ functions that you write to implement the interface for `XXX()` are:

`xxx()` (required)

The main function. This is where the function result is computed. The correspondence between the SQL type and return type of your C/C++ function is shown here:

SQL type	C/C++ type
----------	---------------

STRING	char *
INTEGER	long long
REAL	double

`xxx_init()` (optional)

The initialisation function for `xxx()`. It can be used to:

- Check the number of arguments to `XXX()`.
- Check that the arguments are of a required type or, alternatively, tell MySQL to coerce arguments to the types you want when the main function is called.
- Allocate any memory required by the main function.
- Specify the maximum length of the result.
- Specify (for `REAL` functions) the maximum number of decimals.
- Specify whether the result can be `NULL`.

`xxx_deinit()` (optional)

The deinitialisation function for `xxx()`. It should deallocate any memory allocated by the initialisation function.

When a SQL statement invokes `XXX()`, MySQL calls the initialisation function `xxx_init()` to let it perform any required setup, such as argument checking or memory allocation. If `xxx_init()` returns an error, the SQL statement is aborted with an error message and the main and deinitialisation functions are not called. Otherwise, the main function `xxx()` is called once for each row. After all rows have been processed, the deinitialisation function `xxx_deinit()` is called so it can perform any required cleanup.

For aggregate functions (like `SUM()`), you must also provide the following functions:

`xxx_reset()` (required)

Reset sum and insert the argument as the initial value for a new group.

`xxx_add()` (required)

Add the argument to the old sum.

When using aggregate UDF functions MySQL works the following way:

1. Call `xxx_init()` to let the aggregate function allocate the memory it will need to store results.
2. Sort the table according to the `GROUP BY` expression.
3. For the first row in a new group, call the `xxx_reset()` function.
4. For each new row that belongs in the same group, call the `xxx_add()` function.
5. When the group changes or after the last row has been processed, call `xxx()` to get the result for the aggregate.
6. Repeat 3-5 until all rows has been processed
7. Call `xxx_deinit()` to let the UDF free any memory it has allocated.

All functions must be thread-safe (not just the main function, but the initialisation and deinitialisation functions as well). This means that you are not allowed to allocate any global or static variables that change! If you need memory, you should allocate it in `xxx_init()` and free it in `xxx_deinit()`.

10.2.2.1 UDF Calling Sequences for simple functions

The main function should be declared as shown here. Note that the return type and parameters differ, depending on whether you will declare the SQL function `XXX()` to return `STRING`, `INTEGER`, or `REAL` in the `CREATE FUNCTION` statement:

For `STRING` functions:

```
char *xxx(UDF_INIT *initid, UDF_ARGS *args,
          char *result, unsigned long *length,
          char *is_null, char *error);
```

For `INTEGER` functions:

```
long long xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

For `REAL` functions:

```
double xxx(UDF_INIT *initid, UDF_ARGS *args,
            char *is_null, char *error);
```

The initialisation and deinitialisation functions are declared like this:

```
my_bool xxx_init(UDF_INIT *initid, UDF_ARGS *args, char *message);

void xxx_deinit(UDF_INIT *initid);
```

The `initid` parameter is passed to all three functions. It points to a `UDF_INIT` structure that is used to communicate information between functions. The `UDF_INIT` structure members are listed below. The initialisation function should fill in any members that it wishes to change. (To use the default for a member, leave it unchanged.):

my_bool maybe_null

`xxx_init()` should set `maybe_null` to 1 if `xxx()` can return `NULL`. The default value is 1 if any of the arguments are declared `maybe_null`.

unsigned int decimals

Number of decimals. The default value is the maximum number of decimals in the arguments passed to the main function. (For example, if the function is passed 1.34, 1.345, and 1.3, the default would be 3, because 1.345 has 3 decimals.)

unsigned int max_length

The maximum length of the string result. The default value differs depending on the result type of the function. For string functions, the default is the length of the longest argument. For integer functions, the default is 21 digits. For real functions, the default is 13 plus the number of decimals indicated by `initid->decimals`. (For numeric functions, the length includes any sign or decimal point characters.)

If you want to return a blob, you can set this to 65K or 16M; this memory is not allocated but used to decide which column type to use if there is a need to temporarily store the data.

char *ptr

A pointer that the function can use for its own purposes. For example, functions can use `initid->ptr` to communicate allocated memory between functions. In `xxx_init()`, allocate the memory and assign it to this pointer:

```
initid->ptr = allocated_memory;
```

In `xxx()` and `xxx_deinit()`, refer to `initid->ptr` to use or deallocate the memory.

10.2.2.2 UDF Calling Sequences for aggregate functions

Here follows a description of the different functions you need to define when you want to create an aggregate UDF function.

```
char *xxx_reset(UDF_INIT *initid, UDF_ARGS *args,
               char *is_null, char *error);
```

This function is called when MySQL finds the first row in a new group. In the function you should reset any internal summary variables and then set the given argument as the first argument in the group.

In many cases this is implemented internally by resetting all variables and then calling `xxx_add()`.

```
char *xxx_add(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

This function is called for all rows that belongs to the same group, except for the first row. In this you should add the value in `UDF_ARGS` to your internal summary variable.

The `xxx()` function should be declared identical as when you define a simple UDF function. See [\[UDF calling\]](#), page [\[undefined\]](#).

This function is called when all rows in the group has been processed. You should normally never access the `args` variable here but return your value based on your internal summary variables.

All argument processing in `xxx_reset()` and `xxx_add()` should be done identically as for normal UDF functions. See [\[UDF arguments\]](#), page [\[undefined\]](#).

The return value handling in `xxx()` should be done identically as for a normal UDF. See [\[UDF return values\]](#), page [\[undefined\]](#).

The pointer argument to `is_null` and `error` is the same for all calls to `xxx_reset()`, `xxx_add()` and `xxx()`. You can use this to remember that you got an error or if the `xxx()` function should return `NULL`. Note that you should not store a string into `*error`! This is just a 1 byte flag!

`is_null` is reset for each group (before calling `xxx_reset()`). `error` is never reset.

If `isnull` or `error` are set after `xxx()` then MySQL will return `NULL` as the result for the group function.

10.2.2.3 Argument Processing

The `args` parameter points to a `UDF_ARGS` structure that has the members listed here:

```
unsigned int arg_count
```

The number of arguments. Check this value in the initialisation function if you want your function to be called with a particular number of arguments. For example:

```

if (args->arg_count != 2)
{
    strcpy(message,"XXX() requires two arguments");
    return 1;
}

```

`enum Item_result *arg_type`

The types for each argument. The possible type values are `STRING_RESULT`, `INT_RESULT`, and `REAL_RESULT`.

To make sure that arguments are of a given type and return an error if they are not, check the `arg_type` array in the initialisation function. For example:

```

if (args->arg_type[0] != STRING_RESULT ||
    args->arg_type[1] != INT_RESULT)
{
    strcpy(message,"XXX() requires a string and an integer");
    return 1;
}

```

As an alternative to requiring your function's arguments to be of particular types, you can use the initialisation function to set the `arg_type` elements to the types you want. This causes MySQL to coerce arguments to those types for each call to `xxx()`. For example, to specify coercion of the first two arguments to string and integer, do this in `xxx_init()`:

```

args->arg_type[0] = STRING_RESULT;
args->arg_type[1] = INT_RESULT;

```

`char **args`

`args->args` communicates information to the initialisation function about the general nature of the arguments your function was called with. For a constant argument `i`, `args->args[i]` points to the argument value. (See below for instructions on how to access the value properly.) For a non-constant argument, `args->args[i]` is 0. A constant argument is an expression that uses only constants, such as 3 or $4*7-2$ or `SIN(3.14)`. A non-constant argument is an expression that refers to values that may change from row to row, such as column names or functions that are called with non-constant arguments.

For each invocation of the main function, `args->args` contains the actual arguments that are passed for the row currently being processed.

Functions can refer to an argument `i` as follows:

- An argument of type `STRING_RESULT` is given as a string pointer plus a length, to allow handling of binary data or data of arbitrary length. The string contents are available as `args->args[i]` and the string length is `args->lengths[i]`. You should not assume that strings are null-terminated.
- For an argument of type `INT_RESULT`, you must cast `args->args[i]` to a long long value:

```

long long int_val;
int_val = *((long long*) args->args[i]);

```

- For an argument of type `REAL_RESULT`, you must cast `args->args[i]` to a double value:

```
double    real_val;
real_val = *((double*) args->args[i]);
```

`unsigned long *lengths`

For the initialisation function, the `lengths` array indicates the maximum string length for each argument. You should not change these. For each invocation of the main function, `lengths` contains the actual lengths of any string arguments that are passed for the row currently being processed. For arguments of types `INT_RESULT` or `REAL_RESULT`, `lengths` still contains the maximum length of the argument (as for the initialisation function).

10.2.2.4 Return Values and Error Handling

The initialisation function should return 0 if no error occurred and 1 otherwise. If an error occurs, `xxx_init()` should store a null-terminated error message in the `message` parameter. The message will be returned to the client. The message buffer is `MYSQL_ERRMSG_SIZE` characters long, but you should try to keep the message to less than 80 characters so that it fits the width of a standard terminal screen.

The return value of the main function `xxx()` is the function value, for `long long` and `double` functions. A string functions should return a pointer to the result and store the length of the string in the `length` arguments.

Set these to the contents and length of the return value. For example:

```
memcpy(result, "result string", 13);
*length = 13;
```

The `result` buffer that is passed to the `calc` function is 255 byte big. If your result fits in this, you don't have to worry about memory allocation for results.

If your string function needs to return a string longer than 255 bytes, you must allocate the space for it with `malloc()` in your `xxx_init()` function or your `xxx()` function and free it in your `xxx_deinit()` function. You can store the allocated memory in the `ptr` slot in the `UDF_INIT` structure for reuse by future `xxx()` calls. See [\[UDF calling\]](#), page [\[undefined\]](#).

To indicate a return value of `NULL` in the main function, set `is_null` to 1:

```
*is_null = 1;
```

To indicate an error return in the main function, set the `error` parameter to 1:

```
*error = 1;
```

If `xxx()` sets `*error` to 1 for any row, the function value is `NULL` for the current row and for any subsequent rows processed by the statement in which `XXX()` was invoked. (`xxx()` will not even be called for subsequent rows.) **Note:** in MySQL versions prior to 3.22.10, you should set both `*error` and `*is_null`:

```
*error = 1;
*is_null = 1;
```

10.2.2.5 Compiling and Installing User-definable Functions

Files implementing UDFs must be compiled and installed on the host where the server runs. This process is described below for the example UDF file ‘`udf_example.cc`’ that is included in the MySQL source distribution. This file contains the following functions:

- `metaphon()` returns a metaphon string of the string argument. This is something like a soundex string, but it’s more tuned for English.
- `myfunc_double()` returns the sum of the ASCII values of the characters in its arguments, divided by the sum of the length of its arguments.
- `myfunc_int()` returns the sum of the length of its arguments.
- `sequence([const int])` returns an sequence starting from the given number or 1 if no number has been given.
- `lookup()` returns the IP number for a hostname.
- `reverse_lookup()` returns the hostname for an IP number. The function may be called with a string “`xxx.xxx.xxx.xxx`” or four numbers.

A dynamically loadable file should be compiled as a sharable object file, using a command something like this:

```
shell> gcc -shared -o udf_example.so myfunc.cc
```

You can easily find out the correct compiler options for your system by running this command in the ‘`sql`’ directory of your MySQL source tree:

```
shell> make udf_example.o
```

You should run a compile command similar to the one that `make` displays, except that you should remove the `-c` option near the end of the line and add `-o udf_example.so` to the end of the line. (On some systems, you may need to leave the `-c` on the command.)

Once you compile a shared object containing UDFs, you must install it and tell MySQL about it. Compiling a shared object from ‘`udf_example.cc`’ produces a file named something like ‘`udf_example.so`’ (the exact name may vary from platform to platform). Copy this file to some directory searched by `ld`, such as ‘`/usr/lib`’. On many systems, you can set the `LD_LIBRARY` or `LD_LIBRARY_PATH` environment variable to point at the directory where you have your UDF function files. The `dlopen` manual page tells you which variable you should use on your system. You should set this in `mysql.server` or `safe_mysqld` startup scripts and restart `mysqld`.

After the library is installed, notify `mysqld` about the new functions with these commands:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME "udf_example.so";
mysql> CREATE FUNCTION myfunc_double RETURNS REAL SONAME "udf_example.so";
mysql> CREATE FUNCTION myfunc_int RETURNS INTEGER SONAME "udf_example.so";
mysql> CREATE FUNCTION lookup RETURNS STRING SONAME "udf_example.so";
mysql> CREATE FUNCTION reverse_lookup
-> RETURNS STRING SONAME "udf_example.so";
mysql> CREATE AGGREGATE FUNCTION avgcost
-> RETURNS REAL SONAME "udf_example.so";
```

Functions can be deleted using `DROP FUNCTION`:

```
mysql> DROP FUNCTION metaphon;
mysql> DROP FUNCTION myfunc_double;
```

```
mysql> DROP FUNCTION myfunc_int;
mysql> DROP FUNCTION lookup;
mysql> DROP FUNCTION reverse_lookup;
mysql> DROP FUNCTION avgcost;
```

The `CREATE FUNCTION` and `DROP FUNCTION` statements update the system table `func` in the `mysql` database. The function's name, type and shared library name are saved in the table. You must have the `INSERT` and `DELETE` privileges for the `mysql` database to create and drop functions.

You should not use `CREATE FUNCTION` to add a function that has already been created. If you need to reinstall a function, you should remove it with `DROP FUNCTION` and then reinstall it with `CREATE FUNCTION`. You would need to do this, for example, if you recompile a new version of your function, so that `mysqld` gets the new version. Otherwise, the server will continue to use the old version.

Active functions are reloaded each time the server starts, unless you start `mysqld` with the `--skip-grant-tables` option. In this case, UDF initialisation is skipped and UDFs are unavailable. (An active function is one that has been loaded with `CREATE FUNCTION` and not removed with `DROP FUNCTION`.)

10.2.3 Adding a New Native Function

The procedure for adding a new native function is described here. Note that you cannot add native functions to a binary distribution because the procedure involves modifying MySQL source code. You must compile MySQL yourself from a source distribution. Also note that if you migrate to another version of MySQL (for example, when a new version is released), you will need to repeat the procedure with the new version.

To add a new native MySQL function, follow these steps:

1. Add one line to `'lex.h'` that defines the function name in the `sql_functions[]` array.
2. If the function prototype is simple (just takes zero, one, two or three arguments), you should in `lex.h` specify `SYM(FUNC_ARG#)` (where `#` is the number of arguments) as the second argument in the `sql_functions[]` array and add a function that creates a function object in `'item_create.cc'`. Take a look at `"ABS"` and `create_funcs_abs()` for an example of this.

If the function prototype is complicated (for example takes a variable number of arguments), you should add two lines to `'sql_yacc.yy'`. One indicates the preprocessor symbol that `yacc` should define (this should be added at the beginning of the file). Then define the function parameters and add an "item" with these parameters to the `simple_expr` parsing rule. For an example, check all occurrences of `ATAN` in `'sql_yacc.yy'` to see how this is done.

3. In `'item_func.h'`, declare a class inheriting from `Item_num_func` or `Item_str_func`, depending on whether your function returns a number or a string.
4. In `'item_func.cc'`, add one of the following declarations, depending on whether you are defining a numeric or string function:

```
double  Item_func_newname::val()
longlong Item_func_newname::val_int()
String  *Item_func_newname::Str(String *str)
```

If you inherit your object from any of the standard items (like `Item_num_func`), you probably only have to define one of the above functions and let the parent object take care of the other functions. For example, the `Item_str_func` class defines a `val()` function that executes `atof()` on the value returned by `::str()`.

5. You should probably also define the following object function:

```
void Item_func_newname::fix_length_and_dec()
```

This function should at least calculate `max_length` based on the given arguments. `max_length` is the maximum number of characters the function may return. This function should also set `maybe_null = 0` if the main function can't return a NULL value. The function can check if any of the function arguments can return NULL by checking the arguments `maybe_null` variable. You can take a look at `Item_func_mod::fix_length_and_dec` for a typical example of how to do this.

All functions must be thread-safe (in other words, don't use any global or static variables in the functions without protecting them with mutexes).

If you want to return NULL, from `::val()`, `::val_int()` or `::str()` you should set `null_value` to 1 and return 0.

For `::str()` object functions, there are some additional considerations to be aware of:

- The `String *str` argument provides a string buffer that may be used to hold the result. (For more information about the `String` type, take a look at the 'sql_string.h' file.)
- The `::str()` function should return the string that holds the result or `(char*) 0` if the result is NULL.
- All current string functions try to avoid allocating any memory unless absolutely necessary!

10.3 Adding New Procedures to MySQL

In MySQL, you can define a procedure in C++ that can access and modify the data in a query before it is sent to the client. The modification can be done on row-by-row or `GROUP BY` level.

We have created an example procedure in MySQL Version 3.23 to show you what can be done.

Additionally we recommend you to take a look at `mylua`. With this you can use the LUA language to load a procedure at runtime into `mysqld`.

10.3.1 Procedure Analyse

```
analyse([max elements],[max memory]])
```

This procedure is defined in the 'sql/sql_analyse.cc'. This examines the result from your query and returns an analysis of the results:

- `max elements` (default 256) is the maximum number of distinct values `analyse` will notice per column. This is used by `analyse` to check if the optimal column type should be of type `ENUM`.
- `max memory` (default 8192) is the maximum memory `analyse` should allocate per column while trying to find all distinct values.

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max elements],[max memory]])
```

10.3.2 Writing a Procedure

For the moment, the only documentation for this is the source.

You can find all information about procedures by examining the following files:

- ‘sql/sql_analyse.cc’
- ‘sql/procedure.h’
- ‘sql/procedure.cc’
- ‘sql/sql_select.cc’

Appendix A Problems and Common Errors

This chapter lists some common problems and error messages that users have run into. You will learn how to figure out what the problem is, and what to do to solve it. You will also find proper solutions to some common problems.

A.1 How to Determine What Is Causing Problems

When you run into problems, the first thing you should do is to find out which program / piece of equipment is causing problems:

- If you have one of the following symptoms, then it is probably a hardware (like memory, motherboard, CPU, or hard disk) or kernel problem:
 - The keyboard doesn't work. This can normally be checked by pressing Caps Lock. If the Caps Lock light doesn't change you have to replace your keyboard. (Before doing this, you should try to reboot your computer and check all cables to the keyboard.)
 - The mouse pointer doesn't move.
 - The machine doesn't answer to a remote machine's pings.
 - Different, unrelated programs don't behave correctly.
 - If your system rebooted unexpectedly (a faulty user level program should **never** be able to take down your system).

In this case you should start by checking all your cables and run some diagnostic tool to check your hardware! You should also check if there are any patches, updates, or service packs for your operating system that could likely solve your problems. Check also that all your libraries (like glibc) are up to date.

It's always good to use a machine with ECC memory to discover memory problems early!

- If your keyboard is locked up, you may be able to fix this by logging into your machine from another machine and execute `kbd_mode -a` on it.
- Please examine your system log file (`/var/log/messages` or similar) for reasons for your problems. If you think the problem is in MySQL then you should also examine MySQL's log files. See [\[Binary log\]](#), page [\[Binary log\]](#).
- If you don't think you have hardware problems, you should try to find out which program is causing problems.
Try using `top`, `ps`, `taskmanager`, or some similar program, to check which program is taking all CPU or is locking the machine.
- Check with `top`, `df`, or a similar program if you are out of memory, disk space, open files, or some other critical resource.
- If the problem is some runaway process, you can always try to kill it. If it doesn't want to die, there is probably a bug in the operating system.

If after you have examined all other possibilities and you have concluded that it's the MySQL server or a MySQL client that is causing the problem, it's time to do a bug report for our mailing list or our support team. In the bug report, try to give a very detailed

description of how the system is behaving and what you think is happening. You should also state why you think it's MySQL that is causing the problems. Take into consideration all the situations in this chapter. State any problems exactly how they appear when you examine your system. Use the 'cut and paste' method for any output and/or error messages from programs and/or log files!

Try to describe in detail which program is not working and all symptoms you see! We have in the past received many bug reports that just state "the system doesn't work". This doesn't provide us with any information about what could be the problem.

If a program fails, it's always useful to know:

- Has the program in question made a segmentation fault (core dumped)?
- Is the program taking up the whole CPU? Check with `top`. Let the program run for a while, it may be evaluating something heavy.
- If it's the `mysqld` server that is causing problems, can you do `mysqladmin -u root ping` or `mysqladmin -u root processlist`?
- What does a client program say (try with `mysql`, for example) when you try to connect to the MySQL server? Does the client jam? Do you get any output from the program?

When sending a bug report, you should of follow the outlines described in this manual. See [\[Asking questions\]](#), page [\[undefined\]](#).

A.2 Common Errors When Using MySQL

This section lists some errors that users frequently get. You will find descriptions of the errors, and how to solve the problem here.

A.2.1 Access denied Error

See [\[Access denied\]](#), page [\[undefined\]](#). See [\[Privileges\]](#), page [\[undefined\]](#).

A.2.2 MySQL server has gone away Error

This section also covers the related `Lost connection to server during query` error.

The most common reason for the `MySQL server has gone away` error is that the server timed out and closed the connection. By default, the server closes the connection after 8 hours if nothing has happened. You can change the time limit by setting the `wait_timeout` variable when you start `mysqld`.

Another common reason to receive the `MySQL server has gone away` error is because you have issued a "close" on your MySQL connection and then tried to run a query on the closed connection.

If you have a script, you just have to issue the query again for the client to do an automatic reconnection.

You normally can get the following error codes in this case (which one you get is OS-dependent):

Error code	Description
------------	-------------

<code>CR_SERVER_GONE_ERROR</code>	The client couldn't send a question to the server.
<code>CR_SERVER_LOST</code>	The client didn't get an error when writing to the server, but it didn't get a full answer (or any answer) to the question.

You will also get this error if someone has kills the running thread with `kill #threadid#`. You can check that the MySQL hasn't died by executing `mysqladmin version` and examining the uptime. If the problem is that `mysqld` crashed you should concentrate one finding the reason for the crash. You should in this case start by checking if issuing the query again will kill MySQL again. See [\[Crashing\]](#), page [\[undefined\]](#).

You can also get these errors if you send a query to the server that is incorrect or too large. If `mysqld` gets a packet that is too large or out of order, it assumes that something has gone wrong with the client and closes the connection. If you need big queries (for example, if you are working with big BLOB columns), you can increase the query limit by starting `mysqld` with the `-O max_allowed_packet=#` option (default 1M). The extra memory is allocated on demand, so `mysqld` will allocate more memory only when you issue a big query or when `mysqld` must return a big result row!

You will also get a lost connection if you are sending a packet $\geq 16\text{M}$ if your client is older than 4.0.8 and your server is 4.0.8 and above, or the other way around.

If you want to make a bug report regarding this problem, be sure that you include the following information:

- Include information if MySQL died or not. (You can find this in the `hostname.err` file. See [\[Crashing\]](#), page [\[undefined\]](#)).
- If a specific query kills `mysqld` and the involved tables were checked with `CHECK TABLE` before you did the query, can you do a test case for this? See [\[Reproduceable test case\]](#), page [\[undefined\]](#).
- What is the value of the `wait_timeout` variable in the MySQL server ? `mysqladmin variables` gives you the value of this
- Have you tried to run `mysqld` with `--log` and check if the issued query appears in the log ?

See [\[Asking questions\]](#), page [\[undefined\]](#).

A.2.3 Can't connect to [local] MySQL server Error

A MySQL client on Unix can connect to the `mysqld` server in two different ways: Unix sockets, which connect through a file in the file system (default `'/tmp/mysql.sock'`) or TCP/IP, which connects through a port number. Unix sockets are faster than TCP/IP but can only be used when connecting to a server on the same computer. Unix sockets are used if you don't specify a hostname or if you specify the special hostname `localhost`.

On Windows, if the `mysqld` server is running on 9x/Me, you can connect only via TCP/IP. If the server is running on NT/2000/XP and `mysqld` is started with `--enable-named-pipe`, you can also connect with named pipes. The name of the named pipe is `MySQL`. If you don't give a hostname when connecting to `mysqld`, a MySQL client will first try to connect to the named pipe, and if this doesn't work it will connect to the TCP/IP port. You can force the use of named pipes on Windows by using `.` as the hostname.

The error (2002) `Can't connect to ...` normally means that there isn't a MySQL server running on the system or that you are using a wrong socket file or TCP/IP port when trying to connect to the `mysqld` server.

Start by checking (using `ps` or the task manager on Windows) that there is a process running named `mysqld` on your server! If there isn't any `mysqld` process, you should start one. See [\[Starting server\]](#), page [\[undefined\]](#).

If a `mysqld` process is running, you can check the server by trying these different connections (the port number and socket pathname might be different in your setup, of course):

```
shell> mysqladmin version
shell> mysqladmin variables
shell> mysqladmin -h 'hostname' version variables
shell> mysqladmin -h 'hostname' --port=3306 version
shell> mysqladmin -h 'ip for your host' version
shell> mysqladmin --protocol=socket --socket=/tmp/mysql.sock version
```

Note the use of backquotes rather than forward quotes with the `hostname` command; these cause the output of `hostname` (that is, the current hostname) to be substituted into the `mysqladmin` command.

Here are some reasons the `Can't connect to local MySQL server` error might occur:

- `mysqld` is not running.
- You are running on a system that uses MIT-pthreads. If you are running on a system that doesn't have native threads, `mysqld` uses the MIT-pthreads package. See [\[Which OS\]](#), page [\[undefined\]](#). However, not all MIT-pthreads versions support Unix sockets. On a system without sockets support you must always specify the hostname explicitly when connecting to the server. Try using this command to check the connection to the server:

```
shell> mysqladmin -h 'hostname' version
```

- Someone has removed the Unix socket that `mysqld` uses (default `'/tmp/mysql.sock'`). You might have a `cron` job that removes the MySQL socket (for example, a job that removes old files from the `/tmp` directory). You can always run `mysqladmin version` and check that the socket `mysqladmin` is trying to use really exists. The fix in this case is to change the `cron` job to not remove `'mysql.sock'` or to place the socket somewhere else. See [\[Problems with mysql.sock\]](#), page [\[undefined\]](#).
- You have started the `mysqld` server with the `--socket=/path/to/socket` option. If you change the socket pathname for the server, you must also notify the MySQL clients about the new path. You can do this by providing the socket path as an argument to the client. See [\[Problems with mysql.sock\]](#), page [\[undefined\]](#).
- You are using Linux and one thread has died (core dumped). In this case you must kill the other `mysqld` threads (for example, with the `mysql_zap` script before you can start a new MySQL server. See [\[Crashing\]](#), page [\[undefined\]](#).
- You may not have read and write privilege to either the directory that holds the socket file or privilege to the socket file itself. In this case you have to either change the privilege for the directory / file or restart `mysqld` so that it uses a directory that you can access.

If you get the error message `Can't connect to MySQL server on some_hostname`, you can try the following things to find out what the problem is :

- Check if the server is up by doing `telnet your-host-name tcp-ip-port-number` and press Enter a couple of times. If there is a MySQL server running on this port you should get a responses that includes the version number of the running MySQL server. If you get an error like `telnet: Unable to connect to remote host: Connection refused`, then there is no server running on the given port.
- Try connecting to the `mysqld` daemon on the local machine and check the TCP/IP port that `mysqld` it's configured to use (variable `port`) with `mysqladmin variables`.
- Check that your `mysqld` server is not started with the `--skip-networking` option.

A.2.4 Host '...' is blocked Error

If you get an error like this:

```
Host 'hostname' is blocked because of many connection errors.
Unblock with 'mysqladmin flush-hosts'
```

this means that `mysqld` has gotten a lot (`max_connect_errors`) of connect requests from the host 'hostname' that have been interrupted in the middle. After `max_connect_errors` failed requests, `mysqld` assumes that something is wrong (like an attack from a cracker), and blocks the site from further connections until someone executes the command `mysqladmin flush-hosts`.

By default, `mysqld` blocks a host after 10 connection errors. You can easily adjust this by starting the server like this:

```
shell> safe_mysqld -O max_connect_errors=10000 &
```

Note that if you get this error message for a given host, you should first check that there isn't anything wrong with TCP/IP connections from that host. If your TCP/IP connections aren't working, it won't do you any good to increase the value of the `max_connect_errors` variable!

A.2.5 Too many connections Error

If you get the error `Too many connections` when you try to connect to MySQL, this means that there is already `max_connections` clients connected to the `mysqld` server.

If you need more connections than the default (100), then you should restart `mysqld` with a bigger value for the `max_connections` variable.

Note that `mysqld` actually allows (`max_connections+1`) clients to connect. The last connection is reserved for a user with the `SUPER` privilege. By not giving this privilege to normal users (they shouldn't need this), an administrator with this privilege can log in and use `SHOW PROCESSLIST` to find out what could be wrong. See [\[SHOW PROCESSLIST\]](#), page [\[SHOW PROCESSLIST\]](#).

The maximum number of connects MySQL is depending on how good the thread library is on a given platform. Linux or Solaris should be able to support 500-1000 simultaneous connections, depending on how much RAM you have and what your clients are doing.

A.2.6 Some non-transactional changed tables couldn't be rolled back Error

If you get the error/warning: `Warning: Some non-transactional changed tables couldn't be rolled back when trying to do a ROLLBACK`, this means that some of the

tables you used in the transaction didn't support transactions. These non-transactional tables will not be affected by the ROLLBACK statement.

The most typical case when this happens is when you have tried to create a table of a type that is not supported by your `mysqld` binary. If `mysqld` doesn't support a table type (or if the table type is disabled by a startup option), it will instead create the table type with the table type that is most resembles to the one you requested, probably MyISAM.

You can check the table type for a table by doing:

```
SHOW TABLE STATUS LIKE 'table_name'. See <undefined> [SHOW TABLE STATUS],  
page <undefined>.
```

You can check the extensions your `mysqld` binary supports by doing:

```
show variables like 'have_%'. See <undefined> [SHOW VARIABLES], page <undefined>.
```

A.2.7 Out of memory Error

If you issue a query and get something like the following error:

```
mysql: Out of memory at line 42, 'malloc.c'  
mysql: needed 8136 byte (8k), memory in use: 12481367 bytes (12189k)  
ERROR 2008: MySQL client ran out of memory
```

note that the error refers to the MySQL client `mysql`. The reason for this error is simply that the client does not have enough memory to store the whole result.

To remedy the problem, first check that your query is correct. Is it reasonable that it should return so many rows? If so, you can use `mysql --quick`, which uses `mysql_use_result()` to retrieve the result set. This places less of a load on the client (but more on the server).

A.2.8 Packet too large Error

When a MySQL client or the `mysqld` server gets a packet bigger than `max_allowed_packet` bytes, it issues a `Packet too large` error and closes the connection.

In MySQL 3.23 the biggest possible packet is 16M (due to limits in the client/server protocol). In MySQL 4.0.1 and up, this is only limited by the amount on memory you have on your server (up to a theoretical maximum of 2G).

A communication packet is a single SQL statement sent to the MySQL server or a single row that is sent to the client.

When a MySQL client or the `mysqld` server gets a packet bigger than `max_allowed_packet` bytes, it issues a `Packet too large` error and closes the connection. With some clients, you may also get `Lost connection to MySQL server during query` error if the communication packet is too big.

Note that both the client and the server has it's own `max_allowed_packet` variable. If you want to handle big packets, you have to increase this variable both in the client and in the server.

It's safe to increase this variable as memory is only allocated when needed; this variable is more a precaution to catch wrong packets between the client/server and also to ensure that you don't accidentally use big packets so that you run out of memory.

If you are using the `mysql` client, you may specify a bigger buffer by starting the client with `mysql --set-variable=max_allowed_packet=8M`. Other clients have different methods to

set this variable. Please note that `--set-variable` is deprecated since MySQL 4.0, just use `--max-allowed-packet=8M` instead.

You can use the option file to set `max_allowed_packet` to a larger size in `mysqld`. For example, if you are expecting to store the full length of a `MEDIUMBLOB` into a table, you'll need to start the server with the `set-variable=max_allowed_packet=16M` option.

You can also get strange problems with large packets if you are using big blobs, but you haven't given `mysqld` access to enough memory to handle the query. If you suspect this is the case, try adding `ulimit -d 256000` to the beginning of the `safe_mysqld` script and restart `mysqld`.

A.2.9 Communication Errors / Aborted Connection

Starting with MySQL 3.23.40 you only get the Aborted connection error if you start `mysqld` with `--warnings`.

If you find errors like the following in your error log.

```
010301 14:38:23 Aborted connection 854 to db: 'users' user: 'josh'
```

See [\(undefined\) \[Error log\]](#), page [\(undefined\)](#).

This means that something of the following has happened:

- The client program did not call `mysql_close()` before exit.
- The client had been sleeping more than `wait_timeout` or `interactive_timeout` without doing any requests. See [\(undefined\) \[wait_timeout\]](#), page [\(undefined\)](#). See [\(undefined\) \[interactive_timeout\]](#), page [\(undefined\)](#).
- The client program ended abruptly in the middle of the transfer.

When the above happens, the server variable `Aborted_clients` is incremented.

The server variable `Aborted_connects` is incremented when:

- When a connection packet doesn't contain the right information.
- When the user didn't have privileges to connect to a database.
- When a user uses a wrong password.
- When it takes more than `connect_timeout` seconds to get a connect package. See [\(undefined\) \[connect_timeout\]](#), page [\(undefined\)](#).

Note that the above could indicate that someone is trying to break into your database!

Other reasons for problems with Aborted clients / Aborted connections.

- Usage of Ethernet protocol with Linux, both half and full duplex. Many Linux Ethernet drivers have this bug. You should test for this bug by transferring a huge file via ftp between these two machines. If a transfer goes in burst-pause-burst-pause ... mode then you are experiencing a Linux duplex syndrome. The only solution is switching duplex mode for both your network card and Hub/Switch to either full duplex or to half duplex and testing the results to decide on the best setting.
- Some problem with the thread library that causes interrupts on reads.
- Badly configured TCP/IP.
- Faulty Ethernets or hubs or switches, cables ... This can be diagnosed properly only by replacing hardware.

- `max_allowed_packet` is too small or queries require more memory than you have allocated for `mysqld`. See [\[Packet too large\]](#), page [\(undefined\)](#).

A.2.10 The table is full Error

There is a couple of different cases when you can get this error:

- You are using an older MySQL version (before 3.23.0) when an in-memory temporary table becomes larger than `tmp_table_size` bytes. To avoid this problem, you can use the `-O tmp_table_size=#` option to make `mysqld` increase the temporary table size or use the SQL option `BIG_TABLES` before you issue the problematic query. See [\[SET\]](#), page [\(undefined\)](#).

You can also start `mysqld` with the `--big-tables` option. This is exactly the same as using `BIG_TABLES` for all queries.

In MySQL Version 3.23, in-memory temporary tables will automatically be converted to a disk-based MyISAM table after the table size gets bigger than `tmp_table_size`.

- You are using InnoDB tables and run out of room in the InnoDB tablespace. In this case the solution is to extend the InnoDB tablespace.
- You are using ISAM or MyISAM tables on an OS that only supports files of 2G in size and you have hit this limit for the data or index file.
- You are using MyISAM tables and the needed data or index size is bigger than what MySQL has allocated pointers for. (If you don't specify `MAX_ROWS` to `CREATE TABLE` MySQL will only allocate pointers to hold 4G of data).

You can check the maximum data/index sizes by doing

```
SHOW TABLE STATUS FROM database LIKE 'table_name';
```

or using `myisamchk -dv database/table_name`.

If this is the problem, you can fix it by doing something like:

```
ALTER TABLE table_name MAX_ROWS=1000000000 AVG_ROW_LENGTH=nnn;
```

You only have to specify `AVG_ROW_LENGTH` for tables with BLOB/TEXT fields as in this case MySQL can't optimise the space required based only on the number of rows.

A.2.11 Can't create/write to file Error

If you get an error for some queries of type:

```
Can't create/write to file '\\sqla3fe_0.ism'.
```

this means that MySQL can't create a temporary file for the result set in the given temporary directory. (The above error is a typical error message on Windows, and the Unix error message is similar.) The fix is to start `mysqld` with `--tmpdir=path` or to add to your option file:

```
[mysqld]
tmpdir=C:/temp
```

assuming that the `'c:\\temp'` directory exists. See [\[Option files\]](#), page [\(undefined\)](#).

Check also the error code that you get with `perror`. One reason may also be a disk full error;

```
shell> perror 28
Error code 28: No space left on device
```

A.2.12 Commands out of sync Error in Client

If you get **Commands out of sync; you can't run this command now** in your client code, you are calling client functions in the wrong order!

This can happen, for example, if you are using `mysql_use_result()` and try to execute a new query before you have called `mysql_free_result()`. It can also happen if you try to execute two queries that return data without a `mysql_use_result()` or `mysql_store_result()` in between.

A.2.13 Ignoring user Error

If you get the following error:

```
Found wrong password for user: 'some_user@some_host'; ignoring user
```

this means that when `mysqld` was started or when it reloaded the permissions tables, it found an entry in the `user` table with an invalid password. As a result, the entry is simply ignored by the permission system.

Possible causes of and fixes for this problem:

- You may be running a new version of `mysqld` with an old `user` table. You can check this by executing `mysqlshow mysql user` to see if the password field is shorter than 16 characters. If so, you can correct this condition by running the `scripts/add_long_password` script.
- The user has an old password (8 characters long) and you didn't start `mysqld` with the `--old-protocol` option. Update the user in the `user` table with a new password or restart `mysqld` with `--old-protocol`.
- You have specified a password in the `user` table without using the `PASSWORD()` function. Use `mysql` to update the user in the `user` table with a new password. Make sure to use the `PASSWORD()` function:

```
mysql> UPDATE user SET password=PASSWORD('your password')
-> WHERE user='XXX';
```

A.2.14 Table 'xxx' doesn't exist Error

If you get the error **Table 'xxx' doesn't exist** or **Can't find file: 'xxx' (errno: 2)**, this means that no table exists in the current database with the name `xxx`.

Note that as MySQL uses directories and files to store databases and tables, the database and table names are **case-sensitive!** (On Windows the databases and tables names are not case-sensitive, but all references to a given table within a query must use the same case!)

You can check which tables you have in the current database with `SHOW TABLES`. See [\(undefined\) \[SHOW\], page \(undefined\)](#).

A.2.15 Can't initialize character set xxx error

If you get an error like:

```
MySQL Connection Failed: Can't initialize character set xxx
```

This means one of the following things:

- The character set is a multi-byte character set and you have no support for the character set in the client.

In this case you need to recompile the client with `--with-charset=xxx` or with `--with-extra-charsets=xxx`. See [\[configure options\]](#), page [\[undefined\]](#).

All standard MySQL binaries are compiled with `--with-extra-character-sets=complex` which will enable support for all multi-byte character sets. See [\[Character sets\]](#), page [\[undefined\]](#).

- The character set is a simple character set which is not compiled into `mysqld` and the character set definition files are not in the place where the client expects to find them.

In this case you need to:

- Recompile the client with support for the character set. See [\[configure options\]](#), page [\[undefined\]](#).
- Specify to the client where the character set definition files are. For many clients you can do this with the `--character-sets-dir=path-to-charset-dir` option.
- Copy the character definition files to the path where the client expects them to be.

A.2.16 File Not Found

If you get `ERROR '...' not found (errno: 23), Can't open file: ... (errno: 24)`, or any other error with `errno 23` or `errno 24` from MySQL, it means that you haven't allocated enough file descriptors for MySQL. You can use the `perror` utility to get a description of what the error number means:

```
shell> perror 23
File table overflow
shell> perror 24
Too many open files
shell> perror 11
Resource temporarily unavailable
```

The problem here is that `mysqld` is trying to keep open too many files simultaneously. You can either tell `mysqld` not to open so many files at once or increase the number of file descriptors available to `mysqld`.

To tell `mysqld` to keep open fewer files at a time, you can make the table cache smaller by using the `-O table_cache=32` option to `safe_mysqld` (the default value is 64). Reducing the value of `max_connections` will also reduce the number of open files (the default value is 90).

To change the number of file descriptors available to `mysqld`, you can use the option `--open-files-limit=#` to `safe_mysqld` or `-O open-files-limit=#` to `mysqld`. See [\[open_files_limit\]](#), page [\[undefined\]](#). The easiest way to do that is to add the option to your option file. See [\[Option files\]](#), page [\[undefined\]](#). If you have an

old `mysqld` version that doesn't support this, you can edit the `safe_mysqld` script. There is a commented-out line `ulimit -n 256` in the script. You can remove the '#' character to uncomment this line, and change the number 256 to affect the number of file descriptors available to `mysqld`.

`ulimit` (and `open-files-limit`) can increase the number of file descriptors, but only up to the limit imposed by the operating system. There is also a 'hard' limit that can only be overridden if you start `safe_mysqld` or `mysqld` as root (just remember that you need to also use the `--user=...` option in this case). If you need to increase the OS limit on the number of file descriptors available to each process, consult the documentation for your operating system.

Note that if you run the `tcsh` shell, `ulimit` will not work! `tcsh` will also report incorrect values when you ask for the current limits! In this case you should start `safe_mysqld` with `sh`!

A.3 Installation Related Issues

A.3.1 Problems When Linking with the MySQL Client Library

If you are linking your program and you get errors for unreferenced symbols that start with `mysql_`, like the following:

```
/tmp/ccFKsdPa.o: In function 'main':
/tmp/ccFKsdPa.o(.text+0xb): undefined reference to 'mysql_init'
/tmp/ccFKsdPa.o(.text+0x31): undefined reference to 'mysql_real_connect'
/tmp/ccFKsdPa.o(.text+0x57): undefined reference to 'mysql_real_connect'
/tmp/ccFKsdPa.o(.text+0x69): undefined reference to 'mysql_error'
/tmp/ccFKsdPa.o(.text+0x9a): undefined reference to 'mysql_close'
```

you should be able to solve this by adding `-Lpath-to-the-mysql-library -lmysqlclient last` on your link line.

If you get `undefined reference` errors for the `uncompress` or `compress` function, add `-lz last` on your link line and try again!

If you get `undefined reference` errors for functions that should exist on your system, like `connect`, check the man page for the function in question, for which libraries you should add to the link line!

If you get `undefined reference` errors for functions that don't exist on your system, like the following:

```
mf_format.o(.text+0x201): undefined reference to '__lxstat'
```

it usually means that your library is compiled on a system that is not 100% compatible with yours. In this case you should download the latest MySQL source distribution and compile this yourself. See [\[Installing source\]](#), page [\[undefined\]](#).

If you are trying to run a program and you then get errors for unreferenced symbols that start with `mysql_` or that the `mysqlclient` library can't be found, this means that your system can't find the share `'libmysqlclient.so'` library.

The fix for this is to tell your system to search after shared libraries where the library is located by one of the following methods:

- Add the path to the directory where you have ‘libmysqlclient.so’ the LD_LIBRARY_PATH environment variable.
- Add the path to the directory where you have ‘libmysqlclient.so’ the LD_LIBRARY environment variable.
- Copy ‘libmysqlclient.so’ to some place that is searched by your system, like ‘/lib’, and update the shared library information by executing ldconfig.

Another way to solve this problem is to link your program statically, with `-static`, or by removing the dynamic MySQL libraries before linking your code. In the second case you should be sure that no other programs are using the dynamic libraries!

A.3.2 How to Run MySQL As a Normal User

The MySQL server `mysqld` can be started and run by any user. In order to change `mysqld` to run as a Unix user `user_name`, you must do the following:

1. Stop the server if it’s running (use `mysqladmin shutdown`).
2. Change the database directories and files so that `user_name` has privileges to read and write files in them (you may need to do this as the Unix root user):

```
shell> chown -R user_name /path/to/mysql/datadir
```

If directories or files within the MySQL data directory are symlinks, you’ll also need to follow those links and change the directories and files they point to. `chown -R` may not follow symlinks for you.

3. Start the server as user `user_name`, or, if you are using MySQL Version 3.22 or later, start `mysqld` as the Unix root user and use the `--user=user_name` option. `mysqld` will switch to run as the Unix user `user_name` before accepting any connections.
4. To start the server as the given user name automatically at system startup time, add a `user` line that specifies the user name to the `[mysqld]` group of the ‘`/etc/my.cnf`’ option file or the ‘`my.cnf`’ option file in the server’s data directory. For example:

```
[mysqld]
user=user_name
```

At this point, your `mysqld` process should be running fine and dandy as the Unix user `user_name`. One thing hasn’t changed, though: the contents of the permissions tables. By default (right after running the permissions table install script `mysql_install_db`), the MySQL user `root` is the only user with permission to access the `mysql` database or to create or drop databases. Unless you have changed those permissions, they still hold. This shouldn’t stop you from accessing MySQL as the MySQL `root` user when you’re logged in as a Unix user other than `root`; just specify the `-u root` option to the client program.

Note that accessing MySQL as `root`, by supplying `-u root` on the command-line, has **nothing** to do with MySQL running as the Unix `root` user, or, indeed, as another Unix user. The access permissions and user names of MySQL are completely separate from Unix user names. The only connection with Unix user names is that if you don’t provide a `-u` option when you invoke a client program, the client will try to connect using your Unix login name as your MySQL user name.

If your Unix box itself isn’t secured, you should probably at least put a password on the MySQL `root` users in the access tables. Otherwise, any user with an account on that machine can run `mysql -u root db_name` and do whatever he likes.

A.3.3 Problems with File Permissions

If you have problems with file permissions, for example, if `mysql` issues the following error message when you create a table:

```
ERROR: Can't find file: 'path/with/filename.frm' (Errcode: 13)
```

then the environment variable `UMASK` might be set incorrectly when `mysqld` starts up. The default `umask` value is `0660`. You can change this behaviour by starting `safe_mysqld` as follows:

```
shell> UMASK=384 # = 600 in octal
shell> export UMASK
shell> /path/to/safe_mysqld &
```

By default MySQL will create database and RAID directories with permission type `0700`. You can modify this behaviour by setting the `UMASK_DIR` variable. If you set this, new directories are created with the combined `UMASK` and `UMASK_DIR`. For example, if you want to give group access to all new directories, you can do:

```
shell> UMASK_DIR=504 # = 770 in octal
shell> export UMASK_DIR
shell> /path/to/safe_mysqld &
```

In MySQL Version 3.23.25 and above, MySQL assumes that the value for `UMASK` and `UMASK_DIR` is in octal if it starts with a zero.

See [\[Environment variables\]](#), page [\[undefined\]](#).

A.4 Administration Related Issues

A.4.1 What To Do If MySQL Keeps Crashing

All MySQL versions are tested on many platforms before they are released. This doesn't mean that there aren't any bugs in MySQL, but it means if there are bugs, they are very few and can be hard to find. If you have a problem, it will always help if you try to find out exactly what crashes your system, as you will have a much better chance of getting this fixed quickly.

First, you should try to find out whether the problem is that the `mysqld` daemon dies or whether your problem has to do with your client. You can check how long your `mysqld` server has been up by executing `mysqladmin version`. If `mysqld` has died, you may find the reason for this in the file `'mysql-data-directory/'hostname'.err'`. See [\[Error log\]](#), page [\[undefined\]](#).

Many crashes of MySQL are caused by corrupted index / data files. MySQL will update the data on disk, with the `write()` system call, after every SQL statement and before the client is notified about the result. (This is not true if you are running with `delay_key_write`, in which case only the data is written.) This means that the data is safe even if `mysqld` crashes, as the OS will ensure that the not flushed data is written to disk. You can force MySQL to sync everything to disk after every SQL command by starting `mysqld` with `--flush`.

The above means that normally you shouldn't get corrupted tables unless:

- Someone/something killed `mysqld` or the machine in the middle of an update.
- You have found a bug in `mysqld` that caused it to die in the middle of an update.
- Someone is manipulating the data/index files outside of `mysqld` without locking the table properly.
- If you are running many `mysqld` servers on the same data on a system that doesn't support good filesystem locks (normally handled by the `lockd` daemon) or if you are running multiple servers with `--skip-external-locking`
- You have a crashed index/datafile that contains very wrong data that got `mysqld` confused.
- You have found a bug in the data storage code. This isn't that likely, but it's at least possible. In this case you can try to change the file type to another storage engine by using `ALTER TABLE` on a repaired copy of the table!

Because it is very difficult to know why something is crashing, first try to check whether things that work for others crash for you. Please try the following things:

Take down the `mysqld` daemon with `mysqldadmin shutdown`, run `myisamchk --silent --force */*.MYI` on all tables, and restart the `mysqld` daemon. This will ensure that you are running from a clean state. See [\[MySQL Database Administration\]](#), page [\[undefined\]](#).

- Use `mysqld --log` and try to determine from the information in the log whether some specific query kills the server. About 95% of all bugs are related to a particular query! Normally this is one of the last queries in the log file just before MySQL restarted. See [\[Query log\]](#), page [\[undefined\]](#). If you can repeatedly kill MySQL with one of the queries, even when you have checked all tables just before doing the query, then you have been able to locate the bug and should do a bug report for this! See [\[Bug reports\]](#), page [\[undefined\]](#).
- Try to make a test case that we can use to reproduce the problem. See [\[Reproducible test case\]](#), page [\[undefined\]](#).
- Try running the included `mysql-test` test and the MySQL benchmarks. See [\[MySQL test suite\]](#), page [\[undefined\]](#). They should test MySQL rather well. You can also add code to the benchmarks that simulates your application! The benchmarks can be found in the 'bench' directory in the source distribution or, for a binary distribution, in the 'sql-bench' directory under your MySQL installation directory.
- Try `fork_test.pl` and `fork2_test.pl`.
- If you configure MySQL for debugging, it will be much easier to gather information about possible errors if something goes wrong. Reconfigure MySQL with the `--with-debug` option or `--with-debug=full` to `configure` and then recompile. See [\[Debugging server\]](#), page [\[undefined\]](#).
- Configuring MySQL for debugging causes a safe memory allocator to be included that can find some errors. It also provides a lot of output about what is happening.
- Have you applied the latest patches for your operating system?
- Use the `--skip-external-locking` option to `mysqld`. On some systems, the `lockd` lock manager does not work properly; the `--skip-external-locking` option tells `mysqld` not to use external locking. (This means that you cannot run 2 `mysqld` servers

on the same data and that you must be careful if you use `myisamchk`, but it may be instructive to try the option as a test.)

- Have you tried `mysqladmin -u root processlist` when `mysqld` appears to be running but not responding? Sometimes `mysqld` is not comatose even though you might think so. The problem may be that all connections are in use, or there may be some internal lock problem. `mysqladmin processlist` will usually be able to make a connection even in these cases, and can provide useful information about the current number of connections and their status.
- Run the command `mysqladmin -i 5 status` or `mysqladmin -i 5 -r status` or in a separate window to produce statistics while you run your other queries.
- Try the following:
 1. Start `mysqld` from `gdb` (or in another debugger). See [Using gdb on mysqld](#), page [undefined](#).
 2. Run your test scripts.
 3. Print the backtrace and the local variables at the 3 lowest levels. In `gdb` you can do this with the following commands when `mysqld` has crashed inside `gdb`:

```
backtrace
info local
up
info local
up
info local
```

With `gdb` you can also examine which threads exist with `info threads` and switch to a specific thread with `thread #`, where `#` is the thread id.

- Try to simulate your application with a Perl script to force MySQL to crash or misbehave.
- Send a normal bug report. See [Bug reports](#), page [undefined](#). Be even more detailed than usual. Because MySQL works for many people, it may be that the crash results from something that exists only on your computer (for example, an error that is related to your particular system libraries).
- If you have a problem with tables with dynamic-length rows and you are not using BLOB/TEXT columns (but only VARCHAR columns), you can try to change all VARCHAR to CHAR with ALTER TABLE. This will force MySQL to use fixed-size rows. Fixed-size rows take a little extra space, but are much more tolerant to corruption!

The current dynamic row code has been in use at MySQL AB for at least 3 years without any problems, but by nature dynamic-length rows are more prone to errors, so it may be a good idea to try the above to see if it helps!

A.4.2 How to Reset a Forgotten Root Password

If you never set a `root` password for MySQL, then the server will not require a password at all for connecting as `root`. It is recommended to always set a password for each user. See [Security](#), page [undefined](#).

If you have set a `root` password, but forgot what it was, you can set a new password with the following procedure:

1. Take down the `mysqld` server by sending a `kill` (not `kill -9`) to the `mysqld` server. The pid is stored in a `.pid` file, which is normally in the MySQL database directory:

```
shell> kill `cat /mysql-data-directory/hostname.pid`
```

You must be either the Unix `root` user or the same user `mysqld` runs as to do this.
2. Restart `mysqld` with the `--skip-grant-tables` option.
3. Set a new password with the `mysqladmin password` command:

```
shell> mysqladmin -u root password 'mynewpassword'
```
4. Now you can either stop `mysqld` and restart it normally, or just load the privilege tables with:

```
shell> mysqladmin -h hostname flush-privileges
```
5. After this, you should be able to connect using the new password.

Alternatively, you can set the new password using the `mysql` client:

1. Take down and restart `mysqld` with the `--skip-grant-tables` option as described above.
2. Connect to the `mysqld` server with:

```
shell> mysql -u root mysql
```
3. Issue the following commands in the `mysql` client:

```
mysql> UPDATE user SET Password=PASSWORD('mynewpassword')
-> WHERE User='root';
mysql> FLUSH PRIVILEGES;
```
4. After this, you should be able to connect using the new password.
5. You can now stop `mysqld` and restart it normally.

A.4.3 How MySQL Handles a Full Disk

When a disk-full condition occurs, MySQL does the following:

- It checks once every minute to see whether there is enough space to write the current row. If there is enough space, it continues as if nothing had happened.
- Every 6 minutes it writes an entry to the log file warning about the disk full condition.

To alleviate the problem, you can take the following actions:

- To continue, you only have to free enough disk space to insert all records.
- To abort the thread, you must send a `mysqladmin kill` to the thread. The thread will be aborted the next time it checks the disk (in 1 minute).
- Note that other threads may be waiting for the table that caused the disk full condition. If you have several “locked” threads, killing the one thread that is waiting on the disk-full condition will allow the other threads to continue.

Exceptions to the above behaviour is when you use `REPAIR` or `OPTIMIZE` or when the indexes are created in a batch after an `LOAD DATA INFILE` or after an `ALTER TABLE` statement.

All of the above commands may use big temporary files that left to themselves would cause big problems for the rest of the system. If MySQL gets disk full while doing any of the above operations, it will remove the big temporary files and mark the table as crashed (except for `ALTER TABLE`, in which the old table will be left unchanged).

A.4.4 Where MySQL Stores Temporary Files

MySQL uses the value of the `TMPDIR` environment variable as the pathname of the directory in which to store temporary files. If you don't have `TMPDIR` set, MySQL uses the system default, which is normally `/tmp` or `/usr/tmp`. If the filesystem containing your temporary file directory is too small, you should edit `safe_mysqld` to set `TMPDIR` to point to a directory in a filesystem where you have enough space! You can also set the temporary directory using the `--tmpdir` option to `mysqld`.

MySQL creates all temporary files as hidden files. This ensures that the temporary files will be removed if `mysqld` is terminated. The disadvantage of using hidden files is that you will not see a big temporary file that fills up the filesystem in which the temporary file directory is located.

When sorting (`ORDER BY` or `GROUP BY`), MySQL normally uses one or two temporary files. The maximum disk-space needed is:

```
(length of what is sorted + sizeof(database pointer))
* number of matched rows
* 2
```

`sizeof(database pointer)` is usually 4, but may grow in the future for really big tables.

For some `SELECT` queries, MySQL also creates temporary SQL tables. These are not hidden and have names of the form `'SQL_*'`.

`ALTER TABLE` creates a temporary table in the same directory as the original table.

If you use MySQL 4.1 or later you can spread load between several physical disks by setting `--tmpdir` to a list of paths separated by colon `:` (semicolon `;` on Windows). They will be used in round-robin fashion. **Note:** These paths should end up on different **physical** disks, not different partitions of the same disk.

A.4.5 How to Protect or Change the MySQL Socket File

`'/tmp/mysql.sock'`

If you have problems with the fact that anyone can delete the MySQL communication socket `'/tmp/mysql.sock'`, you can, on most versions of Unix, protect your `'/tmp'` filesystem by setting the `sticky` bit on it. Log in as `root` and do the following:

```
shell> chmod +t /tmp
```

This will protect your `'/tmp'` filesystem so that files can be deleted only by their owners or the superuser (`root`).

You can check if the `sticky` bit is set by executing `ls -ld /tmp`. If the last permission bit is `t`, the bit is set.

You can change the place where MySQL uses `/` puts the socket file the following ways:

- Specify the path in a global or local option file. For example, put in `/etc/my.cnf`:

```
[client]
socket=path-for-socket-file
```

```
[mysqld]
socket=path-for-socket-file
```

See [\(undefined\) \[Option files\]](#), page [\(undefined\)](#).

- Specifying this on the command-line to `safe_mysqld` and most clients with the `--socket=path-for-socket-file` option.
- Specify the path to the socket in the `MYSQL_UNIX_PORT` environment variable.
- Defining the path with the `configure` option `--with-unix-socket-path=path-for-socket-file`. See [\[configure options\]](#), page [\[undefined\]](#).

You can test that the socket works with this command:

```
shell> mysqladmin --socket=/path/to/socket version
```

A.4.6 Time Zone Problems

If you have a problem with `SELECT NOW()` returning values in GMT and not your local time, you have to set the `TZ` environment variable to your current time zone. This should be done for the environment in which the server runs, for example, in `safe_mysqld` or `mysql.server`. See [\[Environment variables\]](#), page [\[undefined\]](#).

A.5 Query Related Issues

A.5.1 Case-Sensitivity in Searches

By default, MySQL searches are case-insensitive (although there are some character sets that are never case-insensitive, such as `czech`). That means that if you search with `nome_coluna LIKE 'a%'`, you will get all column values that start with A or a. If you want to make this search case-sensitive, use something like `INSTR(nome_coluna, "A")=1` to check a prefix. Or use `STRCMP(nome_coluna, "A") = 0` if the column value must be exactly "A". Simple comparison operations (`>=`, `>`, `=`, `<`, `<=`, sorting and grouping) are based on each character's "sort value". Characters with the same sort value (like E, e and é) are treated as the same character!

In older MySQL versions `LIKE` comparisons were done on the uppercase value of each character (E == e but E <> é). In newer MySQL versions `LIKE` works just like the other comparison operators.

If you want a column always to be treated in case-sensitive fashion, declare it as `BINARY`. See [\[CREATE TABLE\]](#), page [\[undefined\]](#).

If you are using Chinese data in the so-called big5 encoding, you want to make all character columns `BINARY`. This works because the sorting order of big5 encoding characters is based on the order of ASCII codes.

A.5.2 Problems Using DATE Columns

The format of a `DATE` value is `'YYYY-MM-DD'`. According to standard SQL, no other format is allowed. You should use this format in `UPDATE` expressions and in the `WHERE` clause of `SELECT` statements. For example:

```
mysql> SELECT * FROM nome_tabela WHERE date >= '1997-05-05';
```

As a convenience, MySQL automatically converts a date to a number if the date is used in a numeric context (and vice versa). It is also smart enough to allow a "relaxed" string

form when updating and in a `WHERE` clause that compares a date to a `TIMESTAMP`, `DATE`, or a `DATETIME` column. (Relaxed form means that any punctuation character may be used as the separator between parts. For example, `'1998-08-15'` and `'1998#08#15'` are equivalent.) MySQL can also convert a string containing no separators (such as `'19980815'`), provided it makes sense as a date.

The special date `'0000-00-00'` can be stored and retrieved as `'0000-00-00'`. When using a `'0000-00-00'` date through `MyODBC`, it will automatically be converted to `NULL` in `MyODBC` Version 2.50.12 and above, because `ODBC` can't handle this kind of date.

Because MySQL performs the conversions described above, the following statements work:

```
mysql> INSERT INTO nome_tabela (idate) VALUES (19970505);
mysql> INSERT INTO nome_tabela (idate) VALUES ('19970505');
mysql> INSERT INTO nome_tabela (idate) VALUES ('97-05-05');
mysql> INSERT INTO nome_tabela (idate) VALUES ('1997.05.05');
mysql> INSERT INTO nome_tabela (idate) VALUES ('1997 05 05');
mysql> INSERT INTO nome_tabela (idate) VALUES ('0000-00-00');

mysql> SELECT idate FROM nome_tabela WHERE idate >= '1997-05-05';
mysql> SELECT idate FROM nome_tabela WHERE idate >= 19970505;
mysql> SELECT MOD(idate,100) FROM nome_tabela WHERE idate >= 19970505;
mysql> SELECT idate FROM nome_tabela WHERE idate >= '19970505';
```

However, the following will not work:

```
mysql> SELECT idate FROM nome_tabela WHERE STRCMP(idate,'19970505')=0;
```

`STRCMP()` is a string function, so it converts `idate` to a string and performs a string comparison. It does not convert `'19970505'` to a date and perform a date comparison.

Note that MySQL does very limited checking whether the date is correct. If you store an incorrect date, such as `'1998-2-31'`, the wrong date will be stored.

Because MySQL packs dates for storage, it can't store any given date as it would not fit onto the result buffer. The rules for accepting a date are:

- If MySQL can store and retrieve a given date, the wrong date is accepted for `DATE` and `DATETIME` columns.
- All days values between 0-31 are accepted for any date. This makes it very convenient for web applications where you ask year, month and day in 3 different fields.
- The day or month field may be zero. This is convenient if you want to store a birthdate in a `DATE` column and you only know part of the date.

If the date cannot be converted to any reasonable value, a 0 is stored in the `DATE` field, which will be retrieved as `0000-00-00`. This is both a speed and convenience issue as we believe that the database's responsibility is to retrieve the same date you stored (even if the data was not logically correct in all cases). We think it is up to the application to check the dates, and not the server.

A.5.3 Problems with NULL Values

The concept of the `NULL` value is a common source of confusion for newcomers to `SQL`, who often think that `NULL` is the same thing as an empty string `""`. This is not the case! For example, the following statements are completely different:

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ("");
```

Both statements insert a value into the `phone` column, but the first inserts a `NULL` value and the second inserts an empty string. The meaning of the first can be regarded as “phone number is not known” and the meaning of the second can be regarded as “she has no phone”.

In SQL, the `NULL` value is always false in comparison to any other value, even `NULL`. An expression that contains `NULL` always produces a `NULL` value unless otherwise indicated in the documentation for the operators and functions involved in the expression. All columns in the following example return `NULL`:

```
mysql> SELECT NULL,1+NULL,CONCAT('Invisible',NULL);
```

If you want to search for column values that are `NULL`, you cannot use the `=NULL` test. The following statement returns no rows, because `expr = NULL` is `FALSE`, for any expression:

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

To look for `NULL` values, you must use the `IS NULL` test. The following shows how to find the `NULL` phone number and the empty phone number:

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
mysql> SELECT * FROM my_table WHERE phone = "";
```

Note that you can only add an index on a column that can have `NULL` values if you are using MySQL Version 3.23.2 or newer and are using the `MyISAM` or `InnoDB` table type. In earlier versions and with other table types, you must declare such columns `NOT NULL`. This also means you cannot then insert `NULL` into an indexed column.

When reading data with `LOAD DATA INFILE`, empty columns are updated with `''`. If you want a `NULL` value in a column, you should use `\N` in the text file. The literal word `'NULL'` may also be used under some circumstances. See [\[LOAD DATA\]](#), page [\[LOAD DATA\]](#).

When using `ORDER BY`, `NULL` values are presented first. In versions prior to 4.0.2, if you sort in descending order using `DESC`, `NULL` values are presented last. When using `GROUP BY`, all `NULL` values are regarded as equal.

To help with `NULL` handling, you can use the `IS NULL` and `IS NOT NULL` operators and the `IFNULL()` function.

For some column types, `NULL` values are handled specially. If you insert `NULL` into the first `TIMESTAMP` column of a table, the current date and time is inserted. If you insert `NULL` into an `AUTO_INCREMENT` column, the next number in the sequence is inserted.

A.5.4 Problems with alias

You can use an alias to refer to a column in the `GROUP BY`, `ORDER BY`, or in the `HAVING` part. Aliases can also be used to give columns better names:

```
SELECT SQRT(a*b) as rt FROM table_name GROUP BY rt HAVING rt > 0;
SELECT id,COUNT(*) AS cnt FROM table_name GROUP BY id HAVING cnt > 0;
SELECT id AS "Customer identity" FROM table_name;
```

Note that standard SQL doesn't allow you to refer to an alias in a `WHERE` clause. This is because when the `WHERE` code is executed the column value may not yet be determined. For example, the following query is **illegal**:

```
SELECT id,COUNT(*) AS cnt FROM table_name WHERE cnt > 0 GROUP BY id;
```

The `WHERE` statement is executed to determine which rows should be included in the `GROUP BY` part while `HAVING` is used to decide which rows from the result set should be used.

A.5.5 Deleting Rows from Related Tables

As MySQL doesn't support subqueries (prior to Version 4.1), nor the use of more than one table in the `DELETE` statement (prior to Version 4.0), you should use the following approach to delete rows from 2 related tables:

1. `SELECT` the rows based on some `WHERE` condition in the main table.
2. `DELETE` the rows in the main table based on the same condition.
3. `DELETE FROM related_table WHERE related_column IN (selected_rows)`.

If the total number of characters in the query with `related_column` is more than 1,048,576 (the default value of `max_allowed_packet`, you should split it into smaller parts and execute multiple `DELETE` statements. You will probably get the fastest `DELETE` by only deleting 100-1000 `related_column` ids per query if the `related_column` is an index. If the `related_column` isn't an index, the speed is independent of the number of arguments in the `IN` clause.

A.5.6 Solving Problems with No Matching Rows

If you have a complicated query that has many tables and that doesn't return any rows, you should use the following procedure to find out what is wrong with your query:

1. Test the query with `EXPLAIN` and check if you can find something that is obviously wrong. See [\(undefined\) \[EXPLAIN\], page \(undefined\)](#).
2. Select only those fields that are used in the `WHERE` clause.
3. Remove one table at a time from the query until it returns some rows. If the tables are big, it's a good idea to use `LIMIT 10` with the query.
4. Do a `SELECT` for the column that should have matched a row against the table that was last removed from the query.
5. If you are comparing `FLOAT` or `DOUBLE` columns with numbers that have decimals, you can't use `'='`. This problem is common in most computer languages because floating-point values are not exact values. In most cases, changing the `FLOAT` to a `DOUBLE` will fix this. See [\(undefined\) \[Problems with float\], page \(undefined\)](#).
6. If you still can't figure out what's wrong, create a minimal test that can be run with `mysql test < query.sql` that shows your problems. You can create a test file with `mysqldump --quick database tables > query.sql`. Open the file in an editor, remove some insert lines (if there are too many of these), and add your select statement at the end of the file.

Test that you still have your problem by doing:

```
shell> mysqladmin create test2
shell> mysql test2 < query.sql
```

Post the test file using `mysqlbug` to mysql@lists.mysql.com.

A.5.7 Problems with Floating-Point Comparison

floating-point numbers cause confusion sometimes, because these numbers are not stored as exact values inside computer architecture. What one can see on the screen usually is not the exact value of the number.

Field types FLOAT, DOUBLE and DECIMAL are such.

```
CREATE TABLE t1 (i INT, d1 DECIMAL(9,2), d2 DECIMAL(9,2));
INSERT INTO t1 VALUES (1, 101.40, 21.40), (1, -80.00, 0.00),
(2, 0.00, 0.00), (2, -13.20, 0.00), (2, 59.60, 46.40),
(2, 30.40, 30.40), (3, 37.00, 7.40), (3, -29.60, 0.00),
(4, 60.00, 15.40), (4, -10.60, 0.00), (4, -34.00, 0.00),
(5, 33.00, 0.00), (5, -25.80, 0.00), (5, 0.00, 7.20),
(6, 0.00, 0.00), (6, -51.40, 0.00);
```

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b
-> FROM t1 GROUP BY i HAVING a <> b;
```

i	a	b
1	21.40	21.40
2	76.80	76.80
3	7.40	7.40
4	15.40	15.40
5	7.20	7.20
6	-51.40	0.00

The result is correct. Although the first five records look like they shouldn't pass the comparison test, they may do so because the difference between the numbers show up around tenth decimal, or so depending on computer architecture.

The problem cannot be solved by using ROUND() (or similar function), because the result is still a floating-point number. Example:

```
mysql> SELECT i, ROUND(SUM(d1), 2) AS a, ROUND(SUM(d2), 2) AS b
-> FROM t1 GROUP BY i HAVING a <> b;
```

i	a	b
1	21.40	21.40
2	76.80	76.80
3	7.40	7.40
4	15.40	15.40
5	7.20	7.20
6	-51.40	0.00

This is what the numbers in column 'a' look like:

```
mysql> SELECT i, ROUND(SUM(d1), 2)*1.0000000000000000 AS a,
-> ROUND(SUM(d2), 2) AS b FROM t1 GROUP BY i HAVING a <> b;
```

i	a	b
1	21.3999999999999986	21.40
2	76.7999999999999972	76.80
3	7.4000000000000004	7.40
4	15.4000000000000004	15.40
5	7.2000000000000002	7.20
6	-51.399999999999986	0.00

Depending on the computer architecture you may or may not see similar results. Each CPU may evaluate floating-point numbers differently. For example in some machines you may get 'right' results by multiplying both arguments with 1, an example follows.

WARNING: NEVER TRUST THIS METHOD IN YOUR APPLICATION, THIS IS AN EXAMPLE OF A WRONG METHOD!!!

```
mysql> SELECT i, ROUND(SUM(d1), 2)*1 AS a, ROUND(SUM(d2), 2)*1 AS b
-> FROM t1 GROUP BY i HAVING a <> b;
```

i	a	b
6	-51.40	0.00

The reason why the above example seems to be working is that on the particular machine where the test was done, the CPU floating-point arithmetics happens to round the numbers to same, but there is no rule that any CPU should do so, so it cannot be trusted.

The correct way to do floating-point number comparison is to first decide on what is the wanted tolerance between the numbers and then do the comparison against the tolerance number. For example, if we agree on that floating-point numbers should be regarded the same, if they are same with precision of one of ten thousand (0.0001), the comparison should be done like this:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
-> GROUP BY i HAVING ABS(a - b) > 0.0001;
```

i	a	b
6	-51.40	0.00

1 row in set (0.00 sec)

And vice versa, if we wanted to get rows where the numbers are the same, the test would be:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
-> GROUP BY i HAVING ABS(a - b) < 0.0001;
```

i	a	b
1	21.40	21.40
2	76.80	76.80
3	7.40	7.40

```

|    4 | 15.40 | 15.40 |
|    5 |  7.20 |  7.20 |
+-----+-----+-----+

```

A.6 Table Definition Related Issues

A.6.1 Problems with ALTER TABLE.

ALTER TABLE changes a table to the current character set. If you get a duplicate key error during ALTER TABLE, then the cause is either that the new character sets maps two keys to the same value or that the table is corrupted, in which case you should run REPAIR TABLE on the table.

If ALTER TABLE dies with an error like this:

```
Error on rename of './database/name.frm' to './database/B-a.frm' (Errcode: 17)
```

the problem may be that MySQL has crashed in a previous ALTER TABLE and there is an old table named 'A-something' or 'B-something' lying around. In this case, go to the MySQL data directory and delete all files that have names starting with A- or B-. (You may want to move them elsewhere instead of deleting them.)

ALTER TABLE works the following way:

- Create a new table named 'A-xxx' with the requested changes.
- All rows from the old table are copied to 'A-xxx'.
- The old table is renamed 'B-xxx'.
- 'A-xxx' is renamed to your old table name.
- 'B-xxx' is deleted.

If something goes wrong with the renaming operation, MySQL tries to undo the changes. If something goes seriously wrong (this shouldn't happen, of course), MySQL may leave the old table as 'B-xxx', but a simple rename on the system level should get your data back.

A.6.2 How To Change the Order of Columns in a Table

The whole point of SQL is to abstract the application from the data storage format. You should always specify the order in which you wish to retrieve your data. For example:

```
SELECT nome_coluna1, nome_coluna2, nome_coluna3 FROM nome_tabela;
```

will return columns in the order nome_coluna1, nome_coluna2, nome_coluna3, whereas:

```
SELECT nome_coluna1, nome_coluna3, nome_coluna2 FROM nome_tabela;
```

will return columns in the order nome_coluna1, nome_coluna3, nome_coluna2.

If you want to change the order of columns anyway, you can do it as follows:

1. Create a new table with the columns in the right order.
2. Execute `INSERT INTO new_table SELECT fields-in-new-table-order FROM old_table.`
3. Drop or rename old_table.
4. ALTER TABLE new_table RENAME old_table.

You should **never**, in an application, use `SELECT *` and retrieve the columns based on their position, because the order and position in which columns are returned **cannot** may not remain the same (if you add/move/delete columns). A simple change to your database structure would then cause your application to fail. Of course `SELECT *` is quite suitable for testing queries.

A.6.3 TEMPORARY TABLE problems

The following are a list of the limitations with TEMPORARY TABLES.

- A temporary table can only be of type HEAP, ISAM, MyISAM, MERGE, or InnoDB.
- You can't use temporary tables more than once in the same query. For example, the following doesn't work.

```
mysql> SELECT * FROM temporary_table, temporary_table AS t2;
```

- You can't use RENAME on a TEMPORARY table. Note that ALTER TABLE org_name RENAME new_name works!

Appendix B Contributed Programs

Many users of MySQL have contributed *very* useful support tools and add-ons.

A list of some software available from the MySQL website (or any mirror) is shown here.

You can also visit our online listing of MySQL-related software at <http://www.mysql.com/portal/software>. The community facilities there also allow for your input!

If you want to build MySQL support for the Perl DBI/DBD interface, you should fetch the `Data-Dumper`, `DBI`, and `DBD-mysql` files and install them. See [\(undefined\)](#) [Perl support], page [\(undefined\)](#).

Note: The programs listed here can be freely downloaded and used. They are copyrighted by their respective owners. Please see individual product documentation for more details on licensing and terms. MySQL AB assumes no liability for the correctness of the information in this chapter or for the proper operation of the programs listed herein.

B.1 APIs

- Perl Modules
 - <http://www.mysql.com/Downloads/Contrib/KAMXbase1.2.tar.gz> Convert between ‘.dbf’ files and MySQL tables. Perl module written by Prata Pereira pereira@ee.eng.ohio-state.edu, extended by Kevin A. McGrail kmcgrail@digital1.peregrinehw.com. This converter can handle MEMO fields.
 - <http://www.mysql.com/Downloads/Contrib/HandySQL-1.1.tar.gz> HandySQL is a MySQL access module. It offers a C interface embedded in Perl and is approximately 20% faster than regular DBI.
- OLEDB
 - <http://www.mysql.com/Downloads/Win32/MyOLEDB3.exe> MyOLEDB 3.0 installation package from SWSOft.
 - <http://www.mysql.com/Downloads/Win32/mysql-oledb-3.0.0.zip> Source for MyOLEDB 3.0.
 - <http://www.mysql.com/Downloads/Win32/MySamples.zip> Examples and documentation for MyOLEDB.
 - <http://www.mysql.com/Downloads/Win32/MyOLEDB.chm> Help files for MyOLEDB.
 - <http://www.mysql.com/Downloads/Win32/libmyodbc.zip> Static MyODBC library used for build MyOLEDB. Based on MyODBC code.
- C++
 - <http://www.mysql.com/Downloads/Contrib/mysql-c++-0.02.tar.gz> MySQL C++ wrapper library. By Roland Haenel, rh@ginster.net.
 - <http://www.mysql.com/Downloads/Contrib/MyDAO.tar.gz> MySQL C++ API. By Satish spitfire@pn3.vsnl.net.in. Inspired by Roland Haenel’s C++ API and Ed Carp’s MyC library.

- <http://www.mysql.com/products/mysql++/> MySQL C++ API (more than just a wrapper library). Originally by kevina@clark.net. Now maintained by Sinisa at MySQL AB.
- <http://nelsonjr.homepage.com/NJrAPI/> A C++ database independent library that supports MySQL.
- Delphi
 - <http://www.mysql.com/Downloads/Contrib/DelphiMySQL2.zip> Delphi interface to `libmysql.dll`, by bsilva@umesd.k12.or.us.
 - <http://www.mysql.com/Downloads/Contrib/Udmysql.pas> A wrapper for `libmysql.dll` for usage in Delphi. By Reiner Sombrowsky.
 - <http://www.fichtner.net/delphi/mysql.delphi.phtml> A Delphi Interface to MySQL, with source code. By Matthias Fichtner.
 - <http://www.productivity.org/projects/tmysql/> Tmysql, a library to use MySQL with Delphi.
 - <https://sourceforge.net/projects/zeoslib/> Zeos Library is a set of delphi native datasets and database components for MySQL, PostgreSQL, Interbase, MS SQL, Oracle, DB/2. Also it includes development tools such as Database Explorer and Database Designer.
 - <http://www.mysql.com/Downloads/Contrib/Win32/SBMySQL50Share.exe> Delphi 5 Shareware MySQL Dataset Components.
- <http://www.mysql.com/Downloads/Contrib/mysql-ruby-2.2.0.tar.gz> MySQL Ruby module. By TOMITA Masahiro tommy@tmtm.org Ruby is an Object-Oriented Interpreter Language (<http://www.netlab.co.jp/ruby/>).
- <http://www.mysql.com/Downloads/Contrib/JdmMysqlDriver-0.1.0.tar.gz> A VisualWorks 3.0 Smalltalk driver for MySQL. By joshmiller@earthlink.net.
- <http://www.mysql.com/Downloads/Contrib/Db.py> Python module with caching. By gandalf@rosmail.com.
- <http://www.mysql.com/Downloads/Contrib/MySQLmodule-1.4.tar.gz> Python interface for MySQL. By Joseph Skinner joe@earthlight.co.nz. Modified by Joerg Senekowitsch senekow@ibm.net.
- <http://www.mysql.com/Downloads/Contrib/MySQL-python-0.3.0.tar.gz> MySQLdb Python is an DB-API v2.0-compliant interface to MySQL. Transactions are supported if the server and tables support them. It is thread-safe, and contains a compatibility module for older code written for the no-longer-maintained MySQLmodule interface.
- http://www.mysql.com/Downloads/Contrib/mysql_mex_12.tar.gz An interface program for the Matlab program by MathWorks. The interface is done by Kimmo Uutela and John Fisher (not by Mathworks). Check <http://boojum.hut.fi/~kuutela/mysqlmex.html> for more information.
- <http://www.mysql.com/Downloads/Contrib/mysqltcl-1.53.tar.gz> Tcl interface for MySQL. Based on 'mysqltcl-1.50.tar.gz'. For version 2.0 and more info, see <http://www.xdobry.de/mysqltcl/>.
- <http://www.mysql.com/Downloads/Contrib/MyC-0.1.tar.gz> A Visual Basic-like API, by Ed Carp.

- <http://www.mysql.com/Downloads/Contrib/Vdb-dflts-2.1.tar.gz> This is a new version of a set of library utilities intended to provide a generic interface to SQL database engines such that your application becomes a 3-tiered application. The advantage is that you can easily switch between and move to other database engines by implementing one file for the new backend without making any changes to your applications. By damian@cablenet.net.
- <http://www.mysql.com/Downloads/Contrib/DbFramework-1.10.tar.gz> DbFramework is a collection of classes for manipulating MySQL databases. The classes are loosely based on the CDIF Data Model Subject Area. By Paul Sharpe paul@miraclefish.com.
- <http://www.mysql.com/Downloads/Contrib/pike-mysql-1.4.tar.gz> MySQL module for pike. For use with the Roxen web server.
- <http://www.mysql.com/Downloads/Contrib/squile.tar.gz> Module for guile that allows guile to interact with SQL databases. By Hal Roberts.
- <http://www.mysql.com/Downloads/Contrib/stk-mysql.tar.gz> Interface for Stk. Stk is the Tk widgets with Scheme underneath instead of Tcl. By Terry Jones.
- <http://www.mysql.com/Downloads/Contrib/eiffel-wrapper-1.0.tar.gz> Eiffel wrapper by Michael Ravits.
- <http://www.mysql.com/Downloads/Contrib/SQLmy0.06.tgz> FlagShip Replaceable Database Driver (RDD) for MySQL. By Alejandro Fernandez Herrero. The Flagship RDD homepage is at <http://www.fship.com/rdds.html>.
- <http://www.mysql.com/Downloads/Contrib/mydsn-1.0.zip> Binary and source for mydsn.dll. mydsn should be used to build and remove the DSN registry file for the MyODBC driver in Coldfusion applications. By Miguel Angel Solórzano.
- http://www.mysql.com/Downloads/Contrib/MySQL-ADA95_API.zip An ADA95 interface to the MySQL API. By Francois Fabien.
- http://www.mysql.com/Downloads/Contrib/MyTool-DLL_for_VB_and_MySQL.zip A DLL with MySQL C API for Visual Basic. By Ken Menzel kenm@icarz.com.
- <http://www.mysql.com/Downloads/Contrib/MYSQLX.EXE> MySQL ActiveX Object for directly accessing your MySQL servers from IIS/ASP, VB, VC++ skipping the slower ODBC methods. Fully updatable, multi-threaded with full support for all MySQL fieldtypes (version 2001.1.1). By SciBit <http://www.scibit.com/>.
- <http://www.fastflow.it/mylua/> MyLUA home page; how to use the LUA language to write MySQL PROCEDURE that can be loaded runtime.
 - <http://www.mysql.com/Downloads/Contrib/lua-4.0.tar.gz> LUA 4.0
 - <http://www.mysql.com/Downloads/Contrib/mylua-3.23.32.1.tar.gz> Patch for MySQL 3.23.32 to use LUA 4.0. By Cristian Giussani.
- http://www.mysql.com/Downloads/Contrib/patched_myodbc.zip Patch (for Omniform 4.0 support) to the MyODBC driver. By Thomas Thaele tthaele@papenmeier.de

B.2 Converters

- <http://www.mysql.com/Downloads/Contrib/mssql2mysql.txt> Converter from

MS-SQL to MySQL. By Michael Kofler. The mssql2mysql home page is at <http://www.kofler.cc/mysql/mssql2mysql.html>.

- <http://www.mysql.com/Downloads/Contrib/dbf2mysql-1.14.tar.gz> Convert between ‘.dbf’ files and MySQL tables. By Maarten Boekhold (boekhold@cindy.et.tudelft.nl), William Volkman, and Michael Widenius. This converter includes rudimentary read-only support for MEMO fields.
- <http://www.mysql.com/Downloads/Contrib/dbf2mysql-1.13.tgz> Convert between ‘.dbf’ files and MySQL tables. By Maarten Boekhold, boekhold@cindy.et.tudelft.nl, and Michael Widenius. This converter can’t handle MEMO fields.
- <http://www.mysql.com/Downloads/Contrib/dbf2mysql.zip> Convert between Fox-Pro ‘.dbf’ files and MySQL tables on Windows. By Alexander Eltsyn, ae@nica.ru or ae@usa.net.
- <http://www.mysql.com/Downloads/Contrib/dbf2sql.zip> Short and simple prg that can help you transport your data from foxpro table into MySQL table. By Danko Josic.
- <http://www.mysql.com/Downloads/Contrib/dump2h-1.20.gz> Convert from mysqldump output to a C header file. By Harry Brueckner, brueckner@mail.respublica.de.
- <http://www.mysql.com/Downloads/Contrib/exportsql.txt> A script that is similar to `access_to_mysql.txt`, except that this one is fully configurable, has better type conversion (including detection of TIMESTAMP fields), provides warnings and suggestions while converting, quotes **all** special characters in text and binary data, and so on. It will also convert to mSQL v1 and v2, and is free of charge for anyone. See <http://www.cynergi.net/exportsql/> for the latest version. By Pedro Freire, support@cynergi.net. Note: Doesn’t work with Access2!
- http://www.mysql.com/Downloads/Contrib/access_to_mysql.txt Paste this function into an Access module of a database that has the tables you want to export. See also `exportsql`. By Brian Andrews. Note: Doesn’t work with Access2!
- <http://www.mysql.com/Downloads/Contrib/importsml.txt> A script that does the exact reverse of `exportsql.txt`. That is, it imports data from MySQL into an Access database via ODBC. This is very handy when combined with `exportsql`, because it lets you use Access for all DB design and administration, and synchronise with your actual MySQL server either way. Free of charge. See <http://www.netdive.com/freebies/importsml/> for any updates. Created by Laurent Bossavit of NetDIVE. **Note:** doesn’t work with Access2!
- <http://www.mysql.com/Downloads/Contrib/mdb2sql.bas> Converter from Access97 to MySQL by Moshe Gurvich.
- <http://www.mysql.com/Downloads/Contrib/mssql2mysqlWrapper-1.0.tgz> A C wrapper from mSQL to MySQL. By alfred@sb.net
- <http://www.mysql.com/Downloads/Contrib/sqlconv.pl> A simple script that can be used to copy fields from one MySQL table to another in bulk. Basically, you can run `mysqldump` and pipe it to the `sqlconv.pl` script. The script will parse through the `mysqldump` output and will rearrange the fields so they can be inserted into a new table. An example is when you want to create a new table for a different site you are working

on, but the table is just a bit different (that is - fields in different order, etc.). By Steve Shreeve.

- <http://www.mysql.com/Downloads/Contrib/oracledump> Perl program to convert Oracle databases to MySQL. Has same output format as mysqldump. By Johan Andersson.
- <http://www.mysql.com/Downloads/Contrib/excel2mysql> Perl program to import Excel spreadsheets into a MySQL database. By Stephen Hurd shurd@sk.sympatico.ca
- http://www.mysql.com/Downloads/Contrib/T2S_100.ZIP. Windows program to convert text files to MySQL databases. By Asaf Azulay.

B.3 Utilities

- http://worldcommunity.com/opensource/utilities/mysql_backup.html MySQL Backup is a backup script for MySQL. By Peter F. Brown.
- http://www.mysql.com/Downloads/Contrib/mysql_watchdog.pl Monitor the MySQL daemon for possible lockups. By Yermo Lamers, yml@yml.com.
- http://www.mysql.com/Downloads/Contrib/mysql_structure_dumper.tar.gz
- http://www.mysql.com/Downloads/Contrib/mysql_structure_dumper.tgz Prints the structure of every table in a database. By Thomas Wana.
- <http://www.mysql.com/Downloads/Contrib/mysqlsync>. A Perl script to keep remote copies of a MySQL database in sync with a central master copy. By Mark Jeftovic. markjr@easydns.com.
- <http://www.mysql.com/Downloads/Contrib/MySQLTutor-0.2.tar.gz>. MySQL-Tutor. A MySQL tutorial for beginners.
- <http://www.mysql.com/Downloads/Contrib/MySQLDB.zip>
- <http://www.mysql.com/Downloads/Contrib/MySQLDB-readme.html>. A COM library for MySQL by Alok Singh.
- http://www.mysql.com/Downloads/Contrib/mysql_replicate.pl Perl program that handles replication. By elble@icculus.nsg.nwu.edu
- <http://www.mysql.com/Downloads/Contrib/DBIx-TextIndex-0.02.tar.gz> Perl script that uses reverse indexing to handle text searching. By Daniel Koch.
- <http://www.mysql.com/Downloads/Contrib/dbcheck> Perl script that takes a backup of tables before running isamchk on them. By Elizabeth.
- <http://www.mysql.com/Downloads/Contrib/mybackup>.
- <http://www.mswanson.com/mybackup> (mybackup home page) Wrapper for mysqldump to backup all databases. By Marc Swanson.
- <http://www.mysql.com/Downloads/Contrib/mdu.pl.gz> Prints the storage usage of a MySQL database.

Appendix C Credits

This appendix lists the developers, contributors, and supporters that have helped to make MySQL what it is today.

C.1 Developers at MySQL AB

These are the developers that are or have been employed by MySQL AB to work on the MySQL database software, roughly in the order they started to work with us. Following each developer is a small list of the tasks that the developer is responsible for, or the accomplishments they have made. All developers are involved in support.

Michael (Monty) Widenius

- Lead developer and main author of the MySQL server (`mysqld`).
- New functions for the string library.
- Most of the `mysys` library.
- The `ISAM` and `MyISAM` libraries (B-tree index file handlers with index compression and different record formats).
- The `HEAP` library. A memory table system with our superior full dynamic hashing. In use since 1981 and published around 1984.
- The `replace` program (take a look at it, it's **COOL!**).
- `MyODBC`, the ODBC driver for Windows95.
- Fixing bugs in MIT-pthreads to get it to work for MySQL Server. And also `Unireg`, a curses-based application tool with many utilities.
- Porting of `mSQL` tools like `msqlperl`, `DBD/DBI`, and `DB2mysql`.
- Most of `crash-me` and the foundation for the MySQL benchmarks.

David Axmark

- Initial main writer of the **Reference Manual**, including enhancements to `texi2html`.
- Automatic web site updating from the manual.
- Initial `Autoconf`, `Automake`, and `Libtool` support.
- Licensing.
- Parts of all the text files. (Nowadays only the 'README' is left. The rest ended up in the manual.)
- Lots of testing of new features.
- Our in-house Free Software legal expert.
- Mailing list maintainer (who never has the time to do it right...).
- Our original portability code (more than 10 years old now). Nowadays only some parts of `mysys` are left.
- Someone for Monty to call in the middle of the night when he just got that new feature to work.
- Chief "Open Sourcerer" (MySQL community relations).

Jani Tolonen

- `mysqlimport`
- A lot of extensions to the command-line clients.
- `PROCEDURE ANALYSE()`

Sinisa Milivojevic

- Compression (with `zlib`) in the client/server protocol.
- Perfect hashing for the lexical analyser phase.
- Multi-row `INSERT`
- `mysqldump -e` option
- `LOAD DATA LOCAL INFILE`
- `SQL_CALC_FOUND_ROWS SELECT` option
- `--max-user-connections=...` option
- `net_read` and `net_write_timeout`
- `GRANT/REVOKE` and `SHOW GRANTS FOR`
- New client-server protocol for 4.0
- `UNION` in 4.0
- Multi-table `DELETE/UPDATE`
- Derived tables in 4.1
- User resources management
- Initial developer of the MySQL++ C++ API and the MySQLGUI client.

Tonu Samuel (past developer)

- VIO interface (the foundation for the encrypted client/server protocol).
- MySQL Filesystem (a way to use MySQL databases as files and directories).
- The `CASE` expression.
- The `MD5()` and `COALESCE()` functions.
- RAID support for MyISAM tables.

Sasha Pachev

- Initial implementation of replication (up to version 4.0).
- `SHOW CREATE TABLE`.
- `mysql-bench`

Matt Wagner

- MySQL test suite.
- Webmaster (until 2002).
- Coordinator of development.

Miguel Solorzano

- Win32 development and release builds.
- Windows NT server code.
- WinMySQLAdmin

Timothy Smith (past developer)

- Dynamic character sets support.

- configure, RPMs and other parts of the build system.
- Initial developer of `libmysqld`, the embedded server.

Sergei Golubchik

- Full-text search.
- Added keys to the MERGE library.

Jeremy Cole

- Proofreading and editing this fine manual.
- `ALTER TABLE ... ORDER BY ...`
- `UPDATE ... ORDER BY ...`
- `DELETE ... ORDER BY ...`

Indrek Siitan

- Designing/programming of our web interface.
- Author of our newsletter management system.

Jorge del Conde

- MySQLCC (MySQL Control Center)
- Win32 development
- Initial implementation of the website portals.

Venu Anuganti

- Connector/ODBC (MyODBC) 3.51
- New client/server protocol for 4.1 (for prepared statements).
-

Arjen Lentz

- Maintainer of the MySQL Reference Manual.
- Preparing the O'Reilly printed edition of the manual.

Alexander (Bar) Barkov, Alexey (Holyfoot) Botchkov, and Ramil Kalimullin

- Spatial data (GIS) and R-Trees implementation for 4.1
- Unicode and character sets for 4.1

Oleksandr (Sanja) Byelkin

- Query cache in 4.0
- Implementation of subqueries (4.1).

Aleksey (Walrus) Kishkin and Alexey (Ranger) Stroganov

- Benchmarks design and analysis.
- Maintenance of the MySQL test suite.

Zak Greant

- Open Source advocate, MySQL community relations.

Carsten Pedersen

- The MySQL Certification program.

Lenz Grimmer

- Production (build and release) engineering.

Peter Zaitsev

- `SHA1()`, `AES_ENCRYPT()` and `AES_DECRYPT()` functions.

- Debugging, cleaning up various features.

Alexander (Salle) Keremidarski

- Support.
- Debugging.

Per-Erik Martin

- Lead developer for stored procedures (5.0) and triggers.

Jim Winstead

- Lead web developer.

Mark Matthews

- Connector/J driver (Java).

Peter Gulutzan

SQL-99, SQL:2003 standards compliance.

- Documentation of existing MySQL code/algorithms.

C.2 Contributors to MySQL

While MySQL AB owns all copyrights in the MySQL `server` and the MySQL `manual`, we wish to recognise those who have made contributions of one kind or another to the MySQL `distribution`. Contributors are listed here, in somewhat random order:

Paul DuBois

Ongoing help with making this manual correct and understandable. That includes rewriting Monty's and David's attempts at English into English as other people know it.

Gianmassimo Vigazzola qwerg@mbbox.vol.it or qwerg@tin.it

The initial port to Win32/NT.

Kim Aldale

Helped to rewrite Monty's and David's early attempts at English into English.

Per Eric Olsson

For more or less constructive criticism and real testing of the dynamic record format.

Irena Pancirov irena@mail.yacc.it

Win32 port with Borland compiler. `mysqlshutdown.exe` and `mysqlwatch.exe`

David J. Hughes

For the effort to make a shareware SQL database. At TcX, the predecessor of MySQL AB, we started with `mSQL`, but found that it couldn't satisfy our purposes so instead we wrote a SQL interface to our application builder `Unireg`. `mysqladmin` and `mysql` client are programs that were largely influenced by their `mSQL` counterparts. We have put a lot of effort into making the MySQL syntax a superset of `mSQL`. Many of the API's ideas are borrowed from `mSQL` to make it easy to port free `mSQL` programs to the MySQL API. The MySQL software doesn't contain any code from `mSQL`. Two files in the distribution (`client/insert_test.c` and `client/select_test.c`) are based

on the corresponding (non-copyrighted) files in the `mSQL` distribution, but are modified as examples showing the changes necessary to convert code from `mSQL` to MySQL Server. (`mSQL` is copyrighted David J. Hughes.)

Fred Fish For his excellent C debugging and trace library. Monty has made a number of smaller improvements to the library (speed and additional options).

Richard A. O'Keefe
For his public domain string library.

Henry Spencer
For his regex library, used in `WHERE column REGEXP regexp`.

Free Software Foundation
From whom we got an excellent compiler (`gcc`), the `libc` library (from which we have borrowed '`strto.c`' to get some code working in Linux), and the `readline` library (for the `mysql` client).

Free Software Foundation & The XEmacs development team
For a really great editor/environment used by almost everybody at MySQL AB/TcX/detron.

Patrick Lynch
For helping us acquire <http://www.mysql.com/>.

Fred Lindberg
For setting up gmail to handle the MySQL mailing list and for the incredible help we got in managing the MySQL mailing lists.

Igor Romanenko igor@frog.kiev.ua
`mysqldump` (previously `msqldump`, but ported and enhanced by Monty).

Yuri Dario
For keeping up and extending the MySQL OS/2 port.

Tim Bunce, Alligator Descartes
For the DBD (Perl) interface.

Tim Bunce
Author of `mysqlhotcopy`.

Andreas Koenig a.koenig@mind.de
For the Perl interface for MySQL Server.

Eugene Chan eugene@acenet.com.sg
For porting PHP for MySQL Server.

Michael J. Miller Jr. mke@terrapin.turbolift.com
For the first MySQL manual. And a lot of spelling/language fixes for the FAQ (that turned into the MySQL manual a long time ago).

Yan Cailin
First translator of the MySQL Reference Manual into simplified Chinese in early 2000 on which the Big5 and HK coded (<http://mysql.hitstar.com/>) versions were based. [Personal home page at linuxdb.yeah.net](http://linuxdb.yeah.net).

Giovanni Maruzzelli maruzz@matrice.it

For porting iODBC (Unix ODBC).

Chris Provenzano

Portable user level pthreads. From the copyright: This product includes software developed by Chris Provenzano, the University of California, Berkeley, and contributors. We are currently using version 1.60_beta6 patched by Monty (see 'mit-pthreads/Changes-mysql').

Xavier Leroy Xavier.Leroy@inria.fr

The author of LinuxThreads (used by the MySQL Server on Linux).

Zarko Mocnik zarko.mocnik@dem.si

Sorting for Slovenian language and the 'cset.tar.gz' module that makes it easier to add other character sets.

"TAMITO" tommy@valley.ne.jp

The `_MB` character set macros and the `ujis` and `sjis` character sets.

Joshua Chamas joshua@chamas.com

Base for concurrent insert, extended date syntax, debugging on NT, and answering on the MySQL mailing list.

Yves Carlier Yves.Carlier@rug.ac.be

`mysqlaccess`, a program to show the access rights for a user.

Rhys Jones rhys@wales.com (And GWE Technologies Limited)

For JDBC, a module to extract data from a MySQL Database with a Java client.

Dr Xiaokun Kelvin ZHU X.Zhu@brad.ac.uk

Further development of the JDBC driver and other MySQL-related Java tools.

James Cooper pixel@organic.com

For setting up a searchable mailing list archive at his site.

Rick Mehalick Rick_Mehalick@i-o.com

For `xmysql`, a graphical X client for MySQL Server.

Doug Sisk sisk@wix.com

For providing RPM packages of MySQL for Red Hat Linux.

Diemand Alexander V. axeld@vial.ethz.ch

For providing RPM packages of MySQL for Red Hat Linux-Alpha.

Antoni Pamies Olive toni@readysoft.es

For providing RPM versions of a lot of MySQL clients for Intel and SPARC.

Jay Bloodworth jay@pathways.sde.state.sc.us

For providing RPM versions for MySQL Version 3.21.

Jochen Wiedmann wiedmann@neckar-alb.de

For maintaining the Perl DBD::mysql module.

Therrien Gilbert gilbert@ican.net, Jean-Marc Pouyot jmp@scaltaire.fr

French error messages.

- Petr Snajdr, snajdr@pvt.net
Czech error messages.
- Jaroslav Lewandowski jotel@itnet.com.pl
Polish error messages.
- Miguel Angel Fernandez Roiz
Spanish error messages.
- Roy-Magne Mo rmo@www.hivolda.no
Norwegian error messages and testing of Version 3.21.#.
- Timur I. Bakeyev root@timur.tatarstan.ru
Russian error messages.
- brenno@dewinter.com & Filippo Grassilli phil@hyppo.com
Italian error messages.
- Dirk Munzinger dirk@trinity.saar.de
German error messages.
- Billik Stefan billik@sun.uniag.sk
Slovak error messages.
- Stefan Saroiu tzoompy@cs.washington.edu
Romanian error messages.
- Peter Feher
Hungarian error messages.
- Roberto M. Serqueira
Portuguese error messages.
- Carsten H. Pedersen
Danish error messages.
- Arjen G. Lentz
Dutch error messages, completing earlier partial translation (also work on consistency and spelling).
- David Sacerdote davids@secnet.com
Ideas for secure checking of DNS hostnames.
- Wei-Jou Chen jou@nematic.ieo.nctu.edu.tw
Some support for Chinese(BIG5) characters.
- Wei He hewei@mail.ied.ac.cn
A lot of functionality for the Chinese(GBK) character set.
- Zeev Suraski bourbon@netvision.net.il
FROM_UNIXTIME() time formatting, ENCRYPT() functions, and bison advisor.
Active mailing list member.
- Luuk de Boer luuk@wxs.nl
Ported (and extended) the benchmark suite to DBI/DBD. Have been of great help with `crash-me` and running benchmarks. Some new date functions. The `mysql_setpermissions` script.

- Jay Flaherty fty@mediapulse.com
Big parts of the Perl DBI/DBD section in the manual.
- Paul Southworth pauls@etext.org, Ray Loyzaga yar@cs.su.oz.au
Proof-reading of the Reference Manual.
- Alexis Mikhailov root@medinf.chuvashia.su
User-definable functions (UDFs); CREATE FUNCTION and DROP FUNCTION.
- Andreas F. Bobak bobak@relog.ch
The AGGREGATE extension to UDF functions.
- Ross Wakelin R.Wakelin@march.co.uk
Help to set up InstallShield for MySQL-Win32.
- Jethro Wright III jetman@li.net
The 'libmysql.dll' library.
- James Pereria jpereira@iafrica.com
Mysqlmanager, a Win32 GUI tool for administrating MySQL Server.
- Curt Sampson cjs@portal.ca
Porting of MIT-pthreads to NetBSD/Alpha and NetBSD 1.3/i386.
- Antony T. Curtis antony.curtis@olcs.net
Porting of the MySQL Database software to OS/2.
- Martin Ramsch m.ramsch@computer.org
Examples in the MySQL Tutorial.
- Steve Harvey
For making `mysqlaccess` more secure.
- Konark IA-64 Centre of Persistent Systems Private Limited
<http://www.pspl.co.in/konark/>. Help with the Win64 port of the MySQL server.
- Albert Chin-A-Young.
Configure updates for Tru64, large file support and better TCP wrappers support.
- John Birrell
Emulation of `pthread_mutex()` for OS/2.
- Benjamin Pflugmann
Extended MERGE tables to handle INSERTS. Active member on the MySQL mailing lists.
- Guilhem Bichot
Fixed handling of exponents for DECIMAL. Author of `mysql_tableinfo`.
- Jocelyn Fournier
Excellent spotting and reporting innumerable bugs (especially in the MySQL 4.1 subquery code).
- Georg Richter
MySQL 4.1 testing and bug hunting. New PHP 5.0 `mysqli` extension (API) for use with MySQL 4.1 and up.

Marc Liyanage

Maintaining the Mac OS X packages and providing invaluable feedback on how to create Mac OS X PKGs.

Other contributors, bugfinders, and testers: James H. Thompson, Maurizio Menghini, Wojciech Tryc, Luca Berra, Zarko Mocnik, Wim Bonis, Elmar Haneke, jehamby@lightside, psmith@BayNetworks.com, duane@connect.com.au, Ted Deppner ted@psyber.com, Mike Simons, Jaakko Hyvatti.

And lots of bug report/patches from the folks on the mailing list.

A big tribute goes to those that help us answer questions on the mysql@lists.mysql.com mailing list:

Daniel Koch dkoch@amcity.com

Irix setup.

Luuk de Boer luuk@wxs.nl

Benchmark questions.

Tim Sailer tps@users.buoy.com

DBD-mysql questions.

Boyd Lynn Gerber gerberb@zenez.com

SCO-related questions.

Richard Mehalick RM186061@shellus.com

xmysql-related questions and basic installation questions.

Zeev Suraski bourbon@netvision.net.il

Apache module configuration questions (log & auth), PHP-related questions, SQL syntax-related questions and other general questions.

Francesc Guasch frankie@citel.upc.es

General questions.

Jonathan J Smith jsmith@wtp.net

Questions pertaining to OS-specifics with Linux, SQL syntax, and other things that might need some work.

David Sklar sklar@student.net

Using MySQL from PHP and Perl.

Alistair MacDonald A.MacDonald@uel.ac.uk

Not yet specified, but is flexible and can handle Linux and maybe HP-UX. Will try to get user to use `mysqlbug`.

John Lyon jlyon@imag.net

Questions about installing MySQL on Linux systems, using either ‘.rpm’ files or compiling from source.

Lorvid Ltd. lorvid@WOLFENET.com

Simple billing/license/support/copyright issues.

Patrick Sherrill patrick@coconet.com

ODBC and VisualC++ interface questions.

Randy Harmon rjharmon@uptimecomputers.com
DBD, Linux, some SQL syntax questions.

C.3 Supporters to MySQL

While MySQL AB owns all copyrights in the MySQL `server` and the MySQL `manual`, we wish to recognise the following companies, which helped us finance the development of the MySQL `server`, such as by paying us for developing a new feature or giving us hardware for development of the MySQL `server`.

VA Linux / Andover.net

Funded replication.

NuSphere Editing of the MySQL manual.

Stork Design studio

The MySQL web site in use between 1998-2000.

Intel Contributed to development on Windows and Linux platforms.

Compaq Contributed to Development on Linux/Alpha.

SWSOft Development on the embedded `mysqld` version.

FutureQuest

`--skip-show-database`

Appendix D MySQL Change History

This appendix lists the changes from version to version in the MySQL source code.

We are now working actively on MySQL 4.1 & 5.0 and will only provide critical bug fixes for MySQL 4.0 and MySQL 3.23. We update this section as we add new features, so that everybody can follow the development.

Our TODO section contains what further plans we have for 4.1 & 5.0. See [\[TODO\]](#), page [\[undefined\]](#).

Note that we tend to update the manual at the same time we make changes to MySQL. If you find a version listed here that you can't find on the MySQL download page (<http://www.mysql.com/downloads/>), this means that the version has not yet been released!

D.1 Changes in release 5.0.0 (Development)

For the time being, version 5.0 is only available in source code. See [\[Installing source tree\]](#), page [\[undefined\]](#).

The following changelog shows what has already been done in the 5.0 tree:

- Basic support for stored procedures (SQL-99 style).
- Added `SELECT INTO list_of_vars`, which can be of mixed, i.e., global and local type.
- Deprecated the update log (no longer supported). It is fully replaced by the binary log.

D.2 Changes in release 4.1.x (Alpha)

Version 4.1 of the MySQL server includes many enhancements and new features. Binaries for this version are available for download at <http://www.mysql.com/downloads/mysql-4.1.html>.

- Subqueries:


```
SELECT * FROM t1 WHERE t1.a=(SELECT t2.b FROM t2);
```

```
SELECT * FROM t1 WHERE (1,2,3) IN (SELECT a,b,c FROM t2);
```
- Derived tables:


```
SELECT t1.a FROM t1, (SELECT * FROM t2) t3 WHERE t1.a=t3.a;
```
- `INSERT ... ON DUPLICATE KEY UPDATE ...` syntax. This allows you to `UPDATE` an existing row if the insert would cause a duplicate value in a `PRIMARY` or `UNIQUE` key. (`REPLACE` allows you to overwrite an existing row, which is something entirely different.) See [\[INSERT\]](#), page [\[undefined\]](#).
- A newly designed `GROUP_CONCAT()` aggregate function. See [\[Group by functions\]](#), page [\[undefined\]](#).
- Extensive Unicode (UTF8) support.
- Character sets can be defined per column, table and database.
- `BTREE` index on `HEAP` tables.

- Support for OpenGIS (Geographical data). See [\[GIS spatial extensions in MySQL\]](#), page [\[undefined\]](#).
- `SHOW WARNINGS` shows warnings for the last command. See [\[SHOW WARNINGS\]](#), page [\[undefined\]](#).
- Faster binary protocol with prepared statements and parameter binding. See [\[C API Prepared statements\]](#), page [\[undefined\]](#).
- Multi-line queries: You can now issue several queries at once and then read the results in one go. See [\[C API multiple queries\]](#), page [\[undefined\]](#).
- Create Table: `CREATE [TEMPORARY] TABLE [IF NOT EXISTS] table LIKE table`.
- Server based `HELP` command that can be used in the `mysql` command line client (and other clients) to get help for SQL commands.

For a full list of changes, please refer to the changelog sections for each individual 4.1.x release.

D.2.1 Changes in release 4.1.1 (not released yet)

Functionality added or changed:

New global variable `RELAY_LOG_PURGE` to enable/disable automatic relay log purging.
`LOAD DATA` now produces warnings that can be fetched with `SHOW WARNINGS`

Bugs fixed:

Fixed a rare replication bug when a transaction spanned on two or more relay logs, and the slave was stopped while he was executing the part of the transaction which was in the second or further relay log. Then replication would resume at the beginning of the second or further relay log, which was wrong (it should resume at `BEGIN`, in the first relay log). Bug #53.

Fixed double freed memory

Fixed crashing bug with temp. tables in `UNION's`

Fixed a crashing bug in `DERIVED TABLES` when `EXPLAIN` is used on a `DERIVED TABLES` with a join

Fixed a crashing bug in `DELETE` with `ORDER BY` and `LIMIT` caused by non initiated array of reference pointers.

Fixed a bug in `USER()` function caused by the error in the size of the allocated string

Fixed a crashing bug when attempting to create a table with `GEOMETRY` column type with a storage engine that does not support it.

Fixed a crashing bug in `UNION` caused by the empty select list and a non-existent field being used in some of the sub-selects.

Fixed a replication bug when the master is 3.23 and the slave 4.0: the slave lost the replicated temporary tables if `FLUSH LOGS` was issued on the master (bug #254).

D.2.2 Changes in release 4.1.0 (03 Apr 2003: Alpha)

Functionality added or changed:

On Windows, we are now using shared memory to communicate between server and client when they are running on the same machine and you are connecting to `localhost`.

REPAIR of MyISAM tables now uses less temporary disk space when sorting char columns.

DATE/DATETIME checking is now a bit stricter to support the ability to automatically distinguish between date, datetime, and time with microseconds. For example, dates of type `YYYYMMDD HHMMDD` are no longer supported; one must either have separators between each DATE/TIME part or not at all.

Server side help for all MySQL functions. One can now type `help week` in the `mysql` client and get help for the `week()` function.

Added new client function: `mysql_get_server_version()`.

Fixed bug in `libmysqlclient` that fetched field defaults

Fixed bug in `'mysql.cc'` client when skipping comments

Added `record_in_range()` method to MERGE tables to be able to choose the right index when there are many to choose from.

Replication now works with `RAND()` and user variables `@var`.

Allow one to change mode for `ANSI_QUOTES` on the fly.

EXPLAIN SELECT now can be killed. See [\(undefined\) \[KILL\]](#), page [\(undefined\)](#).

REPAIR TABLE now can be killed. See [\(undefined\) \[KILL\]](#), page [\(undefined\)](#).

Allow one to specify empty key lists for `USE|IGNORE|FORCE INDEX`.

DROP TEMPORARY TABLE now only drops temporary tables and doesn't end transactions.

Added a support for UNION in derived tables.

TIMESTAMP is now returned as string of type `'YYYY-MM-DD HH:MM:DD'` and different timestamp lengths are not supported.

This change was necessary for SQL standards compliance. In a future version, a further change will be made (backward compatible with this change), allowing the timestamp length to indicate the desired number of digits of fractions of a second.

New faster client/server protocol which supports prepared statements, bound parameters, and bound result columns, binary transfer of data, warnings.

Added database and real table name (in case of alias) to the `MYSQL_FIELD` structure.

Multi-line queries: You can now issue several queries at once and then read the results in one go.

In `CREATE TABLE foo (a INT not null primary key)` the `PRIMARY` word is now optional.

In `CREATE TABLE` the attribute `SERIAL` is now an alias for `BIGINT NOT NULL AUTO_INCREMENT UNIQUE`.

`SELECT ... FROM DUAL` is an alias for `SELECT ...` (To be compatible with some other databases).

If one creates a too long `CHAR/VARCHAR` it's now automatically changed to `TEXT` or `BLOB`; One will get a warning in this case.

One can specify the different BLOB/TEXT types with the syntax BLOB(length) and TEXT(length). MySQL will automatically change it to one of the internal BLOB/TEXT types.

CHAR BYTE is an alias for CHAR BINARY.

VARCHARACTER is an alias for VARCHAR.

New operators integer MOD integer and integer DIV integer.

SERIAL DEFAULT VALUE added as an alias for AUTO_INCREMENT.

TRUE and FALSE added as alias for 1 and 0, respectively.

Aliases are now forced in derived tables, as per SQL-99.

Fixed SELECT .. LIMIT 0 to return proper row count for SQL_CALC_FOUND_ROWS.

One can specify many temporary directories to be used in a round-robin fashion with: --tmpdir=dirname1:dirname2:dirname3.

Subqueries: SELECT * from t1 where t1.a=(SELECT t2.b FROM t2).

Derived tables:

```
SELECT a.col1, b.col2
      FROM (SELECT MAX(col1) AS col1 FROM root_table) a,
           other_table b
      WHERE a.col1=b.col1;
```

Character sets to be defined per column, table and database.

Unicode (UTF8) support.

BTREE index on HEAP tables.

Faster embedded server (new internal communication protocol).

One can add a comment per column in CREATE TABLE.

SHOW FULL COLUMNS FROM table_name shows column comments.

ALTER DATABASE.

Support for GIS (Geometrical data). See [\[GIS spatial extensions in MySQL\], page \[undefined\]](#).

SHOW [COUNT(*)] WARNINGS shows warnings from the last command.

One can specify a column type for a column in CREATE TABLE ... SELECT by defining the column in the CREATE part.

```
CREATE TABLE foo (a tinyint not null) SELECT b+1 AS 'a' FROM bar;
expr SOUNDS LIKE expr same as SOUNDEX(expr)=SOUNDEX(expr).
```

VARIANCE(expr) returns the variance of expr

One can create a table from the existing table using CREATE [TEMPORARY] TABLE [IF NOT EXISTS] table (LIKE table). The table can be either normal or temporary.

New options --reconnect and disable-reconnect for the mysql client, to reconnect automatically or not if the connection is lost.

START SLAVE (STOP SLAVE) no longer returns an error if the slave is already started (stopped); it returns a warning instead.

SLAVE START and SLAVE STOP are no longer accepted by the query parser; use START SLAVE and STOP SLAVE instead.

D.3 Changes in release 4.0.x (Production)

Version 4.0 of the MySQL server includes many enhancements and new features:

- The InnoDB table type is now included in the standard binaries, adding transactions, row-level locking, and foreign keys. See [\[InnoDB\]](#), page [\[undefined\]](#).
- A query cache, offering vastly increased performance for many applications. By caching complete result sets, later identical queries can return instantly. See [\[Query Cache\]](#), page [\[undefined\]](#).
- Improved full-text indexing with boolean mode, truncation, and phrase searching. See [\[Fulltext Search\]](#), page [\[undefined\]](#).
- Enhanced MERGE tables, now supporting INSERTs and AUTO_INCREMENT. See [\[MERGE\]](#), page [\[undefined\]](#).
- UNION syntax in SELECT. See [\[UNION\]](#), page [\[undefined\]](#).
- Multi-table DELETE statements. See [\[DELETE\]](#), page [\[undefined\]](#).
- libmysqld, the embedded server library. See [\[libmysqld\]](#), page [\[undefined\]](#).
- Additional GRANT privilege options for even tighter control and security. See [\[GRANT\]](#), page [\[undefined\]](#).
- Management of user resources in the GRANT system, particularly useful for ISPs and other hosting providers. See [\[User resources\]](#), page [\[undefined\]](#).
- Dynamic server variables, allowing configuration changes without having to take down the server. See [\[SET OPTION\]](#), page [\[undefined\]](#).
- Improved replication code and features. See [\[Replication\]](#), page [\[undefined\]](#).
- Numerous new functions and options.
- Changes to existing code for enhanced performance and reliability.

For a full list of changes, please refer to the changelog sections for each individual 4.0.x release.

D.3.1 Changes in release 4.0.14 (not released yet)

Functionality added or changed:

Added multi-threaded MyISAM repair optimisation and `myisam_repair_threads` variable to enable it. See [\[myisam_repair_threads\]](#), page [\[undefined\]](#).

Added `innodb_max_dirty_pages_pct` variable which controls amount of dirty pages allowed in InnoDB buffer pool.

Bugs fixed:

LOAD DATA INFILE will now read 000000 as a zero date instead as "2000-00-00".

Fixed bug that caused DELETE FROM table WHERE `const_expression` always to delete the whole table (even if expression result was false). (bug #355)

D.3.2 Changes in release 4.0.13 (to be released soon)

Functionality added or changed:

Disabled MyISAM RAID in the Max binary packages - several users reported assertion failures even when not using RAID (bug #346). Some of its functionality can be achieved by using MERGE tables instead. See [\[MERGE tables\]](#), page [\[undefined\]](#).

CURRENT_USER() and "access denied" error messages now report hostname exactly as it was specified in the GRANT command.

Removed Benchmark results from the source and binary distributions. They are still available in the BK source tree, though.

InnoDB tables now support ANALYZE TABLE.

MySQL now issues a warning when it opens a table that was created with MySQL 4.1. 'mysql' command line client no longer looks for * commands inside backtick-quoted strings.

Option --new now changes binary items (0xFFDF) to be treated as binary strings instead of numbers by default. This fixes some problems with character sets where it's convenient to input the string as a binary item. After this change you have to convert the binary string to INTEGER with a CAST if you want to compare two binary items with each other and know which one is bigger than the other. SELECT CAST(0xff AS UNSIGNED) < CAST(0xff AS UNSIGNED). This will be the default behaviour in MySQL 4.1. (Bug #152)

Fixed bug with NATURAL LEFT JOIN, NATURAL RIGHT JOIN and RIGHT JOIN when using many joined tables. The problem was that the JOIN method was not always associated with the tables surrounding the JOIN method. If you have a query that uses many RIGHT JOIN or NATURAL ... JOINS you should check that they work as you expected after upgrading MySQL to this version. (Bug #291)

Enabled delayed_insert_timeout on Linux (most modern glibc libraries have a fixed pthread_cond_timedwait). (Bug #211)

Don't create more insert delayed threads than given by max_insert_delayed_threads. (Bug #211)

Fixed that SET SQL_BIG_SELECTS=1 works as documented (New bug in 4.0)

Changed UPDATE ... LIMIT to also count accepted, but not changed rows.

Tuned optimizer to favour clustered index over table scan.

BIT_AND() and BIT_OR() now return an unsigned 64 bit value.

Added warnings to error log of why a secure connection failed (when running with --log-warnings).

Deprecated options --skip-symlink and --use-symbolic-links and replaced these with --symbolic-links.

The default option for innodb_flush_log_at_trx_commit was changed from 0 to 1 to make InnoDB tables ACID by default. See [\[InnoDB start\]](#), page [\[undefined\]](#).

Added a feature to SHOW KEYS to display keys that are disabled by ALTER TABLE DISABLE KEYS command.

When using a non-existing table type with `CREATE TABLE`, first try if the default table type exists before falling back to MyISAM.

Added `MEMORY` as an alias for `HEAP`.

Renamed function `rnd` to `my_rnd` as the name was too generic and is an exported symbol in `libmysqlclient` (thanks to Dennis Haney for the initial patch).

Portability fix: renamed `'include/dbug.h'` to `'include/my_debug.h'`.

`mysqldump` no longer silently deletes the binlogs when called with `--master-data` or `--first-slave`; while this behaviour was convenient for some users, others may suffer from it. Now one has to explicitly ask for this deletion with the new `--delete-master-logs` option.

If the slave is configured (using for example `replicate-wild-ignore-table=mysql.%`) to exclude `mysql.user`, `mysql.host`, `mysql.db`, `mysql.tables_priv` and `mysql.columns_priv` from replication, then `GRANT` and `REVOKE` will not be replicated.

Bugs fixed:

Fixed wrong result from truncation operator (`*`) in `MATCH ... AGAINST()` in some complex joins.

Fixed a crash in `REPAIR ... USE_FRM` command, when used on read-only or nonexisting table.

Fixed a crashing bug in `mysql` monitor program. It occurred if program was started with `--no-defaults`, with a prompt that contained hostname and connection to non-existing db was requested

Fixed problem when comparing a key for a multi-byte-character set. (Bug #152)

Fixed bug in `LEFT`, `RIGHT` and `MID` when used with multi-byte character sets and some `GROUP BY` queries. (Bug #314)

Fix problem with `ORDER BY` being discarded for some `DISTINCT` queries. (Bug #275)

Fixed some serious bugs in `UPDATE ... ORDER BY`. (Bug #241)

Fixed unlikely problem in optimising `WHERE` clause with constant expression like in `WHERE 1 AND (a=1 AND b=1)`.

Fixed that `SET SQL_BIG_SELECTS=1` works again.

Introduced proper backtick quoting for `db.table` in `SHOW GRANTS`

Bug #283 (`FULLTEXT` index stops working after `ALTER TABLE` that converts `TEXT` field to `CHAR`) fixed.

Mark a MyISAM table as "analyzed" only when all the keys are indeed analyzed.

Only ignore world-writeable `'my.cnf'` files that are regular files (and not e.g. named pipes or character devices).

Fixed few smaller issues with `SET PASSWORD`.

Fixed error message which contained deprecated text.

Fixed a bug with two `NATURAL JOINS` in the query.

`SUM()` didn't return `NULL` when there was no rows in result or when all values was `NULL`.

On Unix symbolic links handling was not enabled by default and there was no way to turn this on.

Added missing dashes to parameter `--open-files-limit` in `'mysqld_safe'` (bug #264).

Fixed wrong hostname for TCP/IP connections displayed in `SHOW PROCESSLIST`.

Fixed a bug with NAN in `FORMAT(...)` function ...

Fixed a bug with improperly cached database privileges.

Fixed a bug in `ALTER TABLE ENABLE / DISABLE KEYS` which failed to force a refresh of table data in the cache.

Fixed bugs in replication of `LOAD DATA INFILE` for custom parameters (`ENCLOSED`, `TERMINATED` and so on) and temporary tables (Bugs #183 and #222).

Fixed a replication bug when the master is 3.23 and the slave 4.0: the slave lost the replicated temporary tables if `FLUSH LOGS` was issued on the master (bug #254).

Fixed bug in `multi-table updates` which occurred whenever a temporary table, containing update data, had to be converted from `HEAP` to `MyISAM`.

Fixed a bug when doing `LOAD DATA INFILE IGNORE` : when reading the binary log, `mysqlbinlog` and the replication code read `REPLACE` instead of `IGNORE`. This could make the slave's table become different from the master's table. Bug #218.

Fixed a deadlock when `relay_log_space_limit` was set to a too small value. Bug #79.

Fixed a bug in `HAVING` clause when an alias is used from the `select list`.

Fixed a bug in `MyISAM` when a row is inserted into a table with a large number of `NULL` columns. Bug was caused by wrong calculation of the record length, as the space required for storage of `NULL` bits was not added to the total record length.

Fixed a bug when `SELECT @nonexistent_variable` caused the error in client - server protocol due to `net_printf()` being sent to the client twice.

Fixed a bug in setting `SQL_BIG_SELECTS` option.

Fixed a bug in `SHOW PROCESSLIST` which only displayed a localhost in the "Host" column. This was caused by a glitch that only used current thread info instead info from the linked list of threads.

Removed unnecessary Mac OS X helper files from server RPM (bug #144).

Allow optimisation of `multi-table-update` for InnoDB tables as well.

Fixed a bug in `multi-table-updates` that caused some rows to be updated several times.

Fixed a bug in `mysqldump` when it was called with `--master-data`: the `CHANGE MASTER TO` commands appended to the SQL dump had wrong coordinates (bug #159).

Fixed a bug when an updating query using `USER()` was replicated on the slave ; this caused segfault on the slave (bug #178). `USER()` is still badly replicated on the slave (it is replicated to "").

D.3.3 Changes in release 4.0.12 (15 Mar 2003: Production)

Functionality added or changed:

`'mysqld'` no longer reads options from world-writable config files.

Integer values between 9223372036854775807 and 9999999999999999 are now regarded as unsigned longlongs, not as floats. This makes these values work similar to values between 1000000000000000000 and 18446744073709551615.

`SHOW PROCESSLIST` will now include the client TCP port after the hostname to make it easier to know from which client the request originated.

Bugs fixed:

Fixed 'mysqld' crash on extremely small values of `sort_buffer` variable.

`INSERT INTO u SELECT ... FROM t` was written too late to the binary log if `t` was very frequently updated during the execution of this query. This could cause a problem with `mysqlbinlog` or replication. The master must be upgraded, not the slave. (bug #136).

Fixed checking of random part of `WHERE` clause (bug #142).

Fixed a bug with `multi-table updates` with `InnoDB` tables. This bug occurred as, in many cases, `InnoDB` tables can not be updated "on the fly", but offsets to the records have to be stored in a temporary table.

Added missing file 'mysql_secure_installation' to the `server` RPM subpackage (bug #141).

Fixed MySQL (and 'myisamchk') crash on artificially corrupted `.MYI` files.

Don't allow `BACKUP TABLE` to overwrite existing files.

Fixed a bug with `multi-table UPDATES` when user had all privileges on the database where tables are located and there were any entries in `tables_priv` table, i.e. `grant_option` was true.

Fixed a bug that allowed a user with table or column grants on some table, `TRUNCATE` any table in the same database.

Fixed deadlock when doing `LOCK TABLE` followed by `DROP TABLE` in the same thread. In this case one could still kill the thread with `KILL`.

`LOAD DATA LOCAL INFILE` was not properly written to the binary log (hence not properly replicated). (bug #82).

`RAND()` entries were not read correctly by `mysqlbinlog` from the binary log which caused problems when restoring a table that was inserted with `RAND()`. `INSERT INTO t1 VALUES(RAND())`. In replication this worked ok.

`SET SQL_LOG_BIN=0` was ignored for `INSERT DELAYED` queries. (bug #104).

`SHOW SLAVE STATUS` reported too old positions (columns `Relay_Master_Log_File` and `Exec_master_log_pos`) for the last executed statement from the master, if this statement was the `COMMIT` of a transaction. The master must be upgraded for that, not the slave. (bug #52).

`LOAD DATA INFILE` was not replicated by the slave if `replicate_*_table` was set on the slave. (bug #86).

After `RESET SLAVE`, the coordinates displayed by `SHOW SLAVE STATUS` looked un-reset (though they were, but only internally). (bug #70).

Fixed query cache invalidation on `LOAD DATA`.

Fixed memory leak on `ANALYZE` procedure with error.

Fixed a bug in handling `CHAR(0)` columns that could cause wrong results from the query.

Fixed rare bug with wrong initialization of `AUTO_INCREMENT` column, as a secondary column in a multi-column key (see [AUTO_INCREMENT-snt \[AUTO_INCREMENT on secondary column in a multi-column key\]](#), page [AUTO_INCREMENT-pg](#)), when data was inserted with `INSERT . . . SELECT` or `LOAD DATA` into an empty table.

On windows, `STOP SLAVE` didn't stop the slave until the slave got one new command from the master (this bug has been fixed for MySQL 4.0.11 by releasing updated 4.0.11a windows packages, which include this individual fix on top of the 4.0.11 sources). (bug #69).

Fixed a crash when no database was selected and `LOAD DATA` command was issued with full table name specified, including database prefix.

Fixed a crash when shutting down replication on some platforms (e.g. Mac OS X).

Fixed a portability bug with `pthread_attr_getstacksize` on HP-UX 10.20 (Patch was also included in 4.0.11a sources).

Fixed the `bigint` test to not fail on some platforms (e.g. HP-UX and Tru64) due to different return values of the `atof()` function.

Fixed the `rpl_rotate_logs` test to not fail on certain platforms (e.g. Mac OS X) due to a too long file name (changed `slave-master-info.opt` to `.slave-mi`).

D.3.4 Changes in release 4.0.11 (20 Feb 2003)

Functionality added or changed:

`NULL` is now sorted **LAST** if you use `ORDER BY . . . DESC` (as it was before before MySQL 4.0.2). This change was required to comply with the SQL-99 standard. (The original change was made because we thought that SQL-99 required `NULL` to be always sorted at the same position, but this was wrong).

Added `START TRANSACTION` (SQL-99 syntax) as alias for `BEGIN`. This is recommended to use instead of `BEGIN` to start a transaction.

Added `OLD_PASSWORD()` as a synonym for `PASSWORD()`.

Allow keyword `ALL` in group functions.

Added support for some new `INNER JOIN` and `JOIN` syntaxes. For example, `SELECT * FROM t1 INNER JOIN t2` didn't work before.

Novell NetWare 6.0 porting effort completed, Novell patches merged into the main source tree.

Bugs fixed:

Fixed problem with `multi-table-delete` and InnoDB tables.

Fixed a problem with `BLOB NOT NULL` columns used with `IS NULL`.

Re-added missing pre- and post(un)install scripts to the Linux RPM packages (they were missing after the renaming of the server subpackage).

Fixed that table locks are not released with multi-table updates and deletes with InnoDB storage engine.

Fixed bug in updating `BLOB` columns with long strings.

Fixed integer-wraparound when giving big integer (≥ 10 digits) to function that requires an unsigned argument, like `CREATE TABLE (...) AUTO_INCREMENT=#`.

`MIN(key_column)` could in some cases return NULL on a column with NULL and other values.

`MIN(key_column)` and `MAX(key_column)` could in some cases return wrong values when used in `OUTER JOIN`.

`MIN(key_column)` and `MAX(key_column)` could return wrong values if one of the tables was empty.

Fixed rare crash in compressed MyISAM tables with blobs.

Fixed bug in using aggregate functions as argument for `INTERVAL`, `CASE`, `FIELD`, `CONCAT_WS`, `ELT` and `MAKE_SET` functions.

When running with `--lower-case-table-names` (default on windows) and you had tables or databases with mixed case on disk, then executing `SHOW TABLE STATUS` followed with `DROP DATABASE` or `DROP TABLE` could fail with Errcode 13.

D.3.5 Changes in release 4.0.10 (29 Jan 2003)

Functionality added or changed:

Added option `--log-error[=file_name]` to `mysqld_safe` and `mysqld`. This option will force all error messages to be put in a log file if the option `--console` is not given. On Windows `--log-error` is enabled as default, with a default name of `host_name.err` if the name is not specified.

Changed some things from **Warning:** to **Note:** in the log files.

The `mysqld` server should now compile on Netware.

Added optimisation that if one does `GROUP BY ... ORDER BY NULL` then result is not sorted.

New command-line option for 'mysqld' to replace/disable built-in stopword list, that is used in fulltext search. See [\[ft_stopword_file\]](#), page [\[undefined\]](#).

Changed default stack size from 64K to 192K; This fixes a core dump problem on Red Hat 8.0 and other systems with a glibc that requires > 128 K stack for `gethostbyaddr()` to resolve a hostname. You can fix this for earlier MySQL versions by starting `mysqld` with `--thread-stack=192K`.

Added `mysql_waitpid` to the binary distribution and the `MySQL-client` RPM subpackage (required for `mysql-test-run`).

Renamed the main MySQL RPM package to `MySQL-server`. When updating from an older version, `MySQL-server.rpm` will simply replace `MySQL.rpm`.

If a slave is configured with `replicate_wild_do_table=db.%` or `replicate_wild_ignore_table=db.%`, these rules will be applied to `CREATE/DROP DATABASE` too.

Added timeout value for `MASTER_POS_WAIT()`.

Bugs fixed:

Fixed initialization of the random seed for newly created threads to give a better `rand()` distribution from the first call.

Fixed a bug that caused `mysqld` to hang when a table was opened with the `HANDLER` command and then dropped without being closed.

Fixed bug in logging to binary log (which affects replication) a query that inserts a `NULL` in an `auto_increment` field and also uses `LAST_INSERT_ID()`.

Fixed an unlikely bug that could cause a memory overrun when using `ORDER BY constant_expression`.

Fixed a table corruption in `'myisamchk'`'s parallel repair mode.

Fixed bug in query cache invalidation on simple table renaming.

Fixed bug in `mysqladmin --relative`.

On some 64 bit systems, `show status` reported a strange number for `Open_files` and `Open_streams`.

Fixed wrong number of columns in `EXPLAIN` on empty table.

Fixed bug in `LEFT JOIN` that caused zero rows to be returned in the case the `WHERE` condition was evaluated as `FALSE` after reading const tables. (Unlikely condition).

`FLUSH PRIVILEGES` didn't correctly flush table/column privileges when `mysql.tables_priv` is empty.

Fixed bug in replication when using `LOAD DATA INFILE` one a file that updated and `auto_increment` field with `NULL` or `0`. This bug only affected MySQL 4.0 masters (not slaves or MySQL 3.23 masters). **NOTE:** If you have a slave that has replicated a file with generated `auto_increment` fields then the slave data is corrupted and you should reinitialise the affected tables from the master.

Fixed possible memory overrun when sending a blob > 16M to the client.

Fixed wrong error message when setting a `NOT NULL` field to an expression that returned `NULL`.

Fixed core dump bug in `str LIKE "%other_str%"` where `str` or `other_str` contained characters >= 128.

Fixed bug: When executing on master `LOAD DATA` and InnoDB failed with `table full` error the binary log was corrupted.

D.3.6 Changes in release 4.0.9 (09 Jan 2003)

Functionality added or changed:

`OPTIMIZE TABLE` will for MyISAM tables treat all `NULL` values as different when calculating cardinality. This helps in optimising joins between tables where one of the tables has a lot of `NULL` values in a indexed column:

```
SELECT * from t1,t2 where t1.a=t2.key_with_a_lot_of_null;
```

Added join operator `FORCE INDEX (key_list)`. This acts likes `USE INDEX (key_list)` but with the addition that a table scan is assumed to be `VERY` expensive. One bad thing with this is that it makes `FORCE` a reserved word.

Reset internal row buffer in MyISAM after each query. This will reduce memory in the case you have a lot of big blobs in a table.

Bugs fixed:

A security patch in 4.0.8 causes the mysqld server to die if the remote hostname can't be resolved. This is now fixed.

Fixed crash when replication big `LOAD DATA INFILE` statement that caused log rotation.

D.3.7 Changes in release 4.0.8 (07 Jan 2003)

Functionality added or changed:

Default `max_packet_length` for `libmysqld.c` is now `1024*1024*1024`.

One can now specify `max_allowed_packet` in a file ready by `mysql_options(MYSQL_READ_DEFAULT_FILE)`. for clients.

When sending a too big packet to the server with the not compressed protocol, the client now gets an error message instead of a lost connection.

We now send big queries/result rows in bigger hunks, which should give a small speed improvement.

Fixed some bugs with the compressed protocol for rows > 16M.

InnoDB tables now also supports `ON UPDATE CASCADE` in `FOREIGN KEY` constraints. See the InnoDB section in the manual for the InnoDB changelog.

Bugs fixed:

Fixed bug in `ALTER TABLE` with BDB tables.

Fixed core dump bug in `QUOTE()` function.

Fixed a bug in handling communication packets bigger than 16M. Unfortunately this required a protocol change; If you upgrade the server to 4.0.8 and above and have clients that uses packets $\geq 255*255*255$ bytes (=16581375) you must also upgrade your clients to at least 4.0.8. If you don't upgrade, the clients will hang when sending a big packet.

Fixed bug when sending blobs longer than 16M to client.

Fixed bug in `GROUP BY` when used on BLOB column with NULL values.

Fixed a bug in handling NULLs in `CASE ... WHEN ...`

D.3.8 Changes in release 4.0.7 (20 Dec 2002)

Functionality added or changed:

`mysqlbug` now also reports the compiler version used for building the binaries (if the compiler supports the option `--version`).

Bugs fixed:

Fixed compilation problems on OpenUnix and HP-UX 10.20.

Fixed some optimisation problems when compiling MySQL with `-DBIG_TABLES` on a 32 bit system.

`mysql_drop_db()` didn't check permissions properly so anyone could drop another users database. `DROP DATABASE` is checked properly.

D.3.9 Changes in release 4.0.6 (14 Dec 2002: Gamma)

Functionality added or changed:

Added syntax support for `CHARACTER SET xxx` and `CHARSET=xxx` table options (to be able to read table dumps from 4.1).

Fixed replication bug that caused the slave to lose its position in some cases when the replication log was rotated.

Fixed that a slave will restart from the start of a transaction if it's killed in the middle of one.

Moved the manual pages from `'man'` to `'man/man1'` in the binary distributions.

The default type returned by `IFNULL(A,B)` is now set to be the more 'general' of the types of A and B. (The order is `STRING`, `REAL` or `INTEGER`).

Moved the `'mysql.server'` startup script in the RPM packages from `'/etc/rc.d/init.d/mysql'` to `'/etc/init.d/mysql'` (which almost all current Linux distributions support for LSB compliance).

Added `Qcache_lowmem_prunes` status variable (number of queries that were deleted from cache because of low memory).

Fixed `mysqlcheck` so it can deal with table names containing dashes.

Bulk insert optimisation (see [<undefined> \[bulk_insert_buffer_size\]](#), page [<undefined>](#)) is no longer used when inserting small (less than 100) number of rows.

Optimisation added for queries like `SELECT ... FROM merge_table WHERE indexed_column=constant_expr`.

Added functions `LOCALTIME` and `LOCALTIMESTAMP` as synonyms for `NOW()`.

`CEIL` is now an alias for `CEILING`.

The `CURRENT_USER()` function can be used to get a `user@host` value as it was matched in the `GRANT` system. See [<undefined> \[CURRENT_USER\(\)\]](#), page [<undefined>](#).

Fixed `CHECK` constraints to be compatible with SQL-99. This made `CHECK` a reserved word. (Checking of `CHECK` constraints is still not implemented).

Added `CAST(... as CHAR)`.

Added PostgreSQL compatible `LIMIT` syntax: `SELECT ... LIMIT # OFFSET #`

`mysql_change_user()` will now reset the connection to the state of a fresh connect (Ie, `ROLLBACK` any active transaction, close all temporary tables, reset all user variables etc..)

Bugs fixed:

Fixed number of found rows returned in `multi table updates`

Make `--lower-case-table-names` default on Mac OS X as the file system is case sensitive.

Transactions in `AUTOCOMMIT=0` mode didn't rotate binary log.

A fix for the bug in a `SELECT` with joined tables with `ORDER BY` and `LIMIT` clause when `filesort` had to be used. In that case `LIMIT` was applied to `filesort` of one of the tables, although it could not be. This fix solved problems with `LEFT JOIN` too.

`mysql_server_init()` now makes a copy of all arguments. This fixes a problem when using the embedded server in C# program.

Fixed buffer overrun in `libmysqlclient` library that allowed a malicious MySQL server to crash the client application.

Fixed security-related bug in `mysql_change_user()` handling. All users are strongly recommended to upgrade to version 4.0.6.

Fixed bug that prevented `--chroot` command-line option of `mysqld` from working.

Fixed bug in phrase operator "... " in boolean full-text search.

Fixed bug that caused `OPTIMIZE TABLE` to corrupt the table under some rare circumstances.

Part rewrite of multi-table-update to optimise it, make it safer and more bug free.

`LOCK TABLES` now works together with multi-table-update and multi-table-delete.

`--replicate-do=xxx` didn't work for `UPDATE` commands. (Bug introduced in 4.0.0)

Fixed shutdown problem on Mac OS X.

Major InnoDB bugs in `REPLACE`, `AUTO_INCREMENT`, `INSERT INTO ... SELECT ...` were fixed. See the InnoDB changelog in the InnoDB section of the manual.

D.3.10 Changes in release 4.0.5 (13 Nov 2002)

Functionality added or changed:

Port number was added to host name (if it is known) in `SHOW PROCESSLIST` command

Changed handling of last argument in `WEEK()` so that one can get week number according to the ISO 8601 specification. (Old code should still work).

Fixed that `INSERT DELAYED` threads doesn't hang on `Waiting for INSERT` when one sends a `SIGHUP` to `mysqld`.

Change that `AND` works according to SQL-99 when it comes to `NULL` handling. In practice, this only affects queries where you do something like `WHERE ... NOT (NULL AND 0)`.

`mysqld` will now resolve `basedir` to its full path (with `realpath()`). This enables one to use relative symlinks to the MySQL installation directory. This will however cause `show variables` to report different directories on systems where there is a symbolic link in the path.

Fixed that MySQL will not use index scan on index disabled with `IGNORE INDEX` or `USE INDEX`. to be ignored.

Added `--use-frm` option to `mysqlcheck`. When used with `REPAIR`, it gets the table structure from the `.frm` file, so the table can be repaired even if the `.MYI` header is corrupted.

Fixed bug in `MAX()` optimisation when used with `JOIN` and `ON` expressions.

Added support for reading of MySQL 4.1 table definition files.

`BETWEEN` behaviour changed (see [\[Comparison Operators\]](#), page [\[undefined\]](#)). Now `datetime_col BETWEEN timestamp AND timestamp` should work as expected.

One can create `TEMPORARY MERGE` tables now.

`DELETE FROM myisam_table` now shrinks not only the `.MYD` file but also the `.MYI` file.

When one uses the `--open-files-limit=#` option to `mysqld_safe` it's now passed on to `mysqld`.

Changed output from `EXPLAIN` from `'where used'` to `'Using where'` to make it more in line with other output.

Removed variable `safe_show_database` as it was no longer used.

Updated source tree to be built using `automake 1.5` and `libtool 1.4`.

Fixed an inadvertently changed option (`--ignore-space`) back to the original `--ignore-spaces` in `mysqlclient`. (Both syntaxes will work).

Don't require `UPDATE` privilege when using `REPLACE`.

Added support for `DROP TEMPORARY TABLE ...`, to be used to make replication safer.

When transactions are enabled, all commands that update temporary tables inside a `BEGIN/COMMIT` are now stored in the binary log on `COMMIT` and not stored if one does `ROLLBACK`. This fixes some problems with non-transactional temporary tables used inside transactions.

Allow braces in joins in all positions. Formerly, things like `SELECT * FROM (t2 LEFT JOIN t3 USING (a)), t1` worked, but not `SELECT * FROM t1, (t2 LEFT JOIN t3 USING (a))`. Note that braces are simply removed, they do not change the way the join is executed.

InnoDB now supports also isolation levels `READ UNCOMMITTED` and `READ COMMITTED`. For a detailed InnoDB changelog, see [\[InnoDB change history\]](#), page [\[undefined\]](#) in this manual.

Bugs fixed:

Fixed bug in `MAX()` optimisation when used with `JOIN` and `ON` expressions.

Fixed that `INSERT DELAY` threads don't hang on `Waiting for INSERT` when one sends a `SIGHUP` to `mysqld`.

Fixed that MySQL will not use an index scan on an index that has been disabled with `IGNORE INDEX` or `USE INDEX`.

Corrected test for `root` user in `mysqld_safe`.

Fixed error message issued when storage engine cannot do `CHECK` or `REPAIR`.

Fixed rare core dump problem in complicated `GROUP BY` queries that didn't return any result.

Fixed `mysqlshow` to work properly with wildcarded database names and with database names that contain underscores.

Portability fixes to get MySQL to compile cleanly with Sun Forte 5.0.

Fixed MyISAM crash when using dynamic-row tables with huge numbers of packed fields.

Fixed query cache behaviour with BDB transactions.

Fixed possible floating point exception in `MATCH` relevance calculations.

Fixed bug in full-text search `IN BOOLEAN MODE` that made `MATCH` to return incorrect relevance value in some complex joins.

Fixed a bug that limited MyISAM key length to a value slightly less than 500. It is exactly 500 now.

Fixed that `GROUP BY` on columns that may have a `NULL` value doesn't always use disk based temporary tables.

The filename argument for the `--des-key-file` argument to `mysqld` is interpreted relative to the data directory if given as a relative pathname.

Removed a condition that temp table with index on column that can be `NULL` has to be MyISAM. This was okay for 3.23, but not needed in 4.*. This resulted in slowdown in many queries since 4.0.2.

Small code improvement in multi-table updates.

Fixed a newly introduced bug that caused `ORDER BY ... LIMIT #` to not return all rows.

Fixed a bug in multi-table deletes when outer join is used on an empty table, which gets first to be deleted.

Fixed a bug in multi-table updates when a single table is updated.

Fixed bug that caused `REPAIR TABLE` and `myisamchk` to corrupt `FULLTEXT` indexes.

Fixed bug with caching the `mysql` grant table database. Now queries in this database are not cached in the query cache.

Small fix in `mysqld_safe` for some shells.

Give error if a MyISAM `MERGE` table has more than 2^{32} rows and MySQL was not compiled with `-DBIG_TABLES`.

Fixed some `ORDER BY ... DESC` problems with InnoDB tables.

D.3.11 Changes in release 4.0.4 (29 Sep 2002)

- Fixed bug where `GRANT/REVOKE` failed if hostname was given in non-matching case.
- Don't give warning in `LOAD DATA INFILE` when setting a `timestamp` to a string value of '0'.
- Fixed bug in `myisamchk -R` mode.
- Fixed bug that caused `mysqld` to crash on `REVOKE`.
- Fixed bug in `ORDER BY` when there is a constant in the `SELECT` statement.
- One didn't get an error message if `mysqld` couldn't open the privilege tables.
- `SET PASSWORD FOR ...` closed the connection in case of errors (bug from 4.0.3).
- Increased max possible `max_allowed_packet` in `mysqld` to 1 GB.
- Fixed bug when doing a multi-line `INSERT` on a table with an `AUTO_INCREMENT` key which was not in the first part of the key.
- Changed `LOAD DATA INFILE` to not recreate index if the table had rows from before.
- Fixed overrun bug when calling `AES_DECRYPT()` with incorrect arguments.
- `--skip-ssl` can now be used to disable SSL in the MySQL clients, even if one is using other SSL options in an option file or previously on the command line.
- Fixed bug in `MATCH ... AGAINST(... IN BOOLEAN MODE)` used with `ORDER BY`.
- Added `LOCK TABLES` and `CREATE TEMPORARY TABLES` privilege on the database level. One must run the `mysql_fix_privilege_tables` script on old installations to activate these.

- In `SHOW TABLE ... STATUS`, compressed tables sometimes showed up as `dynamic`.
- `SELECT @@[global|session].var_name` didn't report `global | session` in the result column name.
- Fixed problem in replication that `FLUSH LOGS` in a circular replication setup created an infinite number of binary log files. Now a `rotate-binary-log` command in the binary log will not cause slaves to rotate logs.
- Removed `STOP EVENT` from binary log when doing `FLUSH LOGS`.
- Disable the use of `SHOW NEW MASTER FOR SLAVE` as this needs to be completely changed in 4.1.
- Fixed a bug with constant expression (e.g. field of a one-row table, or field from a table, referenced by a `UNIQUE` key) appeared in `ORDER BY` part of `SELECT DISTINCT`.
- `--log-binary=a.b.c` now properly strips off `.b.c`.
- `FLUSH LOGS` removed numerical extension for all future update logs.
- `GRANT ... REQUIRE` didn't store the SSL information in the `mysql.user` table if SSL was not enabled in the server.
- `GRANT ... REQUIRE NONE` can now be used to remove SSL information.
- `AND` is now optional between `REQUIRE` options.
- `REQUIRE` option was not properly saved, which could cause strange output in `SHOW GRANTS`.
- Fixed that `mysqld --help` reports correct values for `--datadir` and `--bind-address`.
- Fixed that one can drop UDFs that didn't exist when `mysqld` was started.
- Fixed core dump problem with `SHOW VARIABLES` on some 64 bit systems (like Solaris sparc).
- Fixed a bug in `my_getopt`; `--set-variable` syntax didn't work for those options that didn't have a valid variable in `my_option` struct. This affected at least `default-table-type` option.
- Fixed a bug from 4.0.2 that caused `REPAIR TABLE` and `myisamchk --recover` to fail on tables with duplicates in a unique key.
- Fixed a bug from 4.0.3 in calculating the default data type for some functions. This affected queries of type `CREATE TABLE table_name SELECT expression(),...`
- Fixed bug in queries of type `SELECT * FROM table-list GROUP BY ...` and `SELECT DISTINCT * FROM`
- Fixed bug with the `--slow-log` when logging an administrator command (like `FLUSH TABLES`).
- Fixed a bug that `OPTIMIZE` of locked and modified table, reported table corruption.
- Fixed a bug in `my_getopt` in handling of special prefixes (`--skip-`, `--enable-`). `--skip-external-locking` didn't work and the bug may have affected other similar options.
- Fixed bug in checking for output file name of the `tee` option.
- Added some more optimisation to use index for `SELECT ... FROM many_tables .. ORDER BY key limit #`
- Fixed problem in `SHOW OPEN TABLES` when a user didn't have access permissions to one of the opened tables.

D.3.12 Changes in release 4.0.3 (26 Aug 2002: Beta)

- Fixed problem with `configure ... --localstatedir=...`
- Cleaned up `mysql.server` script.
- Fixed a bug in `mysqladmin shutdown` when pid file was modified while `mysqladmin` was still waiting for the previous one to disappear. This could happen during a very quick restart and caused `mysqladmin` to hang until `shutdown_timeout` seconds had passed.
- Don't increment warnings when setting `AUTO_INCREMENT` columns to `NULL` in `LOAD DATA INFILE`.
- Fixed all boolean type variables/options to work with the old syntax, e.g. all of these work: `--lower-case-table-names`, `--lower-case-table-names=1`, `-O lower-case-table-names=1`, `--set-variable=lower-case-table-names=1`
- Fixed shutdown problem (SIGTERM signal handling) on Solaris. (Bug from 4.0.2).
- `SHOW MASTER STATUS` now returns an empty set if binary log is not enabled.
- `SHOW SLAVE STATUS` now returns an empty set if slave is not initialised.
- Don't update MyISAM index file on update if not strictly necessary.
- Fixed bug in `SELECT DISTINCT ... FROM many_tables ORDER BY not-used-column`.
- Fixed a bug with `BIGINTs` and quoted strings.
- Added `QUOTE()` function that performs SQL quoting to produce values that can be used as data values in queries.
- Changed variable `DELAY_KEY_WRITE` to an enum to allow one set `DELAY_KEY_WRITE` for all tables without taking down the server.
- Changed behaviour of `IF(condition, column, NULL)` so that it returns the value of the column type.
- Made `safe_mysql` a symlink to `mysqld_safe` in binary distribution.
- Fixed security bug when having an empty database name in the `user.db` table.
- Fixed some problems with `CREATE TABLE ... SELECT function()`.
- `mysqld` now has the option `--temp-pool` enabled by default as this gives better performance with some operating systems.
- Fixed problem with too many allocated alarms on slave when connecting to master many times (normally not a very critical error).
- Fixed hang in `CHANGE MASTER TO` if the slave thread died very quickly.
- Big cleanup in replication code (less logging, better error messages, etc..)
- If the `--code-file` option is specified, the server calls `setrlimit()` to set the maximum allowed core file size to unlimited, so core files can be generated.
- Fixed bug in query cache after temporary table creation.
- Added `--count=N (-c)` option to `mysqladmin`, to make the program do only N iterations. To be used with `--sleep (-i)`. Useful in scripts.
- Fixed bug in multi-table `UPDATE`: when updating a table, `do_select()` became confused about reading records from a cache.
- Fixed bug in multi-table `UPDATE` when several fields were referenced from a single table

- Fixed bug in truncating nonexisting table.
- Fixed bug in REVOKE that caused user resources to be randomly set.
- Fixed bug in GRANT for the new CREATE TEMPORARY TABLE privilege.
- Fixed bug in multi-table DELETE when tables are re-ordered in the table initialisation method and ref_lengths are of different sizes.
- Fixed two bugs in SELECT DISTINCT with large tables.
- Fixed bug in query cache initialisation with very small query cache size.
- Allow DEFAULT with INSERT statement.
- The startup parameters `myisam_max_sort_file_size` and `myisam_max_extra_sort_file_size` are now given in bytes, not megabytes.
- External system locking of MyISAM/ISAM files is now turned off by default. One can turn this on with `--external-locking`. (For most users this is never needed).
- Fixed core dump bug with `INSERT ... SET db_name.table_name.colname=''`.
- Fixed client hangup bug when using some SQL commands with wrong syntax.
- Fixed a timing bug in DROP DATABASE
- New SET [GLOBAL | SESSION] syntax to change thread-specific and global server variables at runtime.
- Added variable `slave_compressed_protocol`.
- Renamed variable `query_cache_startup_type` to `query_cache_type`, `myisam_bulk_insert_tree_size` to `bulk_insert_buffer_size`, `record_buffer` to `read_buffer_size` and `record_rnd_buffer` to `record_rnd_buffer_size`.
- Renamed some SQL variables, but old names will still work until 5.0. See [from-3.23-snt \[Upgrading-from-3.23\]](#), page [from-3.23-pg](#).
- Renamed `--skip-locking` to `--skip-external-locking`.
- Removed unused variable `query_buffer_size`.
- Fixed a bug that made the pager option in the mysql client non-functional.
- Added full AUTO_INCREMENT support to MERGE tables.
- Extended LOG() function to accept an optional arbitrary base parameter. See [\(undefined\) \[Mathematical functions\]](#), page [\(undefined\)](#).
- Added LOG2() function (useful for finding out how many bits a number would require for storage).
- Added LN() natural logarithm function for compatibility with other databases. It is synonymous with LOG(X).

D.3.13 Changes in release 4.0.2 (01 Jul 2002)

- Cleaned up NULL handling for default values in DESCRIBE `table_name`.
- Fixed `truncate()` to round up negative values to the nearest integer.
- Changed `--chroot=path` option to execute `chroot()` immediately after all options have been parsed.
- Don't allow database names that contain `'\'`.
- `lower_case_table_names` now also affects database names.

- Added XOR operator (logical and bitwise XOR) with `^` as a synonym for bitwise XOR.
- Added function `IS_FREE_LOCK("lock_name")`. Based on code contributed by Hartmut Holzgraefe hartmut@six.de.
- Removed `mysql_ssl_clear()` from C API, as it was not needed.
- DECIMAL and NUMERIC types can now read exponential numbers.
- Added `SHA1()` function to calculate 160 bit hash value as described in RFC 3174 (Secure Hash Algorithm). This function can be considered a cryptographically more secure equivalent of `MD5()`. See [\[Miscellaneous functions\]](#), page [\[undefined\]](#).
- Added `AES_ENCRYPT()` and `AES_DECRYPT()` functions to perform encryption according to AES standard (Rijndael). See [\[Miscellaneous functions\]](#), page [\[undefined\]](#).
- Added `--single-transaction` option to `mysqldump`, allowing a consistent dump of InnoDB tables. See [\[mysqldump\]](#), page [\[undefined\]](#).
- Fixed bug in `innodb_log_group_home_dir` in `SHOW VARIABLES`.
- Fixed a bug in optimiser with merge tables when non-unique values are used in summing up (causing crashes).
- Fixed a bug in optimiser when a range specified makes index grouping impossible (causing crashes).
- Fixed a rare bug when FULLTEXT index is present and no tables are used.
- Added privileges `CREATE TEMPORARY TABLES`, `EXECUTE`, `LOCK TABLES`, `REPLICATION CLIENT`, `REPLICATION SLAVE`, `SHOW DATABASES` and `SUPER`. To use these, you must have run the `mysql_fix_privilege_tables` script after upgrading.
- Fixed query cache align data bug.
- Fixed mutex bug in replication when reading from master fails.
- Added missing mutex in `TRUNCATE TABLE`; This fixes some core dump/hangup problems when using `TRUNCATE TABLE`.
- Fixed bug in multi-table `DELETE` when optimiser uses only indices.
- Fixed that `ALTER TABLE table_name RENAME new_table_name` is as fast as `RENAME TABLE`.
- Fixed bug in `GROUP BY` with two or more fields, where at least one field can contain NULL values.
- Use Turbo Boyer-Moore algorithm to speed up `LIKE "%keyword%"` searches.
- Fixed bug in `DROP DATABASE` with symlink.
- Fixed crash in `REPAIR ... USE_FRM`.
- Fixed bug in `EXPLAIN` with `LIMIT offset != 0`.
- Fixed bug in phrase operator `"..."` in boolean full-text search.
- Fixed bug that caused duplicated rows when using truncation operator `*` in boolean full-text search.
- Fixed bug in truncation operator of boolean full-text search (wrong results when there are only `+word*s` in the query).
- Fixed bug in boolean full-text search that caused a crash when an identical `MATCH` expression that did not use an index appeared twice.

- Query cache is now automatically disabled in `mysqldump`.
- Fixed problem on Windows 98 that made sending of results very slow.
- Boolean full-text search weighting scheme changed to something more reasonable.
- Fixed bug in boolean full-text search that caused MySQL to ignore queries of `ft_min_word_len` characters.
- Boolean full-text search now supports “phrase searches”.
- New configure option `--without-query-cache`.
- Memory allocation strategy for “root memory” changed. Block size now grows with number of allocated blocks.
- `INET_NTOA()` now returns NULL if you give it an argument that is too large (greater than the value corresponding to 255.255.255.255).
- Fix `SQL_CALC_FOUND_ROWS` to work with UNIONS. It will work only if the first `SELECT` has this option and if there is global `LIMIT` for the entire statement. For the moment, this requires using parentheses for individual `SELECT` queries within the statement.
- Fixed bug in `SQL_CALC_FOUND_ROWS` and `LIMIT`.
- Don’t give an error for `CREATE TABLE ... (... VARCHAR(0))`.
- Fixed `SIGINT` and `SIGQUIT` problems in ‘`mysql.cc`’ on Linux with some `glibc` versions.
- Fixed bug in ‘`convert.cc`’, which is caused by having an incorrect `net_store_length()` linked in the `CONVERT::store()` method.
- `DOUBLE` and `FLOAT` columns now honor the `UNSIGNED` flag on storage.
- InnoDB now retains foreign key constraints through `ALTER TABLE` and `CREATE/DROP INDEX`.
- InnoDB now allows foreign key constraints to be added through the `ALTER TABLE` syntax.
- InnoDB tables can now be set to automatically grow in size (`autoextend`).
- Added `--ignore-lines=n` option to `mysqlimport`. This has the same effect as the `IGNORE n LINES` clause for `LOAD DATA`.
- Fixed bug in `UNION` with last offset being transposed to total result set.
- `REPAIR ... USE_FRM` added.
- Fixed that `DEFAULT_SELECT_LIMIT` is always imposed on `UNION` result set.
- Fixed that some `SELECT` options can appear only in the first `SELECT`.
- Fixed bug with `LIMIT` with `UNION`, where last select is in the braces.
- Fixed that full-text works fine with `UNION` operations.
- Fixed bug with indexless boolean full-text search.
- Fixed bug that sometimes appeared when full-text search was used with “const” tables.
- Fixed incorrect error value when doing a `SELECT` with an empty `HEAP` table.
- Use `ORDER BY column DESC` now sorts NULL values first. (In other words, NULL values sort first in all cases, whether or not `DESC` is specified.). This is changed back in 4.0.10.
- Fixed bug in `WHERE key_name='constant' ORDER BY key_name DESC`.
- Fixed bug in `SELECT DISTINCT ... ORDER BY DESC` optimisation.
- Fixed bug in `... HAVING 'GROUP_FUNCTION'(xxx) IS [NOT] NULL`.

- Fixed bug in truncation operator for boolean full-text search.
- Allow value of `--user=#` option for `mysqld` to be specified as a numeric user ID.
- Fixed a bug where `SQL_CALC_ROWS` returned an incorrect value when used with one table and `ORDER BY` and with InnoDB tables.
- Fixed that `SELECT 0 LIMIT 0` doesn't hang thread.
- Fixed some problems with `USE/IGNORE INDEX` when using many keys with the same start column.
- Don't use table scan with BerkeleyDB and InnoDB tables when we can use an index that covers the whole row.
- Optimised InnoDB sort-buffer handling to take less memory.
- Fixed bug in multi-table `DELETE` and InnoDB tables.
- Fixed problem with `TRUNCATE` and InnoDB tables that produced the error `Can't execute the given command because you have active locked tables or an active transaction`.
- Added `NO_UNSIGNED_SUBTRACTION` to the set of flags that may be specified with the `--sql-mode` option for `mysqld`. It disables unsigned arithmetic rules when it comes to subtraction. (This will make MySQL 4.0 behave more closely to 3.23 with `UNSIGNED` columns).
- The result returned for all bit functions (`|`, `<<`, ...) is now of type `unsigned integer`.
- Added detection of `nan` values in MyISAM to make it possible to repair tables with `nan` in float or double columns.
- Fixed new bug in `myisamchk` where it didn't correctly update number of "parts" in the MyISAM index file.
- Changed to use `autoconf 2.52` (from `autoconf 2.13`).
- Fixed optimisation problem where the MySQL Server was in "preparing" state for a long time when selecting from an empty table which had contained a lot of rows.
- Fixed bug in complicated join with `const` tables. This fix also improves performance a bit when referring to another table from a `const` table.
- First pre-version of multi-table `UPDATE` statement.
- Fixed bug in multi-table `DELETE`.
- Fixed bug in `SELECT CONCAT(argument_list) ... GROUP BY 1`.
- `INSERT ... SELECT` did a full rollback in case of an error. Fixed so that we only roll back the last statement in the current transaction.
- Fixed bug with empty expression for boolean full-text search.
- Fixed core dump bug in updating full-text key from/to `NULL`.
- ODBC compatibility: Added `BIT_LENGTH()` function.
- Fixed core dump bug in `GROUP BY BINARY` column.
- Added support for `NULL` keys in `HEAP` tables.
- Use index for `ORDER BY` in queries of type: `SELECT * FROM t WHERE key_part1=1 ORDER BY key_part1 DESC, key_part2 DESC`
- Fixed bug in `FLUSH QUERY CACHE`.

- Added `CAST()` and `CONVERT()` functions. The `CAST` and `CONVERT` functions are nearly identical and mainly useful when you want to create a column with a specific type in a `CREATE ... SELECT` statement. For more information, read [\[Cast Functions\]](#), page [\[undefined\]](#).
- `CREATE ... SELECT` on `DATE` and `TIME` functions now create columns of the expected type.
- Changed order in which keys are created in tables.
- Added new columns `Null` and `Index_type` to `SHOW INDEX` output.
- Added `--no-beep` and `--prompt` options to `mysql` command-line client.
- New feature: management of user resources.


```
GRANT ... WITH MAX_QUERIES_PER_HOUR N1
             MAX_UPDATES_PER_HOUR N2
             MAX_CONNECTIONS_PER_HOUR N3;
```

 See [\[User resources\]](#), page [\[undefined\]](#).
- Added `mysql_secure_installation` to the `'scripts/'` directory.

D.3.14 Changes in release 4.0.1 (23 Dec 2001)

- Added `system` command to `mysql`.
- Fixed bug when `HANDLER` was used with some unsupported table type.
- `mysqldump` now puts `ALTER TABLE nome_tabela DISABLE KEYS` and `ALTER TABLE nome_tabela ENABLE KEYS` in the sql dump.
- Added `mysql_fix_extensions` script.
- Fixed stack overrun problem with `LOAD DATA FROM MASTER` on OSF/1.
- Fixed shutdown problem on HP-UX.
- Added `DES_ENCRYPT()` and `DES_DECRYPT()` functions.
- Added `FLUSH DES_KEY_FILE` statement.
- Added `--des-key-file` option to `mysqld`.
- `HEX(string)` now returns the characters in `string` converted to hexadecimal.
- Fixed problem with `GRANT` when using `lower_case_table_names=1`.
- Changed `SELECT ... IN SHARE MODE` to `SELECT ... LOCK IN SHARE MODE` (as in MySQL 3.23).
- A new query cache to cache results from identical `SELECT` queries.
- Fixed core dump bug on 64-bit machines when it got an incorrect communication packet.
- `MATCH ... AGAINST(... IN BOOLEAN MODE)` can now work without `FULLTEXT` index.
- Fixed slave to replicate from 3.23 master.
- Miscellaneous replication fixes/cleanup.
- Got shutdown to work on Mac OS X.
- Added `'myisam/ft_dump'` utility for low-level inspection of `FULLTEXT` indexes.
- Fixed bug in `DELETE ... WHERE ... MATCH ...`.

- Added support for `MATCH ... AGAINST(... IN BOOLEAN MODE)`. **Note: you must rebuild your tables with `ALTER TABLE tablename TYPE=MyISAM` to be able to use boolean full-text search.**
- `LOCATE()` and `INSTR()` are now case-sensitive if either argument is a binary string.
- Changed `RAND()` initialisation so that `RAND(N)` and `RAND(N+1)` are more distinct.
- Fixed core dump bug in `UPDATE ... ORDER BY`.
- Changed `INSERT INTO ... SELECT` to stop on errors by default.
- Ignore `DATA DIRECTORY` and `INDEX DIRECTORY` directives on Windows.
- Added boolean full-text search code. It should be considered early alpha.
- Extended `MODIFY` and `CHANGE` in `ALTER TABLE` to accept the `FIRST` and `AFTER` keywords.
- Indexes are now used with `ORDER BY` on a whole InnoDB table.

D.3.15 Changes in release 4.0.0 (Oct 2001: Alpha)

- Added `--xml` option to `mysql` for producing XML output.
- Added full-text variables `ft_min_word_len`, `ft_max_word_len`, and `ft_max_word_len_for_sort`.
- Added documentation for `libmysqld`, the embedded MySQL server library. Also added example programs (a `mysql` client and `mysqltest` test program) which use `libmysqld`.
- Removed all Gemini hooks from MySQL server.
- Removed `my_thread_init()` and `my_thread_end()` from `'mysql_com.h'`, and added `mysql_thread_init()` and `mysql_thread_end()` to `'mysql.h'`.
- Support for communication packets > 16M. In 4.0.1 we will extend MyISAM to be able to handle these.
- Secure connections (with SSL).
- Unsigned `BIGINT` constants now work. `MIN()` and `MAX()` now handle signed and unsigned `BIGINT` numbers correctly.
- New character set `latin1_de` which provides correct German sorting.
- `STRCMP()` now uses the current character set when doing comparisons, which means that the default comparison behaviour now is case-insensitive.
- `TRUNCATE TABLE` and `DELETE FROM nome_tabela` are now separate functions. One bonus is that `DELETE FROM nome_tabela` now returns the number of deleted rows, rather than zero.
- `DROP DATABASE` now executes a `DROP TABLE` on all tables in the database, which fixes a problem with InnoDB tables.
- Added support for `UNION`.
- Added support for multi-table `DELETE` operations.
- A new `HANDLER` interface to MyISAM tables.
- Added support for `INSERT` on `MERGE` tables. Patch from Benjamin Pflugmann.
- Changed `WEEK(#,0)` to match the calendar in the USA.
- `COUNT(DISTINCT)` is about 30% faster.
- Speed up all internal list handling.

- Speed up IS NULL, ISNULL() and some other internal primitives.
- Full-text index creation now is much faster.
- Tree-like cache to speed up bulk inserts and `myisam_bulk_insert_tree_size` variable.
- Searching on packed (CHAR/VARCHAR) keys is now much faster.
- Optimised queries of type: `SELECT DISTINCT * from nome_tabela ORDER by key_part1 LIMIT #`.
- `SHOW CREATE TABLE` now shows all table attributes.
- `ORDER BY ... DESC` can now use keys.
- `LOAD DATA FROM MASTER` “automatically” sets up a slave.
- Renamed `safe_mysqld` to `mysqld_safe` to make this name more in line with other MySQL scripts/commands.
- Added support for symbolic links to MyISAM tables. Symlink handling is now enabled by default for Windows.
- Added `SQL_CALC_FOUND_ROWS` and `FOUND_ROWS()`. This makes it possible to know how many rows a query would have returned without a `LIMIT` clause.
- Changed output format of `SHOW OPEN TABLES`.
- Allow `SELECT expression LIMIT`
- Added `IDENTITY` as a synonym for `AUTO_INCREMENT` (like Sybase).
- Added `ORDER BY` syntax to `UPDATE` and `DELETE`.
- `SHOW INDEXES` is now a synonym for `SHOW INDEX`.
- Added `ALTER TABLE nome_tabela DISABLE KEYS` and `ALTER TABLE nome_tabela ENABLE KEYS` commands.
- Allow use of `IN` as a synonym for `FROM` in `SHOW` commands.
- Implemented “repair by sort” for `FULLTEXT` indexes. `REPAIR TABLE`, `ALTER TABLE`, and `OPTIMIZE TABLE` for tables with `FULLTEXT` indexes are now up to 100 times faster.
- Allow SQL-99 syntax `X'hexadecimal-number'`.
- Cleaned up global lock handling for `FLUSH TABLES WITH READ LOCK`.
- Fixed problem with `DATETIME = constant` in `WHERE` optimisation.
- Added `--master-data` and `--no-autocommit` options to `mysqldump`. (Thanks to Brian Aker for this.)
- Added script `mysql_explain_log.sh` to distribution. (Thanks to mobile.de).

D.4 Changes in release 3.23.x (Recent; still supported)

Please note that since release 4.0 is now production level, only critical fixes are done in the 3.23 release series. You are recommended to upgrade when possible, to take advantage of all speed and feature improvements in 4.0. See [from-3.23-snt \[Upgrading-from-3.23\]](#), [page from-3.23-pg](#).

The 3.23 release has several major features that are not present in previous versions. We have added three new table types:

MyISAM A new ISAM library which is tuned for SQL and supports large files.

InnoDB A transaction-safe storage engine that supports row level locking, and many Oracle-like features.

BerkeleyDB or BDB

Uses the Berkeley DB library from Sleepycat Software to implement transaction-safe tables.

Note that only MyISAM is available in the standard binary distribution.

The 3.23 release also includes support for database replication between a master and many slaves, full-text indexing, and much more.

All new features are being developed in the 4.x version. Only bug fixes and minor enhancements to existing features will be added to 3.23.

The replication code and BerkeleyDB code is still not as tested and as the rest of the code, so we will probably need to do a couple of future releases of 3.23 with small fixes for this part of the code. As long as you don't use these features, you should be quite safe with MySQL 3.23!

Note that the above doesn't mean that replication or Berkeley DB don't work. We have done a lot of testing of all code, including replication and BDB without finding any problems. It only means that not as many users use this code as the rest of the code and because of this we are not yet 100% confident in this code.

D.4.1 Changes in release 3.23.57 (not released yet)

Fixed a security problem with `SELECT` and wildcarded select list, when user only had partial column `SELECT` privileges on the table.

Fixed unlikely problem in optimising `WHERE` clause with constant expression like in `WHERE 1 AND (a=1 AND b=1)`.

Fixed problem on IA64 with timestamps which caused `mysqlbinlog` to fail.

The default option for `innodb_flush_log_at_trx_commit` was changed from 0 to 1 to make InnoDB tables ACID by default. See [\[InnoDB start\]](#), page [\[undefined\]](#).

Fixed problem with too many allocated alarms on slave when connecting to master many times (normally not a very critical error).

Fixed a bug in replication of temporary tables (bug #183).

Fixed 64 bit bug that affected at least AMD hammer systems.

Fixed a bug when doing `LOAD DATA INFILE IGNORE` : when reading the binary log, `mysqlbinlog` and the replication code read `REPLACE` instead of `IGNORE`. This could make the slave's table become different from the master's table (bug #218).

Fixed a bug in MyISAM when a row is inserted into a table with a large number of `NULL` columns. Bug was caused by wrong calculation of the record length, as the space required for storage of `NULL` bits was not added to the total record length.

Binary log was not locked during a `TRUNCATE table_name` or `DELETE FROM table_name` which could cause an `INSERT` to `table_name` to be written to the binary log before the `TRUNCATE/ DELETE` command.

Fixed rare bug in `UPDATE` of InnoDB tables where one row could be updated multiple times.

Give an error for empty table and column names.

Changed `PROCEDURE ANALYSE()` to report `DATE` instead of `NEWDATE`.

Changed `PROCEDURE ANALYSE(#)` to restrict number of values in enum to `#` also for string values.

`mysqldump` no longer silently deletes the binlogs when called with `--master-data` or `--first-slave`; while this behaviour was convenient for some users, others may suffer from it. Now one has to explicitly ask for this deletion with the new `--delete-master-logs` option.

Fixed a bug in `mysqldump` when it was called with `--master-data`: the `CHANGE MASTER TO` commands appended to the SQL dump had wrong coordinates (bug #159).

D.4.2 Changes in release 3.23.56 (13 Mar 2003)

Fixed ‘`mysqld`’ crash on extremely small values of `sort_buffer` variable.

Fixed a bug in privilege system for `GRANT UPDATE` on column level.

Fixed a rare bug when using a date in `HAVING` with `GROUP BY`.

Fixed checking of random part of `WHERE` clause (Bug #142).

Fixed MySQL (and ‘`myisamchk`’) crash on artificially corrupted `.MYI` files.

Security enhancement: ‘`mysqld`’ no longer reads options from world-writable config files.

Security enhancement: ‘`mysqld`’ and ‘`safe_mysqld`’ now only use the first `--user` option specified on the command line. (Normally this comes from ‘`/etc/my.cnf`’)

Security enhancement: Don’t allow `BACKUP TABLE` to overwrite existing files.

Fixed unlikely deadlock bug when one thread did a `LOCK TABLE` and another thread did a `DROP TABLE`. In this case one could do a `KILL` on one of the threads to resolve the deadlock.

`LOAD DATA INFILE` was not replicated by slave if `replicate_*_table` was set on the slave.

Fixed a bug in handling `CHAR(0)` columns that could cause wrong results from the query.

Fixed a bug in `SHOW VARIABLES` on 64-bit platforms. The bug was caused by wrong declaration of variable `server_id`.

The `Comment` column in `SHOW TABLE STATUS` now reports that it can contain `NULL` values (which is the case for a crashed ‘`.frm`’ file).

Fixed the `rpl_rotate_logs` test to not fail on certain platforms (e.g. Mac OS X) due to a too long file name (changed `slave-master-info.opt` to `.slave-mi`).

Fixed a problem with `BLOB NOT NULL` columns used with `IS NULL`.

Fixed bug in `MAX()` optimisation in `MERGE` tables.

Better `RAND()` initialization for new connections.

Fixed bug with connect timeout. This bug was manifested on OS’s with `poll()` system call, which resulted in timeout the value specified as it was executed in both `select()` and `poll()`.

Fixed bug in `SELECT * FROM table WHERE datetime1 IS NULL OR datetime2 IS NULL`.

Fixed bug in using aggregate functions as argument for `INTERVAL`, `CASE`, `FIELD`, `CONCAT_WS`, `ELT` and `MAKE_SET` functions.

When running with `--lower-case-table-names` (default on windows) and you had tables or databases with mixed case on disk, then executing `SHOW TABLE STATUS` followed with `DROP DATABASE` or `DROP TABLE` could fail with `Errcode 13`.

Fixed bug in logging to binary log (which affects replication) a query that inserts a `NULL` in an `auto_increment` field and also uses `LAST_INSERT_ID()`.

Fixed bug in `mysqladmin --relative`.

On some 64 bit systems, `show status` reported a strange number for `Open_files` and `Open_streams`.

D.4.3 Changes in release 3.23.55 (23 Jan 2003)

Fixed double `free`'d pointer bug in `mysql_change_user()` handling, that enabled a specially hacked version of MySQL client to crash `mysqld`. **Note**, that one needs to login to the server by using a valid user account to be able to exploit this bug.

Fixed bug with the `--slow-log` when logging an administrator command (like `FLUSH TABLES`).

Fixed bug in `GROUP BY` when used on `BLOB` column with `NULL` values.

Fixed a bug in handling `NULLs` in `CASE ... WHEN ...`.

Bugfix for `--chroot` (see 3.23.54-snt [News-3.23.54], page 3.23.54-pg) is reverted. Unfortunately, there is no way to make it to work, without introducing backward-incompatible changes in `'my.cnf'`. Those who need `--chroot` functionality, should upgrade to MySQL 4.0. (The fix in the 4.0 branch did not break backward-compatibility).

Make `--lower-case-table-names` default on Mac OS X as the file system is case insensitive.

Fixed a bug in `'scripts/mysqld_safe.sh'` in `NOHUP_NICENESS` testing.

Transactions in `AUTOCOMMIT=0` mode didn't rotate binary log.

Fixed a bug in `scripts/make_binary_distribution` that resulted in a remaining `@HOSTNAME@` variable instead of replacing it with the correct path to the `hostname` binary.

Fixed a very unlikely bug that could cause `SHOW PROCESSLIST` to core dump in `pthread_mutex_unlock()` if a new thread was connecting.

Forbid `SLAVE STOP` if the thread executing the query has locked tables. This removes a possible deadlock situation.

D.4.4 Changes in release 3.23.54 (05 Dec 2002)

Fixed a bug, that allowed to crash `mysqld` with a specially crafted packet.

Fixed a rare crash (double `free`'d pointer) when altering a temporary table.

Fixed buffer overrun in `libmysqlclient` library that allowed malicious MySQL server to crash the client application.

Fixed security-related bug in `mysql_change_user()` handling. All users are strongly recommended to upgrade to the version 3.23.54.

Fixed bug that prevented `--chroot` command-line option of `mysqld` from working.

Fixed bug that made `OPTIMIZE TABLE` to corrupt the table under some rare circumstances.

Fixed `mysqlcheck` so it can deal with table names containing dashes.

Fixed shutdown problem on Mac OS X.

Fixed bug with comparing an indexed `NULL` field with `<=> NULL`.

Fixed bug that caused `IGNORE INDEX` and `USE INDEX` sometimes to be ignored.

Fixed rare core dump problem in complicated `GROUP BY` queries that didn't return any result.

Fixed a bug where `MATCH ... AGAINST () >=0` was treated as if it was `>`.

Fixed core dump in `SHOW PROCESSLIST` when running with an active slave (unlikely timing bug).

Allow one to start multiple MySQL servers on windows (code backported from 4.0.2).

One can create `TEMPORARY MERGE` tables now.

Fixed that `--core-file` works on Linux (at least on kernel 2.4.18).

Fixed a problem with BDB and `ALTER TABLE`.

Fixed reference to freed memory when doing complicated `GROUP BY ... ORDER BY` queries. Symptom was that `mysqld` died in function `send_fields`.

Allocate heap rows in smaller blocks to get better memory usage.

Fixed memory allocation bug when storing `BLOB` values in internal temporary tables used for some (unlikely) `GROUP BY` queries.

Fixed a bug in key optimising handling where the expression `WHERE column_name = key_column_name` was calculated as true for `NULL` values.

Fixed core dump bug when doing `LEFT JOIN ... WHERE key_column=NULL`.

Fixed MyISAM crash when using dynamic-row tables with huge numbers of packed fields.

Updated source tree to be built using `automake 1.5` and `libtool 1.4`.

D.4.5 Changes in release 3.23.53 (09 Oct 2002)

- Fixed crash when `SHOW INNODB STATUS` was used and `skip-innodb` was defined.
- Fixed possible memory corruption bug in binary log file handling when slave rotated the logs (only affected 3.23, not 4.0).
- Fixed problem in `LOCK TABLES` on windows when one connects to a database that contains upper case letters.
- Fixed that `--skip-show-databases` doesn't reset the `--port` option.
- Small fix in `safe_mysqld` for some shells.
- Fixed that `FLUSH STATUS` doesn't reset `delayed_insert_threads`.
- Fixed core dump bug when using the `BINARY` cast on a `NULL` value.
- Fixed race condition when someone did a `GRANT` at the same time a new user logged in or did a `USE DATABASE`.
- Fixed bug in `ALTER TABLE` and `RENAME TABLE` when running with `-O lower_case_table_names=1` (typically on windows) when giving the table name in uppercase.

- Fixed that `-O lower_case_table_names=1` also converts database names to lower case.
- Fixed unlikely core dump with `SELECT ... ORDER BY ... LIMIT`.
- Changed AND/OR to report that they can return NULL. This fixes a bug in GROUP BY on AND/OR expressions that return NULL.
- Fixed a bug that OPTIMIZE of locked and modified MyISAM table, reported table corruption.
- Fixed a BDB-related ALTER TABLE bug with dropping a column and shutting down immediately thereafter.
- Fixed problem with `configure ... --localstatedir=...`
- Fixed problem with UNSIGNED BIGINT on AIX (again).
- Fixed bug in `pthread_mutex_trylock()` on HP-UX 11.0.
- Multithreaded stress tests for InnoDB.

D.4.6 Changes in release 3.23.52 (14 Aug 2002)

- Wrap BEGIN/COMMIT around transaction in the binary log. This makes replication honour transactions.
- Fixed security bug when having an empty database name in the `user.db` table.
- Changed initialisation of `RND()` to make it less predicatable.
- Fixed problem with GROUP BY on result with expression that created a BLOB field.
- Fixed problem with GROUP BY on columns that have NULL values. To solve this we now create an MyISAM temporary table when doing a GROUP BY on a possible NULL item. From MySQL 4.0.5 we can use in memory HEAP tables for this case.
- Fixed problem with privilege tables when downgrading from 4.0.2 to 3.23.
- Fixed thread bug in SLAVE START, SLAVE STOP and automatic repair of MyISAM tables that could cause table cache to be corrupted.
- Fixed possible thread related key-cache-corruption problem with OPTIMIZE TABLE and REPAIR TABLE.
- Added name of 'administrator command' logs.
- Fixed bug with creating an auto-increment value on second part of a UNIQUE() key where first part could contain NULL values.
- Don't write slave-timeout reconnects to the error log.
- Fixed bug with slave net read timeouting
- Fixed a core-dump bug with MERGE tables and MAX() function.
- Fixed bug in ALTER TABLE with BDB tables.
- Fixed bug when logging LOAD DATA INFILE to binary log with no active database.
- Fixed a bug in range optimiser (causing crashes).
- Fixed possible problem in replication when doing DROP DATABASE on a database with InnoDB tables.
- Fixed that `mysql_info()` returns 0 for 'Duplicates' when using INSERT DELAYED IGNORE.
- Added `-DHAVE_BROKEN_REALPATH` to the Mac OS X (darwin) compile options in 'configure.in' to fix a failure under high load.

D.4.7 Changes in release 3.23.51 (31 May 2002)

- Fix bug with closing tags missing slash for `mysqldump` XML output.
- Remove end space from ENUM values. (This fixed a problem with `SHOW CREATE TABLE`.)
- Fixed bug in `CONCAT_WS()` that cut the result.
- Changed name of server variables `Com_show_master_stat` to `Com_show_master_status` and `Com_show_slave_stat` to `Com_show_slave_status`.
- Changed handling of `gethostbyname()` to make the client library thread-safe even if `gethostbyname_r` doesn't exist.
- Fixed core-dump problem when giving a wrong password string to `GRANT`.
- Fixed bug in `DROP DATABASE` with symlinked directory.
- Fixed optimisation problem with `DATETIME` and value outside `DATETIME` range.
- Removed Sleepycat's BDB doc files from the source tree, as they're not needed (MySQL covers BDB in its own documentation).
- Fixed MIT-pthreads to compile with `glibc 2.2` (needed for `make dist`).
- Fixed the `FLOAT(X+1,X)` is not converted to `FLOAT(X+2,X)`. (This also affected `DECIMAL`, `DOUBLE` and `REAL` types)
- Fixed the result from `IF()` is case in-sensitive if the second and third arguments are case sensitive.
- Fixed core dump problem on OSF/1 in `gethostbyname_r`.
- Fixed that underflowed decimal fields are not zero filled.
- If we get an overflow when inserting `'+11111'` for `DECIMAL(5,0) UNSIGNED` columns, we will just drop the sign.
- Fixed optimisation bug with `ISNULL(expression_which_cannot_be_null)` and `ISNULL(constant_expression)`.
- Fixed host lookup bug in the `glibc` library that we used with the 3.23.50 Linux-x86 binaries.

D.4.8 Changes in release 3.23.50 (21 Apr 2002)

- Fixed buffer overflow problem if someone specified a too long `datadir` parameter to `mysqld`
- Add missing `<row>` tags for `mysqldump` XML output.
- Fixed problem with `crash-me` and `gcc 3.0.4`.
- Fixed that `@@unknown_variable` doesn't hang server.
- Added `@@VERSION` as a synonym for `VERSION()`.
- `SHOW VARIABLES LIKE 'xxx'` is now case-insensitive.
- Fixed timeout for `GET_LOCK()` on HP-UX with DCE threads.
- Fixed memory allocation bug in the `glibc` library used to build Linux binaries, which caused `mysqld` to die in `'free()'`.
- Fixed `SIGINT` and `SIGQUIT` problems in `mysql`.
- Fixed bug in character table converts when used with big (`> 64K`) strings.

- InnoDB now retains foreign key constraints through ALTER TABLE and CREATE/DROP INDEX.
- InnoDB now allows foreign key constraints to be added through the ALTER TABLE syntax.
- InnoDB tables can now be set to automatically grow in size (autoextend).
- Our Linux RPMS and binaries are now compiled with gcc 3.0.4, which should make them a bit faster.
- Fixed some buffer overflow problems when reading startup parameters.
- Because of problems on shutdown we have now disabled named pipes on windows by default. One can enable named pipes by starting mysqld with `--enable-named-pipe`.
- Fixed bug when using `WHERE key_column = 'J'` or `key_column='j'`.
- Fixed core-dump bug when using `--log-bin` with `LOAD DATA INFILE` without an active database.
- Fixed bug in `RENAME TABLE` when used with `lower_case_table_names=1` (default on Windows).
- Fixed unlikely core-dump bug when using `DROP TABLE` on a table that was in use by a thread that also used queries on only temporary tables.
- Fixed problem with `SHOW CREATE TABLE` and `PRIMARY KEY` when using 32 indexes.
- Fixed that one can use `SET PASSWORD` for the anonymous user.
- Fixed core dump bug when reading client groups from option files using `mysql_options()`.
- Memory leak (16 bytes per every **corrupted** table) closed.
- Fixed binary builds to use `--enable-local-infile`.
- Update source to work with new version of bison.
- Updated shell scripts to now agree with new POSIX standard.
- Fixed bug where `DATE_FORMAT()` returned empty string when used with `GROUP BY`.

D.4.9 Changes in release 3.23.49

- Don't give warning for a statement that is only a comment; this is needed for `mysqldump --disable-keys` to work.
- Fixed unlikely caching bug when doing a join without keys. In this case the last used field for a table always returned NULL.
- Added options to make `LOAD DATA LOCAL INFILE` more secure.
- MySQL binary release 3.23.48 for Linux contained a new `glibc` library, which has serious problems under high load and Red Hat 7.2. The 3.23.49 binary release doesn't have this problem.
- Fixed shutdown problem on NT.

D.4.10 Changes in release 3.23.48 (07 Feb 2002)

- Added `--xml` option to `mysqldump` for producing XML output.
- Changed to use `autoconf 2.52` (from `autoconf 2.13`)
- Fixed bug in complicated join with `const` tables.

- Added internal safety checks for InnoDB.
- Some InnoDB variables were always shown in `SHOW VARIABLES` as `OFF` on high-byte-first systems (like SPARC).
- Fixed problem with one thread using an InnoDB table and another thread doing an `ALTER TABLE` on the same table. Before that, `mysqld` could crash with an assertion failure in `'row0row.c'`, line 474.
- Tuned the InnoDB SQL optimiser to favor index searches more often over table scans.
- Fixed a performance problem with InnoDB tables when several large `SELECT` queries are run concurrently on a multiprocessor Linux computer. Large CPU-bound `SELECT` queries will now also generally run faster on all platforms.
- If MySQL binlogging is used, InnoDB now prints after crash recovery the latest MySQL binlog name and the offset InnoDB was able to recover to. This is useful, for example, when resynchronising a master and a slave database in replication.
- Added better error messages to help in installation problems of InnoDB tables.
- It is now possible to recover MySQL temporary tables that have become orphaned inside the InnoDB tablespace.
- InnoDB now prevents a `FOREIGN KEY` declaration where the signedness is not the same in the referencing and referenced integer columns.
- Calling `SHOW CREATE TABLE` or `SHOW TABLE STATUS` could cause memory corruption and make `mysqld` crash. Especially at risk was `mysqldump`, because it frequently calls `SHOW CREATE TABLE`.
- If inserts to several tables containing an `AUTO_INCREMENT` column were wrapped inside one `LOCK TABLES`, InnoDB asserted in `'lock0lock.c'`.
- In 3.23.47 we allowed several `NULL` values in a `UNIQUE` secondary index for an InnoDB table. But `CHECK TABLE` was not relaxed: it reports the table as corrupt. `CHECK TABLE` no longer complains in this situation.
- `SHOW GRANTS` now shows `REFERENCES` instead of `REFERENCE`.

D.4.11 Changes in release 3.23.47 (27 Dec 2001)

- Fixed bug when using the following construct: `SELECT ... WHERE key=@var_name OR key=@var_name2`
- Restrict InnoDB keys to 500 bytes.
- InnoDB now supports `NULL` in keys.
- Fixed shutdown problem on HP-UX. (Introduced in 3.23.46)
- Fixed core dump bug in replication when using `SELECT RELEASE_LOCK()`.
- Added new command: `DO expression, [expression]`
- Added `slave-skip-errors` option.
- Added statistics variables for all MySQL commands. (`SHOW STATUS` is now much longer.)
- Fixed default values for InnoDB tables.
- Fixed that `GROUP BY expr DESC` works.
- Fixed bug when using `t1 LEFT JOIN t2 ON t2.key=constant`.

- `mysql_config` now also works with binary (relocated) distributions.

D.4.12 Changes in release 3.23.46 (29 Nov 2001)

- Fixed problem with aliased temporary table replication.
- InnoDB and BDB tables will now use index when doing an `ORDER BY` on the whole table.
- Fixed bug where one got an empty set instead of a `DEADLOCK` error when using BDB tables.
- One can now kill `ANALYZE`, `REPAIR`, and `OPTIMIZE TABLE` when the thread is waiting to get a lock on the table.
- Fixed race condition in `ANALYZE TABLE`.
- Fixed bug when joining with caching (unlikely to happen).
- Fixed race condition when using the binary log and `INSERT DELAYED` which could cause the binary log to have rows that were not yet written to MyISAM tables.
- Changed caching of binary log to make replication slightly faster.
- Fixed bug in replication on Mac OS X.

D.4.13 Changes in release 3.23.45 (22 Nov 2001)

- `(UPDATE|DELETE) ...WHERE MATCH` bugfix.
- shutdown should now work on Darwin (Mac OS X).
- Fixed core dump when repairing corrupted packed MyISAM files.
- `--core-file` now works on Solaris.
- Fix a bug which could cause InnoDB to complain if it cannot find free blocks from the buffer cache during recovery.
- Fixed bug in InnoDB insert buffer B-tree handling that could cause crashes.
- Fixed bug in InnoDB lock timeout handling.
- Fixed core dump bug in `ALTER TABLE` on a `TEMPORARY` InnoDB table.
- Fixed bug in `OPTIMIZE TABLE` that reset index cardinality if it was up to date.
- Fixed problem with `t1 LEFT JOIN t2 ... WHERE t2.date_column IS NULL` when `date_column` was declared as `NOT NULL`.
- Fixed bug with BDB tables and keys on BLOB columns.
- Fixed bug in `MERGE` tables on OS with 32-bit file pointers.
- Fixed bug in `TIME_TO_SEC()` when using negative values.

D.4.14 Changes in release 3.23.44 (31 Oct 2001)

- Fixed `Rows_examined` count in slow query log.
- Fixed bug when using a reference to an `AVG()` column in `HAVING`.
- Fixed that date functions that require correct dates, like `DAYOFYEAR(column)`, will return `NULL` for 0000-00-00 dates.
- Fixed bug in const-propagation when comparing columns of different types. (`SELECT * FROM date_col="2001-01-01" and date_col=time_col`)

- Fixed bug that caused error message `Can't write, because of unique constraint` with some `GROUP BY` queries.
- Fixed problem with `sjis` character strings used within quoted table names.
- Fixed core dump when using `CREATE ... FULLTEXT` keys with other storage engines than `MyISAM`.
- Don't use `signal()` on Windows because this appears to not be 100% reliable.
- Fixed bug when doing `WHERE nome_coluna=NULL` on an indexed column that had `NULL` values.
- Fixed bug when doing `LEFT JOIN ... ON (nome_coluna = constant) WHERE nome_coluna = constant`.
- When using replications, aborted queries that contained `%` could cause a core dump.
- `TCP_NODELAY` was not used on some systems. (Speed problem.)
- Applied portability fixes for OS/2. (Patch by Yuri Dario.)

The following changes are for `InnoDB` tables:

- Add missing `InnoDB` variables to `SHOW VARIABLES`.
- Foreign keys checking is now done for `InnoDB` tables.
- `DROP DATABASE` now works also for `InnoDB` tables.
- `InnoDB` now supports datafiles and raw disk partitions bigger than 4 GB on those operating systems that have big files.
- `InnoDB` calculates better table cardinality estimates for the MySQL optimiser.
- Accent characters in the default character set `latin1` are ordered according to the MySQL ordering.

Note: if you are using `latin1` and have inserted characters whose code is greater than 127 into an indexed `CHAR` column, you should run `CHECK TABLE` on your table when you upgrade to 3.23.44, and drop and reimport the table if `CHECK TABLE` reports an error!

- A new `'my.cnf'` parameter, `innodb_thread_concurrency`, helps in performance tuning in heavily concurrent environments.
- A new `'my.cnf'` parameter, `innodb_fast_shutdown`, speeds up server shutdown.
- A new `'my.cnf'` parameter, `innodb_force_recovery`, helps to save your data in case the disk image of the database becomes corrupt.
- `innodb_monitor` has been improved and a new `innodb_table_monitor` added.
- Increased maximum key length from 500 to 7000 bytes.
- Fixed a bug in replication of `AUTO_INCREMENT` columns with multiple-line inserts.
- Fixed a bug when the case of letters changes in an update of an indexed secondary column.
- Fixed a hang when there are > 24 datafiles.
- Fixed a crash when `MAX(col)` is selected from an empty table, and `col` is not the first column in a multi-column index.
- Fixed a bug in purge which could cause crashes.

D.4.15 Changes in release 3.23.43 (04 Oct 2001)

- Fixed a bug in `INSERT DELAYED` and `FLUSH TABLES` introduced in 3.23.42.
- Fixed unlikely bug, which returned non-matching rows, in `SELECT` with many tables and multi-column indexes and 'range' type.
- Fixed an unlikely core dump bug when doing `EXPLAIN SELECT` when using many tables and `ORDER BY`.
- Fixed bug in `LOAD DATA FROM MASTER` when using table with `CHECKSUM=1`.
- Added unique error message when one gets a `DEADLOCK` during a transaction with BDB tables.
- Fixed problem with BDB tables and `UNIQUE` columns defined as `NULL`.
- Fixed problem with `myisampack` when using pre-space filled `CHAR` columns.
- Applied patch from Yuri Dario for OS/2.
- Fixed bug in `--safe-user-create`.

D.4.16 Changes in release 3.23.42 (08 Sep 2001)

- Fixed problem when using `LOCK TABLES` and BDB tables.
- Fixed problem with `REPAIR TABLE` on MyISAM tables with row lengths in the range from 65517 to 65520 bytes.
- Fixed rare hang when doing `mysqladmin shutdown` when there was a lot of activity in other threads.
- Fixed problem with `INSERT DELAYED` where delay thread could be hanging on upgrading locks with no apparent reason.
- Fixed problem with `myisampack` and `BLOB`.
- Fixed problem when one edited '.MRG' tables by hand. (Patch from Benjamin Pflugmann).
- Enforce that all tables in a `MERGE` table come from the same database.
- Fixed bug with `LOAD DATA INFILE` and transactional tables.
- Fix bug when using `INSERT DELAYED` with wrong column definition.
- Fixed core dump during `REPAIR` of some particularly broken tables.
- Fixed bug in InnoDB and `AUTO_INCREMENT` columns.
- Fixed bug in InnoDB and `RENAME TABLE` columns.
- Fixed critical bug in InnoDB and `BLOB` columns. If you have used `BLOB` columns larger than 8000 bytes in an InnoDB table, it is necessary to dump the table with `mysqldump`, drop it and restore it from the dump.
- Applied large patch for OS/2 from Yuri Dario.
- Fixed problem with InnoDB when one could get the error `Can't execute the given command...` even when no transaction was active.
- Applied some minor fixes that concern Gemini.
- Use real arithmetic operations even in integer context if not all arguments are integers. (Fixes uncommon bug in some integer contexts).

- Don't force everything to lowercase on Windows. (To fix problem with Windows and ALTER TABLE). Now `--lower_case_names` also works on Unix.
- Fixed that automatic rollback is done when thread end doesn't lock other threads.

D.4.17 Changes in release 3.23.41 (11 Aug 2001)

- Added `--sql-mode=option[,option[,option]]` option to `mysqld`. See [line options-snt \[Command-line options\]](#), page [line options-pg](#).
- Fixed possible problem with `shutdown` on Solaris where the `.pid` file wasn't deleted.
- InnoDB now supports < 4 GB rows. The former limit was 8000 bytes.
- The `doublewrite` file flush method is used in InnoDB. It reduces the need for Unix `fsync()` calls to a fraction and improves performance on most Unix flavors.
- You can now use the InnoDB Monitor to print a lot of InnoDB state information, including locks, to the standard output. This is useful in performance tuning.
- Several bugs which could cause hangs in InnoDB have been fixed.
- Split `record_buffer` to `record_buffer` and `record_rnd_buffer`. To make things compatible to previous MySQL versions, if `record_rnd_buffer` is not set, then it takes the value of `record_buffer`.
- Fixed optimising bug in ORDER BY where some ORDER BY parts were wrongly removed.
- Fixed overflow bug with ALTER TABLE and MERGE tables.
- Added prototypes for `my_thread_init()` and `my_thread_end()` to `'mysql_com.h'`
- Added `--safe-user-create` option to `mysqld`.
- Fixed bug in `SELECT DISTINCT ... HAVING` that caused error message `Can't find record in #...`

D.4.18 Changes in release 3.23.40

- Fixed problem with `--low-priority-updates` and INSERT statements.
- Fixed bug in slave thread when under some rare circumstances it could get 22 bytes ahead on the offset in the master.
- Added `slave_net_timeout` for replication.
- Fixed problem with UPDATE and BDB tables.
- Fixed hard bug in BDB tables when using key parts.
- Fixed problem when using `GRANT FILE ON database.* ...`; previously we added the DROP privilege for the database.
- Fixed `DELETE FROM nome_tabela ... LIMIT 0` and `UPDATE FROM nome_tabela ... LIMIT 0`, which acted as though the LIMIT clause was not present (they deleted or updated all selected rows).
- `CHECK TABLE` now checks if an `AUTO_INCREMENT` column contains the value 0.
- Sending a `SIGHUP` to `mysqld` will now only flush the logs, not reset the replication.
- Fixed parser to allow floats of type `1.0e1` (no sign after `e`).
- Option `--force` to `myisamchk` now also updates states.

- Added option `--warnings` to `mysqld`. Now `mysqld` prints the error `Aborted connection` only if this option is used.
- Fixed problem with `SHOW CREATE TABLE` when you didn't have a `PRIMARY KEY`.
- Properly fixed the rename of `innodb_unix_file_flush_method` variable to `innodb_flush_method`.
- Fixed bug when converting `BIGINT UNSIGNED` to `DOUBLE`. This caused a problem when doing comparisons with `BIGINT` values outside of the signed range.
- Fixed bug in BDB tables when querying empty tables.
- Fixed a bug when using `COUNT(DISTINCT)` with `LEFT JOIN` and there weren't any matching rows.
- Removed all documentation referring to the `GEMINI` table type. `GEMINI` is not released under an `Open Source` license.

D.4.19 Changes in release 3.23.39 (12 Jun 2001)

- The `AUTO_INCREMENT` sequence wasn't reset when dropping and adding an `AUTO_INCREMENT` column.
- `CREATE ... SELECT` now creates non-unique indexes delayed.
- Fixed problem where `LOCK TABLES nome_tabela READ` followed by `FLUSH TABLES` put an exclusive lock on the table.
- `REAL @variable` values were represented with only 2 digits when converted to strings.
- Fixed problem that client "hung" when `LOAD TABLE FROM MASTER` failed.
- `myisamchk --fast --force` will no longer repair tables that only had the open count wrong.
- Added functions to handle symbolic links to make life easier in 4.0.
- We are now using the `-lcma` thread library on HP-UX 10.20 so that MySQL will be more stable on HP-UX.
- Fixed problem with `IF()` and number of decimals in the result.
- Fixed date-part extraction functions to work with dates where day and/or month is 0.
- Extended argument length in option files from 256 to 512 chars.
- Fixed problem with shutdown when `INSERT DELAYED` was waiting for a `LOCK TABLE`.
- Fixed core dump bug in InnoDB when tablespace was full.
- Fixed problem with `MERGE` tables and big tables (> 4G) when using `ORDER BY`.

D.4.20 Changes in release 3.23.38 (09 May 2001)

- Fixed a bug when `SELECT` from `MERGE` table sometimes results in incorrectly ordered rows.
- Fixed a bug in `REPLACE()` when using the `ujis` character set.
- Applied Sleepycat BDB patches 3.2.9.1 and 3.2.9.2.
- Added `--skip-stack-trace` option to `mysqld`.
- `CREATE TEMPORARY` now works with InnoDB tables.
- InnoDB now promotes sub keys to whole keys.

- Added option `CONCURRENT` to `LOAD DATA`.
- Better error message when slave `max_allowed_packet` is too low to read a very long log event from the master.
- Fixed bug when too many rows were removed when using `SELECT DISTINCT ... HAVING`.
- `SHOW CREATE TABLE` now returns `TEMPORARY` for temporary tables.
- Added `Rows_examined` to slow query log.
- Fixed problems with function returning empty string when used together with a group function and a `WHERE` that didn't match any rows.
- New program `mysqlcheck`.
- Added database name to output for administrative commands like `CHECK`, `REPAIR`, `OPTIMIZE`.
- Lots of portability fixes for InnoDB.
- Changed optimiser so that queries like `SELECT * FROM nome_tabela,nome_tabela2 ... ORDER BY key_part1 LIMIT #` will use index on `key_part1` instead of `filesort`.
- Fixed bug when doing `LOCK TABLE to_table WRITE,...; INSERT INTO to_table... SELECT ...` when `to_table` was empty.
- Fixed bug with `LOCK TABLE` and BDB tables.

D.4.21 Changes in release 3.23.37 (17 Apr 2001)

- Fixed a bug when using `MATCH()` in `HAVING` clause.
- Fixed a bug when using `HEAP` tables with `LIKE`.
- Added `--mysql-version` option to `safe_mysqld`
- Changed `INNOBASE` to `InnoDB` (because the `INNOBASE` name was already used). All `configure` options and `mysqld` start options now use `innodb` instead of `innobase`. This means that before upgrading to this version, you have to change any configuration files where you have used `innobase` options!
- Fixed bug when using indexes on `CHAR(255) NULL` columns.
- Slave thread will now be started even if `master-host` is not set, as long as `server-id` is set and valid `'master.info'` is present.
- Partial updates (terminated with kill) are now logged with a special error code to the binary log. Slave will refuse to execute them if the error code indicates the update was terminated abnormally, and will have to be recovered with `SET SQL_SLAVE_SKIP_COUNTER=1; SLAVE START` after a manual sanity check/correction of data integrity.
- Fixed bug that erroneously logged a drop of internal temporary table on thread termination to the binary log -- this bug affected replication.
- Fixed a bug in `REGEXP` on 64-bit machines.
- `UPDATE` and `DELETE` with `WHERE unique_key_part IS NULL` didn't update/delete all rows.
- Disabled `INSERT DELAYED` for tables that support transactions.
- Fixed bug when using date functions on `TEXT/BLOB` column with wrong date format.

- UDFs now also work on Windows. (Patch by Ralph Mason.)
- Fixed bug in `ALTER TABLE` and `LOAD DATA INFILE` that disabled key-sorting. These commands should now be faster in most cases.
- Fixed performance bug where reopened tables (tables that had been waiting for `FLUSH` or `REPAIR`) would not use indexes for the next query.
- Fixed problem with `ALTER TABLE` to InnoDB tables on FreeBSD.
- Added `mysqld` variables `myisam_max_sort_file_size` and `myisam_max_extra_sort_file_size`.
- Initialise signals early to avoid problem with signals in InnoDB.
- Applied patch for the `tis620` character set to make comparisons case-independent and to fix a bug in `LIKE` for this character set. **Note:** All tables that uses the `tis620` character set must be fixed with `myisamchk -r` or `REPAIR TABLE` !
- Added `--skip-safemalloc` option to `mysqld`.

D.4.22 Changes in release 3.23.36 (27 Mar 2001)

- Fixed a bug that allowed use of database names containing a `'.` character. This fixes a serious security issue when `mysqld` is run as root.
- Fixed bug when thread creation failed (could happen when doing a **lot** of connections in a short time).
- Fixed some problems with `FLUSH TABLES` and `TEMPORARY` tables. (Problem with freeing the key cache and error `Can't reopen table....`)
- Fixed a problem in InnoDB with other character sets than `latin1` and another problem when using many columns.
- Fixed bug that caused a core dump when using a very complex query involving `DISTINCT` and summary functions.
- Added `SET TRANSACTION ISOLATION LEVEL ...`
- Added `SELECT ... FOR UPDATE`.
- Fixed bug where the number of affected rows was not returned when MySQL was compiled without transaction support.
- Fixed a bug in `UPDATE` where keys weren't always used to find the rows to be updated.
- Fixed a bug in `CONCAT_WS()` where it returned incorrect results.
- Changed `CREATE ... SELECT` and `INSERT ... SELECT` to not allow concurrent inserts as this could make the binary log hard to repeat. (Concurrent inserts are enabled if you are not using the binary or update log.)
- Changed some macros to be able to use fast mutex with `glibc 2.2`.

D.4.23 Changes in release 3.23.35 (15 Mar 2001)

- Fixed newly introduced bug in `ORDER BY`.
- Fixed wrong define `CLIENT_TRANSACTIONS`.
- Fixed bug in `SHOW VARIABLES` when using `INNOBASE` tables.
- Setting and using user variables in `SELECT DISTINCT` didn't work.

- Tuned `SHOW ANALYZE` for small tables.
- Fixed handling of arguments in the benchmark script `run-all-tests`.

D.4.24 Changes in release 3.23.34a

- Added extra files to the distribution to allow `INNOBASE` support to be compiled.

D.4.25 Changes in release 3.23.34 (10 Mar 2001)

- Added the `INNOBASE` storage engine and the `BDB` storage engine to the MySQL source distribution.
- Updated the documentation about `GEMINI` tables.
- Fixed a bug in `INSERT DELAYED` that caused threads to hang when inserting `NULL` into an `AUTO_INCREMENT` column.
- Fixed a bug in `CHECK TABLE / REPAIR TABLE` that could cause a thread to hang.
- `REPLACE` will not replace a row that conflicts with an `AUTO_INCREMENT` generated key.
- `mysqld` now only sets `CLIENT_TRANSACTIONS` in `mysql->server_capabilities` if the server supports a transaction-safe storage engine.
- Fixed `LOAD DATA INFILE` to allow numeric values to be read into `ENUM` and `SET` columns.
- Improved error diagnostic for slave thread exit.
- Fixed bug in `ALTER TABLE ... ORDER BY`.
- Added `max_user_connections` variable to `mysqld`.
- Limit query length for replication by `max_allowed_packet`, not the arbitrary limit of 4 MB.
- Allow space around `=` in argument to `--set-variable`.
- Fixed problem in automatic repair that could leave some threads in state `Waiting for table`.
- `SHOW CREATE TABLE` now displays the `UNION()` for `MERGE` tables.
- `ALTER TABLE` now remembers the old `UNION()` definition.
- Fixed bug when replicating timestamps.
- Fixed bug in bidirectional replication.
- Fixed bug in the `BDB` storage engine that occurred when using an index on multi-part key where a key part may be `NULL`.
- Fixed `MAX()` optimisation on sub-key for `BDB` tables.
- Fixed problem where garbage results were returned when using `BDB` tables and `BLOB` or `TEXT` fields when joining many tables.
- Fixed a problem with `BDB` tables and `TEXT` columns.
- Fixed bug when using a `BLOB` key where a const row wasn't found.
- Fixed that `mysqlbinlog` writes the timestamp value for each query. This ensures that one gets same values for date functions like `NOW()` when using `mysqlbinlog` to pipe the queries to another server.
- Allow `--skip-gemini`, `--skip-bdb`, and `--skip-innodb` options to be specified when invoking `mysqld`, even if these storage engines are not compiled in to `mysqld`.

- One can now do `GROUP BY ... DESC`.
- Fixed a deadlock in the `SET` code, when one ran `SET @foo=bar`, where `bar` is a column reference, an error was not properly generated.

D.4.26 Changes in release 3.23.33 (09 Feb 2001)

- Fixed DNS lookups not to use the same mutex as the hostname cache. This will enable known hosts to be quickly resolved even if a DNS lookup takes a long time.
- Added `--character-sets-dir` option to `myisampack`.
- Removed warnings when running `REPAIR TABLE ... EXTENDED`.
- Fixed a bug that caused a core dump when using `GROUP BY` on an alias, where the alias was the same as an existing column name.
- Added `SEQUENCE()` as an example UDF function.
- Changed `mysql_install_db` to use `BINARY` for `CHAR` columns in the privilege tables.
- Changed `TRUNCATE nome_tabela` to `TRUNCATE TABLE nome_tabela` to use the same syntax as Oracle. Until 4.0 we will also allow `TRUNCATE nome_tabela` to not crash old code.
- Fixed “no found rows” bug in MyISAM tables when a BLOB was first part of a multi-part key.
- Fixed bug where `CASE` didn’t work with `GROUP BY`.
- Added `--sort-recover` option to `myisamchk`.
- `myisamchk -S` and `OPTIMIZE TABLE` now work on Windows.
- Fixed bug when using `DISTINCT` on results from functions that referred to a group function, like:


```
SELECT a, DISTINCT SEC_TO_TIME(SUM(a))
FROM nome_tabela GROUP BY a, b;
```
- Fixed buffer overrun in `libmysqlclient` library. Fixed bug in handling `STOP` event after `ROTATE` event in replication.
- Fixed another buffer overrun in `DROP DATABASE`.
- Added `Table_locks_immediate` and `Table_locks_waited` status variables.
- Fixed bug in replication that broke slave server start with existing `‘master.info’`. This fixes a bug introduced in 3.23.32.
- Added `SET SQL_SLAVE_SKIP_COUNTER=n` command to recover from replication glitches without a full database copy.
- Added `max_binlog_size` variable; the binary log will be rotated automatically when the size crosses the limit.
- Added `Last_error`, `Last_errno`, and `Slave_skip_counter` variables to `SHOW SLAVE STATUS`.
- Fixed bug in `MASTER_POS_WAIT()` function.
- Execute core dump handler on `SIGILL`, and `SIGBUS` in addition to `SIGSEGV`.
- On x86 Linux, print the current query and thread (connection) id, if available, in the core dump handler.

- Fixed several timing bugs in the test suite.
- Extended `mysqltest` to take care of the timing issues in the test suite.
- `ALTER TABLE` can now be used to change the definition for a `MERGE` table.
- Fixed creation of `MERGE` tables on Windows.
- Portability fixes for OpenBSD and OS/2.
- Added `--temp-pool` option to `mysqld`. Using this option will cause most temporary files created to use a small set of names, rather than a unique name for each new file. This is to work around a problem in the Linux kernel dealing with creating a bunch of new files with different names. With the old behaviour, Linux seems to "leak" memory, as it's being allocated to the directory entry cache instead of the disk cache.

D.4.27 Changes in release 3.23.32 (22 Jan 2001: Production)

- Changed code to get around compiler bug in Compaq C++ on OSF/1, that broke `BACKUP`, `RESTORE`, `CHECK`, `REPAIR`, and `ANALYZE TABLE`.
- Added option `FULL` to `SHOW COLUMNS`. Now we show the privilege list for the columns only if this option is given.
- Fixed bug in `SHOW LOGS` when there weren't any BDB logs.
- Fixed a timing problem in replication that could delay sending an update to the client until a new update was done.
- Don't convert field names when using `mysql_list_fields()`. This is to keep this code compatible with `SHOW FIELDS`.
- `MERGE` tables didn't work on Windows.
- Fixed problem with `SET PASSWORD=...` on Windows.
- Added missing `'my_config.h'` to RPM distribution.
- `TRIM("foo" from "foo")` didn't return an empty string.
- Added `--with-version-suffix` option to `configure`.
- Fixed core dump when client aborted connection without `mysql_close()`.
- Fixed a bug in `RESTORE TABLE` when trying to restore from a non-existent directory.
- Fixed a bug which caused a core dump on the slave when replicating `SET PASSWORD`.
- Added `MASTER_POS_WAIT()`.

D.4.28 Changes in release 3.23.31 (17 Jan 2001)

- The test suite now tests all reachable BDB interface code. During testing we found and fixed many errors in the interface code.
- Using `HAVING` on an empty table could produce one result row when it shouldn't.
- Fixed the MySQL RPM so it no longer depends on Perl5.
- Fixed some problems with `HEAP` tables on Windows.
- `SHOW TABLE STATUS` didn't show correct average row length for tables larger than 4G.
- `CHECK TABLE ... EXTENDED` didn't check row links for fixed size tables.
- Added option `MEDIUM` to `CHECK TABLE`.

- Fixed problem when using `DECIMAL()` keys on negative numbers.
- `HOURL()` (and some other `TIME` functions) on a `CHAR` column always returned `NULL`.
- Fixed security bug in something (please upgrade if you are using an earlier MySQL 3.23 version).
- Fixed buffer overflow bug when writing a certain error message.
- Added usage of `setrlimit()` on Linux to get `-O --open-files-limit=#` to work on Linux.
- Added `bdb_version` variable to `mysqld`.
- Fixed bug when using expression of type:


```
SELECT ... FROM t1 LEFT JOIN t2 ON (t1.a=t2.a) WHERE t1.a=t2.a
```

 In this case the test in the `WHERE` clause was wrongly optimised away.
- Fixed bug in `MyISAM` when deleting keys with possible `NULL` values, but the first key-column was not a prefix-compressed text column.
- Fixed `mysql.server` to read the `[mysql.server]` option file group rather than the `[mysql_server]` group.
- Fixed `safe_mysqld` and `mysql.server` to also read the `server` option section.
- Added `Threads_created` status variable to `mysqld`.

D.4.29 Changes in release 3.23.30 (04 Jan 2001)

- Added `SHOW OPEN TABLES` command.
- Fixed that `myisamdump` works against old `mysqld` servers.
- Fixed `myisamchk -k#` so that it works again.
- Fixed a problem with replication when the binary log file went over 2G on 32-bit systems.
- `LOCK TABLES` will now automatically start a new transaction.
- Changed `BDB` tables to not use internal subtransactions and reuse open files to get more speed.
- Added `--mysqld=#` option to `safe_mysqld`.
- Allow hex constants in the `--fields-**-by` and `--lines-terminated-by` options to `mysqldump` and `mysqlimport`. By Paul DuBois.
- Added `--safe-show-database` option to `mysqld`.
- Added `have_bdb`, `have_gemini`, `have_innobase`, `have_raid` and `have_openssl` to `SHOW VARIABLES` to make it easy to test for supported extensions.
- Added `--open-files-limit` option to `mysqld`.
- Changed `--open-files` option to `--open-files-limit` in `safe_mysqld`.
- Fixed a bug where some rows were not found with `HEAP` tables that had many keys.
- Fixed that `--bdb-no-sync` works.
- Changed `--bdb-recover` to `--bdb-no-recover` as `recover` should be on by default.
- Changed the default number of `BDB` locks to 10000.
- Fixed a bug from 3.23.29 when allocating the shared structure needed for `BDB` tables.

- Changed `mysqld_multi.sh` to use configure variables. Patch by Christopher McCrory.
- Added fixing of include files for Solaris 2.8.
- Fixed bug with `--skip-networking` on Debian Linux.
- Fixed problem that some temporary files were reported as having the name `UNOPENED` in error messages.
- Fixed bug when running two simultaneous `SHOW LOGS` queries.

D.4.30 Changes in release 3.23.29 (16 Dec 2000)

- Configure updates for Tru64, large file support, and better TCP wrapper support. By Albert Chin-A-Young.
- Fixed bug in `<=>` operator.
- Fixed bug in `REPLACE` with BDB tables.
- `LPAD()` and `RPAD()` will shorten the result string if it's longer than the length argument.
- Added `SHOW LOGS` command.
- Remove unused BDB logs on shutdown.
- When creating a table, put `PRIMARY` keys first, followed by `UNIQUE` keys.
- Fixed a bug in `UPDATE` involving multi-part keys where one specified all key parts both in the update and the `WHERE` part. In this case MySQL could try to update a record that didn't match the whole `WHERE` part.
- Changed drop table to first drop the tables and then the `.frm` file.
- Fixed a bug in the hostname cache which caused `mysqld` to report the hostname as `' '` in some error messages.
- Fixed a bug with `HEAP` type tables; the variable `max_heap_table_size` wasn't used. Now either `MAX_ROWS` or `max_heap_table_size` can be used to limit the size of a `HEAP` type table.
- Changed the default server-id to 1 for masters and 2 for slaves to make it easier to use the binary log.
- Renamed `bdb_lock_max` variable to `bdb_max_lock`.
- Added support for `AUTO_INCREMENT` on sub-fields for BDB tables.
- Added `ANALYZE` of BDB tables.
- In BDB tables, we now store the number of rows; this helps to optimise queries when we need an approximation of the number of rows.
- If we get an error in a multi-row statement, we now only roll back the last statement, not the entire transaction.
- If you do a `ROLLBACK` when you have updated a non-transactional table you will get an error as a warning.
- Added `--bdb-shared-data` option to `mysqld`.
- Added `Slave_open_temp_tables` status variable to `mysqld`
- Added `binlog_cache_size` and `max_binlog_cache_size` variables to `mysqld`.
- `DROP TABLE`, `RENAME TABLE`, `CREATE INDEX` and `DROP INDEX` are now transaction end-points.

- If you do a `DROP DATABASE` on a symbolically linked database, both the link and the original database is deleted.
- Fixed `DROP DATABASE` to work on OS/2.
- Fixed bug when doing a `SELECT DISTINCT ... table1 LEFT JOIN table2 ...` when `table2` was empty.
- Added `--abort-slave-event-count` and `--disconnect-slave-event-count` options to `mysqld` for debugging and testing of replication.
- Fixed replication of temporary tables. Handles everything except slave server restart.
- `SHOW KEYS` now shows whether key is `FULLTEXT`.
- New script `mysqld_multi`. See [\(undefined\) \[mysqld_multi\], page \(undefined\)](#).
- Added new script, `mysql-multi.server.sh`. Thanks to Tim Bunce Tim.Bunce@ig.co.uk for modifying `mysql.server` to easily handle hosts running many `mysqld` processes.
- `safe_mysqld`, `mysql.server`, and `mysql_install_db` have been modified to use `mysql_print_defaults` instead of various hacks to read the 'my.cnf' files. In addition, the handling of various paths has been made more consistent with how `mysqld` handles them by default.
- Automatically remove Berkeley DB transaction logs that no longer are in use.
- Fixed bug with several `FULLTEXT` indexes in one table.
- Added a warning if number of rows changes on `REPAIR/OPTIMIZE`.
- Applied patches for OS/2 by Yuri Dario.
- `FLUSH TABLES nome_tabela` didn't always flush the index tree to disk properly.
- `--bootstrap` is now run in a separate thread. This fixes a problem that caused `mysql_install_db` to core dump on some Linux machines.
- Changed `mi_create()` to use less stack space.
- Fixed bug with optimiser trying to over-optimize `MATCH()` when used with `UNIQUE` key.
- Changed `crash-me` and the MySQL benchmarks to also work with FrontBase.
- Allow `RESTRICT` and `CASCADE` after `DROP TABLE` to make porting easier.
- Reset status variable which could cause problem if one used `--slow-log`.
- Added `connect_timeout` variable to `mysql` and `mysqladmin`.
- Added `connect-timeout` as an alias for `timeout` for option files read by `mysql_options()`.

D.4.31 Changes in release 3.23.28 (22 Nov 2000: Gamma)

- Added new options `--pager[=...]`, `--no-pager`, `--tee=...` and `--no-tee` to the `mysql` client. The new corresponding interactive commands are `pager`, `nopager`, `tee` and `notee`. See [\(undefined\) \[mysql\], page \(undefined\)](#), `mysql --help` and the interactive help for more information.
- Fixed crash when automatic repair of MyISAM table failed.
- Fixed a major performance bug in the table locking code when one constantly had a lot of `SELECT`, `UPDATE` and `INSERT` statements running. The symptom was that the

UPDATE and INSERT queries were locked for a long time while new SELECT statements were executed before the updates.

- When reading `options_files` with `mysql_options()` the `return-found-rows` option was ignored.
- One can now specify `interactive-timeout` in the option file that is read by `mysql_options()`. This makes it possible to force programs that run for a long time (like `mysqlhotcopy`) to use the `interactive_timeout` time instead of the `wait_timeout` time.
- Added to the slow query log the time and the user name for each logged query. If you are using `--log-long-format` then also queries that do not use an index are logged, even if the query takes less than `long_query_time` seconds.
- Fixed a problem in LEFT JOIN which caused all columns in a reference table to be NULL.
- Fixed a problem when using NATURAL JOIN without keys.
- Fixed a bug when using a multi-part keys where the first part was of type TEXT or BLOB.
- DROP of temporary tables wasn't stored in the update/binary log.
- Fixed a bug where `SELECT DISTINCT * ... LIMIT #` only returned one row.
- Fixed a bug in the assembler code in `strstr()` for SPARC and cleaned up the 'global.h' header file to avoid a problem with bad aliasing with the compiler submitted with Red Hat 7.0. (Reported by Trond Eivind Glomsrd)
- The `--skip-networking` option now works properly on NT.
- Fixed a long outstanding bug in the ISAM tables when a row with a length of more than 65K was shortened by a single byte.
- Fixed a bug in MyISAM when running multiple updating processes on the same table.
- Allow one to use `FLUSH TABLE nome_tabela`.
- Added `--replicate-ignore-table`, `--replicate-do-table`, `--replicate-wild-ignore-table`, and `--replicate-wild-do-table` options to `mysqld`.
- Changed all log files to use our own IO_CACHE mechanism instead of FILE to avoid OS problems when there are many files open.
- Added `--open-files` and `--timezone` options to `safe_mysqld`.
- Fixed a fatal bug in `CREATE TEMPORARY TABLE ... SELECT`
- Fixed a problem with `CREATE TABLE ... SELECT NULL`.
- Added variables `large_file_support`, `net_read_timeout`, `net_write_timeout` and `query_buffer_size` to `SHOW VARIABLES`.
- Added status variables `created_tmp_files` and `sort_merge_passes` to `SHOW STATUS`.
- Fixed a bug where we didn't allow an index name after the FOREIGN KEY definition.
- Added `TRUNCATE table_name` as a synonym for `DELETE FROM table_name`.
- Fixed a bug in a BDB key compare function when comparing part keys.
- Added `bdb_lock_max` variable to `mysqld`.
- Added more tests to the benchmark suite.
- Fixed an overflow bug in the client code when using overly long database names.

- `mysql_connect()` now aborts on Linux if the server doesn't answer in `timeout` seconds.
- `SLAVE START` did not work if you started with `--skip-slave-start` and had not explicitly run `CHANGE MASTER TO`.
- Fixed the output of `SHOW MASTER STATUS` to be consistent with `SHOW SLAVE STATUS`. (It now has no directory in the log name.)
- Added `PURGE MASTER LOGS TO`.
- Added `SHOW MASTER LOGS`.
- Added `--safemalloc-mem-limit` option to `mysqld` to simulate memory shortage when compiled with the `--with-debug=full` option.
- Fixed several core dumps in out-of-memory conditions.
- `SHOW SLAVE STATUS` was using an uninitialised mutex if the slave had not been started yet.
- Fixed bug in `ELT()` and `MAKE_SET()` when the query used a temporary table.
- `CHANGE MASTER TO` without specifying `MASTER_LOG_POS` would set it to 0 instead of 4 and hit the magic number in the master binlog.
- `ALTER TABLE ... ORDER BY ...` syntax added. This will create the new table with the rows in a specific order.

D.4.32 Changes in release 3.23.27 (24 Oct 2000)

- Fixed a bug where the automatic repair of MyISAM tables sometimes failed when the datafile was corrupt.
- Fixed a bug in `SHOW CREATE` when using `AUTO_INCREMENT` columns.
- Changed BDB tables to use new compare function in Berkeley DB 3.2.3.
- You can now use Unix sockets with MIT-pthreads.
- Added the `latin5` (turkish) character set.
- Small portability fixes.

D.4.33 Changes in release 3.23.26 (18 Oct 2000)

- Renamed `FLUSH MASTER` and `FLUSH SLAVE` to `RESET MASTER` and `RESET SLAVE`.
- Fixed `<>` to work properly with `NULL`.
- Fixed a problem with `SUBSTRING_INDEX()` and `REPLACE()`. (Patch by Alexander Igonichev)
- Fix `CREATE TEMPORARY TABLE IF NOT EXISTS` not to produce an error if the table exists.
- If you don't create a `PRIMARY KEY` in a BDB table, a hidden `PRIMARY KEY` will be created.
- Added read-only-key optimisation to BDB tables.
- `LEFT JOIN` in some cases preferred a full table scan when there was no `WHERE` clause.
- When using `--log-slow-queries`, don't count the time waiting for a lock.
- Fixed bug in lock code on Windows which could cause the key cache to report that the key file was crashed even if it was okay.
- Automatic repair of MyISAM tables if you start `mysqld` with `--myisam-recover`.

- Removed the `TYPE=` keyword from `CHECK` and `REPAIR`. Allow `CHECK` options to be combined. (You can still use `TYPE=`, but this usage is deprecated.)
- Fixed mutex bug in the binary replication log -- long update queries could be read only in part by the slave if it did it at the wrong time, which was not fatal, but resulted in a performance-degrading reconnect and a scary message in the error log.
- Changed the format of the binary log -- added magic number, server version, binlog version. Added server id and query error code for each query event.
- Replication thread from the slave now will kill all the stale threads from the same server.
- Long replication user names were not being handled properly.
- Added `--replicate-rewrite-db` option to `mysqld`.
- Added `--skip-slave-start` option to `mysqld`.
- Updates that generated an error code (such as `INSERT INTO foo(some_key) values (1), (1)`) erroneously terminated the slave thread.
- Added optimisation of queries where `DISTINCT` is only used on columns from some of the tables.
- Allow floating-point numbers where there is no sign after the exponent (like `1e1`).
- `SHOW GRANTS` didn't always show all column grants.
- Added `--default-extra-file=#` option to all MySQL clients.
- Columns referenced in `INSERT` statements now are initialised properly.
- `UPDATE` didn't always work when used with a range on a timestamp that was part of the key that was used to find rows.
- Fixed a bug in `FULLTEXT` index when inserting a `NULL` column.
- Changed to use `mkstemp()` instead of `tempnam()`. Based on a patch from John Jones.

D.4.34 Changes in release 3.23.25 (29 Sep 2000)

- Fixed that `databasename` works as second argument to `mysqlhotcopy`.
- The values for the `UMASK` and `UMASK_DIR` environment variables now can be specified in octal by beginning the value with a zero.
- Added `RIGHT JOIN`. This makes `RIGHT` a reserved word.
- Added `@@IDENTITY` as a synonym for `LAST_INSERT_ID()`. (This is for MSSQL compatibility.)
- Fixed a bug in `myisamchk` and `REPAIR` when using `FULLTEXT` index.
- `LOAD DATA INFILE` now works with FIFOs. (Patch by Toni L. Harbaugh-Blackford.)
- `FLUSH LOGS` broke replication if you specified a log name with an explicit extension as the value of the `log-bin` option.
- Fixed a bug in `MyISAM` with packed multi-part keys.
- Fixed crash when using `CHECK TABLE` on Windows.
- Fixed a bug where `FULLTEXT` index always used the `koi8_ukr` character set.
- Fixed privilege checking for `CHECK TABLE`.
- The `MyISAM` repair/reindex code didn't use the `--tmpdir` option for its temporary files.

- Added `BACKUP TABLE` and `RESTORE TABLE`.
- Fixed core dump on `CHANGE MASTER TO` when the slave did not have the master to start with.
- Fixed incorrect `Time` in the processlist for `Connect` of the slave thread.
- The slave now logs when it connects to the master.
- Fixed a core dump bug when doing `FLUSH MASTER` if you didn't specify a filename argument to `--log-bin`.
- Added missing `'ha_berkeley.x'` files to the MySQL Windows distribution.
- Fixed some mutex bugs in the log code that could cause thread blocks if new log files couldn't be created.
- Added lock time and number of selected processed rows to slow query log.
- Added `--memlock` option to `mysqld` to lock `mysqld` in memory on systems with the `mlockall()` call (like in Solaris).
- `HEAP` tables didn't use keys properly. (Bug from 3.23.23.)
- Added better support for `MERGE` tables (keys, mapping, creation, documentation...). See [\[MERGE\]](#), page [\[MERGE\]](#).
- Fixed bug in `mysqldump` from 3.23 which caused some `CHAR` columns not to be quoted.
- Merged `analyze`, `check`, `optimize` and `repair` code.
- `OPTIMIZE TABLE` is now mapped to `REPAIR` with statistics and sorting of the index tree. This means that for the moment it only works on `MyISAM` tables.
- Added a pre-allocated block to `root_malloc` to get fewer mallocs.
- Added a lot of new statistics variables.
- Fixed `ORDER BY` bug with `BDB` tables.
- Removed warning that `mysqld` couldn't remove the `'pid'` file under Windows.
- Changed `--log-isam` to log `MyISAM` tables instead of `isam` tables.
- Fixed `CHECK TABLE` to work on Windows.
- Added file mutexes to make `pwrite()` safe on Windows.

D.4.35 Changes in release 3.23.24 (08 Sep 2000)

- Added `created_tmp_disk_tables` variable to `mysqld`.
- To make it possible to reliably dump and restore tables with `TIMESTAMP(X)` columns, MySQL now reports columns with `X` other than 14 or 8 to be strings.
- Changed sort order for `latin1` as it was before MySQL Version 3.23.23. Any table that was created or modified with 3.23.22 must be repaired if it has `CHAR` columns that may contain characters with ASCII values greater than 128!
- Fixed small memory leak introduced from 3.23.22 when creating a temporary table.
- Fixed problem with `BDB` tables and reading on a unique (not primary) key.
- Restored the `win1251` character set (it's now only marked deprecated).

D.4.36 Changes in release 3.23.23 (01 Sep 2000)

- Changed sort order for 'German'; all tables created with 'German' sortorder must be repaired with `REPAIR TABLE` or `myisamchk` before use!
- Added `--core-file` option to `mysqld` to get a core file on Linux if `mysqld` dies on the `SIGSEGV` signal.
- MySQL client `mysql` now starts with option `--no-named-commands (-g)` by default. This option can be disabled with `--enable-named-commands (-G)`. This may cause incompatibility problems in some cases, for example, in SQL scripts that use named commands without a semicolon, etc. ! Long format commands still work from the first line.
- Fixed a problem when using many pending `DROP TABLE` statements at the same time.
- Optimiser didn't use keys properly when using `LEFT JOIN` on an empty table.
- Added shorter help text when invoking `mysqld` with incorrect options.
- Fixed non-fatal `free()` bug in `mysqlimport`.
- Fixed bug in MyISAM index handling of `DECIMAL/NUMERIC` keys.
- Fixed a bug in concurrent insert in MyISAM tables. In some contexts, usage of `MIN(key_part)` or `MAX(key_part)` returned an empty set.
- Updated `mysqlhotcopy` to use the new `FLUSH TABLES table_list` syntax. Only tables which are being backed up are flushed now.
- Changed behaviour of `--enable-thread-safe-client` so that both non-threaded (`-lmysqlclient`) and threaded (`-lmysqlclient_r`) libraries are built. Users who linked against a threaded `-lmysqlclient` will need to link against `-lmysqlclient_r` now.
- Added atomic `RENAME TABLE` command.
- Don't count `NULL` values in `COUNT(DISTINCT ...)`.
- Changed `ALTER TABLE`, `LOAD DATA INFILE` on empty tables and `INSERT ... SELECT ...` on empty tables to create non-unique indexes in a separate batch with sorting. This will make the above calls much faster when you have many indexes.
- `ALTER TABLE` now logs the first used `insert_id` correctly.
- Fixed crash when adding a default value to a `BLOB` column.
- Fixed a bug with `DATE_ADD/DATE_SUB` where it returned a datetime instead of a date.
- Fixed a problem with the thread cache which made some threads show up as `***DEAD***` in `SHOW PROCESSLIST`.
- Fixed a lock in our `thr_rwlock` code, which could make selects that run at the same time as concurrent inserts crash. This only affects systems that don't have the `pthread_rwlock_rdlock` code.
- When deleting rows with a non-unique key in a `HEAP` table, all rows weren't always deleted.
- Fixed bug in range optimiser for `HEAP` tables for searches on a part index.
- Fixed `SELECT` on part keys to work with `BDB` tables.
- Fixed `INSERT INTO bdb_table ... SELECT` to work with `BDB` tables.
- `CHECK TABLE` now updates key statistics for the table.

- **ANALYZE TABLE** will now only update tables that have been changed since the last **ANALYZE**. Note that this is a new feature and tables will not be marked to be analysed until they are updated in any way with 3.23.23 or newer. For older tables, you have to do **CHECK TABLE** to update the key distribution.
- Fixed some minor privilege problems with **CHECK**, **ANALYZE**, **REPAIR** and **SHOW CREATE** commands.
- Added **CHANGE MASTER TO** statement.
- Added **FAST**, **QUICK EXTENDED** check types to **CHECK TABLES**.
- Changed **myisamchk** so that **--fast** and **--check-only-changed** are also honored with **--sort-index** and **--analyze**.
- Fixed fatal bug in **LOAD TABLE FROM MASTER** that did not lock the table during index re-build.
- **LOAD DATA INFILE** broke replication if the database was excluded from replication.
- More variables in **SHOW SLAVE STATUS** and **SHOW MASTER STATUS**.
- **SLAVE STOP** now will not return until the slave thread actually exits.
- Full-text search via the **MATCH()** function and **FULLTEXT** index type (for MyISAM files). This makes **FULLTEXT** a reserved word.

D.4.37 Changes in release 3.23.22 (31 Jul 2000)

- Fixed that **lex_hash.h** is created properly for each MySQL distribution.
- Fixed that **MASTER** and **COLLECTION** are not reserved words.
- The log generated by **--slow-query-log** didn't contain the whole queries.
- Fixed that open transactions in BDB tables are rolled back if the connection is closed unexpectedly.
- Added workaround for a bug in **gcc 2.96 (intel)** and **gcc 2.9 (IA64)** in **gen_lex_hash.c**.
- Fixed memory leak in the client library when using **host=** in the 'my.cnf' file.
- Optimised functions that manipulate the hours/minutes/seconds.
- Fixed bug when comparing the result of **DATE_ADD()/DATE_SUB()** against a number.
- Changed the meaning of **-F**, **--fast** for **myisamchk**. Added **-C**, **--check-only-changed** option to **myisamchk**.
- Added **ANALYZE nome_tabela** to update key statistics for tables.
- Changed binary items **0x...** to be regarded as integers by default.
- Fix for **SCO** and **SHOW PROCESSLIST**.
- Added **auto-rehash** on reconnect for the **mysql** client.
- Fixed a newly introduced bug in MyISAM, where the index file couldn't get bigger than 64M.
- Added **SHOW MASTER STATUS** and **SHOW SLAVE STATUS**.

D.4.38 Changes in release 3.23.21

- Added **mysql_character_set_name()** function to the MySQL C API.
- Made the update log ASCII 0 safe.

- Added the `mysql_config` script.
- Fixed problem when using `<` or `>` with a char column that was only partly indexed.
- One would get a core dump if the log file was not readable by the MySQL user.
- Changed `mysqladmin` to use `CREATE DATABASE` and `DROP DATABASE` statements instead of the old deprecated API calls.
- Fixed `chown` warning in `safe_mysqld`.
- Fixed a bug in `ORDER BY` that was introduced in 3.23.19.
- Only optimise the `DELETE FROM nome_tabela` to do a drop+create of the table if we are in `AUTOCOMMIT` mode (needed for BDB tables).
- Added extra checks to avoid index corruption when the ISAM/MyISAM index files get full during an `INSERT/UPDATE`.
- `myisamchk` didn't correctly update row checksum when used with `-ro` (this only gave a warning in subsequent runs).
- Fixed bug in `REPAIR TABLE` so that it works with tables without indexes.
- Fixed buffer overrun in `DROP DATABASE`.
- `LOAD TABLE FROM MASTER` is sufficiently bug-free to announce it as a feature.
- `MATCH` and `AGAINST` are now reserved words.

D.4.39 Changes in release 3.23.20

- Fixed bug in 3.23.19; `DELETE FROM nome_tabela` removed the `.frm` file.
- Added `SHOW CREATE TABLE`.

D.4.40 Changes in release 3.23.19

- Changed copyright for all files to GPL for the server code and utilities and to LGPL for the client libraries.
- Fixed bug where all rows matching weren't updated on a MyISAM table when doing update based on key on a table with many keys and some key changed values.
- The Linux MySQL RPMs and binaries are now statically linked with a linuxthread version that has faster mutex handling when used with MySQL.
- `ORDER BY` can now use `REF` keys to find subsets of the rows that need to be sorted.
- Changed name of `print_defaults` program to `my_print_defaults` to avoid name confusion.
- Fixed `NULLIF()` to work as required by SQL-99.
- Added `net_read_timeout` and `net_write_timeout` as startup parameters to `mysqld`.
- Fixed bug that destroyed index when doing `myisamchk --sort-records` on a table with prefix compressed index.
- Added `pack_isam` and `myisampack` to the standard MySQL distribution.
- Added the syntax `BEGIN WORK` (the same as `BEGIN`).
- Fixed core dump bug when using `ORDER BY` on a `CONV()` expression.
- Added `LOAD TABLE FROM MASTER`.
- Added `FLUSH MASTER` and `FLUSH SLAVE`.
- Fixed big/little endian problem in the replication.

D.4.41 Changes in release 3.23.18

- Fixed a problem from 3.23.17 when choosing character set on the client side.
- Added `FLUSH TABLES WITH READ LOCK` to make a global lock suitable for making a copy of MySQL datafiles.
- `CREATE TABLE ... SELECT ... PROCEDURE` now works.
- Internal temporary tables will now use compressed index when using `GROUP BY` on `VARCHAR/CHAR` columns.
- Fixed a problem when locking the same table with both a `READ` and a `WRITE` lock.
- Fixed problem with `myisamchk` and `RAID` tables.

D.4.42 Changes in release 3.23.17

- Fixed a bug in `FIND_IN_SET()` when the first argument was `NULL`.
- Added table locks to Berkeley DB.
- Fixed a bug with `LEFT JOIN` and `ORDER BY` where the first table had only one matching row.
- Added 4 sample `'my.cnf'` example files in the `'support-files'` directory.
- Fixed `duplicate key` problem when doing big `GROUP BY` operations. (This bug was probably introduced in 3.23.15.)
- Changed syntax for `INNER JOIN` to match SQL-99.
- Added `NATURAL JOIN` syntax.
- A lot of fixes in the BDB interface.
- Added handling of `--no-defaults` and `--defaults-file` to `safe_mysqld.sh` and `mysql_install_db.sh`.
- Fixed bug in reading compressed tables with many threads.
- Fixed that `USE INDEX` works with `PRIMARY` keys.
- Added `BEGIN` statement to start a transaction in `AUTOCOMMIT` mode.
- Added support for symbolic links for Windows.
- Changed protocol to let client know if the server is in `AUTOCOMMIT` mode and if there is a pending transaction. If there is a pending transaction, the client library will give an error before reconnecting to the server to let the client know that the server did a rollback. The protocol is still backward-compatible with old clients.
- `KILL` now works on a thread that is locked on a 'write' to a dead client.
- Fixed memory leak in the replication slave thread.
- Added new `log-slave-updates` option to `mysqld`, to allow daisy-chaining the slaves.
- Fixed compile error on FreeBSD and other systems where `pthread_t` is not the same as `int`.
- Fixed master shutdown aborting the slave thread.
- Fixed a race condition in `INSERT DELAYED` code when doing `ALTER TABLE`.
- Added deadlock detection sanity checks to `INSERT DELAYED`.

D.4.43 Changes in release 3.23.16

- Added `SLAVE START` and `SLAVE STOP` statements.
- Added `TYPE=QUICK` option to `CHECK` and to `REPAIR`.
- Fixed bug in `REPAIR TABLE` when the table was in use by other threads.
- Added a thread cache to make it possible to debug MySQL with `gdb` when one does a lot of reconnects. This will also improve systems where you can't use persistent connections.
- Lots of fixes in the Berkeley DB interface.
- `UPDATE IGNORE` will not abort if an update results in a `DUPLICATE_KEY` error.
- Put `CREATE TEMPORARY TABLE` commands in the update log.
- Fixed bug in handling of masked IP numbers in the privilege tables.
- Fixed bug with `delay_key_write` tables and `CHECK TABLE`.
- Added `replicate-do-db` and `replicate-ignore-db` options to `mysqld`, to restrict which databases get replicated.
- Added `SQL_LOG_BIN` option.

D.4.44 Changes in release 3.23.15 (May 2000: Beta)

- To start `mysqld` as `root`, you must now use the `--user=root` option.
- Added interface to Berkeley DB. (This is not yet functional; play with it at your own risk!)
- Replication between master and slaves.
- Fixed bug that other threads could steal a lock when a thread had a lock on a table and did a `FLUSH TABLES` command.
- Added the `slow_launch_time` variable and the `Slow_launch_threads` status variable to `mysqld`. These can be examined with `mysqladmin variables` and `mysqladmin extended-status`.
- Added functions `INET_NTOA()` and `INET_ATON()`.
- The default type of `IF()` now depends on the second and third arguments and not only on the second argument.
- Fixed case when `myisamchk` could go into a loop when trying to repair a crashed table.
- Don't write `INSERT DELAYED` to update log if `SQL_LOG_UPDATE=0`.
- Fixed problem with `REPLACE` on `HEAP` tables.
- Added possible character sets and time zone to `SHOW VARIABLES` output.
- Fixed bug in locking code that could result in locking problems with concurrent inserts under high load.
- Fixed a problem with `DELETE` of many rows on a table with compressed keys where MySQL scanned the index to find the rows.
- Fixed problem with `CHECK` on table with deleted keyblocks.
- Fixed a bug in reconnect (at the client side) where it didn't free memory properly in some contexts.

- Fixed problems in update log when using `LAST_INSERT_ID()` to update a table with an `AUTO_INCREMENT` key.
- Added `NULLIF()` function.
- Fixed bug when using `LOAD DATA INFILE` on a table with `BLOB/TEXT` columns.
- Optimised MyISAM to be faster when inserting keys in sorted order.
- `EXPLAIN SELECT . . .` now also prints out whether MySQL needs to create a temporary table or use file sorting when resolving the `SELECT`.
- Added optimisation to skip `ORDER BY` parts where the part is a constant expression in the `WHERE` part. Indexes can now be used even if the `ORDER BY` doesn't match the index exactly, as long as all the unused index parts and all the extra `ORDER BY` columns are constants in the `WHERE` clause. See [\[MySQL indexes\]](#), page [\(undefined\)](#).
- `UPDATE` and `DELETE` on a whole unique key in the `WHERE` part are now faster than before.
- Changed `RAID_CHUNKSIZE` to be in 1024-byte increments.
- Fixed core dump in `LOAD_FILE(NULL)`.

D.4.45 Changes in release 3.23.14

- Added `mysql_real_escape_string()` function to the MySQL C API.
- Fixed a bug in `CONCAT()` where one of the arguments was a function that returned a modified argument.
- Fixed a critical bug in `myisamchk`, where it updated the header in the index file when one only checked the table. This confused the `mysqld` daemon if it updated the same table at the same time. Now the status in the index file is only updated if one uses `--update-state`. With older `myisamchk` versions you should use `--read-only` when only checking tables, if there is the slightest chance that the `mysqld` server is working on the table at the same time!
- Fixed that `DROP TABLE` is logged in the update log.
- Fixed problem when searching on `DECIMAL()` key field where the column data contained leading zeros.
- Fix bug in `myisamchk` when the `AUTO_INCREMENT` column isn't the first key.
- Allow `DATETIME` in ISO8601 format: 2000-03-12T12:00:00
- Dynamic character sets. A `mysqld` binary can now handle many different character sets (you can choose which when starting `mysqld`).
- Added command `REPAIR TABLE`.
- Added `mysql_thread_safe()` function to the MySQL C API.
- Added the `UMASK_DIR` environment variable.
- Added `CONNECTION_ID()` function to return the client connection thread ID.
- When using `=` on `BLOB` or `VARCHAR BINARY` keys, where only a part of the column was indexed, the whole column of the result row wasn't compared.
- Fix for `sjis` character set and `ORDER BY`.
- When running in ANSI mode, don't allow columns to be used that aren't in the `GROUP BY` part.

D.4.46 Changes in release 3.23.13

- Fixed problem when doing locks on the same table more than 2 times in the same `LOCK TABLE` command; this fixed the problem one got when running the test-ATIS test with `--fast` or `--check-only-changed`.
- Added `SQL_BUFFER_RESULT` option to `SELECT`.
- Removed end space from double/float numbers in results from temporary tables.
- Added `CHECK TABLE` command.
- Added changes for MyISAM in 3.23.12 that didn't get into the source distribution because of CVS problems.
- Fixed bug so that `mysqladmin shutdown` will wait for the local server to close down.
- Fixed a possible endless loop when calculating timestamp.
- Added `print_defaults` program to the '.rpm' files. Removed `mysqlbug` from the client '.rpm' file.

D.4.47 Changes in release 3.23.12 (07 Mar 2000)

- Fixed bug in MyISAM involving `REPLACE ... SELECT ...` which could give a corrupted table.
- Fixed bug in `myisamchk` where it incorrectly reset the `AUTO_INCREMENT` value.
- LOTS of patches for Linux Alpha. MySQL now appears to be relatively stable on Alpha.
- Changed `DISTINCT` on `HEAP` temporary tables to use hashed keys to quickly find duplicated rows. This mostly concerns queries of type `SELECT DISTINCT ... GROUP BY ...`. This fixes a problem where not all duplicates were removed in queries of the above type. In addition, the new code is MUCH faster.
- Added patches to make MySQL compile on Mac OS X.
- Added `IF NOT EXISTS` clause to `CREATE DATABASE`.
- Added `--all-databases` and `--databases` options to `mysqldump` to allow dumping of many databases at the same time.
- Fixed bug in compressed `DECIMAL()` index in MyISAM tables.
- Fixed bug when storing 0 into a timestamp.
- When doing `mysqladmin shutdown` on a local connection, `mysqladmin` now waits until the PID file is gone before terminating.
- Fixed core dump with some `COUNT(DISTINCT ...)` queries.
- Fixed that `myisamchk` works properly with RAID tables.
- Fixed problem with `LEFT JOIN` and `key_field IS NULL`.
- Fixed bug in `net_clear()` which could give the error `Aborted connection` in the MySQL clients.
- Added options `USE INDEX (key_list)` and `IGNORE INDEX (key_list)` as parameters in `SELECT`.
- `DELETE` and `RENAME` should now work on RAID tables.

D.4.48 Changes in release 3.23.11

- Allow the ALTER TABLE nome_tabela ADD (field_list) syntax.
- Fixed problem with optimiser that could sometimes use incorrect keys.
- Fixed that GRANT/REVOKE ALL PRIVILEGES doesn't affect GRANT OPTION.
- Removed extra ')' from the output of SHOW GRANTS.
- Fixed problem when storing numbers in timestamps.
- Fix problem with timezones that have half hour offsets.
- Allow the syntax UNIQUE INDEX in CREATE statements.
- `mysqlhotcopy` - fast online hot-backup utility for local MySQL databases. By Tim Bunce.
- New more secure `mysqlaccess`. Thanks to Steve Harvey for this.
- Added `--i-am-a-dummy` and `--safe-updates` options to `mysql`.
- Added `select_limit` and `max_join_size` variables to `mysql`.
- Added `SQL_MAX_JOIN_SIZE` and `SQL_SAFE_UPDATES` options.
- Added READ LOCAL lock that doesn't lock the table for concurrent inserts. (This is used by `mysqldump`.)
- Changed that LOCK TABLES ... READ doesn't anymore allow concurrent inserts.
- Added `--skip-delay-key-write` option to `mysqld`.
- Fixed security problem in the protocol regarding password checking.
- `_rowid` can now be used as an alias for an integer type unique indexed column.
- Added back blocking of SIGPIPE when compiling with `--thread-safe-clients` to make things safe for old clients.

D.4.49 Changes in release 3.23.10

- Fixed bug in 3.23.9 where memory wasn't properly freed when using LOCK TABLES.

D.4.50 Changes in release 3.23.9

- Fixed problem that affected queries that did arithmetic on group functions.
- Fixed problem with timestamps and INSERT DELAYED.
- Fixed that `date_col BETWEEN const_date AND const_date` works.
- Fixed problem when only changing a 0 to NULL in a table with BLOB/TEXT columns.
- Fixed bug in range optimiser when using many key parts and or on the middle key parts: WHERE K1=1 and K3=2 and (K2=2 and K4=4 or K2=3 and K4=5)
- Added `source` command to `mysql` to allow reading of batch files inside the `mysql` client. Original patch by Matthew Vanecek.
- Fixed critical problem with the WITH GRANT OPTION option.
- Don't give an unnecessary GRANT error when using tables from many databases in the same query.
- Added VIO wrapper (needed for SSL support; by Andrei Errapart and Tõnu Samuel).

- Fixed optimiser problem on **SELECT** when using many overlapping indexes. MySQL should now be able to choose keys even better when there are many keys to choose from.
- Changed optimiser to prefer a range key instead of a ref key when the range key can use more columns than the ref key (which only can use columns with =). For example, the following type of queries should now be faster: **SELECT * from key_part_1=const and key_part_2 > const2**
- Fixed bug that a change of all **VARCHAR** columns to **CHAR** columns didn't change row type from dynamic to fixed.
- Disabled floating-point exceptions for FreeBSD to fix core dump when doing **SELECT FLOOR(POW(2,63))**.
- Renamed **mysqld** startup option from **--delay-key-write** to **--delay-key-write-for-all-tables**.
- Added **read-next-on-key** to **HEAP** tables. This should fix all problems with **HEAP** tables when using non-**UNIQUE** keys.
- Added option to print default arguments to all clients.
- Added **--log-slow-queries** option to **mysqld** to log all queries that take a long time to a separate log file with a time indicating how long the query took.
- Fixed core dump when doing **WHERE key_col=RAND(...)**.
- Fixed optimisation bug in **SELECT ... LEFT JOIN ... key_col IS NULL**, when **key_col** could contain **NULL** values.
- Fixed problem with 8-bit characters as separators in **LOAD DATA INFILE**.

D.4.51 Changes in release 3.23.8 (02 Jan 2000)

- Fixed problem when handling indexfiles larger than 8G.
- Added latest patches to MIT-pthreads for NetBSD.
- Fixed problem with timezones that are < GMT - 11.
- Fixed a bug when deleting packed keys in **NISAM**.
- Fixed problem with **ISAM** when doing some **ORDER BY ... DESC** queries.
- Fixed bug when doing a join on a text key which didn't cover the whole key.
- Option **--delay-key-write** didn't enable delayed key writing.
- Fixed update of **TEXT** column which involved only case changes.
- Fixed that **INSERT DELAYED** doesn't update timestamps that are given.
- Added function **YEARWEEK()** and options **x**, **X**, **v** and **V** to **DATE_FORMAT()**.
- Fixed problem with **MAX(indexed_column)** and **HEAP** tables.
- Fixed problem with **BLOB NULL** keys and **LIKE "prefix%"**.
- Fixed problem with **MyISAM** and fixed-length rows < 5 bytes.
- Fixed problem that could cause MySQL to touch freed memory when doing very complicated **GROUP BY** queries.
- Fixed core dump if you got a crashed table where an **ENUM** field value was too big.

D.4.52 Changes in release 3.23.7 (10 Dec 1999)

- Fixed workaround under Linux to avoid problems with `pthread_mutex_timedwait`, which is used with `INSERT DELAYED`. See [\[Linux\]](#), page [\[Linux\]](#).
- Fixed that one will get a 'disk full' error message if one gets disk full when doing sorting (instead of waiting until we got more disk space).
- Fixed a bug in MyISAM with keys > 250 characters.
- In MyISAM one can now do an `INSERT` at the same time as other threads are reading from the table.
- Added `max_write_lock_count` variable to `mysqld` to force a `READ` lock after a certain number of `WRITE` locks.
- Inverted flag `delay_key_write` on `show variables`.
- Renamed concurrency variable to `thread_concurrency`.
- The following functions are now multi-byte-safe: `LOCATE(substr,str)`, `POSITION(substr IN str)`, `LOCATE(substr,str,pos)`, `INSTR(str,substr)`, `LEFT(str,len)`, `RIGHT(str,len)`, `SUBSTRING(str,pos,len)`, `SUBSTRING(str FROM pos FOR len)`, `MID(str,pos,len)`, `SUBSTRING(str,pos)`, `SUBSTRING(str FROM pos)`, `SUBSTRING_INDEX(str,delim,count)`, `RTRIM(str)`, `TRIM([[BOTH | TRAILING] [remstr] FROM] str)`, `REPLACE(str,from_str,to_str)`, `REVERSE(str)`, `INSERT(str,pos,len,newstr)`, `LCASE(str)`, `LOWER(str)`, `UCASE(str)` and `UPPER(str)`; patch by Wei He.
- Fix core dump when releasing a lock from a non-existent table.
- Remove locks on tables before starting to remove duplicates.
- Added option `FULL` to `SHOW PROCESSLIST`.
- Added option `--verbose` to `mysqladmin`.
- Fixed problem when automatically converting `HEAP` to `MyISAM`.
- Fixed bug in `HEAP` tables when doing insert + delete + insert + scan the table.
- Fixed bugs on Alpha with `REPLACE()` and `LOAD DATA INFILE`.
- Added `interactive_timeout` variable to `mysqld`.
- Changed the argument to `mysql_data_seek()` from `ulong` to `ulonglong`.

D.4.53 Changes in release 3.23.6

- Added `-O lower_case_table_names={0|1}` option to `mysqld` to allow users to force table names to lowercase.
- Added `SELECT ... INTO DUMPFILE`.
- Added `--ansi` option to `mysqld` to make some functions SQL-99 compatible.
- Temporary table names now start with `#sql`.
- Added quoting of identifiers with `'` (`"` in `--ansi` mode).
- Changed to use `snprintf()` when printing floats to avoid some buffer overflows on FreeBSD.
- Made `FLOOR()` overflow safe on FreeBSD.
- Added `--quote-names` option to `mysqldump`.

- Fixed bug that one could make a part of a PRIMARY KEY NOT NULL.
- Fixed `encrypt()` to be thread-safe and not reuse buffer.
- Added `mysql_odbc_escape_string()` function to support big5 characters in MyODBC.
- Rewrote the storage engine to use classes. This introduces a lot of new code, but will make table handling faster and better.
- Added patch by Sasha for user-defined variables.
- Changed that FLOAT and DOUBLE (without any length modifiers) no longer are fixed decimal point numbers.
- Changed the meaning of FLOAT(X): Now this is the same as FLOAT if $X \leq 24$ and a DOUBLE if $24 < X \leq 53$.
- DECIMAL(X) is now an alias for DECIMAL(X,0) and DECIMAL is now an alias for DECIMAL(10,0). The same goes for NUMERIC.
- Added option `ROW_FORMAT={default | dynamic | fixed | compressed}` to CREATE_TABLE.
- DELETE FROM `table_name` didn't work on temporary tables.
- Changed function CHAR_LENGTH() to be multi-byte character safe.
- Added function ORD(string).

D.4.54 Changes in release 3.23.5 (20 Oct 1999)

- Fixed some Y2K problems in the new date handling in 3.23.
- Fixed problem with SELECT DISTINCT ... ORDER BY RAND().
- Added patches by Sergei A. Golubchik for text searching on the MyISAM level.
- Fixed cache overflow problem when using full joins without keys.
- Fixed some configure issues.
- Some small changes to make parsing faster.
- Adding a column after the last field with ALTER TABLE didn't work.
- Fixed problem when using an AUTO_INCREMENT column in two keys
- With MyISAM, you now can have an AUTO_INCREMENT column as a key sub part: CREATE TABLE foo (a INT NOT NULL AUTO_INCREMENT, b CHAR(5), PRIMARY KEY (b,a))
- Fixed bug in MyISAM with packed char keys that could be NULL.
- AS on field name with CREATE TABLE `table_name` SELECT ... didn't work.
- Allow use of NATIONAL and NCHAR when defining character columns. This is the same as not using BINARY.
- Don't allow NULL columns in a PRIMARY KEY (only in UNIQUE keys).
- Clear LAST_INSERT_ID() if one uses this in ODBC: WHERE `auto_increment_column` IS NULL. This seems to fix some problems with Access.
- SET SQL_AUTO_IS_NULL=0|1 now turns on/off the handling of searching after the last inserted row with WHERE `auto_increment_column` IS NULL.
- Added new variable `concurrency` to `mysqld` for Solaris.

- Added `--relative` option to `mysqladmin` to make `extended-status` more useful to monitor changes.
- Fixed bug when using `COUNT(DISTINCT ...)` on an empty table.
- Added support for the Chinese character set GBK.
- Fixed problem with `LOAD DATA INFILE` and BLOB columns.
- Added bit operator `~` (negation).
- Fixed problem with UDF functions.

D.4.55 Changes in release 3.23.4 (28 Sep 1999)

- Inserting a `DATETIME` into a `TIME` column no longer will try to store 'days' in it.
- Fixed problem with storage of float/double on little endian machines. (This affected `SUM()`.)
- Added connect timeout on TCP/IP connections.
- Fixed problem with `LIKE "%"` on an index that may have NULL values.
- `REVOKE ALL PRIVILEGES` didn't revoke all privileges.
- Allow creation of temporary tables with same name as the original table.
- When granting a user a `GRANT` option for a database, he couldn't grant privileges to other users.
- New command: `SHOW GRANTS FOR user` (by Sinisa).
- New `date_add` syntax: `date/datetime + INTERVAL # interval_type`. By Joshua Chamas.
- Fixed privilege check for `LOAD DATA REPLACE`.
- Automatic fixing of broken include files on Solaris 2.7
- Some configure issues to fix problems with big filesystem detection.
- `REGEXP` is now case-insensitive if you use non-binary strings.

D.4.56 Changes in release 3.23.3

- Added patches for MIT-pthreads on NetBSD.
- Fixed range bug in MyISAM.
- `ASC` is now the default again for `ORDER BY`.
- Added `LIMIT` to `UPDATE`.
- Added `mysql_change_user()` function to the MySQL C API.
- Added character set to `SHOW VARIABLES`.
- Added support of `--[whitespace]` comments.
- Allow `INSERT into nome_tabela VALUES ()`, that is, you may now specify an empty value list to insert a row in which each column is set to its default value.
- Changed `SUBSTRING(text FROM pos)` to conform to SQL-99. (Before this construct returned the rightmost `pos` characters.)
- `SUM()` with `GROUP BY` returned 0 on some systems.
- Changed output for `SHOW TABLE STATUS`.

- Added DELAY_KEY_WRITE option to CREATE TABLE.
- Allow AUTO_INCREMENT on any key part.
- Fixed problem with YEAR(NOW()) and YEAR(CURDATE()).
- Added CASE construct.
- New function COALESCE().

D.4.57 Changes in release 3.23.2 (09 Aug 1999)

- Fixed range optimiser bug: SELECT * FROM table_name WHERE key_part1 >= const AND (key_part2 = const OR key_part2 = const). The bug was that some rows could be duplicated in the result.
- Running myisamchk without -a updated the index distribution incorrectly.
- SET SQL_LOW_PRIORITY_UPDATES=1 was causing a parse error.
- You can now update index columns that are used in the WHERE clause. UPDATE nome_tabela SET KEY=KEY+1 WHERE KEY > 100
- Date handling should now be a bit faster.
- Added handling of fuzzy dates (dates where day or month is 0), such as '1999-01-00'.
- Fixed optimisation of SELECT ... WHERE key_part1=const1 AND key_part_2=const2 AND key_part1=const4 AND key_part2=const4; indextype should be range instead of ref.
- Fixed egcs 1.1.2 optimiser bug (when using BLOBs) on Linux Alpha.
- Fixed problem with LOCK TABLES combined with DELETE FROM table.
- MyISAM tables now allow keys on NULL and BLOB/TEXT columns.
- The following join is now much faster: SELECT ... FROM t1 LEFT JOIN t2 ON ... WHERE t2.not_null_column IS NULL.
- ORDER BY and GROUP BY can be done on functions.
- Changed handling of 'const_item' to allow handling of ORDER BY RAND().
- Indexes are now used for WHERE key_column = function.
- Indexes are now used for WHERE key_column = nome_coluna even if the columns are not identically packed.
- Indexes are now used for WHERE nome_coluna IS NULL.
- Changed heap tables to be stored in low_byte_first order (to make it easy to convert to MyISAM tables)
- Automatic change of HEAP temporary tables to MyISAM tables in case of 'table is full' errors.
- Added --init-file=file_name option to mysqld.
- Added COUNT(DISTINCT value, [value, ...]).
- CREATE TEMPORARY TABLE now creates a temporary table, in its own namespace, that is automatically deleted if connection is dropped.
- New reserved words (required for CASE): CASE, THEN, WHEN, ELSE and END.
- New functions EXPORT_SET() and MD5().
- Support for the GB2312 Chinese character set.

D.4.58 Changes in release 3.23.1

- Fixed some compilation problems.

D.4.59 Changes in release 3.23.0 (05 Aug 1999: Alpha)

- A new storage engine library (MyISAM) with a lot of new features. See [\[MyISAM\]](#), page [\[undefined\]](#).
- You can create in-memory HEAP tables which are extremely fast for lookups.
- Support for big files (63-bit) on OSs that support big files.
- New function LOAD_FILE(filename) to get the contents of a file as a string value.
- New operator <=> which will act as = but will return TRUE if both arguments are NULL. This is useful for comparing changes between tables.
- Added the ODBC 3.0 EXTRACT(interval FROM datetime) function.
- Columns defined as FLOAT(X) are not rounded on storage and may be in scientific notation (1.0 E+10) when retrieved.
- REPLACE is now faster than before.
- Changed LIKE character comparison to behave as =; This means that 'e' LIKE 'é' is now true. (If the line doesn't display correctly, the latter 'e' is a French 'e' with a dot above.)
- SHOW TABLE STATUS returns a lot of information about the tables.
- Added LIKE to the SHOW STATUS command.
- Added Privileges column to SHOW COLUMNS.
- Added Packed and Comment columns to SHOW INDEX.
- Added comments to tables (with CREATE TABLE ... COMMENT "xxx").
- Added UNIQUE, as in CREATE TABLE table_name (col INT not null UNIQUE)
- New create syntax: CREATE TABLE table_name SELECT ...
- New create syntax: CREATE TABLE IF NOT EXISTS ...
- Allow creation of CHAR(0) columns.
- DATE_FORMAT() now requires '%' before any format character.
- DELAYED is now a reserved word (sorry about that :().
- An example procedure is added: analyse, file: 'sql_analyse.c'. This will describe the data in your query. Try the following:

```
SELECT ... FROM ...
WHERE ... PROCEDURE ANALYSE([max elements],[max memory])
```

This procedure is extremely useful when you want to check the data in your table!

- BINARY cast to force a string to be compared in case-sensitive fashion.
- Added --skip-show-database option to mysqld.
- Check whether a row has changed in an UPDATE now also works with BLOB/TEXT columns.
- Added the INNER join syntax. **NOTE:** This made INNER a reserved word!

- Added support for netmasks to the hostname in the MySQL grant tables. You can specify a netmask using the IP/NETMASK syntax.
- If you compare a NOT NULL DATE/DATETIME column with IS NULL, this is changed to a compare against 0 to satisfy some ODBC applications. (By shreeve@uci.edu.)
- NULL IN (...) now returns NULL instead of 0. This will ensure that null_column NOT IN (...) doesn't match NULL values.
- Fix storage of floating-point values in TIME columns.
- Changed parsing of TIME strings to be more strict. Now the fractional second part is detected (and currently skipped). The following formats are supported:


```
[[[DAYS] [H]H:]MM:]SS[.fraction]
[[[[[H]H]H]H]MM]SS[.fraction]
```
- Detect (and ignore) fractional second part from DATETIME.
- Added the LOW_PRIORITY attribute to LOAD DATA INFILE.
- The default index name now uses the same case as the column name on which the index name is based.
- Changed default number of connections to 100.
- Use bigger buffers when using LOAD DATA INFILE.
- DECIMAL(x,y) now works according to SQL-99.
- Added aggregate UDF functions. Thanks to Andreas F. Bobak (bobak@relog.ch) for this!
- LAST_INSERT_ID() is now updated for INSERT INTO ... SELECT.
- Some small changes to the join table optimiser to make some joins faster.
- SELECT DISTINCT is much faster; it uses the new UNIQUE functionality in MyISAM. One difference compared to MySQL Version 3.22 is that the output of DISTINCT is no longer sorted.
- All C client API macros are now functions to make shared libraries more reliable. Because of this, you can no longer call mysql_num_fields() on a MYSQL object, you must use mysql_field_count() instead.
- Added use of LIBWRAP; patch by Henning P. Schmiedehausen.
- Don't allow AUTO_INCREMENT for other than numerical columns.
- Using AUTO_INCREMENT will now automatically make the column NOT NULL.
- Show NULL as the default value for AUTO_INCREMENT columns.
- Added SQL_BIG_RESULT; SQL_SMALL_RESULT is now default.
- Added a shared library RPM. This enhancement was contributed by David Fox (dsfox@cogsci.ucsd.edu).
- Added --enable-large-files and --disable-large-files switches to configure. See 'configure.in' for some systems where this is automatically turned off because of broken implementations.
- Upgraded readline to 4.0.
- New CREATE TABLE options: PACK_KEYS and CHECKSUM.
- Added --default-table-type option to mysqld.

D.5 Changes in release 3.22.x (Old; discontinued)

The 3.22 version has faster and safer connect code than version 3.21, as well as a lot of new nice enhancements. As there aren't really any major changes, upgrading from 3.21 to 3.22 should be very easy and painless. See [from-3.21-snt \[Upgrading-from-3.21\]](#), page [from-3.21-pg](#).

D.5.1 Changes in release 3.22.35

- Fixed problem with `STD()`.
- Merged changes from the newest ISAM library from 3.23.
- Fixed problem with `INSERT DELAYED`.
- Fixed a bug core dump when using a `LEFT JOIN/STRAIGHT_JOIN` on a table with only one row.

D.5.2 Changes in release 3.22.34

- Fixed problem with `GROUP BY` on `TINYBLOB` columns; this caused bugzilla to not show rows in some queries.
- Had to do total recompile of the Windows binary version as VC++ didn't compile all relevant files for 3.22.33 :(

D.5.3 Changes in release 3.22.33

- Fixed problems in Windows when locking tables with `LOCK TABLE`.
- Quicker kill of `SELECT DISTINCT` queries.

D.5.4 Changes in release 3.22.32 (14 Feb 2000)

- Fixed problem when storing numbers in timestamps.
- Fix problem with timezones that have half hour offsets.
- Added `mysqlhotcopy`, a fast online hot-backup utility for local MySQL databases. By Tim Bunce.
- New more secure `mysqlaccess`. Thanks to Steve Harvey for this.
- Fixed security problem in the protocol regarding password checking.
- Fixed problem that affected queries that did arithmetic on `GROUP` functions.
- Fixed a bug in the ISAM code when deleting rows on tables with packed indexes.

D.5.5 Changes in release 3.22.31

- A few small fixes for the Windows version.

D.5.6 Changes in release 3.22.30

- Fixed optimiser problem on `SELECT` when using many overlapping indexes.
- Disabled floating-point exceptions for FreeBSD to fix core dump when doing `SELECT FLOOR(POW(2,63))`.

- Added print of default arguments options to all clients.
- Fixed critical problem with the `WITH GRANT OPTION` option.
- Fixed non-critical Y2K problem when writing short date to log files.

D.5.7 Changes in release 3.22.29 (02 Jan 2000)

- Upgraded the configure and include files to match the latest 3.23 version. This should increase portability and make it easier to build shared libraries.
- Added latest patches to MIT-pthreads for NetBSD.
- Fixed problem with timezones that are < GMT -11.
- Fixed a bug when deleting packed keys in NISAM.
- Fixed problem that could cause MySQL to touch freed memory when doing very complicated `GROUP BY` queries.
- Fixed core dump if you got a crashed table where an `ENUM` field value was too big.
- Added `mysqlshutdown.exe` and `mysqlwatch.exe` to the Windows distribution.
- Fixed problem when doing `ORDER BY` on a reference key.
- Fixed that `INSERT DELAYED` doesn't update timestamps that are given.

D.5.8 Changes in release 3.22.28 (20 Oct 1999)

- Fixed problem with `LEFT JOIN` and `COUNT()` on a column which was declared `NULL +` and it had a `DEFAULT` value.
- Fixed core dump problem when using `CONCAT()` in a `WHERE` clause.
- Fixed problem with `AVG()` and `STD()` with `NULL` values.

D.5.9 Changes in release 3.22.27

- Fixed prototype in `'my_ctype.h'` when using other character sets.
- Some configure issues to fix problems with big filesystem detection.
- Fixed problem when sorting on big `BLOB` columns.
- `ROUND()` will now work on Windows.

D.5.10 Changes in release 3.22.26 (16 Sep 1999)

- Fixed core dump with empty `BLOB/TEXT` column argument to `REVERSE()`.
- Extended `/*! */` with version numbers.
- Changed `SUBSTRING(text FROM pos)` to conform to SQL-99. (Before this construct returned the rightmost 'pos' characters.)
- Fixed problem with `LOCK TABLES` combined with `DELETE FROM table`
- Fixed problem that `INSERT ... SELECT` didn't use `BIG_TABLES`.
- `SET SQL_LOW_PRIORITY_UPDATES=#` didn't work.
- Password wasn't updated correctly if privileges didn't change on: `GRANT ... IDENTIFIED BY`

- Fixed range optimiser bug in `SELECT * FROM table_name WHERE key_part1 >= const AND (key_part2 = const OR key_part2 = const)`.
- Fixed bug in compression key handling in ISAM.

D.5.11 Changes in release 3.22.25

- Fixed some small problems with the installation.

D.5.12 Changes in release 3.22.24 (05 Jul 1999)

- `DATA` is no longer a reserved word.
- Fixed optimiser bug with tables with only one row.
- Fixed bug when using `LOCK TABLES table_name READ; FLUSH TABLES;`
- Applied some patches for HP-UX.
- `isamchk` should now work on Windows.
- Changed 'configure' to not use big file handling on Linux as this crashes some Red Hat 6.0 systems

D.5.13 Changes in release 3.22.23 (08 Jun 1999)

- Upgraded to use Autoconf 2.13, Automake 1.4 and libtool 1.3.2.
- Better support for SCO in `configure`.
- Added option `--defaults-file=###` to option file handling to force use of only one specific option file.
- Extended `CREATE` syntax to ignore MySQL Version 3.23 keywords.
- Fixed deadlock problem when using `INSERT DELAYED` on a table locked with `LOCK TABLES`.
- Fixed deadlock problem when using `DROP TABLE` on a table that was locked by another thread.
- Add logging of `GRANT/REVOKE` commands in the update log.
- Fixed `isamchk` to detect a new error condition.
- Fixed bug in `NATURAL LEFT JOIN`.

D.5.14 Changes in release 3.22.22 (30 Apr 1999)

- Fixed problem in the C API when you called `mysql_close()` directly after `mysql_init()`.
- Better client error message when you can't open socket.
- Fixed `delayed_insert_thread` counting when you couldn't create a new `delayed_insert` thread.
- Fixed bug in `CONCAT()` with many arguments.
- Added patches for DEC 3.2 and SCO.
- Fixed path-bug when installing MySQL as a service on NT.
- The MySQL-Windows version is now compiled with VC++ 6.0 instead of with VC++ 5.0.
- New installation setup for MySQL-Windows.

D.5.15 Changes in release 3.22.21

- Fixed problem with `DELETE FROM TABLE` when table was locked by another thread.
- Fixed bug in `LEFT JOIN` involving empty tables.
- Changed the `mysql.db` column from `CHAR(32)` to `CHAR(60)`.
- `MODIFY` and `DELAYED` are no longer reserved words.
- Fixed a bug when storing days in a `TIME` column.
- Fixed a problem with `Host '...' is not allowed to connect to this MySQL server` after one had inserted a new MySQL user with a `GRANT` command.
- Changed to use `TCP_NODELAY` also on Linux (should give faster TCP/IP connections).

D.5.16 Changes in release 3.22.20 (18 Mar 1999)

- Fixed `STD()` for big tables when result should be 0.
- The update log didn't have newlines on some operating systems.
- `INSERT DELAYED` had some garbage at end in the update log.

D.5.17 Changes in release 3.22.19 (Mar 1999: Production)

- Fixed bug in `mysql_install_db` (from 3.22.17).
- Changed default key cache size to 8M.
- Fixed problem with queries that needed temporary tables with `BLOB` columns.

D.5.18 Changes in release 3.22.18

- Fixes a fatal problem in 3.22.17 on Linux; after `shutdown` not all threads died properly.
- Added option `-O flush_time=#` to `mysqld`. This is mostly useful on Windows and tells how often MySQL should close all unused tables and flush all updated tables to disk.
- Fixed problem that a `VARCHAR` column compared with `CHAR` column didn't use keys efficiently.

D.5.19 Changes in release 3.22.17

- Fixed a core dump problem when using `--log-update` and connecting without a default database.
- Fixed some `configure` and portability problems.
- Using `LEFT JOIN` on tables that had circular dependencies caused `mysqld` to hang forever.

D.5.20 Changes in release 3.22.16 (Feb 1999: Gamma)

- `mysqladmin processlist` could kill the server if a new user logged in.
- `DELETE FROM nome_tabela WHERE key_column=nome_coluna` didn't find any matching rows. Fixed.
- `DATE_ADD(column, ...)` didn't work.
- `INSERT DELAYED` could deadlock with status 'upgrading lock'

- Extended `ENCRYPT()` to take longer salt strings than 2 characters.
- `longlong2str` is now much faster than before. For Intel x86 platforms, this function is written in optimised assembler.
- Added the `MODIFY` keyword to `ALTER TABLE`.

D.5.21 Changes in release 3.22.15

- `GRANT` used with `IDENTIFIED BY` didn't take effect until privileges were flushed.
- Name change of some variables in `SHOW STATUS`.
- Fixed problem with `ORDER BY` with 'only index' optimisation when there were multiple key definitions for a used column.
- `DATE` and `DATETIME` columns are now up to 5 times faster than before.
- `INSERT DELAYED` can be used to let the client do other things while the server inserts rows into a table.
- `LEFT JOIN USING (col1,col2)` didn't work if one used it with tables from 2 different databases.
- `LOAD DATA LOCAL INFILE` didn't work in the Unix version because of a missing file.
- Fixed problems with `VARCHAR/BLOB` on very short rows (< 4 bytes); error 127 could occur when deleting rows.
- Updating `BLOB/TEXT` through formulas didn't work for short (< 256 char) strings.
- When you did a `GRANT` on a new host, `mysqld` could die on the first connect from this host.
- Fixed bug when one used `ORDER BY` on column name that was the same name as an alias.
- Added `BENCHMARK(loop_count,expression)` function to time expressions.

D.5.22 Changes in release 3.22.14

- Allow empty arguments to `mysqld` to make it easier to start from shell scripts.
- Setting a `TIMESTAMP` column to `NULL` didn't record the timestamp value in the update log.
- Fixed lock handler bug when one did `INSERT INTO TABLE ... SELECT ... GROUP BY`.
- Added a patch for `localtime_r()` on Windows so that it will no lonher crash if your date is > 2039, but instead will return a time of all zero.
- Names for user-defined functions are no longer case-sensitive.
- Added escape of `^Z` (ASCII 26) to `\Z` as `^Z` doesn't work with pipes on Windows.
- `mysql_fix_privileges` adds a new column to the `mysql.func` to support aggregate UDF functions in future MySQL releases.

D.5.23 Changes in release 3.22.13

- Saving `NOW()`, `CURDATE()` or `CURTIME()` directly in a column didn't work.
- `SELECT COUNT(*) ... LEFT JOIN ...` didn't work with no `WHERE` part.
- Updated 'config.guess' to allow MySQL to configure on UnixWare 7.0.x.

- Changed the implementation of `pthread_cond()` on the Windows version. `get_lock()` now correctly times out on Windows!

D.5.24 Changes in release 3.22.12

- Fixed problem when using `DATE_ADD()` and `DATE_SUB()` in a `WHERE` clause.
- You can now set the password for a user with the `GRANT ... TO user IDENTIFIED BY 'password'` syntax.
- Fixed bug in `GRANT` checking with `SELECT` on many tables.
- Added missing file `mysql_fix_privilege_tables` to the RPM distribution. This is not run by default because it relies on the client package.
- Added option `SQL_SMALL_RESULT` to `SELECT` to force use of fast temporary tables when you know that the result set will be small.
- Allow use of negative real numbers without a decimal point.
- Day number is now adjusted to maximum days in month if the resulting month after `DATE_ADD/DATE_SUB()` doesn't have enough days.
- Fix that `GRANT` compares columns in case-insensitive fashion.
- Fixed a bug in `'sql_list.h'` that made `ALTER TABLE` dump core in some contexts.
- The hostname in `user@hostname` can now include `'.'` and `'-'` without quotes in the context of the `GRANT`, `REVOKE` and `SET PASSWORD FOR ...` statements.
- Fix for `isamchk` for tables which need big temporary files.

D.5.25 Changes in release 3.22.11

- **Important:** You must run the `mysql_fix_privilege_tables` script when you upgrade to this version! This is needed because of the new `GRANT` system. If you don't do this, you will get `Access denied` when you try to use `ALTER TABLE`, `CREATE INDEX`, or `DROP INDEX`.
- `GRANT` to allow/deny users table and column access.
- Changed `USER()` to return a value in `user@host` format. Formerly it returned only `user`.
- Changed the syntax for how to set `PASSWORD` for another user.
- New command `FLUSH STATUS` that resets most status variables to zero.
- New status variables: `aborted_threads`, `aborted_connects`.
- New option variable: `connection_timeout`.
- Added support for Thai sorting (by Pruet Boonma pruet@ds90.intanon.nectec.or.th).
- Slovak and Japanese error messages.
- Configuration and portability fixes.
- Added option `SET SQL_WARNINGS=1` to get a warning count also for simple inserts.
- MySQL now uses `SIGTERM` instead of `SIGQUIT` with shutdown to work better on FreeBSD.
- Added option `\G` (print vertically) to `mysql`.
- `SELECT HIGH_PRIORITY ...` killed `mysqld`.

- IS NULL on a AUTO_INCREMENT column in a LEFT JOIN didn't work as expected.
- New function MAKE_SET().

D.5.26 Changes in release 3.22.10

- `mysql_install_db` no longer starts the MySQL server! You should start `mysqld` with `safe_mysqld` after installing it! The MySQL RPM will, however, start the server as before.
- Added `--bootstrap` option to `mysqld` and recoded `mysql_install_db` to use it. This will make it easier to install MySQL with RPMs.
- Changed `+`, `-` (sign and minus), `*`, `/`, `%`, `ABS()` and `MOD()` to be BIGINT aware (64-bit safe).
- Fixed a bug in `ALTER TABLE` that caused `mysqld` to crash.
- MySQL now always reports the conflicting key values when a duplicate key entry occurs. (Before this was only reported for `INSERT`.)
- New syntax: `INSERT INTO nome_tabela SET nome_coluna=value, nome_coluna=value, ...`
- Most errors in the `.err` log are now prefixed with a time stamp.
- Added option `MYSQL_INIT_COMMAND` to `mysql_options()` to make a query on connect or reconnect.
- Added option `MYSQL_READ_DEFAULT_FILE` and `MYSQL_READ_DEFAULT_GROUP` to `mysql_options()` to read the following parameters from the MySQL option files: `port`, `socket`, `compress`, `password`, `pipe`, `timeout`, `user`, `init-command`, `host` and `database`.
- Added `maybe_null` to the UDF structure.
- Added option `IGNORE` to `INSERT` statements with many rows.
- Fixed some problems with sorting of the `koi8` character sets; users of `koi8` **must** run `isamchk -rq` on each table that has an index on a `CHAR` or `VARCHAR` column.
- New script `mysql_setpermission`, by Luuk de Boer. It allows easy creation of new users with permissions for specific databases.
- Allow use of hexadecimal strings (`0x...`) when specifying a constant string (like in the column separators with `LOAD DATA INFILE`).
- Ported to OS/2 (thanks to Antony T. Curtis antony.curtis@olcs.net).
- Added more variables to `SHOW STATUS` and changed format of output to be like `SHOW VARIABLES`.
- Added `extended-status` command to `mysqladmin` which will show the new status variables.

D.5.27 Changes in release 3.22.9

- `SET SQL_LOG_UPDATE=0` caused a lockup of the server.
- New SQL command: `FLUSH [TABLES | HOSTS | LOGS | PRIVILEGES] [, ...]`
- New SQL command: `KILL thread_id`.

- Added casts and changed include files to make MySQL easier to compile on AIX and DEC OSF/1 4.x
- Fixed conversion problem when using ALTER TABLE from a INT to a short CHAR() column.
- Added SELECT HIGH_PRIORITY; this will get a lock for the SELECT even if there is a thread waiting for another SELECT to get a WRITE LOCK.
- Moved wild_compare() to string class to be able to use LIKE on BLOB/TEXT columns with \0.
- Added ESCAPE option to LIKE.
- Added a lot more output to mysqladmin debug.
- You can now start mysqld on Windows with the --flush option. This will flush all tables to disk after each update. This makes things much safer on the Windows platforms but also **much** slower.

D.5.28 Changes in release 3.22.8

- Czech character sets should now work much better. You must also install <http://www.mysql.com/Downloads/Patches/czech-3.22.8-patch>. This patch should also be installed if you are using a character set which uses my_strcoll()! The patch should always be safe to install (for any system), but as this patch changes ISAM internals it's not yet in the default distribution.
- DATE_ADD() and DATE_SUB() didn't work with group functions.
- mysql will now also try to reconnect on USE DATABASE commands.
- Fix problem with ORDER BY and LEFT JOIN and const tables.
- Fixed problem with ORDER BY if the first ORDER BY column was a key and the rest of the ORDER BY columns wasn't part of the key.
- Fixed a big problem with OPTIMIZE TABLE.
- MySQL clients on NT will now by default first try to connect with named pipes and after this with TCP/IP.
- Fixed a problem with DROP TABLE and mysqladmin shutdown on Windows (a fatal bug from 3.22.6).
- Fixed problems with TIME columns and negative strings.
- Added an extra thread signal loop on shutdown to avoid some error messages from the client.
- MySQL now uses the next available number as extension for the update log file.
- Added patches for UNIXWARE 7.

D.5.29 Changes in release 3.22.7 (Sep 1998: Beta)

- Added LIMIT clause for the DELETE statement.
- You can now use the /*! ... */ syntax to hide MySQL-specific keywords when you write portable code. MySQL will parse the code inside the comments as if the surrounding /*! and */ comment characters didn't exist.

- `OPTIMIZE TABLE nome_tabela` can now be used to reclaim disk space after many deletes. Currently, this uses `ALTER TABLE` to regenerate the table, but in the future it will use an integrated `isamchk` for more speed.
- Upgraded `libtool` to get the configure more portable.
- Fixed slow `UPDATE` and `DELETE` operations when using `DATETIME` or `DATE` keys.
- Changed optimiser to make it better at deciding when to do a full join and when using keys.
- You can now use `mysqladmin proc` to display information about your own threads. Only users with the `PROCESS` privilege can get information about all threads. (In 4.0.2 one needs the `SUPER` privilege for this.)
- Added handling of formats `YYMMDD`, `YYYYMMDD`, `YYMMDDHHMMSS` for numbers when using `DATETIME` and `TIMESTAMP` types. (Formerly these formats only worked with strings.)
- Added connect option `CLIENT_IGNORE_SPACE` to allow use of spaces after function names and before `'` (Powerbuilder requires this). This will make all function names reserved words.
- Added the `--log-long-format` option to `mysqld` to enable timestamps and `INSERT_IDS` in the update log.
- Added `--where` option to `mysqldump` (patch by Jim Faucette).
- The lexical analyser now uses “perfect hashing” for faster parsing of SQL statements.

D.5.30 Changes in release 3.22.6

- Faster `mysqldump`.
- For the `LOAD DATA INFILE` statement, you can now use the new `LOCAL` keyword to read the file from the client. `mysqlimport` will automatically use `LOCAL` when importing with the TCP/IP protocol.
- Fixed small optimise problem when updating keys.
- Changed makefiles to support shared libraries.
- MySQL-NT can now use named pipes, which means that you can now use MySQL-NT without having to install TCP/IP.

D.5.31 Changes in release 3.22.5

- All table lock handing is changed to avoid some very subtle deadlocks when using `DROP TABLE`, `ALTER TABLE`, `DELETE FROM TABLE` and `mysqladmin flush-tables` under heavy usage. Changed locking code to get better handling of locks of different types.
- Updated DBI to 1.00 and DBD to 1.2.0.
- Added a check that the error message file contains error messages suitable for the current version of `mysqld`. (To avoid errors if you accidentally try to use an old error message file.)
- All count structures in the client (`affected_rows()`, `insert_id()`, ...) are now of type `BIGINT` to allow 64-bit values to be used. This required a minor change in the MySQL protocol which should affect only old clients when using tables with `AUTO_INCREMENT` values > 16M.

- The return type of `mysql_fetch_lengths()` has changed from `uint *` to `ulong *`. This may give a warning for old clients but should work on most machines.
- Change `mysys` and `debug` libraries to allocate all thread variables in one struct. This makes it easier to make a threaded `'libmysql.dll'` library.
- Use the result from `gethostname()` (instead of `uname()`) when constructing `'pid'` file names.
- New better compressed server/client protocol.
- `COUNT()`, `STD()` and `AVG()` are extended to handle more than 4G rows.
- You can now store values in the range `-838:59:59 <= x <= 838:59:59` in a `TIME` column.
- **Warning: incompatible change!!** If you set a `TIME` column to too short a value, MySQL now assumes the value is given as: `[[D]HH:]MM:]SS` instead of `HH[:MM[:SS]]`.
- `TIME_TO_SEC()` and `SEC_TO_TIME()` can now handle negative times and hours up to 32767.
- Added new option `SET SQL_LOG_UPDATE={0|1}` to allow users with the `PROCESS` privilege to bypass the update log. (Modified patch from Sergey A Mukhin violet@rosnet.net.)
- Fixed fatal bug in `LPAD()`.
- Initialise line buffer in `'mysql.cc'` to make `BLOB` reading from pipes safer.
- Added `-O max_connect_errors=#` option to `mysqld`. Connect errors are now reset for each correct connection.
- Increased the default value of `max_allowed_packet` to 1M in `mysqld`.
- Added `--low-priority-updates` option to `mysqld`, to give table-modifying operations (`INSERT`, `REPLACE`, `UPDATE`, `DELETE`) lower priority than retrievals. You can now use `{INSERT | REPLACE | UPDATE | DELETE} LOW_PRIORITY ...`. You can also use `SET SQL_LOW_PRIORITY_UPDATES={0|1}` to change the priority for one thread. One side effect is that `LOW_PRIORITY` is now a reserved word. :(
- Add support for `INSERT INTO table ... VALUES(...),(...),(...)`, to allow inserting multiple rows with a single statement.
- `INSERT INTO nome_tabela` is now also cached when used with `LOCK TABLES`. (Previously only `INSERT ... SELECT` and `LOAD DATA INFILE` were cached.)
- Allow `GROUP BY` functions with `HAVING`:


```
mysql> SELECT col FROM table GROUP BY col HAVING COUNT(*)>0;
```
- `mysqld` will now ignore trailing `';` characters in queries. This is to make it easier to migrate from some other SQL servers that require the trailing `';`.
- Fix for corrupted fixed-format output generated by `SELECT INTO OUTFILE`.
- **Warning: incompatible change!** Added Oracle `GREATEST()` and `LEAST()` functions. You must now use these instead of the `MAX()` and `MIN()` functions to get the largest/smallest value from a list of values. These can now handle `REAL`, `BIGINT` and string (`CHAR` or `VARCHAR`) values.
- **Warning: incompatible change!** `DAYOFWEEK()` had offset 0 for Sunday. Changed the offset to 1.

- Give an error for queries that mix **GROUP BY** columns and fields when there is no **GROUP BY** specification.
- Added `--vertical` option to `mysql`, for printing results in vertical mode.
- Index-only optimisation; some queries are now resolved using only indexes. Until MySQL 4.0, this works only for numeric columns. See [\[MySQL indexes\]](#), page [\[MySQL indexes\]](#).
- Lots of new benchmarks.
- A new C API chapter and lots of other improvements in the manual.

D.5.32 Changes in release 3.22.4

- Added `--tmpdir` option to `mysqld`, for specifying the location of the temporary file directory.
- MySQL now automatically changes a query from an ODBC client:


```
SELECT ... FROM table WHERE auto_increment_column IS NULL
```

 to:


```
SELECT ... FROM table WHERE auto_increment_column == LAST_INSERT_ID()
```

 This allows some ODBC programs (Delphi, Access) to retrieve the newly inserted row to fetch the `AUTO_INCREMENT` id.
- `DROP TABLE` now waits for all users to free a table before deleting it.
- Fixed small memory leak in the new connect protocol.
- New functions `BIN()`, `OCT()`, `HEX()` and `CONV()` for converting between different number bases.
- Added function `SUBSTRING()` with 2 arguments.
- If you created a table with a record length smaller than 5, you couldn't delete rows from the table.
- Added optimisation to remove const reference tables from `ORDER BY` and `GROUP BY`.
- `mysqld` now automatically disables system locking on Linux and Windows, and for systems that use MIT-pthreads. You can force the use of locking with the `--enable-external-locking` option.
- Added `--console` option to `mysqld`, to force a console window (for error messages) when using Windows.
- Fixed table locks for Windows.
- Allow '\$' in identifiers.
- Changed name of user-specific configuration file from `'my.cnf'` to `'my.cnf'` (Unix only).
- Added `DATE_ADD()` and `DATE_SUB()` functions.

D.5.33 Changes in release 3.22.3

- Fixed a lock problem (bug in MySQL Version 3.22.1) when closing temporary tables.
- Added missing `mysql_ping()` to the client library.
- Added `--compress` option to all MySQL clients.
- Changed `byte` to `char` in `'mysql.h'` and `'mysql_com.h'`.

D.5.34 Changes in release 3.22.2

- Searching on multiple constant keys that matched more than 30% of the rows didn't always use the best possible key.
- New functions `<<`, `>>`, `RPAD()` and `LPAD()`.
- You can now save default options (like passwords) in a configuration file (`'my.cnf'`).
- Lots of small changes to get `ORDER BY` to work when no records are found when using fields that are not in `GROUP BY` (MySQL extension).
- Added `--chroot` option to `mysqld`, to start `mysqld` in a chroot environment (by Nikki Chumakov nikkic@cityline.ru).
- Trailing spaces are now ignored when comparing case-sensitive strings; this should fix some problems with ODBC and flag 512!
- Fixed a core dump bug in the range optimiser.
- Added `--one-thread` option to `mysqld`, for debugging with LinuxThreads (or `glibc`). (This replaces the `-T32` flag)
- Added `DROP TABLE IF EXISTS` to prevent an error from occurring if the table doesn't exist.
- `IF` and `EXISTS` are now reserved words (they would have to be sooner or later).
- Added lots of new options to `mysqldump`.
- Server error messages are now in `'mysqld_error.h'`.
- The server/client protocol now supports compression.
- All bug fixes from MySQL Version 3.21.32.

D.5.35 Changes in release 3.22.1 (Jun 1998: Alpha)

- Added new C API function `mysql_ping()`.
- Added new API functions `mysql_init()` and `mysql_options()`. You now **MUST** call `mysql_init()` before you call `mysql_real_connect()`. You don't have to call `mysql_init()` if you only use `mysql_connect()`.
- Added `mysql_options(...,MYSQL_OPT_CONNECT_TIMEOUT,...)` so you can set a timeout for connecting to a server.
- Added `--timeout` option to `mysqladmin`, as a test of `mysql_options()`.
- Added `AFTER` column and `FIRST` options to `ALTER TABLE ... ADD columns`. This makes it possible to add a new column at some specific location within a row in an existing table.
- `WEEK()` now takes an optional argument to allow handling of weeks when the week starts on Monday (some European countries). By default, `WEEK()` assumes the week starts on Sunday.
- `TIME` columns weren't stored properly (bug in MySQL Version 3.22.0).
- `UPDATE` now returns information about how many rows were matched and updated, and how many "warnings" occurred when doing the update.
- Fixed incorrect result from `FORMAT(-100,2)`.
- `ENUM` and `SET` columns were compared in binary (case-sensitive) fashion; changed to be case-insensitive.

D.5.36 Changes in release 3.22.0

- New (backward-compatible) connect protocol that allows you to specify the database to use when connecting, to get much faster connections to a specific database.

The `mysql_real_connect()` call is changed to:

```
mysql_real_connect(MYSQL *mysql, const char *host, const char *user,
                  const char *passwd, const char *db, uint port,
                  const char *unix_socket, uint client_flag)
```

- Each connection is handled by its own thread, rather than by the master `accept()` thread. This fixes permanently the telnet bug that was a topic on the mail list some time ago.
- All TCP/IP connections are now checked with backward-resolution of the hostname to get better security. `mysqld` now has a local hostname resolver cache so connections should actually be faster than before, even with this feature.
- A site automatically will be blocked from future connections if someone repeatedly connects with an “improper header” (like when one uses telnet).
- You can now refer to tables in different databases with references of the form `nome_tabela@db_name` or `db_name.nome_tabela`. This makes it possible to give a user read access to some tables and write access to others simply by keeping them in different databases!
- Added `--user` option to `mysqld`, to allow it to run as another Unix user (if it is started as the Unix `root` user).
- Added caching of users and access rights (for faster access rights checking)
- Normal users (not anonymous ones) can change their password with `mysqladmin password 'new_password'`. This uses encrypted passwords that are not logged in the normal MySQL log!
- All important string functions are now coded in assembler for x86 Linux machines. This gives a speedup of 10% in many cases.
- For tables that have many columns, the column names are now hashed for much faster column name lookup (this will speed up some benchmark tests a lot!)
- Some benchmarks are changed to get better individual timing. (Some loops were so short that a specific test took < 2 seconds. The loops have been changed to take about 20 seconds to make it easier to compare different databases. A test that took 1-2 seconds before now takes 11-24 seconds, which is much better)
- Re-arranged `SELECT` code to handle some very specific queries involving group functions (like `COUNT(*)`) without a `GROUP BY` but with `HAVING`. The following now works:


```
mysql> SELECT COUNT(*) as C FROM table HAVING C > 1;
```
- Changed the protocol for field functions to be faster and avoid some calls to `malloc()`.
- Added `-T32` option to `mysqld`, for running all queries under the main thread. This makes it possible to debug `mysqld` under Linux with `gdb`!
- Added optimisation of `not_null_column IS NULL` (needed for some Access queries).
- Allow `STRAIGHT_JOIN` to be used between two tables to force the optimiser to join them in a specific order.

- String functions now return `VARCHAR` rather than `CHAR` and the column type is now `VARCHAR` for fields saved as `VARCHAR`. This should make the MyODBC driver better, but may break some old MySQL clients that don't handle `FIELD_TYPE_VARCHAR` the same way as `FIELD_TYPE_CHAR`.
- `CREATE INDEX` and `DROP INDEX` are now implemented through `ALTER TABLE`. `CREATE TABLE` is still the recommended (fast) way to create indexes.
- Added `--set-variable` option `wait_timeout` to `mysqld`.
- Added time column to `mysqladmin processlist` to show how long a query has taken or how long a thread has slept.
- Added lots of new variables to `show variables` and some new to `show status`.
- Added new type `YEAR`. `YEAR` is stored in 1 byte with allowable values of 0, and 1901 to 2155.
- Added new `DATE` type that is stored in 3 bytes rather than 4 bytes. All new tables are created with the new date type if you don't use the `--old-protocol` option to `mysqld`.
- Fixed bug in record caches; for some queries, you could get `Error from table handler: #` on some operating systems.
- Added `--enable-asm` option to `configure`, for x86 machines (tested on Linux + gcc). This will enable assembler functions for the most important string functions for more speed!

D.6 Changes in release 3.21.x

Version 3.21 is quite old now, and should be avoided if possible. This information is kept here for historical purposes only.

D.6.1 Changes in release 3.21.33

- Fixed problem when sending `SIGHUP` to `mysqld`; `mysqld` core dumped when starting from boot on some systems.
- Fixed problem with losing a little memory for some connections.
- `DELETE FROM nome_tabela` without a `WHERE` condition is now done the long way when you use `LOCK TABLES` or if the table is in use, to avoid race conditions.
- `INSERT INTO TABLE (timestamp_column) VALUES (NULL)`; didn't set timestamp.

D.6.2 Changes in release 3.21.32

- Fixed some possible race conditions when doing many `reopen/close` on the same tables under heavy load! This can happen if you execute `mysqladmin refresh` often. This could in some very rare cases corrupt the header of the index file and cause error 126 or 138.
- Fixed fatal bug in `refresh()` when running with the `--skip-external-locking` option. There was a "very small" time gap after a `mysqladmin refresh` when a table could be corrupted if one thread updated a table while another thread did `mysqladmin refresh` and another thread started a new update on the same table before the first thread had finished. A `refresh` (or `--flush-tables`) will now not return until all used tables are closed!

- `SELECT DISTINCT` with a `WHERE` clause that didn't match any rows returned a row in some contexts (bug only in 3.21.31).
- `GROUP BY + ORDER BY` returned one empty row when no rows were found.
- Fixed a bug in the range optimiser that wrote `Use_count: Wrong count for ...` in the error log file.

D.6.3 Changes in release 3.21.31

- Fixed a sign extension problem for the `TINYINT` type on Irix.
- Fixed problem with `LEFT("constant_string",function)`.
- Fixed problem with `FIND_IN_SET()`.
- `LEFT JOIN` core dumped if the second table is used with a constant `WHERE/ON` expression that uniquely identifies one record.
- Fixed problems with `DATE_FORMAT()` and incorrect dates. `DATE_FORMAT()` now ignores `'%'` to make it possible to extend it more easily in the future.

D.6.4 Changes in release 3.21.30

- `mysql` now returns an exit code `> 0` if the query returned an error.
- Saving of command-line history to file in `mysql` client. By Tommy Larsen tommy@mix.hive.no.
- Fixed problem with empty lines that were ignored in `'mysql.cc'`.
- Save the pid of the signal handler thread in the pid file instead of the pid of the main thread.
- Added patch by tommy@valley.ne.jp to support Japanese characters SJIS and UJIS.
- Changed `safe_mysqld` to redirect startup messages to `'hostname'.err` instead of `'hostname'.log` to reclaim file space on `mysqladmin refresh`.
- `ENUM` always had the first entry as default value.
- `ALTER TABLE` wrote two entries to the update log.
- `sql_acc()` now closes the `mysql` grant tables after a reload to save table space and memory.
- Changed `LOAD DATA` to use less memory with tables and `BLOB` columns.
- Sorting on a function which made a division `/ 0` produced a wrong set in some cases.
- Fixed `SELECT` problem with `LEFT()` when using the `czech` character set.
- Fixed problem in `isamchk`; it couldn't repair a packed table in a very unusual case.
- `SELECT` statements with `&` or `|` (bit functions) failed on columns with `NULL` values.
- When comparing a field `=` field, where one of the fields was a part key, only the length of the part key was compared.

D.6.5 Changes in release 3.21.29

- `LOCK TABLES + DELETE from nome_tabela` never removed locks properly.
- Fixed problem when grouping on an `OR` function.

- Fixed permission problem with `umask()` and creating new databases.
- Fixed permission problem on result file with `SELECT ... INTO OUTFILE ...`.
- Fixed problem in range optimiser (core dump) for a very complex query.
- Fixed problem when using `MIN(integer)` or `MAX(integer)` in `GROUP BY`.
- Fixed bug on Alpha when using integer keys. (Other keys worked on Alpha.)
- Fixed bug in `WEEK("XXXX-xx-01")`.

D.6.6 Changes in release 3.21.28

- Fixed socket permission (clients couldn't connect to Unix socket on Linux).
- Fixed bug in record caches; for some queries, you could get `Error from table handler: #` on some operating systems.

D.6.7 Changes in release 3.21.27

- Added user level lock functions `GET_LOCK(string,timeout)`, `RELEASE_LOCK(string)`.
- Added `Opened_tables` to show status.
- Changed connect timeout to 3 seconds to make it somewhat harder for crackers to kill `mysqld` through telnet + TCP/IP.
- Fixed bug in range optimiser when using `WHERE key_part_1 >= something AND key_part_2 <= something_else`.
- Changed `configure` for detection of FreeBSD 3.0 9803xx and above
- `WHERE` with `string_col_key = constant_string` didn't always find all rows if the column had many values differing only with characters of the same sort value (like e and é).
- Strings keys looked up with 'ref' were not compared in case-sensitive fashion.
- Added `umask()` to make log files non-readable for normal users.
- Ignore users with old (8-byte) password on startup if not using `--old-protocol` option to `mysqld`.
- `SELECT` which matched all key fields returned the values in the case of the matched values, not of the found values. (Minor problem.)

D.6.8 Changes in release 3.21.26

- `FROM_DAYS(0)` now returns "0000-00-00".
- In `DATE_FORMAT()`, PM and AM were swapped for hours 00 and 12.
- Extended the default maximum key size to 256.
- Fixed bug when using BLOB/TEXT in `GROUP BY` with many tables.
- An `ENUM` field that is not declared `NOT NULL` has `NULL` as the default value. (Previously, the default value was the first enumeration value.)
- Fixed bug in the join optimiser code when using many part keys on the same key: `INDEX (Organisation,Surname(35),Initials(35))`.
- Added some tests to the table order optimiser to get some cases with `SELECT ... FROM many_tables` much faster.

- Added a retry loop around `accept()` to possibly fix some problems on some Linux machines.

D.6.9 Changes in release 3.21.25

- Changed typedef `'string'` to typedef `'my_string'` for better portability.
- You can now kill threads that are waiting on a disk-full condition.
- Fixed some problems with UDF functions.
- Added long options to `isamchk`. Try `isamchk --help`.
- Fixed a bug when using 8 bytes long (alpha); `filesort()` didn't work. Affects `DISTINCT`, `ORDER BY` and `GROUP BY` on 64-bit processors.

D.6.10 Changes in release 3.21.24

- Dynamic loadable functions. Based on source from Alexis Mikhailov.
- You couldn't delete from a table if no one had done a `SELECT` on the table.
- Fixed problem with range optimiser with many `OR` operators on key parts inside each other.
- Recoded `MIN()` and `MAX()` to work properly with strings and `HAVING`.
- Changed default `umask` value for new files from 0664 to 0660.
- Fixed problem with `LEFT JOIN` and constant expressions in the `ON` part.
- Added Italian error messages from brenno@dewinter.com.
- `configure` now works better on OSF/1 (tested on 4.0D).
- Added hooks to allow `LIKE` optimisation with international character support.
- Upgraded DBI to 0.93.

D.6.11 Changes in release 3.21.23

- The following symbols are now reserved words: `TIME`, `DATE`, `TIMESTAMP`, `TEXT`, `BIT`, `ENUM`, `NO`, `ACTION`, `CHECK`, `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE`, `SECOND`, `STATUS`, `VARIABLES`.
- Setting a `TIMESTAMP` to `NULL` in `LOAD DATA INFILE ...` didn't set the current time for the `TIMESTAMP`.
- Fix `BETWEEN` to recognise binary strings. Now `BETWEEN` is case-sensitive.
- Added `--skip-thread-priority` option to `mysqld`, for systems where `mysqld`'s thread scheduling doesn't work properly (BSDI 3.1).
- Added ODBC functions `DAYNAME()` and `MONTHNAME()`.
- Added function `TIME_FORMAT()`. This works like `DATE_FORMAT()`, but takes a time string (`'HH:MM:DD'`) as argument.
- Fixed unlikely(?) key optimiser bug when using `ORs` of key parts inside `ANDs`.
- Added `variables` command to `mysqladmin`.
- A lot of small changes to the binary releases.
- Fixed a bug in the new protocol from MySQL Version 3.21.20.
- Changed `ALTER TABLE` to work with Windows (Windows can't rename open files). Also fixed a couple of small bugs in the Windows version.

- All standard MySQL clients are now ported to MySQL-Windows.
- MySQL can now be started as a service on NT.

D.6.12 Changes in release 3.21.22

- Starting with this version, all MySQL distributions will be configured, compiled and tested with `crash-me` and the benchmarks on the following platforms: SunOS 5.6 sun4u, SunOS 5.5.1 sun4u, SunOS 4.14 sun4c, SunOS 5.6 i86pc, Irix 6.3 mips5k, HP-UX 10.20 hppa, AIX 4.2.1 ppc, OSF/1 V4.0 alpha, FreeBSD 2.2.2 i86pc and BSDI 3.1 i386.
- Fix `COUNT(*)` problems when the `WHERE` clause didn't match any records. (Bug from 3.21.17.)
- Removed that `NULL = NULL` is true. Now you must use `IS NULL` or `IS NOT NULL` to test whether a value is `NULL`. (This is according to SQL-99 but may break old applications that are ported from `mSQL`.) You can get the old behaviour by compiling with `-DmSQL_COMPLIANT`.
- Fixed bug that core dumped when using many `LEFT OUTER JOIN` clauses.
- Fixed bug in `ORDER BY` on string formula with possible `NULL` values.
- Fixed problem in range optimiser when using `<=` on sub index.
- Added functions `DAYOFYEAR()`, `DAYOFMONTH()`, `MONTH()`, `YEAR()`, `WEEK()`, `QUARTER()`, `HOUR()`, `MINUTE()`, `SECOND()` and `FIND_IN_SET()`.
- Added `SHOW VARIABLES` command.
- Added support of "long constant strings" from SQL-99:


```
mysql> SELECT 'first ' 'second';          -> 'first second'
```
- Upgraded `Msql-Mysql-modules` to 1.1825.
- Upgraded `mysqlaccess` to 2.02.
- Fixed problem with Russian character set and `LIKE`.
- Ported to OpenBSD 2.1.
- New Dutch error messages.

D.6.13 Changes in release 3.21.21a

- Configure changes for some operating systems.

D.6.14 Changes in release 3.21.21

- Fixed optimiser bug when using `WHERE data_field = date_field2 AND date_field2 = constant`.
- Added `SHOW STATUS` command.
- Removed `'manual.ps'` from the source distribution to make it smaller.

D.6.15 Changes in release 3.21.20

- Changed the maximum table name and column name lengths from 32 to 64.
- Aliases can now be of "any" length.

- Fixed `mysqladmin stat` to return the right number of queries.
- Changed protocol (downward compatible) to mark if a column has the `AUTO_INCREMENT` attribute or is a `TIMESTAMP`. This is needed for the new Java driver.
- Added Hebrew sorting order by Zeev Suraski.
- Solaris 2.6: Fixed `configure` bugs and increased maximum table size from 2G to 4G.

D.6.16 Changes in release 3.21.19

- Upgraded DBD to 1.1823. This version implements `mysql_use_result` in DBD-MySQL.
- Benchmarks updated for `empress` (by Luuk).
- Fixed a case of slow range searching.
- `Configure` fixes ('Docs' directory).
- Added function `REVERSE()` (by Zeev Suraski).

D.6.17 Changes in release 3.21.18

- Issue error message if client C functions are called in wrong order.
- Added automatic reconnect to the '`libmysql.c`' library. If a write command fails, an automatic reconnect is done.
- Small sort sets no longer use temporary files.
- Upgraded DBI to 0.91.
- Fixed a couple of problems with `LEFT OUTER JOIN`.
- Added `CROSS JOIN` syntax. `CROSS` is now a reserved word.
- Recoded `yacc/bison` stack allocation to be even safer and to allow MySQL to handle even bigger expressions.
- Fixed a couple of problems with the update log.
- `ORDER BY` was slow when used with key ranges.

D.6.18 Changes in release 3.21.17

- Changed documentation string of `--with-unix-socket-path` to avoid confusion.
- Added ODBC and SQL-99 style `LEFT OUTER JOIN`.
- The following are new reserved words: `LEFT`, `NATURAL`, `USING`.
- The client library now uses the value of the environment variable `MYSQL_HOST` as the default host if it's defined.
- `SELECT nome_coluna, SUM(expr)` now returns `NULL` for `nome_coluna` when there are matching rows.
- Fixed problem with comparing binary strings and BLOBs with ASCII characters over 127.
- Fixed lock problem: when freeing a read lock on a table with multiple read locks, a thread waiting for a write lock would have been given the lock. This shouldn't affect data integrity, but could possibly make `mysqld` restart if one thread was reading data that another thread modified.

- `LIMIT offset,count` didn't work in `INSERT ... SELECT`.
- Optimised key block caching. This will be quicker than the old algorithm when using bigger key caches.

D.6.19 Changes in release 3.21.16

- Added ODBC 2.0 & 3.0 functions `POWER()`, `SPACE()`, `COT()`, `DEGREES()`, `RADIANS()`, `ROUND(2 arg)` and `TRUNCATE()`.
- **Warning: Incompatible change!** `LOCATE()` parameters were swapped according to ODBC standard. Fixed.
- Added function `TIME_TO_SEC()`.
- In some cases, default values were not used for `NOT NULL` fields.
- Timestamp wasn't always updated properly in `UPDATE SET ...` statements.
- Allow empty strings as default values for `BLOB` and `TEXT`, to be compatible with `mysqldump`.

D.6.20 Changes in release 3.21.15

- **Warning: Incompatible change!** `mysqlperl` is now from `Msql-Mysql-modules`. This means that `connect()` now takes `host`, `database`, `user`, `password` arguments! The old version took `host`, `database`, `password`, `user`.
- Allow `DATE '1997-01-01'`, `TIME '12:10:10'` and `TIMESTAMP '1997-01-01 12:10:10'` formats required by SQL-99. **Warning: Incompatible change!** This has the unfortunate side-effect that you no longer can have columns named `DATE`, `TIME` or `TIMESTAMP`. :(Old columns can still be accessed through `tablename.columnname!`)
- Changed Makefiles to hopefully work better with BSD systems. Also, `manual.dvi` is now included in the distribution to avoid having stupid `make` programs trying to rebuild it.
- `readline` library upgraded to version 2.1.
- A new sortorder `german-1`. That is a normal ISO-Latin1 with a german sort order.
- Perl DBI/DBD is now included in the distribution. DBI is now the recommended way to connect to MySQL from Perl.
- New portable benchmark suite with DBD, with test results from `mSQL 2.0.3`, `MySQL`, `PostgreSQL 6.2.1` and `Solid server 2.2`.
- `crash-me` is now included with the benchmarks; this is a Perl program designed to find as many limits as possible in a SQL server. Tested with `mSQL`, `PostgreSQL`, `Solid` and `MySQL`.
- Fixed bug in range-optimiser that crashed MySQL on some queries.
- Table and column name completion for `mysql` command-line tool, by Zeev Suraski and Andi Gutmans.
- Added new command `REPLACE` that works like `INSERT` but replaces conflicting records with the new record. `REPLACE INTO TABLE ... SELECT ...` works also.
- Added new commands `CREATE DATABASE db_name` and `DROP DATABASE db_name`.
- Added `RENAME` option to `ALTER TABLE`: `ALTER TABLE name RENAME TO new_name`.

- `make_binary_distribution` now includes `'libgcc.a'` in `'libmysqlclient.a'`. This should make linking work for people who don't have `gcc`.
- Changed `net_write()` to `my_net_write()` because of a name conflict with Sybase.
- New function `DAYOFWEEK()` compatible with ODBC.
- Stack checking and `bison` memory overrun checking to make MySQL safer with weird queries.

D.6.21 Changes in release 3.21.14b

- Fixed a couple of small `configure` problems on some platforms.

D.6.22 Changes in release 3.21.14a

- Ported to SCO Openserver 5.0.4 with FSU Pthreads.
- HP-UX 10.20 should work.
- Added new function `DATE_FORMAT()`.
- Added `NOT IN`.
- Added automatic removal of 'ODBC function conversions': `{fn now() }`
- Handle ODBC 2.50.3 option flags.
- Fixed comparison of `DATE` and `TIME` values with `NULL`.
- Changed language name from `germany` to `german` to be consistent with the other language names.
- Fixed sorting problem on functions returning a `FLOAT`. Previously, the values were converted to `INTs` before sorting.
- Fixed slow sorting when sorting on key field when using `key_column=constant`.
- Sorting on calculated `DOUBLE` values sorted on integer results instead.
- `mysql` no longer requires a database argument.
- Changed the place where `HAVING` should be. According to the SQL standards, it should be after `GROUP BY` but before `ORDER BY`. MySQL Version 3.20 incorrectly had it last.
- Added Sybase command `USE DATABASE` to start using another database.
- Added automatic adjusting of number of connections and table cache size if the maximum number of files that can be opened is less than needed. This should fix that `mysqld` doesn't crash even if you haven't done a `ulimit -n 256` before starting `mysqld`.
- Added lots of limit checks to make it safer when running with too little memory or when doing weird queries.

D.6.23 Changes in release 3.21.13

- Added retry of interrupted reads and clearing of `errno`. This makes Linux systems much safer!
- Fixed locking bug when using many aliases on the same table in the same `SELECT`.
- Fixed bug with `LIKE` on number key.
- New error message so you can check whether the connection was lost while the command was running or whether the connection was down from the start.

- Added `--table` option to `mysql` to print in table format. Moved time and row information after query result. Added automatic reconnect of lost connections.
- Added `!=` as a synonym for `<>`.
- Added function `VERSION()` to make easier logs.
- New multi-user test `'tests/fork_test.pl'` to put some strain on the thread library.

D.6.24 Changes in release 3.21.12

- Fixed `ftruncate()` call in MIT-pthreads. This made `isamchk` destroy the `‘.ISM’` files on (Free)BSD 2.x systems.
- Fixed broken `__P_` patch in MIT-pthreads.
- Many memory overrun checks. All string functions now return `NULL` if the returned string should be longer than `max_allowed_packet` bytes.
- Changed the name of the `INTERVAL` type to `ENUM`, because `INTERVAL` is used in SQL-99.
- In some cases, doing a `JOIN + GROUP + INTO OUTFILE`, the result wasn't grouped.
- `LIKE` with `'_'` as last character didn't work. Fixed.
- Added extended SQL-99 `TRIM()` function.
- Added `CURTIME()`.
- Added `ENCRYPT()` function by Zeev Suraski.
- Fixed better `FOREIGN KEY` syntax skipping. New reserved words: `MATCH`, `FULL`, `PARTIAL`.
- `mysqld` now allows IP number and hostname for the `--bind-address` option.
- Added `SET CHARACTER SET cp1251_koi8` to enable conversions of data to and from the `cp1251_koi8` character set.
- Lots of changes for Windows 95 port. In theory, this version should now be easily portable to Windows 95.
- Changed the `CREATE COLUMN` syntax of `NOT NULL` columns to be after the `DEFAULT` value, as specified in the SQL-99 standard. This will make `mysqldump` with `NOT NULL` and default values incompatible with MySQL Version 3.20.
- Added many function name aliases so the functions can be used with ODBC or SQL-92 syntax.
- Fixed syntax of `ALTER TABLE nome_tabela ALTER COLUMN nome_coluna SET DEFAULT NULL`.
- Added `CHAR` and `BIT` as synonyms for `CHAR(1)`.
- Fixed core dump when updating as a user who has only `SELECT` privilege.
- `INSERT ... SELECT ... GROUP BY` didn't work in some cases. An Invalid use of group function error occurred.
- When using `LIMIT`, `SELECT` now always uses keys instead of record scan. This will give better performance on `SELECT` and a `WHERE` that matches many rows.
- Added Russian error messages.

D.6.25 Changes in release 3.21.11

- Configure changes.
- MySQL now works with the new thread library on BSD/OS 3.0.
- Added new group functions `BIT_OR()` and `BIT_AND()`.
- Added compatibility functions `CHECK` and `REFERENCES`. `CHECK` is now a reserved word.
- Added `ALL` option to `GRANT` for better compatibility. (`GRANT` is still a dummy function.)
- Added partly-translated Dutch error messages.
- Fixed bug in `ORDER BY` and `GROUP BY` with `NULL` columns.
- Added function `LAST_INSERT_ID()` SQL function to retrieve last `AUTO_INCREMENT` value. This is intended for clients to ODBC that can't use the `mysql_insert_id()` API function, but can be used by any client.
- Added `--flush-logs` option to `mysqladmin`.
- Added command `STATUS` to `mysql`.
- Fixed problem with `ORDER BY/GROUP BY` because of bug in `gcc`.
- Fixed problem with `INSERT ... SELECT ... GROUP BY`.

D.6.26 Changes in release 3.21.10

- New `mysqlaccess`.
- `CREATE` now supports all ODBC types and the `mSQL TEXT` type. All ODBC 2.5 functions are also supported (added `REPEAT`). This provides better portability.
- Added text types `TINYTEXT`, `TEXT`, `MEDIUMTEXT` and `LONGTEXT`. These are actually `BLOB` types, but all searching is done in case-insensitive fashion.
- All old `BLOB` fields are now `TEXT` fields. This only changes that all searching on strings is done in case-sensitive fashion. You must do an `ALTER TABLE` and change the data type to `BLOB` if you want to have tests done in case-sensitive fashion.
- Fixed some `configure` issues.
- Made the locking code a bit safer. Fixed very unlikely deadlock situation.
- Fixed a couple of bugs in the range optimiser. Now the new range benchmark `test-select` works.

D.6.27 Changes in release 3.21.9

- Added `--enable-unix-socket=pathname` option to `configure`.
- Fixed a couple of portability problems with include files.
- Fixed bug in range calculation that could return empty set when searching on multiple key with only one entry (very rare).
- Most things ported to FSU Pthreads, which should allow MySQL to run on Caldera (SCO). See [\(undefined\) \[Caldera\], page \(undefined\)](#).

D.6.28 Changes in release 3.21.8

- Works now in Solaris 2.6.

- Added handling of calculation of SUM() functions. For example, you can now use SUM(column)/COUNT(column).
- Added handling of trigonometric functions: PI(), ACOS(), ASIN(), ATAN(), COS(), SIN() and TAN().
- New languages: Norwegian, Norwegian-ny and Portuguese.
- Fixed parameter bug in net_print() in 'procedure.cc'.
- Fixed a couple of memory leaks.
- Now allow also the old SELECT ... INTO OUTFILE syntax.
- Fixed bug with GROUP BY and SELECT on key with many values.
- mysql_fetch_lengths() sometimes returned incorrect lengths when you used mysql_use_result(). This affected at least some cases of mysqldump --quick.
- Fixed bug in optimisation of WHERE const op field.
- Fixed problem when sorting on NULL fields.
- Fixed a couple of 64-bit (Alpha) problems.
- Added --pid-file=# option to mysqld.
- Added date formatting to FROM_UNIXTIME(), originally by Zeev Suraski.
- Fixed bug in BETWEEN in range optimiser (did only test = of the first argument).
- Added machine-dependent files for MIT-threads i386-SCO. There is probably more to do to get this to work on SCO 3.5.

D.6.29 Changes in release 3.21.7

- Changed 'Makefile.am' to take advantage of Automake 1.2.
- Added the beginnings of a benchmark suite.
- Added more secure password handling.
- Added new client function mysql_errno(), to get the error number of the error message. This makes error checking in the client much easier. This makes the new server incompatible with the 3.20.x server when running without --old-protocol. The client code is backward-compatible. More information can be found in the 'README' file!
- Fixed some problems when using very long, illegal names.

D.6.30 Changes in release 3.21.6

- Fixed more portability issues (incorrect sigwait and sigset defines).
- configure should now be able to detect the last argument to accept().

D.6.31 Changes in release 3.21.5

- Should now work with FreeBSD 3.0 if used with 'FreeBSD-3.0-libc_r-1.0.diff', which can be found at <http://www.mysql.com/downloads/patches.html>.
- Added new -O tmp_table_size=# option to mysqld.
- New function FROM_UNIXTIME(timestamp) which returns a date string in 'YYYY-MM-DD HH:MM:DD' format.
- New function SEC_TO_TIME(seconds) which returns a string in 'HH:MM:SS' format.
- New function SUBSTRING_INDEX(), originally by Zeev Suraski.

D.6.32 Changes in release 3.21.4

- Should now configure and compile on OSF/1 4.0 with the DEC compiler.
- Configuration and compilation on BSD/OS 3.0 works, but due to some bugs in BSD/OS 3.0, `mysqld` doesn't work on it yet.
- Configuration and compilation on FreeBSD 3.0 works, but I couldn't get `pthread_create` to work.

D.6.33 Changes in release 3.21.3

- Added reverse check lookup of hostnames to get better security.
- Fixed some possible buffer overflows if filenames that are too long are used.
- `mysqld` doesn't accept hostnames that start with digits followed by a '.', because the hostname may look like an IP number.
- Added `--skip-networking` option to `mysqld`, to allow only socket connections. (This will not work with MIT-pthreads!)
- Added check of too long table names for alias.
- Added check if database name is okay.
- Added check if too long table names.
- Removed incorrect `free()` that killed the server on `CREATE DATABASE` or `DROP DATABASE`.
- Changed some `mysqld -O` options to better names.
- Added `-O join_cache_size=#` option to `mysqld`.
- Added `-O max_join_size=#` option to `mysqld`, to be able to set a limit how big queries (in this case big = slow) one should be able to handle without specifying `SET SQL_BIG_SELECTS=1`. A `# =` is about 10 examined records. The default is "unlimited".
- When comparing a `TIME`, `DATE`, `DATETIME` or `TIMESTAMP` column to a constant, the constant is converted to a time value before performing the comparison. This will make it easier to get ODBC (particularly Access97) to work with the above types. It should also make dates easier to use and the comparisons should be quicker than before.
- Applied patch from Jochen Wiedmann that allows `query()` in `mysqlperl` to take a query with `\0` in it.
- Storing a timestamp with a 2-digit year (YYMMDD) didn't work.
- Fix that timestamp wasn't automatically updated if set in an `UPDATE` clause.
- Now the automatic timestamp field is the `FIRST` timestamp field.
- `SELECT * INTO OUTFILE`, which didn't correctly if the outfile already existed.
- `mysql` now shows the thread ID when starting or doing a reconnect.
- Changed the default sort buffer size from 2M to 1M.

D.6.34 Changes in release 3.21.2

- The range optimiser is coded, but only 85% tested. It can be enabled with `--new`, but it crashes core a lot yet...

- More portable. Should compile on AIX and alpha-digital. At least the `isam` library should be relatively 64-bit clean.
- New `isamchk` which can detect and fix more problems.
- New options for `isamlog`.
- Using new version of Automake.
- Many small portability changes (from the AIX and alpha-digital port) Better checking of `pthread(s)` library.
- czech error messages by snajdr@pvt.net.
- Decreased size of some buffers to get fewer problems on systems with little memory. Also added more checks to handle “out of memory” problems.
- `mysqladmin`: you can now do `mysqladmin kill 5,6,7,8` to kill multiple threads.
- When the maximum connection limit is reached, one extra connection by a user with the `process.acl` privilege is granted.
- Added `-O backlog=#` option to `mysqld`.
- Increased maximum packet size from 512K to 1024K for client.
- Almost all of the function code is now tested in the internal test suite.
- `ALTER TABLE` now returns warnings from field conversions.
- Port changed to 3306 (got it reserved from ISI).
- Added a fix for Visual FoxBase so that any schema name from a table specification is automatically removed.
- New function `ASCII()`.
- Removed function `BETWEEN(a,b,c)`. Use the standard SQL syntax instead: `expr BETWEEN expr AND expr`.
- MySQL no longer has to use an extra temporary table when sorting on functions or `SUM()` functions.
- Fixed bug that you couldn't use `nome_tabela.field_name` in `UPDATE`.
- Fixed `SELECT DISTINCT` when using 'hidden group'. For example:


```
mysql> SELECT DISTINCT MOD(some_field,10) FROM test
->          GROUP BY some_field;
```

Note: `some_field` is normally in the `SELECT` part. Standard SQL should require it.

D.6.35 Changes in release 3.21.0

- New reserved words used: `INTERVAL`, `EXPLAIN`, `READ`, `WRITE`, `BINARY`.
- Added ODBC function `CHAR(num, ...)`.
- New operator `IN`. This uses a binary search to find a match.
- New command `LOCK TABLES nome_tabela [AS alias] {READ|WRITE} ...`
- Added `--log-update` option to `mysqld`, to get a log suitable for incremental updates.
- New command `EXPLAIN SELECT ...` to get information about how the optimiser will do the join.

- For easier client code, the client should no longer use `FIELD_TYPE_TINY_BLOB`, `FIELD_TYPE_MEDIUM_BLOB`, `FIELD_TYPE_LONG_BLOB` or `FIELD_TYPE_VAR_STRING` (as previously returned by `mysql_list_fields`). You should instead only use `FIELD_TYPE_BLOB` or `FIELD_TYPE_STRING`. If you want exact types, you should use the command `SHOW FIELDS`.
- Added varbinary syntax: `0x#####` which can be used as a string (default) or a number.
- `FIELD_TYPE_CHAR` is renamed to `FIELD_TYPE_TINY`.
- Changed all fields to C++ classes.
- Removed FORM struct.
- Fields with `DEFAULT` values no longer need to be `NOT NULL`.
- New field types:

<code>ENUM</code>	A string which can take only a couple of defined values. The value is stored as a 1-3 byte number that is mapped automatically to a string. This is sorted according to string positions!
<code>SET</code>	A string which may have one or many string values separated with <code>'</code> , <code>'</code> . The string is stored as a 1-, 2-, 3-, 4- or 8-byte number where each bit stands for a specific set member. This is sorted according to the unsigned value of the stored packed number.
- Now all function calculation is done with `double` or `long long`. This will provide the full 64-bit range with bit functions and fix some conversions that previously could result in precision losses. One should avoid using `unsigned long long` columns with full 64-bit range (numbers bigger than 9223372036854775807) because calculations are done with `signed long long`.
- `ORDER BY` will now put `NULL` field values first. `GROUP BY` will also work with `NULL` values.
- Full `WHERE` with expressions.
- New range optimiser that can resolve ranges when some keypart prefix is constant. Example:

```
mysql> SELECT * FROM nome_tabela
->         WHERE key_part_1="customer"
->         AND key_part_2>=10 AND key_part_2<=10;
```

D.7 Changes in release 3.20.x

Version 3.20 is quite old now, and should be avoided if possible. This information is kept here for historical purposes only.

Changes from 3.20.18 to 3.20.32b are not documented here because the 3.21 release branched here. And the relevant changes are also documented as changes to the 3.21 version.

D.7.1 Changes in release 3.20.18

- Added `-p#` (remove `#` directories from path) to `isamlog`. All files are written with a relative path from the database directory. Now `mysqld` shouldn't crash on shutdown when using the `--log-isam` option.
- New `mysqlperl` version. It is now compatible with `msqlperl-0.63`.

- New DBD module available.
- Added group function `STD()` (standard deviation).
- The `mysqld` server is now compiled by default without debugging information. This will make the daemon smaller and faster.
- Now one usually only has to specify the `--basedir` option to `mysqld`. All other paths are relative in a normal installation.
- BLOB columns sometimes contained garbage when used with a `SELECT` on more than one table and `ORDER BY`.
- Fixed that calculations that are not in `GROUP BY` work as expected (SQL-99 extension). Example:

```
mysql> SELECT id,id+1 FROM table GROUP BY id;
```
- The test of using `MYSQL_PWD` was reversed. Now `MYSQL_PWD` is enabled as default in the default release.
- Fixed conversion bug which caused `mysqld` to core dump with Arithmetic error on SPARC-386.
- Added `--unbuffered` option to `mysql`, for new `mysqlaccess`.
- When using overlapping (unnecessary) keys and join over many tables, the optimiser could get confused and return 0 records.

D.7.2 Changes in release 3.20.17

- You can now use BLOB columns and the functions `IS NULL` and `IS NOT NULL` in the `WHERE` clause.
- All communication packets and row buffers are now allocated dynamically on demand. The default value of `max_allowed_packet` is now 64K for the server and 512K for the client. This is mainly used to catch incorrect packets that could trash all memory. The server limit may be changed when it is started.
- Changed stack usage to use less memory.
- Changed `safe_mysqld` to check for running daemon.
- The `ELT()` function is renamed to `FIELD()`. The new `ELT()` function returns a value based on an index: `FIELD()` is the inverse of `ELT()` Example: `ELT(2,"A","B","C")` returns "B". `FIELD("B","A","B","C")` returns 2.
- `COUNT(field)`, where `field` could have a `NULL` value, now works.
- A couple of bugs fixed in `SELECT ... GROUP BY`.
- Fixed memory overrun bug in `WHERE` with many unoptimisable brace levels.
- Fixed some small bugs in the grant code.
- If hostname isn't found by `get_hostname`, only the IP is checked. Previously, you got `Access denied`.
- Inserts of timestamps with values didn't always work.
- `INSERT INTO ... SELECT ... WHERE` could give the error `Duplicated field`.
- Added some tests to `safe_mysqld` to make it "safer".
- `LIKE` was case-sensitive in some places and case-insensitive in others. Now `LIKE` is always case-insensitive.

- ‘mysql.cc’: Allow ‘#’ anywhere on the line.
- New command SET SQL_SELECT_LIMIT=#. See the FAQ for more details.
- New version of the mysqlaccess script.
- Change FROM_DAYS() and WEEKDAY() to also take a full TIMESTAMP or DATETIME as argument. Before they only took a number of type YYYYMMDD or YMMDD.
- Added new function UNIX_TIMESTAMP(timestamp_column).

D.7.3 Changes in release 3.20.16

- More changes in MIT-pthreads to get them safer. Fixed also some link bugs at least in SunOS.
- Changed mysqld to work around a bug in MIT-pthreads. This makes multiple small SELECT operations 20 times faster. Now lock_test.pl should work.
- Added mysql_FetchHash(handle) to mysqlperl.
- The mysqlbug script is now distributed built to allow for reporting bugs that appear during the build with it.
- Changed ‘libmysql.c’ to prefer getpwuid() instead of cuserid().
- Fixed bug in SELECT optimiser when using many tables with the same column used as key to different tables.
- Added new latin2 and Russian KOI8 character tables.
- Added support for a dummy GRANT command to satisfy Powerbuilder.

D.7.4 Changes in release 3.20.15

- Fixed fatal bug packets out of order when using MIT-pthreads.
- Removed possible loop when a thread waits for command from client and fcntl() fails. Thanks to Mike Bretz for finding this bug.
- Changed alarm loop in ‘mysqld.cc’ because shutdown didn’t always succeed in Linux.
- Removed use of termbits from ‘mysql.cc’. This conflicted with glibc 2.0.
- Fixed some syntax errors for at least BSD and Linux.
- Fixed bug when doing a SELECT as superuser without a database.
- Fixed bug when doing SELECT with group calculation to outfile.

D.7.5 Changes in release 3.20.14

- If one gives -p or --password option to mysql without an argument, the user is solicited for the password from the tty.
- Added default password from MYSQL_PWD (by Elmar Haneke).
- Added command kill to mysqladmin to kill a specific MySQL thread.
- Sometimes when doing a reconnect on a down connection this succeeded first on second try.
- Fixed adding an AUTO_INCREMENT key with ALTER_TABLE.
- AVG() gave too small value on some SELECTs with GROUP BY and ORDER BY.

- Added new DATETIME type (by Giovanni Maruzzelli maruzz@matrice.it).
- Fixed that defining DONT_USE_DEFAULT_FIELDS works.
- Changed to use a thread to handle alarms instead of signals on Solaris to avoid race conditions.
- Fixed default length of signed numbers. (George Harvey georgeh@pinacl.co.uk.)
- Allow anything for CREATE INDEX.
- Add prezeros when packing numbers to DATE, TIME and TIMESTAMP.
- Fixed a bug in OR of multiple tables (gave empty set).
- Added many patches to MIT-pthreads. This fixes at least one lookup bug.

D.7.6 Changes in release 3.20.13

- Added standard SQL-92 DATE and TIME types.
- Fixed bug in SELECT with AND-OR levels.
- Added support for Slovenian characters. The ‘Contrib’ directory contains source and instructions for adding other character sets.
- Fixed bug with LIMIT and ORDER BY.
- Allow ORDER BY and GROUP BY on items that aren’t in the SELECT list. (Thanks to Wim Bonis bonis@kiss.de, for pointing this out.)
- Allow setting of timestamp values in INSERT.
- Fixed bug with SELECT ... WHERE ... = NULL.
- Added changes for glibc 2.0. To get glibc to work, you should add the ‘glibc-2.0-sigwait-patch’ before compiling glibc.
- Fixed bug in ALTER TABLE when changing a NOT NULL field to allow NULL values.
- Added some SQL-92 synonyms as field types to CREATE TABLE. CREATE TABLE now allows FLOAT(4) and FLOAT(8) to mean FLOAT and DOUBLE.
- New utility program mysqlaccess by Yves.Carlier@rug.ac.be. This program shows the access rights for a specific user and the grant rows that determine this grant.
- Added WHERE const op field (by bonis@kiss.de).

D.7.7 Changes in release 3.20.11

- When using SELECT ... INTO outfile, all temporary tables are ISAM instead of HEAP to allow big dumps.
- Changed date functions to be string functions. This fixed some “funny” side effects when sorting on dates.
- Extended ALTER TABLE for SQL-92 compliance.
- Some minor compatibility changes.
- Added --port and --socket options to all utility programs and mysqld.
- Fixed MIT-pthreads readdir_r(). Now mysqladmin create database and mysqladmin drop database should work.
- Changed MIT-pthreads to use our tempnam(). This should fix the “sort aborted” bug.
- Added sync of records count in sql_update. This fixed slow updates on first connection. (Thanks to Vaclav Bittner for the test.)

D.7.8 Changes in release 3.20.10

- New insert type: `INSERT INTO ... SELECT ...`
- `MEDIUMBLOB` fixed.
- Fixed bug in `ALTER TABLE` and `BLOBs`.
- `SELECT ... INTO OUTFILE` now creates the file in the current database directory.
- `DROP TABLE` now can take a list of tables.
- Oracle synonym `DESCRIBE (DESC)`.
- Changes to `make_binary_distribution`.
- Added some comments to installation instructions about `configure`'s C++ link test.
- Added `--without-perl` option to `configure`.
- Lots of small portability changes.

D.7.9 Changes in release 3.20.9

- `ALTER TABLE` didn't copy null bit. As a result, fields that were allowed to have `NULL` values were always `NULL`.
- `CREATE` didn't take numbers as `DEFAULT`.
- Some compatibility changes for SunOS.
- Removed `'config.cache'` from old distribution.

D.7.10 Changes in release 3.20.8

- Fixed bug with `ALTER TABLE` and multi-part keys.

D.7.11 Changes in release 3.20.7

- New commands: `ALTER TABLE`, `SELECT ... INTO OUTFILE` and `LOAD DATA INFILE`.
- New function: `NOW()`.
- Added new field `File_priv` to `mysql/user` table.
- New script `add_file_priv` which adds the new field `File_priv` to the `user` table. This script must be executed if you want to use the new `SELECT ... INTO` and `LOAD DATA INFILE ...` commands with a version of MySQL earlier than 3.20.7.
- Fixed bug in locking code, which made `lock_test.pl` test fail.
- New files `'NEW'` and `'BUGS'`.
- Changed `'select_test.c'` and `'insert_test.c'` to include `'config.h'`.
- Added `status` command to `mysqladmin` for short logging.
- Increased maximum number of keys to 16 and maximum number of key parts to 15.
- Use of sub keys. A key may now be a prefix of a string field.
- Added `-k` option to `mysqlshow`, to get key information for a table.
- Added long options to `mysqldump`.

D.7.12 Changes in release 3.20.6

- Portable to more systems because of MIT-pthreads, which will be used automatically if `configure` cannot find a `-lpthreads` library.
- Added GNU-style long options to almost all programs. Test with `program --help`.
- Some shared library support for Linux.
- The FAQ is now in `.texi` format and is available in `.html`, `.txt` and `.ps` formats.
- Added new SQL function `RAND([init])`.
- Changed `sql_lex` to handle `\0` unquoted, but the client can't send the query through the C API, because it takes a `str` pointer. You must use `mysql_real_query()` to send the query.
- Added API function `mysql_get_client_info()`.
- `mysqld` now uses the `N_MAX_KEY_LENGTH` from `'nisam.h'` as the maximum allowable key length.
- The following now works:


```
mysql> SELECT filter_nr,filter_nr FROM filter ORDER BY filter_nr;
```

 Previously, this resulted in the error: `Column: 'filter_nr' in order clause is ambiguous`.
- `mysql` now outputs `'\0'`, `'\t'`, `'\n'` and `'\'` when encountering ASCII 0, tab, new-line or `'\'` while writing tab-separated output. This is to allow printing of binary data in a portable format. To get the old behaviour, use `-r` (or `--raw`).
- Added german error messages (60 of 80 error messages translated).
- Added new API function `mysql_fetch_lengths(MYSQL_RES *)`, which returns an array of column lengths (of type `uint`).
- Fixed bug with `IS NULL` in `WHERE` clause.
- Changed the optimiser a little to get better results when searching on a key part.
- Added `SELECT` option `STRAIGHT_JOIN` to tell the optimiser that it should join tables in the given order.
- Added support for comments starting with `'--'` in `'mysql.cc'` (Postgres syntax).
- You can have `SELECT` expressions and table columns in a `SELECT` which are not used in the group part. This makes it efficient to implement lookups. The column that is used should be a constant for each group because the value is calculated only once for the first row that is found for a group.


```
mysql> SELECT id,lookup.text,SUM(*) FROM test,lookup
->          WHERE test.id=lookup.id GROUP BY id;
```
- Fixed bug in `SUM(function)` (could cause a core dump).
- Changed `AUTO_INCREMENT` placement in the SQL query:


```
INSERT INTO table (auto_field) VALUES (0);
```

 inserted 0, but it should insert an `AUTO_INCREMENT` value.
- `'mysqlshow.c'`: Added number of records in table. Had to change the client code a little to fix this.
- `mysql` now allows doubled `'` or `"` within strings for embedded `'` or `"`.
- New math functions: `EXP()`, `LOG()`, `SQRT()`, `ROUND()`, `CEILING()`.

D.7.13 Changes in release 3.20.3

- The `configure` source now compiles a thread-free client library `-lmysqlclient`. This is the only library that needs to be linked with client applications. When using the binary releases, you must link with `-lmysql -lmysys -ldbbug -lmystrings` as before.
- New `readline` library from `bash-2.0`.
- LOTS of small changes to `configure` and `makefiles` (and related source).
- It should now be possible to compile in another directory using `VPATH`. Tested with GNU Make 3.75.
- `safe_mysqld` and `mysql.server` changed to be more compatible between the source and the binary releases.
- `LIMIT` now takes one or two numeric arguments. If one argument is given, it indicates the maximum number of rows in a result. If two arguments are given, the first argument indicates the offset of the first row to return, the second is the maximum number of rows. With this it's easy to do a poor man's next page/previous page WWW application.
- Changed name of SQL function `FIELDS()` to `ELT()`. Changed SQL function `INTERVALL()` to `INTERVAL()`.
- Made `SHOW COLUMNS` a synonym for `SHOW FIELDS`. Added compatibility syntax `FRIEND KEY` to `CREATE TABLE`. In MySQL, this creates a non-unique key on the given columns.
- Added `CREATE INDEX` and `DROP INDEX` as compatibility functions. In MySQL, `CREATE INDEX` only checks if the index exists and issues an error if it doesn't exist. `DROP INDEX` always succeeds.
- `'mysqladmin.c'`: added client version to version information.
- Fixed core dump bug in `sql_acl` (core on new connection).
- Removed `host`, `user` and `db` tables from database `test` in the distribution.
- `FIELD_TYPE_CHAR` can now be signed (-128 to 127) or unsigned (0 to 255) Previously, it was always unsigned.
- Bug fixes in `CONCAT()` and `WEEKDAY()`.
- Changed a lot of source to get `mysqld` to be compiled with SunPro compiler.
- SQL functions must now have a `'` (`'` immediately after the function name (no intervening space). For example, `'USER('` is regarded as beginning a function call, and `'USER` (`'` is regarded as an identifier `USER` followed by a `'` (`'`, not as a function call.

D.7.14 Changes in release 3.20.0

- The source distribution is done with `configure` and Automake. It will make porting much easier. The `readline` library is included in the distribution.
- Separate client compilation: the client code should be very easy to compile on systems which don't have threads.
- The old Perl interface code is automatically compiled and installed. Automatic compiling of DBD will follow when the new DBD code is ported.
- Dynamic language support: `mysqld` can now be started with Swedish or English (default) error messages.
- New functions: `INSERT()`, `RTRIM()`, `LTRIM()` and `FORMAT()`.

- `mysqldump` now works correctly for all field types (even `AUTO_INCREMENT`). The format for `SHOW FIELDS FROM nome_tabela` is changed so the `Type` column contains information suitable for `CREATE TABLE`. In previous releases, some `CREATE TABLE` information had to be patched when re-creating tables.
- Some parser bugs from 3.19.5 (`BLOB` and `TIMESTAMP`) are corrected. `TIMESTAMP` now returns different date information depending on its create length.
- Changed parser to allow a database, table or field name to start with a number or `'_'`.
- All old C code from Unireg changed to C++ and cleaned up. This makes the daemon a little smaller and easier to understand.
- A lot of small bug fixes done.
- New `'INSTALL'` files (not final version) and some information regarding porting.

D.8 Changes in release 3.19.x

Version 3.19 is quite old now, and should be avoided if possible. This information is kept here for historical purposes only.

D.8.1 Changes in release 3.19.5

- Some new functions, some more optimisation on joins.
- Should now compile clean on Linux (2.0.x).
- Added functions `DATABASE()`, `USER()`, `POW()`, `LOG10()` (needed for ODBC).
- In a `WHERE` with an `ORDER BY` on fields from only one table, the table is now preferred as first table in a multi-join.
- `HAVING` and `IS NULL` or `IS NOT NULL` now works.
- A group on one column and a sort on a group function (`SUM()`, `AVG()`...) didn't work together. Fixed.
- `mysqldump`: Didn't send password to server.

D.8.2 Changes in release 3.19.4

- Fixed horrible locking bug when inserting in one thread and reading in another thread.
- Fixed one-off decimal bug. 1.00 was output as 1.0.
- Added attribute `'Locked'` to process list as info if a query is locked by another query.
- Fixed full magic timestamp. Timestamp length may now be 14, 12, 10, 8, 6, 4 or 2 bytes.
- Sort on some numeric functions could sort incorrectly on last number.
- `IF(arg,syntax_error,syntax_error)` crashed.
- Added functions `CEILING()`, `ROUND()`, `EXP()`, `LOG()` and `SQRT()`.
- Enhanced `BETWEEN` to handle strings.

D.8.3 Changes in release 3.19.3

- Fixed `SELECT` with grouping on `BLOB` columns not to return incorrect `BLOB` info. Grouping, sorting and distinct on `BLOB` columns will not yet work as expected (probably it will group/sort by the first 7 characters in the `BLOB`). Grouping on formulas with a fixed string size (use `MID()` on a `BLOB`) should work.
- When doing a full join (no direct keys) on multiple tables with `BLOB` fields, the `BLOB` was garbage on output.
- Fixed `DISTINCT` with calculated columns.

Appendix E Porting to Other Systems

This appendix will help you port MySQL to other operating systems. Do check the list of currently supported operating systems first. See [\[Which OS\]](#), page [\[undefined\]](#). If you have created a new port of MySQL, please let us know so that we can list it here and on our web site (<http://www.mysql.com/>), recommending it to other users.

Note: If you create a new port of MySQL, you are free to copy and distribute it under the GPL license, but it does not make you a copyright holder of MySQL.

A working Posix thread library is needed for the server. On Solaris 2.5 we use Sun PThreads (the native thread support in 2.4 and earlier versions is not good enough), on Linux we use LinuxThreads by Xavier Leroy, Xavier.Leroy@inria.fr.

The hard part of porting to a new Unix variant without good native thread support is probably to port MIT-pthreads. See 'mit-pthreads/README' and Programming POSIX Threads (<http://www.humanfactor.com/pthreads/>).

Up to MySQL 4.0.2, the MySQL distribution included a patched version of Chris Provenzano's Pthreads from MIT (see the MIT Pthreads web page at <http://www.mit.edu/afs/sipb/project/pthreads/> and a programming introduction at http://www.mit.edu:8001/people/proven/IAP_2000/). These can be used for some operating systems that do not have POSIX threads. See [pthreads-snt](#) [[MIT-pthreads](#)], page [pthreads-pg](#).

It is also possible to use another user level thread package named FSU Pthreads (see <http://moss.csc.ncsu.edu/~mueller/pthreads/>). This implementation is being used for the SCO port.

See the 'thr_lock.c' and 'thr_alarm.c' programs in the 'mysys' directory for some tests/examples of these problems.

Both the server and the client need a working C++ compiler. We use gcc on many platforms. Other compilers that are known to work are SPARCworks, Sun Forte, Irix cc, HP-UX aCC, IBM AIX x1C_r), Intel ecc and Compaq cxx).

To compile only the client use `./configure --without-server`.

There is currently no support for only compiling the server, nor is it likely to be added unless someone has a good reason for it.

If you want/need to change any 'Makefile' or the configure script you will also need GNU Automake and Autoconf. See [\[Installing source tree\]](#), page [\[undefined\]](#).

All steps needed to remake everything from the most basic files.

```

/bin/rm */.deps/*.P
/bin/rm -f config.cache
aclocal
autoheader
aclocal
automake
autoconf
./configure --with-debug=full --prefix='your installation directory'

# The makefiles generated above need GNU make 3.75 or newer.
# (called gmake below)
```

```
gmake clean all install init-db
```

If you run into problems with a new port, you may have to do some debugging of MySQL! See [\[Debugging server\]](#), page [\[undefined\]](#).

Note: before you start debugging `mysqld`, first get the test programs `mysys/thr_alarm` and `mysys/thr_lock` to work. This will ensure that your thread installation has even a remote chance to work!

E.1 Debugging a MySQL server

If you are using some functionality that is very new in MySQL, you can try to run `mysqld` with the `--skip-new` (which will disable all new, potentially unsafe functionality) or with `--safe-mode` which disables a lot of optimisation that may cause problems. See [\[Crashing\]](#), page [\[undefined\]](#).

If `mysqld` doesn't want to start, you should check that you don't have any 'my.cnf' files that interfere with your setup! You can check your 'my.cnf' arguments with `mysqld --print-defaults` and avoid using them by starting with `mysqld --no-defaults ...`

If `mysqld` starts to eat up CPU or memory or if it "hangs", you can use `mysqladmin processlist status` to find out if someone is executing a query that takes a long time. It may be a good idea to run `mysqladmin -i10 processlist status` in some window if you are experiencing performance problems or problems when new clients can't connect.

The command `mysqladmin debug` will dump some information about locks in use, used memory and query usage to the mysql log file. This may help solve some problems. This command also provides some useful information even if you haven't compiled MySQL for debugging!

If the problem is that some tables are getting slower and slower you should try to optimise the table with `OPTIMIZE TABLE` or `myisamchk`. See [\[MySQL Database Administration\]](#), page [\[undefined\]](#). You should also check the slow queries with `EXPLAIN`.

You should also read the OS-specific section in this manual for problems that may be unique to your environment. See [\[Operating System Specific Notes\]](#), page [\[undefined\]](#).

E.1.1 Compiling MYSQL for Debugging

If you have some very specific problem, you can always try to debug MySQL. To do this you must configure MySQL with the `--with-debug` or the `--with-debug=full` option. You can check whether MySQL was compiled with debugging by doing: `mysqld --help`. If the `--debug` flag is listed with the options then you have debugging enabled. `mysqladmin ver` also lists the `mysqld` version as `mysql ... --debug` in this case.

If you are using gcc or egcs, the recommended configure line is:

```
CC=gcc CFLAGS="-O2" CXX=gcc CXXFLAGS="-O2 -felide-constructors \
-fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql \
--with-debug --with-extra-charsets=complex
```

This will avoid problems with the `libstdc++` library and with C++ exceptions (many compilers have problems with C++ exceptions in threaded code) and compile a MySQL version with support for all character sets.

If you suspect a memory overrun error, you can configure MySQL with `--with-debug=full`, which will install a memory allocation (SAFEMALLOC) checker. Running with SAFEMALLOC is however quite slow, so if you get performance problems you should start `mysqld` with the `--skip-safemalloc` option. This will disable the memory overrun checks for each call to `malloc` and `free`.

If `mysqld` stops crashing when you compile it with `--with-debug`, you have probably found a compiler bug or a timing bug within MySQL. In this case you can try to add `-g` to the `CFLAGS` and `CXXFLAGS` variables above and not use `--with-debug`. If `mysqld` now dies, you can at least attach to it with `gdb` or use `gdb` on the core file to find out what happened.

When you configure MySQL for debugging you automatically enable a lot of extra safety check functions that monitor the health of `mysqld`. If they find something “unexpected,” an entry will be written to `stderr`, which `safe_mysqld` directs to the error log! This also means that if you are having some unexpected problems with MySQL and are using a source distribution, the first thing you should do is to configure MySQL for debugging! (The second thing, of course, is to send mail to mysql@lists.mysql.com and ask for help. Please use the `mysqlbug` script for all bug reports or questions regarding the MySQL version you are using!

In the Windows MySQL distribution, `mysqld.exe` is by default compiled with support for trace files.

E.1.2 Creating Trace Files

If the `mysqld` server doesn't start or if you can cause the `mysqld` server to crash quickly, you can try to create a trace file to find the problem.

To do this you have to have a `mysqld` that is compiled for debugging. You can check this by executing `mysqld -V`. If the version number ends with `-debug`, it's compiled with support for trace files.

Start the `mysqld` server with a trace log in `'/tmp/mysqld.trace'` (or `'C:\mysqld.trace'` on Windows):

```
mysqld --debug
```

On Windows you should also use the `--standalone` flag to not start `mysqld` as a service:

In a DOS window do:

```
mysqld --debug --standalone
```

After this you can use the `mysql.exe` command-line tool in a second DOS window to reproduce the problem. You can take down the above `mysqld` server with `mysqladmin shutdown`.

Note that the trace file will get **very big**! If you want to have a smaller trace file, you can use something like:

```
mysqld --debug=d,info,error,query,general,where:0,/tmp/mysqld.trace
```

which only prints information with the most interesting tags in `'/tmp/mysqld.trace'`.

If you make a bug report about this, please only send the lines from the trace file to the appropriate mailing list where something seems to go wrong! If you can't locate the wrong place, you can ftp the trace file, together with a full bug report, to

<ftp://support.mysql.com/pub/mysql/secret/> so that a MySQL developer can take a look at this.

The trace file is made with the **DBUG** package by Fred Fish. See [\[The DBUG package\]](#), page [\[undefined\]](#).

E.1.3 Debugging `mysqld` under `gdb`

On most systems you can also start `mysqld` from `gdb` to get more information if `mysqld` crashes.

With some older `gdb` versions on Linux you must use `run --one-thread` if you want to be able to debug `mysqld` threads. In this case you can only have one thread active at a time. We recommend you to upgrade to `gdb` 5.1 ASAP as thread debugging works much better with this version!

When running `mysqld` under `gdb`, you should disable the stack trace with `--skip-stack-trace` to be able to catch segfaults within `gdb`.

It's very hard to debug MySQL under `gdb` if you do a lot of new connections the whole time as `gdb` doesn't free the memory for old threads. You can avoid this problem by starting `mysqld` with `-O thread_cache_size= 'max_connections +1'`. In most cases just using `-O thread_cache_size=5` will help a lot!

If you want to get a core dump on Linux if `mysqld` dies with a SIGSEGV signal, you can start `mysqld` with the `--core-file` option. This core file can be used to make a backtrace that may help you find out why `mysqld` died:

```
shell> gdb mysqld core
gdb> backtrace full
gdb> exit
```

See [\[undefined\]](#) [\[Crashing\]](#), page [\[undefined\]](#).

If you are using `gdb` 4.17.x or above on Linux, you should install a `‘.gdb’` file, with the following information, in your current directory:

```
set print sevenbit off
handle SIGUSR1 nostop noprint
handle SIGUSR2 nostop noprint
handle SIGWAITING nostop noprint
handle SIGLWP nostop noprint
handle SIGPIPE nostop
handle SIGALRM nostop
handle SIGHUP nostop
handle SIGTERM nostop noprint
```

If you have problems debugging threads with `gdb`, you should download `gdb` 5.x and try this instead. The new `gdb` version has very improved thread handling!

Here is an example how to debug `mysqld`:

```
shell> gdb /usr/local/libexec/mysqld
gdb> run
...
backtrace full # Do this when mysqld crashes
```

Include the above output in a mail generated with `mysqlbug` and mail this to `mysql@lists.mysql.com`.

If `mysqld` hangs you can try to use some system tools like `strace` or `/usr/proc/bin/pstack` to examine where `mysqld` has hung.

```
strace /tmp/log libexec/mysqld
```

If you are using the Perl DBI interface, you can turn on debugging information by using the `trace` method or by setting the `DBI_TRACE` environment variable. See [\(undefined\)](#) [Perl DBI Class], page [\(undefined\)](#).

E.1.4 Using a Stack Trace

On some operating systems, the error log will contain a stack trace if `mysqld` dies unexpectedly. You can use this to find out where (and maybe why) `mysqld` died. See [\(undefined\)](#) [Error log], page [\(undefined\)](#). To get a stack trace, you must not compile `mysqld` with the `-fomit-frame-pointer` option to `gcc`. See [\(undefined\)](#) [Compiling for debugging], page [\(undefined\)](#).

If the error file contains something like the following:

```
mysqld got signal 11;
The manual section 'Debugging a MySQL server' tells you how to use a
stack trace and/or the core file to produce a readable backtrace that may
help in finding out why mysqld died
Attempting backtrace. You can use the following information to find out
where mysqld died. If you see no messages after this, something went
terribly wrong
stack range sanity check, ok, backtrace follows
0x40077552
0x81281a0
0x8128f47
0x8127be0
0x8127995
0x8104947
0x80ff28f
0x810131b
0x80ee4bc
0x80c3c91
0x80c6b43
0x80c1fd9
0x80c1686
```

you can find where `mysqld` died by doing the following:

1. Copy the above numbers to a file, for example `'mysqld.stack'`.
2. Make a symbol file for the `mysqld` server:

```
nm -n libexec/mysqld > /tmp/mysqld.sym
```

Note that most MySQL binary distributions (except for the "debug" packages, where this information is included inside of the binaries themselves) already ship with the above file, named `mysqld.sym.gz`. In this case you can simply unpack it by doing:

```
gunzip < bin/mysqld.sym.gz > /tmp/mysqld.sym
```

3. Execute `resolve_stack_dump -s /tmp/mysqld.sym -n mysqld.stack`.

This will print out where `mysqld` died. If this doesn't help you find out why `mysqld` died, you should make a bug report and include the output from the above command with the bug report.

Note however that in most cases it will not help us to just have a stack trace to find the reason for the problem. To be able to locate the bug or provide a workaround, we would in most cases need to know the query that killed `mysqld` and preferable a test case so that we can repeat the problem! See [\[Bug reports\]](#), page [\[undefined\]](#).

E.1.5 Using Log Files to Find Cause of Errors in `mysqld`

Note that before starting `mysqld` with `--log` you should check all your tables with `myisamchk`. See [\[MySQL Database Administration\]](#), page [\[undefined\]](#).

If `mysqld` dies or hangs, you should start `mysqld` with `--log`. When `mysqld` dies again, you can examine the end of the log file for the query that killed `mysqld`.

If you are using `--log` without a file name, the log is stored in the database directory as `'hostname'.log`. In most cases it's the last query in the log file that killed `mysqld`, but if possible you should verify this by restarting `mysqld` and executing the found query from the `mysql` command-line tools. If this works, you should also test all complicated queries that didn't complete.

You can also try the command `EXPLAIN` on all `SELECT` statements that takes a long time to ensure that `mysqld` is using indexes properly. See [\[EXPLAIN\]](#), page [\[undefined\]](#).

You can find the queries that take a long time to execute by starting `mysqld` with `--log-slow-queries`. See [\[Slow query log\]](#), page [\[undefined\]](#).

If you find the text `mysqld restarted` in the error log file (normally named `'hostname.err'`) you have probably found a query that causes `mysqld` to fail. If this happens you should check all your tables with `myisamchk` (see [\[MySQL Database Administration\]](#), page [\[undefined\]](#)), and test the queries in the MySQL log files to see if one doesn't work. If you find such a query, try first upgrading to the newest MySQL version. If this doesn't help and you can't find anything in the `mysql` mail archive, you should report the bug to mysql@lists.mysql.com. Links to mail archives are available online at <http://lists.mysql.com/>.

If you have started `mysqld` with `myisam-recover`, MySQL will automatically check and try to repair MyISAM tables if they are marked as 'not closed properly' or 'crashed'. If this happens, MySQL will write an entry in the `hostname.err` file `'Warning: Checking table ...'` which is followed by `Warning: Repairing table` if the table needs to be repaired. If you get a lot of these errors, without `mysqld` having died unexpectedly just before, then something is wrong and needs to be investigated further. See [line options-snt \[Command-line options\]](#), page [line options-pg](#).

It's of course not a good sign if `mysqld` did die unexpectedly, but in this case one shouldn't investigate the `Checking table...` messages but instead try to find out why `mysqld` died.

E.1.6 Making a Test Case If You Experience Table Corruption

If you get corrupted tables or if `mysqld` always fails after some update commands, you can test if this bug is reproducible by doing the following:

- Take down the MySQL daemon (with `mysqladmin shutdown`).
- Make a backup of the tables (to guard against the very unlikely case that the repair will do something bad).
- Check all tables with `myisamchk -s database/*.MYI`. Repair any wrong tables with `myisamchk -r database/table.MYI`.
- Make a second backup of the tables.
- Remove (or move away) any old log files from the MySQL data directory if you need more space.
- Start `mysqld` with `--log-bin`. See [\[Binary log\]](#), page [\[undefined\]](#). If you want to find a query that crashes `mysqld`, you should use `--log --log-bin`.
- When you have gotten a crashed table, stop the `mysqld` server.
- Restore the backup.
- Restart the `mysqld` server **without** `--log-bin`
- Re-execute the commands with `mysqlbinlog update-log-file | mysql`. The update log is saved in the MySQL database directory with the name `hostname-bin.#`.
- If the tables are corrupted again or you can get `mysqld` to die with the above command, you have found reproducible bug that should be easy to fix! FTP the tables and the binary log to <ftp://support.mysql.com/pub/mysql/secret/> and enter it into our bugs system at <http://bugs.mysql.com/>. If you are a support customer), you can also support@mysql.com to alert the MySQL team about the problem and have it fixed as soon as possible.

You can also use the script `mysql_find_rows` to just execute some of the update statements if you want to narrow down the problem.

E.2 Debugging a MySQL client

To be able to debug a MySQL client with the integrated debug package, you should configure MySQL with `--with-debug` or `--with-debug=full`. See [\[configure options\]](#), page [\[undefined\]](#).

Before running a client, you should set the `MYSQL_DEBUG` environment variable:

```
shell> MYSQL_DEBUG=d:t:0,/tmp/client.trace
shell> export MYSQL_DEBUG
```

This causes clients to generate a trace file in `'/tmp/client.trace'`.

If you have problems with your own client code, you should attempt to connect to the server and run your query using a client that is known to work. Do this by running `mysql` in debugging mode (assuming you have compiled MySQL with debugging on):

```
shell> mysql --debug=d:t:0,/tmp/client.trace
```

This will provide useful information in case you mail a bug report. See [\[Bug reports\]](#), page [\[undefined\]](#).

If your client crashes at some 'legal' looking code, you should check that your `'mysql.h'` include file matches your `mysql` library file. A very common mistake is to use an old `'mysql.h'` file from an old MySQL installation with new MySQL library.

E.3 The DEBUG Package

The MySQL server and most MySQL clients are compiled with the DEBUG package originally made by Fred Fish. When one has configured MySQL for debugging, this package makes it possible to get a trace file of what the program is debugging. See [\[Making trace files\]](#), page [\[undefined\]](#).

One uses the debug package by invoking the program with the `--debug="..."` or the `-#...` option.

Most MySQL programs has a default debug string that will be used if you don't specify an option to `--debug`. The default trace file is usually `/tmp/programname.trace` on Unix and `\programname.trace` on Windows.

The debug control string is a sequence of colon separated fields as follows:

```
<field_1>:<field_2>:...:<field_N>
```

Each field consists of a mandatory flag character followed by an optional "," and comma-separated list of modifiers:

```
flag[,modifier,modifier,...,modifier]
```

The currently recognised flag characters are:

Flag	Description
d	Enable output from DEBUG.<N> macros for the current state. May be followed by a list of keywords which selects output only for the DEBUG macros with that keyword. An empty list of keywords implies output for all macros.
D	Delay after each debugger output line. The argument is the number of tenths of seconds to delay, subject to machine capabilities. That is, <code>-#D,20</code> is delay two seconds.
f	Limit debugging and/or tracing, and profiling to the list of named functions. Note that a null list will disable all functions. The appropriate "d" or "t" flags must still be given, this flag only limits their actions if they are enabled.
F	Identify the source file name for each line of debug or trace output.
i	Identify the process with the pid or thread id for each line of debug or trace output.
g	Enable profiling. Create a file called 'dbugmon.out' containing information that can be used to profile the program. May be followed by a list of keywords that select profiling only for the functions in that list. A null list implies that all functions are considered.
L	Identify the source file line number for each line of debug or trace output.
n	Print the current function nesting depth for each line of debug or trace output.
N	Number each line of dbug output.
o	Redirect the debugger output stream to the specified file. The default output is stderr.
O	As o but the file is really flushed between each write. When needed the file is closed and reopened between each write.
p	Limit debugger actions to specified processes. A process must be identified with the DEBUG_PROCESS macro and match one in the list for debugger actions to occur.
P	Print the current process name for each line of debug or trace output.
r	When pushing a new state, do not inherit the previous state's function nesting level. Useful when the output is to start at the left margin.
S	Do function <code>_sanity(_file_,_line_)</code> at each debugged function until <code>_sanity()</code> returns something that differs from 0. (Mostly used with <code>safemalloc</code> to find memory leaks)

- t Enable function call/exit trace lines. May be followed by a list (containing only one modifier) giving a numeric maximum trace level, beyond which no output will occur for either debugging or tracing macros. The default is a compile time option.

Some examples of debug control strings which might appear on a shell command-line (the "#-" is typically used to introduce a control string to an application program) are:

```
-#d:t
-#d:f,main,subr1:F:L:t,20
-#d,input,output,files:n
-#d:t:i:0,\\mysqld.trace
```

In MySQL, common tags to print (with the d option) are: `enter`, `exit`, `error`, `warning`, `info` and `loop`.

E.4 Locking methods

Currently MySQL only supports table locking for ISAM/MyISAM and HEAP tables, page-level locking for BDB tables and row-level locking for InnoDB tables. See [\[Internal locking\]](#), page [\[undefined\]](#). With MyISAM tables one can freely mix INSERT and SELECT without locks, if the INSERTs are non-conflicting (i.e. whenever they append to the end of the table file rather than filling freespace from deleted rows/data).

Starting in version 3.23.33, you can analyse the table lock contention on your system by checking `Table_locks_waited` and `Table_locks_immediate` environment variables.

To decide if you want to use a table type with row-level locking, you will want to look at what the application does and what the select/update pattern of the data is.

Pros for row locking:

- Fewer lock conflicts when accessing different rows in many threads.
- Fewer changes for rollbacks.
- Makes it possible to lock a single row a long time.

Cons:

- Takes more memory than page level or table locks.
- Is slower than page level or table locks when used on a big part of the table, because one has to do many more locks.
- Is definitely much worse than other locks if you do often do GROUP BY on a large part of the data or if one has to often scan the whole table.
- With higher level locks one can also more easily support locks of different types to tune the application as the lock overhead is less notable as for row level locks.

Table locks are superior to page level / row level locks in the following cases:

- Mostly reads
- Read and updates on strict keys; this is where one updates or deletes a row that can be fetched with one key read:

```
UPDATE table_name SET column=value WHERE unique_key#
DELETE FROM table_name WHERE unique_key=#
```

- SELECT combined with INSERT (and very few UPDATES and DELETES).

- Many scans / **GROUP BY** on the whole table without any writers.

Other options than row / page level locking:

Versioning (like we use in MySQL for concurrent inserts) where you can have one writer at the same time as many readers. This means that the database/table supports different views for the data depending on when one started to access it. Other names for this are time travel, copy on write or copy on demand.

Copy on demand is in many case much better than page or row level locking; the worst case does, however, use much more memory than when using normal locks.

Instead of using row level locks one can use application level locks (like `get.lock/release.lock` in MySQL). This works of course only in well-behaved applications.

In many cases one can do an educated guess which locking type is best for the application, but generally it's very hard to say that a given lock type is better than another; everything depends on the application and different part of the application may require different lock types.

Here are some tips about locking in MySQL:

Most web applications do lots of selects, very few deletes, updates mainly on keys, and inserts in some specific tables. The base MySQL setup is very well tuned for this.

Concurrent users are not a problem if one doesn't mix updates with selects that need to examine many rows in the same table.

If one mixes inserts and deletes on the same table then `INSERT DELAYED` may be of great help.

One can also use `LOCK TABLES` to speed up things (many updates within a single lock is much faster than updates without locks). Splitting thing to different tables will also help.

If you get speed problems with the table locks in MySQL, you may be able to solve these by converting some of your tables to InnoDB or BDB tables. See [\[InnoDB\]](#), page [\[undefined\]](#). See [\[BDB\]](#), page [\[undefined\]](#).

The optimisation section in the manual covers a lot of different aspects of how to tune applications. See [\[Tips\]](#), page [\[undefined\]](#).

E.5 Comments about RTS threads

I have tried to use the RTS thread packages with MySQL but stumbled on the following problems:

They use an old version of a lot of POSIX calls and it is very tedious to make wrappers for all functions. I am inclined to think that it would be easier to change the thread libraries to the newest POSIX specification.

Some wrappers are already written. See `'mysys/my_pthread.c'` for more info.

At least the following should be changed:

`pthread_get_specific` should use one argument. `sigwait` should take two arguments. A lot of functions (at least `pthread_cond_wait`, `pthread_cond_timedwait`) should return the error code on error. Now they return -1 and set `errno`.

Another problem is that user-level threads use the `ALRM` signal and this aborts a lot of functions (`read`, `write`, `open`...). MySQL should do a retry on interrupt on all of these but it is not that easy to verify it.

The biggest unsolved problem is the following:

To get thread-level alarms I changed `'mysys/thr_alarm.c'` to wait between alarms with `pthread_cond_timedwait()`, but this aborts with error `EINTR`. I tried to debug the thread library as to why this happens, but couldn't find any easy solution.

If someone wants to try MySQL with RTS threads I suggest the following:

- Change functions MySQL uses from the thread library to POSIX. This shouldn't take that long.
- Compile all libraries with the `-DHAVE_rts_threads`.
- Compile `thr_alarm`.
- If there are some small differences in the implementation, they may be fixed by changing `'my_pthread.h'` and `'my_pthread.c'`.
- Run `thr_alarm`. If it runs without any "warning", "error" or aborted messages, you are on the right track. Here is a successful run on Solaris:

```

Main thread: 1
Thread 0 (5) started
Thread: 5 Waiting
process_alarm
Thread 1 (6) started
Thread: 6 Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6 Slept for 1 (1) sec
Thread: 6 Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6 Slept for 2 (2) sec
Thread: 6 Simulation of no alarm needed
Thread: 6 Slept for 0 (3) sec
Thread: 6 Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6 Slept for 4 (4) sec
Thread: 6 Waiting
process_alarm
thread_alarm
Thread: 5 Slept for 10 (10) sec
Thread: 5 Waiting
process_alarm
process_alarm
thread_alarm

```

```

Thread: 6 Slept for 5 (5) sec
Thread: 6 Waiting
process_alarm
process_alarm

...
thread_alarm
Thread: 5 Slept for 0 (1) sec
end

```

E.6 Differences between different thread packages

MySQL is very dependent on the thread package used. So when choosing a good platform for MySQL, the thread package is very important.

There are at least three types of thread packages:

- User threads in a single process. Thread switching is managed with alarms and the threads library manages all non-thread-safe functions with locks. Read, write and select operations are usually managed with a thread-specific select that switches to another thread if the running threads have to wait for data. If the user thread packages are integrated in the standard libs (FreeBSD and BSDI threads) the thread package requires less overhead than thread packages that have to map all unsafe calls (MIT-pthreads, FSU Pthreads and RTS threads). In some environments (for example, SCO), all system calls are thread-safe so the mapping can be done very easily (FSU Pthreads on SCO). Downside: All mapped calls take a little time and it's quite tricky to be able to handle all situations. There are usually also some system calls that are not handled by the thread package (like MIT-pthreads and sockets). Thread scheduling isn't always optimal.
- User threads in separate processes. Thread switching is done by the kernel and all data are shared between threads. The thread package manages the standard thread calls to allow sharing data between threads. LinuxThreads is using this method. Downside: Lots of processes. Thread creating is slow. If one thread dies the rest are usually left hanging and you must kill them all before restarting. Thread switching is somewhat expensive.
- Kernel threads. Thread switching is handled by the thread library or the kernel and is very fast. Everything is done in one process, but on some systems, `ps` may show the different threads. If one thread aborts, the whole process aborts. Most system calls are thread-safe and should require very little overhead. Solaris, HP-UX, AIX and OSF/1 have kernel threads.

In some systems kernel threads are managed by integrating user level threads in the system libraries. In such cases, the thread switching can only be done by the thread library and the kernel isn't really "thread aware".

Appendix F Environment Variables

Here is a list of all the environment variables that are used directly or indirectly by MySQL. Most of these can also be found in other places in this manual.

Note that any options on the command-line will take precedence over values specified in configuration files and environment variables, and values in configuration files take precedence over values in environment variables.

In many cases it's preferable to use a configure file instead of environment variables to modify the behaviour of MySQL. See [<undefined> \[Option files\], page <undefined>](#).

Variable	Description
CCX	Set this to your C++ compiler when running configure.
CC	Set this to your C compiler when running configure.
CFLAGS	Flags for your C compiler when running configure.
CXXFLAGS	Flags for your C++ compiler when running configure.
DBI_USER	The default user name for Perl DBI.
DBI_TRACE	Used when tracing Perl DBI.
HOME	The default path for the <code>mysql</code> history file is <code>'\$HOME/.mysql_history'</code> .
LD_RUN_PATH	Used to specify where your <code>'libmysqlclient.so'</code> is.
MYSQL_DEBUG	Debug-trace options when debugging.
MYSQL_HISTFILE	The path to the <code>mysql</code> history file.
MYSQL_HOST	Default host name used by the <code>mysql</code> command-line client.
MYSQL_PS1	Command prompt to use in the <code>mysql</code> command-line client. See <undefined> [mysql], page <undefined> .
MYSQL_PWD	The default password when connecting to <code>mysqld</code> . Note that use of this is insecure!
MYSQL_TCP_PORT	The default TCP/IP port.
MYSQL_UNIX_PORT	The default socket; used for connections to <code>localhost</code> .
PATH	Used by the shell to find the MySQL programs.
TMPDIR	The directory where temporary tables/files are created.
TZ	This should be set to your local time zone. See <undefined> [Timezone problems], page <undefined> .
UMASK_DIR	The user-directory creation mask when creating directories. Note that this is ANDed with <code>UMASK</code> !
UMASK	The user-file creation mask when creating files.
USER	The default user on Windows to use when connecting to <code>mysqld</code> .

Appendix G MySQL Regular Expressions

A regular expression (regex) is a powerful way of specifying a complex search.

MySQL uses Henry Spencer's implementation of regular expressions, which is aimed at conformance with POSIX 1003.2. MySQL uses the extended version.

This is a simplistic reference that skips the details. To get more exact information, see Henry Spencer's `regex(7)` manual page that is included in the source distribution. See [Credits](#), page [Credits](#).

A regular expression describes a set of strings. The simplest regex is one that has no special characters in it. For example, the regex `hello` matches `hello` and nothing else.

Non-trivial regular expressions use certain special constructs so that they can match more than one string. For example, the regex `hello|word` matches either the string `hello` or the string `word`.

As a more complex example, the regex `B[an]*s` matches any of the strings `Bananas`, `Baaaaas`, `Bs`, and any other string starting with a `B`, ending with an `s`, and containing any number of `a` or `n` characters in between.

A regular expression may use any of the following special characters/constructs:

<code>^</code>	Match the beginning of a string.	
	<code>mysql> SELECT "fo\nfo" REGEXP "^fo\$";</code>	<code>-> 0</code>
	<code>mysql> SELECT "fofo" REGEXP "^fo";</code>	<code>-> 1</code>
<code>\$</code>	Match the end of a string.	
	<code>mysql> SELECT "fo\no" REGEXP "fo\no\$";</code>	<code>-> 1</code>
	<code>mysql> SELECT "fo\no" REGEXP "fo\$";</code>	<code>-> 0</code>
<code>.</code>	Match any character (including newline).	
	<code>mysql> SELECT "fofo" REGEXP "^f.*";</code>	<code>-> 1</code>
	<code>mysql> SELECT "fo\nfo" REGEXP "^f.*";</code>	<code>-> 1</code>
<code>a*</code>	Match any sequence of zero or more a characters.	
	<code>mysql> SELECT "Ban" REGEXP "^Ba*n";</code>	<code>-> 1</code>
	<code>mysql> SELECT "Baaan" REGEXP "^Ba*n";</code>	<code>-> 1</code>
	<code>mysql> SELECT "Bn" REGEXP "^Ba*n";</code>	<code>-> 1</code>
<code>a+</code>	Match any sequence of one or more a characters.	
	<code>mysql> SELECT "Ban" REGEXP "^Ba+n";</code>	<code>-> 1</code>
	<code>mysql> SELECT "Bn" REGEXP "^Ba+n";</code>	<code>-> 0</code>
<code>a?</code>	Match either zero or one a character.	
	<code>mysql> SELECT "Bn" REGEXP "^Ba?n";</code>	<code>-> 1</code>
	<code>mysql> SELECT "Ban" REGEXP "^Ba?n";</code>	<code>-> 1</code>
	<code>mysql> SELECT "Baan" REGEXP "^Ba?n";</code>	<code>-> 0</code>
<code>de abc</code>	Match either of the sequences <code>de</code> or <code>abc</code> .	
	<code>mysql> SELECT "pi" REGEXP "pi apa";</code>	<code>-> 1</code>
	<code>mysql> SELECT "axe" REGEXP "pi apa";</code>	<code>-> 0</code>
	<code>mysql> SELECT "apa" REGEXP "pi apa";</code>	<code>-> 1</code>
	<code>mysql> SELECT "apa" REGEXP "^(pi apa)\$";</code>	<code>-> 1</code>

- ```
mysql> SELECT "pi" REGEXP "^(pi|apa)$"; -> 1
mysql> SELECT "pix" REGEXP "^(pi|apa)$"; -> 0
```
- (abc)\*** Match zero or more instances of the sequence `abc`.
- ```
mysql> SELECT "pi" REGEXP "^(pi)*$";        -> 1
mysql> SELECT "pip" REGEXP "^(pi)*$";       -> 0
mysql> SELECT "pipi" REGEXP "^(pi)*$";      -> 1
```
- {1}**
- {2,3}** There is a more general way of writing regexps that match many occurrences of the previous atom.
- a*** Can be written as `a{0,}`.
- a+** Can be written as `a{1,}`.
- a?** Can be written as `a{0,1}`.
- To be more precise, an atom followed by a bound containing one integer `i` and no comma matches a sequence of exactly `i` matches of the atom. An atom followed by a bound containing one integer `i` and a comma matches a sequence of `i` or more matches of the atom. An atom followed by a bound containing two integers `i` and `j` matches a sequence of `i` through `j` (inclusive) matches of the atom.
- Both arguments must be in the range from 0 to `RE_DUP_MAX` (default 255), inclusive. If there are two arguments, the second must be greater than or equal to the first.
- [a-dX]**
- [^a-dX]** Matches any character which is (or is not, if `^` is used) either `a`, `b`, `c`, `d` or `X`. To include a literal `]` character, it must immediately follow the opening bracket `[`. To include a literal `-` character, it must be written first or last. So `[0-9]` matches any decimal digit. Any character that does not have a defined meaning inside a `[]` pair has no special meaning and matches only itself.
- ```
mysql> SELECT "aXbc" REGEXP "[a-dXYZ]"; -> 1
mysql> SELECT "aXbc" REGEXP "[a-dXYZ]$"; -> 0
mysql> SELECT "aXbc" REGEXP "[a-dXYZ]+$"; -> 1
mysql> SELECT "aXbc" REGEXP "[^a-dXYZ]+$"; -> 0
mysql> SELECT "gheis" REGEXP "[^a-dXYZ]+$"; -> 1
mysql> SELECT "gheisa" REGEXP "[^a-dXYZ]+$"; -> 0
```
- [.characters.]**
- The sequence of characters of that collating element. The sequence is a single element of the bracket expression's list. A bracket expression containing a multi-character collating element can thus match more than one character, for example, if the collating sequence includes a `ch` collating element, then the regular expression `[.ch.]*c` matches the first five characters of `chchcc`.
- [=character\_class=]**
- An equivalence class, standing for the sequences of characters of all collating elements equivalent to that one, including itself.

For example, if `o` and `(+)` are the members of an equivalence class, then `[o=]`, `[=(+)=]`, and `[o(+)]` are all synonymous. An equivalence class may not be an endpoint of a range.

#### `[[:character_class:]]`

Within a bracket expression, the name of a character class enclosed in `[[: and :]]` stands for the list of all characters belonging to that class. Standard character class names are:

| Name  | Name  | Name   |
|-------|-------|--------|
| alnum | digit | punct  |
| alpha | graph | space  |
| blank | lower | upper  |
| cntrl | print | xdigit |

These stand for the character classes defined in the `ctype(3)` manual page. A locale may provide others. A character class may not be used as an endpoint of a range.

```
mysql> SELECT "justalnums" REGEXP "[[:alnum:]]+"; -> 1
mysql> SELECT "!!" REGEXP "[[:alnum:]]+"; -> 0
```

#### `[[:<:]]`

#### `[[:>:]]`

These match the null string at the beginning and end of a word respectively. A word is defined as a sequence of word characters which is neither preceded nor followed by word characters. A word character is an alphanumeric character (as defined by `ctype(3)`) or an underscore (`_`).

```
mysql> SELECT "a word a" REGEXP "[[:<:]]word[[:>:]]"; -> 1
mysql> SELECT "a xword a" REGEXP "[[:<:]]word[[:>:]]"; -> 0
mysql> SELECT "weeknights" REGEXP "^((wee|week)(knights|nights))$"; -> 1
```

# Appendix H GNU General Public License

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.  
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

## Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software---to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

## TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The “Program”, below, refers to any such program or work, and a “work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification”.) Each licensee is addressed as “you”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
  - a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
  - b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
  - c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
  - a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
  - b. Accompany it with a written offer, valid for at least three years, to give any third-party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
  - c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software

which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

## **NO WARRANTY**

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

## **END OF TERMS AND CONDITIONS**

## How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

*one line to give the program's name and a brief idea of what it does.*

Copyright (C) yyyy *name of author*

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) 19yy *name of author*

Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.

This is free software, and you are welcome to redistribute it

under certain conditions; type ‘show c’ for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘show w’ and ‘show c’; they could even be mouse-clicks or menu items---whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program ‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

*signature of Ty Coon, 1 April 1989*

Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

# Appendix I GNU Lesser General Public License

Version 2.1, February 1999

Copyright © 1991, 1999 Free Software Foundation, Inc.  
59 Temple Place -- Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

## Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software---to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software---typically libraries---of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by

obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the *Lesser* General Public License because it does *Less* to protect the user's freedom than the ordinary General Public License. It also provides other free software developers *Less* of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is *Less* protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

## **TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION**

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The “Library”, below, refers to any such software library or work which has been distributed under these terms. A “work based on the Library” means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term “modification”.)

“Source code” for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library’s complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
  - a. The modified work must itself be a software library.
  - b. You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
  - c. You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
  - d. If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply

to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a “work that uses the Library”. Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a “work that uses the Library” with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a “work that uses the library”. The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a “work that uses the Library” uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a “work that uses the Library” with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer’s own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

- a. Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable “work that uses the Library”, as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
- b. Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at runtime a copy of the library already present on the user’s computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.
- c. Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
- d. If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
- e. Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the “work that uses the Library” must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components

(compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:
  - a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
  - b. Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.
8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.
10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.
11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.
14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

## **NO WARRANTY**

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

**END OF TERMS AND CONDITIONS**

## How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

*one line to give the library’s name and an idea of what it does.*  
 Copyright (C) year name of author

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library ‘Frob’ (a library for tweaking knobs) written by James Random Hacker.

*signature of Ty Coon, 1 April 1990*  
 Ty Coon, President of Vice

That’s all there is to it!

## Índices dos comandos, tipos e funções SQL

(Index is nonexistent)

## Concept Index

(Index is nonexistent)