MySQL 4.1 リファレンスマニュアル

MySQL 4.1 リファレンスマニュアル

This is a translation of the MySQL Reference Manual that can be found at dev.mysql.com. The original Reference Manual is in English, and this translation is not necessarily as up to date as the English version.

概要

Document generated on: 2010-03-11 (改訂: 548)

目次

Preface

これは MySQL データベースシステム のリファレンスマニュアルです。バージョン 5.0.6-beta の MySQL サーバ について記述しています。変更点を常に説明しています。そのため、古いバージョン (3.23 または 4.0 製品版など) を使用している場合でも、このマニュアルは参考になります。バージョン 5.0 (開発版) のリファレンスもあります。

第1章 一般情報

MySQL (R) ソフトウェアによって、非常に高速で堅牢なマルチスレッド形式のマルチユーザ SQL (Structured Query Language) データベースサーバが実現します。MySQL サーバは、ミッションクリティカルで負荷が高い運用システムにも、大規模に展開されるソフトウェアへの組み込みにも対応するように設計されています。MySQL は、MySQL AB の商標です。

MySQL ソフトウェアはデュアルライセンス製品です。ユーザは、GNU 一般公衆利用許諾契約書 (http://www.fsf.org/licenses/) の条件に基づいてオープンソース/フリーソフトウェア製品として MySQL ソフトウェアを使用することも、MySQL AB から標準の商用ライセンスを購入することもできます。 See 項1.4.「MySQL のサポートとライセンス」。

MySQL の Web サイト (http://www.mysql.com/) には、MySQL ソフトウェアに関する最新情報が掲載されています。

このマニュアルで特に興味深いセクションは以下のとおりです。

- MySQL データベースサーバの背景にある会社に関する情報については、項1.3.「MySQL AB の概要」 を参照。
- MySQL データベースサーバの機能に関する説明については、項1.2.2.「MySQL の主な機能」 を参照。
- インストールの手順については、章 2. MySQL のインストール を参照。
- 新しいアーキテクチャまたはオペレーティングシステムへの MySQL Database Software の移植に関するヒントについては、付録 E. 他システムへの移植 を参照。
- バージョン 4.0 リリースからのアップグレードに関する情報については、項2.5.1.「バージョン 4.0 から 4.1 へのアップグレード」 を参照。
- バージョン 3.23 リリースからのアップグレードに関する情報については、項2.5.2.「バージョン 3.23 から 4.0 へのアップグレード」 を参照。
- バージョン 3.22 リリースからのアップグレードに関する情報については、項2.5.3.「バージョン 3.22 から 3.23 へのアップグレード」 を参照。
- MySQL データベースサーバのチュートリアルについては、章 3. MySQL チュートリアル を参照。
- SQL のサンプルとベンチマーク情報については、ベンチマークディレクトリ (ディストリビューションの sql-bench) を参照。
- 新機能とバグ修正の履歴については、付録 D. MySQL Change History を参照。
- 既知のバグや機能上の問題の一覧については、項1.8.6.「MySQL の既知のエラーと設計上の問題」 を参照。
- 今後の計画については、項1.6.「MySQL の今後 (TODO) 」 を参照。
- このプロジェクトへの全貢献者の一覧については、付録 C. 協力者 を参照。

重要:

エラー (多くの場合、バグと呼ばれます) の報告は、質問やコメントと同様に、通常の MySQL メーリングリストにお送

りください。 See 項1.7.1.1.「MySQL メーリングリスト」。 See 項1.7.1.3.「バグまたは問題を報告する方法」。

Unix では、mysqlbug スクリプトを使用してバグレポートを生成します (Windows ディストリビューションについては、basedir にファイル mysqlbug.txt があります。このファイルをバグレポートのテンプレートとして使用してください)。

ソースディストリビューションについては、mysqlbug スクリプトは scripts ディレクトリにあります。バイナリディストリビューションについては、mysqlbug は bin ディレクトリ (MySQL サーバ RPM パッケージの場合は /usr/bin) にあります。

MySQL サーバで重大なセキュリティバグを見つけた場合は、<security@mysql.com> に電子メールをお送りください。

1.1. このマニュアルについて

これは、MySQL のリファレンスマニュアルです。バージョン 5.0.6-beta の MySQL について記述しています。機能に関する変更は常に、バージョンに言及して説明しています。そのため、古いバージョンの MySQL ソフトウェア (3.23 または 4.0 製品版など) を使用している場合でも、このマニュアルは参考になります。バージョン 5.0 (開発版) のリファレンスもあります。

これはリファレンスマニュアルなので、SQL やリレーショナルデータベースの概念に関する一般的な説明は記載していません。

MySQL データベースソフトウェア は継続して開発が行われているので、マニュアルも頻繁に更新されます。このマニュアルの最新版は、http://www.mysql.com/documentation/ から HTML、PDF、Windows HLP 版などのさまざまな形式で入手することができます。

元の文書は Texinfo ファイルです。HTML 版は、変更された texi2html を使用して自動的に生成されます。プレーンテキスト版と Info 版は、makeinfo を使用して生成されます。PostScript 版は、texi2dvi と dvips を使用して生成されます。PDF 版は、pdftex を使用して生成されます。

マニュアル内で情報を見つけるのが困難な場合は、http://www.mysql.com/doc/ にある検索可能なバージョンを試してみてください。

このマニュアルへの追加や修正に関する提案は、マニュアルチーム (http://www.mysql.com/company/contact/) にお送りください。

このマニュアルは当初、David Axmark と Michael (Monty) Widenius によって執筆されました。現在は、Arjen Lentz、Paul DuBois、および Stefan Hinz で構成される MySQL マニュアルチームによって管理されています。その他多数の貢献者については、付録 C. 協力者 を参照してください。

1.1.1. このマニュアルの表記規則

このマニュアルは、以下の表記規則に従って記載されています。

- constant

  固定幅フォントは、コマンドの名前やオプション、SQL ステートメント、データベース名、テーブル名、カラム名、C

コード、Perl コード、および環境変数に使用する。例:``mysqladmin の動作を確認するには、--help オプションを指定して呼び出す。"

- filename

  引用符で囲まれた固定幅フォントは、ファイル名およびパス名に使用する。例:``ディストリビューションは、/usr/local/ ディレクトリ下にインストールされる。"

- 'c'

  引用符で囲まれた固定幅フォントは、文字のシーケンスを示す場合にも使用する。例:``ワイルドカードを指定するには、'%' 文字を使用する。"

- italic

  イタリックフォントは、このように強調する場合に使用する。

- boldface

  太字フォントは、表の見出しや特に強い強調を表す場合に使用する。

特定のプログラムからコマンドを実行する必要があることを示す場合、コマンドの前にプログラムとプロンプトを記述します。たとえば、shell> はログインシェルから実行するコマンドを示し、mysql> は mysql クライアントプログラムから実行するコマンドを示します。

```
shell> シェルコマンド
mysql> mysql コマンド
```

``シェル" はコマンドインタープリタです。Unix では通常、sh や csh などのプログラムです。Windows では、Windows コンソールで通常実行される command.com や cmd.exe です。

シェルコマンドは、Bourne シェル構文を使用して表します。csh 形式のシェルを使用している場合は、多少異なるコマンドを発行しなければならないことがあります。たとえば、環境変数を設定し、コマンドを実行するシーケンスは、Bourne シェル構文では次のようになります。

```
shell> VARNAME=value some_command
```

csh または tcsh の場合は、このシーケンスを次のように実行します。

```
shell> setenv VARNAME value
shell> some_command
```

データベース名、テーブル名、およびカラム名は多くの場合、コマンドに代入する必要があります。このような代入が必要であることを示す場合、このマニュアルでは db_name、tbl_name、および col_name を使用します。たとえば、次のようなステートメントがあるとします。

```
mysql> SELECT col_name FROM db_name.tbl_name;
```

これは、同様のステートメントを入力する場合、データベース名、テーブル名、およびカラム名を自分で指定することを意味します。たとえば、次のようになります。

```
mysql> SELECT author_name FROM biblio_db.author_list;
```

SQL キーワードは大文字と小文字を区別しないので、大文字で記述されることも小文字で記述されることもあります。ただし、このマニュアルでは大文字を使用します。

構文の説明で省略可能な単語や節を表す場合、角かっこ ('[' および ']') を使用します。たとえば、次のステートメントでは、IF EXISTS は省略可能です。

```
DROP TABLE [IF EXISTS] tbl_name
```

構文要素が複数の選択候補で構成される場合、それらの候補を縦線 ('|') で区切ります。選択候補のいずれかを選択できる場合は、次のように、それらの候補を角かっこ ('[' および ']') 内に列挙します。

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

選択候補のいずれかを選択しなければならない場合は、次のように、それらの候補を中かっこ ('{' および '}') 内に列挙します。

```
{DESCRIBE | DESC} tbl_name {col_name | wild}
```

1.2. MySQL データベース管理システムの概要

MySQL は最もよく知られているオープンソース SQL データベース管理システムで、MySQL AB によって開発、提供、サポートが行われています。MySQL AB は MySQL の開発者によって創設された営利会社で、MySQL データベース管理システムに関連するサービスを提供することでビジネスを構築しています。 See 項1.3. 「MySQL AB の概要」。

MySQL の Web サイト (http://www.mysql.com/) には、MySQL ソフトウェアと MySQL AB に関する最新情報が掲載されています。

- MySQL はデータベース管理システムである。

  データベースは、構造化されたデータの集合である。データベースには、簡単なショッピングリストから絵画のギャラリー、または社内ネットワークの膨大な量の情報など、さまざまなものがある。コンピュータデータベースに格納されているデータに対して追加、アクセス、および処理などを行うには、MySQL サーバのようなデータベース管理システムが必要となる。コンピュータは大量のデータの処理に非常に優れているので、データベース管理システムは、スタンドアロンのユーティリティまたは他のアプリケーションの一部として、データ処理の中心的な役割を果たす。

- MySQL はリレーショナルデータベース管理システムである。

  リレーショナルデータベースでは、1 つの大きな保管領域にすべてのデータが格納されるのではなく、個別のテーブルにデータが格納される。これにより、速度と柔軟性が向上する。``MySQL'' の SQL は ``Structured Query Language'' を表す。SQL はデータベースへのアクセスに使用される最も一般的な標準化言語で、ANSI/ISO SQL 標準によって定義されている (SQL 標準は 1986 年以降開発が繰り返され、複数のバージョンがある。このマニュアルでは、``SQL-92'' は 1992 年にリリースされた標準、``SQL-99'' は 1999 年にリリースされた標準、``SQL:2003'' は 2003 年半ばにリリース予定の標準を表す。``SQL 標準'' という用語は、任意の時点における現行バージョンの SQL 標準を表す場合に使用する) 。

- MySQL ソフトウェアはオープンソースである。

  オープンソースとは、だれもがそのソフトウェアを使用し、変更できることを意味する。MySQL ソフトウェアは、だれもが無料でインターネットからダウンロードし、使用することができる。必要に応じて、ソースコードを調べ、ニーズに合わせて変更することができる。MySQL ソフトウェアでは、GPL (GNU 一般公衆利用許諾契約書 (http://www.fsf.org/licenses/)) に基づいて、さまざまな状況でのソフトウェアの使用において許可されている事項と禁止されている事項が規定されている。GPL では不都合な場合や商用アプリケーションに MySQL コードを組み込む必要がある場合は、商用ライセンス版を購入することができる。 See 項1.4.3.「MySQL ライセンス」。

- MySQL データベースサーバを使用するメリット

  MySQL データベースサーバ は、非常に高速で信頼性があり、簡単に使用することができる。それを求めていたのなら、ぜひ試していただきたい。また、MySQL サーバには、ユーザと密接に協力して開発された実用的な機能が備わっている。ベンチマークページで、他のデータベース管理システムと MySQL サーバのパフォーマンスを比較することができる。 See 項5.1.4.「MySQL ベンチマークスィート」。

  MySQL サーバは当初、既存のソリューションよりもはるかに速く大規模なデータベースを処理することを目的として開発され、多くを要求される運用環境で数年間にわたって使用されている。引き続き開発が行われているが、MySQL サーバは現在、便利で多彩な機能を備えている。その接続性、速度、安全性によって、MySQL サーバはインターネット上のデータベースへのアクセスに非常に適している。

- MySQL サーバの技術的な機能

  詳細な技術情報については、章 6. MySQL SQL言語リファレンス を参照。MySQL データベースソフトウェア は、さまざまなバックエンドをサポートするマルチスレッド形式の SQL サーバ、複数の異なるクライアントプログラムおよびライブラリ、管理ツール、幅広いアプリケーションプログラミングインタフェース (API) から成るクライアント/サーバシステムである。

  また、MySQL サーバはアプリケーションへのリンクが可能なマルチスレッドライブラリとして提供されており、さらに小規模で高速な、管理しやすい製品を作り出すことができる。

- MySQL をサポートする多数のソフトウェア

  必要なアプリケーションや言語ですでに MySQL データベースサーバがサポートされていると思われる。

MySQL の正式な発音は ``My Ess Que Ell'' ですが ( ``my sequel'' ではありません) 、``my sequel'' と発音したり、ローカライズされた他の方法で発音してもかまいません。

1.2.1. MySQL の歴史

当初は、高速で低レベルな独自の (ISAM) ルーチンを使用してテーブルに接続するために mSQL を使用するつもりでした。しかし、いくつかのテストを行った結果、mSQL はそれほど高速でも柔軟でもないため、ニーズに合わないという結論に至りました。これが要因となって、私たちのデータベースへの新しい SQL インタフェースを開発することになりました。ただし、mSQL とほとんど同じ API インタフェースを使用することにしました。この API を選択したのは、mSQL で使用するために記述されたサードパーティのコードを MySQL で使用するために簡単に移植できるようにするためです。

MySQL という名前の由来は明らかではありません。当社の基本ディレクトリおよび多数のライブラリやツールには、10 年以上にわたって ``my'' というプリフィックスが使われています。また、共同創設者 Monty Widenius の娘の名前も My です。そのため、このうちのどちらが MySQL の名前の由来となったのかは、私たちにとってもいまだに謎なのです。

MySQL のイルカ (当社のロゴ) の名前は Sakila です。Sakila は、当社の「イルカのネーミング」コンテストで、ユーザによって提案された膨大な数の名前の中から MySQL AB の創設者が選んだものです。この名前を提案したのは、アフリカのスワジランド出身の Ambrose Twebaze というオープンソースソフトウェアの開発者でした。Ambrose によると、Sakila という名前は、スワジランドの現地語であるシスワティ語にルーツがあるということです。また、Sakila は、Ambrose の出生国であるウガンダに近い、タンザニアのアルーシャにある町の名前でもあります。

1.2.2. MySQL の主な機能

MySQL データベースソフトウェア の重要な特徴の一部を以下で説明します。 See 項1.5.1. 「MySQL 4.0 の概要」。

- 内部および移植性
  - C および C++ で記述されている。
  - さまざまなコンパイラでテストされている。
  - さまざまなプラットフォームで動作する。 See 項2.2.3. 「MySQL がサポートしているオペレーティングシステム」。
  - 移植性のために GNU Automake、Autoconf、および Libtool を使用している。
  - C、C++、Eiffel、Java、Perl、PHP、Python、Ruby、および Tcl の API が使用可能である。 See 章 11. MySQL API。
  - カーネルスレッドを使用した完全なマルチスレッド。そのため、使用可能な場合、複数の CPU を簡単に使用することができる。
  - トランザクションストレージエンジンと非トランザクションストレージエンジンを備えている。
  - インデックス圧縮を備えた非常に高速な B-tree ディスクテーブル (MyISAM) を使用している。
  - 別のストレージエンジンの追加が比較的容易である。これは、社内データベースへの SQL インタフェースを追加する場合に便利である。
  - スレッドベースの非常に高速なメモリ割り当てシステム。
  - 最適化された one-sweep multi-join を使用した非常に迅速な結合。
  - テンポラリテーブルとして使用されるメモリ内ハッシュテーブル。
  - 高度に最適化されたクラスライブラリから SQL 関数が実装されるため、最大限の速度が確保される。通常は、クエリの初期化後にメモリ割り当てが行われることはない。
  - MySQL コードは Purify (市販のメモリリーク検出システム) と Valgrind と呼ばれる GPL ツール (http://developer.kde.org/~sewardj/) を使用してテストされている。
  - クライアント/サーバまたは組み込み (リンク) バージョンとして使用可能である。
- カラム型
  - 多数のカラム型: 1、2、3、4、および 8 バイト長の符号付き/符号なし整数、FLOAT、DOUBLE、CHAR、VARCHAR、TEXT、BLOB、DATE、TIME、DATETIME、TIMESTAMP、YEAR、SET、および ENUM 型。 See 項

6.2.「カラム型」。

- 固定長および可変長のレコード。

- コマンドと関数
  - クエリの SELECT 節および WHERE 節での演算子と関数の完全なサポート。たとえば、次のように使用することができる。

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

  - SQL の GROUP BY 節および ORDER BY 節の完全なサポート。グループ関数 (COUNT()、COUNT(DISTINCT ...) 、AVG()、STD()、SUM()、MAX()、MIN()、および GROUP_CONCAT()) のサポート。
  - 標準の SQL 構文および ODBC 構文での LEFT OUTER JOIN および RIGHT OUTER JOIN のサポート。
  - SQL-92 で必要な、テーブルおよびカラムにおけるエイリアスのサポート。
  - DELETE、INSERT、REPLACE、および UPDATE は、変更された (影響を受けた) レコードの数を返す。サーバに接続する際にフラグを設定することで、代わりに一致したレコードの数を返すことも可能である。
  - MySQL 固有の SHOW コマンドを使用すると、データベース、テーブル、およびインデックスに関する情報を取得することができる。EXPLAIN コマンドを使用すると、オプティマイザによるクエリの解決方法を決定することができる。
  - 関数名は、テーブル名やカラム名と衝突しない。たとえば、ABS は有効なカラム名である。関数呼び出しで、関数名とその後に続く '(' との間にスペースを使用できない点が唯一の制限事項である。 See 項6.1.7.「MySQL での予約語の扱い」。
  - 同一のクエリにさまざまなデータベース内のテーブルを混在させることができる (バージョン 3.22 以降) 。

- セキュリティ
  - 非常に柔軟で安全な特権およびパスワードシステム。ホストベースの検証が可能である。サーバに接続する際にすべてのパスワードトラフィックが暗号化されるので、パスワードは安全である。

- 拡張性と範囲
  - 大規模なデータベースを処理する。当社は、MySQL サーバを使用して 50,000,000 レコードが格納されたデータベースを処理している。また、MySQL サーバを使用して 60,000 テーブル、約 5,000,000,000 レコードを処理しているユーザもいる。
  - 各テーブルで最高 32 個のインデックスが使用可能である。各インデックスは、1 から 16 個のカラムまたはカラムの一部で構成される。インデックスの最大幅は 500 バイトである (これは、MySQL サーバのコンパイル時に変更可能である) 。インデックスでは、CHAR 型または VARCHAR 型のカラムのプリフィックスを使用することができる。

- 接続性
  - クライアントは、あらゆるプラットフォームで TCP/IP ソケットを使用して MySQL サーバに接続することができる

。NT ファミリ (NT、2000、または XP) の Windows システムでは、クライアントは名前付きパイプを使用して接続することができる。Unix システムでは、Unix ドメインソケットファイルを使用して接続することができる。

- Connector/ODBC インタフェースによって、ODBC (Open-DataBase-Connectivity) 接続を使用するクライアントプログラムに MySQL サポートが提供される。たとえば、MS Access を使用して MySQL サーバに接続することができる。クライアントは、Windows と Unix のどちらで実行されていてもかまわない。Connector/ODBC ソースが使用可能である。他の多くの機能と同様に、ODBC 2.5 のすべての機能がサポートされる。 See 項11.2. 「MySQL の ODBC サポート」。

- ローカライズ
  - サーバからクライアントへ多数の言語でエラーメッセージを送信することができる。 See 項4.7.2. 「英語以外のエラーメッセージ」。
  - ISO-8859-1 (Latin1) 、german、big5、ujis などのさまざまなキャラクタセットの完全なサポート。たとえば、スカンジナビア語の文字 'å'、'ä'、および 'ö' をテーブル名やカラム名で使用することができる。
  - すべてのデータが、選択したキャラクタセットで保存される。通常の文字列カラムの比較はすべて、大文字と小文字を区別しない。
  - ソートは、選択したキャラクタセットに基づいて行われる (デフォルトはスウェーデン語によるソート) 。これは、MySQL サーバの起動時に変更することができる。非常に高度なソートの例については、チェコ語のソートコードを参照。MySQL サーバではさまざまなキャラクタセットがサポートされており、コンパイル時および実行時に指定することができる。

- クライアントとツール
  - MySQL サーバには、テーブルのチェック、最適化、および修復を行う SQL ステートメントのサポートが組み込まれている。これらのステートメントは、mysqlcheck クライアントを介してコマンドラインから使用可能である。また、MySQL には、MyISAM テーブルでこれらの操作を実行するための myisamchk という非常に高速なコマンドラインユーティリティが組み込まれている。 See 章 4. データベース管理。
  - --help または -? オプションを指定して呼び出すと、すべての MySQL プログラムでオンラインヘルプを参照することができる。

1.2.3. MySQL の安定性

このセクションでは、``MySQL サーバの安定性" と ``プロジェクトにおける MySQL サーバの信頼性" という問題を扱います。これらの問題を明らかにし、多数の潜在ユーザに関連する重要な質問に答えます。このセクションに記載されている情報は、問題の特定や使用方法の報告が非常に活発に行われているメーリングリストから収集したデータに基づいています。

元のコードは 1980 年代初期に記述されたもので、安定したコードベースとなっています。また、ISAM テーブル形式には下位互換性が維持されています。MySQL AB の前身である TcX では、1996 年半ば以来、MySQL コードはプロジェクトで問題なく動作していました。MySQL データベースソフトウェア がより広く一般にリリースされるとすぐに、新しいユーザが ``テストされていないコード" をいくつか見つけました。それ以降の新しいリリースごとに、移植性に関する問題は少なくなっています (また一方では、新しいリリースごとに多数の新機能が追加されています) 。

MySQL サーバの各リリースは常に実用的です。問題が発生するのは、ユーザが ``グレーゾーン" のコードを使用しようと

したときだけです。もっとも、新しいユーザはグレーゾーンとは何かを知りません。そのため、このセクションでは、現時点で認識されているその領域を説明します。説明は主に、MySQL サーバのバージョン 3.23 および 4.0 を対象とします。最新バージョンでは、バグセクションに記載されている設計関連のバグを除き、報告されている既知のバグはすべて修正されています。 See 項1.8.6.「MySQL の既知のエラーと設計上の問題」。

MySQL サーバは、複数層の独立したモジュールで構成されています。新しいモジュールの一部とそれらのテストステータスを以下に示します。

- レプリケーション --- ガンマ

  レプリケーションによる大規模なサーバグループが運用されており、良好な結果を得ている。MySQL 4.x では、拡張されたレプリケーション機能に対する取り組みが引き続き行われている。

- InnoDB テーブル --- 安定（3.23.49 以降の 3.23）

  InnoDB トランザクションストレージエンジンは、MySQL 3.23 ツリーのバージョン 3.23.49 以降で安定していると宣言されている。InnoDB は、大規模で負荷が高い運用システムで使用されている。

- BDB テーブル --- ガンマ

  Berkeley DB コードは非常に安定しているが、MySQL サーバでは引き続き BDB トランザクションストレージエンジンインタフェースの改良を行っている。そのため、他のテーブル型のように十分にテストされるにはしばらく時間が必要である。

- 全文検索 --- ベータ

  全文検索は動作するが、まだ広範には使用されていない。MySQL 4.0 には、重要な拡張機能が実装されている。

- MyODBC 3.51（ODBC SDK 3.51 を使用）--- 安定

  広く運用されている。ODBC ドライバや基盤となっているデータベースサーバとは無関係の、アプリケーションに関連していると思われる問題が発生している。

- MyISAM テーブルの自動リカバリ --- ガンマ

  このステータスは、MyISAM ストレージエンジンの新しいコードのみに該当する。このコードでは、テーブルを開く際に、前にそのテーブルが正しく閉じられたかどうかがチェックされ、正しく閉じられていない場合はテーブルの自動チェック/修復が実行される。

- 一括挿入 --- アルファ

  多数のレコードをより速く挿入するための MySQL 4.0 における MyISAM テーブルの新機能である。

- ロック --- ガンマ

  これはシステム依存である。一部のシステムでは、オペレーティングシステム標準のロック（fcntl()）を使用することに大きな問題がある。このような場合、--skip-external-locking フラグを指定して mysqld を実行する必要がある。問題は、一部の Linux システムと、NFS によってマウントされたファイルシステム使用時の SunOS で発生することが確認されている。

MySQL AB では、高品質のサポートを有料でお客様に提供しております。また、だれもが質問することができるコミュニティリソースとして MySQL メーリングリストを提供しております。

バグは通常、パッチによって至急修正されます。重大なバグについては、ほとんどの場合、新しいリリースが提供されます。

1.2.4. MySQL テーブルの最大サイズ

MySQL バージョン 3.22 では、テーブルのサイズは 4 GB (4 ギガバイト) に制限されていました。MySQL バージョン 3.23 の MyISAM テーブル型では、テーブルの最大サイズが 800万テラバイト (2^63 バイト) に増えました。有効なテーブルのサイズがこのように大きくなったことで、現在では、MySQL データベースに効果的なテーブルの最大サイズは通常、MySQL 内部の制限ではなく、オペレーティングシステムのファイルサイズに関する制限によって決まります。

次の表は、オペレーティングシステムのファイルサイズに関する制限の例を示しています。

オペレーティングシステム	ファイルサイズの制限
Linux-Intel 32-bit	2 GB、LFS 使用の場合はそれ以上
Linux-Alpha	8 TB (?)
Solaris 2.5.1	2 GB (パッチにより 4GB まで可)
Solaris 2.6	4 GB (フラグにより変更可能)
Solaris 2.7 Intel	4 GB
Solaris 2.7 UltraSPARC	512 GB

Linux 2.2 では、ext2 ファイルシステム用の LFS パッチを使用することで、2 GB より大きいサイズのテーブルを使用することができます。また、Linux 2.4 には ReiserFS および ext3 が組み込まれており、これらを使用することで大きいファイルがサポートされます。現在の Linux ディストリビューションのほとんどはカーネル 2.4 に基づいており、必要な Large File Support (LFS) パッチすべてがすでに組み込まれています。しかし、それでも、有効な最大ファイルサイズはいくつかの要因によって左右されます。そのうちの 1 つが、MySQL テーブルを格納するために使用されるファイルシステムです。

Linux における LFS の詳細については、Andreas Jaeger の「Large File Support in Linux」 (http://www.suse.de/~aj/linux_lfs.html) を参照してください。

デフォルトでは、MySQL では有効な最大サイズが約 4 GB の内部構造を持つ MyISAM テーブルが作成されます。SHOW TABLE STATUS コマンドまたは myisamchk -dv table_name を使用して、テーブルの最大サイズをチェックすることができます。 See 項4.6.8.「SHOW 構文」。

4 GB より大きいサイズのテーブルが必要な場合 (オペレーティングシステムで大規模ファイルがサポートされていれば)、CREATE TABLE ステートメントで AVG_ROW_LENGTH および MAX_ROWS オプションを指定することができます。4 GB を超えるテーブルを作成するには、これらのオプションを使用してください。See 項6.5.3.「CREATE TABLE 構文」。また、ALTER TABLE を使用して、後でこれらのオプションを設定することもできます。 See 項6.5.4.「ALTER TABLE 構文」。

MyISAM テーブルのファイルサイズに関する制限に対処する方法として、他に次のような方法があります。

- 読み込み専用の大規模テーブルの場合、myisampack を使用してテーブルを圧縮することができる。myisampack では通常、テーブルが少なくとも 50% 圧縮されるので、結果的にはるかに大きいテーブルを使用することが可能である。また、myisampack を使用して、複数のテーブルを 1 つのテーブルにマージすることもできる。 See 項4.8.4.「

myisampack (MySQL 圧縮読み取り専用テーブルジェネレータ) 」。

- また、MyISAM データファイルに関するオペレーティングシステムのファイル制限に対処する方法として、RAID オプションを使用することもできる。 See 項6.5.3. 「CREATE TABLE 構文」。
- MySQL に組み込まれている MERGE ライブラリを使用して、同一のテーブルの集合を 1 つのテーブルとして処理することもできる。See 項7.2. 「MERGE テーブル」。

1.2.5. 西暦 2000 年対応

MySQL サーバ自体は、西暦 2000 年 (Y2K) 対応に関しては何の問題もありません。

- MySQL サーバでは、TIMESTAMP 値については日付を 2037 年に処理する Unix 時間関数を使用する。DATE 値および DATETIME 値については、9999 年までの日付が使用可能である。
- MySQL 日付関数すべてが 1 つのファイル sql/time.cc に格納され、西暦 2000 年に対応するように非常に慎重にコード化されている。
- MySQL バージョン 3.22 以降では、YEAR 型のカラムに 0 年および 1901 年から 2155 年までの年を 1 バイトで格納し、2 桁または 4 桁で表示することができる。2 桁の年はすべて、1970 年から 2069 年までの範囲にあると見なされる。つまり、YEAR 型のカラムに 01 を格納した場合、MySQL サーバでは 2001 として処理される。

次の簡単な例で、MySQL サーバに 2030 年までの日付に関する問題がないことを示します。

```
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)
```

TIMESTAMP 型のカラムの最後の値がゼロになっているのは、最後の年（2050）が TIMESTAMP の上限を超えているためです。TIMESTAMP データ型は現在の時刻を格納するために使用され、32 ビットマシンでは 19700101000000 から 20300101000000 までの値（符号付きの値）をサポートします。64 ビットマシンでは、2106 までの値（符号なしの値）を処理します。

この例は、DATE および DATETIME データ型にも日付の使用に関する問題がないことを示しています。これらのデータ型では、9999 年までの日付が処理されます。

MySQL サーバ自体は Y2K に対応していますが、Y2K に対応していないアプリケーションとともに使用すると、問題が発生することがあります。たとえば、多くの古いアプリケーションでは、4 桁の値ではなく、あいまいな 2 桁の値を使用して年の格納や処理が行われます。この問題は、値が ``ない'' ことを示す場合に 00 や 99 などの値を使用するアプリケーションでは、さらに複雑になる可能性があります。それぞれのアプリケーションはさまざまなプログラマによって記述されており、プログラマごとに異なる規則や日付処理関数を使用している可能性があるので、これらの問題を修正するのは困難な場合があります。

そのため、MySQL サーバに Y2K 問題がなくても、アプリケーションとしてあいまいでない情報を提供することが必要です。年を表す 2 桁の値を含むあいまいな日付入力データの処理に関する MySQL サーバのルールについては、項6.2.2.1.「西暦 2000 年問題と日付型」 を参照してください。

1.3. MySQL AB の概要

MySQL AB は MySQL の創始者と主な開発者の会社で、最初は David Axmark、Allan Larsson、および Michael ``Monty'' Widenius によってスウェーデンに創設されました。

MySQL サーバの開発者はすべて、当社の従業員です。当社は、世界中の多数の国の人々から成るバーチャルな組織です。毎日ネットを使って相互に、またはユーザ、サポータ、およびパートナと広くコミュニケーションを図っています。

当社は、MySQL ソフトウェアの開発と新しいユーザへの当社のデータベースの普及に専念しております。MySQL AB は、MySQL のソースコード、MySQL のロゴと商標、およびこのマニュアルの著作権を所有しています。 See 項1.2.「MySQL データベース管理システムの概要」。

MySQL の本質的価値が、MySQL およびオープンソースへの当社の献身を示しています。

当社は、MySQL データベースソフトウェア が次のようなものであることを願っています。

- 世界中で最も広く使用される、世界最高のデータベースである。
- だれもが入手でき、価格が手ごろである。
- 使いやすい。
- 速度と安全性を確保しながら、改良を続ける。
- 楽しく使用および改良することができる。
- バグがない。

MySQL AB と MySQL AB の従業員は、以下のことを心がけています。

- オープンソースの理念を推進し、オープンソースコミュニティをサポートする。
- 健全なメンバーである。
- 当社の価値観と考え方を共有するパートナを優先する。
- 電子メールに回答し、サポートを提供する。
- 他者とネットワークで結ばれたバーチャルな企業である。
- ソフトウェアの特許権に反対する立場をとる。

MySQL の Web サイト (http://www.mysql.com/) には、MySQL と MySQL AB に関する最新情報が掲載されています。

会社名の ``AB" の部分は、スウェーデン語の ``aktiebolag"、つまり ``株式会社" の頭字語です。したがって、``MySQL 株式会社" という意味です。実際、MySQL AB の子会社として、MySQL Inc. や MySQL GmbH などがあります。これらの子会社はそれぞれ、アメリカとドイツにあります。

1.3.1. MySQL AB のビジネスモデルとサービス

当社に最もよく寄せられる質問の 1 つが ``製品を無償で提供してどのように採算をとっているのですか?" というものです。これに対する回答は次のとおりです。

MySQL AB は、サポート、サービス、商用ライセンス、および口イヤルティで利益を上げています。当社では、これらの収益を製品開発や MySQL ビジネスの拡大に充てています。

当社は、創設以来、常に利益を上げています。2001 年 10 月には、代表的な北欧の投資家と何人かの資金援助者から事業投資を受けました。この投資は、当社のビジネスモデルを固め、継続的な成長の基盤を確立するために使われています。

1.3.1.1. サポート

MySQL AB は、MySQL データベースの創始者と主な開発者が所有し、経営しています。開発者は、お客様やその他のユーザのニーズや問題を常に把握しておくために、サポートの提供に取り組んでいます。当社のサポートはすべて、経験豊富な開発者によって提供されます。実際、難しい質問には、MySQL サーバの最も重要な作成者である Michael Monty Widenius が回答します。 See 項1.4.1.「MySQL AB によって提供されるサポート」。

詳細とさまざまなレベルのサポートの申し込みについては、http://www.mysql.com/support/ を参照するか、当社の販売スタッフ (<sales@mysql.com>) にお問い合わせください。

1.3.1.2. トレーニングと検定

MySQL AB は、MySQL 関連のトレーニングを世界中で実施しています。当社では、オープンコースと企業固有のニーズに合わせた社内トレーニング用コースの両方を用意しております。また、当社のパートナである MySQL 認定トレーニングセンターによる MySQL トレーニングもあります。

当社のトレーニング教材には、当社のマニュアルで使用されているものと同じサンプルデータベースやサンプルアプリケーションが使用されています。また、教材は最新版の MySQL を反映するように常に更新されています。さらに、当社の講師は開発チームによってサポートされているため、トレーニングの質とコース教材の継続的な開発を保証いたします。また、トレーニング中に発生した質問には必ずお答えします。

トレーニングコースに参加することで、MySQL アプリケーションの目標を達成することができます。さらに、以下のことを実現することができます。

- 時間を節約する。
- アプリケーションのパフォーマンスを向上させる。
- 余分なハードウェアの必要性を低減または排除して、コストを削減する。
- セキュリティを強化する。
- 顧客および同僚の満足度を向上させる。
- MySQL 検定の準備をする。

受講者として、またはトレーニングパートナとして当社のトレーニングに関心をお持ちの方は、トレーニングセクション (http://www.mysql.com/training/) を参照するか、当社 (<training@mysql.com>) にお問い合わせください。

MySQL 検定プログラムの詳細については、http://www.mysql.com/certification/ を参照してください。

1.3.1.3. コンサルティング

MySQL AB とその認定パートナは、世界中の MySQL サーバユーザおよび独自のソフトウェアに MySQL サーバを組み込むユーザにコンサルティングサービスを提供しています。

当社のコンサルタントは、データベースの設計と調整、効率的なクエリの作成、最適なパフォーマンスを確保するためのプラットフォームの調整、移行に関する問題の解決、レプリケーションの設定、堅牢なトランザクションアプリケーションの構築などをお手伝いします。また、大規模展開を目的として、独自の製品やアプリケーションに MySQL サーバを組み込むお客様のお手伝いもいたします。

当社のコンサルタントは、開発チームと密接に協力しています。そのため、専門的なサービスの技術的な品質を保証いたします。コンサルティングは、2 日間のパワースタートセッションから、何週間、何か月にもわたるプロジェクトまでさまざまです。当社の専門家は、MySQL サーバだけを扱うわけではありません。PHP や Perl などのプログラミング言語およびスクリプト言語についても対応します。

コンサルティングサービスおよびコンサルティングパートナに関心をお持ちの方は、当社の Web サイトのコンサルティン

グセクション（http://www.mysql.com/consulting/）を参照するか、コンサルティングスタッフ（<consulting@mysql.com>）にお問い合わせください。

1.3.1.4. 商用ライセンス

MySQL データベースは、GNU 一般公衆利用許諾契約書（GPL）に基づいてリリースされています。つまり、MySQL ソフトウェアは、GPL の下、無償で使用することができます。ただし、GPL の条件（アプリケーションも GPL に基づいていなければならないという要件など）に拘束されたくない場合は、MySQL AB から同じ製品の商用ライセンスを購入することができます。http://www.mysql.com/products/pricing.html を参照してください。MySQL AB は MySQL のソースコードの著作権を所有しているので、デュアルライセンスを採用して、GPL および商用ライセンスの下で同じ製品を提供することができます。これにより、オープンソースへの MySQL AB の取り組みに影響が生じることはありません。商用ライセンスが必要な場合の詳細については、項1.4.3.「MySQL ライセンス」 を参照してください。

当社は、MySQL サーバに価値を追加するサードパーティのオープンソース GPL ソフトウェアの商用ライセンスも販売しております。わかりやすい例として、ACID サポート、行レベルのロック、クラッシュリカバリ、マルチバージョニング、外部キーサポートなどを提供する InnoDB トランザクションストレージエンジンがあります。 See 項7.5.「InnoDB テーブル」。

1.3.1.5. パートナ提携

MySQL AB には、トレーニングコース、コンサルティングとサポート、出版、MySQL および関連製品の再販と提供を扱う世界的なパートナプログラムがあります。MySQL AB パートナは Web サイト（http://www.mysql.com/）で公開されるとともに、個々の製品を区別し、ビジネスを促進するために特殊なバージョンの MySQL の商標を使用する権利を得られます。

MySQL AB パートナに関心をお持ちの方は、<partner@mysql.com> に電子メールをお送りください。

MySQL という言葉と MySQL のイルカのロゴは、MySQL AB の商標です。See 項1.4.4.「MySQL AB のロゴと商標」。これらの商標は、MySQL の創始者が何年にもわたって築いてきた意義深い価値を表しています。

MySQL の Web サイト（http://www.mysql.com/）は、開発者やユーザによく知られています。2001 年 10 月には、1,000 万回のページアクセスがありました。当社の Web サイトへの訪問者は、ソフトウェアとハードウェアの両方について購入の決定と推奨を行うグループです。訪問者の 12% が正式に購入を決定し、購入の決定にまったく関連しないのはわずか 9% です。65% を超える訪問者が過去半年以内に 1 件以上のオンライン取り引きを行ったことがあり、70% の訪問者はその後数か月内に取り引きを予定しています。

1.3.2. 問い合わせ先

MySQL の Web サイト（http://www.mysql.com/）には、MySQL と MySQL AB に関する最新情報が掲載されています。

当社のニュースリリース（http://www.mysql.com/news/）に掲載されていないプレス関連のサービスや質問については、<press@mysql.com> に電子メールをお送りください。

MySQL AB と有効なサポート契約を結んでいる方は、MySQL ソフトウェアに関する技術的な質問に対する明確な回答をタイムリーに得ることができます。詳細については、項1.4.1.「MySQL AB によって提供されるサポート」 を参照してください。または、Web サイトの http://www.mysql.com/support/ を参照するか、<sales@mysql.com> に電子メールをお送りください。

MySQL トレーニングについては、トレーニングセクション（http://www.mysql.com/training/）を参照してください。イン

ターネットにアクセスできない場合は、MySQL AB のトレーニングスタッフ（<training@mysql.com>）に電子メールでお問い合わせください。 See 項1.3.1.2.「トレーニングと検定」。

MySQL 検定プログラムについては、http://www.mysql.com/certification/ を参照してください。 See 項1.3.1.2.「トレーニングと検定」。

コンサルティングに関心をお持ちの方は、Web サイトのコンサルティングセクション（http://www.mysql.com/consulting/）を参照してください。インターネットにアクセスできない場合は、MySQL AB のコンサルティングスタッフ（<consulting@mysql.com>）に電子メールでお問い合わせください。 See 項1.3.1.3.「コンサルティング」。

https://order.mysql.com/ では、商用ライセンスをオンラインで購入することができます。また、MySQL AB に FAX で購入申込書を送信する方法を参照することもできます。ライセンスの詳細については、http://www.mysql.com/products/pricing.html を参照してください。ライセンスに関する質問がある場合やハイボリュームライセンス契約の見積もりが必要な場合は、Web サイト（http://www.mysql.com/）にある問い合わせ用紙に記入するか、<licensing@mysql.com>（ライセンスに関する質問）または <sales@mysql.com>（販売見積もり）に電子メールをお送りください。 See 項1.4.3.「MySQL ライセンス」。

MySQL AB とのパートナ提携に関心をお持ちの企業は、<partner@mysql.com> に電子メールをお送りください。 See 項1.3.1.5.「パートナ提携」。

MySQL の商標ポリシーの詳細については、http://www.mysql.com/company/trademark.html を参照するか、<trademark@mysql.com> に電子メールをお送りください。 See 項1.4.4.「MySQL AB の口ゴと商標」。

採用セクション（http://www.mysql.com/company/jobs/）に掲載されている MySQL AB の職種に関心をお持ちの方は、<jobs@mysql.com> に電子メールをお送りください。履歴書は、添付ファイルとして送信するのではなく、電子メールメッセージの最後にプレーンテキスト形式で記載してください。

多数のユーザ間の一般的なディスカッションについては、該当するメーリングリストを参照してください。 See 項1.7.1.「MySQL メーリングリスト」。

エラー（多くの場合、バグと呼ばれます）の報告は、質問やコメントと同様に、通常の MySQL メーリングリストにお送りください。See 項1.7.1.1.「MySQL メーリングリスト」。MySQL サーバで重大なセキュリティバグを見つけた場合は、<security@mysql.com> に電子メールをお送りください。 See 項1.7.1.3.「バグまたは問題を報告する方法」。

公開可能なベンチマーク結果をお持ちの方は、<benchmarks@mysql.com> に電子メールでお問い合わせください。

このマニュアルへの追加や修正に関する提案については、マニュアルチーム（http://www.mysql.com/company/contact/）に電子メールをお送りください。

MySQL の Web サイト（http://www.mysql.com/）の運営や内容に関する質問やコメントについては、<webmaster@mysql.com> に電子メールをお送りください。

MySQL AB にはプライバシーポリシーがあります。http://www.mysql.com/company/privacy.html を参照してください。このポリシーに関する質問については、<privacy@mysql.com> に電子メールをお送りください。

その他の質問についてはすべて、<info@mysql.com> に電子メールをお送りください。

1.4. MySQL のサポートとライセンス

このセクションでは、MySQL のサポートおよびライセンスに関する取り決めについて説明します。

1.4.1. MySQL AB によって提供されるサポート

MySQL AB からのテクニカルサポートでは、MySQL データベースエンジンをコード化するソフトウェアエンジニアからお客様個々の問題に応じた回答が提供されます。

当社は、テクニカルサポートを広く、包括的にとらえます。お客様にとって重要な MySQL ソフトウェアに関するほとんどの問題は、当社にとっても重要です。ユーザは通常、さまざまなコマンドやユーティリティを使用する方法、パフォーマンスボトルネックを除去する方法、クラッシュしたシステムをリストアする方法、MySQL に対するオペレーティングシステムやネットワークの影響を理解する方法、バックアップおよびリカバリに関するベストプラクティスを設定する方法、API を利用する方法などについてのヘルプを必要としています。当社のサポートは、できる限りのお手伝いをすることを心がけていますが、MySQL サーバと独自のユーティリティのみを対象とし、MySQL サーバにアクセスするサードパーティ製品は対象としておりません。

さまざまなサポートオプションの詳細については、http://www.mysql.com/support/ を参照してください。ここでは、サポート契約をオンラインで申し込むこともできます。インターネットにアクセスできない場合は、販売スタッフ (<sales@mysql.com>) に電子メールでお問い合わせください。

テクニカルサポートは生命保険のようなものです。生命保険がなくても何年間も快適に暮らせますが、そのときがくれば、非常に重要になります。しかし、それから保険に入ったのでは遅すぎるのです。重要なアプリケーションに MySQL サーバを使用している場合、問題が突然発生したときに、自分だけですべてを解決するのは非常に時間がかかります。そのような場合、MySQL AB で採用している経験豊富な MySQL のトラブルシューティング担当者への直接のアクセスが必要になることがあります。

1.4.2. MySQL で使用されている著作権とライセンス

MySQL AB は、MySQL のソースコード、MySQL の口ゴと商標、およびこのマニュアルの著作権を所有しています。See 項1.3.「MySQL AB の概要」。MySQL ディストリビューションには、さまざまなライセンスが関連しています。

1. サーバ、mysqlclient ライブラリ、クライアント、および GNU readline ライブラリ内の MySQL 固有のソースはすべて、GNU 一般公衆利用許諾契約書の対象である。See 付録 H. GNU General Public License。このライセンスのテキストは、ディストリビューションのファイル COPYING にある。

2. GNU getopt ライブラリは、GNU 劣等一般公衆利用許諾契約書の対象である。http://www.fsf.org/licenses/ を参照。

3. ソース (regexp ライブラリ) の一部は、Berkeley スタイルの著作権の対象である。

4. MySQL の古いバージョン (3.22 以前) には、さらに厳密なライセンス (http://www.mysql.com/products/mypl.html) が適用される。個々のバージョンのマニュアルを参照。

5. MySQL のリファレンスマニュアルは現在、GPL スタイルのライセンス下では提供されていない。このマニュアルの使用には、以下の条件が適用される。

   - 他の形式への変換が可能である。ただし、実際の内容を変更したり、編集したりすることはできない。
   - 個人的な使用を目的とする場合、マニュアルを印刷することができる。
   - 印刷したマニュアルの販売や別の出版物でのマニュアルの使用など、その他の使用方法についてはすべて、MySQL AB からの書面による事前の合意が必要である。

   詳細や翻訳に関心をお持ちの方は、http://www.mysql.com/company/contact/ に電子メールをお送りください。

MySQL ライセンスの施行については、項1.4.3.「MySQL ライセンス」 を参照してください。項1.4.4.「MySQL AB の口ゴと商標」 も参照してください。

1.4.3. MySQL ライセンス

MySQL ソフトウェアは、最もよく知られていると思われるオープンソースライセンスである GNU 一般公衆利用許諾契約書 (GPL) に基づいてリリースされています。GPL ライセンスの正式な条件については、http://www.fsf.org/licenses/ を参照してください。http://www.fsf.org/licenses/gpl-faq.html および http://www.gnu.org/philosophy/enforcing-gpl.html も参照してください。

MySQL ソフトウェアは GPL に基づいてリリースされているので、多くの場合、無償で使用することができます。ただし、特定の用途については、https://order.mysql.com/ で MySQL AB から商用ライセンスを購入しなければならない場合があります。詳細については、http://www.mysql.com/products/licensing.html を参照してください。

MySQL の古いバージョン (3.22 以前) には、さらに厳密なライセンス (http://www.mysql.com/products/mypl.html) が適用されます。個々のバージョンのマニュアルを参照してください。

商用ライセンス、GPL、または古い MySQL ライセンスに基づいて MySQL ソフトウェアを使用しても、MySQL AB の商標を使用する権利が自動的に与えられるわけではないことに注意してください。 See 項1.4.4.「MySQL AB の口ゴと商標」。

1.4.3.1. 商用ライセンスに基づく MySQL ソフトウェアの使用

GPL ライセンスは、プログラムが GPL プログラムにリンクしている場合、作成された製品のすべての部分のソースコードもすべて GPL に基づいてリリースする必要があるという意味で、伝染性があります。この GPL 要件に従わない場合、ライセンス条件に違反することとなり、GPL プログラムを使用する権利をすべて失います。また、損害を受ける危険を冒すことにもなります。

以下のような場合、商用ライセンスが必要です。

- MySQL ソフトウェアの GPL コードとプログラムをリンクし、商用製品を作成するため、または追加した非 GPL コードを他の理由でクローズソースにするために、作成する製品に GPL に基づくライセンスを付与しない場合。商用ライセンスを購入すると、MySQL ソフトウェアを使用する際に、同じコードでも GPL が適用されなくなる。
- MySQL ソフトウェアのみを使用する非 GPL アプリケーションを提供し、MySQL ソフトウェアとともに出荷する場合。このタイプのソリューションは、ネットワーク経由で行われてもリンクしていると見なされる。
- GPL ライセンスの下で必要とされるソースコードの提供を行わずに、MySQL ソフトウェアのコピーを提供する場合。
- 形式上は商用ライセンスが必要でなくても、MySQL データベースの今後の開発をサポートする場合。MySQL AB からサポートを直接購入することは、MySQL ソフトウェアの開発に貢献するもう 1 つのよい方法である。また、ユーザにも直接メリットがある。 See 項1.4.1.「MySQL AB によって提供されるサポート」。

ライセンスを要求する場合、MySQL ソフトウェアの各インストールに 1 つずつ必要です。これは、1 台のマシン上の CPU の数には関係ありません。また、サーバに接続するクライアントの数に理論上の制限はありません。

商用ライセンスについては、当社の Web サイト (http://www.mysql.com/products/licensing.html) を参照してください。サポート契約については、http://www.mysql.com/support/ を参照してください。特殊なニーズがある場合やインターネットにアクセスできない場合は、販売スタッフ (<sales@mysql.com>) に電子メールでお問い合わせください。

1.4.3.2. GPL に基づく MySQL ソフトウェアの無償使用

GPL の条件に従っている場合、GPL の下で MySQL ソフトウェアを無償で使用することができます。GPL に関する一般的な質問に対する回答などの詳細については、Free Software Foundation の一般的な FAQ (http://www.fsf.org/licenses/gpl-faq.html) を参照してください。一般に、以下のような場合に GPL を使用します。

- 独自のアプリケーションと GPL に基づく MySQL のソースコードの両方を製品とともに提供する場合。
- 商用としてディストリビューションを販売する場合でも、機能に関して MySQL システムにリンクしていない、または MySQL システムに依存していない他のプログラムとバンドルされた MySQL のソースコードを提供する場合。これは、GPL ライセンスでは単に集約と呼ばれる。
- MySQL システムのいずれの部分も提供しない場合、無償で使用することができる。
- インターネットサービスプロバイダ (ISP) として、MySQL サーバを使用した Web ホスティングを顧客に提供する場合。当社は、MySQL サポートがある ISP を使用するように推奨している。これは、MySQL のインストールに関して発生しうる問題を解決するためのリソースが、実際にそれらの ISP にあるという信頼感が得られるためである。ISP に MySQL サーバの商用ライセンスがなくても、適切なパッチが適用されていることを確認できるように、顧客に少なくとも MySQL インストールのソースへの読み取りアクセス権を付与する必要がある。
- MySQL データベースソフトウェアを Web サーバとともに使用する場合 (提供する製品でない限り) 、商用ライセンスは不要である。MySQL サーバを使用する市販の Web サーバを実行する場合も、同様である。これは、MySQL システムのいずれの部分も提供しないためである。ただし、この場合、MySQL ソフトウェアによって企業がサポートされているので、MySQL サポートを購入していただくよう希望する。

MySQL データベースソフトウェアの使用に商用ライセンスが不要な場合でも、MySQL AB からサポートを購入することをお勧めします。これにより、MySQL の開発に貢献することになるとともに、お客様自身も直接メリットが得られます。See 項1.4.1.「MySQL AB によって提供されるサポート」。

MySQL データベースソフトウェアの使用によって利益を得るような商用コンテキストで MySQL データベースソフトウェアを使用する場合は、いずれかのレベルのサポートを購入することで MySQL ソフトウェアの開発のお手伝いをお願いしています。MySQL データベースがお客様のビジネスに役立っているのなら、今度はお客様が MySQL AB に手を貸してください (そうでないと、サポートに関する質問をする場合、多くの労力を要した成果を無償で使用しているだけでなく、当社に無償でサポートを提供するように依頼していることにもなります) 。

1.4.4. MySQL AB のロゴと商標

Web サイト、書籍、またはボックス製品に MySQL AB のイルカの口ゴを掲載したいという MySQL データベースユーザが多数いらっしゃいます。当社はこれを歓迎し、推奨しておりますが、MySQL という言葉と MySQL のイルカの口ゴは MySQL AB の商標であり、http://www.mysql.com/company/trademark.html に記載されている当社の商標ポリシーに従って使用する必要があることに注意してください。

1.4.4.1. MySQL のオリジナルロゴ

MySQL のイルカの口ゴは、2001 年にフィンランドの広告代理店 Priority によってデザインされたものです。頭がよく、敏捷でスリムな動物なので、データの大海原を楽々と進むということから、MySQL データベースに適したシンボルとしてイルカが選ばれました。また、私たちもイルカが好きでした。

MySQL のオリジナルロゴを使用できるのは、MySQL AB の代表者と、書面による合意によって使用が許可されたユーザのみです。

1.4.4.2. 書面による許可なしに使用できる MySQL のロゴ

当社は、Web サイト (http://www.mysql.com/press/logos.html) からダウンロードし、MySQL AB からの書面による許可なしにサードパーティの Web サイトで使用できる特殊な条件付き使用ロゴをデザインしております。これらのロゴの使用にはまったく制限はありませんが、名前のとおり、当社の商標ポリシーが適用されます。商標ポリシーも、Web サイトで参照することができます。ロゴを使用する場合は、商標ポリシーをよくお読みください。基本的な条件は以下のとおりです。

- http://www.mysql.com/ サイトに表示されているように、必要なロゴを使用する。必要に応じてロゴのサイズを変更することはできるが、色やデザインを変更したり、グラフィックを変更したりすることはできない。
- MySQL の商標を掲載するサイトの作成者および所有者が MySQL AB ではなく、ユーザ自身であることを明確にする。
- MySQL AB や MySQL AB の商標の価値にマイナスとなるような使い方をしない。当社は、MySQL AB の商標の使用権を無効にする権利を留保している。
- Web サイトで商標を使用する場合、クリックすると、直接 http://www.mysql.com/ にジャンプするようにする。
- アプリケーションで GPL に基づく MySQL データベースを使用する場合、アプリケーションはオープンソースであると同時に、MySQL サーバに接続可能でなければならない。

ニーズに合わせた特別な調整に関する質問については、<trademark@mysql.com> に電子メールでお問い合わせください。

1.4.4.3. MySQL のロゴの使用に書面による許可が必要な場合

以下の場合、MySQL のロゴを使用するには、MySQL AB からの書面による許可が事前に必要となります。

- Web サイト以外の場所に MySQL AB のロゴを掲載する場合
- 前述の条件付き使用ロゴ以外の MySQL AB のロゴを Web サイトやその他の場所に掲載する場合

法律上および商業上の理由から、当社は、製品や書籍などにおける MySQL の商標の使用を監視します。通常、商用製品に MySQL AB のロゴを掲載する場合には料金をいただきます。収益の一部が MySQL データベースの今後の開発資金として使用されるのが適切だと考えるためです。

1.4.4.4. MySQL AB のパートナシップロゴ

MySQL のパートナシップロゴを使用できるのは、MySQL AB と書面によるパートナシップ契約を結んでいる企業および個人のみです。パートナシップには、MySQL の講師またはコンサルタントとしての認定も含まれます。詳細については、項1.3.1.5.「パートナ提携」 を参照してください。

1.4.4.5. 印刷テキストまたはプレゼンテーションにおける MySQL という言葉の使用

MySQL AB は MySQL データベースの参照を歓迎しますが、MySQL という言葉は MySQL AB の商標であることに注意し

てください。そのため、テキスト内で MySQL という言葉を最初に使用する箇所や最も目立つように使用する箇所には商標記号（ TM ）を添え、必要に応じて、MySQL が MySQL AB の商標であることを記載する必要があります。詳細については、商標ポリシー（ http://www.mysql.com/company/trademark.html ）を参照してください。

1.4.4.6. 企業名および製品名における MySQL という言葉の使用

製品名、企業名、またはインターネットドメイン名に MySQL という言葉を使用する場合は、MySQL AB からの書面による許可が必要です。

1.5. MySQL の開発ロードマップ

このセクションでは、MySQL 4.0、4.1、5.0、および 5.1 で実装または計画されている主な機能を含む、MySQL の開発ロードマップのスナップショットを提供します。バージョンシリーズごとの情報については、以下のセクションを参照してください。次の表は、最も要望があった機能の一部についての計画をまとめたものです。

機能	MySQL バージョン
UNION	4.0
サブクエリ	4.1
R-tree	4.1（ MyISAM テーブル用 ）
ストアドプロシージャ	5.0
ビュー	5.0 または 5.1
カーソル	5.0
外部キー	5.1（ InnoDBについては 3.23 で実装済み ）
トリガ	5.1
Full OUTER JOIN	5.1
制約	5.1

1.5.1. MySQL 4.0 の概要

ユーザが長い間待ち望んでいた MySQL サーバ 4.0 が運用ステータスで使用可能になりました。

MySQL 4.0 は、http://www.mysql.com/ および当社のミラーからダウンロードすることができます。MySQL 4.0 は多数のユーザによってテストされ、多数の大規模サイトで運用されています。

MySQL サーバ 4.0 の主な新機能は、当社の既存のビジネスおよびコミュニティユーザ向けに設計されており、ミッションクリティカルで負荷が高いデータベースシステムに対応したソリューションとして MySQL データベースソフトウェアを拡張します。組み込みデータベースのユーザを対象とした新機能もあります。

MySQL 4.0 は、2003 年 3 月にリリースされたバージョン 4.0.12 で、運用について安定していると宣言されました。そのため、今後、4.0 シリーズについてはバグ修正のみが行われ、古い 3.23 シリーズについては重大なバグ修正のみが行われることになります。 See 項2.5.2.「バージョン 3.23 から 4.0 へのアップグレード」。

すでに使用可能な MySQL 4.1（ アルファバージョン ）には、MySQL ソフトウェアの新機能が追加されます。 See 項 1.5.2.「MySQL 4.1 の概要」。

1.5.1.1. MySQL 4.0 で使用可能な機能

- 速度の向上
  - MySQL 4.0 にはクエリキャッシュがあり、反復クエリを使用するアプリケーションの速度を大幅に向上させることができる。 See 項6.9.「MySQL クエリキャッシュ」。
  - バージョン 4.0 では、一括 INSERT ステートメント、パックされたインデックスの検索、FULLTEXT インデックスの作成、および COUNT(DISTINCT) など、いくつかの領域で MySQL サーバの速度がさらに向上している。
- 組み込み MySQL サーバの導入
  - 新しい組み込みサーバライブラリを使用すると、スタンドアロンアプリケーションおよび組み込みアプリケーションを容易に作成することができる。組み込みサーバは、クライアント/サーバ環境で MySQL を使用する 1 つの方法である。 See 項1.5.1.2.「組み込み MySQL サーバ」。
- 標準としての InnoDB ストレージエンジン
  - InnoDB ストレージエンジンが MySQL サーバの標準機能として提供されるようになった。つまり、ACID トランザクションの完全サポート、連鎖 UPDATE および DELETE を使用した外部キー、および行レベルのロックが標準機能となっている。 See 項7.5.「InnoDB テーブル」。
- 新機能
  - MySQL サーバ 4.0 の拡張された FULLTEXT 検索機能によって、大きいテキストの FULLTEXT インデックスが、バイナリと自然言語の両方の検索ロジックで作成可能になった。最小ワード長をカスタマイズし、任意の自然言語で独自のストップワード一覧を定義することができるため、MySQL サーバ上に新しいアプリケーションセットを構築することができる。 See 項6.8.「MySQL 全文検索」。
- 標準への準拠、移植性、および移行
  - 他のデータベースシステムから MySQL サーバへの移行を容易にする機能として、（Oracle にある）TRUNCATE TABLE がある。
  - MySQL サーバで UNION ステートメントがサポートされるようになった。これは、多数のユーザが長い間待ち望んでいた標準の SQL 機能である。
  - MySQL が Novell NetWare 6.0 プラットフォーム上でネイティブに稼動するようになった。 See 項2.6.8.「Novell NetWare の注意事項」。
- 国際化
  - ドイツ、オーストリア、およびスイスのユーザ向けに MySQL で新しいキャラクタセット latin1_de がサポートされるようになり、ドイツ語のソート順では、ウムラウトを含む単語がドイツの電話帳と同じ順序でソートされる。
- 使いやすさの向上

  新しいユーザ向けの機能を実装するプロセスでも、既存のユーザの忠実なコミュニティからの要望を忘れることはありません。

  - ほとんどの mysqld パラメータ（スタートアップオプション）が、サーバを停止しなくても設定できるようになっ

た。これは、データベース管理者（DBA）にとって便利な機能である。 See 項5.5.6.「SET 構文」。

- 複数テーブルの DELETE および UPDATE ステートメントが追加された。
- MyISAM ストレージエンジンにテーブルレベルの（以前のようにデータベースレベルのみではない）シンボリックリンクのサポートが追加された。また、Windows では、データベースレベルにおけるシンボリックリンクの処理がデフォルトで有効である。
- 新しく追加された関数 SQL_CALC_FOUND_ROWS と FOUND_ROWS() を使用すると、LIMIT 節を含む SELECT クエリがその節を含めない場合に返すレコードの数を調べることができる。

このマニュアルのニュースセクションに、機能の詳細な一覧があります。 See 項D.3.「Changes in release 4.0.x (Production)」。

1.5.1.2. 組み込み MySQL サーバ

libmysqld によって、MySQL サーバはアプリケーションの大規模展開に対応します。組み込み MySQL サーバライブラリを使用すると、さまざまなアプリケーションや電子機器に MySQL サーバを組み込むことができます。それらのアプリケーションや電子機器のエンドユーザには、実際はそれらの基になっているデータベースがあるということはわかりません。組み込み MySQL サーバは、インターネット機器、パブリックキオスク、ハードウェアとソフトウェアが組み合わさったターンキー装置、高性能インターネットサーバ、CD-ROMで配布されている独立言語型データベースなどの背後での使用に最適です。

libmysqld の多くのユーザは、MySQL のデュアルライセンスの恩恵を受けます。GPL に拘束されたくないユーザは、商用ライセンスに基づいてソフトウェアを使用することもできます。組み込み MySQL ライブラリでは通常のクライアントライブラリと同じインタフェースが使用されているので、使いやすく、便利です。 See 項11.1.15.「組み込み MySQL サーバライブラリ libmysqld」。

1.5.2. MySQL 4.1 の概要

MySQL サーバ 4.0 は、サブクエリや Unicode などの新機能（バージョン 4.1 で実装）および SQL-99 ストアドプロシージャの開発（バージョン 5.0 で実装）の基礎となっています。これらの機能は、当社のお客様からの要望が最も高いものです。

これらの追加機能によって、MySQL データベースサーバの批評家は、MySQL データベース管理システムの問題を指摘する際に、これまで以上に想像力が必要になります。その安定性、速度、使いやすさですでによく知られている MySQL サーバは、非常に多くを望む購入者の条件リストを満たすことができます。

1.5.2.1. MySQL 4.1 で使用可能な機能

このセクションに記載されている機能は、MySQL 4.1 に実装されています。MySQL 4.1 には、他にも計画されている機能があります。 See 項1.6.1.「4.1 で計画されている新機能」。

ストアドプロシージャなど、コード化されるほとんどの新機能は MySQL 5.0 で使用できるようになります。 See 項1.6.2.「5.0 で計画されている新機能」。

- サブクエリと派生テーブルのサポート

  - サブクエリとは、別のステートメント内にネストされた SELECT ステートメントである。派生テーブル（名前のないビュー）とは、別のステートメントの FROM 節内のサブクエリである。 See 項6.4.2.「サブクエリ構文」。

- 速度の向上

  - 準備されたステートメントとパラメータのバインディングによる、より速いバイナリプロトコル。 See 項11.1.4.「C API のプリペアドステートメント」。

  - HEAP テーブルについて B-TREE インデックスの作成がサポートされるようになり、正確でない検索の応答時間が大幅に短縮される。

- 新機能

  - CREATE TABLE table_name2 LIKE table_name1 を使用することで、既存のテーブルとまったく同じ構造の新しいテーブルを 1 つのステートメントで作成することができる。

  - OpenGIS 空間データ型（地理データ）のサポート。 See 章 10. MySQL における空間情報の機能。

  - SSL 接続を介したレプリケーションを実行することができる。

- 標準への準拠、移植性、および移行

  - 新しいクライアント/サーバプロトコルにより、1 つの結果だけでなく、複数の警告をクライアントに渡す機能が追加される。これにより、データの一括ロードなどのジョブが追跡しやすくなる。SHOW WARNINGS は、最後のコマンドの警告を表示する。 See 項4.6.8.9.「SHOW WARNINGS | ERRORS」。

- 国際化

  - アプリケーションで各国語を使用するユーザベースの拡大に対応するために、MySQL ソフトウェアでは、utf8 および ucs2 キャラクタセットによる幅広い Unicode サポートが提供されるようになった。

  - カラム、テーブル、およびデータベースごとにキャラクタセットを定義できるようになった。これにより、特に多言語 Web サイトを対象としたアプリケーション設計で高度な柔軟性が実現する。

  - 改善されたこのキャラクタセットサポートについては、章 9. 各国キャラクタセットと Unicode を参照。

- 使いやすさの向上

  - よく寄せられる要望に応じて、SQL ステートメントに関するヘルプ情報を取得するために使用できるサーバベースの HELP コマンドを追加した。サーバ側でこの情報を取得できるメリットは、情報が常にその特定のサーババージョンに該当する点である。この情報は SQL ステートメントを発行することで参照できるので、任意のクライアントをそれにアクセスするように記述することができる。たとえば、mysql コマンドラインクライアントはこの機能を使用できるように変更されている。

  - 新しいクライアント/サーバプロトコルでは、1 回の呼び出しで複数のステートメントを発行することができる。See 項11.1.8.「C API における複数クエリの実行の取り扱い」。

  - 新しいクライアント/サーバプロトコルでは、複数の結果セットを返すこともできる。これは、たとえば、複数のステートメントを送信した結果として行われることがある（前の項目を参照）。

  - 新しい INSERT ... ON DUPLICATE KEY UPDATE ... 構文が実装された。この構文を使用すると、INSERT によって PRIMARY または UNIQUE キー（インデックス）で重複が発生した場合に既存のレコードを更新することができる

。 See 項6.4.3.「INSERT 構文」。

- 新しい集約関数 GROUP_CONCAT() を実装した。この関数によって、グループ化されたレコードのカラムを 1 つの結果文字列に連結する非常に便利な機能が追加される。 See 項6.3.7.「GROUP BY 節で使用する関数と修飾子」。

このマニュアルのニュースセクションに、機能の詳細な一覧があります。 See 項D.2.「Changes in release 4.1.x (Alpha)」。

1.5.2.2. 段階的ロールアウト

MySQL 4.1 には、新しい機能が追加されます。アルファバージョンは、すでにダウンロードすることができます。 See 項1.5.2.3.「即時の開発用途への対応」。

バージョン 4.1 に追加される機能は、ほとんど確定しています。すでに、バージョン 5.0 に向けて新たな開発が行われています。MySQL 4.1 はアルファ (さらに新機能が追加/変更される可能性がある段階) 、ベータ (機能が固定され、バグ修正のみが行われる段階) 、ガンマ (製品版としてのリリースを数週間後に控えた段階) というステップをたどります。このプロセスの最後に、MySQL 4.1 は新しい製品版となります。

1.5.2.3. 即時の開発用途への対応

現在、MySQL 4.1 はアルファ段階にあり、http://www.mysql.com/downloads/mysql-4.1.html でバイナリリリースをダウンロードすることができます。すべてのバイナリリリースは、テストするプラットフォームでエラーが発生することなく、一連の広範なテストに合格しています。 See 項D.2.「Changes in release 4.1.x (Alpha)」。

MySQL 4.1 の最新の開発ソースの使用を希望するユーザのために、4.1 BitKeeper リポジトリを公開しています。 See 項2.3.4.「開発ソースツリーからのインストール」。

1.5.3. MySQL 5.0、次期開発リリース

MySQL の新たな開発は、ストアドプロシージャなどの新機能を特徴とする 5.0 リリースに焦点が置かれています。 See 項1.6.2.「5.0 で計画されている新機能」。

MySQL 開発の最前線の参照を希望するユーザのために、MySQL バージョン 5.0 の BitKeeper リポジトリを公開しています。 See 項2.3.4.「開発ソースツリーからのインストール」。

1.6. MySQL の今後 (TODO)

このセクションでは、MySQL サーバに実装予定の機能の概要を説明します。一覧はバージョンごとに分かれており、項目はほぼ、実装される順序に並んでいます。

注意: 特定の機能を至急必要とする企業レベルのユーザは、支援オプションについて <sales@mysql.com> にお問い合わせください。スポンサー企業からの特定の目的に絞った資金提供によって、その目的のために特別なリソースを割り当てることができます。過去に資金提供を受けた機能の例として、レプリケーションがあります。

1.6.1. 4.1 で計画されている新機能

以下の機能は、MySQL 4.1 にまだ実装はされていませんが、MySQL 4.1 がベータ段階に移行する前に実装される予定です

。MySQL 4.1 にすでに実装されている機能の一覧については、項1.5.2.1.「MySQL 4.1 で使用可能な機能」 を参照してください。

- 安定した OpenSSL サポート (MySQL 4.0 でも基本的にはサポートされているが、100% テストされているわけではない) 。
- 準備されたステートメントのより詳細なテスト。
- 1 つのテーブルで使用する複数のキャラクタセットのより詳細なテスト。

1.6.2. 5.0 で計画されている新機能

MySQL 5.0 には、以下の機能が組み込まれる予定です。多数の開発者がさまざまなプロジェクトで作業しているので、他にも多くの機能があることに注意してください。これらの機能の一部は、MySQL 4.1 に追加される可能性もあります。MySQL 4.1 にすでに実装されている機能の一覧については、項1.5.2.1.「MySQL 4.1 で使用可能な機能」 を参照してください。

MySQL 開発の最前線の参照を希望するユーザのために、MySQL バージョン 5.0 の BitKeeper リポジトリを公開しています。 See 項2.3.4.「開発ソースツリーからのインストール」。

- ストアドプロシージャ
  - ストアドプロシージャは、現在実装中である。この試みは SQL-99 に基づくもので、Oracle PL/SQL に類似した (ただし、同じではない) 基本構文を使用する。また、外部言語でフックする SQL-99 フレームワーク、および (可能な場合) PL/SQL や T-SQL などとの互換性も実装する予定である。
- 新機能
  - 基本的なカーソルサポート。
  - MyISAM テーブルでインデックスが R-TREE インデックスとして作成されるように明示的に指定する機能。4.1 では、地理データ (GIS データ型) について R-TREE インデックスが内部で使用されるが、必要に応じて作成することはできない。
  - HEAP テーブルの可変長レコード。
- 標準への準拠、移植性、および移行
  - VARCHAR の完全なサポート (255 を超えるカラム長、後続のスペースを削除しない) の追加 (MyISAM ストレージエンジンではすでにこれがサポートされているが、ユーザレベルではまだ使用できない) 。
- 速度の向上
  - SHOW COLUMNS FROM table_name (カラム名の表示を可能にするために mysql クライアントによって使用される) によって、テーブルではなく定義ファイルのみが開くようにする。これによって、使用されるメモリが少なくて済むとともに、速度がはるかに向上する。
  - MyISAM テーブルの DELETE でレコードキャッシュを使用できるようにする。そのためには、.MYD ファイルを更新する際にスレッドレコードキャッシュを更新する必要がある。

  - メモリ内 (HEAP) テーブルの改良。
    - 可変長レコード
    - より迅速なレコード処理 (コピーの削減)
- 国際化
  - SET CHARACTER SET を使用する場合、文字列だけでなく、クエリ全体が一度に変換されるようにする必要がある。これによって、ユーザはデータベース名、テーブル名、およびカラム名で変換された文字を使用できるようになる。
- 使いやすさの向上
  - アクティブな MERGE テーブルで使用されているテーブルに対して RENAME TABLE を発行した場合にテーブルが破損する可能性がある問題の解決。

1.6.3. 5.1 で計画されている新機能

- 新機能
  - すべてのテーブル型での FOREIGN KEY のサポート。
  - カラムレベルの制約。
  - 緊急時のレプリケーション。
  - パフォーマンスへの影響が非常に少ないオンラインバックアップ。オンラインバックアップによって、マスタを停止せずに新しいレプリケーションスレーブを簡単に追加できるようになる。
- 速度の向上
  - テキストベースの新しいテーブル定義ファイル形式 (.frm ファイル) とテーブル定義に使用されるテーブルキャッシュ。これによって、テーブル構造のクエリが高速になるとともに、より効率的な外部キーサポートが実現する。
  - 1 ビットを使用するように BIT 型を最適化する (現在、BIT は 1 バイトを必要とし、TINYINT のシノニムとして扱われている)。
- 使いやすさの向上
  - クライアント/サーバプロトコルに、長時間実行されているコマンドの進捗状況を参照するオプションを追加する。
  - RENAME DATABASE を実装する。すべてのストレージエンジンについてこれを安全にするには、以下のように動作する必要がある。
    - 新しいデータベースを作成する。

    - RENAME コマンドを使用して行われるように、すべてのテーブルについて、別のデータベースへのテーブルの名前変更を行う。
    - 元のデータベースを破棄する。
- 内部ファイルインタフェースの変更。これによって、すべてのファイル処理がはるかに一般的になるとともに、RAID のように簡単に拡張子を追加できるようになる。

1.6.4. 近い将来に計画されている新機能

- 新機能
    - ツリー形式 (階層状) の構造を検索する Oracle のような CONNECT BY PRIOR ...。
    - 不足しているすべての SQL-92 および ODBC 3.0 型を追加する。
    - SUM(DISTINCT) を追加する。
    - テーブルの読み取りがロックされている場合、テーブルの最後で同時挿入を行う INSERT SQL_CONCURRENT および mysqld --concurrent-insert。
    - UPDATE TABLE foo SET @a=a+b,a=@a, b=@a+c のように、UPDATE ステートメントでの変数の更新を有効にする。
    - ユーザ変数が更新される場合、SELECT id, @a:=COUNT(*), SUM(sum_col)/@a FROM table_name GROUP BY id のように GROUP BY とともに使用できるように変更する。
    - TIMESTAMP および AUTO_INCREMENT フィールドを更新しないように LOAD DATA INFILE に IMAGE オプションを追加する。
    - 次のように動作する LOAD DATA INFILE ... UPDATE 構文を追加する。
        - 主キーがあるテーブルについて、入力レコードに主キー値が含まれている場合、その主キー値と一致する既存のレコードはその他の入力カラムから更新される。ただし、入力レコードにないカラムに対応するカラムはそのままになる。
        - 主キーがあるテーブルについて、入力レコードに主キー値が含まれていない場合やキーの一部がない場合、レコードは LOAD DATA INFILE ... REPLACE INTO として処理される。
    - LOAD DATA INFILE で次のような構文が認識されるようにする。

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

    これを使用すると、テキストファイル内の余分なカラムをスキップしたり、読み取ったデータの式に基づいてカラムを更新したりすることができる。

- SET 型のカラムを使用する新しい関数。
  - ADD_TO_SET(value,set)
  - REMOVE_FROM_SET(value,set)
- クエリの途中で mysql を停止する場合、別の接続を開き、実行中の元のクエリを終了する必要がある。または、サーバでこれが検出されるようにする必要がある。
- システムテーブルとして使用できるように、テーブル情報のストレージエンジンインタフェースを追加する。これは、すべてのテーブルに関する情報を要求した場合には多少遅くなるが、非常に柔軟である。基本的なテーブル情報を対象とした SHOW INFO FROM tbl_name が実装される。
- SELECT a FROM table_name1 LEFT JOIN table_name2 USING (a) を有効にする。この場合、a は、table_name1 テーブルに由来すると想定される。
- UPDATE ステートメントの DELETE および REPLACE オプション（これによって、更新時に重複キーエラーが発生した場合、レコードが削除されるようになる）。
- 秒の小数部を格納するように DATETIME の書式を変更する。
- 現在のライブラリの代わりに、新しい GNU regexp ライブラリを使用できるようにする（新しいライブラリは、現在のライブラリよりはるかに高速である）。

- 標準への準拠、移植性、および移行
  - カラムに自動 DEFAULT 値が追加されないようにする。DEFAULT がないカラムについては、INSERT ステートメントに値がない場合、エラーを生成する。
  - ANY()、EVERY()、および SOME() グループ関数を追加する。標準の SQL では、これらはブール型のカラムのみで動作するが、0 値を FALSE、0 以外の値を TRUE として処理することで、あらゆるカラム/式で動作するように拡張することができる。
  - MAX(column) の型をカラム型と同じになるように固定する。

```
mysql> CREATE TABLE t1 (a DATE);
mysql> INSERT INTO t1 VALUES (NOW());
mysql> CREATE TABLE t2 SELECT MAX(a) FROM t1;
mysql> SHOW COLUMNS FROM t2;
```

- 速度の向上
  - 定義された数を超えるスレッドで同時に MyISAM リカバリが実行されないようにする。
  - 必要に応じて同時挿入を使用するように INSERT ... SELECT を変更する。
  - しばらく使用されていない場合、遅延キーが設定されているテーブルのキーページを定期的にフラッシュするオプションを追加する。
  - キー部分での結合を有効にする（最適化の問題）。
  - Windows での pread()/pwrite() のシミュレーションを追加して、同時挿入を有効にする。

    - 最も頻繁にヒットするテーブル、複数テーブルの結合が実行される頻度などに関する情報を解析できるログファイルアナライザを追加する。これは、はるかに効率的なクエリを実行するように最適化することができる領域やテーブル設計を特定する上で便利である。
- 国際化
- 使いやすさの向上
    - SELECT MIN(column) ... GROUP BY の実行時に、元のフィールド型を返す。
    - long_query_time をマイクロ秒単位で詳細に指定できるようにする。
    - PACK または COMPRESS 操作を実行できるように、myisampack コードをサーバにリンクする。
    - インデックスファイルがいっぱいになった場合にリカバリできるように、INSERT/DELETE/UPDATE 時にテンポラリキーバッファキャッシュを追加する。
    - 別のディスクにシンボリックリンクされているテーブルで ALTER TABLE を実行する場合、そのディスクにテンポラリテーブルを作成する。
    - タイムゾーンが異なる日付の処理を容易にするために、タイムゾーン情報を適切に処理する DATE/DATETIME 型を実装する。
    - スレッドを使用せずにすべてのライブラリ（MyISAM など）をコンパイルできるように、configure を固定する。
    - LIMIT @a,@b のように、SQL 変数を LIMIT の引数として使用できるようにする。
    - mysql から Web ブラウザへの自動出力。
    - LOCK DATABASES（およびさまざまなオプション）。
    - SHOW STATUS によって表されるさまざまな数値。レコードの読み取りおよび更新。1 つのテーブルに対する SELECT および結合を使用した SELECT。 SELECT されたテーブルの平均数。ORDER BY および GROUP BY クエリの数。
    - mysqladmin copy database new-database。このためには、COPY 操作を mysqld に追加する必要がある。
    - プロセス一覧の出力にクエリ/スレッドの数が示されるようにする。
    - ホスト名キャッシュに関する情報を出力するための SHOW HOSTS。
    - 計算されたカラムについて、空白文字列から NULL にテーブル名を変更する。
    - SELECT COUNT(*)*(id+0) FROM table_name GROUP BY id の場合、数字 -> 文字列 -> 数字という変換を避けるために、数値に対して Item_copy_string を使用しない。
    - INSERT DELAYED を実行するクライアントが ALTER TABLE によって停止されないように変更する。
    - UPDATE 節でカラムが参照されている場合、更新が開始される前の古い値がカラムに含まれるように固定する。
- 新しいオペレーティングシステム
    - LynxOS への MySQL クライアントの移植。

1.6.5. 中期的な将来に計画されている新機能

- 関数 get_changed_tables(timeout,table1,table2,...) を実装する。
- 可能な場合、テーブルの読み取りで memmap が使用されるように変更する。現時点では、圧縮されたテーブルのみで memmap が使用されている。
- 自動タイムスタンプコードをより適切なものにする。SET TIMESTAMP=#; を使用して更新ログにタイムスタンプを追加する。
- 速度を向上させるために、いくつかの場所で読み取り/書き込み相互排他ロック(mutex)を使用する。
- シンプルなビュー（完全な機能装備まで段階的に実装）。 See 項1.8.4.6.「ビュー」。
- テーブル、テンポラリテーブル、またはテンポラリファイルでエラー 23（開いているファイルが十分でない）が発生した場合、一部のテーブルを自動的に閉じる。
- 定数の伝搬を改良する。式で col_name=n が見つかると、定数 n の場合、式内の他の col_name を n に置換する。現時点では、単純なケースのみでこれが行われている。
- 可能な場合、計算式を使用してすべての定数式を変更する。
- キー = 式を最適化する。現時点では、キー = フィールドまたはキー = 定数のみが最適化されている。
- より適切なコード化のためにコピー関数の一部を結合する。
- サイズを削減し、より適切なエラーメッセージを取得するように、sql_yacc.yy をインラインパーサに変更する（5 日）。
- 関数内のさまざまな数の引数について 1 つのルールのみが使用されるようにパーサを変更する。
- 命令部で完全な計算名を使用する（ACCESS97 に対応）。
- MINUS、INTERSECT、および FULL OUTER JOIN（現時点では、UNION（4.0）と LEFT|RIGHT OUTER JOIN がサポートされている）。
- クエリに時間制限を設定するために、SQL_OPTION MAX_SELECT_TIME=# を使用できるようにする。
- 更新ログからデータベースへの書き込みを可能にする。
- 結果セットの最後からデータを取得できるように LIMIT を拡張する。
- クライアントの接続/読み取り/書き込み機能に関する警告。
- mysqld_safe に対する変更には注意が必要である。Debian が準拠しようとしている FSSTND に従って、PID ファイルは /var/run/<progname>.pid に、ログファイルは /var/log に格納される必要がある。"pidfile" および "log" の最初の宣言に "DATADIR" を挿入すると、これらのファイルの場所を 1 つのステートメントで変更することができるので、便利である。
- クライアントがログを要求できるようにする。

- gzip によって圧縮されたファイルをステートメントで読み込めるように、LOAD DATA INFILE に zlib() の使用を追加する。
- BLOB 型のカラムのソートおよびグループ化を固定する（現時点では、部分的に解決されている）。
- スレッドをカウントする際にセマフォを使用するように変更する。まず、MIT-pthreads のセマフォライブラリを実装する必要がある。
- かっこを含む JOIN の完全サポートを追加する。
- シングルスレッド/接続の代わりに、スレッドのプールを利用してクエリを処理する。
- GET_LOCK() で複数のロックを取得できるようにする。これを実行した場合、この変更によって発生する可能性があるデッドロックも処理する必要がある。

時間は、実際の時間ではなく、作業量に基づいて記されています。

1.6.6. 計画されていない新機能

SQL-92/SQL-99 への完全な準拠を目標としています。実装を計画していない機能はありません。

1.7. MySQL の情報源

1.7.1. MySQL メーリングリスト

このセクションでは MySQL メーリングリストの概要を説明するとともに、リストの使用方法に関するガイドラインを提供します。メーリングリストを購読すると、リストへのすべての投稿が電子メールメッセージとして送信されます。また、自分の質問や回答をリストに送信することもできます。

1.7.1.1. MySQL メーリングリスト

このセクションに記載されているメーリングリストを購読する場合または購読を中止する場合は、http://lists.mysql.com/ にアクセスしてください。購読または購読中止に関するメッセージはいずれのメーリングリストにも送信しないでください。送信した場合、そのメッセージが多数の他のユーザに自動的に配信されてしまいます。

ローカルサイトに MySQL メーリングリストの購読者が多数いる場合、lists.mysql.com からサイトに送信されたメッセージがローカルリストに配信されるように、ローカルのメーリングリストが作成されている場合があります。このような場合、ローカルの MySQL リストへの追加またはリストからの削除については、システム管理者にお問い合わせください。

メールプログラム内の個別のメールボックスにメーリングリストが送信されるようにするには、メッセージヘッダに基づいてフィルタを設定してください。List-ID: または Delivered-To: ヘッダを使用して、リストのメッセージを識別することができます。

MySQL メーリングリストは以下のとおりです。

- announce

  このリストは、新しいバージョンの MySQL および関連するプログラムの通知に使用される。これは、すべての MySQL ユーザが購読する必要がある、ボリュームの小さいリストである。

- mysql

  これは、MySQL に関する一般的なディスカッションに使用されるメインのリストである。トピックによっては、より専門的なリストでディスカッションを行ったほうがよいものもある。適切でないリストに投稿すると、回答が得られないことがある。

- mysql-digest

  これは、ダイジェスト形式の mysql リストである。このリストを購読すると、全リストのメッセージを受け取ることになる。これは、1 日 1 回大きなメールメッセージとして送信される。

- bugs

  これは、MySQL の最新リリース以降に報告された問題を常に把握しておきたいユーザや、バグの検出と修正のプロセスに積極的に参加したいユーザにとって興味深いリストである。 See 項1.7.1.3.「バグまたは問題を報告する方法」。

- bugs-digest

  これは、ダイジェスト形式の bugs リストである。

- internals

  このリストは、MySQL コードに携わる人を対象としている。また、MySQL の開発およびポストパッチに関するディスカッションを行うフォーラムでもある。

- internals-digest

  これは、ダイジェスト形式の internals リストである。

- mysqldoc

  このリストは、MySQL AB の従業員、翻訳者、およびその他のコミュニティメンバなど、MySQL のマニュアルに携わる人を対象としている。

- mysqldoc-digest

  これは、ダイジェスト形式の mysqldoc リストである。

- benchmarks

  このリストは、パフォーマンスに関する問題に関心がある人を対象としている。データベースのパフォーマンス (MySQL に限らない) に関するディスカッションが中心だが、カーネル、ファイルシステム、ディスクシステムのパフォーマンスなど、より広範なカテゴリも含まれる。

- benchmarks-digest

  これは、ダイジェスト形式の benchmarks リストである。

- packagers

  このリストは、MySQL のパッケージおよびディストリビューションに関するディスカッションに使用される。これは、MySQL のパッケージに関するアイデアや、サポートされているすべてのプラットフォームおよびオペレーティングシステムでできる限り同じような MySQL の外観を確保するためのアイデアを交換するためにディストリビューション

管理者によって使用されるフォーラムである。

- packagers-digest

  これは、ダイジェスト形式の packagers リストである。

- java

  このリストは、MySQL サーバおよび Java に関するディスカッションに使用される。MySQL Connector/J を含む、JDBC ドライバに関するディスカッションに使用されることがほとんどである。

- java-digest

  これは、ダイジェスト形式の java リストである。

- win32

  このリストは、Windows 9x/Me/NT/2000/XP などの Microsoft オペレーティングシステムで実行される MySQL ソフトウェアに関するすべてのトピックに使用される。

- win32-digest

  これは、ダイジェスト形式の win32 リストである。

- myodbc

  このリストは、ODBC を使用した MySQL サーバへの接続に関するすべてのトピックに使用される。

- myodbc-digest

  これは、ダイジェスト形式の myodbc リストである。

- mysqlcc

  このリストは、MySQL Control Center グラフィカルクライアントに関するすべてのトピックに使用される。

- mysqlcc-digest

  これは、ダイジェスト形式の mysqlcc リストである。

- plusplus

  このリストは、MySQL への C++ API を使用したプログラミングに関するすべてのトピックに使用される。

- plusplus-digest

  これは、ダイジェスト形式の plusplus リストである。

- msql-mysql-modules

  このリストは、現在 DBD-mysql と呼ばれている msql-mysql-modules を使用した MySQL の Perl サポートに関するすべてのトピックに使用される。

- msql-mysql-modules-digest

これは、ダイジェスト形式の msql-mysql-modules リストである。

質問に対する回答が MySQL メーリングリストから得られなかった場合、MySQL AB から有料でサポートを受ける方法があります。その場合は、MySQL の開発者と直接連絡を取ることができます。 See 項1.4.1.「MySQL AB によって提供されるサポート」。

以下は、英語以外の言語による MySQL メーリングリストです。これらのリストは MySQL AB が運営しているものではないので、品質は保証いたしません。

- <mysql-france-subscribe@yahoogroups.com> フランス語のメーリングリスト , <list@tinc.net> 韓国語のメーリングリスト

  このリストに subscribe mysql your@e-mail.address を電子メールでお送りください。

- <mysql-de-request@lists.4t2.com> ドイツ語のメーリングリスト

  このリストに subscribe mysql-de your@e-mail.address を電子メールでお送りください。このメーリングリストについては、http://www.4t2.com/mysql/ を参照してください。

- <mysql-br-request@listas.linkway.com.br> ポルトガル語のメーリングリスト

  このリストに subscribe mysql-br your@e-mail.address を電子メールでお送りください。

- <mysql-alta@elistas.net> スペイン語のメーリングリスト

  このリストに subscribe mysql your@e-mail.address を電子メールでお送りください。

1.7.1.2. 質問またはバグの報告

バグレポートまたは質問を投稿する前に、以下のことを行ってください。

- 次の場所で MySQL のオンラインマニュアルを検索する。 http://www.mysql.com/doc/ マニュアルは、常に最新の状態にしておくために、新しく見つかった問題に対する解決策によって頻繁に更新されている。問題に対する解決策が新しいバージョンにすでに組み込まれている可能性が高いので、変更履歴に関する付録 (http://www.mysql.com/doc/en/News.html) は特に便利である。
- バグデータベース (http://bugs.mysql.com/) を検索して、バグがすでに報告/解決されているかどうかを確認する。
- 次の場所で MySQL メーリングリストのアーカイブを検索する。 http://lists.mysql.com/
- また、http://www.mysql.com/search/ を使用して、http://www.mysql.com/ にある Web ページすべて (マニュアルを含む) を検索することもできる。

マニュアルやアーカイブで回答を見つけることができなかった場合、ローカルの MySQL の専門家とともに調べてください。それでも質問に対する回答を見つけることができなかった場合は、当社に問い合わせる前に、次のセクションに記載されているガイドラインに従って MySQL メーリングリストにメールをお送りください。

1.7.1.3. バグまたは問題を報告する方法

当社のバグデータベースは公開されているので、すべてのユーザが http://bugs.mysql.com/ で参照および検索することができます。システムにログインすると、新しいレポートを入力することもできます。

適切なバグレポートを作成するには忍耐が必要ですが、最初に正しく作成しておくと、当社にとってもユーザ自身にとっても時間の節約になります。バグの詳細なテストケースを含む適切なバグレポートの場合、次のリリースでバグが修正される可能性が非常に高くなります。このセクションでは、当社にとってあまり、またはまったく役に立たない作業にユーザが時間を費やさなくて済むように、レポートを正しく作成する方法を説明します。

バグレポート (または問題に関するレポート) の生成には mysqlbug スクリプトを使用することをお勧めします。mysqlbug は、scripts ディレクトリ (ソースディストリビューション) および MySQL インストールディレクトリ下の bin ディレクトリ (バイナリディストリビューション) にあります。mysqlbug を使用することができない場合でも (Windows を使用している場合など)、このセクションに記されている必要な情報すべてをレポートに記載する必要があります (最も重要なのは、オペレーティングシステムの説明と MySQL バージョンです)。

mysqlbug スクリプトを使用すると、以下の情報のほとんどを自動的に確認することができるので、簡単にレポートを生成することができます。ただし、重要な情報が不足している場合は、その情報をメッセージに追加してください。このセクションをよく読んで、ここに記されているすべての情報をレポートに記載してください。

できれば、MySQL サーバの最新の製品版または開発版を使用して問題をテストしてから投稿してください。だれもが記載されているテストケースで mysql test < script を使用するだけでバグを再現したり、バグレポートに含まれているシェルまたは Perl スクリプトを実行したりすることができるようにする必要があります。

バグデータベース (http://bugs.mysql.com/) に投稿されたすべてのバグは、次の MySQL リリースで修正または文書化されます。小規模なコードの変更のみで問題を修正できる場合は、当社でも問題を修正するパッチを投稿します。

バグを報告する場所は通常、http://bugs.mysql.com/ です。

MySQL で重大なセキュリティバグを見つけた場合は、<security@mysql.com> に電子メールをお送りください。

再現可能なバグレポートがある場合、バグデータベース (http://bugs.mysql.com/) に報告してください。この場合も、まず mysqlbug スクリプトを実行して、システムに関する情報を確認することをお勧めします。再現可能なバグは、次の MySQL リリースで修正される可能性が高くなります。

他の問題を報告する場合は、MySQL メーリングリストのいずれかを使用してください。

情報が多すぎるメッセージに対応することはできますが、少なすぎるメッセージに対応することはできません。多くの場合、問題の原因がわかっていると思い、細部を重要でないと考えるため、事実を省略してしまいます。記載するかどうかを迷ったときは、記載することをお勧めします。最初に十分な情報を記載していなかったために再度質問して、回答を待たなければならなくなるよりも、レポートに数行を追加する方が、はるかに時間が節約される上に、煩わしくありません

バグレポートで最もよくある誤りは、(a) 使用している MySQL ディストリビューションのバージョン番号を記載していない、(b) MySQL サーバがインストールされているプラットフォームの説明 (プラットフォームの種類およびバージョン番号を含む) が十分でないというものです。これは非常に重要な情報なので、ほとんどの場合、この情報が記載されていないバグレポートは役に立ちません。``この機能が使用できないのはなぜですか?" という質問を受けることがよくあります。その場合、要求した機能がその MySQL バージョンに実装されていなかったり、レポートに記載されているバグが新しい MySQL バージョンですでに修正されていたりすることがあります。エラーがプラットフォーム依存である場合もあります。そのような場合、オペレーティングシステムやプラットフォームのバージョン番号を知らないで問題を修正することはほとんど不可能です。

また、コンパイラが問題に関連している場合は、コンパイラに関する情報も記載してください。ユーザがコンパイラのバ

グを見つけると、MySQL 関連の問題であると考えることがよくあります。ほとんどのコンパイラは常に開発中なので、バージョンごとに改良されています。問題がコンパイラに関連するものであるかどうかを判断するには、使用しているコンパイラを知る必要があります。コンパイルに関するすべての問題はバグと見なし、適宜報告してください。

問題に関する適切な説明がバグレポートに記載されていると、最も効果的です。そのため、問題につながったすべての操作の適切な例を挙げ、問題自体を詳細に記述してください。最も効果的なレポートは、バグや問題を再現する方法を示す詳細な例が記載されたものです。 See 項E.1.6.「テーブルが破損した場合にテストケースを作成する」。

プログラムでエラーメッセージが生成された場合、そのメッセージをレポートに記載することが非常に重要です。プログラムを使用するアーカイブから情報を検索しようとする場合、報告されたエラーメッセージがプログラムで生成されたものと正確に一致している方が効果的です（大文字小文字の違いにも注意してください）。エラーメッセージを覚えるのではなく、メッセージ全体をレポートにコピーして貼り付けてください。

MyODBC に関する問題がある場合、MyODBC トレースファイルを生成し、レポートともに送信してください。 See 項11.2.7.「MyODBC に関する問題の報告」。

レポートを読むユーザの多くが 80 桁ディスプレイを使用します。したがって、mysql コマンドラインツールを使用してレポートまたはサンプルを生成する際には、そのようなディスプレイの有効な幅を超える出力については、--vertical オプション（または \G ステートメント終端記号）を使用する必要があります（たとえば、EXPLAIN SELECT ステートメントの場合。このセクションの後述のサンプルを参照）。

レポートには以下の情報を記載してください。

- 使用している MySQL ディストリビューションのバージョン番号（MySQL バージョン 4.0.12 など）。実行しているバージョンを確認するには、mysqladmin version を実行する。mysqladmin は、MySQL インストールディレクトリ下の bin ディレクトリにある。
- 問題が発生したマシンの製造元とモデル。
- オペレーティングシステムの名前とバージョン。ほとんどのオペレーティングシステムでは、この情報を取得するには、Unix コマンド uname -a を実行する。Windows を使用している場合、名前とバージョン番号を取得するには、通常 ``マイ コンピュータ" アイコンをダブルクリックし、``ヘルプ/バージョン情報" メニューをプルダウンする。
- メモリ（実メモリと仮想メモリ）の量が関連することもある。関連していると思われる場合は、これらの値を記載する。
- MySQL ソフトウェアのソースディストリビューションを使用している場合、使用しているコンパイラの名前とバージョン番号が必要である。バイナリディストリビューションを使用している場合は、ディストリビューション名が必要である。
- コンパイル時に問題が発生した場合、正確なエラーメッセージ、およびエラーが発生したファイル内の問題のあるコード付近の数行のコンテキストを記載する。
- mysqld が停止した場合、mysqld のクラッシュの原因となったクエリも報告する必要がある。通常、ログを有効にして mysqld を実行すると、これを検出することができる。 See 項E.1.5.「mysqld におけるエラーの原因をログファイルを使用して特定する」。
- データベーステーブルが問題に関連している場合、mysqldump --no-data db_name tbl_name1 tbl_name2 ... からの出力を記載する。これを行うのは非常に簡単だが、データベース内のテーブルに関する情報を取得する強力な方法である。この情報は、発生した状況と同じ状況を再現する際に役立つ。

- SELECT ステートメントの速度に関するバグや問題については、必ず EXPLAIN SELECT ... の出力と、少なくとも SELECT ステートメントによって生成されたレコードの数を記載する必要がある。また、関連する各テーブルの SHOW CREATE TABLE tbl_name からの出力も記載する。発生した状況を説明する情報が詳細であるほど、的確な回答を得られる可能性が高くなる。次は、非常に適切なバグレポートの例である（当然のことながら、mysqlbug スクリプトを使用して投稿されたものである）。

  mysql コマンドラインツールを使用して実行したサンプル（出力幅が 80 桁ディスプレイ装置の出力幅を超えるステートメントへの \G ステートメント終端記号の使用に注意）

  ```
  mysql> SHOW VARIABLES;
  mysql> SHOW COLUMNS FROM ...\G
  <SHOW COLUMNS からの出力>
  mysql> EXPLAIN SELECT ...\G
      <EXPLAIN からの出力>
  mysql> FLUSH STATUS;
  mysql> SELECT ...;
      <クエリの実行に要した時間を含む、
      SELECT からの簡略な出力>
  mysql> SHOW STATUS;
      <SHOW STATUS からの出力>
  ```

- mysqld の実行中にバグまたは問題が発生した場合、その異常を再現する入力スクリプトを提供する。このスクリプトには、必要なソースファイルを含める必要がある。スクリプトによって再現される状況が実際に発生した状況に近いほど、効果的である。再現可能なテストケースを作成できる場合は、優先的に処理されるように http://bugs.mysql.com/ にそのテストケースを投稿する。

  スクリプトを提供できない場合は、少なくとも mysqladmin variables extended-status processlist からの出力をメールに記載して、システムの動作に関する情報を提供する必要がある。

- 数行ではテストケースを再現できない場合や、テストテーブルが大きすぎてメーリングリストに送信できない場合（10 レコードを超えるテーブル）、mysqldump を使用してテーブルをダンプし、問題を説明する README ファイルを作成する必要がある。

  tar と gzip または zip を使用してファイルの圧縮アーカイブを作成し、ftp を使用してそのアーカイブを ftp://support.mysql.com/pub/mysql/secret/ に転送する。その後、バグデータベース（http://bugs.mysql.com/）に問題を入力する。

- MySQL サーバでクエリによって正しくない結果が生成されたと思う場合、結果だけでなく、正しいと考える結果と、その考えの根拠を示す説明も記載する。

- 問題のサンプルを提供する際には、新しい名前を使用するよりも、実際の状況で使用している変数名やテーブル名などを使用するのが適切である。問題が、変数やテーブルの名前に関連している可能性があるためである。このようなケースはほとんどないと思われるが、後悔するよりも安全策をとるべきである。結局、実際の状況に基づいたサンプルを提供する方がユーザにとって簡単であると同時に、当社にとっても効率的である。データを他のユーザに公開したくない場合は、ftp を使用して ftp://support.mysql.com/pub/mysql/secret/ に転送することができる。データが実際に最高機密であり、当社にも公開したくない場合は、他の名前を使用したサンプルを提供することもできるが、これは最後の選択肢である。

- 可能な場合、関連するプログラムのすべてのオプションを記載する。たとえば、mysqld デーモンを開始する際に使用したオプションや MySQL クライアントプログラムの実行に使用したオプションを記載する。mysqld、mysql のようなプログラムのオプションや configure スクリプトのオプションは多くの場合、回答への手がかりとなり、非常に重要で

ある。これらを記載することは、決して無駄ではない。Perl や PHP などのモジュールを使用している場合は、それらのバージョン番号も記載する。

- 質問が特権システムに関連する場合、mysqlaccess の出力、mysqladmin reload の出力、および接続しようとした際に表示されたすべてのエラーメッセージを記載する。権限をテストする際には、まず mysqlaccess を実行する。その後、mysqladmin reload version を実行し、問題が発生したプログラムに接続する。mysqlaccess は、MySQL インストールディレクトリ下の bin ディレクトリにある。

- バグのパッチを作成した場合、それを含める。ただし、当社はパッチだけを必要としているわけではない。パッチによって修正されるバグを示すテストケースなどの必要な情報が記載されていない場合、当社はそのパッチを使用しない。当社がパッチに関連する問題を見つける場合もあれば、パッチをまったく理解できない場合もある。そのような場合は、パッチを使用することはできない。

  パッチの目的を正確に確認することができない場合、当社はそのパッチを使用しない。この場合、テストケースが役立つことになる。発生する可能性があるすべての状況がパッチによって処理されることを示す必要がある。パッチが機能しないボーダーラインケースが見つかった場合 (それがまれなケースでも) 、そのパッチは役に立たない可能性がある。

- 何がバグであるか、そのバグがなぜ発生するのか、そのバグが何に関連するのかを推測することは、通常、間違いである。MySQL チームでさえも、最初にデバッガを使用せずにそのようなことを推測して、バグの実際の原因を特定することはできない。

- ユーザ自身で問題を解決しようとしたことを他のユーザがわかるように、リファレンスマニュアルやメールアーカイブを確認したことをバグレポートに記載する。

- 解析エラーが発生した場合、構文を詳細に確認する必要がある。構文に問題が見つからなかった場合は、使用している構文が現在のバージョンの MySQL サーバでサポートされていない可能性が非常に高い。現在のバージョンを使用している場合、使用している構文が http://www.mysql.com/doc/ のマニュアルで扱われていなければ、そのクエリは MySQL サーバでサポートされていない。その場合は、自分で構文を実装するか、<licensing@mysql.com> に電子メールを送信して、その構文を実装するように依頼するしかない。

  使用している構文がマニュアルで扱われていても、古いバージョンの MySQL サーバを使用している場合は、MySQL の変更履歴で構文が実装された時期を確認する必要がある。この場合は、新しいバージョンの MySQL サーバにアップグレードする方法がある。 See 付録 D. MySQL Change History。

- データが壊れている点が問題である場合や、特定のテーブルにアクセスした際にエラーが発生した場合、myisamchk または CHECK TABLE と REPAIR TABLE を使用して、まずテーブルをチェックしてから、修復を試みる必要がある。 See 章 4. データベース管理。

- テーブルが頻繁に壊れる場合、その問題が発生する状況と理由を特定する必要がある。この場合、mysql-data-directory/'hostname'.err ファイルに、発生した問題に関する情報が格納されていることがある。See 項4.10.1.「エラーログ」。このファイル内の関連する情報をバグレポートに記載する。通常、更新の途中で mysqld が停止されない限り、mysqld によってテーブルが破壊されることはない。mysqld が停止した原因が特定されれば、当社はその問題の解決策をはるかに簡単に提供することができる。 See 項A.1.「問題の原因を突き止める方法」。

- 可能な場合、最新バージョンの MySQL サーバをダウンロードしてインストールし、これによって問題が解決されるかどうかを確認する。MySQL ソフトウェアのすべてのバージョンが詳細にテストされているため、問題なく動作するはずである。すべてにおいて最大限の下位互換性が確保されているため、MySQL のバージョンを問題なく切り替えることができると当社は確信している。 See 項2.2.4.「使用すべき MySQL のバージョン」。

サポート契約を結んでいるお客様は、他のユーザが同じ問題に遭遇しているかどうか（また、その問題が解決されているかどうか）を確認するために該当するメーリングリストにバグレポートを投稿するだけでなく、優先的に処理されるように <mysql-support@mysql.com> にも投稿してください。

MyODBC におけるバグの報告については、項11.2.4.「MyODBC に関する問題を報告する方法」 を参照してください。

一般的な問題に対する解決策については、付録 A. 問題と一般的なエラー を参照してください。

メーリングリストではなく、個人宛てに回答が送信された場合、問題の解決に役立った回答が他のユーザにも役立つように、回答を要約してメーリングリストに送信するのが工チケットであると思います。

1.7.1.4. メーリングリストで質問に回答する際のガイドライン

多数のユーザにとって興味深いと思われる回答については、質問した個人に直接返信するのではなく、メーリングリストに投稿することができます。その場合、元の投稿者以外のユーザにも役立つように、一般的な回答をするようにしてください。リストに投稿する際には、回答が前の回答と重複していないことを確認してください。

元のメッセージ全体を引用するのではなく、質問の重要な部分を回答に要約してください。

HTML モードがオンになっているブラウザからメールメッセージを投稿しないでください。多くのユーザはブラウザでメールを読みません。

1.7.2. IRC（インターネットリレーチャット）の MySQL コミュニティサポート

さまざまな MySQL メーリングリストに加え、IRC（インターネットリレーチャット）にも経験豊富なコミュニティがあります。以下は、当社が現在把握している最もすぐれたネットワーク/チャネルです。

- freenode（サーバについては、http://www.freenode.net/ を参照）
  - #mysql MySQL に関する質問が中心であるが、他のデータベースや SQL に関する質問も受け付けている。
  - #mysqlphp MySQL+PHP という一般的な組み合わせに関する質問。
  - #mysqlperl もう 1 つの一般的な組み合わせ、MySQL+Perl に関する質問。
- EFnet（サーバについては、http://www.efnet.org/ を参照）
  - #mysql MySQL に関する質問。

IRC ネットワークに接続する IRC クライアントソフトウェアを探している場合は、X-Chat（http://www.xchat.org/）を参照してください。X-Chat（GPL ライセンス）は、Unix および Windows プラットフォームで使用することができます。

1.8. MySQL の標準への準拠

このセクションでは、MySQL と ANSI/ISO SQL 標準との関連を説明します。MySQL サーバには SQL 標準に対する多数の拡張機能があります。ここでは、それらの拡張機能と使用方法を説明します。また、MySQL サーバに不足している機能と、差異を解決する方法に関する情報も提供します。

当社の目標は、正当な理由なく、あらゆる用途に対して MySQL サーバの使いやすさを制限しないことです。考えられるすべての用途に対応する開発を行うリソースがそろっていなくても、新しい領域で MySQL サーバを使用しようとしてい

るユーザを常に喜んで援助し、アドバイスを提供します。

製品に関する当社の主な目標の 1 つは、速度や信頼性を犠牲にすることなく、SQL-99 標準への準拠に向けて開発を続けることです。大部分のユーザにとって MySQL サーバが大幅に使いやすくなるのであれば、当社は SQL に対する拡張機能や SQL 以外の機能のサポートを積極的に追加します (MySQL サーバ 4.0 の新しい HANDLER インタフェースは、この方針の一例です。 See 項6.4.9.「HANDLER 構文」。)

当社は、負荷が高い Web/ログの使用とミッションクリティカルな年中無休の使用の両方に対応するために、トランザクションデータベースと非トランザクションデータベースのサポートを継続します。

MySQL サーバは、小規模なコンピュータシステムで中規模のデータベース (1,000 万 〜 1 億レコード、または 1 テーブルあたり約 100 MB) を使用することを目的として最初から設計されました。テラバイト規模のデータベースでも適切に動作するとともに、ハンドヘルドデバイスや組み込み用途により適した小規模な MySQL をコンパイルできるように、MySQL サーバの拡張を続けます。MySQL サーバのコンパクトな設計によって、ソースツリーで競合することなく、これらのどちらの方向も実現することができます。

現時点では、リアルタイムのサポートは考えていません (ただし、レプリケーションサービスを使用して、すでに多くのことを実行することができます)。

データベースクラスタのサポートは、新しいストレージエンジンの実装によって 2004 年内に開始される予定です。

データベースサーバでの XML サポートの提供を考えています。

1.8.1. MySQL が準拠する標準

エントリレベルは SQL-92 です。ODBC レベルは 0 〜 3.51 です。

コードの速度と品質を犠牲にすることなく、SQL-99 標準を完全にサポートすることを目標としています。

1.8.2. ANSI モードでの MySQL の実行

--ansi または --sql-mode=ANSI オプションを指定して mysqld を開始すると、MySQL サーバの動作は次のように変わります。

- || は、OR のシノニムではなく、文字列連結演算子になる。
- '"' は、文字列引用符ではなく、MySQL サーバの '`' 引用符のように識別子引用符として扱われる。'`' は、ANSI モードでも識別子の引用に使用することができる。つまり、文字列の引用に二重引用符を使用することはできない。二重引用符を使用すると、識別子として解釈されてしまうためである。
- 関数名と '(' の間にスペースをいくつでも配置することができる。これによって、すべての関数名が予約語として扱われる。そのため、予約語となっているデータベース名、テーブル名、またはカラム名にアクセスするには、それらの名前を引用符で囲む必要がある。たとえば、USER() 関数があるので、mysql データベース内の user テーブルの名前とそのテーブル内の User カラムの名前が予約語となるため、これらを引用符で囲まなければならない。

```
SELECT "User" FROM mysql."user";
```

- REAL は、DOUBLE のシノニムではなく、FLOAT のシノニムとなる。
- デフォルトのトランザクション分離レベルは、SERIALIZABLE となる。 See 項6.7.6.「SET TRANSACTION 構文」。

- フィールド一覧にない GROUP BY でフィールド/式を使用することができる。

ANSI モードでサーバを実行するのは、以下のオプションを指定してサーバを起動するのと同じことです。

```
--sql-mode=REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ONLY_FULL_GROUP_BY
--transaction-isolation=SERIALIZABLE
```

MySQL 4.1 では、次の 2 つのステートメントで同じ動作を実現することができます。

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET GLOBAL sql_mode =
 "REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ONLY_FULL_GROUP_BY";
```

MySQL 4.1.1 では、sql_mode オプションを次のように使用することもできます。

```
SET GLOBAL sql_mode="ansi";
```

この場合、sql_mode 変数の値は、ANSI モードに関連するすべてのオプションに設定されます。結果を確認するには、以下を実行します。

```
mysql> SET GLOBAL sql_mode="ansi";
mysql> SELECT @@GLOBAL.sql_mode;
+-----------------------------------------------------------------------+
| @global.sql_mode                                                      |
+-----------------------------------------------------------------------+
| REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ONLY_FULL_GROUP_BY |
+-----------------------------------------------------------------------+
1 row in set (0.00 sec)
```

1.8.3. SQL-92 標準に対する MySQL 拡張機能

MySQL サーバには、他の SQL データベースにはない拡張機能があります。それらの拡張機能を使用した場合、他の SQL サーバにコードを移植できなくなるので注意してください。場合によっては、MySQL 拡張機能を含むコードを記述しても、/*! ... */ 形式のコメントを使用することで移植することができます。その場合、MySQL サーバでは、他の MySQL ステートメントと同様にコメント内のコードが解析および実行されますが、他の SQL サーバでは拡張機能が無視されます。次に例を示します。

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

'!' の後ろにバージョン番号を追加すると、MySQL バージョンが、使用されているバージョン番号以降の場合にのみ、構文が実行されます。

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

つまり、バージョン 3.23.02 以降を使用している場合、MySQL サーバで TEMPORARY キーワードが使用されます。

以下は、MySQL 拡張機能の一覧です。

- フィールド型 MEDIUMINT、SET、ENUM、およびさまざまな BLOB および TEXT 型。

- フィールド属性 AUTO_INCREMENT、BINARY、NULL、UNSIGNED、および ZEROFILL。
- すべての文字列比較は、デフォルトでは大文字と小文字を区別せず、現在のキャラクタセット（デフォルトでは ISO-8859-1 Latin1）で決められたソート順で行われる。これを変更するには、BINARY 属性を指定してカラムを宣言するか、BINARY キャストを使用して、MySQL サーバホストで使用される ASCII の順序に従って比較が行われるようにする必要がある。
- MySQL サーバでは、各データベースは MySQL データディレクトリ下のディレクトリに、データベース内のテーブルはデータベースディレクトリ内のファイル名にマップされる。

  これには、次のような意味がある。

  - ほとんどの Unix システムのようにファイル名で大文字と小文字が区別されるオペレーティングシステム上の MySQL サーバでは、データベース名およびテーブル名は大文字と小文字を区別する。 See 項6.1.3.「名前におけるケース依存」。
  - データベース名、テーブル名、インデックス名、カラム名、またはエイリアス名を数字で始めることができる（ただし、数字のみから成る名前を使用することはできない）。
  - 標準のシステムコマンドを使用して、テーブルのバックアップ、名前の変更、移動、削除、およびコピーを行うことができる。たとえば、テーブルの名前を変更するには、テーブルに対応する .MYD、.MYI、および .frm ファイルの名前を変更する。
- SQL ステートメントで、db_name.tbl_name 構文を使用して、さまざまなデータベース内のテーブルにアクセスすることができる。同様の機能を備えた SQL サーバもあるが、これはユーザスペースと呼ばれる。MySQL サーバでは、CREATE TABLE ralph.my_table...IN my_tablespace のようなテーブルスペースはサポートされない。
- 数値カラムで LIKE を使用することができる。
- SELECT ステートメントでの INTO OUTFILE および STRAIGHT_JOIN の使用。 See 項6.4.1.「SELECT 構文」。
- SELECT ステートメントでの SQL_SMALL_RESULT オプション。
- テーブルの結合方法に関する説明を取得する EXPLAIN SELECT。
- インデックス名、フィールドのプリフィックス上のインデックス、および CREATE TABLE ステートメントでの INDEX または KEY の使用。 See 項6.5.3.「CREATE TABLE 構文」。
- CREATE TABLE での TEMPORARY または IF NOT EXISTS の使用。
- list に複数の要素がある COUNT(DISTINCT list) の使用。
- ALTER TABLE ステートメントでの CHANGE col_name、DROP col_name、DROP INDEX、IGNORE、または RENAME の使用。 See 項6.5.4.「ALTER TABLE 構文」。
- RENAME TABLE の使用。 See 項6.5.5.「RENAME TABLE 構文」。
- ALTER TABLE ステートメントでの複数の ADD、ALTER、DROP、または CHANGE 節の使用。
- DROP TABLE でのキーワード IF EXISTS の使用。
- 1 つの DROP TABLE ステートメントで複数のテーブルを破棄することができる。

- UPDATE および DELETE ステートメントの ORDER BY および LIMIT 節。
- INSERT INTO ... SET col_name = ... 構文。
- INSERT および REPLACE ステートメントの DELAYED 節。
- INSERT、REPLACE、DELETE、および UPDATE ステートメントの LOW_PRIORITY 節。
- LOAD DATA INFILE の使用。多くの場合、この構文は Oracle の LOAD DATA INFILE と互換性がある。 See 項6.4.8.「LOAD DATA INFILE 構文」。
- ANALYZE TABLE、CHECK TABLE、OPTIMIZE TABLE、および REPAIR TABLE ステートメント。
- SHOW ステートメント。 See 項4.6.8.「SHOW 構文」。
- ‘’ だけでなく、‘"’ または ‘`’ で文字列を囲むことができる。
- エスケープ文字 ‘\’ の使用。
- SET ステートメント。 See 項5.5.6.「SET 構文」。
- GROUP BY 部で、選択したすべてのカラムの名前を列挙する必要がない。これにより、ごく一部ではあるが、きわめて一般的なクエリのパフォーマンスが向上する。 See 項6.3.7.「GROUP BY 節で使用する関数と修飾子」。
- GROUP BY で ASC および DESC を指定することができる。
- 他の SQL 環境を使用していたユーザにわかりやすいように、MySQL サーバでは多数の関数のエイリアスがサポートされている。たとえば、すべての文字列関数で、標準の SQL 構文と ODBC 構文の両方がサポートされている。
- MySQL サーバでは、C プログラミング言語のように、|| および && 演算子が論理 OR および AND を意味すると解釈される。MySQL サーバでは、|| と OR、および && と AND はシノニムである。このすぐれた構文のために、MySQL サーバでは、文字列の連結に標準 SQL-99 の || 演算子を使用することができない。代わりに、CONCAT() を使用する。CONCAT() には引数をいくつでも使用できるので、|| 演算子の使用を MySQL サーバに変換するのは簡単である。
- CREATE DATABASE または DROP DATABASE。 See 項6.5.1.「CREATE DATABASE 構文」。
- % 演算子は MOD() のシノニムである。したがって、N % M は MOD(N,M) と同じである。% は、C プログラマを対象として、また PostgreSQL との互換性を確保するためにサポートされている。
- =、<>、<=、<、>=、>、<<、>>、<=>、AND、OR、または LIKE 演算子を、SELECT ステートメントの FROM の左側のカラム比較で使用することができる。次に例を示す。

```
mysql> SELECT col1=1 AND col2=2 FROM tbl_name;
```

- LAST_INSERT_ID() 関数。 See 項11.1.3.32.「mysql_insert_id()」。
- REGEXP および NOT REGEXP 拡張正規表現演算子。
- 1 つまたは複数の引数を使用する CONCAT() または CHAR() (MySQL サーバでは、これらの関数は引数をいくつでも使用することができる) 。
- BIT_COUNT()、CASE、ELT()、FROM_DAYS()、FORMAT()、IF()、PASSWORD()、ENCRYPT()、MD5()、ENCODE()、DECODE()、PERIOD_ADD()、PERIOD_DIFF()、TO_DAYS()、または WEEKDAY() 関数。

- 部分文字列を削除する TRIM() の使用。SQL-99 では、1 つの文字しか削除できない。
- GROUP BY 関数 STD()、BIT_OR()、BIT_AND()、BIT_XOR()、および GROUP_CONCAT()。 See 項6.3.7.「GROUP BY 節で使用する関数と修飾子」。
- DELETE + INSERT の代わりの REPLACE の使用。 See 項6.4.7.「REPLACE 構文」。
- FLUSH、RESET、および DO ステートメント。
- 次のように、:= を使用してステートメント内で変数を設定することができる。

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

1.8.4. MySQL と SQL-92 との違い

MySQL サーバを ANSI SQL 標準（SQL-92/SQL-99）および ODBC SQL 標準に準拠したものにすることを目標としていますが、以下のように、MySQL サーバが異なる動作を示すことがあります。

- VARCHAR 型のカラムでは、値が格納される際に後続のスペースが削除される。 See 項1.8.6.「MySQL の既知のエラーと設計上の問題」。
- CHAR 型のカラムが自動的に VARCHAR 型のカラムに変更されることがある。 See 項6.5.3.1.「カラムの暗黙的な変更」。
- テーブルを削除しても、テーブルの特権が自動的に取り消されない。テーブルの特権を取り消すには、明示的に REVOKE を発行する必要がある。 See 項4.4.1.「GRANT および REVOKE の構文」。

優先順位に従って並べられた、新しい拡張機能が MySQL サーバに追加される時期を示す一覧については、オンラインの MySQL TODO リスト（http://www.mysql.com/doc/en/TODO.html）を参照してください。これは、このマニュアルの TODO リストの最新版です。 See 項1.6.「MySQL の今後（TODO）」。

1.8.4.1. サブクエリ

MySQL バージョン 4.1 では、サブクエリと派生テーブル（名前のないビュー）がサポートされています。 See 項6.4.2.「サブクエリ構文」。

4.1 より前のバージョンの MySQL については、結合などの方法によって、ほとんどのサブクエリを記述し直すことができます。 See 項6.4.2.11.「初期の MySQL バージョンに合わせたサブクエリの書き換え」。

1.8.4.2. SELECT INTO TABLE

MySQL サーバでは、Sybase SQL 拡張機能 SELECT ... INTO TABLE ... はまだサポートされていません。代わりに、SQL-99 構文 INSERT INTO ... SELECT ... がサポートされています。これらは、基本的には同じです。 See 項6.4.3.1.「INSERT ... SELECT 構文」。

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

また、SELECT INTO OUTFILE... または CREATE TABLE ... SELECT を使用することもできます。

1.8.4.3. トランザクションとアトミックオペレーション

MySQL サーバ (バージョン 3.23-max およびすべてのバージョン 4.0 以降) では、InnoDB および BDB トランザクションストレージエンジンでトランザクションがサポートされています。InnoDB は、完全に ACID に準拠しています。 See 章 7. MySQL のテーブル型。

MySQL サーバのその他の非トランザクションテーブル型 (MyISAM など) は、``アトミックオペレーション"と呼ばれる別のデータ整合性のパラダイムに従います。トランザクションの観点では、MyISAM テーブルは事実上、常に AUTOCOMMIT=1 モードで動作すると言えます。アトミックオペレーションでは多くの場合、パフォーマンスの高さに値する整合性が確保されます。

両方のパラダイムをサポートする MySQL サーバでは、ユーザはアトミックオペレーションの速度を必要とするか、アプリケーションでトランザクション機能を使用する必要があるかを選択することができます。この選択は、テーブルごとに行うことができます。

ほとんどの場合、トランザクションテーブル型と非トランザクションテーブル型のどちらを選ぶかの決め手となるのはパフォーマンスです。トランザクションテーブルの場合、はるかに大きなメモリとディスク領域が必要で、CPU のオーバーヘッドも大きくなります。しかし、InnoDB のようなトランザクションテーブル型には特有の機能も多数あります。MySQL サーバのモジュール設計によって、このようなさまざまなストレージエンジンすべてを同時に使用することができるので、さまざまな要件に合わせて、あらゆる条件で最適なパフォーマンスを確保することが可能です。

しかし、MySQL サーバの機能を使用して、非トランザクションの MyISAM テーブルでも厳密な整合性を確保するにはどのようにすればよいのでしょうか。また、非トランザクションテーブルの機能はトランザクションテーブル型にどのように対抗できるのでしょうか。

1. トランザクションパラダイムでは、重大な状況で COMMIT ではなく ROLLBACK の呼び出しに依存するようにアプリケーションが作成されている場合、トランザクションの方が便利である。また、トランザクションでは、完了していない更新や失敗した活動がデータベースにコミットされることはない。サーバには自動ロールバックを行う機会が与えられ、データベースは保護される。

   MySQL サーバでは、ほとんどの場合、更新前に簡単なチェックを組み込んだり、データベースの不整合をチェックして、不整合が発生した場合には自動的に修復または警告する簡単なスクリプトを実行したりすることで、発生する可能性がある問題を解決することができる。MySQL 口グを使用したり、別の口グを 1 つ追加したりするだけで、通常、データの整合性が失われることなく、完全にテーブルを修復することができる。

2. ほとんどの場合、重要なトランザクション更新はアトミックな更新に記述し直すことができる。通常、トランザクションによって解決される整合性の問題はすべて、LOCK TABLES またはアトミックな更新を使用して解決することができる。これによって、サーバが自動的に停止するという、トランザクションデータベースシステムと共通する問題が回避される。

3. サーバが停止すると、トランザクションシステムでもデータが失われる可能性がある。個々のシステムの違いは、データが失われる可能性がある時間がどれだけ短いかという点だけである。100% 安全なシステムはなく、``十分に安全 " なだけである。最も安全なトランザクションデータベースシステムと言われている Oracle でさえも、そのような状況ではデータが失われることがあると報告されている。

   MySQL サーバを安全に使用するには、トランザクションテーブルを使用するかどうかに関係なく、バックアップを作

成し、バイナリログをオンにすればよいだけである。これにより、他のトランザクションデータベースシステムで可能なように、どのような状況からもリカバリすることができる。使用するデータベースシステムに関係なく、どのような場合でもバックアップを作成することが望ましいのは当然である。

トランザクションパラダイムには長所と短所があります。多数のユーザおよびアプリケーション開発者は、停止が発生する、または停止が必要な問題に関するコード化の容易さに頼っています。しかし、アトミックオペレーションのパラダイムに慣れていなかったり、トランザクションの方が詳しいという場合でも、非トランザクションテーブルの速度が、最も高速で最適に調整されたトランザクションテーブルの速度の 3 倍から 5 倍も速いという長所を考えてみてください。

整合性が最も重要な状況では、MySQL サーバは、非トランザクションテーブルでもトランザクションレベルの信頼性と整合性を提供します。LOCK TABLES を使用してテーブルをロックすると、整合性チェックが行われるまで、すべての更新が延期されます。読み取りロック (書き込みロックの逆) のみが設定されている場合、読み取りと挿入は引き続き行うことができます。新しく挿入したレコードは、読み取りがロックされているクライアントには、読み取りロックが解除されるまで表示されません。INSERT DELAYED を使用すると、ロックが解除されるまで挿入をローカルキューに入れることができ、クライアントは挿入が完了するまで待機する必要はありません。 See 項6.4.3.2.「INSERT DELAYED 構文」。

ここでいう ``アトミック" とは、魔法のようなものではありません。個々の更新の実行中は、他のユーザがそれを妨害できないようにするとともに、自動ロールバック (あまり注意を払わなかった場合に、トランザクションテーブルで行われることがあります) が行われないようにすることができるというだけです。また、MySQL サーバでは、ダーティリードが行われることもありません。

非トランザクションテーブルを使用する際のテクニックは、以下のとおりです。

- LOCK TABLES を使用して、通常はトランザクションを必要とするループをコード化することができる。そのため、実行中にレコードを更新するカーソルが不要である。
- ROLLBACK を使用しないように、以下の方法を使用することができる。

  1. LOCK TABLES ... を使用して、アクセスするすべてのテーブルをロックする。
  2. 条件をテストする。
  3. すべてに問題がなければ、更新する。
  4. UNLOCK TABLES を使用して、ロックを解除する。

  これは通常、ロールバックが行われる可能性があるトランザクションを使用するよりも、はるかに速い方法である。ただし、更新の途中でスレッドが停止された場合は、この方法では対応することができない。その場合、すべてのロックが解除されるが、一部の更新が実行されていない可能性がある。

- 関数を使用して、1 回の操作でレコードを更新することもできる。次のようなテクニックを使用すると、非常に効率的なアプリケーションを取得することができる。

  - 現在の値に関連してフィールドを変更する。
  - 実際に変更されたフィールドのみを更新する。

  たとえば、顧客情報に対して更新を行う場合、変更された顧客データのみを更新し、変更されたデータ、または変更されたデータに依存するデータが元のレコードと比較して変更されていないことだけをテストする。変更されたデータの

テストは、UPDATE ステートメントで WHERE 節を使用して行われる。レコードが更新されなかった場合、"Some of the data you have changed has been changed by another user." というメッセージがクライアントに表示される。その場合、ウィンドウに元のレコードと新しいレコードを表示して、使用する必要がある顧客レコードのバージョンをユーザが決定できるようにする。

これによって、カラムロックと類似しているが、現在の値に関連する値を使用して、カラムの一部のみが更新される点で、実際はカラムロックよりすぐれた機能が実現する。つまり、一般的な UPDATE ステートメントは次のようになる。

```
UPDATE tablename SET pay_back=pay_back+125;

UPDATE customer
 SET
   customer_date='current_date',
   address='new address',
   phone='new phone',
   money_he_owes_us=money_he_owes_us-125
 WHERE
   customer_id=id AND address='old address' AND phone='old phone';
```

これは非常に効率的で、別のクライアントが pay_back または money_he_owes_us カラムの値を変更していても動作する。

- 多くの場合、ユーザは、複数のテーブルの一意の識別子を管理するために ROLLBACK や LOCK TABLES を使用していた。これは、AUTO_INCREMENT カラムおよび SQL 関数 LAST_INSERT_ID() または C API 関数 mysql_insert_id() を使用することで、はるかに効率的に処理することができる。 See 項11.1.3.32.「mysql_insert_id()」。

  通常、行レベルのロックをコード化することができる。状況によっては実際にこれが必要なので、InnoDB テーブルでは行レベルのロックがサポートされている。MyISAM では、テーブルでフラグカラムを使用し、以下のようなことを実行することができる。

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

  レコードが見つかり、元のレコードで row_flag がすでに 1 でなくなっていた場合、MySQL は、影響を受けたレコードの数として 1 を返す。

  MySQL サーバでは前述のクエリが次のように変更されると考えることができる。

  ```
  UPDATE tbl_name SET row_flag=1 WHERE id=ID AND row_flag <> 1;
  ```

1.8.4.4. ストアドプロシージャとトリガ

ストアドプロシージャは、バージョン 5.0 開発ツリーで実装されます。 See 項2.3.4.「開発ソースツリーからのインストール」。

この試みは SQL-99 に基づくもので、Oracle PL/SQL に類似した（ただし、同じではありません）基本構文を使用します。これに加え、外部言語でフックする SQL-99 フレームワークも実装します。

ストアドプロシージャは、サーバでコンパイルし、格納することができる SQL コマンドのセットです。これが行われると、クライアントはクエリ全体の再発行を繰り返す必要がなく、ストアドプロシージャを参照することができます。その結

果、クエリを一度解析するだけで済み、サーバとクライアント間で送信される情報が少なくなるので、全体的なパフォーマンスが向上します。また、サーバに関数のライブラリを作成することで、概念レベルを高めることもできます。ただし、サーバ側で行われる作業が増え、クライアント（アプリケーション）側で行われる作業が少なくなるので、ストアドプロシージャによってデータベースサーバシステムの負荷が高くなります。

トリガは、MySQL バージョン 5.1 で実装される予定です。トリガは、実質的にはストアドプロシージャの一種で、特定のイベントが発生すると呼び出されます。たとえば、トランザクションテーブルからレコードが削除されるたびにトリガされるストアドプロシージャや、すべてのトランザクションが削除されると、顧客テーブルから対応する顧客を自動的に削除するストアドプロシージャを設定することができます。

1.8.4.5. 外部キー

MySQL サーバ 3.23.44 以降では、CASCADE、ON DELETE、および ON UPDATE を含む外部キー制約のチェックが InnoDB テーブルでサポートされています。 See 項7.5.5.2.「FOREIGN KEY 制約」。

他のテーブル型については、MySQL サーバでは現在、CREATE TABLE コマンドで FOREIGN KEY 構文のみが解析されますが、この情報は使用/保存されません。近いうちに、この情報がテーブル仕様ファイルに保存され、mysqldump および ODBC によって取得できるように、この実装を拡張する予定です。さらにその後には、MyISAM テーブルについても外部キー制約を実装する予定です。

SQL の外部キーはテーブルの結合に使用されるのではなく、参照整合性（外部キー制約）のチェックと確保に使用されます。SELECT ステートメントを使用して複数のテーブルから結果を得るには、次のようにテーブルを結合します。

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

See 項6.4.1.1.「JOIN 構文」。 See 項3.6.6.「外部キーの使用」。

制約として使用する際、アプリケーションによって MyISAM テーブルに適切な順序でレコードが挿入される場合は、外部キーを使用する必要はありません。

MyISAM テーブルについては、ON DELETE が実装されていないという問題に対処するには、外部キーがあるテーブルからレコードを削除する際に適切な DELETE ステートメントをアプリケーションに追加します。実際、この方法は外部キーを使用する場合と同じくらい簡単で（場合によっては、この方が簡単です）、移植性はそれよりもはるかに高くなります。

MySQL サーバ 4.0 では、複数テーブルの削除を使用して、1 つのコマンドで多数のテーブルからレコードを削除することができます。 See 項6.4.5.「DELETE 構文」。

ON DELETE ... を含まない FOREIGN KEY 構文は多くの場合、自動 WHERE 節を生成する ODBC アプリケーションによって使用されます。

外部キーを誤って使用すると、深刻な問題が発生することがあるので注意してください。適切に使用した場合でも、外部キーのサポートは、参照整合性の問題を解決する上で役に立つことはあっても、重要な解決策にはなりません。

外部キーを使用するメリットは、以下のとおりです。

- 関係が適切に設計されている場合、外部キー制約によって、プログラマがデータベースで不整合を引き起こすことが少なくなる。
- 連鎖更新および削除を使用すると、クライアントコードを単純化することができる。

- 適切に設計された外部キールールは、テーブル間の関係の記述に役立つ。

外部キーを使用するデメリットは、以下のとおりです。

- キー関係を設計する上で犯しやすい間違いによって、循環ルール、連鎖削除の不適切な組み合わせなどの深刻な問題が生じることがある。
- データベースレベルでの余分なチェックによって、パフォーマンスに影響が生じる。そのため、一部の主要な商用アプリケーションでは、アプリケーションレベルでこのロジックがコード化されている。
- DBA にとって、個々のテーブルのバックアップやリストアが非常に困難になり、場合によっては不可能になるような複雑な関係のトポロジを作成することはめったにない。

1.8.4.6. ビュー

ビューは現在実装中で、MySQL サーババージョン 5.0 または 5.1 で実装されます。

これまで、MySQL サーバは、アプリケーション作成者がデータベースの使用を完全に制御することができるアプリケーションや Web システムで最も多く使用されてきました。しかし、時間の経過とともに用途は変わってきており、ビューを重要視するユーザが増えています。

名前のないビュー (派生テーブル、SELECT の FROM 節内のサブクエリ) は、バージョン 4.1 ですでに実装されています。

ビューは、1 つのテーブルのように一連の関係 (テーブル) にユーザがアクセスできるようにし、ユーザのアクセスをそれのみに制限する場合に便利です。また、ビューを使用して、レコード (特定のテーブルのサブセット) へのアクセスを制限することもできます。MySQL サーバには高度な特権システムがあるので、カラムへのアクセスを制限する場合にはビューを使用する必要はありません。 See 項4.3.「一般的なセキュリティ関連事項と MySQL アクセス制御システム」。

多数の DBMS では、ビューに対する更新を行うことはできません。代わりに、個々のテーブルに対して更新を実行する必要があります。当社はビューの実装を設計する際に、理論的に更新可能なすべてのビューは実際にも更新可能でなければならないという、リレーショナルデータベースシステムに関する ``コッドのルール #6" に SQL の範囲内で可能な限り準拠することを目標としています。

1.8.4.7. コメントの開始記号としての '--'

一部の他の SQL データベースでは、コメントを開始するために '--' が使用されます。MySQL サーバでは、コメント開始文字として '#' が使用されます。また、C コメントスタイルの /* this is a comment */ を使用することもできます。 See 項6.1.6.「コメント構文」。

MySQL サーババージョン 3.23.3 以降では、'--' コメントスタイルを使用することができます。ただし、コメントの後にスペース (または、改行などの制御文字) が続く場合に限ります。これは、このコメントスタイルによって、!payment! の値を自動的に挿入する次のようなコードを使用した自動生成の SQL クエリで多数の問題が発生していたためです。

```
UPDATE tbl_name SET credit=credit-!payment!
```

payment の値が負数の場合、どうなるかを考えてみてください。SQL では 1--1 が有効なので、コメントを '--' で開始できるようにした場合の影響は重大です。

MySQL サーババージョン 3.23.3 以降でコメントのこの方法の実装を使用した場合、1-- This is a comment は実際に安全です。

もう 1 つの安全な機能は、mysql コマンドラインクライアントによって '--' で始まるすべての行が削除されるというものです。

以下の情報は、3.23.3 より前のバージョンの MySQL を実行している場合にのみ関連します。

テキストファイルの SQL プログラムに '--' コメントが含まれている場合、次のコマンドを使用する必要があります。

```
shell> replace " --" " #" < text-file-with-funny-comments.sql \
       | mysql database
```

通常の次のコマンドの代わりに、上記のコマンドを使用してください。

```
shell> mysql database < text-file-with-funny-comments.sql
```

また、コマンドファイルを ``その場で" 編集して、'--' コメントを '#' コメントに変更することもできます。

```
shell> replace " --" " #" -- text-file-with-funny-comments.sql
```

次のコマンドで元に戻してください。

```
shell> replace " #" " --" -- text-file-with-funny-comments.sql
```

1.8.5. MySQL における制約の処理

MySQL ではトランザクションテーブルとロールバックが有効でない非トランザクションテーブルの両方を使用することができるので、MySQL と他のデータベースとでは制約の処理が多少異なります。

エラー時にロールバックすることができない非トランザクションテーブルで多数のレコードを更新した場合の処理を実装しなければなりません。

基本的な概念は、検出可能なエラーをコンパイル時に生成し、受け取ったエラーから実行時にリカバリするというものです。ほとんどの場合にこれが実装されていますが、まだすべてについて実装されているわけではありません。 See 項 1.6.4. 「近い将来に計画されている新機能」。

MySQL における基本的なオプションは、途中でステートメントを中止するか、問題からリカバリするためにできる限りのことを行って、処理を続行するかです。

さまざまな種類の制約に関する問題を以下で説明します。

1.8.5.1. PRIMARY KEY/UNIQUE KEY 制約

通常、主キー、ユニークキー、または外部キー違反を引き起こすレコードの挿入/更新を行おうとすると、エラーが発生します。InnoDB のようなトランザクションストレージエンジンを使用している場合、MySQL ではトランザクションが自動的にロールバックされます。非トランザクションストレージエンジンを使用している場合、MySQL はエラーが発生したレコードで停止し、残りのレコードは未処理のままになります。

この問題を解決するために、MySQL では、キー違反が発生する可能性があるほとんどのコマンドに IGNORE ディレクティブのサポートを追加しました (INSERT IGNORE ... など) 。この場合、キー違反は無視され、引き続き次のレコードが

処理されます。MySQL における処理に関する情報を取得するには、mysql_info() API 関数を使用します。MySQL バージョン 4.1 では SHOW WARNINGS コマンドを使用します。 See 項11.1.3.30.「mysql_info()」。 See 項4.6.8.9.「SHOW WARNINGS | ERRORS」。

現時点では、外部キーがサポートされているのは InnoDB テーブルのみです。See 項7.5.5.2.「FOREIGN KEY 制約」。MyISAM テーブルでの外部キーサポートは、MySQL 5.0 ソースツリーで実装される予定です。

1.8.5.2. NOT NULL および DEFAULT 値制約

非トランザクションテーブルを簡単に処理できるように、MySQL のすべてのフィールドにはデフォルト値が設定されています。

NOT NULL カラムにおける NULL や数値カラムにおける大きすぎる数値のように '正しくない' 値をカラムに挿入した場合、MySQL ではエラーを生成するのではなく、カラムを '最適可能値' に設定します。数値については、0、使用可能な最小値、または使用可能な最大値です。文字列については、空白文字列、またはカラム内で使用可能な最長文字列です。

つまり、NULL 値を使用することができないカラムに NULL を格納しようとすると、0 または '' (空白文字列) が代わりに格納されます。単一レコードの挿入については、この後者の動作を -DDONT_USE_DEFAULT_FIELDS コンパイルオプションを使用して変更することができます。See 項2.3.3.「一般的な configure オプション」。この場合、非 NULL 値を必要とするすべてのカラムについて明示的に値を指定しない限り、INSERT ステートメントでエラーが生成されます。

前述のルールの理由は、クエリの実行が開始される前にこれらの条件をチェックできないためです。複数のレコードを更新した後で問題が発生した場合、テーブル型でサポートされていない可能性があるため、単にロールバックすることはできません。停止するというオプションは、この場合、更新が '未完了' のため、最悪のシナリオになる可能性があるので、適切ではありません。'できる限りのことを行って'、何も問題が発生していないものとして処理を続行する方が適切です。MySQL 5.0 では、自動フィールド変換に関する警告と、トランザクションテーブルのみを使用するステートメントで許可されていないフィールド割り当てが行われた場合にステートメントをロールバックすることができるオプションを提供することで、これを改善する予定です。

このことは、通常、フィールドの内容をチェックするために MySQL を使用するのではなく、アプリケーションでこれを処理しなければならないことを意味します。

1.8.5.3. ENUM および SET 制約

MySQL 4.x では、ENUM は実際の制約ではなく、特定の値セットのみを使用することができるフィールドを格納するためのより効率的な方法です。これは、NOT NULL が受け付けられないのと同じ理由によります。 See 項1.8.5.2.「NOT NULL および DEFAULT 値制約」。

ENUM フィールドに正しくない値を挿入した場合、予約された列挙数 0 に設定され、文字列コンテキストでは空白文字列として表示されます。 See 項6.2.3.3.「ENUM 型」。

SET フィールドに正しくない値を挿入した場合、その値は無視されます。 See 項6.2.3.4.「SET 型」。

1.8.6. MySQL の既知のエラーと設計上の問題

1.8.6.1. その後の MySQL バージョンで修正された 3.23 のエラー

以下の既知のエラー/バグは、MySQL 3.23 では修正されていません。これらの修正には、より深刻な他のバグを引き起こす可能性がある多数のコード変更が関連するためです。また、これらのバグは '非致命的' または '許容範囲' として分類されています。

- 複数のテーブルに対して LOCK TABLE を実行した後、別のスレッドがテーブルをロックしようとしている間に同じ接続でそれらのテーブルのいずれかに対して DROP TABLE を実行した場合、デッドロックが発生することがある。ただし、関連するスレッドのすべてに KILL を実行することで、この問題を解決することができる。4.0.12 で修正済み。
- テーブルのいずれかが空の場合、SELECT MAX(key_column) FROM t1,t2,t3... が NULL を返す代わりに、カラムの最大値を返す。4.0.11 で修正済み。
- ロックされた HEAP テーブルで、WHERE を含まない DELETE FROM heap_table が動作しない。

1.8.6.2. MySQL の未解決のバグと設計上の問題

以下の既知の問題は、優先して修正されます。

- FLUSH TABLES WITH READ LOCK によって CREATE TABLE または COMMIT がブロックされず、テーブルおよびバイナリログの完全バックアップを実行すると、バイナリログの位置に関する問題が発生する。
- BDB テーブルでの ANALYZE TABLE によって、mysqld を再実行するまでテーブルが使用できなくなることがある。この問題が発生した場合は、MySQL エラーファイルに次のようなエラーが生成される。

```
001207 22:07:56  bdb:  log_flush: LSN past current end-of-log
```

- MySQL では FROM 部でかっこを使用することができるが、自動的に無視される。エラーを生成しない理由は、クエリを自動的に生成する多くのクライアントで、不要な場合でも FROM 部にかっこが付け加えられるためである。
- 同じクエリ内で多数の RIGHT JOINS を連結した場合や LEFT および RIGHT 結合を組み合わせた場合、LEFT 結合の前のテーブルまたは RIGHT 結合の前のテーブルについて NULL 行のみが生成されるので、正しい結果が得られないことがある。これは、FROM 部におけるかっこのサポートの追加と同時に、5.0 で修正する予定である。
- マルチステートメントトランザクションを実行している BDB テーブルでは、それらのトランザクションすべてが完了するまで ALTER TABLE を実行しない（ほとんどの場合、トランザクションは無視される）。
- ANALYZE TABLE、OPTIMIZE TABLE、および REPAIR TABLE によって、INSERT DELAYED を使用しているテーブルで問題が発生することがある。
- LOCK TABLE ... および FLUSH TABLES ... を実行しても、処理が進行中の未完了のトランザクションがテーブルにないとは限らない。
- BDB テーブルを開くのに多少時間がかかる。データベースに多数の BDB テーブルがあるときに、-A オプションを使用していない場合や rehash を使用している場合、データベース上で mysql クライアントを使用するのに時間がかかる。これは、大きなテーブルキャッシュがある場合に特に顕著である。
- レプリケーションではクエリレベルのログが使用される。つまり、マスタによって、実行されたクエリがバイナリログに書き込まれる。これは非常に速く、コンパクトで効率的なログ方法であり、ほとんどの場合に問題なく機能する。実際に発生した例を聞いたことはないが、データの変更が非決定論的、つまりクエリオプティマイザの判断で行われるようにクエリが設計されている場合（レプリケーションの外部でも、通常は適切な方法でない）、マスタおよびスレーブのデータが理論的に異なる可能性がある。たとえば、次のような場合である。
  - AUTO_INCREMENT カラムにゼロまたは NULL 値を挿入する CREATE ... SELECT または INSERT ... SELECT ステートメント。

- ON DELETE CASCADE プロパティが設定された外部キーがあるテーブルからレコードを削除する場合の DELETE。
- 挿入されるデータに重複するキー値がある場合の REPLACE ... SELECT、INSERT IGNORE ... SELECT。

これらのクエリすべてに決定論的順序を保証する ORDER BY 節がない場合のみである。

実際、たとえば、ORDER BY がない INSERT ... SELECT の場合、SELECT は、マスタおよびスレーブでオプティマイザによって行われる選択によっては、異なる順序でレコードを返すことがある（その結果、レコードに異なるランクが設定され、auto_increment カラムで異なる数を取得することになる）。マスタとスレーブでクエリが異なって最適化されるのは、以下の場合のみである。

- 2 つのクエリによって使用されるファイルが、正確には同じでない。たとえば、OPTIMIZE TABLE がマスタテーブルでは実行されたが、スレーブテーブルでは実行されなかった場合などである（これを修正するために、MySQL 4.1.1 以降では、OPTIMIZE、ANALYZE、および REPAIR がバイナリログに書き込まれる）。
- マスタとスレーブで異なるストレージエンジンにテーブルが格納されている（スレーブとマスタで異なるストレージエンジンを使用することができる。たとえば、スレーブの方が使用可能なディスク領域が少ない場合、マスタでは InnoDB を使用し、スレーブでは MyISAM を使用することができる）。
- MySQL バッファのサイズ（key_buffer_size など）が、マスタとスレーブで異なる。
- マスタとスレーブで異なる MySQL バージョンが実行されており、オプティマイザのコードがこれらのバージョン間で異なる。

また、この問題は、mysqlbinlog|mysql によるデータベースのリストアにも影響を及ぼすことがある。

あらゆるケースでこの問題を回避する最も簡単な方法は、このような非決定論的なクエリに ORDER BY 節を追加して、レコードが常に同じ順序で格納/変更されるようにすることである。今後の MySQL バージョンでは、必要に応じて自動的に ORDER BY 節が追加されるようにする予定である。

以下の既知の問題は、近いうちに修正されます。

- テンポラリテーブルを使用して解決する必要があるクエリで RPAD 関数、または最終的に右側に空白を追加する他の文字列関数を使用した場合、すべての結果文字列で右側の空白が削除される。次に、このクエリの例を示す。

  SELECT RPAD(t1.field1, 50, ' ') AS f2, RPAD(t2.field2, 50, ' ') AS f1 FROM table1 as t1 LEFT JOIN table2 AS t2 ON t1.record=t2.joinID ORDER BY t2.record;

  このバグの最終的な結果は、生成されるフィールドの右側の空白を取得できなくなることである。

  この動作は、すべてのバージョンの MySQL に見られる。

  この原因は、テンポラリテーブルとして最初に使用される HEAP テーブルで VARCHAR 型のカラムを処理できないことである。

  この動作は、4.1 シリーズのいずれかのリリースで修正される予定である。

- テーブル定義ファイルの格納方法のために、テーブル名、カラム名、または列挙に文字 255（CHAR(255)）を使用することができない。これは、新しいテーブル定義ファイル形式が導入されるバージョン 5.1 で修正される予定である。

- SET CHARACTER SET を使用した場合、データベース名、テーブル名、およびカラム名で変換された文字を使用することができない。
- LIKE ... ESCAPE で ESCAPE とともに _ または % を使用することができない。
- DECIMAL 型のカラムにさまざまな形式 (+01.00、1.00、01.00) で数字が格納されている場合、GROUP BY によって各値が異なる値として見なされることがある。
- WHERE なしで DELETE FROM merge_table を使用すると、テーブルに対するマッピングのみが消去され、マップされたテーブルの内容は一切削除されない。
- MIT-pthreads を使用した場合、別のディレクトリにサーバを構築することができない。これには MIT-pthreads の変更が必要なので、これを修正する予定はない。 See 項2.3.6.「MIT-pthreads に関する注意事項」。
- GROUP BY、ORDER BY、または DISTINCT で BLOB 値を ``確実に" 使用することができない。これらの場合に BLOB 値を比較した場合、最初の max_sort_length バイト (デフォルトは 1024) のみが使用される。これは、mysqld の -O max_sort_length オプションを使用して変更することができる。ほとんどの場合の回避策は、SELECT DISTINCT LEFT(blob,2048) FROM tbl_name のように部分文字列を使用することである。
- BIGINT または DOUBLE で計算が実行される (通常はどちらの長さも 64 ビット)。取得する精度は関数によって異なる。一般的なルールとして、ビット関数は BIGINT 精度、IF および ELT() は BIGINT または DOUBLE 精度、残りは DOUBLE 精度で実行される。ビットフィールド以外については、符号なしの長い値が 63 ビット (9223372036854775807) より大きい値に解決される場合、そのような値を使用しないようにする必要がある。MySQL サーバ 4.0 では、3.23 に比べて BIGINT の処理が向上している。
- BLOB 型および TEXT 型のカラムを除くすべての文字列カラムで、取得時に後続のスペースすべてが自動的に削除される。CHAR 型の場合、これには問題はなく、SQL-92 に準拠した機能と見なすことができる。バグは、MySQL サーバで VARCHAR 型のカラムが同様に処理される点である。
- 1 つのテーブルに ENUM 型および SET 型のカラムを 255 個までしか作成することができない。
- MySQL では現在、MIN()、MAX()、およびその他の集約関数において、ENUM 型と SET 型のカラムがセット内の文字列の相対的な位置ではなく、文字列値によって比較される。
- mysqld_safe によって、mysqld から mysqld ログにすべてのメッセージがリダイレクトされる。これに関する 1 つの問題として、mysqladmin refresh を実行してログを閉じた後、もう一度開いた場合、stdout および stderr が引き続き元のログにリダイレクトされる点がある。--log を広範に使用する場合、'hostname'.log ではなく 'hostname'.err にログが記録されるように mysqld_safe を編集して、元のログを削除し、mysqladmin refresh を実行することで、元のログのスペースを簡単に解放できるようにする必要がある。
- UPDATE ステートメントで、カラムが左から右に更新される。更新されたカラムを参照した場合、元の値ではなく更新された値を取得する。たとえば、次のようなコマンドがあるとする。

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

  この場合、KEY は 1 ではなく、2 によって更新される。

- 1 つのクエリで複数のテンポラリテーブルを参照することができるが、特定のテンポラリテーブルを複数回参照することはできない。たとえば、次のクエリは動作しない。

```
mysql> SELECT * FROM temporary_table, temporary_table AS t2;
```

- RENAME が、TEMPORARY テーブル、または MERGE テーブル内で使用されているテーブルで動作しない。
- 結合で '非表示の' カラムを使用している場合と使用していない場合とで、オプティマイザによって DISTINCT が異なって処理されることがある。結合では、非表示のカラムは結果の一部としてカウントされるが（表示されていない場合も同様）、通常のクエリでは、非表示のカラムは DISTINCT 比較に関連しない。DISTINCT を実行した場合も非表示のカラムが比較されないように、今後これを変更する予定である。

  この例は以下のとおりである。

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

  および

```
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;
```

  2 番目の例の場合、MySQL サーバ 3.23.x では、結果セットで 2 つの同じレコードを取得することがある（非表示の id カラムが異なる場合があるため）。

  この問題は、結果に ORDER BY カラムがないクエリのみで発生する。これは、SQL-92 では許可されていない。

- MySQL サーバではトランザクションをサポートしないためにデータをロールバックすることができないテーブル型を使用することができるので、MySQL サーバと他の SQL サーバでは動作が多少異なる。これは単に、MySQL サーバでは SQL コマンドに対してロールバックが実行されないようにするためである。これは、アプリケーションでカラム値をチェックする必要があるときには多少不便な場合があるが、他の方法では非常に困難な最適化を MySQL サーバで実行することができるので、実質的には速度が大幅に向上する。

  カラムを正しくない値に設定した場合、ロールバックが行われる代わりに、カラムに取りうる値のうち最適な値が格納される。

  - 数値カラムに範囲外の値を格納しようとした場合、使用可能な最小値または最大値が代わりに格納される。
  - 数値カラムに数字以外で始まる文字列を格納しようとした場合、0 が格納される。
  - NULL 値を使用することができないカラムに NULL を格納しようとした場合、0 または ''（空白文字列）が代わりに格納される（ただし、この動作は、-DDONT_USE_DEFAULT_FIELDS コンパイルオプションを使用して変更することができる）。
  - MySQL では、DATE 型および DATETIME 型のカラムに正しくない日付値（2000-02-31 や 2000-02-00 など）を格納することができる。その考えは、日付を検証するのは SQL サーバの役目ではないというものである。MySQL で日付を格納し、同じ日付を正確に取得できる場合、その日付は格納される。日付がまったく正しくない（サーバで日付を格納することができない）場合は、特殊な日付値 0000-00-00 がカラムに格納される。
  - ENUM 型のカラムをサポートされていない値に設定した場合、エラー値である空白文字列に設定される。数値の場合は、0 に設定される。

  - SET 型のカラムをサポートされていない値に設定した場合、その値は無視される。
- 空のセットを返すクエリで PROCEDURE を実行した場合、PROCEDURE によってカラムが変換されないことがある。
- MERGE 型のテーブルを作成する際に、基になるテーブルの型に互換性があるかどうかがチェックされない。
- MySQL サーバではまだ、NaN、-Inf、および Inf 値を倍精度で処理することができない。これらを使用した場合、データをエクスポートおよびインポートしようとすると問題が発生する。中間的な解決策として、NaN を NULL（可能な場合）、-Inf および Inf をそれぞれ使用可能な最小および最大倍精度値に変更する必要がある。
- ALTER TABLE を使用して MERGE テーブルで使用されているテーブルに UNIQUE インデックスを追加した後、ALTER TABLE を使用して MERGE テーブルに通常のインデックスを追加すると、一意でない古いキーがテーブルに存在していた場合、テーブルでキーの順序が変わる。これは、重複キーをできるだけ早く検出できるように、ALTER TABLE が UNIQUE キーを通常のキーの前に配置するためである。

以下は、MySQL の初期のバージョンにおける既知のバグです。

- LOCK TABLES を使用してロックされている多数のテーブルのうちの 1 つのテーブルで DROP TABLE を実行した場合、ハングスレッドを取得することがある。
- 以下の場合、コアダンプを取得することがある。
  - 遅延挿入ハンドラがテーブルへの挿入を保留している場合。
  - WRITE を含む LOCK table。
  - FLUSH TABLES。
- MySQL サーババージョン 3.23.2 より前は、同じキーで WHERE を使用してキーを更新した UPDATE でエラーが発生することがあった。これは、そのキーがレコードの検索に使用され、同じレコードが複数回検出されることがあったためである。次に例を示す。

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

  回避策は次のとおりである。

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

  これが機能するのは、MySQL サーバでは WHERE 節内の式にインデックスが使用されないためである。
- MySQL サーババージョン 3.23 より前は、すべての数値型が固定小数点型フィールドとして扱われていた。そのため、固定小数点型フィールドの小数部桁数を指定しなければならなかった。すべての結果が、正確な小数部桁数で返されていた。

プラットフォーム固有のバグについては、コンパイルおよび移植に関するセクションを参照してください。 See 項2.3.「MySQL ソースディストリビューションのインストール」。 See 付録 E. 他システムへの移植。

第2章 MySQL のインストール

この章では、MySQL の入手方法とインストール方法について説明します。

- MySQL を入手できるサイトの一覧については、項2.2.1.「MySQL の入手方法」 を参照。
- サポートされているプラットフォームを確認するには、項2.2.3.「MySQL がサポートしているオペレーティングシステム」 を参照。サポートされているすべてのシステムが、MySQL の実行に同じように適しているとは限らないことに注意する。 一部のシステムは、他のシステムよりもはるかに堅牢で効率的である。詳細については、項2.2.3.「MySQL がサポートしているオペレーティングシステム」 を参照。
- MySQL のいくつかのバージョンについては、バイナリディストリビューションとソースディストリビューションの両方が用意されている。また、最新の開発を確認したいユーザや新しいコードのテストに協力したいユーザのために、当社の現在のソースツリーへも公開している。使用すべきディストリビューションのバージョンとタイプを判断するには、項2.2.4.「使用すべき MySQL のバージョン」 を参照。判断がつかない場合は、バイナリディストリビューションを使用する。
- バイナリディストリビューションとソースディストリビューションのインストール手順については、それぞれ、項2.2.9.「MySQL バイナリディストリビューションのインストール」 と項2.3.「MySQL ソースディストリビューションのインストール」 を参照。各手順セットには、発生する可能性のあるシステム固有の問題に関するセクションが含まれている。
- インストール後の手順については、項2.4.「インストール後の設定とテスト」 を参照。これらの手順は、バイナリディストリビューションとソースディストリビューションのどちらを使用して MySQL をインストールする場合にも適用される。

2.1. 標準 MySQL のクイックインストール

この章では、それぞれのプラットフォームのネイティブパッケージ化形式を使用してパッケージが提供されているプラットフォームへの MySQL のインストールについて説明します。ただし、その他のさまざまなプラットフォーム用の MySQL バイナリディストリビューションも用意されています。すべてのプラットフォームに適用されるパッケージの一般インストール手順については、項2.2.9.「MySQL バイナリディストリビューションのインストール」 を参照してください。

入手できるその他のバイナリディストリビューションとその入手方法の詳細については、項2.2.「インストール関連の一般的な問題」 を参照してください。

2.1.1. Windows への MySQL のインストール

Windows での MySQL のインストールは、以下のステップになります。

1. ディストリビューションのインストール
2. オプション設定ファイルの設定 (必要時)
3. 使用するサーバの選択
4. サーバの起動

Windows 用 MySQL は、以下の2種類の形式で提供されます。

- バイナリディストリビューション。セットアッププログラムが含まれており、このプログラムによって必要なものがすべてインストールされるので、サーバを直ちに起動できる。
- ソースディストリビューション。VC++ 6.0 コンパイラを使用して実行ファイルをビルドするためのすべてのコードとサポートファイルが含まれている。 See 項2.3.7.「Windows 上でソースから MySQL をインストールする」。

一般的には、バイナリディストリビューションを使用することをお勧めします。このディストリビューションの方が単純で、MySQL のセットアップと実行に追加のツールを必要としないからです。

2.1.1.1. Windows システム要件

Windows で MySQL を実行する場合の要件は以下のとおりです。

- 9x、Me、NT、2000、XP などの 32 ビットの Windows オペレーティングシステム。 NT ファミリ (Windows NT、2000、および XP) では、MySQL サーバをサービスとして実行することができる。 See 項2.1.1.7.「Windows NT、2000、または XP での MySQL の起動」。
- TCP/IP プロトコルサポート。
- Windows 用 MySQL バイナリディストリビューションのコピー。これは、http://www.mysql.com/downloads/ からダウンロードできる。

  注意: ディストリビューションファイルは、zip 形式で提供されます。ダウンロードプロセス中のファイルの破損を避けるために、レジューム機能を持つ適切な FTP クライアントを使用することをお勧めします。
- ディストリビューションファイルをアンパックするための ZIP プログラム。
- 要件に応じたデータベースのアンパック、インストール、作成を行うための十分な空き領域がハードディスクドライブ上にあること。
- ODBC を経由して MySQL サーバに接続する予定の場合は、MyODBC ドライバも必要である。 See 項11.2.「MySQL の ODBC サポート」。
- 4 GB より大きいサイズのテーブルが必要な場合は、NTFS またはそれより新しいファイルシステムに MySQL をインストールする。テーブルを作成するときに、MAX_ROWS と AVG_ROW_LENGTH を必ず使用する。 See 項6.5.3.「CREATE TABLE 構文」。

2.1.1.2. Windows バイナリディストリビューションのインストール

バイナリディストリビューションを使用して Windows に MySQL をインストールするには、以下の手順に従って行います。

1. Windows NT、2000、または XP が搭載されたマシンで作業する場合は、必ず管理者権限を持つユーザとしてログインする。
2. 旧バージョンの MySQL インストールをアップグレードする場合は、現在のサーバを停止する必要がある。 Windows

NT、2000、または XP が搭載されたマシンで、Windows サービスとしてサーバを実行している場合は、コマンドプロンプトで以下のように入力してサーバを停止する。

```
C:\> NET STOP MySQL
```

アップグレード後に別のサーバを使用する予定の場合（mysqld ではなく mysqld-max を実行する場合など）は、以下のように入力して既存のサービスを削除する。

```
C:\mysql\bin> mysqld --remove
```

アップグレード後にサービスを再インストールして適切なサーバを使用することができる。

MySQL サーバをサービスとして実行していない場合は、以下のように入力してサーバを停止する。

```
C:\mysql\bin> mysqladmin -u root shutdown
```

3. WinMySQLAdmin プログラムを実行している場合は、このプログラムを終了する。
4. テンポラリディレクトリにディストリビューションファイルを展開する。
5. setup.exe プログラムを実行して、インストールプロセスを開始する。 デフォルトのディレクトリ（C:\mysql）以外の場所に MySQL をインストールする場合は、Browse ボタンを使用して、目的のディレクトリを指定する。MySQL をデフォルト以外の場所にインストールすると、サーバを起動するたびにその場所を指定しなければならなくなる。これを行う最も簡単な方法は、項2.1.1.3.「Windows MySQL 環境の準備」 で説明するように、オプション設定ファイルを使用することである。
6. インストールプロセスを終了する。

2.1.1.3. Windows MySQL 環境の準備

サーバを実行するときに起動オプションを指定する必要がある場合は、コマンドラインで起動オプションを指定するか、オプション設定ファイルに起動オプションを配置します。サーバを起動するたびに使用されるオプションの場合、オプション設定ファイルを使用して MySQL のオプションを指定する方が便利です。この方法は、以下のような場合に特に便利です。

- インストールディレクトリまたはデータディレクトリの場所がデフォルトの場所（C:\mysql と C:\mysql\data）と異なる場合。
- サーバの設定を調整する必要がある場合。 たとえば、MySQL バージョン 3.23 の InnoDB トランザクションテーブルを使用するには、InnoDB のデータファイルとログファイルを格納するための 2 つのディレクトリ（C:\ibdata と C:\iblogs など）を手動で作成しなければならない。 また、項7.5.3.「InnoDB 起動オプション」で説明するように、オプション設定ファイルに行を追加する必要もある（MySQL 4.0 以降では、InnoDB はデフォルトでデータディレクトリにデータファイルとログファイルを作成する。したがって、InnoDB の設定は必須ではない。 ただし、必要に応じて明示的に設定することもできるが、その場合もオプション設定ファイルを使用すると便利である）。

Windows では、MySQL インストーラは、MySQL のインストール先のディレクトリの直下にデータディレクトリを格納します。別の場所のデータディレクトリを使用する場合は、data ディレクトリの内容全体をその新しい場所にコピーしてください。たとえば、デフォルトでは、インストーラは、MySQL を C:\mysql に、データディレクトリを C:\mysql\data に

それぞれ格納します。E:\mydata のデータディレクトリを使用する場合は、以下の 2 つの処理を行う必要があります。

- データディレクトリを C:\mysql\data から E:\mydata に移動する。
- サーバを起動するたびに、--datadir オプションを使用して新しいデータディレクトリの場所を指定する。

MySQL サーバは、Windows 上で起動すると、Windows ディレクトリにある my.ini ファイル内と C:\my.cnf ファイル内でオプションを探します。通常、Windows ディレクトリには、C:\WINDOWS や C:\WinNT のような名前が付いています。以下のコマンドを使用して、%WINDIR% 環境変数の値から正確な場所を判別できます。

```
C:\> echo %WINDIR%
```

MySQL は、最初に my.ini ファイル内でオプションを探し、次に my.cnf ファイル内でオプションを探します。しかし、混乱を避けるために、1 つのファイルだけを使用することをお勧めします。C: ドライブがブートドライブではないときに PC がブートローダを使用する場合は、my.ini ファイルしか使用できません。どちらのファイルを使用する場合でも、ファイルはプレーンテキストファイルでなければなりません。

オプション設定ファイルの作成や修正は、Notepad プログラムなどのテキストエディタを使用して行うことができます。たとえば、MySQL が D:\mysql にインストールされていて、データディレクトリが D:\mydata\data として配置されている場合は、オプション設定ファイルを作成して、以下のように [mysqld] セクションを設定して、basedir パラメータと datadir パラメータの値を指定できます。

```
[mysqld]
# set basedir to your installation path
basedir=D:/mysql
# set datadir to the location of your data directory
datadir=D:/mydata/data
```

注意: Windows のパス名は、オプション設定ファイル内でバックスラッシュではなくスラッシュを使用して指定します。バックスラッシュを使用する場合は、スラッシュを二重にする必要があります。

オプション設定ファイルを管理するもう 1 つの方法は、WinMySQLAdmin ツールを使用する方法です。MySQL インストールの bin ディレクトリに、WinMySQLAdminとその使用説明が含まれたヘルプファイルがあります。WinMySQLAdmin にはオプション設定ファイルを編集する機能がありますが、以下の点に注意してください。

- WinMySQLAdmin は、%WINDIR%\my.ini ファイルだけを使用する。
- WinMySQLAdmin は、C:\my.cnf ファイルを検出すると、このファイルの名前を C:\my_cnf.bak に変更して事実上無効にする。

これで、サーバの起動をテストする準備ができました。

2.1.1.4. Windows サーバの選択

MySQL 3.23.38 以降、Windows ディストリビューションに標準の MySQL と MySQL-Max の両方のサーババイナリが含まれるようになりました。 以下に各種の MySQL サーバの一覧を示します。この中からサーバを選択できます。

バイナリ	説明
mysqld	デバッグコードおよび自動メモリ割り当てのチェック、シンボリックリンクの機能、InnoDB テ

	ーブルおよび BDB テーブルを組み込んでコンパイルされている。
mysqld-opt	最適化されたバイナリ。バージョン 4.0 以降、InnoDB が有効になっている。4.0 より前のバージョンでは、このサーバにトランザクションテーブルの機能は含まれていない。
mysqld-nt	名前付きパイプの機能を組み込んで NT/2000/XP 向けに最適化されたバイナリ。
mysqld-max	シンボリックリンクと、InnoDB テーブルおよび BDB テーブルを組み込んで最適化されたバイナリ。
mysqld-max-nt	mysqld-max と似ているが、名前付きパイプの機能を組み込んでコンパイルされている。

上記のバイナリはすべて、最新の Intel プロセッサ向けに最適化されていますが、Intel i386 クラス以上のすべてのプロセッサで動作します。

mysqld-nt サーバまたは mysqld-max-nt サーバは、名前付きパイプ接続をサポートします。これらのサーバのどちらかを使用する場合、名前付きパイプの使用には以下の制約があることに留意してください。

- サーバは、名前付きパイプをサポートする Windows のバージョン (NT、2000、XP) で実行する必要がある。
- バージョン 3.23.50 からは、--enable-named-pipe オプションを使用してサーバを起動した場合にのみ、名前付きパイプが有効になる。
- これらのサーバは、Windows 98 または Me で実行することができるが、TCP/IP がインストールされている場合に限られる。つまり、名前付きパイプを使用した接続はできない。
- Windows 95 では、これらのサーバを使用することはできない。

2.1.1.5. サーバを初めて起動する

Windows 95、98、または Me では、MySQL クライアントは常に TCP/IP を使用してサーバに接続します。Windows NT、2000、XP などの NT ベースのシステムでは、クライアントには 2 つの選択肢があります。つまり、TCP/IP を使用するか、サーバが名前付きパイプ接続をサポートしている場合は名前付きパイプを使用します。

サーババイナリの実行の詳細については、項2.1.1.3. 「Windows MySQL 環境の準備」 を参照してください。

このセクションでは、MySQL サーバの起動方法の一般的な概要を説明します。 特定のバージョンの Windows に固有の詳細については、後続の各セクションで説明します。

これらの各セクションで示される例では、MySQL がデフォルトの場所である C:\mysql にインストールされていることが前提となっています。MySQL を別の場所にインストールしている場合は、例に示されているパス名を調整してください。

テストは、コンソールウィンドウ (``DOS 窓'') のコマンドプロンプトから行うことをお勧めします。この方法をとると、サーバからのステータスメッセージを見やすいウィンドウで表示することができます。設定に問題がある場合に、これらのメッセージによって問題の特定と修正が容易になります。

サーバが格納されているディレクトリにいることを確認してから、以下のコマンドを入力します。

```
shell> mysqld --console
```

InnoDB が含まれているサーバの場合は、サーバの起動時に以下のメッセージが表示されます。

```
InnoDB: The first specified datafile c:\ibdata\ibdata1 did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file c:\ibdata\ibdata1 size to 209715200
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file c:\iblogs\ib_logfile0 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile0 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile1 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile1 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile2 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile2 size to 31457280
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: creating foreign key constraint system tables
InnoDB: foreign key constraint system tables created
011024 10:58:25  InnoDB: Started
```

サーバが起動シーケンスを終了すると、クライアント接続をサービスする準備ができたことを示す以下のようなメッセージが表示されます。

```
mysqld: ready for connections
Version: '4.0.14-log'  socket: ''  port: 3306
```

続いて、サーバが生成したその他の診断出力がコンソールに出力されます。新しいコンソールウィンドウを開いて、そのウィンドウでクライアントプログラムを実行することができます。

--console オプションを省略すると、診断出力はデータディレクトリ内のエラーログに書き込まれます。エラーログは、.err 拡張子を持つファイルです。

2.1.1.6. Windows 95、98、または Me での MySQL の起動

Windows 95、98、または Me では、MySQL は TCP/IP を使用してクライアントをサーバに接続します (これによって、ネットワーク上のすべてのマシンが MySQL サーバに接続できます) 。そのため、MySQL を起動する前に、TCP/IP がインストールされていることを確認する必要があります。TCP/IP は、Windows の CD-ROM に収められています。

注意: 旧リリースの Windows 95 (OSR2 など) を使用している場合は、旧バージョンの Winsock パッケージがインストールされている可能性があります。MySQL は Winsock 2 を必要とします。最新の Winsock は、http://www.microsoft.com/ から入手できます。Windows 98 は、Winsock 2 ライブラリを備えています。したがって、ライブラリを更新する必要はありません。

mysqld サーバを起動するには、コンソールウィンドウ (``DOS'' ウィンドウ) を起動して、以下のコマンドを入力します。

```
shell> C:\mysql\bin\mysqld
```

このコマンドによって、バックグラウンドで mysqld が起動されます。したがって、サーバが起動した後、別のコマンドプロンプトが表示されます。注意: Windows NT、2000、XP でこの方法でサーバを起動すると、サーバはフォアグラウンドで実行され、サーバの実行が終了するまでコマンドプロンプトは表示されません。 そのため、サーバの実行中は、別のコンソールウィンドウを開いてクライアントプログラムを実行する必要があります。

以下のコマンドを実行して、MySQL サーバを停止することができます。

```
shell> C:\mysql\bin\mysqladmin -u root shutdown
```

このコマンドは、MySQL 管理ユーティリティ mysqladmin を起動して、サーバに接続してシャットダウンするよう指示します。このコマンドは、MySQL 権限システムでのデフォルトの管理者アカウントである root として接続します。MySQL 権限システム内のユーザは、Windows のログインユーザとは完全に独立しています。

mysqld が起動しない場合は、エラーログをチェックして、問題の原因を示すメッセージが書き込まれていないかどうかを確認してください。 エラーログは、C:\mysql\data ディレクトリにあります。これは、.err というサフィックスが付いたファイルです。また、mysqld --console としてサーバを起動することもできます。その場合、問題を解決するのに役立つ有益な情報が画面に表示されます。

最後の選択肢は、--standalone --debug を指定して mysqld を起動する方法です。 この場合、mysqld は、mysqld が起動しない理由をログファイル C:\mysqld.trace に書き込みます。 See 項E.1.2.「トレースファイルの作成」。

mysqld --help を使用すると、mysqld が認識できるすべてのオプションが表示されます。

2.1.1.7. Windows NT、2000、または XP での MySQL の起動

NT ファミリ (Windows NT、2000、または XP) では、MySQL を Windows サービスとしてインストールして実行することをお勧めします。Windows の起動時または停止時に、MySQL サーバが自動的に起動または停止されます。サービスとしてインストールされたサーバは、NET コマンドを使用してコマンドラインから制御するか、Services ユーティリティを使用して制御することができます。

Services ユーティリティ (Windows Service Control Manager) は、Windows Control Panel (Windows 2000 の Administrative Toolsの下) にあります。このコマンドラインからサーバのインストールや削除の操作を実行している間は、Services ユーティリティを終了することをお勧めします。これによって、ある種の予期しないエラーが防止されます。

Windows NT 4 で、MySQL を TCP/IP と連動させるには、Service Pack 3 以上をインストールする必要があります。

MySQL を Windows サービスとしてインストールする前に、まず、以下のコマンドを使用して現在のサーバを停止してください (サーバが稼動している場合) 。

```
shell> C:\mysql\bin\mysqladmin -u root shutdown
```

このコマンドは、MySQL 管理ユーティリティ mysqladmin を起動して、サーバに接続してシャットダウンするよう指示します。このコマンドは、MySQL 権限システム内のデフォルトの管理者アカウントである root として接続します。MySQL 権限システム内のユーザは、Windows のログインユーザから完全に独立しています。

次に、以下のコマンドでサーバをサービスとしてインストールします。

```
shell> mysqld --install
```

サーバ名だけを使用して mysqld をサービスとしてインストールする際に問題が発生した場合は、完全パス名を使用してインストールしてみてください。

```
shell> C:\mysql\bin\mysqld --install
```

MySQL 4.0.2 以降では、--install オプションの後ろに特定のサービス名を指定できます。さらに、MySQL 4.0.3 以降では、サービス名の後ろに --defaults-file オプションを指定して、起動時のオプションを取得する場所をサーバに示すことができます。サーバが使用するサービス名とオプション設定ファイルは、以下のルールに従って決定されます。

- サービス名を指定しないと、サーバはデフォルトサービス名 MySQL を使用し、デフォルトのオプション設定ファイルの [mysqld] グループからオプションを読み取る。
- --install オプションの後ろにサービス名を指定すると、サーバは [mysqld] オプショングループを無視し、そのサービスと同じ名前のグループからオプションを読み取る。サーバは、デフォルトのオプション設定ファイルからオプションを読み取る。
- サービス名の後ろに --defaults-file オプションを指定すると、サーバはデフォルトのオプション設定ファイルを無視し、指定ファイルの [mysqld] グループだけからオプションを読み取る。

注意: 4.0.17 より前の MySQL バージョンでは、Windows サービスとしてインストールされたサーバのパス名やサービス名にスペースが含まれていると、起動時に問題が発生します。したがって、C:\Program Files などのディレクトリに MySQL をインストールしたり、スペースを含んだサービス名を使用したりすることは避けてください。

通常どおりに --install を使用してサーバをインストールした場合は、サーバは MySQL のサービス名でインストールされます。

さらに複雑な例として、以下のコマンドを考察してみます。

```
shell> C:\mysql\bin\mysqld --install mysql --defaults-file=C:\my-opts.cnf
```

ここでは、--install オプションの後ろにサービスを指定しています。このコマンドで --defaults-file オプションを指定しないと、サーバは標準のオプション設定ファイルから [mysql] グループを読み取ります (このオプショングループは mysql クライアントプログラムが使用するものであるため、これは推奨できません)。ただし、--defaults-file オプションを指定しているため、サーバは、指定ファイルの [mysqld] オプショングループだけからオプションを読み取ります。

MySQL サービスを起動する前に、Windows サービス ユーティリティで ``Start parameters'' としてオプションを指定することもできます。

MySQL サーバをサービスとしてインストールすると、Windows が起動するたびに MySQL サービスが自動的に起動されます。サービスは、サービス ユーティリティから直接起動するか、コマンド NET START MYSQL を使用して起動することもできます。NET コマンドは、ケース依存ではありません。

注意: mysqld は、サービスとして実行されるとコンソールウィンドウにアクセスできないので、コンソールウィンドウにメッセージは表示されません。mysqld が起動しない場合は、エラーログをチェックして、問題の原因を示すメッセージが書き込まれていないかどうかを確認してください。 エラーログは、C:\mysql\data ディレクトリにあります。これは、.err というサフィックスが付いたファイルです。

サービスとして稼動している mysqld は、サービス ユーティリティ、NET STOP MYSQL コマンド、または mysqladmin shutdown コマンドを使用して停止することができます。Windows がシャットダウンするときにサービスが稼動している場合は、Windows によってサーバが自動的に停止されます。

MySQL バージョン 3.23.44 からは、ブートプロセス中にサービスを自動的に起動させたくない場合、Manual サービスとしてサーバをインストールすることができます。その場合は、以下のように、--install オプションではなく、--install-manual オプションを使用します。

```
shell> C:\mysql\bin\mysqld --install-manual
```

サービスとしてインストールされたサーバを削除するには、サーバが稼動している場合は、まずサーバを停止します。次

に、--remove オプションを使用して、サーバを削除します。

```
shell> mysqld --remove
```

MySQL サービス自動シャットダウンの 1 つの問題は、3.23.49 より前のバージョンの MySQL の場合、Windows がシャットダウンの完了まで数秒間しか待機せず、この制限時間を超過するとデータベースサーバプロセスが強制終了されるという点です。これによって、問題が発生することがありました (たとえば、InnoDB ストレージエンジンは次回の起動時にクラッシュリカバリを実行する必要があるなど)。MySQL バージョン 3.23.49 以降は、MySQL サーバのシャットダウンの完了までの Windows の待機時間が延長されています。この待機時間でもインストールに十分でないと思われる場合、最も安全なのは、MySQL サーバをサービスとして実行しないことです。代わりに、MySQL サーバの起動はコマンドラインプロンプトから行い、停止は mysqladmin shutdown を使用して行ってください。

MySQL サーバの停止時に待機時間を延長するよう Windows に指示するための変更は、Windows 2000 および XP には有効ですが、Windows NT には有効ではありません。Windows NT の場合、サービスのシャットダウンのための待機時間は 20 秒だけで、この時間が経過すると、サービスプロセスは強制終了されます。レジストリエディタ \winnt\system32\regedt32.exe をたち上げ、レジストリツリー内の HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control の WaitToKillServiceTimeout の値を編集して、このデフォルト値を増やすことができます。デフォルト値より大きい新しい値をミリ秒単位で指定します (たとえば、値 120000 を指定すると、Windows NT は最大で 120 秒間待機します)。

mysqld をサービスとして起動しない場合は、NT ベース以外の Windows のバージョンの場合と同様にコマンドラインから起動することができます。手順については、項2.1.1.6.「Windows 95、98、または Me での MySQL の起動」 を参照してください。

2.1.1.8. Windows での MySQL の実行

MySQL は、すべての Windows プラットフォームで TCP/IP をサポートします。mysqld-nt サーバと mysql-max-nt サーバは、NT、2000、および XP で名前付きパイプをサポートします。 ただし、デフォルトでは、プラットフォームに関係なく TCP/IP が使用されます。

- 多くの Windows 設定では、名前付きパイプは、事実上 TCP/IP より低速である。
- 名前付きパイプを使用した場合、MySQL サーバをシャットダウンする際に問題が発生することがある。

3.23.50 以降のバージョンの mysqld-nt および mysql-max-nt の場合、--enable-named-pipe オプションを使用してこれらのサーバを起動したときにのみ名前付きパイプが有効になります。

--pipe オプションを指定するか、ホスト名として . (ピリオド) を指定して、MySQL クライアントに強制的に名前付きパイプを使用させることができます。パイプの名前は、--socket オプションを使用して指定します。MySQL 4.1 では、--protocol=PIPE オプションを使用してください。

以下のいずれかのコマンドを実行して、MySQL サーバが動作しているかどうかをテストすることができます。

```
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
```

Windows 9x/Me 上で、接続に対する mysqld の応答が遅い場合は、使用している DNS に問題がある可能性があります。

その場合は、--skip-name-resolve オプションを使用して mysqld を起動し、MySQL 権限テーブルの Host カラムの localhost と IP 番号だけを使用します。

MySQL コマンドラインツールには以下の 2 つのバージョンがあります。

バイナリ	説明
mysql	ネイティブ Windows 上にコンパイルされ、限られたテキスト編集機能を提供する。
mysqlc	Cygnus GNU コンパイラとライブラリを組み込んでコンパイルされ、readline の編集機能を提供する。

mysqlc を使用する場合は、mysqlc が検出できる場所に cygwinb19.dll ライブラリのコピーをインストールする必要があります。MySQL の最新のディストリビューションでは、mysqlc と同じディレクトリにこのライブラリが含まれています (ご使用の MySQL インストールの基本ディレクトリの下の bin ディレクトリ) 。ご使用のディストリビューションの bin ディレクトリに cygwinb19.dll ライブラリがない場合は、lib ディレクトリ内でこのライブラリを探し、Windows システムディレクトリ (\Windows\system か、これに類似した場所) にコピーします。

Windows のデフォルトの権限の設定では、ローカルユーザーは、パスワードなしで全データベースに対して全ての操作が行えるようになっています。MySQL をより安全にするために、すべてのユーザにパスワードを設定し、Host='localhost' と User='' が記述された mysql.user テーブルのレコードを削除してください。

また、root ユーザのパスワードも追加してください。以下の例では、まず全権限を持つ匿名ユーザを削除し、次に root ユーザパスワードを設定します。

```
C:\> C:\mysql\bin\mysql mysql
mysql> DELETE FROM user WHERE Host='localhost' AND User='';
mysql> FLUSH PRIVILEGES;
mysql> QUIT
C:\> C:\mysql\bin\mysqladmin -u root password your_password
```

パスワードを設定した後で mysqld サーバをシャットダウンする場合は、このコマンドを使用してシャットダウンすることができます。

```
C:\> mysqladmin --user=root --password=your_password shutdown
```

MySQL バージョン 3.21 の古い Windows シェアウェアディストリビューションのサーバを使用している場合、パスワードを設定するための mysqladmin コマンドはparse error near 'SET password' というエラーによって失敗します。この問題は、このバージョンより新しい MySQL にアップグレードすることで解決されます。

最新バージョンの MySQL では、GRANT コマンドと REVOKE コマンドを使用して、新しいユーザの追加と特権の変更を簡単に行うことができます。 See 項4.4.1.「GRANT および REVOKE の構文」。

2.1.2. Linux への MySQL のインストール

Linux に MySQL をインストールする場合は、RPM パッケージを使用することをお勧めします。MySQL RPM は、現在、SuSE Linux 7.3 システム上でビルドされていますが、rpm をサポートし glibc を使用するほとんどのバージョンの Linux 上で動作します。

RPM ファイルに関する問題が発生した場合 (たとえば、``Sorry, the host 'xxxx' could not be looked up'' というエラーが表示された場合) は、項2.6.2.1.「Linux の注意事項 (バイナリディストリビューション) 」 を参照してください。

ほとんどの場合、MySQL-server パッケージと MySQL-client パッケージをインストールするだけで済みます。最小インストールの場合は、その他のパッケージは必要ありません。 追加機能を備えた MySQL Max サーバを実行したい場合は、MySQL-server RPM をインストールした後に MySQL-Max RPM をインストールしてください。 See 項4.8.5.「mysqld-max (拡張 mysqld サーバ) 」。

MySQL 4.0 パッケージのインストール中に依存関係エラーが発生した場合 (たとえば、``error: removing these packages would break dependencies: libmysqlclient.so.10 is needed by ...") 、MySQL-shared-compat パッケージもインストールする必要があります。このパッケージには、下位互換性のために両方の共有ライブラリ (MySQL 4.0 の libmysqlclient.so.12 と MySQL 3.23 の libmysqlclient.so.10) が含まれています。

多くの Linux ディストリビューションにはまだ MySQL 3.23 が同梱されており、アプリケーションを動的にリンクしてディスク領域を節約しています。これらの共有ライブラリが別個のパッケージ (MySQL-shared など) に含まれている場合は、このパッケージをインストールして、MySQL サーバおよびクライアントのパッケージをアップグレードするだけで十分です (これらのパッケージは静的にリンクされ共有ライブラリに依存しません) 。MySQL サーバと同じパッケージに共有ライブラリが含まれているディストリビューションの場合 (Red Hat Linux など) は、バージョン 3.23 の MySQL-shared RPM をインストールするか、MySQL-shared-compat パッケージを使用します。

以下の RPM パッケージが用意されています。

- MySQL-server-VERSION.i386.rpm

  MySQL サーバ。別のマシン上で稼動している MySQL サーバに接続するだけの場合を除いて、このパッケージが必要。注意: MySQL 4.0.10 より前のバージョンでは、このパッケージは MySQL-VERSION.i386.rpm と呼ばれていた。

- MySQL-Max-VERSION.i386.rpm

  MySQL Max サーバ。このサーバには、MySQL-server RPM のサーバには含まれていない追加機能がある。MySQL-Max RPM は MySQL-server RPM に依存するので、まず MySQL-server RPM をインストールしておく必要がある。

- MySQL-client-VERSION.i386.rpm

  標準 MySQL クライアントプログラム。ほとんどの場合、このパッケージをインストールする必要がある。

- MySQL-bench-VERSION.i386.rpm

  テストとベンチマーク。Perl と DBD-mysql モジュールを必要とする。

- MySQL-devel-VERSION.i386.rpm

  Perl モジュールなど、その他の MySQL クライアントをコンパイルする場合に必要なライブラリとインクルードファイル。

- MySQL-shared-VERSION.i386.rpm

  このパッケージには、特定の言語とアプリケーションが MySQL を動的にロードして使用するために必要な共有ライブラリ (libmysqlclient.so*) が含まれている。

- MySQL-shared-compat-VERSION.i386.rpm

  このパッケージには、MySQL 3.23 と MySQL 4.0 の両方の共有ライブラリが含まれている。MySQL 3.23 に動的にリンクされるアプリケーションをインストールしている場合に、ライブラリの依存関係を壊さずに MySQL 4.0 にアップ

グレードしたいときは、MySQL-shared の代わりにこのパッケージをインストールする。このパッケージは、MySQL 4.0.13 以降のバージョンで用意されている。

- MySQL-embedded-VERSION.i386.rpm

  組み込みの MySQL サーバライブラリ（MySQL 4.0 以降）

- MySQL-VERSION.src.rpm

  このパッケージには、これまでのすべてのパッケージのソースコードが含まれている。このパッケージを使用して、他のアーキテクチャ（Alpha や SPARC など）上で RPM を再構築することもできる。

RPM パッケージ（MySQL-server RPM など）内のすべてのファイルを確認するには、以下のコマンドを実行します。

```
shell> rpm -qpl MySQL-server-VERSION.i386.rpm
```

最小インストールを行うには、以下のコマンドを実行します。

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

クライアントパッケージだけをインストールするには、以下のコマンドを実行します。

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

RPM には、パッケージをインストールする前にその完全性と信頼性を検証する機能が備えられています。この機能の詳細については、項2.2.2.「MD5 チェックサムまたは GnuPG によるパッケージ完全性の検証」 を参照してください。

サーバ RPM は、/var/lib/mysql ディレクトリの下にデータを配置します。また、ブート時にサーバを自動的に起動するための適切なエントリを /etc/init.d/ に作成します（したがって、旧バージョンのインストールを実行しており、起動スクリプトに変更を加えている場合、新しいバージョンの RPM のインストール時にそのスクリプトが失われないようにするためにはスクリプトのコピーを作成する必要があります）。システム起動時に MySQL が自動的に起動されるようにする方法については、項2.4.3.「MySQL を自動的に起動および停止する」 を参照してください。

/etc/init.d の初期化スクリプトをサポートしていない旧バージョンの Linux ディストリビューションに MySQL RPM をインストールする（直接または symlink を使用して）場合は、初期化スクリプトが実際にインストールされる場所を指すシンボリックリンクを作成してください。たとえば、その場所が /etc/rc.d/init.d である場合、RPM をインストールする前に以下のコマンドを使用して、その場所を指すシンボリックリンクとして /etc/init.d を作成します。

```
shell> cd /etc; ln -s rc.d/init.d .
```

ただし、/etc/init.d を使用するこの新しいディレクトリレイアウトは、LSB（Linux Standard Base）に準拠するために必要であるので、最新のすべての主要 Linux ディストリビューションではすでにサポートされています。

インストールする RPM ファイルに MySQL-server が含まれている場合は、インストール後に mysqld デーモンが起動します。 これで、MySQL の使用を開始できるようになりました。 See 項2.4.「インストール後の設定とテスト」。

何か問題があった場合は、バイナリインストールの章を参照してください。 See 項2.2.9.「MySQL バイナリディストリビューションのインストール」。

2.1.3. Mac OS X への MySQL のインストール

MySQL 4.0.11 からは、バイナリ tarball ディストリビューションではなく Mac OS X PKG バイナリパッケージを使用して、Mac OS X 10.2 (``Jaguar'') に MySQL をインストールできるようになりました。注意: このパッケージは、これより古いバージョンの Mac OS X (10.1.x など) はサポートしていません。

パッケージは、ディスクイメージ (.dmg) ファイル内にあります。まず、ファインダ内でこのファイルのアイコンをダブルクリックしてマウントする必要があります。ディスクイメージファイルをマウントすると、イメージがマウントされその内容が表示されます。

注意: インストールを続行する前に、MySQL Manager Application (Mac OS X Server 上で) を使用するかコマンドラインで mysqladmin shutdown を使用して、稼動しているすべての MySQL サーバインスタンスを必ずシャットダウンしてください。

MySQL PKG を実際にインストールするには、パッケージのアイコンをダブルクリックします。これにより、Mac OS Package Installer が起動されます。このインストーラの指示に従って MySQL のインストールを行います。

MySQL の Mac OS X PKG は、自身を /usr/local/mysql-<version> にインストールします。また、新しい場所を指すシンボリックリンク /usr/local/mysql もインストールします。/usr/local/mysql という名前のディレクトリがすでに存在する場合は、まずそのディレクトリの名前を /usr/local/mysql.bak に変更します。さらに、インストール後に mysql_install_db を実行して、mysql データベースに権限テーブルをインストールします。

インストールレイアウトは、バイナリディストリビューションのインストールレイアウトと類似しています。つまり、すべての MySQL バイナリが /usr/local/mysql/bin ディレクトリに配置されます。 MySQL ソケットファイルは、デフォルトで /tmp/mysql.sock として作成されます。 See 項2.2.5.「インストールレイアウト」。

MySQL インストールには、mysql という名前の Mac OS X ユーザアカウントが必要です (Mac OS X 10.2 以降の場合、この名前を持つユーザアカウントがデフォルトで存在します)。

Mac OS X Server を実行している場合、MySQL はすでにインストールされています。

- Mac OS X Server 10.2 〜 10.2.2 には MySQL 3.23.51 がインストールされている。
- Mac OS X Server 10.2.3 〜 10.2.6 には MySQL 3.23.53 が同梱されている。
- Mac OS X Server 10.3 には MySQL 4.0.14 が同梱されている。

ここでは、公式の MySQL Mac OS X PKG のインストールについてのみ説明します。必ず、MySQL のインストールに関する Apple のヘルプを参照してください (``Help View'' アプリケーションを実行して、``Mac OS X Server'' ヘルプを選択し、``MySQL'' を検索して、``Installing MySQL'' という項目を参照してください)。

特に、Mac OS X Server 上のプリインストール版の MySQL は、mysqld_safe コマンドではなく safe_mysqld コマンドで起動することに注意してください。

http://www.entropy.ch から入手した Marc Liyanage の Mac OS X 用 MySQL パッケージを使用している場合は、同氏のページに示されているバイナリインストールレイアウトによるパッケージの更新手順説明に従うだけで済みます。

Marc の 3.23.xx バージョンまたは Mac OS X Server バージョンの MySQL から公式の MySQL PKG にアップグレードする場合は、いくつかの新しいセキュリティ特権が追加されているので、既存の MySQL 特権テーブルを現在の形式に変換する必要もあります。 See 項2.5.6.「権限テーブルのアップグレード」。

また、システムのブートアップ時に MySQL を自動的に起動する場合は、MySQL Startup Item をインストールする必要があります。MySQL 4.0.15 からは、MySQL Startup Item は Mac OS X インストールディスクイメージの一部として別個のインストールパッケージとなっています。MySQLStartupItem.pkg アイコンをダブルクリックし、指示に従ってインストールします。

注意: このインストールは 1 回だけ行います。MySQL パッケージをアップグレードするたびに Startup Item をインストールする必要はありません。

Mac OS X パッケージインストーラにバグがあるため、インストール先ディスク選択画面に You cannot install this software on this disk. (null) というエラーメッセージが表示されることがあります。このエラーが発生した場合は、Go Back ボタンをクリックして前の画面に戻ります。次に、Continue をクリックして、もう一度インストール先ディスク選択画面に進み、適切なインストール先ディスクを選択することができます。この問題は、Apple に報告済みで、現在同社で調査中です。

Startup Item は、/Library/StartupItems/MySQL にインストールされます。 これによって、システム設定ファイル /etc/hostconfig に変数 MYSQLCOM=-YES- が追加されます。MySQL の自動起動を無効にする場合は、この変数を MYSQLCOM=-NO- に変更します。

Mac OS X Server では、Startup Item インストールスクリプトが自動的に /etc/hostconfig の変数 MYSQL を MYSQL=-NO- に変更して、デフォルト MySQL インストールの起動を無効にします。これは、ブートアップ時のコンフリクトを回避するために行われます。ただし、すでに稼動している MySQL サーバがシャットダウンされることはありません。

インストール後は、ターミナルウィンドウで以下のコマンドを実行して MySQL を起動することができます。この作業を行うには管理者特権が必要であることに注意してください。

Startup Item をインストールした場合は、以下のコマンドを入力します。

```
shell> sudo /Library/StartupItems/MySQL/MySQL start
(必要ならパスワードを入力)
(Control-D か "exit" を入れてシェルから抜ける)
```

Startup Item を使用しない場合は、以下のコマンドシーケンスを入力します。

```
shell> cd /usr/local/mysql
shell> sudo ./bin/mysqld_safe
(必要ならパスワードを入力)
(Control-Z)
shell> bg
(Control-D か "exit" を入れてシェルから抜ける)
```

これで、/usr/local/mysql/bin/mysql などを実行して MySQL サーバに接続することができるようになりました。

初めて MySQL をインストールした場合は、必ず、MySQL root ユーザのパスワードを設定してください。

パスワードの設定は、以下の 2 つのコマンドを使用して行います。

```
/usr/local/mysql/bin/mysqladmin -u root password <password>
/usr/local/mysql/bin/mysqladmin -u root -h `hostname` password <password>
```

2 行目の hostname コマンドは、必ず、バッククォート (`) で囲んでください。そうすることで、シェルは、このコマンドをその出力 (このシステムのホスト名) に置き換えることができます。

以下のようにシェルのリソースファイルにエイリアスを追加して、コマンドラインから mysql および mysqladmin にアクセスすることもできます。

```
alias mysql '/usr/local/mysql/bin/mysql'
alias mysqladmin '/usr/local/mysql/bin/mysqladmin'
```

あるいは、たとえば以下の行を $HOME/.tcshrc に追加して、$PATH 環境変数に /usr/local/mysql/bin を追加することができます。

```
setenv PATH ${PATH}:/usr/local/mysql/bin
```

新しい MySQL PKG をインストールしても既存のインストールのディレクトリは削除されないことに注意してください。Mac OS X インストーラには、以前にインストールされたパッケージを適切にアップグレードするために必要な機能はまだ備わっていません。

旧バージョンの MySQL データベースファイルをコピーし、新しいバージョンを正常に起動した後、ディスク領域節約のために旧バージョンのインストールファイルを削除することをお勧めします。 さらに、/Library/Receipts/mysql-<version>.pkg にある旧バージョンの Package Receipt ディレクトリも削除してください。

2.1.4. NetWare への MySQL のインストール

バージョン 4.0.11 以降、Novell NetWare 向け MySQL サーバがバイナリパッケージ形式で提供されています。NetWare サーバは、MySQL をホストするために以下の要件を満たしていなければなりません。

- NetWare バージョン 6.5 または Support Pack 3 がインストールされた NetWare 6.0（Support Pack 3は http://support.novell.com/filefinder/13659/index.html で入手できる）。 システムは、各バージョンの NetWare を実行するための Novell の必要条件を満たしている必要がある。
- MySQL データとバイナリ自体が NSS ボリュームにインストールされていること（従来のボリュームはサポートされていない）。

NetWare 用バイナリパッケージは、http://www.mysql.com/downloads/ で入手できます。

NetWare 6.0 で MySQL を実行する場合は、コマンドラインで --skip-external-locking オプションを指定することを強くお勧めします。また、myisamchk は外部ロックを利用するので、myisamchk ではなく CHECK TABLE および REPAIR TABLE を使用する必要もあります。 NetWare 6.0 では、外部ロックに関する既知の問題があります。この問題は NetWare 6.5 では解決されています。

2.1.4.1. NetWare 用 MySQL バイナリのインストール

1. 既存のインストールからアップグレードする場合は、MySQL サーバを停止する。 MySQL サーバの停止は、サーバコンソールから以下のコマンドを使用して行う。

   ```
   SERVER:  mysqladmin -u root shutdown
   ```

2. MySQL のインストール場所にアクセスできるクライアントマシンから目的のサーバにログオンする。

3. バイナリパッケージ zip ファイルをサーバに抽出する。必ず、zip ファイルのパスを使用できるようにする。ファイル

を単純に SYS:\ に抽出しても問題はない。

既存のインストールからアップグレードする場合は、ここで my.cnf のほかにデータディレクトリ（SYS:MYSQL\DATA など）をコピーする必要がある場合がある（データディレクトリをカスタマイズしている場合）。その後、旧バージョンの MySQL のコピーを削除することができる。

4. 必要に応じて、ディレクトリの名前をより一貫性のある使いやすい名前に変更する。SYS:MYSQL という名前の使用をお勧めする（このマニュアルの例では、一般にインストールディレクトリを指すのにこの名前を使用している）。

5. サーバコンソールで、MySQL NLM が格納されているディレクトリの検索パスを追加する。たとえば、以下のように入力する。

```
SERVER:  SEARCH ADD SYS:MYSQL\BIN
```

6. 必要に応じて、サーバコンソールで mysql_install_db を実行して初期データベースをインストールする。

7. サーバコンソールで mysqld_safe を使用して MySQL サーバを起動する。

8. インストールを完了するには、autoexec.ncf に以下のコマンドも追加する必要がある。たとえば、MySQL インストールが SYS:MYSQL にあり、MySQL を自動的に起動する場合は、以下の行を追加する。

```
#Starts the MySQL 4.0.x database server
SEARCH ADD SYS:MYSQL\BIN
MYSQLD_SAFE
```

NetWare 6.0 を使用している場合は、--skip-external-locking フラグを追加する。

```
#Starts the MySQL 4.0.x database server
SEARCH ADD SYS:MYSQL\BIN
MYSQLD_SAFE --skip-external-locking
```

サーバ上に既存の MySQL インストールがある場合は、必ず、autoexec.ncf に既存の MySQL 起動コマンドがないかチェックして、必要に応じてそれらの起動コマンドを編集または削除する。

2.2. インストール関連の一般的な問題

2.2.1. MySQL の入手方法

最新のバージョンとダウンロード手順については、MySQL のホームページ（http://www.mysql.com/）を参照してください。

主要ミラーは、http://mirrors.sunsite.dk/mysql/ にあります。

MySQL Web/ダウンロードミラーの最新の完全な一覧については、http://www.mysql.com/downloads/mirrors.html を参照してください。 このサイトでは、MySQL ミラーサイトになる方法と不適切なミラーや無効なミラーの報告方法についての情報も提供しています。

2.2.2. MD5 チェックサムまたは GnuPG によるパッケージ完全性の検証

目的に合った MySQL パッケージをダウンロードした後、インストールする前に、そのパッケージが破損していないこと、そして改ざんされていないことを確認する必要があります。

MySQL AB は、MD5 checksums と、GnuPG による暗号化署名である GNU Privacy Guard の 2 種類の完全性チェック手段を提供しています。

MD5 Checksum の検証

パッケージをダウンロードした後、MD5 チェックサムが MySQL ダウンロードページに記載されているものと一致しているかどうかを確認してください。各パッケージには個別のチェックサムがあります。以下のコマンドを使用してチェックサムを検証することができます。

```
shell> md5sum <package>
```

注意: オペレーティングシステムによっては、md5sum コマンドがサポートされていないことがあります。一部のオペレーティングシステムでは、このコマンドを md5 という名前で呼んでいます。また、このコマンドをまったく提供していないオペレーティングシステムもあります。Linux では、このコマンドはさまざまなプラットフォームで使用できる GNU Text Utilities パッケージに組み込まれています。http://www.gnu.org/software/textutils/ からソースコードを入手することもできます。OpenSSL をインストールしている場合は、このコマンドの代わりに openssl md5 <package> コマンドを使用することもできます。md5 コマンドの DOS/Windows 実装は、http://www.fourmilab.ch/md5/ から入手できます。

例:

```
shell> md5sum mysql-standard-4.0.10-gamma-pc-linux-i686.tar.gz
155836a7ed8c93aee6728a827a6aa153
            mysql-standard-4.0.10-gamma-pc-linux-i686.tar.gz
```

結果のチェックサムが、ダウンロードページで各パッケージの直下に記載されているものと一致しているかどうかを確認します。

GnuPG による署名チェック

パッケージの完全性を検証するためのより信頼性の高い方法は、暗号化署名を使用する方法です。MySQL AB は、GNU Privacy Guard (GnuPG) を使用します。これは、Phil Zimmermann が開発した非常に有名な Pretty Good Privacy (PGP) に代わる オープンソースのソフトウェアです。 OpenPGP/GnuPG の詳細と、GnuPG の入手方法およびご使用のシステムへのインストール方法については、http://www.gnupg.org/ および http://www.openpgp.org/ を参照してください。ほとんどの Linux ディストリビューションには、デフォルトで GnuPG がインストールされています。

MySQL 4.0.10 (2003 年 2 月) 以降、MySQL AB はダウンロード可能パッケージに GnuPG による署名を付けています。暗号化署名は、ファイルの完全性と信頼性を検証するためのより信頼性の高い方法です。

特定のパッケージの署名を検証するには、まず MySQL AB の 公開 GPG ビルドキー <build@mysql.com> のコピーを入手する必要があります。以下に記載したものを直接カットアンドペーストするか、http://www.keyserver.net/ から入手してください。

```
Key ID:
pub  1024D/5072E1F5 2003-02-03
    MySQL Package signing key (www.mysql.com) <build@mysql.com>
Fingerprint: A4A9 4068 76FC BD3C 4567  70C8 8C71 8D3B 5072 E1F5

Public Key (ASCII-armored):
```

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.0.6 (GNU/Linux)
Comment: For info see http://www.gnupg.org

mQGiBD4+owwRBAC14GIfUfCyEDSIePvEW3SAFUdJBtoQHH/nJKZyQT7h9bPlUWC3
RODjQReyCITRrdwyrKUGku2FmeVGwn2u2WmDMNABLnpprWPkBdCk96+OmSLN9brZ
fw2vOUgCmYv2hW0hyDHuvYlQA/BThQoADgj8AW6/0Lo7V1W9/8VuHP0gQwCgvzV3
BqOxRznNCRCRxAuAuVztHRcEAJooQK1+iSiunZMYD1WufeXfshc57S/+yeJkegNW
hxwR9pRWVArNYJdDRT+rf2RUe3vpquKNQU/hnEIUHJRQqYHo8gTxvxXNQc7fJYLV
K2HtkrPbP72vwsEKMYhhr0eKCbtLGfls9krjJ6sBgACyP/Vb7hiPwxh6rDZ7ITnE
kYpXBACmWpP8NJTkamEnPCia2ZoOHODANwpUkP43I7jsDmgtobZX9qnrAXw+uNDI
QJEXM6FSbi0LLtZciNIYsafwAPEOMDKpMqAK6IyisNtPvaLd8lH0bPAnWqcyefep
rv0sxxqUEMcM3o7wwgfN83POkDasDbs3pjwPhxvhz6//62zQJ7Q7TXlTUUwgUGFj
a2FnZSBzaWduaW5nIGtleSAod3d3Lm15c3FsLmNvbSkgPGJ1aWxkQG15c3FsLmNv
bT6IXQQTEQIAHQUCPj6jDAUJCWYBgAULBwoDBAMVAwIDFgIBAheAAAoJEIxxjTtQ
cuH1cY4AnilUwTXn8MatQOiG0a/bPxrvK/gCAJ4oinSNZRYTnblChwFaazt7PF3q
zIhMBBMRAgAMBQI+PqPRBYMJZgC7AAoJEEIQ4SqycpHyJOEAn1mxHijft00bKXvu
cSo/pECUmppiAJ41M9MRVj5VcdH/KN/KjRtW6tHFPYhMBBMRAgAMBQI+QoIDBYMJ
YiKJAAoJELb1zU3GuiQ/lpEAoIhpp6BozKI8p6eaabzF5MlJH58pAKCu/ROofK8J
Eg2aLos+5zEYrB/LsrkCDQQ+PqMdEAgA7+GJfxbMdY4wslPnjH9rF4N2qfWsEN/l
xaZoJYc3a6M02WCnHl6ahT2/tBK2w1QI4YFteR47gCvtgb6O1JHffOo2HfLmRDRi
Rjd1DTCHqeyX7CHhcghj/dNRlW2Z0l5QFEcmV9U0Vhp3aFfWC4Ujfs3LU+hkAWzE
7zaD5cH9J7yv/6xuZVw411x0h4UqsTcWMu0iM1BzELqX1DY7LwoPEb/O9Rkbf4fm
Le11EzIaCa4PqARXQZc4dhSinMt6K3X4BrRsKTfozBu74F47D8IIbf5vSYHbuE5p
/1oIDznkg/p8kW+3FxuWrycciqFTcNz215yyX39LXFnlLzKUb/F5GwADBQf+Lwqq
a8CGrRfsOAJxim63CHfty5mUc5rUSnTslGYEIOCR1BeQauyPZbPDsDD9MZ1ZaSaf
anFvwFG6Llx9xkU7tzq+vKLoWkm4u5xf3vn55VjnSd1aQ9eQnUcXiL4cnBGoTbOW
l39EcyzgslzBdC++MPjcQTcA7p6JUVsP6oAB3FQWg54tuUo0Ec8bsM8b3Ev42Lmu
QT5NdKHGwHsXTPtl0klk4bQk4OajHsiy1BMahpT27jWjJlMiJc+IWJ0mghkKHt92
6s/ymfdf5HkdQ1cyvsz5tryVl3Fx78XeSYfQvuuwqp2H139pXGEkg0n6KdUOetdZ
Whe70YGNPw1yjWJT1IhMBBgRAgAMBQI+PqMdBQkJZgGAAAoJEIxxjTtQcuH17p4A
n3r1QpVC9yhnW2cSAjq+kr72GX0eAJ4295kl6NxYEuFApmr1+0uUq/SlsQ==
=YJkx
-----END PGP PUBLIC KEY BLOCK-----
```

このキーは、gpg --import を使用して公開 GPG キーリングにインポートすることができます。公開キーの処理方法については、GPG のマニュアルを参照してください。

公開ビルドキーをダウンロードしてインポートしたら、必要な MySQL パッケージと、それに対応する署名 (これもダウンロードページから入手できます) をダウンロードします。 署名には、.asc というファイル名拡張子が付いています。たとえば、mysql-standard-4.0.10-gamma-pc-linux-i686.tar.gz の署名は、mysql-standard-4.0.10-gamma-pc-linux-i686.tar.gz.asc です。 パッケージと署名の両方のファイルが同じディレクトリに格納されていることを確認してから、以下のコマンドを実行して、パッケージファイルの署名を検証します。

```
shell> gpg --verify <package>.asc

Example:

shell> gpg --verify mysql-standard-4.0.10-gamma-pc-linux-i686.tar.gz.asc
gpg: Warning: using insecure memory!
gpg: Signature made Mon 03 Feb 2003 08:50:39 PM MET using DSA key ID 5072E1F5
gpg: Good signature from
    "MySQL Package signing key (www.mysql.com) <build@mysql.com>"
```

"Good signature" というメッセージが表示されたら、問題ありません。

RPM による署名チェック

RPM パッケージの場合、別個の署名はありません。RPM パッケージには GPG 署名と MD5 チェックサムが組み込まれています。この署名と MD5 チェックサムは、以下のコマンドを実行して検証できます。

```
shell> rpm --checksig <package>.rpm

Example:

shell> rpm --checksig MySQL-server-4.0.10-0.i386.rpm
MySQL-server-4.0.10-0.i386.rpm: md5 gpg OK
```

注意: RPM 4.1 を使用している場合、GPG 公開キーリングにインポートしたにもかかわらず、(GPG) NOT OK (MISSING KEYS: GPG#5072e1f5) というメッセージが表示されたら、まず RPM キーリングにキーをインポートする必要があります。RPM 4.1 では GPG キーリング (および GPG 自体) の使用が中止され、自身のキーリングが維持されています (これは、RPM がシステムワイドなアプリケーションであり、GPG 公開キーリングはユーザ固有のファイルであるためです) 。MySQL 公開キーを RPM キーリングにインポートする場合は、以下のコマンドを使用してください。

```
shell> rpm --import <pubkey>

Example:

shell> rpm --import mysql_pubkey.asc
```

MD5 チェックサムまたは GPG 署名が一致しない場合は、まず別のミラーサイトからパッケージをもう一度ダウンロードしてみてください。再度、パッケージの完全性を検証できなかった場合は、そのパッケージの完全名と利用したダウンロードサイトを記載して、電子メールで当社にお知らせください (アドレスは <webmaster@mysql.com> または <build@mysql.com>) 。

2.2.3. MySQL がサポートしているオペレーティングシステム

当社では GNU Autoconf を使用しているので、正常に機能する Posix スレッドおよび C++ コンパイラによってすべての最新のシステムに MySQL を移植することができます (クライアントコードだけをコンパイルする場合は、C++ コンパイラは必要ですがスレッドは不要です) 。当社では、主に Linux (SuSE および Red Hat) 、FreeBSD、および Sun Solaris (バージョン 8 および 9) 上で MySQL ソフトウェアを使用および開発しています

多くのオペレーティングシステムでは、ネイティブスレッドは最新バージョンのOSでのみ有効であることに注意してください。MySQL は、以下のオペレーティングシステムとスレッドパッケージの組み合わせで正常にコンパイルすることが報告されています。

- AIX 4.x、5.x とネイティブスレッド。 See 項2.6.6.4.「IBM-AIX の注意事項」。
- Amiga
- BSDI 2.x と MIT-pthreads パッケージ。 See 項2.6.4.5.「BSD/OS バージョン 2.x の注意事項」。
- BSDI 3.0、3.1、および 4.x とネイティブスレッド。 See 項2.6.4.5.「BSD/OS バージョン 2.x の注意事項」。
- SCO OpenServer と最新の移植版の FSU Pthreads パッケージ。 See 項2.6.6.9.「SCO の注意事項」。
- SCO UnixWare 7.1.x。See 項2.6.6.10.「SCO UnixWare バージョン 7.1.x の注意事項」。

- DEC Unix 4.x とネイティブスレッド。 See 項2.6.6.6.「Alpha-DEC-UNIX (Tru64) の注意事項」。
- FreeBSD 2.x と MIT-pthreads パッケージ。 See 項2.6.4.1.「FreeBSD の注意事項」。
- FreeBSD 3.x および 4.x とネイティブスレッド。 See 項2.6.4.1.「FreeBSD の注意事項」。
- FreeBSD 4.x と Linuxthreads。 See 項2.6.4.1.「FreeBSD の注意事項」。
- HP-UX 10.20 と DCE スレッドまたは MIT-pthreads パッケージ。 See 項2.6.6.2.「HP-UX バージョン 10.20 の注意事項」。
- HP-UX 11.x とネイティブスレッド。 See 項2.6.6.3.「HP-UX バージョン 11.x の注意事項」。
- Linux 2.0+ と LinuxThreads 0.7.1+ または glibc 2.0.7+。 See 項2.6.2.「Linux の注意事項 (すべての Linux バージョン) 」。
- Mac OS X。See 項2.6.5.「Mac OS X の注意事項」。
- NetBSD 1.3/1.4 Intel および NetBSD 1.3 Alpha (GNU make が必要) 。 See 項2.6.4.2.「NetBSD の注意事項」。
- Novell NetWare 6.0。 See 項2.6.8.「Novell NetWare の注意事項」。
- OpenBSD 2.5 以降のバージョンとネイティブスレッド。OpenBSD 2.5 より前のバージョンと MIT-pthreads パッケージ。 See 項2.6.4.3.「OpenBSD 2.5 の注意事項」。
- OS/2 Warp 3,FixPack 29 および OS/2 Warp 4, FixPack 4。See 項2.6.7.「OS/2 の注意事項」。
- SGI Irix 6.x とネイティブスレッド。 See 項2.6.6.8.「SGI Irix の注意事項」。
- Solaris 2.5 以降と SPARC および x86 上のネイティブスレッド。 See 項2.6.3.「Solaris の注意事項」。
- SunOS 4.x と MIT-pthreads パッケージ。 See 項2.6.3.「Solaris の注意事項」。
- Tru64 Unix
- Windows 9x、Me、NT、2000、および XP。 See 項2.6.1.「Windows の注意事項」。

注意: すべてのシステムが、MySQL の実行に同じように適しているとは限りません。特定のプラットフォームが高負荷でミッションクリティカルな MySQL サーバにどのぐらい適しているかは、以下の要因によって決まります。

- スレッドライブラリの全般的な安定性。他の点で高い評価を得ているプラットフォームでも、MySQL が呼び出すスレッドライブラリのコードが不安定である場合は、それ以外のすべての点で完璧であっても、MySQL の動作は不安定になる。
- カーネルおよびスレッドライブラリまたはそのいずれかがマルチプロセッサシステム上で SMP を活用できるかどうか。つまり、プロセスがスレッドを作成した場合に、そのスレッドは元のプロセスとは異なる CPU 上で動作できなければならない。
- カーネル/スレッドライブラリが、 過剰なコンテキストスイッチを行うことなく、 相互排他ロック(mutex)を頻繁に取得/解放する多くのスレッドを、実行できるかどうか。 つまり、pthread_mutex_lock() の実装が CPU 時間を生み出すことに重点を置きすぎている場合、そのことで MySQL のパフォーマンスは大きく損なわれることになる。この問題に対処しないと、CPU をマシンに追加すると MySQL の処理速度が実質的に遅くなる。

- ファイルシステムの全般的な安定性とパフォーマンス
- ファイルシステムがテーブルが大きい場合に大きなファイルを効率的に処理できるかどうか。
- プラットフォームに関する MySQL AB の専門知識のレベル。MySQL AB は、熟知しているプラットフォームについては、コンパイル時に有効になるプラットフォーム固有の最適化や修正を導入する。また、MySQL 向けにシステム設定を最適化する方法について助言することもできる。
- 似たような設定をした環境に対して、MySQL AB が社内で行ったテストの量。
- 目的のプラットフォーム上で MySQL を問題なく実行したユーザの数。この数が多ければ、プラットフォーム固有の予期せぬ問題が発生する可能性が非常に低くなる。

以上の基準から、現時点で MySQL を実行するのに最適なプラットフォームは、SuSE Linux 8.2、2.4 カーネル、および ReiserFS（または類似の Linux ディストリビューション）を搭載した x86 と、Solaris（2.7 〜 9）を搭載した SPARC です。FreeBSD は第 3 位ですが、スレッドライブラリが改善されればベストグループに仲間入りするはずです。また、MySQL がコンパイルされ、正常に動作する（安定性とパフォーマンスのレベルがまったく同じではないにしても）他のすべてのプラットフォームをいつの日かベストグループに含めることができるようになることを当社は切に希望しています。そのためには、MySQL が依存する OS やライブラリの開発者と協力して当社が努力する必要があります。それらのコンポーネントのいずれかを改善することに興味があり、その開発に影響を与える立場にあり、MySQL の動作を改善するために必要な事柄についての詳細な説明が必要なユーザは、MySQL の社内メーリングリストに電子メールをお送りください。See 項1.7.1.1.「MySQL メーリングリスト」。

上記の比較は、各 OS の全般的な優劣をつけるためのものではありません。ここでは、特定用途（つまり MySQL の実行）のために特定の OS を選択することに焦点をあてており、その点に関してのみプラットフォームを比較しています。これを念頭に置けば、この比較にさらに多くの論点を含めた場合は、結果は異なったものになります。ある OS が別の OS よりも優れている理由が、単に当社がそのプラットフォームのテストと最適化に他のプラットフォームよりも多くの労力を費やしたためだけであるということもあります。 ここでは、ご使用のセットアップ内で MySQL を使用するプラットフォームを決定する際に参考にしていただけるように当社の所見を述べているにすぎません。

2.2.4. 使用すべき MySQL のバージョン

最初に、最新の開発版リリースと最新の製品版（安定版）リリースのどちらを使用するかを決めます。

- 初めて MySQL を使用を開始する場合や、専用のバイナリディストリビューションがないシステムに MySQL を移植する場合は、通常は製品版リリース（現バージョンは 4.0）を選択することをお勧めする。 注意: すべての MySQL リリースは、リリース前に MySQL ベンチマークと広範なテストスィートによってチェックされている（開発版リリースの場合も同様）。
- それ以外の場合、既存の旧バージョンのシステムをアップグレードしたいが、シームレスでないアップグレードを使用するのが不安な場合は、使用している同じブランチを最新のものにアップグレードする（最新のバージョンが、使用しているものより新しい場合）。最新バージョンでは、重大なバグだけが修正され、比較的小さく安全な変更が行われている。

次に、ソースディストリビューションとバイナリディストリビューションのどちらを使用するかを決めます。ご使用のプラットフォーム用のバイナリディストリビューションがある場合、バイナリディストリビューションを使用してください。これは、一般的にソースディストリビューションよりもインストールが簡単なためです。

以下の場合は、ソースインストールを使用した方がうまく行く可能性があります。

- MySQL を明示的に指定した場所にインストールする場合（標準のバイナリディストリビューションは、任意の場所で``実行可能" だが、よりいっそうの柔軟性が得られる）。
- さまざまなユーザの要件を満たすことができるように、2 種類のバイナリバージョンが用意されている。1 つは非トランザクションストレージエンジンを組み込んでコンパイルされたバイナリ（高速で小さなバイナリ）で、もう 1 つはトランザクションセーフテーブルなどの最も重要な機能が組み込まれたバイナリである。どちらのバージョンも同じソースディストリビューションからコンパイルされている。すべての MySQL ネイティブクライアントは、両方の MySQL バージョンに接続できる。

  拡張 MySQL バイナリディストリビューションには、-max というサフィックスが付いており、mysqld-max と同じオプションが指定されている。 See 項4.8.5.「mysqld-max（拡張 mysqld サーバ）」。

  MySQL-Max RPM を使用する場合は、まず標準の MySQL-server RPM をインストールする必要がある。
- 標準のバイナリディストリビューションに含まれていない追加機能を mysqld に組み込む場合。以下に、最もよく使用される追加オプションの一覧を示す。
  - --with-innodb（MySQL 4.0 以降の場合はデフォルト）
  - --with-berkeley-db（使用できないプラットフォームもある）
  - --with-raid
  - --with-libwrap
  - --with-named-z-libs（一部のバイナリ用）
  - --with-debug[=full]
- デフォルトのバイナリディストリビューションは、通常、すべてのキャラクタセットのサポートを組み込んでコンパイルされ、同じプロセッサファミリの各種プロセッサ上で動作する。

  MySQL サーバをより高速化したい場合は、必要なキャラクタセットのサポートだけを組み込んで MySQL サーバを再コンパイルする必要がある。その際は、より適切なコンパイラ（pgcc など）を使用するか、使用しているプロセッサ向けにより最適化されたコンパイラオプションを使用する。
- バグを見つけて、MySQL 開発チームに報告すると、パッチが送付される。そのパッチをソースディストリビューションに適用して、バグを修正する必要がある。
- MySQL を構成する C および C++ のコードの読み取りや修正を行う場合は、ソースディストリビューションを入手する。ソースコードは、常に最高のマニュアルとなる。また、ソースディストリビューションには、バイナリディストリビューションよりも多くのテストやサンプルが含まれている。

MySQL の名前付けスキームでは、3 つの数字と 1 つのサフィックスで構成されるリリース番号が使用されます。たとえば、mysql-4.1.0-alpha のようなリリース名は、以下のように解釈されます。

- 最初の数字（4）は、メジャーバージョン番号であり、同時にファイル形式も表している。バージョン 4 の全リリースは同じファイル形式を持っている。

- 2 番目の数字 (1) はリリースレベルである。
- 3 番目の数字 (0) は、リリースレベル内のバージョン番号である。この数字は、新しいディストリビューションごとに増えていく。通常は、選択したリリースレベルの最新のバージョンが必要である。
- サフィックス (alpha) は、リリースの安定性のレベルを示す。 以下のようなサフィックスがある。
  - alpha は、そのリリースに 100% はテストされていない新しいコードが含まれていることを示している。既知のバグ (通常はない) は、「News」のセクションに記載される。See 付録 D. MySQL Change History。 ほとんどの alpha リリースには新しいコマンドと拡張機能も含まれている。 alpha リリースでは、大きなコード変更がおきるような開発が行われる可能性があるが、リリースの前にすべてがテストされている。MySQL リリースには既知のバグはない。
  - beta は、すべての新しいコードがテストされていることを示している。古いコードを破壊するような大きな新しい機能は追加されていない。既知のバグはない。少なくとも 1 か月間、alpha バージョンで重大なバグが報告されず、古いコマンドの信頼性を下げるような機能を追加する計画がない場合は、バージョンはアルファからベータに変わる。
  - gamma は、一定期間出回っていて、正常に動作すると思われるベータバージョンである。小さな修正だけが追加される。これは、他の多くの企業でリリースと呼ばれているものである。
  - サフィックスが付いていない場合は、そのバージョンがさまざまなサイトでプラットフォーム固有のバグ以外のバグが報告されることなく一定期間稼動していたことを意味する。そのようなリリースには重大なバグ修正だけが適用される。これは、製品版 (安定版) リリースと呼ばれているものである。

MySQL 開発プロセスでは、段階の異なるさまざまなバージョンが共存します。当然、旧バージョンのシリーズの関連バグ修正は新バージョンに伝播されます。

- 以前の安定版/製品版シリーズ 3.23 の場合、新バージョンは重大なバグの場合のみリリースされる。
- 現行のシリーズ 4.0 は、安定版/製品版レベルの品質で、バグ修正のために新しいバージョンがリリースされている。コードの安定性に影響を及ぼすような新機能は追加されていない。
- alpha ブランチ 4.1 では、重要な新機能が追加されている。開発システムで使用およびテストするためのソースとバイナリが用意されている。
- 開発ブランチ 5.0 は、BitKeeper ツリーからのみ入手できる。

MySQL のすべてのバージョンは、標準テストとベンチマークテストが行われ、比較的安全であることが確認されています。標準テストは、過去に見つかったすべてのバグをチェックするために時間の経過とともに拡張されているので、テストスィートは常に改良されています。

注意: すべてのリリースは少なくとも以下のテストが実行されています。

- 内部テストスィート

  mysql-test ディレクトリには充実したテストケースセットが含まれている。

  これらのテストは、事実上すべてのサーババイナリに対して行われる。

- MySQL ベンチマークスィート

  このテストスィートは、各種の一般的クエリを実行する。これは、また、最新の一連の最適化が実際にコードを高速化するかどうかを確認するためのテストでもある。 See 項5.1.4.「MySQL ベンチマークスィート」。

- crash-me テスト

  データベースがサポートする機能と、その能力と制約の判定が試行される。 See 項5.1.4.「MySQL ベンチマークスィート」。

もう 1 つのテストは、当社が社内の実稼動環境で少なくとも 1 台のマシン上で最新の MySQL バージョンを使用することです。当社には 100 ギガバイトを超える処理用データがあります。

2.2.5. インストールレイアウト

ここでは、バイナリディストリビューションおよびソースディストリビューションのインストールによって作成されるディレクトリのデフォルトレイアウトについて説明します。

バイナリディストリビューションは、選択したインストール場所 (通常は /usr/local/mysql) でアンパックすることでインストールされ、その場所に以下のディレクトリを作成します。

ディレクトリ	ディレクトリの内容
bin	クライアントプログラムと mysqld サーバ
data	ログファイル、データベース
docs	ドキュメント、変更ログ
include	インクルード (ヘッダ) ファイル
lib	ライブラリ
scripts	mysql_install_db
share/mysql	エラーメッセージファイル
sql-bench	ベンチマーク

ソースディストリビューションは、コンフィギャしてコンパイルした後にインストールされます。デフォルトでは、インストールステップによって、以下のサブディレクトリ内の /usr/local の下にファイルがインストールされます。

ディレクトリ	ディレクトリの内容
bin	クライアントプログラムとスクリプト
include/mysql	インクルード (ヘッダ) ファイル
info	Info 形式のマニュアル
lib/mysql	ライブラリ
libexec	mysqld サーバ
share/mysql	エラーメッセージファイル
sql-bench	ベンチマークと crash-me テスト

var	データベースとログファイル

インストールディレクトリ内では、ソースインストールのレイアウトは、バイナリインストールのレイアウトとは以下のように異なります。

- mysqld サーバは、bin ディレクトリではなく、libexec ディレクトリにインストールされる。
- データディレクトリは data ではなく var である。
- mysql_install_db は、/usr/local/mysql/scripts ディレクトリではなく、/usr/local/bin ディレクトリにインストールされる。
- ヘッダファイルとライブラリのディレクトリは、include と lib ではなく、include/mysql と lib/mysql である。

スクリプト scripts/make_binary_distribution を実行して、コンパイルされたソースディストリビューションから独自のバイナリインストールを作成できます。

2.2.6. リリースの方法と時期

MySQL AB 社内で MySQL は非常に急速に進化しており、当社は他の MySQL ユーザとこれを共有したいと考えています。他のユーザが必要としていると思われる非常に有用な機能がある場合は、リリースを作成します。

また、簡単に実装できる機能を求めているユーザの支援にも努めています。ライセンスユーザ、特に拡張電子メールサポート対象のユーザが何を要望しているかに注意し、これらのユーザを支援する努力を行っています。

ユーザは新しいリリースをダウンロードする必要はありません。本当に必要なものが新しいリリースに含まれている場合は、「News」のセクションで通知されます。 See 付録 D. MySQL Change History。

MySQL を更新する場合、当社は以下のポリシーを使用します。

- マイナーな更新が行われるたびに、バージョン文字列の最後の数字が増分される。 重要な新機能がある場合や旧バージョンとのわずかな非互換性がある場合は、バージョン文字列の 2 番目の数字が増分される。ファイル形式が変更された場合は、最初の数字が増分される。
- 製品版 (安定テスト済み) リリースは、1 年に約 1 〜 2 回公開されることになっているが、小さなバグが見つかった場合は、バグ修正だけを含んだリリースが公開される。
- 運用リリース/旧リリースに対するバグ修正は、約 4 〜 8 週ごとに公開される。
- メジャーリリースについては、MySQL AB によって一部のプラットフォーム用のバイナリディストリビューションが作成される。 第三者が、その他のシステム用のバイナリディストリビューションを作成することがあるが、その頻度は少ない。
- 通常、MySQL AB が小さなバグを発見して修正した場合はすぐにパッチを公開する。これらのパッチは、直ちに BitKeeper 公開リポジトリに格納される。また、次のリリースにも組み込まれる。
- 重大ではないが問題を引き起こすバグは、MySQL ソースリポジトリに追加され、次のリリースで修正される。
- 万が一、リリースに重大なバグがあった場合は、当社はできる限り速やかに新しいリリースを作成する。他社にもこの

作業をお願いしたい。

現行の製品版リリースは、バージョン 4.0 です。アクティブな開発はすでにバージョン 4.1 および 5.0 に進んでいます。バージョン 4.0 のバグの修正と、3.23 シリーズの重大なバグの修正は今後も続けられます。 当社は、完全な開発凍結はよしとしません。これによって、バグ修正と "行うべき" ことが放置されることになるからです。"一部凍結" というのは、"すでに稼動しているものにほぼ間違いなく影響を与えない" 小さな変更が追加されることがあるということを意味します。

MySQL では、大部分の他製品とは少し異なる名前付けスキームが使用されます。 一般に、新しいバージョンに置き換えられることなく 2 週間公開されたバージョンは、使用しても比較的安全です。 See 項2.2.4.「使用すべき MySQL のバージョン」。

2.2.7. リリース理念 - リリースに既知のバグがない

当社は、リリースにバグがないようにするために多くの時間と労力を費やしています。 当社の知る限り、既知の '重大な' 再現可能なバグがある MySQL バージョンは 1 つもリリースされていません。

重大なバグとは、通常の使用で MySQL をクラッシュさせる、一般的なクエリに対して誤った応答を返すバグ、またはセキュリティに問題があるバグです。

設計上の決定に依存する未解決の障害、バグ、問題はすべて文書化されています。 See 項1.8.6.「MySQL の既知のエラーと設計上の問題」。

当社の目標は、安定版の MySQL が不安定になるような危険を冒すことなく、修正可能なものはすべて修正することです。場合によって、これは、開発版の問題は修正できるが、安定版 (製品版) の問題は修正できないということを意味します。当然、そのような問題は、ユーザに知らせるために文書化されます。

以下に、当社のビルドプロセスの仕組みを説明します。

- カスタマサポートリスト、MySQL 社外メーリングリスト、およびバグデータベース (http://bugs.mysql.com/) からバグをモニタする。
- 有効なバージョンに関して報告されたすべてのバグがバグデータベースに入力される。
- バグが修正されると、そのバグのテストケースを作成して、当社のテストシステムに組み込み、バグが再現しないことを確認する (修正済みバグの約 90% にはテストケースがある) 。
- また、MySQL に追加されるすべての新機能についてテストケースが作成される。
- 新しい MySQL リリースの作成を開始する前に、MySQL バージョン (3.23.x や 4.0.x など) について報告されたすべての再現可能なバグを確実に修正する。MySQL の内部の設計上のなんらかの決定が理由で修正が不可能なバグがある場合は、そのことをマニュアルに記載する。 See 項1.8.6.「MySQL の既知のエラーと設計上の問題」。
- 専用のバイナリがサポートされているすべてのプラットフォーム (15 以上のプラットフォーム) 上でビルドを行い、それらのすべてのプラットフォーム上で当社のテストスィートとベンチマークスィートを実行する。
- テストまたはベンチマークスィートが失敗したプラットフォームのバイナリは公開されない。ソースの一般的なエラーである場合は、それを修正し、すべてのシステムで最初からビルドとテストを再実行する。
- ビルドおよびテストのプロセス (2、3 日を要する) 中に、重大なバグ (コアダンプを発生させるバグなど) に関する報告を受け取った場合は、そのバグを修正した上で、ビルドプロセスを再開する。

- http://www.mysql.com/ でバイナリを公開した後、mysql と announce のメーリングリストで告知メールを送信する。See 項1.7.1.1.「MySQL メーリングリスト」。 告知メッセージには、リリースに対するすべての変更と、リリースに関する既知の問題の一覧が記載される（リリースノートに「既知の問題」のセクションが必要であったのは、ほんの一握りのリリースだけである）。
- ユーザが MySQL の最新の機能に迅速にアクセスできるように、4 〜 8 週間ごとに新しい MySQL リリースを行う。 ソースコードのスナップショットは毎日作成され、http://downloads.mysql.com/snapshots.php で公開される。
- リリース後に、特定のプラットフォーム上のビルドに（それでも）重大な問題があったというバグレポートを受けた場合は、バグを直ちに修正し、そのプラットフォーム用の新しい 'a' リリースをビルドする。当社の巨大なユーザ基盤のおかげで、問題は迅速に発見される。
- 当社は、優れたリリースを作成することに関して高い実績を持つ。過去の 150 リリースにおいて、新しいビルドを行う必要があったのは 10 リリース未満である（そのうちの 3 つは、当社のビルドマシンの 1 つで glibc ライブラリに欠陥があったためであった。これを突き止めるのに長い時間を要した）。

2.2.8. MySQL AB がコンパイルした MySQL バイナリ

MySQL AB は、サービスとして MySQL の一連のバイナリディストリビューションを提供します。これらのバイナリディストリビューションは、社内でコンパイルするか、ユーザが当社にマシンへのアクセスを提供してくれる場所でコンパイルされます。

プラットフォーム固有のパッケージ形式で提供するバイナリ（項2.1.「標準 MySQL のクイックインストール」 を参照）のほかに、tar 形式の圧縮アーカイブ（.tar.gz）によってさまざまなプラットフォーム用のバイナリディストリビューションも提供します。

これらのディストリビューションは、Build-tools/Do-compile スクリプトを使用して生成されます。このスクリプトは、ソースコードをコンパイルし、scripts/make_binary_distribution を使用してバイナリ tar.gz アーカイブを作成します。これらのバイナリは、以下のコンパイラとオプションによってコンフィギャおよびビルドされます。

MySQL AB の開発システムでビルドされるバイナリは以下のとおりです。

- Linux 2.4.xx x86 with gcc 2.95.3

  CFLAGS="-O2 -mcpu=pentiumpro" CXX=gcc CXXFLAGS="-O2 -mcpu=pentiumpro -felide-constructors" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --enable-assembler --disable-shared --with-client-ldflags=-all-static --with-mysqld-ldflags=-all-static

- Linux 2.4.xx Intel Itanium 2 with ecc (Intel C++ Itanium Compiler 7.0)

  CC=ecc CFLAGS="-O2 -tpp2 -ip -nolib_inline" CXX=ecc CXXFLAGS="-O2 -tpp2 -ip -nolib_inline" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile

- Linux 2.4.xx Intel Itanium with ecc (Intel C++ Itanium Compiler 7.0)

  CC=ecc CFLAGS=-tpp1 CXX=ecc CXXFLAGS=-tpp1 ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile

- Linux 2.4.xx alpha with ccc (Compaq C V6.2-505 / Compaq C++ V6.3-006)

```
CC=ccc CFLAGS="-fast -arch generic" CXX=cxx CXXFLAGS="-fast -arch generic -noexceptions -nortti" ./configure
-prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile
-with-mysqld-ldflags=-non_shared --with-client-ldflags=-non_shared --disable-shared
```

- Linux 2.4.xx s390 with gcc 2.95.3

```
CFLAGS="-O2" CXX=gcc CXXFLAGS="-O2 -felide-constructors" ./configure --prefix=/usr/local/mysql
-with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --disable-shared
-with-client-ldflags=-all-static --with-mysqld-ldflags=-all-static
```

- Linux 2.4.xx x86_64 (AMD64) with gcc 3.2.1

```
CXX=gcc ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client
-enable-local-infile --disable-shared
```

- Sun Solaris 8 x86 with gcc 3.2.3

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -fno-omit-frame-pointer
felide-constructors -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
-libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile
-disable-shared --with-innodb
```

- Sun Solaris 8 sparc with gcc 3.2

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -fno-omit-frame-pointer
felide-constructors -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex
-enable-thread-safe-client --enable-local-infile --enable-assembler --with-named-z-libs=no
-with-named-curses-libs=-lcurses --disable-shared
```

- Sun Solaris 8 sparc 64bit with gcc 3.2

```
CC=gcc CFLAGS="-O3 -m64 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -m64 -fno-omit-frame-pointer
felide-constructors -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex
-enable-thread-safe-client --enable-local-infile --enable-assembler --with-named-z-libs=no
-with-named-curses-libs=-lcurses --disable-shared
```

- Sun Solaris 9 sparc with gcc 2.95.3

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -fno-omit-frame-pointer
felide-constructors -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex
-enable-thread-safe-client --enable-local-infile --enable-assembler --with-named-curses-libs=-lcurses --disable-shared
```

- Sun Solaris 9 sparc with cc-5.0 (Sun Forte 5.0)

```
CC=cc-5.0 CXX=CC ASFLAGS="-xarch=v9" CFLAGS="-Xa -xstrconst -mt -D_FORTEC_ -xarch=v9"
CXXFLAGS="-noex -mt -D_FORTEC_ -xarch=v9" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex
-enable-thread-safe-client --enable-local-infile --enable-assembler --with-named-z-libs=no --enable-thread-safe-client
-disable-shared
```

- IBM AIX 4.3.2 ppc with gcc 3.2.3

CFLAGS="-O2 -mcpu=powerpc -Wa,-many " CXX=gcc CXXFLAGS="-O2 -mcpu=powerpc -Wa,-many -felide-constructors -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --with-named-z-libs=no --disable-shared

- IBM AIX 4.3.3 ppc with xlC_r (IBM Visual Age C/C++ 6.0)

  CC=xlc_r CFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192" CXX=xlC_r CXXFLAGS ="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192" ./configure --prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data --libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --with-named-z-libs=no --disable-shared --with-innodb

- IBM AIX 5.1.0 ppc with gcc 3.3

  CFLAGS="-O2 -mcpu=powerpc -Wa,-many" CXX=gcc CXXFLAGS="-O2 -mcpu=powerpc -Wa,-many -felide-constructors -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --with-server-suffix="-pro" --enable-thread-safe-client --enable-local-infile --with-named-z-libs=no --disable-shared

- HP-UX 10.20 pa-risc1.1 with gcc 3.1

  CFLAGS="-DHPUX -I/opt/dce/include -O3 -fPIC" CXX=gcc CXXFLAGS="-DHPUX -I/opt/dce /include -felide-constructors -fno-exceptions -fno-rtti -O3 -fPIC" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --with-pthread --with-named-thread-libs=-ldce --with-lib-ccflags=-fPIC --disable-shared

- HP-UX 11.11 pa-risc2.0 64bit with aCC (HP ANSI C++ B3910B A.03.33)

  CC=cc CXX=aCC CFLAGS=+DD64 CXXFLAGS=+DD64 ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --disable-shared

- HP-UX 11.11 pa-risc2.0 32bit with aCC (HP ANSI C++ B3910B A.03.33)

  CC=cc CXX=aCC CFLAGS="+DAportable" CXXFLAGS="+DAportable" ./configure --prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data --libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --disable-shared --with-innodb

- Apple Mac OS X 10.2 powerpc with gcc 3.1

  CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --disable-shared

- FreeBSD 4.7 i386 with gcc 2.95.4

  CFLAGS=-DHAVE_BROKEN_REALPATH ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --enable-assembler --with-named-z-libs=not-used --disable-shared

- QNX Neutrino 6.2.1 i386 with gcc 2.95.3qnx-nto 20010315

  CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --disable-shared

以下のバイナリは、他のユーザによって MySQL AB に提供されたサードパーティのシステム上でビルドされています。これらのバイナリは、好意によって提供されています。MySQL AB は、これらのシステムを完全に管理しているわけではないので、これらのシステム上でビルドされたバイナリに対しては限られたサポートしか提供できません。

- SCO Unix 3.2v5.0.6 i386 with gcc 2.95.3

  CFLAGS="-O3 -mpentium" LDFLAGS=-static CXX=gcc CXXFLAGS="-O3 -mpentium -felide-constructors" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --with-named-z-libs=no --enable-thread-safe-client --disable-shared

- SCO OpenUnix 8.0.0 i386 with CC 3.2

  CC=cc CFLAGS="-O" CXX=CC ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --with-named-z-libs=no --enable-thread-safe-client --disable-shared

- Compaq Tru64 OSF/1 V5.1 732 alpha with cc/cxx (Compaq C V6.3-029i / DIGITAL C++ V6.1-027)

  CC="cc -pthread" CFLAGS="-O4 -ansi_alias -ansi_args -fast -inline speed -speculate all" CXX="cxx -pthread" CXXFLAGS="-O4 -ansi_alias -fast -inline speed -speculate all -noexceptions -nortti" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --with-prefix=/usr/local/mysql --with-named-thread-libs="-lpthread -lmach -lexc -lc" --disable-shared --with-mysqld-ldflags=-all-static

- SGI Irix 6.5 IP32 with gcc 3.0.1

  CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --disable-shared

- FreeBSD 5.0 sparc64 with gcc 3.2.1

  CFLAGS=-DHAVE_BROKEN_REALPATH ./configure --prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data --libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --disable-shared --with-innodb

以下のコンパイルオプションは、MySQL AB がかつて提供していたバイナリパッケージに使用されていたものです。これらのバイナリは現在では更新されていませんが、参考のためにここにコンパイルオプションを記載します。

- Linux 2.2.xx sparc with egcs 1.1.2

  CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --enable-assembler --disable-shared

- Linux 2.2.x with x686 with gcc 2.95.2

  CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro -felide-constructors -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql --enable-assembler --with-mysqld-ldflags=-all-static --disable-shared --with-extra-charsets=complex

- SunOS 4.1.4 2 sun4c with 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-assembler

- SunOS 5.5.1 (and above) sun4u with 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-assembler

- SunOS 5.6 i86pc with gcc 2.8.1

  CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql --with-low-memory --with-extra-charsets=complex

- BSDI BSD/OS 3.1 i386 with 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 with gcc 2.7.2

  CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex

- AIX 2 4 with gcc 2.7.2.2

  CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex

上記のいずれかの設定に関してさらに最適なオプションをお持ちの方は、そのオプションを MySQL 社内メーリングリストに随時お送りください。 See 項1.7.1.1.「MySQL メーリングリスト」。

MySQL バージョン3.22 より前の RPM ディストリビューションは、ユーザによる寄贈です。バージョン 3.22 以降の RPM は、MySQL AB によって作成されています。

MySQL のデバッグバージョンをコンパイルする場合は、前述の configure 行に --with-debug オプションまたは --with-debug=full オプションを追加し、すべての -fomit-frame-pointer オプションを削除してください。

Windows ディストリビューションについては、項2.1.1.「Windows への MySQL のインストール」 を参照してください。

2.2.9. MySQL バイナリディストリビューションのインストール

ここでは、さまざまなプラットフォーム用の MySQL バイナリディストリビューション (.tar.gz アーカイブ) について説明します (詳細な一覧については、項2.2.8.「MySQL AB がコンパイルした MySQL バイナリ」 を参照してください)。

これらの一般的なパッケージのほかに、選択したプラットフォームのプラットフォーム固有のパッケージ形式のバイナリも提供しています。 これらのパッケージのインストール方法の詳細については、項2.1.「標準 MySQL のクイックインストール」を参照してください。

一般的な MySQL バイナリディストリビューションは、gzip 形式で圧縮された GNU tar アーカイブ (.tar.gz) としてパッケージ化されています。MySQL バイナリディストリビューションをインストールするためには以下のツールが必要です。

- ディストリビューションを解凍するための GNU gunzip。

- ディストリビューションをアンパックするための適切な tar プログラム。GNU tar は、有効であることで知られている。オペレーティングシステムにプリインストールされている tar 実装の中には、長いファイル名などに関する問題が発生することが知られているものがある（Sun tar など）。その場合は、まず GNU tar をインストールする。

問題が発生した場合は、MySQL メーリングリストに質問を投稿するときに、必ず mysqlbug を使用してください。その問題がバグでない場合でも、mysqlbug は、他のユーザが同じ問題を解決する際に役立つシステム情報を収集します。mysqlbug を使用しないと、問題を解決できる可能性が小さくなります。ディストリビューションをアンパックすると、bin ディレクトリに mysqlbug が格納されます。 See 項1.7.1.3.「バグまたは問題を報告する方法」。

MySQL バイナリディストリビューションをインストールして使用するために実行する必要のある基本的なコマンドは、以下のとおりです。

```
shell> groupadd mysql
shell> useradd -g mysql mysql
shell> cd /usr/local
shell> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
shell> ln -s full-path-to-mysql-VERSION-OS mysql
shell> cd mysql
shell> scripts/mysql_install_db
shell> chown -R root  .
shell> chown -R mysql data
shell> chgrp -R mysql .
shell> bin/mysqld_safe --user=mysql &
```

4.0 より前のバージョンの MySQL を使用している場合は、最後のコマンドの bin/mysqld_safe を bin/safe_mysqld に置き換えてください。

DBI および DBD-mysql Perl モジュールをインストールしている場合は、bin/mysql_setpermission スクリプトを使用して新しいユーザを追加できます。

以下に詳細な説明を示します。

バイナリディストリビューションをインストールするには、以下のステップに従って処理を行った後、項2.4.「インストール後の設定とテスト」 に進み、インストール後の設定とテストを行います。

1. ディストリビューションのアンパック先とするディレクトリを決め、そのディレクトリに移動する。以下の例では、ディストリビューションは /usr/local にアンパックされる（したがって、以降の操作説明は、/usr/local 内にファイルおよびディレクトリを作成するためのアクセス権をユーザが持っていることを前提として書かれている。 そのディレクトリが保護されている場合は、root としてインストールを実行する必要がある）。

2. 項2.2.1.「MySQL の入手方法」 に記載されているいずれかのサイトからディストリビューションファイルを入手する。

   MySQL バイナリディストリビューションは、圧縮された tar アーカイブとして提供され、mysql-VERSION-OS.tar.gz などの名前が付いている。この場合、VERSION はバージョン番号（3.21.15 など）を表し、OS はそのディストリビューションが対象とするオペレーティングシステムを示す（pc-linux-gnu-i586 など）。 注意: すべてのバイナリは、同じ MySQL ソースディストリビューションからビルドされている

3. mysqld の実行時に使用するユーザとグループを追加する。

   ```
   shell> groupadd mysql
   ```

```
shell> useradd -g mysql mysql
```

上記のコマンドで、mysql グループと mysql ユーザが追加される。useradd と groupadd の構文は、Unix のバージョンの種類によって少し異なる。これらのコマンドは、それぞれ、adduser および addgroup と呼ばれることもある。ユーザとグループを mysql 以外の名前にしてもかまわない。

4. 以下のコマンドで、目的のインストールディレクトリに移動する。

```
shell> cd /usr/local
```

5. ディストリビューションをアンパックする。これによって、インストールディレクトリが作成される。 次に、そのディレクトリへのシンボリックリンクを作成する。

```
shell> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
shell> ln -s full-path-to-mysql-VERSION-OS mysql
```

GNU tar を使用して、最初の行を以下の代替コマンドに置き換えて、ディストリビューションの展開と抽出を一度に行うこともできる。

```
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
```

最初のコマンドで、mysql-VERSION-OS という名前のディレクトリが作成される。2 番目のコマンドで、そのディレクトリへのシンボリックリンクが作成される。これによって、インストールディレクトリを /usr/local/mysql としてより簡単に参照できるようになる。

6. 以下のコマンドで、インストールディレクトリに移動する。

```
shell> cd mysql
```

mysql ディレクトリ内には、いくつかのファイルとサブディレクトリがある。 インストールのために最も重要なのは、bin サブディレクトリと scripts サブディレクトリである。

- bin

  このディレクトリにはクライアントプログラムとサーバが格納されている。 シェルが MySQL プログラムを正しく検索するように、PATH 環境変数にこのディレクトリの完全パス名を追加する必要がある。 See 付録 F. 環境変数 。

- scripts

  このディレクトリには、サーバアクセス権を保存する権限テーブルが格納された mysql データベースの初期化に使用する mysql_install_db スクリプトが格納されている。

7. mysqlaccess を使用する場合、標準以外の場所に MySQL ディストリビューションがあるときは、mysqlaccess が mysql クライアントを検索する場所を変更する必要がある。18 行目ぐらいにある bin/mysqlaccess スクリプトを編集する。以下のような行を検索する。

```
$MYSQL     = '/usr/local/bin/mysql';    # path to mysql executable
```

mysql が実際に格納されているシステム内の場所を反映するようにパスを変更する。この処理を行わないと、

mysqlaccess を実行したときに Broken pipe というエラーが表示される。

8. 以下のコマンドで、MySQL 権限テーブルを作成する（初めて MySQL をインストールする場合にのみ必要）。

```
shell> scripts/mysql_install_db
```

注意: バージョン 3.22.10 より前の MySQL バージョンでは、mysql_install_db を実行すると MySQL サーバが起動されていた。現バージョンでは、MySQL サーバは起動されない。

9. バイナリの所有者を root に変更し、データディレクトリの所有者を mysqld の実行時に使用するユーザに変更する。

```
shell> chown -R root  /usr/local/mysql/.
shell> chown -R mysql /usr/local/mysql/data
shell> chgrp -R mysql /usr/local/mysql/.
```

最初のコマンドでファイルの owner 属性が root ユーザに変更され、2 番目のコマンドで data ディレクトリの owner 属性が mysql ユーザに変更される。3 番目のコマンドで、group 属性が mysql グループに変更される。

10. Perl の DBI/DBD インタフェースのサポートをインストールする場合は、項2.7.「Perl インストールについてのコメント」 を参照。

11. マシンのブート時に MySQL を自動的に起動させる場合は、使用しているシステムの起動ファイルが格納されている場所に support-files/mysql.server をコピーする。詳細については、support-files/mysql.server スクリプト自体と 項 2.4.3.「MySQL を自動的に起動および停止する」 を参照。

すべてのものをアンパックしてインストールしたら、ディストリビューションの初期化とテストを行ってください。

以下のコマンドで MySQL サーバを起動することができます。

```
shell> bin/mysqld_safe --user=mysql &
```

4.0 より前のバージョンの MySQL を使用している場合は、このコマンドの bin/mysqld_safe を bin/safe_mysqld に置き換えてください。

次に、項4.8.2.「mysqld_safe（mysqld のラッパ）」 に進み、See 項2.4.「インストール後の設定とテスト」 を参照してください。

2.3. MySQL ソースディストリビューションのインストール

ソースのインストールに進む前に、使用しているプラットフォーム用のバイナリが入手可能かどうかと、それが自分のシステムでうまく機能するかどうかをまず確認します。当社は、バイナリを最適なオプションでビルドするよう尽力しています。

ソースから MySQL ビルドしてインストールするためには以下のツールが必要です。

- ディストリビューションを伸長するための GNU gunzip。
- ディストリビューションをアンパックするための適切な tar プログラム。GNU tar は、有効であることで知られている。オペレーティングシステムにプリインストールされている tar 実装の中には、長いファイル名などに関する問題が発

生することが知られているものがある（Sun tar など）。その場合は、まず GNU tar をインストールする。

- 正しく機能する ANSI C++ コンパイラ。正しく機能することが知られているコンパイラとしては、2.95.2 以降の gcc、1.0.2 以降の egcs、egcs 2.91.66、SGI C++、および SunPro C++ などがある。gcc を使用する場合は、libg++ は必要ない。gcc 2.7.x には、sql/sql_base.cc などのまったく問題のない C++ ファイルをコンパイルできなくなるというバグがある。gcc 2.7.x しかない場合、MySQL をコンパイルするためには、gcc をアップグレードする必要がある。 gcc また、2.8.1 は、一部のプラットフォームで問題が発生することが知られている。したがって、使用しているプラットフォーム用の新しいコンパイラが存在する場合は、2.8.1 の使用は避ける必要がある。

  MySQL バージョン 3.23.x をコンパイルする場合は、gcc 2.95.2 以降の使用を推奨する。

- 適切な make プログラム。GNU make は、どの場合でも推奨されるものであり、必須のこともある。問題があった場合は、GNU make 3.75 以降の使用をお勧めする。

-fno-exceptions オプションを認識できる最新バージョンの gcc を使用している場合は、このオプションを使用することが非常に大切です。このオプションを使用しないと、無秩序にクラッシュするバイナリをコンパイルしてしまうことになりかねません。また、-fno-exceptions と共に、-felide-constructors および -fno-rtti を使用することもお勧めします。判断がつかないは、以下を行ってください。

```
CFLAGS="-O3" CXX=gcc CXXFLAGS="-O3 -felide-constructors -fno-exceptions \
    -fno-rtti" ./configure --prefix=/usr/local/mysql --enable-assembler \
    --with-mysqld-ldflags=-all-static
```

ほとんどのシステムでは、これによって高速で安定したバイナリが提供されます。

問題が発生した場合は、MySQL メーリングリストに質問を投稿するときに、必ず mysqlbug を使用してください。その問題がバグでない場合でも、mysqlbug は、他のユーザが同じ問題を解決する際に役立つシステム情報を収集します。mysqlbug を使用しないと、問題を解決できる可能性が小さくなります。ディストリビューションをアンパックすると、scripts ディレクトリに mysqlbug が格納されます。 See 項1.7.1.3.「バグまたは問題を報告する方法」。

2.3.1. クイックインストールの概要

MySQL ソースディストリビューションをインストールするために実行する必要のある基本的なコマンドは、以下のとおりです。

```
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/mysqld_safe --user=mysql &
```

バージョン 4.0 より前の MySQL を使用している場合は、最後のコマンドの bin/mysqld_safe を bin/safe_mysqld に置き換えてください。

InnoDB テーブルのサポートが必要な場合は、/etc/my.cnf ファイルを編集して、innodb_... で始まるパラメータの前にある # 文字を削除してください。 See 項4.1.2.「my.cnf オプション設定ファイル」 と 項7.5.3.「InnoDB 起動オプション」 を参照してください。

ソース RPM から開始する場合は、以下のコマンドを実行します。

```
shell> rpm --rebuild --clean MySQL-VERSION.src.rpm
```

これによって、インストールできるバイナリ RPM が作成されます。

DBI および DBD-mysql Perl モジュールをインストールしている場合は、bin/mysql_setpermission スクリプトを使用して新しいユーザを追加できます。

以下に詳細な説明を示します。

ソースディストリビューションをインストールするには、以下のステップに従って処理を行った後、項2.4.「インストール後の設定とテスト」 に進み、ポストインストール初期化とテストを行います。

1. ディストリビューションのアンパック先とするディレクトリを決め、そのディレクトリに移動する。
2. 項2.2.1.「MySQL の入手方法」 に記載されているいずれかのサイトからディストリビューションファイルを入手する。
3. MySQL で Berkeley DB テーブルを使用することを考えている場合は、Berkeley DB ソースコードのパッチ適用済みバージョンを入手する必要がある。次に進む前に、Berkeley DB テーブルに関する章を読むことをお勧めする。 See 項7.6.「BDB または BerkeleyDB テーブル」。

   MySQL ソースディストリビューションは、tar 形式の圧縮アーカイブとして提供され、mysql-VERSION.tar.gz などの名前が付いている。この場合、VERSION は、5.0.6-beta などのバージョン番号である。
4. mysqld の実行時に使用するユーザとグループを追加する。

   ```
   shell> groupadd mysql
   shell> useradd -g mysql mysql
   ```

   上記のコマンドで、mysql グループと mysql ユーザが追加される。useradd と groupadd の構文は、Unix のバージョンの種類によって少し異なる。これらのコマンドは、それぞれ、adduser および addgroup と呼ばれることもある。ユーザとグループを mysql 以外の名前にしてもかまわない。
5. ディストリビューションをカレントディレクトリにアンパックする。

   ```
   shell> gunzip < /path/to/mysql-VERSION.tar.gz | tar xvf -
   ```

   このコマンドによって、mysql-VERSION という名前のディレクトリが作成される。
6. アンパックしたディストリビューションの最上位ディレクトリに移動する。

   ```
   shell> cd mysql-VERSION
   ```

注意: 現時点では、最上位ディレクトリから MySQL をコンフィギャしてビルドする。他のディレクトリで MySQL をビルドすることはできない。

7. コンフィギャして、コンパイルする。

```
shell> ./configure --prefix=/usr/local/mysql
shell> make
```

configure を実行するときに、いくつかのオプションを指定することができる。 オプションの一覧が必要な場合は、./configure --help を実行する。 項2.3.3.「一般的な configure オプション」 では、さらに便利ないくつかのオプションについて説明する。

configure が失敗し、MySQL メーリングリストにメールを送信して支援を求める場合は、config.log の中の、問題の解決に役立つと思われる行をメールに含める。configure が中断した場合は、configure からの出力の最後の 2、3 行も含める。mysqlbug スクリプトを使用して、バグレポートを投稿する。 See 項1.7.1.3.「バグまたは問題を報告する方法」。

コンパイルが失敗した場合は、項2.3.5.「MySQL のコンパイルに関する問題への対処」 で、数多くの一般的な問題に関するヘルプを参照。

8. 以下のコマンドで、すべてのものをインストールする。

```
shell> make install
```

このコマンドは、root として実行する必要がある場合がある。

9. 以下のコマンドで、MySQL 権限テーブルを作成します (初めて MySQL をインストールする場合にのみ必要) 。

```
shell> scripts/mysql_install_db
```

注意: バージョン 3.22.10 より前の MySQL バージョンでは、mysql_install_db を実行すると MySQL サーバが起動されていた。現バージョンでは、MySQL サーバは起動されない。

10. バイナリの所有者を root に変更し、データディレクトリの所有者を mysqld の実行時に使用するユーザに変更する。

```
shell> chown -R root  /usr/local/mysql
shell> chown -R mysql /usr/local/mysql/var
shell> chgrp -R mysql /usr/local/mysql
```

最初のコマンドでファイルの owner 属性が root ユーザに変更され、2 番目のコマンドでデータディレクトリの owner 属性が mysql ユーザに変更される。3 番目のコマンドで、group 属性が mysql グループに変更される。

11. Perl の DBI/DBD インタフェースのサポートをインストールする場合は、項2.7.「Perl インストールについてのコメント」 を参照。

12. マシンのブート時に MySQL を自動的に起動させる場合は、使用しているシステムの起動ファイルが格納されている場所に support-files/mysql.server をコピーする。詳細については、support-files/mysql.server スクリプト自体と 項2.4.3.「MySQL を自動的に起動および停止する」 を参照。

すべてのものをインストールしたら、以下のコマンドを使用してディストリビューションの初期化とテストを行ってくだ

さい。

```
shell> /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

バージョン 4.0 より前の MySQL を使用している場合は、このコマンドの mysqld_safe を safe_mysqld に置き換えてください。

そのコマンドが、mysqld daemon ended というエラーですぐに失敗した場合は、ファイル mysql-data-directory/'hostname'.err に何らかの情報が記述されています。 原因としては、すでに別の mysqld サーバを実行していることが考えられます。 See 項4.2.「同じマシン上で複数の MySQL サーバを実行する」。

次に、項2.4.「インストール後の設定とテスト」 に進みます。

2.3.2. パッチの適用

パッチは、メーリングリストまたは MySQL の Web サイト (http://www.mysql.com/downloads/patches.html) の「patches」エリアにときどき公開されます。

メーリングリストにあるパッチを適用するには、パッチが含まれたメッセージをファイルに保存し、MySQL ソースツリーの最上位ディレクトリに移動して以下のコマンドを実行します。

```
shell> patch -p1 < patch-file-name
shell> rm config.cache
shell> make clean
```

FTP サイトにあるパッチは、プレーンテキストファイルまたは gzip で圧縮されたファイルとして配布されます。プレーンテキストのパッチの適用は、メーリングリストのパッチの箇所で説明した方法で行います。圧縮されたパッチを適用するには、MySQL ソースツリーの最上位ディレクトリに移動して、以下のコマンドを実行します。

```
shell> gunzip < patch-file-name.gz | patch -p1
shell> rm config.cache
shell> make clean
```

パッチを適用した後に、通常のソースインストール手順に従いますが、./configure ステップから開始します。make install ステップを実行したら、MySQL サーバを再起動します。

make install を実行する前に、現在稼動しているサーバを停止させる必要があります (サーバを停止させるには mysqladmin shutdown を使用します) 。一部のシステムでは、新しいバージョンのプログラムが現在実行されているバージョンを置き換える場合は、新しいバージョンのインストールが許可されないことがあります。

2.3.3. 一般的な configure オプション

configure スクリプトを使用すると、MySQL ディストリビューションの設定を自由に調整できます。設定方法の調整は、通常は configure コマンドラインでオプションを使用して行います。また、環境変数を使用して configure を調整することもできます。See 付録 F. 環境変数。 configure がサポートするオプションの一覧が必要な場合は、以下のコマンドを実行します。

```
shell> ./configure --help
```

ここでは、広く使用されている configure オプションの一部を説明します。

- MySQL のクライアントライブラリとクライアントプログラムだけをコンパイルして、サーバをコンパイルしない場合は、以下のように --without-server オプションを使用する。

```
shell> ./configure --without-server
```

  C++ コンパイラがない場合は、mysql はコンパイルを実行しない (mysql は C++ を必要とする唯一のクライアントプログラム)。その場合は、configure の、C++ コンパイラをテストするコードを削除し、--without-server オプションを指定して ./configure を実行する。compile ステップは依然として mysql のビルドを試行するが、mysql.cc に関する警告は無視してもかまわない (make が停止したら、make -k を実行して、エラーが発生しても残りのビルドを続行するよう指示する) 。

- 組み込みの MySQL サーバライブラリ (libmysqld.a) が必要な場合は、--with-embedded-server オプションを使用する。

- ログファイルとデータベースディレクトリを /usr/local/var の下に配置しない場合は、以下のいずれかと類似した configure コマンドを使用する。

```
shell> ./configure --prefix=/usr/local/mysql
shell> ./configure --prefix=/usr/local \
        --localstatedir=/usr/local/mysql/data
```

  最初のコマンドでは、デフォルトの /usr/local ではなく /usr/local/mysql にすべてのものがインストールされるようにインストールプリフィックスが変更される。2 番目のコマンドでは、デフォルトのインストールプリフィックスは維持されるが、データベースディレクトリのデフォルトの場所 (通常は /usr/local/var) が無効にされ、/usr/local/mysql/data に変更される。これらのオプションは、MySQL をコンパイルした後に、オプション設定ファイルで変更できる。 See 項4.1.2.「my.cnf オプション設定ファイル」。

- Unix を使用しており、MySQL ソケットファイルをデフォルトの場所 (通常は、/tmp または /var/run) 以外の場所に配置する場合は、以下のような configure コマンドを使用する。

```
shell> ./configure --with-unix-socket-path=/usr/local/mysql/tmp/mysql.sock
```

  注意: 指定するファイルは絶対パス名でなければならない。 また、後で MySQL オプション設定ファイルを使用して、場所 mysql.sock を変更することもできる。 See 項A.4.5.「MySQL ソケットファイル /tmp/mysql.sock の保護または変更方法」。

- 静的にリンクされたプログラムをコンパイルする場合 (バイナリディストリビューションの作成、高速化、一部の Red Hat Linux ディストリビューションの問題の回避などのために) 、以下のような configure コマンドを実行します。

```
shell> ./configure --with-client-ldflags=-all-static \
        --with-mysqld-ldflags=-all-static
```

- gcc を使用しており、libg++ または libstdc++ がインストールされていない場合、C++ コンパイラとして gcc を使用するように configure を設定することができる。

```
shell> CC=gcc CXX=gcc ./configure
```

  C++ コンパイラとして gcc を使用すると、libg++ または libstdc++ へのリンクは試行されない。上記のライブラリがインストールされている場合でも、これらのライブラリの一部のバージョンは過去に MySQL ユーザにとって未知の問題を引き起こしたことがあるので、このように処理することをお勧めする。

以下に、使用しているコンパイラに応じて設定する一般的な環境変数をいくつか示す。

コンパイラ	推奨オプション
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 以降	CFLAGS="-O3 -mpentiumpro -mstack-align-double" CXX=gcc CXXFLAGS="-O3 -mpentiumpro -mstack-align-double -felide-constructors -fno-exceptions -fno-rtti"

ほとんどの場合、上記の表に記載されているオプションを使用して、configure 行に以下のオプションを追加して、MySQL バイナリを適度に最適化することができる。

```
--prefix=/usr/local/mysql --enable-assembler \
--with-mysqld-ldflags=-all-static
```

つまり、最新のすべての gcc バージョン向けの完全な configure 行は、以下のようなものになる。

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro \
-felide-constructors -fno-exceptions -fno-rtti" ./configure \
--prefix=/usr/local/mysql --enable-assembler \
--with-mysqld-ldflags=-all-static
```

当社が MySQL Web サイト (http://www.mysql.com/) で提供するバイナリはすべて、完全に最適化されてコンパイルされており、ほとんどのユーザにとって最適である。See 項2.2.8.「MySQL AB がコンパイルした MySQL バイナリ」。さらに高速なバイナリを作成するためにユーザが微調整できる設定がいくつかあるが、これは上級ユーザ向けである。See 項5.5.3.「MySQL の速度に対するコンパイルとリンクの影響」。

ビルドが失敗し、コンパイラまたはリンカが共有ライブラリ libmysqlclient.so.# ('#' はバージョン番号です) を作成できないというエラーが生成された場合は、configure に --disable-shared オプションを指定してこの問題に対処できる。この場合、configure は共有ライブラリ libmysqlclient.so.# をビルドしない。

- 非 NULL 属性のカラム (つまり、NULL にすることが許されないカラム) に DEFAULT の値を使用しないように MySQL をコンフィギャすることができる。See 項1.8.5.2.「NOT NULL および DEFAULT 値制約」。

```
shell> CXXFLAGS=-DDONT_USE_DEFAULT_FIELDS ./configure
```

- MySQL は、デフォルトで ISO-8859-1 (Latin1) キャラクタセットを使用する。デフォルトのキャラクタセットを変更するには、以下のように --with-charset オプションを使用する。

```
shell> ./configure --with-charset=CHARSET
```

CHARSET は、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、または win1251ukr のいずれかである。See 項4.7.1.「データおよびソート用キャラクタセット」。

サーバとクライアントの間で文字を変換する場合は、SET CHARACTER SET コマンドに注目する。See 項5.5.6.「SET 構文」。

警告: テーブルを作成した後にキャラクタセットを変更する場合は、すべてのテーブルで myisamchk -r -q --set-character-set=charset を実行する必要がある。これを行わないと、インデックスが正しくソートされないことがある（MySQL をインストールし、いくつかのテーブルを作成し、別のキャラクタセットを使用するように MySQL をコンフィギャし直して、その MySQL を再インストールした場合に、この問題が起きる可能性がある）。

--with-extra-charsets=LIST オプションを使用して、サーバにコンパイルする追加のキャラクタセットを定義できる。

この場合、LIST は、カンマで区切ったキャラクタセットのリスト、動的にロードできないすべての文字を含めるための complex、またはすべてのキャラクタセットをバイナリに含めるための all のいずれかである。

- コードをデバッグするように MySQL をコンフィギャするには、--with-debug オプションを使用する。

```
shell> ./configure --with-debug
```

これによって、エラーを検出し、エラーの内容に関する出力を提供する安全なメモリアロケータが組み込まれる。 See 項E.1.「MySQL サーバのデバッグ」。

- クライアントプログラムがスレッドを使用する場合、--enable-thread-safe-client configure オプションを使用して、MySQL クライアントライブラリのスレッドセーフバージョンをコンパイルする必要もある。これによって、スレッドアプリケーションとリンクさせる libmysqlclient_r ライブラリが作成される。 See 項11.1.14.「スレッドクライアントの作成方法」。

- 特定のシステムに関係するオプションについては、このマニュアルのシステム固有のセクションを参照。 See 項2.6.「オペレーティングシステム固有の注意事項」。

2.3.4. 開発ソースツリーからのインストール

注意:このセクションは、当社の新しいコードのテストに協力することに関心のある場合にのみお読みください。MySQL をビルドしてシステム上で稼動させるだけの場合は、標準のリリースディストリビューション（ソースディストリビューションまたはバイナリディストリビューション）を使用してください。

最新の開発ソースツリーを入手するには、以下の説明に従ってください。

1. http://www.bitmover.com/cgi-bin/download.cgi から BitKeeper をダウンロードする。当社のリポジトリにアクセスするには、Bitkeeper 3.0 以降が必要である。
2. 指示に従って、ダウンロードした BitKeeper をインストールする。
3. BitKeeper をインストールしたら、まず、作業するディレクトリに移動する。次に、以下のいずれかのコマンドを使用して、必要な MySQL バージョンブランチをコピーする。

   3.23（旧バージョン）ブランチをコピーするには、以下のコマンドを使用する。

   ```
   shell> bk clone bk://mysql.bkbits.net/mysql-3.23 mysql-3.23
   ```

   4.0（安定版/製品版）ブランチをコピーするには、以下のコマンドを使用する。

   ```
   shell> bk clone bk://mysql.bkbits.net/mysql-4.0 mysql-4.0
   ```

4.1 alpha ブランチをコピーするには、以下のコマンドを使用する。

```
shell> bk clone bk://mysql.bkbits.net/mysql-4.1 mysql-4.1
```

5.0 開発ブランチをコピーするには、以下のコマンドを使用する。

```
shell> bk clone bk://mysql.bkbits.net/mysql-5.0 mysql-5.0
```

上記の例では、カレントディレクトリの、mysql-3.23/、mysql-4.0/、mysql-4.1/、mysql-5.0/ のいずれかのサブディレクトリにソースツリーがセットアップされる。

ファイアウォールの背後にいて、HTTP 接続しか開始できない場合は、HTTP を通じて BitKeeper を使用することもできる。

プロキシサーバを使用する必要がある場合は、使用するプロキシを指すように環境変数 http_proxy を設定する

```
shell> export http_proxy="http://your.proxy.server:8080/"
```

次に、コピーを実行するときに、bk:// を http:// に置き換える。たとえば、以下のように指定する。

```
shell> bk clone http://mysql.bkbits.net/mysql-4.1 mysql-4.1
```

接続の速度によって、ソースツリーの初期ダウンロードには多少時間がかかることがある。

4. 次の一連のコマンドを実行するには、GNU make、autoconf 2.53 (以降)、automake 1.5、libtool 1.4、および m4 が必要である。多くのオペレーティングシステムには独自の make 実装が搭載されているが、未知のエラーメッセージが出力されてコンパイルが失敗する可能性が高い。したがって、必ず GNU make （ gmake と呼ばれることもある ）を使用する。

   幸い、多くのオペレーティングシステムは GNU ツールチェーンがプリインストールされているか、インストール可能な GNU ツールのパッケージを提供する。いずれの場合も、以下の場所からダウンロードすることもできる。

   - http://www.gnu.org/software/autoconf/
   - http://www.gnu.org/software/automake/
   - http://www.gnu.org/software/libtool/
   - http://www.gnu.org/software/make/

   MySQL 4.1 をコンフィギャする場合は、GNU bison 1.75 も必要である。旧バージョンの bison は、次のようなエラーを報告することがある。 sql_yacc.yy:#####: fatal error: maximum table size (32767) exceeded。 注意: 実際に最大テーブルサイズを超過しているわけではない。このエラーは、旧バージョンの bison のバグが原因で発生する。

   バージョン 4.1 より前の MySQL バージョンは、他の yacc 実装 (BSD yacc 91.7.30 など) によってもコンパイルする。4.1 以降のバージョンの場合は、GNU bison は必要条件である。

   シェルで実行する標準的なコマンドは以下のとおりである。

   ```
   cd mysql-4.0
   bk -r edit
   aclocal; autoheader; autoconf; automake
   ```

```
(cd innobase; aclocal; autoheader; autoconf; automake) # for InnoDB
(cd bdb/dist; sh s_all ) # for Berkeley DB
./configure  # Add your favorite options here
make
```

この段階で未知のエラーが発生した場合は、libtool が実際にインストールされているかどうかをチェックする。

当社の標準 configure スクリプトのコレクションは、BUILD/ サブディレクトリにある。手間を省きたい場合は、BUILD/compile-pentium-debug を使用することもできる。別のアーキーテクチャ上でコンパイルするには、このスクリプトから Pentium 固有のフラグを削除して修正する。

5. ビルドが完了したら、make install を実行する。実稼動マシン上でこれを行う場合は、このコマンドによって稼動中のリリースインストールが上書きされるという点に注意する。別の MySQL がインストールされている場合は、prefix、with-tcp-port、unix-socket-path の各オプションに実稼動サーバで使用している値と異なる値を設定して、./configure を実行することをお勧めする。

6. 新しいインストールを酷使して、新しい機能をクラッシュさせてみる。make test の実行から始める。 See 項13.1.2.「MySQL テストスイート」。

7. make の段階に到達しても、ディストリビューションがコンパイルできない場合は、http://bugs.mysql.com/ にあるバグデータベースに報告をお願いしたい。必要な GNU ツールの最新バージョンがインストールされていて、それらのツールが設定ファイルの処理中にクラッシュする場合も同様である。 ただし、aclocal を実行して command not found というエラーや類似の問題が発生した場合は、報告は不要である。代わりに、必要なツールがすべてインストールされているかどうかと、シェルがそれらのツールを検出できるように PATH 変数が正しく設定されているかどうかを確認する。

8. ソースツリーを取得するための最初の bk clone 操作の後、定期的に bk pull を実行して更新を入手する。

9. bk sccstool を使用して、ツリーの変更履歴ですべての diff を調べることができる。不明な diff やコードがあった場合は、MySQL の内部メーリングリストにメールをお送りいただきたい。 See 項1.7.1.1.「MySQL メーリングリスト」。 また、何かに関する修正案がある場合も、同じアドレスにパッチを添付したメールをお送りいただきたい。 bk diffs は、ユーザがソースを変更するとパッチを作成してくれる。アイデアをコードにする時間がない場合は、説明だけでもお送りいただきたい。

10. BitKeeper には、bk helptool を使用してアクセスできる便利なヘルプユーティリティがある。

11. 注意: いずれのコミット (bk ci または bk citool) も、当社の内部メーリングリストへのチェンジセットを含んだメッセージの投稿や、ChangeSet コメントだけが含まれた通常の openlogging.org サブミッションをトリガすることはない。 一般に、コミットを使用する必要はない (公開ツリーは bk push を許可しないため) が、前述した bk diffs メソッドは使用する。

ChangeSet、コメント、ソースコードは、たとえば MySQL 4.1 の場合は http://mysql.bkbits.net:8080/mysql-4.1 で検索することもできます。

マニュアルは、独立したツリーに格納されています。このツリーは、以下のコマンドを使用してコピーすることができます。

```
shell> bk clone bk://mysql.bkbits.net/mysqldoc mysqldoc
```

MySQL Control Center と Connector/ODBC の公開 BitKeeper ツリーもあります。これらのツリーは、それぞれ以下のコマンドでコピーできます。

MySQL Control Center をコピーするには、以下のコマンドを使用してください。

```
shell> bk clone http://mysql.bkbits.net/mysqlcc mysqlcc
```

Connector/ODBC をコピーするには、以下のコマンドを使用してください。

```
shell> bk clone http://mysql.bkbits.net/myodbc3 myodbc3
```

2.3.5. MySQL のコンパイルに関する問題への対処

すべての MySQL プログラムは、gcc を使用して、Solaris または Linux 上で警告を受け取ることなくきれいにコンパイルできます。他のシステム上では、システムインクルードファイルの違いが原因で警告が発生することがあります。MIT-pthreads を使用している場合に発生する可能性のある警告については、項2.3.6.「MIT-pthreads に関する注意事項」 を参照してください。それ以外の問題については、以下の一覧をチェックしてください。

多くの問題は、解決に再 configure が必要です。再 configure が必要な場合は、以下の点に留意してください。

- configure を 1 度実行した後にもう 1 度実行すると、前回の起動時に収集した情報を使用することがある。この情報は、config.cache に格納されている。configure は、起動時にこのファイルを探し、ファイルが存在する場合は、その情報が現在も適切であるものと想定してファイルの内容を読み取る。再 configure する場合は、この想定は無効である。
- configure を実行するたびに make を実行して再コンパイルする必要がある。ただし、以前のビルドの古いオブジェクトファイルは別の設定オプションを使用してコンパイルされているので、最初にそれらのオブジェクトファイルを削除する必要がある。

古い検出情報やオブジェクトファイルが使用されるのを防ぐために、configure を再実行する前に以下のコマンドを実行します。

```
shell> rm config.cache
shell> make clean
```

または、make distclean を実行します。

以下の一覧で、MySQL のコンパイル時によく発生することがわかっているいくつかの問題を説明します。

- sql_yacc.cc のコンパイル時にここに示されているようなエラーが発生した場合は、メモリまたはスワップ領域が不足している可能性がある。

  ```
  Internal compiler error: program cc1plus got fatal signal 11
    か
  Out of virtual memory
    か
  Virtual memory exhausted
  ```

  ここで問題なのは、gcc がインライン関数を備えた sql_yacc.cc をコンパイルするために大量のメモリを必要とすることである。--with-low-memory オプションを指定して configure を実行する。

```
shell> ./configure --with-low-memory
```

このオプションによって、gcc を使用している場合は -fno-inline、それ以外のコンパイラを使用している場合は -O0 がコンパイル行に追加される。メモリとスワップ領域が十分にあり、不足することはないと思われる場合でも、--with-low-memory オプションを試してみる。この問題は、メモリの豊富なハードウェア構成のシステム上でも発生することがわかっており、通常は --with-low-memory オプションを指定することで修正される。

- デフォルトでは、configure はコンパイラ名として c++ を選び、GNU c++ は -lg++ をリンクする。gcc を使用している場合は、その動作によってコンフィギャ時に以下のような問題が発生することがある。

```
configure: error: installation or configuration problem:
C++ compiler cannot create executables.
```

g++、libg++、または libstdc++ に関連したコンパイル時の問題が発生することもある。

これらの問題の原因の 1 つは、g++ がないか、g++ があっても libg++ または libstdc++ がないことである。config.log ファイルを調べると、このファイルに、C++ コンパイラが機能しない正確な理由が記述されているはずである。これらの問題に対処するために、C++ コンパイラとして gcc を使用することができる。環境変数 CXX を "gcc -O3" にする。たとえば、以下のように指定する。

```
shell> CXX="gcc -O3" ./configure
```

gcc は、g++ と同様に C++ ソースをコンパイルするが、デフォルトでは libg++ または libstdc++ にはリンクしない。

当然ながら、これらの問題を修正するもう 1 つの方法は、g++、libg++、および libstdc++ をインストールすることである。しかし、libg++ または libstdc++ を MySQL と共に使用することはお勧めしない。mysqld のバイナリサイズを増やすだけで何の利点もないためである。 これらのライブラリのバージョンの中には、MySQL ユーザにとって未知の問題を過去に引き起こしたものもある。

GNU gcc バージョン 3 以降を使用しており、RAID 機能 (RAID テーブル型の詳細については 項6.5.3.「CREATE TABLE 構文」 を参照) を組み込んだ MySQL をコンパイルする場合も、C++ コンパイラとして gcc を使用する必要がある。--with-raid オプションを使用してコンパイルするように MySQL をコンフィギャしているときに、リンク段階で以下のいずれかのようなエラーが発生した場合は、前述の環境変数 CXX を定義して C++ コンパイラとして gcc を使用してみる。

```
gcc -O3 -DDBUG_OFF -rdynamic -o isamchk isamchk.o sort.o  libnisam.a
../mysys/libmysys.a ../dbug/libdbug.a ../strings/libmystrings.a -lpthread
-lz -lcrypt -lnsl -lm -lpthread
../mysys/libmysys.a(raid.o)(.text+0x79): In function `my_raid_create':
: undefined reference to `operator new(unsigned)'
../mysys/libmysys.a(raid.o)(.text+0xdd): In function `my_raid_create':
: undefined reference to `operator delete(void*)'
../mysys/libmysys.a(raid.o)(.text+0x129): In function `my_raid_open':
: undefined reference to `operator new(unsigned)'
../mysys/libmysys.a(raid.o)(.text+0x189): In function `my_raid_open':
: undefined reference to `operator delete(void*)'
../mysys/libmysys.a(raid.o)(.text+0x64b): In function `my_raid_close':
: undefined reference to `operator delete(void*)'
collect2: ld returned 1 exit status
```

- 以下のいずれかのようなエラーでコンパイルが失敗した場合は、make のバージョンを GNU make にアップグレード

する必要がある。

```
making all in mit-pthreads
make: Fatal error in reader: Makefile, line 18:
Badly formed macro assignment
 か
make: file `Makefile' line 18: Must be a separator (:
 か
pthread.h: No such file or directory
```

Solaris と FreeBSD は、複雑な make プログラムが組み込まれていることで知られている。

GNU make バージョン 3.75 は、正常に機能することが知られている。

- C コンパイラまたはC++ コンパイラが使用するフラグを定義する場合は、CFLAGS および CXXFLAGS という環境変数にそれらのフラグを追加して定義する。同様に、CC および CXX を使用してコンパイラ名を指定することもできる。以下に例を示す。

```
shell> CC=gcc
shell> CFLAGS=-O3
shell> CXX=gcc
shell> CXXFLAGS=-O3
shell> export CC CFLAGS CXX CXXFLAGS
```

さまざまなシステムで有用であることがわかっているフラグの一覧については、項2.2.8.「MySQL AB がコンパイルした MySQL バイナリ」 を参照。

- 以下のようなエラーメッセージが出力された場合は、gcc コンパイラをアップグレードする必要がある。

```
client/libmysql.c:273: parse error before `__attribute__'
```

gcc 2.8.1 は正しく機能することが知られているが、当社では gcc 2.95.X の使用をお勧めする。

- mysqld のコンパイル時に以下に示すようなエラーが発生した場合は、configure が accept()、getsockname()、または getpeername() の最後の引数の型を正しく検出していない。

```
cxx: Error: mysqld.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);
```

これを修正するには、config.h ファイルを編集する（このファイルは configure によって生成される）。以下の行を検索する。

```
/* Define as the base type of the last arg to accept */
#define SOCKET_SIZE_TYPE XXX
```

使用しているオペレーティングシステムに応じて、XXX を size_t または int に変更する（注意: configure は config.h を生成するので、configure を実行するたびにこの処理を行う必要がある）。

- sql_yacc.cc ファイルは、sql_yacc.yy から生成される。MySQL には生成済みのコピーが付属しているので、通常はビルドプロセスが sql_yacc.cc を作成する必要はない。ただし、このファイルを再作成する必要がある場合は、以下の工

ラーが出力される。

```
"sql_yacc.yy", line xxx fatal: default action causes potential...
```

このエラーは、使用している yacc のバージョンでは不十分であることを示している。 ほとんどの場合、bison （GNU バージョンの yacc ）をインストールして、現在の yacc の代わりに使用する必要がある。

- mysqld または MySQL クライアントをデバッグする必要がある場合は、--with-debug オプションを指定して configure を実行する。次に、再コンパイルして、クライアントを新しいクライアントライブラリにリンクする。 See 項E.2.「MySQL クライアントのデバッグ」。
- Linux （SuSE Linux 8.1 や Red Hat Linux 7.3 など ）で、以下のようなコンパイルエラーが出力された場合

```
libmysql.c:1329: warning: passing arg 5 of `gethostbyname_r' from incompatible pointer type
libmysql.c:1329: too few arguments to function `gethostbyname_r'
libmysql.c:1329: warning: assignment makes pointer from integer without a cast
make[2]: *** [libmysql.lo] Error 1
```

configure スクリプトは、デフォルトで GNU C++ コンパイラである g++ を使用して引数の正しい数を判別しようとする。しかし、g++ がインストールされていないと、このテストで正しくない結果が返される。この問題に対処するには、以下の 2 つの方法がある。

- GNU C++ g++ がインストールされていることを確認する。必要なパッケージの名前は、Linux ディストリビューションの種類に応じて gpp または gcc-c++ である。
- CXX 環境変数を gcc に設定して、C++ コンパイラとして gcc を使用する。

```
export CXX="gcc"
```

注意: 後で configure を再実行する必要があります。

2.3.6. MIT-pthreads に関する注意事項

ここでは、MIT-pthreads の使用に関係するいくつかの問題を説明します。

注意: Linux では、MIT-pthreads を使用しないで、インストールされている LinuxThreads 実装を使用してください。 See 項2.6.2.「Linux の注意事項（すべての Linux バージョン）」。

システムにネイティブスレッドサポートが用意されていない場合は、MIT-pthreads パッケージを使用して MySQL をビルドする必要があります。そのようなシステムとしては、旧バージョンの FreeBSD システム、SunOS 4.x、Solaris 2.4 以前などがあります。 See 項2.2.3.「MySQL がサポートしているオペレーティングシステム」。

注意: MySQL 4.0.2 からは、ソースディストリビューションに MIT-pthreads は含まれていません。このパッケージが必要な場合は、http://www.mysql.com/Downloads/Contrib/pthreads-1_60_beta6-mysql.tar.gz から別にダウンロードしてください。

ダウンロードしたら、このソースアーカイブを MySQL ソースディレクトリの最上位で展開します。これによって、mit-pthreads というサブディレクトリが新しく作成されます。

- ほとんどシステムでは、--with-mit-threads オプションを指定して configure を実行して、強制的に MIT-pthreads を使

用させることができる。

```
shell> ./configure --with-mit-threads
```

MIT-pthreads を使用している場合は、このコードへの変更を最小限に抑えたいので、ソースディレクトリ以外の場所へのビルドはサポートされていない。

- MIT-pthreads を使用するかどうかを判断するチェックは、サーバコードを処理する設定プロセスの一環としてしか行われない。--without-server を使用してディストリビューションをコンフィギャしてクライアントコードだけをビルドした場合は、クライアントは MIT-pthreads を使用するかどうかを判断できず、デフォルトで Unix ソケット接続を使用する。 一部のプラットフォームでは、MIT-pthreads のもとで Unix ソケットが機能しないため、クライアントプログラムを実行するときに -h または --host を使用する必要がある。

- MIT-pthreads を使用して MySQL をコンパイルした場合は、パフォーマンス上の理由から、デフォルトでシステムロックが無効になる。--external-locking オプションを使用して、システムロックの使用をサーバに指示することができる。これは、同じデータファイルに対して 2 つの MySQL サーバを実行できるようにする場合にのみ必要である（このような処理は推奨しない）。

- pthread bind() コマンドがソケットへのバインドに失敗し、エラーメッセージが表示されないことがある（これは、少なくとも Solaris 上では発生する）。したがって、サーバへのすべての接続が失敗するということになる。以下に例を示す。

```
shell> mysqladmin version
mysqladmin: connect to server at '' failed;
error: 'Can't connect to mysql server on localhost (146)'
```

この問題の解決策は、mysqld サーバを強制終了して再起動することである。 これは、当社でサーバを強制的に停止してすぐに再起動したときに、偶然見つかった方法である。

- MIT-pthreadsでは、sleep() システムコールに対して SIGINT（ブレーク）による割り込みを行うことはできない。これは、mysqladmin --sleep を実行したときにのみわかる。sleep() コールが終了してから、割り込みが実行されプロセスが停止する。

- リンク時に、以下のような警告メッセージが表示されることがある（少なくとも Solaris では発生する）。これらのメッセージは無視してかまわない。

```
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
```

- そのほかに、以下のような警告も無視してかまわない。

```
implicit declaration of function `int strtoll(...)'
implicit declaration of function `int strtoul(...)'
```

- MIT-pthreads と連携して動作する readline はまだない（これは、必要のないものであるが、興味のあるユーザもいる

かもしれない) 。

2.3.7. Windows 上でソースから MySQL をインストールする

ここでは、Windows 上でバージョン 4.1 以降のソースから MySQL バイナリをビルドする方法を説明します。標準のソースディストリビューションまたは最新の開発ソースが格納されている BitKeeper ツリーからバイナリをビルドする場合の手順が記載されています。

注意: ここに記載されている手順は、最新のソースディストリビューションまたは BitKeeper ツリーで入手した MySQL を Windows 上でテストするユーザだけを対象としています。 自分でソースからビルドした MySQL サーバを実稼動環境で使用することはお勧めしません。通常、最も望ましいのは、MySQL AB が特に Windows 上で最適なパフォーマンスを達成するようにビルドした、プリコンパイルされた MySQL バイナリディストリビューションを使用することです。 バイナリディストリビューションのインストール手順については、項2.1.1.「Windows への MySQL のインストール」 を参照してください。

Windows 上で MySQL をビルドするには、使用している Windows システムに以下のコンパイラとリソースがなければなりません。

- VC++ 6.0 コンパイラ (SP 4 または 5 とプリプロセッサパッケージで更新したもの) 。 プリプロセッサパッケージは、マクロアセンブラにとって必要である。 詳細については、以下のサイトを参照。 http://msdn.microsoft.com/vstudio/downloads/updates/sp/vs6/sp5/faq.aspx。
- 約 45 MB のディスク領域
- 64 MB の RAM

また、Windows 用の MySQL ソースディストリビューションも必要です。MySQL バージョン 4.1 以降のソースディストリビューションを入手する方法は、以下の2 とおりです。

1. 関心のあるバージョンの MySQL の MySQL AB がパッケージ化したソースディストリビューションを入手する。プリパッケージされたソースディストリビューションは、MySQL のリリース済みバージョン用のものが用意されており、http://www.mysql.com/downloads/ から入手できる。
2. 最新の BitKeeper 開発者ソースツリーから自分でソースディストリビューションをパッケージ化する。これを行う場合は、Unix システム上でパッケージを作成してから、使用している Windows システムに転送する必要がある (設定およびビルドのステップの中には Unix 上でしか動作しないツールが必要なものがあるため) 。したがって、BitKeeper を利用する場合は以下のものが必要である。
   - Unix を実行しているシステムか、Linux などの Unix ライクなシステム
   - そのシステム用の BitKeeper 3.0。BitKeeper は、http://www.bitkeeper.com/ から入手できる。

Windows ソースディストリビューションを使用する場合は、直接 項2.3.7.1.「VC++ を使用した MySQL のビルド」 に進んでください。BitKeeper ツリーからビルドする場合は、項2.3.7.2.「最新の開発ソースから Windows ソースパッケージを作成する」 に進んでください。

予想通りに動作しないものを見つけた場合や、Windows での現在のビルドプロセスの改良方法に関して提案がある場合は

、win32 メーリングリストにメッセージを送ってください。 See 項1.7.1.1.「MySQL メーリングリスト」。

2.3.7.1. VC++ を使用した MySQL のビルド

注意: MySQL 4.1 以降の VC++ ワークスペースファイルは、Microsoft Visual Studio 6.0 以降 (7.0/.NET) のバージョンと互換性があり、リリース前に MySQL AB のスタッフによってテストされています。

MySQL のビルドは、以下の手順で行います。

1. 作業ディレクトリを作成する (workdir など) 。
2. WinZip か、.zip ファイルを読み取ることのできるその他の Windows ツールを使用して、上記のディレクトリにソースディストリビューションをアンパックする。
3. VC++ 6.0 コンパイラを起動する。
4. [File] メニューで、 [Open Workspace] を選択する。
5. 作業ディレクトリにある mysql.dsw ワークスペースを開く。
6. [Build] メニューで、 [Set Active Configuration] メニューを選択する。
7. 画面上でクリックして、 [mysqld - Win32 Debug] を選択し、 [OK] をクリックする。
8. F7 キーを押し、デバッグサーバ、ライブラリ、およびクライアントアプリケーションのビルドを開始する。
9. 同様に、必要なリリースバージョンをコンパイルする。
10. プログラムおよびライブラリのデバッグバージョンは、client_debug ディレクトリおよび lib_debug ディレクトリにある。 プログラムおよびライブラリのリリースバージョンは、client_release ディレクトリおよび lib_release ディレクトリにある。 注意: デバッグバージョンとリリースバージョンの両方をビルドする場合は、 [Build] メニューで [build all] オプションを選択する。
11. サーバをテストする。上記の手順でビルドしたサーバでは、デフォルトで MySQL ベースディレクトリとデータディレクトリが C:\mysql および C:\mysql\data であると想定される。ソースツリーのルートディレクトリとそのデータディレクトリをベースディレクトリおよびデータディレクトリとして使用してサーバをテストする場合は、サーバにそれらのパス名を通知する必要がある。この処理を行うには、コマンドラインで --basedir オプションと --datadir オプションを指定するか、オプション設定ファイル (Windows ディレクトリにある C:\my.cnf ファイルまたは my.ini ファイル) に適切なオプションを配置する。使用したい既存のデータディレクトリが別の場所にある場合は、そのパス名を指定する。
12. 使用するサーバに応じて、client_release ディレクトリまたは client_debug ディレクトリからサーバを起動する。一般的なサーバの起動手順については、項2.1.1.「Windows への MySQL のインストール」 を参照。 別のベースディレクトリまたはデータディレクトリを使用する場合は、それに合わせて手順を変更する必要がある。
13. 設定に基づいてサーバがスタンドアロンまたはサービスとして稼動しているときに、client_release ディレクトリまたは client_debug ディレクトリにある mysql の対話型コマンドラインユーティリティからそのサーバに接続する。

ビルドしたプログラムが正しく動作することを確認したら、サーバを停止します。次に、以下の手順で MySQL をインストールします。

1. MySQL をインストールするディレクトリを作成する。たとえば、C:\mysql にインストールする場合は、以下のコマンドを使用する。

```
C:
mkdir \mysql
mkdir \mysql\bin
mkdir \mysql\data
mkdir \mysql\share
mkdir \mysql\scripts
```

その他のクライアントをコンパイルして MySQL にリンクする場合は、追加のディレクトリも作成する。

```
mkdir \mysql\include
mkdir \mysql\lib
mkdir \mysql\lib\debug
mkdir \mysql\lib\opt
```

MySQL のベンチマークを実行する場合は、以下のディレクトリを作成する。

```
mkdir \mysql\sql-bench
```

ベンチマークを実行するには、Perl が必要である。

2. workdir ディレクトリから以下のディレクトリを C:\mysql ディレクトリにコピーする。

```
copy client_release\*.exe C:\mysql\bin
copy client_debug\mysqld.exe C:\mysql\bin\mysqld-debug.exe
xcopy scripts\*.* C:\mysql\scripts /E
xcopy share\*.* C:\mysql\share /E
```

その他のクライアントをコンパイルして MySQL にリンクする場合は、以下のコマンドも実行する。

```
copy lib_debug\mysqlclient.lib C:\mysql\lib\debug
copy lib_debug\libmysql.* C:\mysql\lib\debug
copy lib_debug\zlib.* C:\mysql\lib\debug
copy lib_release\mysqlclient.lib C:\mysql\lib\opt
copy lib_release\libmysql.* C:\mysql\lib\opt
copy lib_release\zlib.* C:\mysql\lib\opt
copy include\*.h C:\mysql\include
copy libmysql\libmysql.def C:\mysql\include
```

MySQL のベンチマークを実行する場合は、以下のコマンドも実行する。

```
xcopy sql-bench\*.* C:\mysql\bench /E
```

Windows バイナリディストリビューションの場合と同様にサーバを設定して起動します。 See 項2.1.1.「Windows への MySQL のインストール」。

2.3.7.2. 最新の開発ソースから Windows ソースパッケージを作成する

現在の BitKeeper ソースツリーから最新の Windows ソースパッケージをビルドするには、以下の手順で行います。 注意: この手順は、Unix または Unix ライクなオペレーティングシステムを実行しているシステム上で行う必要があります (こ

の手順は、Linux などの場合に有効であることが知られています）。

1. MySQL の BitKeeper ソースツリーをコピーする（必要に応じて、バージョン 4.1 以降）。ソースツリーのコピー方法の詳細については、項2.3.4.「開発ソースツリーからのインストール」の説明を参照。

2. サーババイナリが必要なため、ディストリビューションをコンフィギャしてビルドする。これを行う 1 つの方法は、ソースツリーの最上位ディレクトリで以下のコマンドを実行することである。

   ```
   shell> ./BUILD/compile-pentium-max
   ```

3. ビルドプロセスが正常に完了したことを確認したら、ソースツリーの最上位ディレクトリから以下のユーティリティスクリプトを実行する。

   ```
   shell> ./scripts/make_win_src_distribution
   ```

   このスクリプトによって、Windows システム上で使用する Windows ソースパッケージが作成される。必要に応じて、このスクリプトに別のオプションを指定することができる。指定できるオプションは以下のとおりである。

   ```
   --debug   デバッグ。パッケージは作成しない。
   --tmp     tmp を指定する。
   --suffix  パッケージの名前のサフィックスを指定。
   --dirname ファイルをコピーするディレクトリ名。
   --silent  ファイルの処理経過の詳細を出力しない。
   --tar     tar.gz パッケージを .zip パッケージの代わりに作成。
   --help    ヘルプの表示。
   ```

   デフォルトでは、make_win_src_distribution は、mysql-VERSION-win-src.zip という名前の zip 形式の圧縮アーカイブを作成する。この場合、VERSION は 使用している MySQL ソースツリーのバージョンを表す。

4. 作成した Windows ソースパッケージを Windows マシンにコピーまたはアップロードする。ソースパッケージのコンパイルは、項2.3.7.1.「VC++ を使用した MySQL のビルド」 に記載されている手順で行う。

2.4. インストール後の設定とテスト

MySQL をインストールしたら（バイナリディストリビューションまたはソースディストリビューションから）、権限テーブルを初期化して、サーバを起動し、サーバが正常に動作することを確認する必要があります。システムの起動時およびシャットダウン時にサーバが自動的に起動および停止するように設定することもできます。

ソースディストリビューションからインストールする場合は、通常、以下のように権限テーブルをインストールしてサーバを起動します。

```
shell> ./scripts/mysql_install_db
shell> cd mysql_installation_directory
shell> ./bin/mysqld_safe --user=mysql &
```

バイナリディストリビューション（RPM パッケージまたは pkg パッケージ）の場合は、以下を実行します。

```
shell> cd mysql_installation_directory
shell> ./scripts/mysql_install_db
shell> ./bin/mysqld_safe --user=mysql &
```

mysql_install_db スクリプトは、すべての権限を管理する mysql データベースと、MySQL のテストに使用できる test データベースを作成します。さらに、mysql_install_db を実行するユーザと root ユーザの権限エントリも作成します。これらのエントリはパスワードなしで作成されます。 mysqld_safe スクリプトは、mysqld サーバを起動します (バージョン 4.0 より前の MySQL を使用している場合は、mysqld_safe ではなく safe_mysqld を使用します) 。

mysql_install_db は、古い権限テーブルを上書きしないので、どのような状況で実行しても安全です。test データベースが必要ない場合は、サーバを起動した後に mysqladmin -u root drop test を使用してこのデータベースを削除することができます。

テストは、MySQL ディストリビューションの最上位ディレクトリから簡単に実行できます。バイナリディストリビューションの場合、最上位ディレクトリはインストールディレクトリです (通常は /usr/local/mysql など) 。ソースディストリビューションの場合は、最上位ディレクトリは MySQL ソースツリーのメインディレクトリです。

以下の各項に示されるコマンドでは、BINDIR は mysqladmin や mysqld_safe などのプログラムがインストールされている場所のパスを表します。バイナリディストリビューションの場合、BINDIR はディストリビューション内の bin ディレクトリです。ソースディストリビューションの場合、configure を実行したときに /usr/local 以外のインストールディレクトリを指定していない限り、BINDIR は通常は /usr/local/bin です。 EXECDIR は、mysqld サーバがインストールされている場所です。バイナリディストリビューションの場合、これは BINDIR と同じです。ソースディストリビューションの場合、EXECDIR は通常は /usr/local/libexec です。

テストについて詳細に説明します。

1. 必要に応じて、mysqld サーバを起動して、MySQL 権限テーブルを初期化する。これは、通常 mysql_install_db スクリプトを使用して行う。

   ```
   shell> scripts/mysql_install_db
   ```

   MySQL 権限テーブルには、MySQL サーバに接続できるユーザのホストや権限の情報が記録されている。

   一般に、mysql_install_db は、初めて MySQL をインストールしたときにのみ実行する必要がある。したがって、既存のインストールをアップグレードする場合は、このステップはスキップできる (ただし、mysql_install_db は使用しても非常に安全で、既存のテーブルを更新することはないので、判断がつかない場合は mysql_install_db を実行してかまわない) 。

   mysql_install_db は、mysql データベースに 6 つのテーブル (user、db、host、tables_priv、columns_priv、および func) を作成する。初期権限の説明については、項4.4.4.「MySQL 権限の初期設定」 を参照。 簡単に説明すると、初期権限は、MySQL root ユーザにあらゆる操作を許可し、全ユーザに test という名前か、test_ で始まる名前を持つデータベースを作成または使用する許可を与える。

   権限テーブルを設定しないと、サーバを起動したときにログファイルに以下のエラーが記録される。

   ```
   mysqld: Can't find file: 'host.frm'
   ```

   MySQL バイナリディストリビューションの場合にも、./bin/mysqld_safe を実行して MySQL を起動しないとこのエラーが発生する。 See 項4.8.2.「mysqld_safe (mysqld のラッパ) 」。

   mysql_install_db は、Unix root アカウントで実行する必要があることがある。 ただし、必要に応じて、データベースディレクトリ内のファイルに対して読み取りと書き込みのアクセス権がある一般のアカウント (root 以外の Unix のア

カウント ）で、 MySQL サーバを実行することもできる。一般のアカウントで MySQL を実行する手順については、項A.3.2.「一般ユーザで MySQL を実行する方法」を参照。

mysql_install_db で問題が発生する場合は、項2.4.1.「mysql_install_db の実行に関する問題」 を参照。

MySQL ディストリビューションで提供される mysql_install_db スクリプトを実行する代わりに行えるいくつかの手段がある。

- mysql_install_db を実行する前に、このスクリプトを編集して、権限テーブルにインストールされている初期の権限を変更することができる。スクリプトの変更により、同じ権限を持つ複数のマシンに MySQL をインストールする場合に便利である。この場合、通常は、mysql.user テーブルと mysql.db テーブルにいくつかの INSERT ステートメントを追加するだけで済む。
- 権限テーブルをインストールした後にテーブルの内容を変更する場合は、mysql_install_db を実行して、mysql -u root mysql を使用して MySQL root ユーザとして権限テーブルに接続し、SQL ステートメントを発行して権限テーブルを直接修正する。
- 作成済みの権限テーブルを完全に作成し直すことができる。テーブルをすでにインストールしており、mysql_install_db を編集した後に作成し直す場合に、これを行う必要がある。

これらの代替手段の詳細については、項4.4.4.「MySQL 権限の初期設定」 を参照。

2. 以下のように MySQL サーバを起動する。

```
shell> cd mysql_installation_directory
shell> bin/mysqld_safe &
```

バージョン 4.0 より前の MySQL を使用している場合は、最後のコマンドの bin/mysqld_safe を bin/safe_mysqld に置き換える。

サーバの起動時に問題が発生した場合は、項2.4.2.「MySQL サーバの起動に関する問題」 を参照。

3. mysqladmin を使用して、サーバが稼動していることを確認する。以下のコマンドによって、サーバが稼動しており接続に応答していることをチェックするための簡単なテストが実行される。

```
shell> BINDIR/mysqladmin version
shell> BINDIR/mysqladmin variables
```

mysqladmin version からの出力は、使用しているプラットフォームと MySQL のバージョンによって少し異なるが、以下に示したものと類似した内容となる。

```
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 tables: 0
Queries per second avg: 0.000
Memory in use: 132K  Max memory used: 16773K
```

BINDIR/mysqladmin を使用するとほかにどのようなことが行えるのかを把握するには、--help オプションを指定して起動する。

4. サーバをシャットダウンできるかどうかを確認する。

```
shell> BINDIR/mysqladmin -u root shutdown
```

5. サーバを再起動できるかどうかを確認します。この確認は、mysqld_safe を使用するか、直接 mysqld を起動して行う。以下に例を示す。

```
shell> BINDIR/mysqld_safe --log &
```

mysqld_safe が失敗した場合は、MySQL のインストールディレクトリから実行する（インストールディレクトリ以外のディレクトリに移動している場合）。それでも失敗する場合は、項2.4.2.「MySQL サーバの起動に関する問題」 を参照。

6. いくつかの簡単なテストを実行して、サーバが稼動していることを確認する。 出力は、以下に示すものと類似した内容になる。

```
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_% |      |
+------+--------+------+
```

また、sql-bench ディレクトリ（MySQL インストールディレクトリの下）に、ベンチマークスィートがある。このベ

ンチマークスィートを使用すると、各種のプラットフォーム上での MySQL の動作を比較できる。ベンチマークスィートは、Perl で記述されており、Perl DBI モジュールを使用してデータベースに依存しないインタフェースを各種のデータベースに提供する。ベンチマークスィートを実行するには、以下の追加の Perl モジュールが必要である。

```
DBI
DBD-mysql
Data-Dumper
Data-ShowTable
```

これらのモジュールは、CPAN (http://www.cpan.org/) から入手できる。 See 項2.7.1.「Unix への Perl のインストール」。

sql-bench/Results ディレクトリには、さまざまなデータベースやプラットフォームのベンチマークスィートの実行結果が含まれている。すべてのテストを実行するには、以下のコマンドを実行する。

```
shell> cd sql-bench
shell> run-all-tests
```

sql-bench ディレクトリがない場合は、バイナリディストリビューション用の RPM を使用している可能性がある (ソースディストリビューションの RPM にはベンチマークディレクトリが含まれている)。その場合、ベンチマークスィートを使用するには、最初にベンチマークスィートをインストールする必要がある。MySQL バージョン 3.22 以降では、ベンチマークのコードとデータが含まれた mysql-bench-VERSION-i386.rpm という名前のベンチマーク RPM ファイルがある。

ソースディストリビューションを使用している場合は、tests サブディレクトリにあるテストを実行することもできる。たとえば、auto_increment.tst を実行するには、以下のコマンドを実行する。

```
shell> BINDIR/mysql -vvf test < ./tests/auto_increment.tst
```

結果は、./tests/auto_increment.res ファイルに出力される。

2.4.1. mysql_install_db の実行に関する問題

mysql_install_db スクリプトの目的は、新しい MySQL 権限テーブルを生成することです。このスクリプトがその他のデータに影響を及ぼすことはありません。 MySQL 権限テーブルがすでにインストールされている場合は、何も行いません。

権限テーブルを作成し直す場合は、mysqld サーバを停止して (サーバが稼動している場合)、以下のようなコマンドを実行します。

```
shell> mv mysql-data-directory/mysql mysql-data-directory/mysql-old
shell> mysql_install_db
```

ここでは、mysql_install_db の実行時に発生する可能性のある問題について説明します。

- mysql_install_db が権限テーブルをインストールしない

  mysql_install_db が権限テーブルのインストールに失敗し、以下のメッセージを表示した後に終了することがある。

  ```
  starting mysqld daemon with databases from XXXXXX
  ```

```
mysql daemon ended
```

この場合は、ログファイルを詳しく調べる必要がある。ログは、エラーメッセージに表示されている XXXXXX ディレクトリにある。ログには mysqld が起動しない原因が示される。何が起きたかわからない場合は、mysqlbug を使用してバグレポートを投稿するときにこのログを含める。 See 項1.7.1.3.「バグまたは問題を報告する方法」。

- mysqld デーモンがすでに稼動している

  この場合は、mysql_install_db を実行する必要はない。mysql_install_db は、MySQL を初めてインストールするときに 1 回だけ実行する必要がある。

- mysqld デーモンが稼動しているときに、別の mysqld デーモンをインストールできない

  この問題は、既存の MySQL インストールがある場合に、テストのためや、同時に 2 つのインストールを稼動させるために別の場所に新しいインストールを配置するときに発生する。一般に、2 つ目のサーバを実行するときに起きる問題は、そのサーバが既存のサーバと同じソケットやポートを使用しようとすることである。 この場合は、次のエラーメッセージが表示される。Can't start server: Bind on TCP/IP port: Address already in use または Can't start server: Bind on unix socket...。 See 項4.2.「同じマシン上で複数の MySQL サーバを実行する」。

- /tmp に対する書き込みアクセス権がない

  デフォルトの場所 (/tmp 内) にあるソケットファイルを作成するための書き込みアクセス権がない場合や、/tmp, 内にテンポラリファイルを作成するためのアクセス権がない場合に、mysql_install_db の実行時や、mysqld の起動時または使用時にエラーが表示される。

  以下のコマンドを実行して、別のソケットとテンポラリディレクトリを指定することができる。

  ```
  shell> TMPDIR=/some_tmp_dir/
  shell> MYSQL_UNIX_PORT=/some_tmp_dir/mysqld.sock
  shell> export TMPDIR MYSQL_UNIX_PORT
  ```

  項A.4.5.「MySQL ソケットファイル /tmp/mysql.sock の保護または変更方法」 を参照。

  some_tmp_dir には、自分が書き込みアクセス権を持つディレクトリのパスを指定する。 See 付録 F. 環境変数。

  この後、以下のコマンドを使用して、mysql_install_db を実行し、サーバを起動できる。

  ```
  shell> scripts/mysql_install_db
  shell> BINDIR/mysqld_safe &
  ```

- mysqld がすぐにクラッシュする

  2.0.7-5 より前のバージョンの glibc を備えた Red Hat バージョン 5.0 を実行している場合、glibc のすべてのパッチがインストールされていることを確認する必要がある。 MySQL メールアーカイブには、これに関する大量の情報がある。メールアーカイブへのリンクは、http://lists.mysql.com/ からオンラインで利用できる。 項2.6.2.「Linux の注意事項 (すべての Linux バージョン)」 も参照。

  また、--skip-grant-tables オプションを指定して mysqld を手動で起動し、mysql を使用して自分で特権情報を追加することもできる。

```
shell> BINDIR/mysqld_safe --skip-grant-tables &
shell> BINDIR/mysql -u root mysql
```

mysql から、mysql_install_db で SQL コマンドを手動で実行する。後で、必ず mysqladmin flush-privileges または mysqladmin reload を実行してサーバに権限テーブルの再ロードを指示する。

2.4.2. MySQL サーバの起動に関する問題

トランザクション（InnoDB、BDB）をサポートするテーブルを使用する場合は、最初に my.cnf ファイルを作成して、使用する予定のテーブル型用の起動オプションを設定しておく必要があります。 See 章 7. MySQL のテーブル型。

通常は、以下のいずれかの方法で mysqld サーバを起動します。

- mysql.server を実行する。このスクリプトは、主にシステムの起動とシャットダウンの際に使用される。詳細については、項2.4.3.「MySQL を自動的に起動および停止する」 を参照。
- mysqld_safe を実行する。このスクリプトは、mysqld の適切なオプションを判別し、それらのオプションを指定して mysqld を実行する。 See 項4.8.2.「mysqld_safe（mysqld のラッパ）」。
- Windows NT/2000/XP の場合は、項2.1.1.7.「Windows NT、2000、または XP での MySQL の起動」 を参照。
- mysqld を直接起動する。

起動した mysqld デーモンは、ディレクトリをデータディレクトリに変更します。このディレクトリで、mysqld は、ログファイルと pid（プロセスID）ファイルを書き込み、データベースを検索します。

データディレクトリ(datadir 変数)の場所は、ディストリビューションのコンパイル時に組み込まれます。ただし、mysqld が実際に存在するシステム上の場所以外の場所にデータディレクトリがあると、mysqld は正しく動作しません。パスに問題がある場合は、--help オプションを指定して mysqld を起動して、mysqld が許可するオプションとデフォルトのパス設定を確認することができます。mysqldへのコマンドライン引数として正しいパス名を指定して、デフォルト設定を無効にすることができます（これらのオプションは、mysqld_safe でも同様に使用できます）。

通常、mysqld に指示する必要があるのは、MySQL がインストールされているベースディレクトリだけです。ベースディレクトリの指定は、--basedir オプションを使用して行うことができます。また、--help を使用して、パスオプションを変更した場合の影響をチェックすることもできます（注意: --help は、mysqld コマンドラインの最後に指定する必要があります）。以下に例を示します。

```
shell> EXECDIR/mysqld --basedir=/usr/local --help
```

必要なパス設定を決定したら、--help オプションを指定しないでサーバを起動します。

どの方法でサーバを起動した場合でも、サーバが正常に起動しないときは、ログファイルをチェックして原因を突き止めることができるかどうか確認します。ログファイルは、データディレクトリ（通常、バイナリディストリビューションの場合は /usr/local/mysql/data、ソースディストリビューションの場合は /usr/local/var、Windows 上では \mysql\data\mysql.err）です。データディレクトリ内で host_name.err や host_name.log の形式の名前を持つファイルを探します。この場合、host_name は、使用しているサーバホストの名前です。次に、以下のコマンドを実行して、それらのファイルの最後の数行をチェックします。

```
shell> tail host_name.err
shell> tail host_name.log
```

ログファイル内で以下のような記述を探します。

```
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
```

これは、--bdb-no-recover を指定しないで mysqld を起動したことと、Berkeley DB がデータベースをリカバリするときにログファイルに問題があることを発見したことを意味しています。続行できるようにするには、データベースディレクトリにある既存の Berkeley DB のログファイルを、後で調べることができるような他の場所に移動します。ログファイルには、log.0000000001 という名前が付いています。この場合、数字は時間の経過とともに増えます

BDB テーブルをサポートしている mysqld を起動しようとしてコアを吐いた場合は、BDB リカバリログに問題があるかもしれません。その場合は、--bdb-no-recover を指定して mysqld を起動してみます。この方法でサーバが正常に起動した場合は、データディレクトリからすべての log.* ファイルを削除して、もう一度 mysqld を起動してみます。

以下のエラーが表示された場合は、mysqld が使用しようとした TCP/IP ポートまたはソケットファイルを、他のプログラム（または別の mysqld サーバ）がすでに使用していることを意味します。

```
Can't start server: Bind on TCP/IP port: Address already in use
 か
Can't start server: Bind on unix socket...
```

ps を使用して、別の mysqld サーバが稼動していないことを確認します。別のサーバが稼動していないことが確認できたら、コマンド telnet ホスト名 TCP/IPポート番号 を実行して、Enter キーを 2、3 回押します。telnet: Unable to connect to remote host: Connection refused のようなエラーメッセージが出力されない場合は、mysqld が使用しようとした TCP/IP ポートは他のプログラムによって使用されています。 項2.4.1.「mysql_install_db の実行に関する問題」 および 項4.2.「同じマシン上で複数の MySQL サーバを実行する」 を参照してください。

mysqld が現在稼動している場合は、以下のコマンドを実行して、mysqld が使用しているパス設定を調べることができます。

```
shell> mysqladmin variables
```

または

```
shell> mysqladmin -h 'your-host-name' variables
```

mysqld の起動時に Errcode 13（Permission deniedを表す）が表示された場合は、MySQL データベースまたはログディレクトリ内のファイルの読み取り/書き込み権限がなかったことを意味します。この場合、Unix root アカウントで mysqld を起動するか、関連するファイルおよびディレクトリを mysqld を実行する Unix アカウントが使用できるように、それらのファイルおよびディレクトリのアクセス権を変更します。

mysqld_safe によってサーバが起動されたのに、そのサーバに接続できない場合は、/etc/hosts に以下のようなエントリがあるかどうかを確認します。

```
127.0.0.1       localhost
```

この問題は、正常に機能するスレッドライブラリがなく、MySQL がMIT-pthreads を使用するようにコンフィギャされているシステムで発生します。

mysqld を起動させることができない場合は、トレースファイルを作成して問題を見つけます。 See 項E.1.2.「トレースファイルの作成」。

InnoDB テーブルを使用している場合は、項7.5.3.「InnoDB 起動オプション」 を参照してください。

BDB (Berkeley DB) テーブルを使用している場合は、項7.6.3.「BDB 起動オプション」 をよく理解しておいてください。

2.4.3. MySQL を自動的に起動および停止する

mysql.server スクリプトと mysqld_safe スクリプトを使用すると、システムの起動時にサーバを自動的に起動することができます。mysql.server は、サーバを停止する場合にも使用できます。

mysql.server スクリプトに start 引数または stop 引数を指定して起動すると、サーバを起動または停止することができます。

```
shell> mysql.server start
shell> mysql.server stop
```

mysql.server は、MySQL インストールディレクトリの下の share/mysql ディレクトリか、MySQL ソースツリーの support-files ディレクトリにあります。

注意: Linux RPM パッケージ (MySQL-server-VERSION.rpm) を使用している場合は、mysql.server スクリプトは /etc/init.d/mysql としてすでにインストールされています。したがって、手動でこのスクリプトをインストールする必要はありません。Linux RPM パッケージの詳細については、項2.1.2.「Linux への MySQL のインストール」 を参照してください。

Mac OS X では、別の MySQL Startup Item パッケージをインストールして、システムブートアップ時に MySQL が自動的に起動できるようにすることができます。 詳細については、項2.1.3.「Mac OS X への MySQL のインストール」 を参照してください。

mysql.server は、サーバを起動する前にディレクトリを MySQL インストールディレクトリに変更してから、mysqld_safe を起動します。 インストールしたバイナリディストリビューションが標準以外の場所にある場合は、mysql.server を編集する必要があります。mysqld_safe を実行する前に適切なディレクトリに変更する (cd) ように、mysql.server を修正します。サーバを特定のユーザとして実行する場合は、このセクションで説明するように適切な user 行を /etc/my.cnf ファイルに追加します。

mysql.server stop は、サーバにシグナルを送って停止させます。 mysqladmin shutdown を実行して、サーバを手動で停止させることもできます。

/etc/rc* ファイル内の適切な場所にこれらの起動コマンドおよび停止コマンドを追加する必要があります。

最新の Linux ディストリビューションでは、mysql.server ファイルを /etc/init.d ディレクトリ (旧バージョンの Red Hat システムの場合は /etc/rc.d/init.d) にコピーするだけで済みます。その後、以下のコマンドを実行して、システムブート時に MySQL が起動できるようにします。

```
shell> chkconfig --add mysql.server
```

FreeBSD では、通常、起動スクリプトは /usr/local/etc/rc.d/ に格納されます。 rc(8) マニュアルページには、このディレクトリにあるスクリプトの basename がシェルグロブパターン *.sh にマッチし、実行可能である場合のみ、実行されると述べられています。このディレクトリ内のその他のファイルやディレクトリは、自動的に無視されます。つまり、FreeBSD では、自動起動を有効にするには、ファイル mysql.server を /usr/local/etc/rc.d/mysql.server.sh としてインストールする必要があります。

一部のオペレーティングシステムでは、上記の方法の代わりに、/etc/rc.local または /etc/init.d/boot.local を使用して、ブートアップ時にその他のサービスを起動することもできます。この方法で MySQL を起動するには、以下のような記述を末尾に追加します。

```
shell> /bin/sh -c 'cd /usr/local/mysql; ./bin/mysqld_safe --user=mysql &'
```

グローバル /etc/my.cnf ファイルに mysql.server のオプションを追加することもできます。標準的な /etc/my.cnf ファイルは、以下のようになります。

```
[mysqld]
datadir=/usr/local/mysql/var
socket=/var/tmp/mysql.sock
port=3306
user=mysql

[mysql.server]
basedir=/usr/local/mysql
```

mysql.server スクリプトは、datadir、basedir、および pid-file の各オプションを認識します。.

以下の表に、各起動スクリプトが、オプション設定ファイルから読み取るオプショングループを示します。

スクリプト	オプショングループ
mysqld	[mysqld]、[server]、および [mysqld-major-version]
mysql.server	[mysql.server]、[mysqld]、および [server]
mysqld_safe	[mysqld]、[server]、および [mysqld_safe]

下位互換性のために、mysql.server は [mysql_server] グループも読み取り、mysqld_safe は [safe_mysqld] グループも読み取ります。しかし、これらの代わりに [mysql.server] グループと [mysqld_safe] グループを使用するようにオプション設定ファイルを更新する必要があります。

See 項4.1.2.「my.cnf オプション設定ファイル」。

2.5. MySQL のアップグレードとダウングレード

アップグレードを行う前に、既存のデータベースをバックアップする必要があります。

同じベースバージョンの MySQL である限り、MySQL のデータファイルを同じアーキーテクチャ上の異なるバージョンのプラットフォームに移動することができます。現在のベースバージョンは 4 です。

MySQL を運用中にキャラクタセットを変更する場合は、すべてのテーブルで myisamchk -r -q --set-character-set=charset を実行する必要があります。この処理を行わないと、キャラクタセットを変更したことによってソート順序も変更される

ことがあるため、インデックスが正しく順序付けられないことがあります。

新しいバージョンに不安を感じる場合は、いつでも、旧バージョンの mysqld の名前を mysqld-old-version-number のような名前に変更することができます。 そのようにしておくと、新しいバージョンの mysqld が予期しない動作をしたときに、その mysqld をシャットダウンして旧バージョンの mysqld で再起動するだけで済みます。

アップグレード後に、再コンパイルしたクライアントプログラムで Commands out of sync や予期しないコアダンプなどの問題が発生した場合は、プログラムのコンパイル時に旧バージョンのヘッダファイルやライブラリファイルを使用した可能性があります。この場合、mysql.h ファイルおよび libmysqlclient.a ライブラリの日付をチェックして、それらが新しい MySQL ディストリビューションのものであるかどうかを確認します。新しい MySQL ディストリビューションのものでない場合は、プログラムを再コンパイルしてください。

新しい mysqld サーバが起動しない、またはパスワードなしで接続できないなどの問題が発生した場合は、旧バージョンのインストールの my.cnf ファイルがないかどうかをチェックします。これは、program-name --print-defaults を使用してチェックすることができます。. プログラム名以外のものが出力された場合は、サーバの動作に影響を及ぼすアクティブな my.cnf ファイルがあります。

新しいリリースの MySQL をインストールするときは常に、Perl DBD-mysql モジュールの再ビルドと再インストールを行うことをお勧めします。同じことは、Python MySQLdb モジュールなどのその他の MySQL インタフェースについても当てはまります。

2.5.1. バージョン 4.0 から 4.1 へのアップグレード

MySQL 4.0 と MySQL 4.1 では、重大なバグを修正し、MySQL と ANSI SQL 標準との互換性を向上させるためにいくつかの目立つ動作が変更されています。これらの変更は、アプリケーションに影響を及ぼすことがあります。

一部の 4.1 の動作は、完全に 4.1 にアップグレードする前に 4.0 でテストすることができます。比較的最近の MySQL 4.0 リリース (4.0.12 以降) には、mysqld 用の --new 起動オプションが追加されています。

このオプションを使用すると、最も重大な変更が行われた 4.1 の動作が提供されます。 SET @@new=1 コマンドを使用して特定のクライアント接続に対してこれらの動作を有効にしたり、有効になっている場合は SET @@new=0 コマンドを使用して無効にすることができます。

4.1 の変更点によって影響を受けると思われる場合は、4.1 にアップグレードする前に、最新の MySQL 4.0 バージョンをダウンロードして、設定ファイルに以下を追加し、--new オプションを指定してその MySQL を実行することをお勧めします。

```
[mysqld-4.0]
new
```

このようにすれば、4.0 上で、新しい機能を動作させて、アプリケーションがそれらと連携して正常に機能するかどうかを、確認することができます。これは、4.1 以降への完全アップグレードを行うときに、苦労せずにスムーズに移行するのに役に立ちます。上記の方法でこれを行うと、誤って --new オプションを指定して 4.1 バージョンを実行することはありません。

以下では、アプリケーションに影響を及ぼす可能性のある変更点や、バージョン 4.1 にアップグレードするときに注意する必要のある変更点について説明します。

- TIMESTAMP は、'YYYY-MM-DD HH:MM:SS' 形式の文字列として返されるようになった (4.0.12 以降では、--new オプションを使用してこの点に関して 4.1 のように動作させることができる) 。バージョン 4.0 のように値を数値として

返すようにする場合は、TIMESTAMP カラムを取得するときに、それらのカラムに +0 を追加する。

```
mysql> SELECT ts_col + 0 FROM tbl_name;
```

TIMESTAMP カラムの桁数指定のサポートは中止された。 たとえば、カラムを TIMESTAMP(10) と宣言しても、(10) は無視される。

これらの変更は、標準 SQL に準拠するために必要であった。今後のバージョンで、さらに変更が行われ（この変更と下位互換性がある）、タイムスタンプ長で秒の小数部の桁数を必要なだけ指定できるようになる予定。

- 0xFFDF などのバイナリ値は、数値ではなく文字列であると見なされるようになった。これによって、文字列をバイナリ値として入力するのが便利なキャラクタセットのいくつかの問題が修正される。この変更では、バイナリ値を整数として数値比較する場合は、以下のように CAST() を使用する必要がある。

```
mysql> SELECT CAST(0xFEFF AS UNSIGNED INTEGER) < CAST(0xFF AS UNSIGNED INTEGER);
    -> 0
```

CAST() を使用しないと、以下のように文字列比較が行われる。

```
mysql> SELECT 0xFEFF < 0xFF;
    -> 1
```

数値コンテキストでのバイナリ項目の使用や、= 演算子を使用したバイナリ項目の比較は、従来どおりに機能する（4.0.13 以降では、--new オプションを使用してこの点に関して 4.1 のように動作させることができる）。

- DATE 値、DATETIME 値、または TIME 値を生成する関数の場合、クライアントに返される結果は時間の持つ型に修正されている。たとえば、MySQL 4.1 では、以下のような結果が返される。

```
mysql> SELECT CAST("2001-1-1" as DATETIME);
  -> '2001-01-01 00:00:00'
```

MySQL 4.0 では、以下のように異なる結果が返される。

```
mysql> SELECT CAST("2001-1-1" as DATETIME);
  -> '2001-01-01'
```

- AUTO_INCREMENT カラムに DEFAULT 値を指定できなくなった（4.0 では DEFAULT 値は自動的に無視される。4.1 ではエラーが発生する）。
- LIMIT は負の引数を受け付けなくなった。 -1 の代わりに 18446744073709551615 を使用する。
- SERIALIZE は、sql_mode 変数の有効なオプション値ではなくなった。代わりに、SET TRANSACTION ISOLATION LEVEL SERIALIZABLE を使用する。SERIALIZE は、mysqld の --sql-mode オプションとしても有効ではなくなった。代わりに、--transaction-isolation=SERIALIZABLE を使用する。
- すべてのテーブルと文字列カラムがキャラクタセットを持つようになった。 See 章 9. 各国キャラクタセットと Unicode。 キャラクタセット情報は、SHOW CREATE TABLE と mysqldump によって表示される（MySQL バージョン 4.0.6 以降は、新しいダンプファイルを読み取ることはできる。これより前のバージョンは、新しいダンプファイルを読み取ることはできない）。
- 4.1 では、.frm ファイルで使用されるテーブル定義形式が少し変更されている。4.0.11 以降の MySQL 4.0 バージョン

は、この新しい .frm 形式を直接読読み取ることができるが、これより前のバージョンは新しい形式を読み取ることはできない。 バージョン 4.1 のテーブルを 4.0.11 より前のバージョンに移動する必要がある場合は、mysqldump を使用する。 See 項4.9.7.「mysqldump (テーブル構造とデータのダンプ) 」。

- 同一の Windows マシン上で複数のサーバを実行する場合は、すべてのマシン上で異なる --shared_memory_base_name オプションを使用する必要がある。
- UDF 関数へのインタフェースが少し変更された。集約関数 XXX() ごとに xxx_clear() 関数を宣言しなければならなくなった。

一般に、以前の MySQL バージョンから 4.1 にアップグレードする場合は、以下のステップを実行する必要がある。

- 前述の変更点をチェックして、使用しているアプリケーションに影響を及ぼすような変更があるかどうかを確認する。
- 項D.2.「Changes in release 4.1.x (Alpha)」 を読んで、4.1 で使用できる重要な新機能を確認する。
- Windows 上で MySQL サーバを実行する場合は、項2.5.8.「Windows 上での MySQL のアップグレード」 も参照。
- アップグレード後に、パスワードを安全に処理するために必要な、長くなった新しい Password カラムを生成するように権限テーブルを更新する。テーブルの更新は、mysql_fix_privilege_tables を使用して行う。手順については、項2.5.6.「権限テーブルのアップグレード」 を参照。

4.1 では、より強力なセキュリティを実現するようにパスワードのハッシュメカニズムが変更されています。ただし、4.0 以前のクライアントライブラリを使用するクライアントがある場合に、互換性の問題が発生することがあります (4.1 にアップグレードされていないリモートホストからクライアントが接続するような状況では 4.0 クライアントがある可能性が非常に高いです) 。考えられるいくつかのアップグレード方針を以下に示します。旧バージョンのクライアントとの互換性の目標とセキュリティの目標の間でさまざまなトレードオフが存在します。

- 4.1 にアップグレードしない。動作は変化しないが、当然ながら 4.1 クライアント/サーバプロトコルによって提供される新機能も一切使用できない (MySQL 4.1 には、プリペアドステートメントや複数結果セットなどの機能を提供する拡張クライアント/サーバプロトコルが組み込まれている) 。 See 項11.1.4.「C API のプリペアドステートメント」。
- 4.1 にアップグレードして、mysql_fix_privilege_tables スクリプトを実行して、長いパスワードハッシュを格納できるように user テーブルの Password カラムを拡張する。ただし、--old-passwords オプションを指定してサーバを実行して、4.1 より前のバージョンのクライアントが引き続き短いハッシュのクライアントアカウントに接続できるようにして下位互換性を実現する すべてのクライアントが 4.1 にアップグレードされたら、最終的に --old-passwords サーバオプションの使用を中止できるまた、MySQL アカウントのパスワードを、より安全で新しい形式を使用するように変更することもできる
- 4.1 にアップグレードして、mysql_fix_privilege_tables スクリプトを実行して、user テーブルの Password カラムを拡張する。すべてのクライアントが 4.1 にアップグレードされていることがわかっている場合は、--old-passwords オプションを指定しないでサーバを実行する。代わりに、既存のすべてのアカウントのパスワードを、新しい形式を持つように変更する。純粋な 4.1 インストールが最も安全である。

クライアント認証とパスワード変更操作に関係するパスワードハッシュの予備知識については、項4.3.11.「MySQL 4.1 のパスワードハッシュ」 を参照してください。

2.5.2. バージョン 3.23 から 4.0 へのアップグレード

一般に、以前の MySQL バージョンから 4.0 にアップグレードする場合は、以下を実行する必要があります。

- アップグレード後に、権限テーブルを更新して新しい特権と機能を追加する。テーブルの更新は、mysql_fix_privilege_tables スクリプトを使用して行う。手順については、項2.5.6.「権限テーブルのアップグレード」を参照。
- 廃止予定のオプションを使用しないように MySQL 起動スクリプトまたはオプション設定ファイルを編集する。廃止予定のオプションについては、このセクションの後半で説明する。
- mysql_convert_table_format スクリプトを使用して、旧バージョンの ISAM ファイルを MyISAM ファイルに変換する（このスクリプトは、Perl スクリプトであるため、DBI をインストールする必要がある）。特定のデータベース内のテーブルを変換するには、以下のコマンドを使用する。

  ```
  shell> mysql_convert_table_format データベース名
  ```

  注意: このコマンドは、データベース内のすべてのテーブルが ISAM テーブルまたは MyISAM テーブルである場合にのみ使用する。他の型のテーブルを MyISAM に変換しないようにするには、コマンドラインのデータベース名の後ろに ISAM テーブルの名前を明示的に列記する。

  他に、テーブルを MyISAM に変換するための ALTER TABLE table_name TYPE=MyISAM ステートメントを各 ISAM テーブルに対して発行することもできる。

  特定のテーブルのテーブル型を確認するには、以下のステートメントを使用する。

  ```
  mysql> SHOW TABLE STATUS LIKE 'tbl_name';
  ```

- 共有ライブラリ（Perl DBD-mysql モードなど）を使用する MySQL クライアントがないことを確認する。libmysqlclient.so で使用されるデータ構造が変更されたので、そのようなクライアントがある場合は、それらのクライアントを再コンパイルする必要がある。 同じことは、Python MySQLdb モジュールなどのその他の MySQL インタフェースについても当てはまる。

上記の処理を行わなくても MySQL 4.0 は動作しますが、新しいセキュリティ特権を使用できないため、後で MySQL 4.1 以降にアップグレードするときに問題が発生することがあります。ISAM ファイル形式は、MySQL 4.0 では引き続き機能しますが、廃止予定であり、MySQL 4.1 では無効化されます（デフォルトではコンパイルされない）。代わりに、MyISAM テーブルが使用されます。

旧バージョンのクライアントは、バージョン 4.0 サーバと支障なく連携して動作します。

上記の処理を行った場合でも、MySQL 4.0 シリーズで問題が発生したときは、MySQL 3.23.52 以降にダウングレードできます。その場合、mysqldump を使用して、フルテキストインデックスを使用するすべてのテーブルをダンプし、ダンプファイルを 3.23 サーバに再ロードする必要があります。 これは、4.0 ではフルテキストインデックス作成に新しい形式が使用されるためです。

以下では、バージョン 4.0 にアップグレードするときに注意する必要のある点についてさらに詳しく説明します。

- MySQL 4.0 の mysql.user テーブルには多数の新しい権限がある。 See 項4.4.1.「GRANT および REVOKE の構文」。

これらの新しい権限を機能させるためには、権限テーブルを更新する必要がある。 更新の手順については、項2.5.6.「権限テーブルのアップグレード」 を参照。 権限テーブルを更新するまでは、すべてのユーザが SHOW DATABASES 権限、CREATE TEMPORARY TABLES 権限、および LOCK TABLES 権限を持つ。SUPER 権限と EXECUTE 権限は、PROCESS から自身の値を取得する。 REPLICATION SLAVE 権限と REPLICATION CLIENT 権限は、FILE から自身の値を取得する。

新しいユーザを作成するスクリプトがある場合は、新しい権限を使用するようにそれらのスクリプトを変更する必要がある。それらのスクリプトで GRANT コマンドを使用していない場合は、権限テーブルを直接変更する代わりに GRANT を使用するようにスクリプトを変更するよい機会である。

4.0.2 以降では、オプション --safe-show-database は廃止され、何の動作もしない。 See 項4.3.3.「セキュリティ関連の mysqld スタートアップオプション」。

バージョン 4.0.2 で、新しいユーザに対して Access denied エラーが発生した場合は、これまでは必要のなかったいくつかの新しい権限が必要であるかどうかをチェックする。特に、新しいスレーブに対しては FILE の代わりに REPLICATION SLAVE が必要。

- safe_mysqld は、mysqld_safe という名前に変更された。下位互換性のために、しばらくの間、バイナリディストリビューションには mysqld_safe へのシンボリックリンクとして safe_mysqld が含められている。
- InnoDB がバイナリディストリビューションにデフォルトで組み込まれるようになった。 MySQL をソースからビルドした場合は、デフォルトで InnoDB が組み込まれる。 InnoDB が有効になっているサーバを起動する場合、InnoDB を使用せずにメモリを節約したいときは、--skip-innodb サーバ起動オプションを使用する。InnoDB を組み込まないで MySQL をコンパイルするには、--without-innodb オプションを指定して configure を実行する。
- 起動パラメータ myisam_max_extra_sort_file_size および myisam_max_extra_sort_file_size は、バイト単位で指定されるようになった (4.0.3 より前のバージョンではメガバイト単位で指定されていた)。
- MyISAM/ISAM ファイルの外部システムロックは、デフォルトでオフに設定されるようになった。これは、--external-locking を実行してオンにすることができる (ただし、通常はオンにする必要はない)。
- 以下の起動変数および起動オプションの名前が変更された。

変更前の名前	変更後の名前
myisam_bulk_insert_tree_size	bulk_insert_buffer_size
query_cache_startup_type	query_cache_type
record_buffer	read_buffer_size
record_rnd_buffer	read_rnd_buffer_size
sort_buffer	sort_buffer_size
warnings	log-warnings
--err-log	--log-error (mysqld_safe 用)

record_buffer、sort_buffer、および warnings の各起動オプションは、MySQL 4.0 でも機能するが、廃止予定である。

- 以下の SQL 変数の名前が変更された。

変更前の名前	変更後の名前
SQL_BIG_TABLES	BIG_TABLES
SQL_LOW_PRIORITY_UPDATES	LOW_PRIORITY_UPDATES
SQL_MAX_JOIN_SIZE	MAX_JOIN_SIZE
SQL_QUERY_CACHE_TYPE	QUERY_CACHE_TYPE

変更前の名前は、MySQL 4.0 でも機能するが、廃止予定である。

- SET SQL_SLAVE_SKIP_COUNTER=# の代わりに SET GLOBAL SQL_SLAVE_SKIP_COUNTER=# を使用する必要がある。
- mysqld 起動オプション --skip-locking および --enable-locking は、それぞれ、--skip-external-locking および --external-locking という名前に変更された。
- SHOW MASTER STATUS は、バイナリログが有効になっていない場合は空のセットを返すようになった。
- SHOW SLAVE STATUS は、スレーブが初期化されていない場合は空のセットを返すようになった。
- mysqld の --temp-pool オプションはデフォルトで有効になるようになった。このオプションを使用すると、一部のオペレーティングシステム（最も顕著なのは Linux）でパフォーマンスが向上するためである。
- DOUBLE カラムと FLOAT カラムは、UNSIGNED フラグを受け付けるようになった（以前は、これらのカラムでは UNSIGNED は無視されていた）。
- MySQL 4.0.11 以降では、ORDER BY col_name DESC は、NULL 値を最後にする。3.23 と 4.0.11 より前の 4.0 バージョンでは、場合によって異なっていた。
- SHOW INDEX のカラムは 3.23 よりも 2 つ増えている（Null と Index_type）。
- CHECK、LOCALTIME、および LOCALTIMESTAMP は予約語になった。
- ビットごとの演算子（|、&、<<、>>、および ~）の結果は、符号なしになった。符号付きの結果が必要なコンテキストでこれらを使用すると、問題を引き起こすことがある。 See 項6.3.5.「キャスト関数」。
- 注意:整数値の減算でどちらか一方の整数値が UNSIGNED 型の場合、結果の値は符号なしになる。つまり、MySQL 4.0 にアップグレードする前に、アプリケーションをチェックして、符号なしのエンティティから値を差し引いて、負の答えを得たい場合や、整数カラムから符号なしの値を差し引く場合がないかを確認する。mysqld を起動するときに --sql-mode=NO_UNSIGNED_SUBTRACTION オプションを使用すると、この動作を無効にすることができる。 See 項6.3.5.「キャスト関数」。
- テーブルで MATCH ... AGAINST (... IN BOOLEAN MODE) を使用するには、REPAIR TABLE table_name USE_FRM でテーブルを再ビルドする必要がある。
- LOCATE() と INSTR() は、いずれかの引数がバイナリ文字列である場合にケース依存となる。それ以外の場合は、ケース非依存である。
- STRCMP() は、比較を実行するときに現在のキャラクタセットを使用するようになった。つまり、デフォルトの比較動作はケース非依存になった。

- HEX(string) は、string で 16 進数に変換された文字を返す。数値を 16 進数に変換したい場合は、必ず、数値引数を指定して HEX() を呼び出す。
- 3.23 では、INSERT INTO ... SELECT は、常に IGNORE が有効になっていた。 4.0.1 では、IGNORE が指定されていない限り、MySQL はエラーが発生するとデフォルトで停止する (そして、場合によってロールバックする) 。
- CFLAGS=-DUSE_OLD_FUNCTIONS を指定して MySQL をコンパイルしていない限り、旧バージョンの C API 関数 mysql_drop_db()、mysql_create_db()、および mysql_connect() はサポートされなくなった。ただし、代わりに新しい 4.0 API を使用するようにクライアントプログラムを変更する方が望ましい。
- MYSQL_FIELD 構造体の length および max_length は、unsigned int から unsigned long に変更された。関数の printf() クラスの引数としてこれらを使用した場合に、警告メッセージが生成されることがある以外は、この変更によって問題が発生することはない。
- テーブルからすべてのレコードを削除する場合や、削除されたレコードの数のカウントを取得する必要のない場合は、TRUNCATE TABLE を使用する必要がある (4.0 では DELETE FROM table_name がレコードカウントを返し、TRUNCATE TABLE は速度が上がった) 。
- TRUNCATE TABLE または DROP DATABASE を実行するときに、アクティブな LOCK TABLES またはトランザクションがあると、エラーが発生する。
- 整数を使用して BIGINT カラムに値を格納する必要がある (MySQL 3.23 では文字列を使用していた) 。引き続き文字列を使用することもできるが、整数を使用する方が効率的である。
- SHOW OPEN TABLES の形式が変更された。
- マルチスレッド化されたクライアントは、mysql_thread_init() および mysql_thread_end() を使用する必要がある。 See 項11.1.14.「スレッドクライアントの作成方法」。
- Perl DBD::mysql モジュールを再コンパイルする場合は、DBD-mysql バージョン 1.2218 以降を入手する必要がある。これは、1.2218 より前のバージョンの DBD モジュールでは、廃止になった mysql_drop_db() コールが使用されるためである。 バージョン 2.1022 以降が推奨される。
- 4.0 では、RAND(seed) は 3.23 の場合とは異なる乱数系を返す。この変更は、RAND(seed) と RAND(seed+1) をさらに明確に区別するために行われた。
- IFNULL(A,B) によって返されるデフォルトの型は、A および B の型のうちで、より '総称的な' 方が設定されるようになった (総称性の度合いの高い順に並べると、文字列、REAL、INTEGER となる) 。

Windows 上で MySQL サーバを実行する場合は、項2.5.8.「Windows 上での MySQL のアップグレード」 も参照してください。レプリケーションを使用する場合は、項4.11.2.「レプリケーション実装の概要」 も参照してください。

2.5.3. バージョン 3.22 から 3.23 へのアップグレード

MySQL バージョン 3.23 は、新しい MyISAM 型と以前の ISAM 型のテーブルをサポートします。バージョン 3.23 で以前のテーブルを使用する場合に、それらのテーブルを変換する必要はありません。デフォルトでは、新しいテーブルはすべて、MyISAM 型で作成されます (ただし、--default-table-type=isam オプションを指定して mysqld を起動した場合を除きます) 。ISAM テーブルは、ALTER TABLE table_name TYPE=MyISAM または Perl スクリプト mysql_convert_table_format を使用して MyISAM 形式に変換することができます。

バージョン 3.22 および 3.21 のクライアントは、バージョン 3.23 サーバと支障なく連携して動作します。

以下では、バージョン 3.23 にアップグレードするときに注意する必要のある事項について説明します。

- tis620 キャラクタセットを使用するテーブルはすべて、myisamchk -r または REPAIR TABLE で修復する必要がある。
- シンボリックにリンクされたデータベースに DROP DATABASE を実行した場合、リンクと元のデータベースの両方が削除される（3.22 では、configure は readlink() システムコールの使用可能性を検出しないので、これは行われなかった）。
- OPTIMIZE TABLE は、MyISAM テーブルのみに対応するようになった。 これ以外のテーブル型のテーブルを最適化するには、ALTER TABLE を使用する。 OPTIMIZE TABLE の実行中は、他のスレッドに使用されるのを防ぐためにテーブルがロックされるようになった。
- MySQL クライアント mysql は、デフォルトで --no-named-commands (-g) を指定して起動されるようになった。このオプションは、--enable-named-commands (-G) を指定して無効にすることができる。場合によって、これによって非互換性の問題が発生することがある（セミコロンなしの名前付きコマンドを使用する SQL スクリプトなどの場合）。第 1 行目のロング形式のコマンドは引き続き有効である。
- 日付の部分を処理する日付関数（MONTH() など）は、0000-00-00 日付の場合に 0 を返すようになった（MySQL 3.22 では、これらの関数は NULL を返していた）。
- ISAM テーブルに german 文字ソート順序を使用している場合は、isamchk -r を使用してそれらのテーブルを修復する必要がある。これは、このソート順序にいくつかの変更が加えられたためである。
- IF() のデフォルトの戻り値の型は、最初の引数のみに依存するのではなく、両方の引数に依存するようになった。
- AUTO_INCREMENT カラムを使用して負の数を格納しないようにする。負の数を格納した場合に、-1 から 0 にラップするときに問題が発生したためである。AUTO_INCREMENT カラムには 0 を格納してはならない。CHECK TABLE は、0 値に対して警告を返す。0 値はテーブルをダンプしてリストアした場合に、変化することがあるためである。MyISAM テーブルの AUTO_INCREMENT は、より低いレベルで処理されるようになり、以前よりもかなり高速になっている。さらに、MyISAM テーブルでは、テーブルからレコードを削除した場合でも、古い番号は再利用されなくなった。
- CASE、DELAYED、ELSE、END、FULLTEXT、INNER、RIGHT、THEN、および WHEN は予約語になった。
- FLOAT(X) は、真の浮動小数点型になり、小数部の桁数が固定された値でなくなった。
- DECIMAL(length,dec) 型を使用してカラムを宣言するときに、length 引数に符号と小数点の位置が含まれなくなった。
- TIME 文字列は、以下のいずれかの形式にしなければならなくなった。 [[[DAYS] [H]H:]MM:]SS[.fraction] または [[[[[H]H]H]H]MM]SS[.fraction]。
- LIKE は、= 演算子の場合と同じ文字比較ルールを使用して文字列を比較するようになった。旧バージョンの動作が必要な場合は、CXXFLAGS=-DLIKE_CMP_TOUPPER フラグを指定して MySQL をコンパイルする。
- REGEXP は、どちらの文字列もバイナリ文字列でない場合はケース非依存になった。
- MyISAM（.MYI）テーブルをチェックまたは修復する場合は、CHECK TABLE ステートメントまたは myisamchk コマンドを使用する必要がある。ISAM（.ISM）テーブルの場合は、isamchk コマンドを使用する。

- mysqldump ファイルに MySQL バージョン 3.22 とバージョン 3.23 の間の互換性を持たせたい場合は、mysqldump に --opt オプションまたは --all オプションを指定しない。
- DATE_FORMAT() へのすべてのコールをチェックして、各書式文字の前に '%' があることを確認する（MySQL バージョン 3.22 以降で、この構文はすでに許可されている）。
- mysql_fetch_fields_direct() は関数になり（これまではマクロ）、MYSQL_FIELD の代わりに MYSQL_FIELD へのポインタを返すようになった。
- mysql_num_fields() は、MYSQL* オブジェクトに対して使用できなくなった（引数として MYSQL_RES* 値をとる関数になった）。MYSQL* オブジェクトには、代わりに mysql_field_count() を使用する。
- MySQL バージョン 3.22 では、SELECT DISTINCT ... の出力は、ほとんどの場合ソートされた。バージョン 3.23 では、ソートされた出力を取得するには、GROUP BY または ORDER BY を使用する必要がある。
- SUM() は、一致するレコードがない場合に 0 の代わりに NULL を返すようになった。これは、SQL-99 に対応したものである。
- NULL 値を持つ AND または OR は、0 の代わりに NULL を返すようになった。これは、主に、NOT NULL = NULL のように、AND/OR 式で NOT を使用するクエリに影響を及ぼす。
- LPAD() および RPAD() は、結果の文字列が length 引数より長い場合に、その文字列を短くするようになった。

2.5.4. バージョン 3.21 から 3.22 へのアップグレード

バージョン 3.21 とバージョン 3.22 の間では互換性に影響するような変更は行われていません。 唯一の落とし穴は、DATE 型カラムで作成された新しいテーブルでは、新しい方法で日付が格納されることです。旧バージョンの mysqld からこれらの新しいカラムにアクセスすることはできません。

MySQL バージョン 3.22 をインストールした後に、新しいサーバを起動して、mysql_fix_privilege_tables スクリプトを実行する必要があります。このスクリプトによって、GRANT コマンドを使用するために必要な新しい権限が追加されます。この処理をしないと、ALTER TABLE、CREATE INDEX、または DROP INDEX を使用しようとしたときに、Access denied が返されます。権限テーブルの更新の手順については、項2.5.6.「権限テーブルのアップグレード」 を参照してください。

mysql_real_connect() への C API インタフェースは変更されました。この関数を呼び出す旧バージョンのクライアントプログラムがある場合は、新しい db 引数に 0 を指定します（または db 要素を送信するようにクライアントのコードを作成し直します）。また、mysql_real_connect() を呼び出す前に mysql_init() を呼び出す必要もあります。この変更は、新しい mysql_options() 関数が MYSQL ハンドラ構造にオプションを保存できるようにするために行われました。

mysqld 変数 key_buffer は、key_buffer_size という名前に変更されました。ただし、起動ファイルでは以前の名前を引き続き使用することができます。

2.5.5. バージョン 3.20 から 3.21 へのアップグレード

バージョン 3.20.28 より前のバージョンを実行していて、バージョン 3.21 に切り替える場合は、以下のことを行う必要があります。

--old-protocol オプションを指定して mysqld バージョン 3.21 サーバを起動して、そのサーバをバージョン 3.20 ディスト

リビューションからのクライアントと共に使用することができます。この場合、新しいクライアント関数 mysql_errno() は、サーバエラーは返さず、CR_UNKNOWN_ERROR だけを返します (ただし、この関数はクライアントエラーに対しては機能します)。また、サーバは、新しい方式ではなく、3.21 より前の古い password() チェック方式を使用します。

mysqld に --old-protocol オプションを使用しない場合は、以下の変更を行う必要があります。

- すべてのクライアントコードを再コンパイルする。ODBC を使用している場合は、新しい MyODBC 2.x ドライバを入手すること。
- scripts/add_long_password スクリプトを実行して、mysql.user テーブルの Password フィールドを CHAR(16) に変換する。
- すべてのパスワードを mysql.user テーブルに割り当て直す (31 ビットのパスワードではなく 62 ビットのパスワードを取得するため)。
- テーブル形式は変更されていないので、テーブルを変換する必要はない。

MySQL バージョン 3.20.28 以降では、クライアントに影響を及ぼすことなく新しい user テーブル形式を処理することができます。バージョン 3.20.28 より前の MySQL バージョンを使用している場合、user テーブルを変換すると、パスワードはその MySQL では無効になります。したがって、安全のために、最初にバージョン 3.20.28 以上にアップグレードしてから、バージョン 3.21 にアップグレードしてください。

新しいクライアントコードは 3.20.x mysqld サーバと連携して動作します。したがって、3.21.x で問題が発生した場合は、クライアントを再コンパイルしなくても既存の 3.20.x サーバを使用することができます。

mysqld に --old-protocol オプションを使用しないと、旧バージョンのクライアントは接続できず、以下のエラーメッセージを出力します。

```
ERROR: Protocol mismatch. Server Version = 10 Client Version = 9
```

新しい Perl DBI/DBD インタフェースは、旧バージョンの mysqlperl インタフェースもサポートします。mysqlperl を使用する場合に唯一変更する必要があるのは、connect() 関数の引数です。 新しい引数は、host、database、user、および password です (注意: user 引数と password 引数は順序が入れ替わりました)。 See 項11.5.2.「DBI インタフェース」。

以下の変更は、旧バージョンのアプリケーションのクエリに影響を及ぼすことがあります。

- HAVING は ORDER BY 節の前に指定しなければならなくなった。
- LOCATE() のパラメータの順序が入れ替えられた。
- いくつかの新しい予約語がある。最も注目すべきものは、DATE、TIME、および TIMESTAMP である。

2.5.6. 権限テーブルのアップグレード

一部のリリースでは、権限テーブル (mysql データベース内のテーブル) の構造が変更され、新しい権限または機能が追加されています。新しいバージョンの MySQL に更新したときに権限テーブルが確実に最新のものであるようにするには、権限テーブルも更新する必要があります。

Unix システムまたは Unix ライクなシステムでは、以下のように mysql_fix_privilege_tables スクリプトを実行して権限テーブルを更新します。

```
shell> mysql_fix_privilege_tables
```

このスクリプトは、サーバの稼動中に実行しなければなりません。このスクリプトは、ローカルホスト上で稼動しているサーバに root として接続します。 root アカウントがパスワードを必要とする場合は、コマンドラインでパスワードを指定します。MySQL 4.1 以降のバージョンの場合は、以下のようにパスワードを指定します。

```
shell> mysql_fix_privilege_tables --password=root_password
```

MySQL 4.1 より前のバージョンの場合は、以下のようにパスワードを指定します。

```
shell> mysql_fix_privilege_tables root_password
```

mysql_fix_privilege_tables スクリプトは、権限テーブルを現行の形式に変換するために必要な処理を実行します。スクリプトの実行中にいくつかの Duplicate column name 警告が表示されることがあります。これらの警告は無視してかまいません。

スクリプトの実行後に、サーバを停止してから、再起動します。

Windows システムでは、MySQL 4.0.15 までは、権限テーブルを簡単に更新する方法はありません。バージョン 4.0.15 以降では、mysql クライアントを使用して実行できる mysql_fix_privilege_tables.sql SQL スクリプトが MySQL ディストリビューションに含まれています。MySQL が C:\mysql にインストールされている場合、コマンドは以下のようになります。

```
C:\mysql\bin> mysql -u root -p mysql

mysql> SOURCE C:\mysql\scripts\mysql_fix_privilege_tables.sql
```

MySQL がこれ以外のディレクトリにインストールされている場合は、パス名をそれに合わせて変更してください。

このコマンドは、root のパスワードの入力を求めるプロンプトを表示します。プロンプトが表示されたらパスワードを入力します。

Unix の手順の場合と同様に、mysql が mysql_fix_privilege_tables.sql スクリプトのステートメントを処理しているときにいくつかの Duplicate column name 警告が表示されることがあります。これらの警告は、無視してかまいません。

スクリプトの実行後に、サーバを停止してから、再起動します。

2.5.7. 別のアーキーテクチャへの移行

MySQL バージョン 3.23 を使用している場合、同じ浮動小数点形式をサポートする異なるアーキーテクチャ間で、MyISAM テーブルの .frm、.MYI、および .MYD の各ファイルをコピーできます (MySQL はバイトスワップの問題を処理します) 。 See 項7.1.「MyISAM テーブル」。

MySQL ISAM データファイルおよびインデックスファイル (それぞれ、.ISD および *.ISM) は、アーキーテクチャ依存であり、場合によっては OS にも依存します。現在のマシンとは異なるアーキーテクチャまたは OS を持つ別のマシンにアプリケーションを移動したい場合、そのマシンにファイルを単純にコピーするだけでは、データベースを移動することはできません。代わりに、mysqldump を使用してください。

mysqldump は、デフォルトで SQL ステートメントが格納されたファイルを作成します。 ファイルを移動先のマシンに転送し、mysql クライアントにデータとして入力します。

mysqldump --help を実行して、使用可能なオプションを確認します。 新しいバージョンの MySQL にデータを移動する場合は、高速でコンパクトなダンプを行うために、新しいバージョンで mysqldump --opt を使用してください。

マシン間でデータベースを移動するための最も簡単な (ただし、最も迅速というわけではありません) 方法は、データベースが置かれているマシンで以下のコマンドを実行することです。

```
shell> mysqladmin -h 'other hostname' create db_name
shell> mysqldump --opt db_name \
       | mysql -h 'other hostname' db_name
```

低速のネットワークを通じてリモートマシンのデータベースをコピーする場合は、以下のコマンドを使用します。

```
shell> mysqladmin create db_name
shell> mysqldump -h 'other hostname' --opt --compress db_name \
       | mysql db_name
```

結果をファイルに保存し、そのファイルを移動先のマシンに転送して、そのマシン上のデータベースにロードすることもできます。たとえば、以下のように、移動元のマシン上のファイルにデータベースをダンプします。

```
shell> mysqldump --quick db_name | gzip > db_name.contents.gz
```

(この例で作成されるファイルは圧縮されています。) データベースの内容が格納されたファイルを移動先のマシンに転送し、そのマシン上で以下のコマンドを実行します。

```
shell> mysqladmin create db_name
shell> gunzip < db_name.contents.gz | mysql db_name
```

また、mysqldump と mysqlimport を使用してデータベースを転送することもできます。 大きなテーブルの場合、この方法を使用すると、mysqldump だけを使用した場合よりもはるかに処理が速くなります。 以下のコマンドでは、DUMPDIR は、mysqldump からの出力を格納するのに使用するディレクトリの完全パス名です。

まず、以下のように、出力ファイル用のディレクトリを作成し、データベースをダンプします。

```
shell> mkdir DUMPDIR
shell> mysqldump --tab=DUMPDIR db_name
```

次に、DUMPDIR ディレクトリ内のファイルを移動先マシン上の対応するディレクトリに転送し、そのマシン上の MySQL にロードします。

```
shell> mysqladmin create db_name          # データベース作成
shell> cat DUMPDIR/*.sql | mysql db_name   # テーブル作成
shell> mysqlimport db_name DUMPDIR/*.txt   # load data into tables
```

また、mysql データベースも忘れずにコピーしてください。このデータベースには、権限テーブル (user、db、host) が格納されているためです。mysql データベースを配置するまでは、移動先のマシン上で MySQL root ユーザとしてコマンドを実行する必要があります。

mysql データベースを移動先のマシンにデータベースをインポートしたら、mysqladmin flush-privileges を実行して、サー

バに権限テーブル情報を再ロードさせます。

2.5.8. Windows 上での MySQL のアップグレード

Windows 上で MySQL をアップグレードする場合は、以下の手順に従って行います。

1. 最新のWindows 用 MySQL ディストリビューションをダウンロードする。
2. 保守のためのサーバの停止が許される、使用率の低い時間帯を選ぶ。
3. アクティブになっているユーザに保守のために停止することを通知する。
4. 稼動している MySQL サーバを停止する (MySQL をサービスとして実行している場合は NET STOP mysql または Services ユーティリティ、それ以外の場合は mysqladmin shutdown を使用する) 。
5. WinMySQLAdmin プログラムを実行している場合は、このプログラムを終了する。
6. 現在使用中のデータ(デフォルトでは c:\mysql\data) を念のためバックアップする。
7. WinZip の [Install] ボタンをクリックし、インストールスクリプトのインストールステップに従って、Windows ディストリビューションのインストールスクリプトを実行する。
8. 旧バージョンの MySQL (通常は C:\mysql にある) を上書きするか、C:\mysql4 などの別のディレクトリにインストールする。旧バージョンを上書きすることを推奨する。ただし、data が上書きされないように注意する。
9. サーバを再起動する (MySQL をサービスとして実行している場合は NET START mysql、それ以外の場合は直接 mysqld を起動する) 。
10. 権限テーブルを更新する。更新の手順については、項2.5.6.「権限テーブルのアップグレード」 を参照。

考えられるエラー状況は以下のとおりです。

```
A system error has occurred.
System error 1067 has occurred.
The process terminated unexpectedly.
```

このエラーは、my.cnf ファイル (デフォルトでは C:\my.cnf) に、MySQL が認識できないオプションが含まれていることを示しています。MySQL が my.cnf ファイルを使用しないようにするためにこのファイルの名前を my_cnf.old などに変更して、MySQL を再起動して、これが事実であるかどうかを確認します。事実であることを確認したら、原因となったオプションを特定する必要があります。新しい my.cnf ファイルを作成して、サーバの起動が失敗する原因となるオプションを特定するまで、古いファイルの各パートをそのファイルに移動します (1 つのパートを移動するたびにサーバを再起動します) 。

2.6. オペレーティングシステム固有の注意事項

2.6.1. Windows の注意事項

ここでは、Windows 上で MySQL を使用する場合に固有の問題について説明します。

2.6.1.1. SSH を使用して Windows から MySQL にリモート接続する

ここでは、SSH を使用してリモートの MySQL サーバとの安全な接続を確立するための接続方法について説明します（David Carlson 記（<dcarlson@mplcomm.com>））。

1. 使用しているマシンに SSH クライアントをインストールする。使用してみて最も優れていた有償の SSH クライアントは、http://www.vandyke.com/ の SecureCRT である。 もう 1 つは、http://www.f-secure.com/ の f-secure である。フリーソフトウェアについては、Google（http://directory.google.com/Top/Computers/Security/Products_and_Tools/Cryptography/SSH/Clients/Windows/）で検索できる。
2. Windows SSH クライアントを起動する。 Host_Name = yourmysqlserver_URL_or_IP を設定する。userid=your_userid を設定して、サーバにログインする。この userid 値は、MySQL アカウントのユーザ名と同じでなくてもかまわない。
3. ポート転送を設定する。リモート転送（local_port: 3306、remote_host: yourmysqlservername_or_ip、remote_port: 3306 を設定）か、ローカル転送（port: 3306、host: localhost、remote port: 3306 を設定）を実行する。
4. すべてのものを保存する。保存しないと、次回も上記の設定を行わなければならなくなる。
5. 作成した SSH セッションでサーバにログインする。
6. Windows マシン上で、ODBC アプリケーション（Access など）を起動する。
7. Windows 上に新しいファイルを作成し、通常どおりに ODBC ドライバを使用して MySQL にリンクする。ただし、MySQL ホストサーバとして、yourmysqlservername ではなく localhost を入力する。

これで、SSH を使用して暗号化された、MySQL への ODBC 接続が確立されました。

2.6.1.2. Windows 上での MySQL クライアントのコンパイル

ソースファイル内では、mysql.h の前に my_global.h をインクルードしてください。

```
#include <my_global.h>
#include <mysql.h>
```

my_global.h には、Windows 上でプログラムをコンパイルする場合に Windows との互換性のために必要なその他のファイル（windows.h など）がインクルードされています。

コードは、動的な libmysql.lib ライブラリ（要求に応じて libmysql.dll をロードする単なるラッパ）にリンクすることも、静的な mysqlclient.lib ライブラリにリンクすることもできます。

注意: MySQL クライアントライブラリはスレッド化されたライブラリとしてコンパイルされるので、コードもマルチスレッド化されるようにコンパイルする必要があります。

2.6.1.3. Windows 上の MySQL と Unix 上の MySQL の比較

Windows 用の MySQL は、非常に安定していることを自ら証明しました。Windows バージョンの MySQL には、それに対応する Unix バージョンと同じ機能が組み込まれていますが、以下の例外があります。

- Windows 95 とスレッド

Windows 95 は、スレッド生成のたびに約 200 バイトのメインメモリをリークする。 MySQL に接続するたびに新しいスレッドが生成されるため、サーバが多数の接続を処理する場合は、Windows 95 上で mysqld を長時間実行しない。その他のバージョンの Windows ではこのバグはない。

- 同時読み取り

  MySQL は、pread() 呼び出しおよび pwrite() 呼び出しに依存して、INSERT と SELECT の混在を可能にしている。現時点では、当社は相互排他ロック (mutex) を使用して pread()/pwrite() をエミュレートしている。長期的には、処理速度を向上させるためにNT/2000/XP 上で readfile()/writefile() インタフェースを使用できるように、ファイルレベルのインタフェースを仮想インタフェースに置き換える予定である。 現在の実装では、MySQL が使用できるオープンファイルの数は 1024 個に制限されている。したがって、NT/2000/XP 上では、Unix の場合と同じ数の同時実行スレッドを実行することはできない。

- ブロック読み取り

  MySQL は、接続ごとにブロック読み取りを使用する。これには、以下のような意味がある。

  - Unix バージョンの MySQL の場合のように 8 時間後に接続が自動的に切断されるということはない。
  - 接続がハングした場合、MySQL を強制終了しなくても切断することができる。
  - mysqladmin kill はスリープ状態の接続では機能しない。
  - スリープ状態の接続がある限り、mysqladmin shutdown が中断することはない。

  この問題については、当社の Windows 開発者が適切な回避策を考え出したときに修正する予定である。

- DROP DATABASE

  スレッドが使用しているデータベースを破棄することはできない。

- タスクマネージャから MySQL を強制終了する

  タスクマネージャまたは Windows 95 のシャットダウンユーティリティを使用して MySQL を強制終了することはできない。MySQL を停止するには、mysqladmin shutdown を使用する。

- ケース非依存の名前

  Windows では、ファイル名はケース非依存である。したがって MySQL のデータベース名とテーブル名も Windows ではケース非依存である。唯一の制約は、所定のステートメント全体を通じて同じケースを使用してデータベース名とテーブル名を指定する必要があることである。 See 項6.1.3.「名前におけるケース依存」。

- '\' ディレクトリ文字

  Windows 95 のパス名の構成要素は、'\' 文字で区切られる。これは、MySQL においてもエスケープ文字である。LOAD DATA INFILE または SELECT ... INTO OUTFILE を使用する場合は、以下のように '/' 文字で区切る Unix スタイルのファイル名を使用する。

```
mysql> LOAD DATA INFILE "C:/tmp/skr.txt" INTO TABLE skr;
mysql> SELECT * INTO OUTFILE 'C:/tmp/skr.txt' FROM skr;
```

あるいは、以下のように '\' 文字を二重にする。

```
mysql> LOAD DATA INFILE "C:\\tmp\\skr.txt" INTO TABLE skr;
mysql> SELECT * INTO OUTFILE 'C:\\tmp\\skr.txt' FROM skr;
```

- パイプの問題

  パイプは、Windows のコマンドラインプロンプトでは確実には機能しない。パイプに文字 ^Z / CHAR(24) が含まれている場合は、Windows は、End-Of-File (EOF) を検出したと見なして、プログラムを中止する。

  これは、主に、以下のようにバイナリログを適用しようとしたときに発生する問題である。

```
mysqlbinlog binary-log-name | mysql --user=root
```

  ログを適用するときに問題が発生し、^Z / CHAR(24) 文字が原因であると思われる場合は、以下の回避策をとる。

```
mysqlbinlog binary-log-file --result-file=/tmp/bin.sql
mysql --user=root --execute "source /tmp/bin.sql"
```

  2 番目のコマンドは、バイナリデータが含まれている可能性のある SQL ファイルを確実に読み取とる場合にも使用できる。

- Can't open named pipe エラー

  最新の MySQL クライアントプログラムを組み込んだ NT 上で MySQL バージョン 3.22 サーバを使用している場合に、以下のエラーが発生することがある。

```
error 2017: can't open named pipe to host: . pipe...
```

  このエラーは、リリースバージョンの MySQL がNT 上ではデフォルトで名前付きパイプを使用するということが原因で発生する。新しい MySQL クライアントに --host=localhost オプションを使用してこのエラーを回避するか、以下の情報を格納したオプション設定ファイル C:\my.cnf を作成する。

```
[client]
host = localhost
```

  バージョン 3.23.50 以降では、--enable-named-pipe オプションを指定して mysqld-nt または mysqld-max-nt を起動した場合にのみ、名前付きパイプが有効になる。

- Access denied for user エラー

  MySQL クライアントプログラムを実行して、同じマシン上で稼動しているサーバへの接続を試みたときに、Access denied for user: 'some-user@unknown' to database 'mysql' というエラーが表示された場合は、MySQL がホスト名を正しく解決できなかったことを意味する。

  このエラーを修正するには、以下の情報を格納した \windows\hosts ファイルを作成する。

```
127.0.0.1       localhost
```

- ALTER TABLE

  ALTER TABLE ステートメントの実行中は、他のスレッドによって使用されるのを防ぐためにテーブルがロックされる。これは、Windows 上では他のスレッドが使用中のファイルを削除することはできないという事実に対処する必要がある。将来、この問題の解決策が見つかる可能性がある。

- DROP TABLE

  Windows では、MERGE テーブルが使用しているテーブルに対して DROP TABLE を実行できない。これは、MERGE ハンドラが実行するテーブルマッピングを MySQL の上位レイヤが認識しないためである。Windows では開いているファイルを破棄できないため、対象のテーブルを破棄する前に、FLUSH TABLES ですべての MERGE テーブルをフラッシュするか、MERGE テーブルを破棄する必要がある。これについては、ビューを導入するときに修正する予定である。

- DATA DIRECTORY と INDEX DIRECTORY

  Windows では、CREATE TABLE の DATA DIRECTORY オプションと INDEX DIRECTORY オプションは無視される。これは、Windows がシンボリックリンクをサポートしていないためである。

Windows 上の MySQL の改良に協力してくださるユーザのために、未解決の問題を以下に示す。

- MySQL に使いやすい起動アイコンまたはシャットダウンアイコンを追加する。
- タスクマネージャから mysqld を強制終了できれば非常に便利である。 しばらくは、mysqladmin shutdown を使用する必要がある。
- mysql コマンドラインツールで使用するため Windows に readline を移植する。
- GUI バージョンの標準 MySQL クライアント（ mysql、mysqlshow、mysqladmin、および mysqldump ）があれば便利である。
- net.c のソケット読み取りおよび書き込みの関数が割り込み可能になれば便利である。これによって、Windows で mysqladmin kill を使用して、開いているスレッドを強制終了することができるようになる。
- Windows によって提供されるより高速でスレッドセーフなインクリメント/デクリメントメソッドを使用するマクロを追加する。

2.6.2. Linux の注意事項（ すべての Linux バージョン ）

glibc に関する以下の注意事項は、MySQL を自分でビルドする場合にのみ適用されます。x86 マシン上で Linux を実行している場合、当社が提供するバイナリを使用する方がはるかに適切です。当社のバイナリは、高負荷なサーバに適したものするために、最適なコンパイラオプションを使用して、当社が考案した最適なパッチ適用済みバージョンの glibc をリンクしています。一般のユーザにとって、多数の同時接続や 2GB の制限を超えるテーブルがある構成であっても、ほとんどの場合、当社のバイナリは最良の選択です。以下のテキストを読んで、どうすべきか判断がつかない場合は、最初に当社のバイナリを試して自社の要件を満たすかどうかを確認し、十分でないことがわかった場合にのみ独自のビルドを検討し

てください。その場合は、不足な点を当社にお知らせください。次回のバイナリのビルドの際に参考にさせていただきます。

MySQL は、Linux 上で LinuxThreads を使用します。glibc2 が組み込まれていない旧バージョンの Linux を使用している場合は、MySQL をコンパイルする前に LinuxThreads をインストールする必要があります。LinuxThreads は、http://www.mysql.com/downloads/os-linux.html で入手できます。

注意: SMP システム上の Linux 2.2.14 と MySQL で未知の問題が発生したことがあります。SMP システムを使用している場合は、できるだけ速やかに Linux 2.4 にアップグレードすることをお勧めします。これによって、システムの処理速度と安定性が向上します。

注意: バージョン 2.1.1 以前のバージョンの glibc は、INSERT DELAYED ステートメントを発行したときに使用される pthread_mutex_timedwait 処理に重大なバグがあります。glibc をアップグレードするまでは、INSERT DELAYED を使用しないことをお勧めします。

1,000 を超える同時接続を予定している場合は、LinuxThreads にいくつかの変更を加えて再コンパイルし、新しい libpthread.a を MySQL に再リンクする必要があります。 sysdeps/unix/sysv/linux/bits/local_lim.h の PTHREAD_THREADS_MAX を 4096 に増やし、linuxthreads/internals.h の STACK_SIZE を 256 KB に減らしてください。このパスは glibc のルートからの相対パスです。 注意: STACK_SIZE がデフォルト値の 2MB である場合、MySQL は約 600 ～ 1,000 接続で不安定になります。

MySQL が十分な数のファイルまたは接続を開けない場合は、その数のファイルを処理するように Linux を設定していない可能性があります。

Linux 2.2 以降では、以下を実行して、割り当てられているファイルハンドルの数をチェックできます。

```
cat /proc/sys/fs/file-max
cat /proc/sys/fs/dquot-max
cat /proc/sys/fs/super-max
```

16 MB を超えるメモリがある場合は、init スクリプト（SuSE Linux の /etc/init.d/boot.local など）に以下のような記述を追加してください。

```
echo 65536 > /proc/sys/fs/file-max
echo 8192 > /proc/sys/fs/dquot-max
echo 1024 > /proc/sys/fs/super-max
```

root アカウントでコマンドラインから上記のコマンドを実行することもできますが、次回にコンピュータがリブートしたときにそれらの設定は失われます。

あるいは、sysctl ツールを使用して、ブート時に以下のパラメータを設定することもできます。このツールは、多くの Linux ディストリビューションで使用されています（SuSE Linux 8.0 以降、SuSE でもこのツールが追加されています）。/etc/sysctl.conf という名前のファイルに以下の値を入力してください。

```
# Increase some values for MySQL
fs.file-max = 65536
fs.dquot-max = 8192
fs.super-max = 1024
```

/etc/my.cnf に以下を追加する必要もあります。

```
[mysqld_safe]
```

```
open-files-limit=8192
```

これによって、MySQL は最大で 8,192 の接続およびファイルを作成できるようになります。

LinuxThreads の STACK_SIZE 定数は、アドレス空間でのスレッドスタックのスペーシングを制御します。アドレス空間は、個別のスレッドのスタックに十分なスペースを提供できる程度に大きくなければなりませんが、いくつかのスレッドのスタックがグローバルな mysqld データにぶつからない程度に小さくなければなりません。mmap() の Linux 実装は、すでに使用されているアドレスをマップアウトするように指示されると、エラーを返さずに、マップ済みの領域を正常にマップ解除して、ページ全体のデータを消去することが実験でわかっています。mysqld やその他のスレッドアプリケーションの安全性は、スレッドを作成するコードの "紳士的な" 動作にかかっています。ユーザは、任意の時点で稼動しているスレッドの数が、スレッドスタックがグローバルヒープから離れていられる程度に小さくなるようにするための措置をとる必要があります。mysqld では、max_connections 変数に適切な値を設定してこの "紳士的な" 動作を強制する必要があります。

自分で MySQL をビルドする場合、LinuxThreads にパッチを適用する作業をしたくなければ、max_connections を 500 以下の値に設定してください、大きなキーバッファや大きなヒープテーブルなど、mysqld が大量のメモリを割り当てなければならないものがある場合や、2G のパッチが適用された 2.2 カーネルを実行している場合は、さらに小さな値を設定します。当社のバイナリまたは RPM バージョン 3.23.25 以降を使用している場合は、大量のデータを持つキーバッファまたはヒープテーブルがなければ、max_connections を 1500 に設定しても大丈夫です。LinuxThreads の STACK_SIZE の値を小さくすればするほど、スレッドを支障なく作成できるようになります。128K から 256K までの範囲の値を推奨します。

多数の同時接続を使用する場合、fork 爆弾攻撃を避けるために、子プロセスの fork や複製のプロセスにペナルティを科すという、2.2 カーネルの "機能" に悩まされることがあります。これにより、同時実行クライアントの数を増やすと、MySQL がうまくスケールしなくなります。シングル CPU システムでは、スレッド生成が非常に遅くなることでこれが明らかになりました。これは、MySQL に接続するのに長い時間がかかり (1 分程度)、接続を切断するのにも同じだけ時間がかかることを意味します。マルチ CPU システムでは、クライアント数の増加に伴って、クエリ速度が徐々に落ちていくのが観測されました。解決法を見つける過程で、ユーザの 1 人からカーネルパッチを受け取りました。このユーザのサイトではこのパッチが大きな効果を上げているということでした。このパッチは、http://www.mysql.com/Downloads/Patches/linux-fork.patch で入手できます。当社は、開発システムと実稼動システムの両方で、このパッチに非常に広範囲に及ぶテストを実行しました。このパッチは、何の問題も引き起こすことなく MySQL のパフォーマンスを大幅に向上させました。2.2 カーネル上で高負荷のサーバを実行しているユーザに、このパッチをお勧めします。2.4 カーネルではこの問題は修正されています。したがって、システムの現在のパフォーマンスに満足していない場合は、2.2 カーネルにパッチを適用するよりも、2.4 カーネルにアップグレードする方が簡単です。2.4 カーネルは、この公正バグが修正されたほかに、優れた SMP ブーストも提供します。

2 CPU マシンでバージョン 2.4 上の MySQL をテストした結果、MySQL のスケーラビリティが非常に向上することがわかりました。1,000 クライアントに達するまでクエリスループットが低下することは実質的に皆無で、MySQL のスケーリングファクタ (1クライアントでのスループットに対する最大スループットの比率として算出される) は 180% でした。 4 CPU システムでも同様の結果が観察されました。クライアント数が 1,000 に達するまで実質的に速度低下はなく、スケーリングファクタは 300% でした。したがって、高負荷 SMP サーバの場合は、現時点で 2.4 カーネルを使用することを強くお勧めします。最大パフォーマンスを達成するには、2.4 カーネル上で最も高い優先順位を設定して mysqld プロセスを実行することが不可欠であることがわかりました。 これを行うには、renice -20 $$ コマンドを mysqld_safe に追加します。4 CPU マシン上でのテストでは、優先順位を高くすると、400 クライアントでのスループットが 60% 増加しました。

また、現在、当社は、4-way システムおよび 8-wayシステムで 2.4 カーネル上の MySQL のパフォーマンスがどれだけ向上するかについての情報を収集中です。このようなシステムにアクセスでき、何らかのベンチマークを行った場合は、結果を http://www.mysql.com/company/contact/ にお送りください。将来、このマニュアルにそれらの結果を記載します。

特に SMP システム上で顕著な、MySQL のパフォーマンスを大きく損なうもう 1 つの問題があります。glibc-2.1 の LinuxThreads の相互排他ロック（mutex）の実装は、短時間だけ相互排他ロックを保持する多数のスレッドを持つプログラムに悪影響を及ぼします。皮肉なことに、SMP システムでは、MySQL を未修正の LinuxThreads にリンクしている場合、多くの場合、マシンからプロセッサを取り外すと MySQL のパフォーマンスが向上します。当社は、この動作を修正するための、glibc 2.1.3 用のパッチを用意しました（http://www.mysql.com/Downloads/Linux/linuxthreads-2.1-patch）。

MySQL バージョン 3.23.36 は、glibc-2.2.2 のアダプティブロックを使用します。これは、glibc-2.1.3 のパッチ適用済みの相互排他ロックよりもさらに適切です。ただし、状況によって glibc-2.2.2 の現在の相互排他ロックのコードはオーバースピンし、これによって MySQL のパフォーマンスは損なわれるということに注意してください。mysqld プロセスの優先順位を最も高いものに変更することで、この状況が発生する可能性を下げることができます。オーバースピン動作は、パッチで修正することもできました。このパッチは、http://www.mysql.com/Downloads/Linux/linuxthreads-2.2.2.patch で入手できます。 このパッチは、オーバースピン、スレッドの最大数、およびスタックスペーシングの修正を一体化したものです。patch -p0 </tmp/linuxthreads-2.2.2.patch の linuxthreads ディレクトリでパッチを適用する必要があります。 glibc-2.2 の将来のリリースに何らかの形でこのパッチが組み込まれることが望まれます。いずれにしても、glibc-2.2.2 にリンクしている場合は、STACK_SIZE と PTHREAD_THREADS_MAX を修正する必要があります。将来、デフォルトが高負荷の MySQL 設定として許容できる値に修正されて、その結果ユーザ自身のビルドが ./configure; make; make install まで減ることが望まれます。

上記のパッチを使用して libpthread.a の静的ライブラリを別にビルドし、MySQL に静的にリンクするためにのみ使用することをお勧めします。これらのパッチが MySQL にとって安全であり、MySQL のパフォーマンスを大幅に向上させることはわかっていますが、その他のアプリケーションについてはわかっていません。その他のアプリケーションをこのライブラリのパッチ適用済みバージョンにリンクする場合や、パッチ適用済みの共有バージョンをビルドしてシステムにインストールする場合は、LinuxThreads に依存するその他のアプリケーションに関してはユーザ自身の責任で行ってください。

MySQL のインストール時に未知の問題が発生した場合や、一般的なユーティリティがハングした場合は、それがライブラリまたはコンパイラに関連するものである可能性が高いです。その場合は、当社のバイナリを使用することでそれらの問題は解決されます。

バイナリディストリビューションの既知の問題の 1 つは libc を使用する旧バージョンの Linux システム（Red Hat 4.x や Slackware など）に関連するもので、ホスト名解決で重大ではない問題がいくつか発生します。 See 項2.6.2.1.「Linux の注意事項（バイナリディストリビューション）」。

LinuxThreads を使用している場合は、最小で 3 つのプロセスが稼動しているのが確認されます。これらは、実際はスレッドで、1 つは LinuxThreads マネージャ用のスレッド、1 つは接続を処理するスレッド、1 つはアラームとシグナルを処理するスレッドです。

注意: Linux カーネルと LinuxThreads ライブラリは、デフォルトでは 1,024 のスレッドしか持つことはできません。したがって、パッチ未適用のシステムでは、MySQL への接続は最大で 1,021 しか使用できません。この制限を回避する方法については http://www.volano.com/linuxnotes.html ページを参照してください。

ps で dead 状態の mysqld デーモンプロセスが表示された場合は、通常は、MySQL にバグがあるか、壊れたテーブルがあることを意味します。 See 項A.4.1.「MySQL が何度もクラッシュする場合に行うこと」。

SIGSEGV シグナルで mysqld が停止した場合に Linux 上でコアダンプを取得するには、--core-file オプションを指定して mysqld を起動します。注意: ulimit -c 1000000 を mysqld_safe に追加するか、--core-file-size=1000000 を指定して mysqld_safe を起動して、コアファイルのサイズを増やします。 See 項4.8.2.「mysqld_safe（mysqld のラッパ）」。

独自の MySQL クライアントをリンクしているときに、以下のエラーが発生した場合

```
ld.so.1: ./my: fatal: libmysqlclient.so.4:
```

```
open failed: No such file or directory
```

それらを実行するときに、以下のいずれかの方法で問題を回避できます。

- -Lpath の代わりに、以下のフラグを設定してクライアントをリンクする。 -Wl,r/path-libmysqlclient.so。
- libmysqclient.so を /usr/lib にコピーする。
- クライアントを実行する前に、libmysqlclient.so が配置されているディレクトリのパス名を LD_RUN_PATH 環境変数に追加する。

富士通コンパイラ (fcc / FCC) を使用している場合は、MySQL のコンパイルでいくつかの問題が発生します。これは、Linux ヘッダファイルが非常に gcc 指向であるためです。

fcc/FCC を使用する場合は、以下の configure 行が有効です。

```
CC=fcc CFLAGS="-O -K fast -K lib -K omitfp -Kpreex -D_GNU_SOURCE \
-DCONST=const -DNO_STRTOLL_PROTO" CXX=FCC CXXFLAGS="-O -K fast -K lib \
-K omitfp -K preex --no_exceptions --no_rtti -D_GNU_SOURCE -DCONST=const \
-Dalloca=__builtin_alloca -DNO_STRTOLL_PROTO \
'-D_EXTERN_INLINE=static __inline'" ./configure --prefix=/usr/local/mysql \
--enable-assembler --with-mysqld-ldflags=-all-static --disable-shared \
--with-low-memory
```

2.6.2.1. Linux の注意事項（バイナリディストリビューション）

MySQL は、Linux バージョン 2.0 以降を必要とします。

警告: 一部の MySQL ユーザからLinux カーネル 2.2.14 を使用する MySQL で重大な安定性の問題が発生したことが報告されています。このカーネルを使用している場合は、カーネル 2.2.19 以降またはカーネル 2.4 にアップグレードしてください。マルチ CPU マシンを使用している場合は、カーネル 2.4 によって処理速度が飛躍的に向上するので、使用を検討してください。

バイナリリリースは、-static でリンクされます。したがって、通常は使用するシステムライブラリのバージョンを気にする必要はありません。また、LinuxThreads をインストールする必要もありません。-static でリンクされたプログラムは、動的にリンクされたプログラムよりも少し大きいですが、速度も少し（3 ー 5%）上がります。ただし、静的にリンクされたプログラムにはユーザ定義関数（UDF）を使用できないことが 1 つの問題です。UDFを作成または使用する場合（C または C++ のプログラマの場合のみ）は、動的リンクを使用して MySQL を自分でコンパイルする必要があります。

glibc2 システムではなく、libc ベースのシステムを使用している場合は、バイナリリリースでホスト名解決と getpwnam() に関連する問題が発生することが多いです（これは、glibc が、-static でコンパイルされている場合でも、外部ライブラリに依存してホスト名と getpwent() を解決することが原因です）。その場合、mysql_install_db を実行したときに以下のエラーメッセージが表示されます。

```
Sorry, the host 'xxxx' could not be looked up
```

または、--user オプションを指定して mysqld を実行したときに、以下のエラーが表示されます。

```
getpwnam: No such file or directory
```

この問題は、以下のいずれかの方法で解決できます。

- MySQL ソースディストリビューション（RPM または tar.gz ディストリビューション）を入手して、それをインストールする。
- mysql_install_db --force を実行する。この場合、mysql_install_db の resolveip テストは実行されません。この方法の弱点は、権限テーブルでホスト名を使用できないため、代わりに IP 番号を使用しなくてはならないことです（localhost の場合は除く）。--force をサポートしていない旧バージョンの MySQL リリースを使用している場合、エディタを使用して mysql_install の resolveip テストを削除する必要があります。
- --user を使用する代わりに su を指定して mysqld を起動する。

MySQL の Linux-Intel バイナリと RPM リリースは、最高の速度を実現するようにコンフィギャされています。当社は、常に、入手可能な最速の安定したコンパイラを使用するよう努めています。

MySQL の Perl サポートは、Perl 5.004_03 以降のバージョンを必要とします。

一部の Linux 2.2 バージョンでは、TCP/IP を介して mysqld サーバに多数の接続を行うと、Resource temporarily unavailable というエラーが発生することがあります。

問題は、Linux では、TCP/IP ソケットを閉じてから、そのソケットが実際にシステムによって解放されるまでの間に遅延があることです。限られた数の TCP/IP スロット用のスペースしかないので、TCP/IP を介して test-connect ベンチマークを実行している場合など、短時間に多すぎる数の新しい TCP/IP 接続を実行すると、上記のエラーが発生します。

さまざまなLinux メーリングリストにこの問題を何回かメールで送りましたが、適切な解決法はいまだに得られていません。

この問題の '修正方法' として唯一知られているのは、データベースサーバとクライアントを同じマシン上で実行している場合に、クライアントで永続的な接続を使用するかソケットファイルを使用することです、将来、Linux 2.4 カーネルでこの問題が修正されることが望まれます。

2.6.2.2. Linux x86 の注意事項

MySQL は、libc バージョン 5.4.12 以降を必要とします。MySQL が libc 5.4.46 と連携して正常に動作することは知られています。glibc バージョン 2.0.6 以降も正常に機能するはずです。Red Hat 製の glibc RPM にはいくつかの問題がありました。したがって、問題が発生した場合は、更新がないかどうかチェックしてください。 glibc 2.0.7-19 および 2.0.7-29 の RPM は、正常に機能することが知られています。

Red Hat 8.0 以降の glibc 2.2.x ライブラリを使用している場合は、--thread-stack=192K オプションを指定して mysqld を起動してください（MySQL 4 より前のバージョンの場合は -O thread_stack=192K を使用してください）。このオプションを使用しないと、mysqld は gethostbyaddr() で停止します。これは、新しい glibc ライブラリがこのコールのために 128KB を超えるスタックサイズを必要とするためです。MySQL 4.0.10 以降では、このスタックサイズがデフォルトになりました。

gcc 3.0 を使用して MySQL をコンパイルする場合は、コンパイルの前に libstdc++v3 ライブラリをインストールする必要があります。このライブラリをインストールしないと、リンク時に __cxa_pure_virtual シンボルが見つからないというエラーが出力されます。

一部の旧バージョンの Linux ディストリビューションでは、configure 時に以下のようなエラーが出力されることがありま

す。

```
Syntax error in sched.h. Change _P to __P in the /usr/include/sched.h file.
See the Installation chapter in the Reference Manual.
```

このエラーメッセージの指示に従って、1 つのアンダースコアしかない _P マクロに、アンダースコアをもう 1 つ追加してから、再実行します。

コンパイル中にいくつかの警告が表示されることがあります。以下に示した警告は無視してかまいません。

```
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'
```

mysql.server は、MySQL インストールディレクトリの下の share/mysql ディレクトリか、MySQL ソースツリーの support-files ディレクトリにあります。

mysqld が起動時に必ずコアダンプする場合は、旧バージョンの /lib/libc.a を使用していることが問題である可能性があります。このファイルの名前を変更して、sql/mysqld を削除し、新たに make install を実行して、もう一度起動してみてください。一部の Slackware でこの問題が発生することが報告されています。

mysqld をリンクしているときに以下のエラーが発生した場合は、libg++.a が正しくインストールされていないことを意味します。

```
/usr/lib/libc.a(putc.o): In function `_IO_putc':
putc.o(.text+0x0): multiple definition of `_IO_putc'
```

以下のように configure を実行して、libg++.a を使用しないようにすることができます。

```
shell> CXX=gcc ./configure
```

2.6.2.3. Linux SPARC の注意事項

一部の実装で、readdir_r() が壊れています。SHOW DATABASES が常に空のセットを返すという現象が発生します。この問題は、設定した後、コンパイルする前に、config.h から HAVE_READDIR_R を削除することで修正できます。

2.6.2.4. Linux Alpha の注意事項

MySQL バージョン 3.23.12 は、Linux-Alpha 上でテストされた最初の MySQL バージョンです。Linux-Alpha 上で MySQL を使用する予定の場合は、必ずこのバージョン以降の MySQL を使用してください。

Alpha 上のMySQL は、当社のベンチマークとテストスィートを使用してテストされましたが、適正に動作すると思われます。

現在、当社は、Alpha EV6 プロセッサを搭載した Compaq DS20 マシン上で、SuSE Linux 7.0 for AXP、カーネル 2.4.4-SMP、Compaq の C コンパイラ（V6.2-505）、および Compaq の C++ コンパイラ（V6.3-006）で MySQL バイナリパッケージをビルド中です。

上記のコンパイラは、http://www.support.compaq.com/alpha-tools/ にあります。gcc の代わりにこれらのコンパイラを使用することで、MySQL のパフォーマンスが約 9 〜 14% 向上します。

注意: MySQL バージョン 3.23.52 および 4.0.2 までは、バイナリは最新の CPU だけを対象に最適化されていました (-fast コンパイルオプションを使用して)。これは、Alpha EV6 プロセッサを使用している場合にのみ当社のバイナリを使用できることを意味します。

その後のリリースからは、バイナリが確実にすべての Alpha プロセッサ上で動作するようにする、-arch generic フラグがコンパイルオプションに加えられました。また、当社では、ライブラリの問題を回避するために静的にコンパイルします。

```
CC=ccc CFLAGS="-fast -arch generic" CXX=cxx \
CXXFLAGS="-fast -arch generic -noexceptions -nortti" \
./configure --prefix=/usr/local/mysql --disable-shared \
--with-extra-charsets=complex --enable-thread-safe-client \
--with-mysqld-ldflags=-non_shared --with-client-ldflags=-non_shared
```

egcs を使用する場合は、以下の configure 行を使用します。

```
CFLAGS="-O3 -fomit-frame-pointer" CXX=gcc \
CXXFLAGS="-O3 -fomit-frame-pointer -felide-constructors \
-fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql \
--disable-shared
```

Linux-Alpha 上で MySQL を実行した場合の既知の問題は以下のとおりです。

- gdb 4.18 では MySQL などのスレッドアプリケーションのデバッグが正常に行われない。代わりに、gdb 5.1 をダウンロードして使用する。
- gcc を使用している場合に、mysqld を静的にリンクしようとすると、結果のイメージが起動時にコアダンプする。したがって、gcc と一緒に --with-mysqld-ldflags=-all-static を使用しないようにする。

2.6.2.5. Linux PowerPC の注意事項

MySQL は、最新の glibc パッケージを備えた MkLinux 上で正常に動作します (glibc 2.0.7 でテスト済み)。

2.6.2.6. Linux MIPS の注意事項

Qube2 (Linux Mips) 上で MySQL を機能させるには、最新の glibc ライブラリが必要です (glibc-2.0.7-29C2 は正常に機能することが知られています)。また、egcs C++ コンパイラ (egcs-1.0.2-9、または gcc 2.95.2 以降) を使用する必要もあります。

2.6.2.7. Linux IA-64 の注意事項

Linux IA-64 上でMySQL をコンパイルさせるには、以下の compile 行を使用します (gcc-2.96 を使用)。

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc \
CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors \
-fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql \
"--with-comment=Official MySQL binary" --with-extra-charsets=complex
```

IA-64 では、MySQL クライアントバイナリは共有ライブラリを使用します。これは、/usr/local/mysql 以外の場所にバイナリディストリビューションをインストールした場合は、libmysqlclient.so をインストールしたディレクトリのパスを、/etc/ld.so.conf ファイルまたは LD_LIBRARY_PATH 環境変数の値に追加する必要があるということを意味します。

See 項A.3.1.「MySQL クライアントライブラリにリンク時の問題」。

2.6.3. Solaris の注意事項

Solaris では、MySQL ディストリビューションをアンパックする前でも問題が発生することがあります。Solaris tar は長いファイル名を処理できないので、MySQL をアンパックするときに以下のようなエラーが表示されることがあります。

```
x mysql-3.22.12-beta/bench/Results/ATIS-mysql_odbc-NT_4.0-cmp-db2,\
informix,ms-sql,mysql,oracle,solid,sybase, 0 bytes, 0 tape blocks
tar: directory checksum error
```

この場合、GNU tar (gtar) を使用して、ディストリビューションをアンパックする必要がありますSolaris 用のプリコンパイル済みコピーは、http://www.mysql.com/downloads/os-solaris.html にあります。

Sun のネイティブスレッドは、Solaris 2.5 以降でのみ機能します。バージョン 2.4 以前の Solaris の場合は、MySQL は自動的に MIT-pthreads を使用します。 See 項2.3.6.「MIT-pthreads に関する注意事項」。

configure 時に以下のエラーが出力される場合

```
checking for restartable system calls... configure: error can not run test
programs while cross compiling
```

このエラーは、コンパイラに問題があることを意味します。 この場合、コンパイラを新しいバージョンにアップグレードする必要があります。config.cache ファイルに以下の行を挿入することで、この問題を解決できることもあります。

```
ac_cv_sys_restartable_syscalls=${ac_cv_sys_restartable_syscalls='no'}
```

SPARC 上で Solaris を使用している場合、推奨されるコンパイラは gcc 2.95.2 または 3.2 です。これらのコンパイラは、http://gcc.gnu.org/ にあります。 注意: egcs 1.1.1 および gcc 2.8.1 は、SPARC 上では適正に機能しません。

gcc 2.95.2 を使用している場合は、以下の configure 行を推奨します。

```
CC=gcc CFLAGS="-O3" \
CXX=gcc CXXFLAGS="-O3 -felide-constructors -fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql --with-low-memory --enable-assembler
```

UltraSPARC を使用している場合は、"-mcpu=v8 -Wa,-xarch=v8plusa" を CFLAGS および CXXFLAGS に追加することでパフォーマンスを 4% 向上できます。

Sun の Forte 5.0 以降のコンパイラを使用している場合は、configure を以下のように実行できます。

```
CC=cc CFLAGS="-Xa -fast -native -xstrconst -mt" \
CXX=CC CXXFLAGS="-noex -mt" \
./configure --prefix=/usr/local/mysql --enable-assembler
```

以下のコンパイルフラグを指定して Sun の Forte コンパイラを使用して、64 ビットのバイナリを作成できます。

```
CC=cc CFLAGS="-Xa -fast -native -xstrconst -mt -xarch=v9" \
```

```
CXX=CC CXXFLAGS="-noex -mt -xarch=v9" ASFLAGS="-xarch=v9" \
./configure --prefix=/usr/local/mysql --enable-assembler
```

gcc を使用して 64 ビットの Solaris バイナリを作成するには、CFLAGS および CXXFLAGS に -m64 を追加します。注意: これは、MySQL 4.0 以降のみで機能します。MySQL 3.23 にはこれをサポートするために必要な変更が含まれていません。

MySQL ベンチマークでは、UltraSPARC 上で 32 ビットモードで Forte 5.0 を使用した場合、-mcpu フラグを指定して gcc 3.2 を使用した場合と比較して速度が 4% 向上しました。

64 ビットバイナリを作成した場合、32 ビットバイナリよりも速度が 4% 低下しますが、その代わりに mysqld はより多くのスレッドとメモリを処理できます。

fdatasync または sched_yield で問題が発生した場合は、configure 行に LIBS=-lrt を追加することでその問題を修正できます。

以下の段落は、WorkShop 5.3 より前のバージョンのコンパイラにのみ関連するものです。

configure スクリプトを編集して以下の行を変更する必要がある場合もあります。

```
#if !defined(__STDC__) || __STDC__ != 1
```

これを次のように変更します。

```
#if !defined(__STDC__)
```

-Xc オプションを指定して __STDC__ を有効にすると、Sun のコンパイラは Solaris の pthread.h ヘッダファイルでコンパイルできません。これは Sun のバグです (コンパイラまたはインクルードファイルが壊れています) 。

mysqld を実行したときに、以下のようなエラーメッセージが出力された場合は、マルチスレッドオプション (-mt) を有効にしていない Sun のコンパイラを使用して MySQL をコンパイルしようとしています。

```
libc internal error: _rmutex_unlock: rmutex not held
```

CFLAGS および CXXFLAGS に -mt を追加して、再実行してください。

SFW バージョンの gcc (Solaris 8 に付属している) を使用している場合は、configure を実行する前に、LD_LIBRARY_PATH 環境変数に /opt/sfw/lib を追加する必要があります。

sunfreeware.com から入手できる gcc を使用している場合は、多くの問題が発生することがあります。問題を回避するためには、gcc および GNU binutils を実行するマシン上で、それらを再コンパイルする必要があります。

gcc を使用して MySQL をコンパイルしているときに以下のエラーが出力された場合は、その gcc が使用している Solaris のバージョン用にコンフィギャされていないことを意味します。

```
shell> gcc -O3 -g -O2 -DDBUG_OFF  -o thr_alarm ...
./thr_alarm.c: In function `signal_hand':
./thr_alarm.c:556: too many arguments to function `sigwait'
```

この場合に行うべきことは、最新バージョンの gcc を入手し、現在の gcc コンパイラを使用してそれをコンパイルすることです。少なくとも Solaris 2.5 の場合、gcc のほとんどすべてのバイナリバージョンに古くて利用に適さないインクルー

ドファイルが組み込まれており、それらによって、スレッドを使用するすべてのプログラム (そして、おそらくその他のプログラムも) が破壊されます。

Solaris は、静的バージョンのシステムライブラリ (libpthreads および libdl) は一切提供しません。したがって、--static を指定して MySQL をコンパイルすることはできません。このオプションを指定してコンパイルすると、以下のエラーが表示されます。

```
ld: fatal: library -ldl: not found

または

undefined reference to `dlopen'

または

cannot find -lrt
```

多すぎる数のプロセスが非常に迅速に mysqld に接続を試みると、MySQL ログに以下のエラーが出力されます。

```
Error in accept: Protocol error
```

この問題を回避するために、--set-variable back_log=50 オプションを指定してサーバを起動してみてください。 注意: --set-variable=name=value 構文および -O name=value 構文は、MySQL 4.0 以降廃止されました。代わりに、--back_log=50 を使用してください。 See 項4.1.1.「mysqld コマンドラインオプション」。

独自の MySQL クライアントをリンクしている場合、そのクライアントを実行したときに、以下のエラーが発生することがあります。

```
ld.so.1: ./my: fatal: libmysqlclient.so.#:
open failed: No such file or directory
```

以下のいずれかの方法でこの問題を回避できます。

- -Lpath の代わりに、以下のフラグを設定してクライアントをリンクする。 -Wl,r/full-path-to-libmysqlclient.so。
- libmysqclient.so を /usr/lib にコピーする。
- クライアントを実行する前に、libmysqlclient.so が配置されているディレクトリのパス名を LD_RUN_PATH 環境変数に追加する。

configure が -lz でリンクする際に問題が発生し、zlib をインストールしていない場合、以下の 2 つの選択肢があります。

- 圧縮された通信プロトコルを使用できるようにする場合は、ftp.gnu.org から zlib を入手してインストールする必要がある。
- --with-named-z-libs=no を指定してコンフィギャする。

gcc を使用していて、ユーザ定義関数 (UDF) を MySQL にロードする際に問題が発生した場合は、その UDF をリンクする行に -lgcc を追加してみてください。

MySQL を自動的に起動させる場合は、/etc/init.d に support-files/mysql.server をコピーして、/etc/rc3.d/S99mysql.server という名前でそのディレクトリへのシンボリックリンクを作成します。

Solaris では setuid() アプリケーションのコアファイルがサポートされていないので、--user オプションを指定している場合、mysqld からコアファイルを取得することはできません。

2.6.3.1. Solaris 2.7 および 2.8 の注意事項

Solaris 2.7 および 2.8 上では、Solaris 2.6 バイナリを通常どおりに使用することができます。Solaris 2.6 のほとんどの問題は、Solaris 2.7 および 2.8 の場合にも当てはまります。

注意: MySQL バージョン 3.23.4 以降は、新しいバージョンの Solaris を自動検出できるため、以下の問題の回避を可能にします。

Solaris 2.7 および 2.8 のインクルードファイルにはいくつかのバグがあります。gcc を使用している場合に、以下のエラーが発生することがあります。

```
/usr/include/widec.h:42: warning: `getwc' redefined
/usr/include/wchar.h:326: warning: this is the location of the previous
definition
```

この場合、以下を実行してこの問題を解決できます。

/usr/include/widec.h を .../lib/gcc-lib/os/gcc-version/include に コピーして、以下の行 41 を変更します。

```
#if     !defined(lint) && !defined(__lint)

これを次のように変更します。

#if     !defined(lint) && !defined(__lint) && !defined(getwc)
```

または、直接 /usr/include/widec.h を編集します。いずれの場合も、修正した後、config.cache を削除して、configure を再実行してください。

make を実行したときに以下のようなエラーが発生した場合は、configure が curses.h ファイルを検出できなかったことが原因です (/usr/include/widec.h にエラーがある可能性が高いです) 。

```
In file included from mysql.cc:50:
/usr/include/term.h:1060: syntax error before `,'
/usr/include/term.h:1081: syntax error before `;'
```

この問題を解決するには、以下のいずれかを実行します。

- CFLAGS=-DHAVE_CURSES_H CXXFLAGS=-DHAVE_CURSES_H ./configure を指定してコンフィギャする。
- 前述のように /usr/include/widec.h を編集し、configure を再実行する。
- config.h ファイルから #define HAVE_TERM 行を削除し、make を再実行する。

クライアントプログラムをリンクしているときに、リンカが -lz を検出できないという問題が発生した場合は、libz.so ファイルが /usr/local/lib にインストールされていることが問題である可能性が高いです。この問題は、以下のいずれかの方

法で解決できます。

- /usr/local/lib を LD_LIBRARY_PATH にコピーする。
- /lib にある libz.so へのリンクを追加する。
- Solaris 8 を使用している場合は、Solaris 8 の CD ディストリビューションにあるオプションの zlib をインストールできる。
- --with-named-z-libs=no オプションを指定して、MySQL をコンフィギャする。

2.6.3.2. Solaris x86 の注意事項

x86 上の Solaris 8 では、strip を使用してデバッグシンボルを削除すると、mysqld がコアダンプします。

Solaris x86 上で gcc または egcs を使用していて、ロード中のコアダンプで問題が発生した場合は、以下の configure を使用してください。

```
CC=gcc CFLAGS="-O3 -fomit-frame-pointer -DHAVE_CURSES_H" \
CXX=gcc \
CXXFLAGS="-O3 -fomit-frame-pointer -felide-constructors -fno-exceptions \
-fno-rtti -DHAVE_CURSES_H" \
./configure --prefix=/usr/local/mysql
```

これによって、libstdc++ ライブラリと C++ 例外に関する問題が回避されます。

このコマンドを使用しても問題が解決しない場合は、デバッグバージョンをコンパイルして、トレースファイルを使用するか、gdb でそのデバッグバージョンを実行します。 See 項E.1.3.「mysqld のデバッグ（gdb 使用）」。

2.6.4. BSD の注意事項

ここでは、各種の BSD と、それぞれの個別のバージョンについて説明します。

2.6.4.1. FreeBSD の注意事項

スレッドパッケージがより統合されているので FreeBSD 4.x 以降を推奨します。

最も簡単な、したがって最も望ましいインストール方法は、mysql-server と mysql-client の ports（http://www.freebsd.org/ で入手できる）を使用することです。

これらの ports を使用すると以下のことを実現できます。

- 使用している FreeBSD に最適な MySQL 。
- 自動設定およびビルド。
- /usr/local/etc/rc.d にインストールされた起動スクリプト。
- pkg_info -L を使用して、インストールされているファイルを確認することが可能。そのマシンに MySQL が不要になったら、pkg_delete を使用してそれらのファイルをすべて削除することが可能。

FreeBSD 2.x では MIT-pthreads を、バージョン 3 以降ではネイティブスレッドを、それぞれ使用することをお勧めします。一部の後期 2.2 x バージョンでは、ネイティブスレッドを使用して実行できますが、mysqld をシャットダウンする際に問題が発生することがあります。

FreeBSD 上の一部の関数呼び出しは、まだ完全にはスレッドセーフではありません。最も顕著なのは、MySQL がホスト名を IP アドレスに変換するために使用する gethostbyname() 関数です。特定の状況下で、mysqld プロセスは、突然、CPU 負荷 100% 状態を引き起こして、応答不能になります。この状態が発生したら、--skip-name-resolve オプションを使用して MySQL を起動してみてください。

あるいは、FreeBSD 4.x 上で MySQL を LinuxThreads ライブラリにリンクします。これによって、ネイティブの FreeBSD スレッド実装が持つ 2、3 の問題が回避されます。LinuxThreads とネイティブスレッドの非常に優れた比較については、"FreeBSD or Linux for your MySQL Server?" という Jeremy Zawodny の記事 (http://jeremy.zawodny.com/blog/archives/000697.html) を参照してください。

FreeBSD で LinuxThreads を使用した場合の既知の問題は以下のとおりです。

- wait_timeout は機能していない (FreeBSD/LinuxThreads のシグナル処理の問題である可能性がある) 。この問題は、FreeBSD 5.0 で修正される予定となっている。 永続的な接続がクローズされずに長時間ハングするという現象が発生する。

MySQL Makefile を使用するには、GNU make （ gmake) が機能している必要があります。MySQL をコンパイルする場合は、最初に GNU make をインストールする必要があります。

ネームリゾルバの設定に誤りがないようにしてください。設定に誤りがあると、mysqld に接続するときにリゾルバが遅延または失敗することがあります。

/etc/hosts ファイルの localhost エントリが正確であることを確認してください。正確でないと、データベースに接続する際に問題が発生します。/etc/hosts ファイルは、以下の行で始まっていなければなりません。

```
127.0.0.1       localhost localhost.your.domain
```

gcc (2.95.2 以降) を使用して FreeBSD 上で MySQL のコンパイルとインストールを行う場合は、以下のように行うことをお勧めします。

```
CC=gcc CFLAGS="-O2 -fno-strength-reduce" \
CXX=gcc CXXFLAGS="-O2 -fno-rtti -fno-exceptions -felide-constructors \
-fno-strength-reduce" \
./configure --prefix=/usr/local/mysql --enable-assembler
gmake
gmake install
./scripts/mysql_install_db
cd /usr/local/mysql
./bin/mysqld_safe &
```

configure が MIT-pthreads を使用することがわかった場合は、MIT-pthreads の注意事項を読んでください。 See 項2.3.6.「MIT-pthreads に関する注意事項」。

make install が /usr/include/pthreads を検出できないというエラーを出力した場合は、configure が MIT-pthreads が必要なことを検出しなかったことを意味します。このエラーは、以下のコマンドを実行して修正します。

```
shell> rm config.cache
shell> ./configure --with-mit-threads
```

FreeBSD は、デフォルトのファイルハンドルの上限が非常に低く設定されていることも知られています。See 項A.2.17.「File Not Found エラー」。 mysqld_safe の ulimit -n セクションのコメントを解除するか、/etc/login.conf で mysqld ユーザに対する上限を増やして cap_mkdb /etc/login.conf で再ビルドします。 また、デフォルトの値を使用しない場合は、必ずこのユーザに適切なクラスを設定してください（ chpass mysqld-user-name が使用できる ）。 See 項4.8.2.「mysqld_safe（ mysqld のラッパ ）」。

メモリに余裕がある場合は、MySQL が 512 MB を超える RAM を取得できるようにカーネルを再ビルドすることを検討してください。 詳細情報が必要な場合は、LINT 設定ファイルの option MAXDSIZ を調べてください。

MySQL の現在の日付に関する問題が発生した場合は、TZ 変数を設定することでおそらく解決します。 See 付録 F. 環境変数。

安全で安定したシステムにするには、-RELEASE と指定された FreeBSD カーネルだけを使用してください。

2.6.4.2. NetBSD の注意事項

NetBSD 上でコンパイルするには、GNU make が必要です。GNU make がないと、make が C++ ファイルで lint を実行したときにクラッシュします。

2.6.4.3. OpenBSD 2.5 の注意事項

OpenBSD バージョン 2.5 では、ネイティブスレッドを使用して以下のオプションを指定して MySQL をコンパイルできます。

```
CFLAGS=-pthread CXXFLAGS=-pthread ./configure --with-mit-threads=no
```

2.6.4.4. OpenBSD 2.8 の注意事項

OpenBSD 2.8 には MySQL で問題を発生させるスレッドのバグがあるということがユーザから報告されています。 OpenBSD 開発者はこの問題を修正しましたが、2001 年 1 月 25 日現在で ``-current'' ブランチでしか利用できません。 このスレッドのバグによって発生する現象として、スローレスポンス、高負荷、高 CPU 使用率、クラッシュがあります。

テーブルやディレクトリを開こうとしたときに、Error in accept:: Bad file descriptor というエラーまたはエラー 9 が発生した場合、問題は MySQL に十分なファイルディスクリプタを割り当てていないことだと考えられます。

この場合、以下のオプションを指定して、root として mysqld_safe を起動してみてください。

```
shell> mysqld_safe --user=mysql --open-files-limit=2048 &
```

2.6.4.5. BSD/OS バージョン 2.x の注意事項

MySQL をコンパイルしているときに以下のエラーが発生した場合は、ulimit 値が低すぎます。

```
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
```

ulimit -v 80000 を使用して make を再実行してみてください。この方法で効果がなく、sh を使用している場合は、csh または sh を切り替えてみてください。bash と ulimit に関する問題が BSDI ユーザから報告されています。

gcc を使用している場合は、sql_yacc.cc のコンパイルを可能にするために configure に --with-low-memory フラグを使用する必要もあるかもしれません。

MySQL の現在の日付に関する問題が発生した場合は、TZ 変数を設定することで解決すると考えられます。 See 付録 F. 環境変数。

2.6.4.6. BSD/OS バージョン 3.x の注意事項

BSD/OS バージョン 3.1 にアップグレードしてください。それが不可能な場合は、BSDI patch M300-038 をインストールしてください。

MySQL をコンフィギャするときは以下のようにします。

```
shell> env CXX=shlicc++ CC=shlicc2 \
    ./configure \
      --prefix=/usr/local/mysql \
      --localstatedir=/var/mysql \
      --without-perl \
      --with-unix-socket-path=/var/mysql/mysql.sock
```

以下のラインも有効であることが知られています。

```
shell> env CC=gcc CXX=gcc CXXFLAGS=-O3 \
    ./configure \
      --prefix=/usr/local/mysql \
      --with-unix-socket-path=/var/mysql/mysql.sock
```

必要に応じてディレクトリの場所を変更してもかまいません。また、どの場所も指定しないでデフォルトを使用することもできます。

高負荷の状況でパフォーマンスに問題がある場合は、mysqld に --skip-thread-priority オプションを使用してみてください。これによって、すべてのスレッドが同じ優先順位で実行されます。BSDI バージョン 3.1 では、これによってパフォーマンスが向上します (少なくとも BSDI がスレッドスケジューラを修正するまでの間) 。

コンパイル中に virtual memory exhausted というエラーが発生した場合は、ulimit -v 80000 を使用して make を再実行してみてください。 この方法で効果がなく、sh を使用している場合は、csh または sh を切り替えてみてください。bash と ulimit に関する問題が BSDI ユーザから報告されています。

2.6.4.7. BSD/OS バージョン 4.x の注意事項

BSDI バージョン 4.x にはスレッド関連のバグがいくつかあります。BSDI バージョン 4.x 上で MySQL を使用する場合は、すべてのスレッド関連のパッチをインストールしてください。少なくとも M400-023 はインストールする必要があります。

BSDI バージョン 4.x システムでは、共有ライブラリに関する問題が発生することがあります。 mysqladmin などのクライアントプログラムを実行できないという現象が発生します。この場合、configure に --disable-shared オプションを指定して、共有ライブラリを使用しないように再コンフィギャする必要があります。

mysqld バイナリがしばらくたつとテーブルを開けなくなるという問題が BSDI 4.0.1 上で発生したことがあります。これは、ライブラリまたはシステム関連の何らかのバグによって、ユーザが要求していないのに mysqld がカレントディレクトリを変更したためです。

この問題は、3.23.34 以降にアップグレードするか、configure を実行した後、make を実行する前に config.h から #define HAVE_REALPATH 行を削除することで解決します。

注意: このことは、BSDI 上で、データベースディレクトリを別のデータベースディレクトリにシンボリックリンクすることも、テーブルを別のデータベースにシンボリックリンクすることもできないことを意味します（別のディスクへのシンボリックリンクを作成することはできます）。

2.6.5. Mac OS X の注意事項

2.6.5.1. Mac OS X 10.x

MySQL は Mac OS X 10.x（Darwin）上で問題なく機能します。この OS 用の pthread パッチは必要ありません。

このことは、Mac OS X 10.x Server にも当てはまります。サーバプラットフォーム用のコンパイルは、クライアントバージョンの Mac OS X の場合と同じです。ただし、MySQL は、サーバ にプリインストールされていることに注意してください。

当社の Mac OS X 用バイナリは、以下の configure 行で Darwin 6.3 上でコンパイルされます。

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc \
CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors \
-fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql \
--with-extra-charsets=complex --enable-thread-safe-client \
--enable-local-infile --disable-shared
```

See 項2.1.3.「Mac OS X への MySQL のインストール」。

2.6.5.2. Mac OS X Server 1.2（Rhapsody）

MySQL on Mac OS X Server 1.2（別名 Rhapsody）上でMySQL をコンフィギャする前に、まず、http://www.prnet.de/RegEx/mysql.html にある pthread パッケージをインストールする必要があります。

See 項2.1.3.「Mac OS X への MySQL のインストール」。

2.6.6. その他の Unix の注意事項

2.6.6.1. バイナリディストリビューションの場合の HP-UX の注意事項

HP-UX 用 MySQL のバイナリディストリビューションのいくつかは、HP の depot ファイルおよび tar ファイルとして配布されます。depot ファイルを使用するには、HP-UX 10.x 以降を実行していて、HP's software depot tools にアクセスできなければなりません。

HP バージョンの MySQL は、HP-UX 10.20 を実行する HP 9000/8xx サーバ上でコンパイルされ、MIT-pthreads を使用します。この構成で MySQL が適切に動作することが知られています。MySQL バージョン 3.22.26 以降は、HP のネイティブスレッドパッケージを使用してビルドすることもできます。

その他の有効な構成

- HP-UX 10.20 以降を実行している HP 9000/7xx
- HP-UX 10.30 以降 を実行している HP 9000/8xx

以下の構成では、ほぼ確実に正常に機能しません。

- HP-UX 10.x (10.2 より前) を実行する HP 9000/7xx または 8xx
- HP-UX 9.xを実行する HP 9000/7xx または 8xx

ディストリビューションをインストールするには、以下のいずれかのコマンドを使用します。この場合、/path/to/depot は depot ファイルの完全パス名です。

- サーバ、クライアント、開発ツールなど、すべてのものをインストールする場合

```
shell> /usr/sbin/swinstall -s /path/to/depot mysql.full
```

- サーバだけをインストールする場合

```
shell> /usr/sbin/swinstall -s /path/to/depot mysql.server
```

- クライアントパッケージだけをインストールする場合

```
shell> /usr/sbin/swinstall -s /path/to/depot mysql.client
```

- 開発ツールだけをインストールする場合

```
shell> /usr/sbin/swinstall -s /path/to/depot mysql.developer
```

depot は、バイナリとライブラリを /opt/mysql に配置し、データを /var/opt/mysql に配置します。また、ブート時にサーバを自動的に起動するための適切なエントリを /etc/init.d と /etc/rc2.d に作成します当然ですが、インストールするには root である必要があります。

HP-UX tar.gz ディストリビューションをインストールするには、GNU の tar がなければなりません。

2.6.6.2. HP-UX バージョン 10.20 の注意事項

HP-UX 上で MySQL をコンパイルしているときに、小さな問題がいくつか発生します。HP-UX のネイティブコンパイラの代わりに gcc を使用することをお勧めします。これは、gcc の方が優れたコードを生成するためです。

HP-UX では gcc 2.95 を使用することをお勧めします。高いレベルの最適化フラグ (-O6 など) は、HP-UX 上では安全でない可能性があるので使用しないでください。

gcc 2.95 を使用する場合は、以下の configure 行が有効です。

```
CFLAGS="-I/opt/dce/include -fpic" \
```

```
CXXFLAGS="-I/opt/dce/include -felide-constructors -fno-exceptions \
-fno-rtti" CXX=gcc ./configure --with-pthread \
--with-named-thread-libs='-ldce' --prefix=/usr/local/mysql --disable-shared
```

gcc 3.1 を使用する場合は、以下の configure 行が有効です。

```
CFLAGS="-DHPUX -I/opt/dce/include -O3 -fPIC" CXX=gcc \
CXXFLAGS="-DHPUX -I/opt/dce/include -felide-constructors -fno-exceptions \
-fno-rtti -O3 -fPIC" ./configure --prefix=/usr/local/mysql \
--with-extra-charsets=complex --enable-thread-safe-client \
--enable-local-infile  --with-pthread \
--with-named-thread-libs=-ldce --with-lib-ccflags=-fPIC
--disable-shared
```

2.6.6.3. HP-UX バージョン 11.x の注意事項

HP-UX バージョン 11.x の場合は、MySQL バージョン 3.23.15 以降を推奨します。

標準の HP-UX ライブラリにはいくつかの重大なバグがあるため、HP-UX 11.0 上で MySQL を実行する前に以下のパッチをインストールする必要があります。

```
PHKL_22840 Streams cumulative
PHNE_22397 ARPA cumulative
```

これによって、スレッドアプリケーションで、recv() から EWOULDBLOCK を、accept() から EBADF を受け取るという問題が解決されます。

パッチ未適用の HP-UX 11.x システム上で gcc 2.95.1 を使用している場合、以下のエラーが発生します。

```
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:
```

問題は、HP-UX の pthreads_atfork() の定義に整合性がないことです。 /usr/include/sys/unistd.h:184 と /usr/include/sys/pthread.h:440 に、矛盾したプロトタイプがあります (詳細は以下で述べます) 。

1 つの解決法は、/usr/include/sys/unistd.h を mysql/include にコピーして、unistd.h を編集して、pthread.h 内の定義と一致するように変更することです。以下に diff を示します。

```
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));
```

この後は、以下の configure 行が機能します。

```
CFLAGS="-fomit-frame-pointer -O3 -fpic" CXX=gcc \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti -O3" \
./configure --prefix=/usr/local/mysql --disable-shared
```

HP-UX コンパイラと共に MySQL 4.0.5 を使用している場合は、以下を使用できます（cc B.11.11.04 でテスト済み）。

```
CC=cc CXX=aCC CFLAGS=+DD64 CXXFLAGS=+DD64 ./configure --with-extra-character-set=complex
```

以下のようなエラーは無視してかまいません。

```
aCC: warning 901: unknown option: `-3': use +help for online documentation
```

configure 時に以下のエラーが出力される場合

```
checking for cc option to accept ANSI C... no
configure: error: MySQL requires a ANSI C compiler (and a C++ compiler).
Try gcc. See the Installation chapter in the Reference Manual.
```

HP-UX の C および C++ コンパイラへのパスの前に K&R コンパイラへのパスがないことを確認します。

コンパイルできないもう 1 つの理由は、上で +DD64 フラグを定義していないことです。

HP-UX 11 のもう 1 つの可能性は、HP-UX 10.20 用の MySQL バイナリを使用することです。 これらのバイナリが HP-UX 11.00 で正常に機能することがユーザから報告されています。問題が発生した場合は、必ず HP-UX のパッチのレベルをチェックしてください。

2.6.6.4. IBM-AIX の注意事項

xlC の自動検出が Autoconf からなくなっているため、MySQL をコンパイルするときには以下のような configure の指定が必要です（この例では、IBM コンパイラを使用しています）。

```
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
```

上記は、MySQL ディストリビューションをコンパイルする際に使用されるオプションで、http://www-frec.bull.com/ にあります。

上記の configure 行で -O3 を -O2 に変更した場合は、-qstrict オプションも削除する必要があります（これは、IBM C コンパイラの制限です）。

gcc または egcs を使用して MySQL をコンパイルする場合は、-fno-exceptions フラグを使用する必要があります。gcc お

よび egcs の例外処理がスレッドセーフでないためです（これは、egcs 1.1 を使用してテストされています）。IBM アセンブラにもいくつかの既知の問題があります。これらの問題のために、このアセンブラを gcc と共に使用した場合に、不良コードが生成されることがあります。

AIX 上では、egcs および gcc 2.95 と共に以下の configure 行を使用することをお勧めします。

```
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
```

-Wa,-many は、コンパイルを成功させるために必要です。IBM は、この問題を認識していますが、回避方法があるため修正を急いでいません。gcc 2.95 に -fno-exceptions が必要かどうかはわかりませんが、MySQL は例外を使用しない上に、このオプションによってコードが高速化するので、egcs および gcc には常にこのオプションを使用することをお勧めします。

アセンブラコードで問題が発生する場合は、使用している CPU と一致するように -mcpu=xxx を変更してみてください。通常は、power2、power、または powerpc を指定する必要があるでしょう。あるいは、604 または 604e を指定する必要があるかもしれません。確証はありませんが、ほとんどの場合 "power" を指定しても問題ありません（power2 マシン上であっても同様）。

CPU が何であるかわからない場合は、"uname -m" を実行してください。xxyyyyyymmss の形式を持つ、"000514676700" のような文字列が返されます。この場合、xx と ss は常に 0、yyyyyy は一意のシステム ID、mm は CPU プレーナの ID です。http://publib.boulder.ibm.com/doc_link/en_US/a_doc_lib/cmds/aixcmds5/uname.htm にこれらの値の表があります。 これによってマシンの種類と型がわかります。このマシンの種類と型で、使用している CPU の種類を特定できます。

シグナルに関する問題が発生した場合（高負荷の状況で MySQL が突然停止する）、スレッドとシグナルに関する OS のバグを見つけた可能性があります。この場合、以下のコマンドを使用してコンフィギャすることで、シグナルを使用しないように MySQL に指示することができます。

```
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
```

これは、MySQL のパフォーマンスには影響しませんが、接続上で "スリープ状態" のクライアントを、mysqladmin kill または mysqladmin shutdown を使用して強制終了することができなくなるという副作用があります。代わりに、クライアントは自身が次回コマンドを発行したときに停止します。

AIX の一部のバージョンでは、libbind.a とリンクすると getservbyname がコアダンプします。これは AIX のバグなので、IBM に報告する必要があります。

AIX 4.2.1 と gcc を使用する場合、以下の変更を行う必要があります。

設定の後に、config.h と include/my_config.h を編集して、以下の行を変更します。

```
#define HAVE_SNPRINTF 1
```

これを以下のように変更します。

```
#undef HAVE_SNPRINTF
```

最後に、mysqld.cc の initgoups にプロトタイプを追加する必要があります。

```
#ifdef _AIX41
extern "C" int initgroups(const char *,int);
#endif
```

mysqld プロセスに大量のメモリを割り当てる必要がある場合は、'ulimit -d unlimited' を設定するだけでは十分ではありません。場合によって、mysqld_safe で以下のように設定する必要もあります。

```
export LDR_CNTRL='MAXDATA=0x80000000'
```

以下のサイトに、大量のメモリの使用についての詳細情報があります。
http://publib16.boulder.ibm.com/pseries/en_US/aixprggd/genprogc/lrg_prg_support.htm。

2.6.6.5. SunOS 4 の注意事項

SunOS 4 では、MySQL をコンパイルするために MIT-pthreads が必要です。これは、GNU make が必要であることを意味します。

一部の SunOS 4 システムは、動的ライブラリと libtool に問題があります。 以下の configure 行を実行すると、この問題を回避できます。

```
shell> ./configure --disable-shared --with-mysqld-ldflags=-all-static
```

readline をコンパイルしているときに、重複定義に関する警告が表示されることがあります。 これらの警告は無視してかまいません。

mysqld をコンパイルしているときに、implicit declaration of function という警告が表示されます。これらの警告は無視してかまいません。

2.6.6.6. Alpha-DEC-UNIX（Tru64）の注意事項

Digital Unix 上で egcs 1.1.2 を使用している場合は、gcc 2.95.2 にアップグレードしてください。これは、DEC 上の egcs にいくつかの重大なバグがあるためです。

Digital Unix 上でスレッドプログラムをコンパイルする場合、マニュアルでは、cc および cxx の -pthread オプションとライブラリ -lmach -lexc （-lpthread のほかに）を使用することが推奨されています。以下のような configure を実行してください。

```
CC="cc -pthread" CXX="cxx -pthread -O" \
./configure --with-named-thread-libs="-lpthread -lmach -lexc -lc"
```

mysqld をコンパイルしているときに、以下のようないくつかの警告が表示されることがあります。

```
mysqld.cc: In function void handle_connections()':
mysqld.cc:626: passing long unsigned int *' as argument 3 of
accept(int,sockadddr *, int *)'
```

これらの警告は無視してもかまいません。これらの警告は、configure が検出できるのはエラーだけで、警告は検出できないために発生します。

直接コマンドラインからサーバを起動した場合に、ログアウトするとサーバが停止するという問題が発生することがあります (ログアウトすると、まだ終わっていないプロセスは SIGHUP シグナルを受信します) 。その場合は、以下のようにサーバを起動します。

```
shell> nohup mysqld [options] &
```

nohup は、その後ろに指定されたコマンドに、端末から送信された SIGHUP シグナルを無視させます。あるいは、mysqld_safe を実行してサーバを起動します。mysqld_safe は、nohup を使用して mysqld を起動します。 See 項4.8.2.「mysqld_safe (mysqld のラッパ) 」。

mysys/get_opt.c をコンパイルしているときに問題が発生した場合は、ファイルの先頭から #define _NO_PROTO という行を削除してください。

Compaq の CC コンパイラを使用している場合は、以下の configure 行が有効です。

```
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 \
-noexceptions -nortti"
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
```

上記のように共有ライブラリを有効にしてコンパイルしているときに、mysql のリンク中に libtool で問題が発生した場合は、以下のコマンドを発行してその問題を回避できます。

```
cd mysql
/bin/sh ../libtool --mode=link cxx -pthread  -O3 -DDBUG_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. Alpha-DEC-OSF/1 の注意事項

DEC の CC および gcc がインストールされている場合、コンパイルで問題が発生したら、以下のような configure を実行してみてください。

```
CC=cc CFLAGS=-O CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql
```

c_asm.h ファイルで問題が発生した場合は、以下のコマンドで 'ダミー' の c_asm.h ファイルを作成して使用してください

。

```
touch include/c_asm.h
CC=gcc CFLAGS=-I./include \
CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql
```

注意: ld プログラムに関する以下の問題は、最新の DEC (Compaq) パッチキットを以下のサイトからダウンロードして修正できます。 http://ftp.support.compaq.com/public/unix/。

OSF/1 V4.0D およびコンパイラ "DEC C V5.6-071 on Digital Unix V4.0 (Rev. 878)" では、コンパイラにいくつかの未知の動作があります (未定義の asm シンボル) 。 /bin/ld も壊れているように思われます (mysqld のリンク中に _exit undefined エラーが発生する問題) 。このシステムでは、/bin/ld を OSF 4.0C のバージョンに置き換えた後、以下の configure 行を使用して MySQL をコンパイルできました。

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
```

Digital コンパイラ "C++ V6.1-029" を使用する場合は、以下の指定が有効です。

```
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 -noexceptions -nortti
export CC CFLAGS CXX CXXFLAGS
./configure --prefix=/usr/mysql/mysql --with-mysqld-ldflags=-all-static \
          --disable-shared --with-named-thread-libs="-lmach -lexc -lc"
```

OSF/1 の一部のバージョンでは、alloca() 関数が壊れています。config.h の 'HAVE_ALLOCA' を定義する行を削除して、この問題を解決します。

alloca() 関数も /usr/include/alloca.h に正しくないプロトタイプを持つことがあります。その結果表示される警告は無視してかまいません。

configure は、以下のスレッドライブラリを自動的に使用します。 --with-named-thread-libs="-lpthread -lmach -lexc -lc"。

gcc を使用している場合は、以下のように configure を実行してみてください。

```
shell> CFLAGS=-D_PTHREAD_USE_D4 CXX=gcc CXXFLAGS=-O3 ./configure ...
```

シグナルに関する問題が発生した場合 (高負荷の状況で MySQL が突然停止する) 、スレッドとシグナルに関する OS のバグを見つけた可能性があります。この場合、以下のコマンドを使用して設定することで、シグナルを使用しないように MySQL に指示することができます。

```
shell> CFLAGS=-DDONT_USE_THR_ALARM \
    CXXFLAGS=-DDONT_USE_THR_ALARM \
    ./configure ...
```

これは、MySQL のパフォーマンスには影響しませんが、接続上で "スリープ状態" のクライアントを、mysqladmin kill または mysqladmin shutdown を使用して強制終了することができなくなるという副作用があります。代わりに、クライアントは自身が次回コマンドを発行したときに停止します。

gcc 2.95.2 を使用している場合、以下のコンパイルエラーが発生する可能性が高いです。

```
sql_acl.cc:1456: Internal compiler error in `scan_region', at except.c:2566
Please submit a full bug report.
```

これを修正するには、sql ディレクトリに移動して、末尾の gcc 行を ``カットアンドペースト" しますが、-O3 は -O0 に変更します (または、コンパイル行に -O オプションがない場合は、-O0 を gcc の直後に追加します) 。この処理が終わったら、最上位のディレクトリに戻って、make を再実行します。

2.6.6.8. SGI Irix の注意事項

Irix Version 6.5.3 以降を使用している場合、CAP_SCHED_MGT 特権を持つユーザ (root など) として mysqld を実行した場合か、以下のシェルコマンドを使用して mysqld サーバにこの特権を与えた場合にのみ、mysqld はスレッドを作成することができます。

```
shell> chcap "CAP_SCHED_MGT+epi" /opt/mysql/libexec/mysqld
```

場合によって、configure を実行した後、コンパイルする前に config.h でいくつかのシンボルを再定義する必要があります。

Irix の一部のバージョンでは、alloca() 関数が壊れています。SELECT ステートメントで mysqld サーバが停止する場合は、config.h から HAVE_ALLOC と HAVE_ALLOCA_H を定義している行を削除してください。 mysqladmin create が機能しない場合は、config.h から HAVE_READDIR_R を定義している行を削除します。場合によって、HAVE_TERM_H 行も削除する必要があります。

SGI では、以下のページにあるすべてのパッチをまとめてインストールすることを推奨しています。
http://support.sgi.com/surfzone/patches/patchset/6.2_indigo.rps.html

少なくとも、最新のカーネルロールアップ、最新の rld ロールアップ、および最新の libc ロールアップだけはインストールしてください。

pthread のサポートのために、以下のページにあるすべての POSIX パッチが必要です。

http://support.sgi.com/surfzone/patches/patchset/6.2_posix.rps.html

mysql.cc をコンパイルしているときに、以下のようなエラーが発生することがあります。

```
"/usr/include/curses.h", line 82: error(1084): invalid combination of type
```

MySQL ソースツリーの最上位のディレクトリで、以下を入力します。

```
shell> extra/replace bool curses_bool < /usr/include/curses.h \
> include/curses.h
shell> make
```

スケジューリングに関する問題も報告されています。1 つのスレッドだけが稼動している場合、パフォーマンスが低くなります。別のクライアントを起動して、この問題を回避してください。これによって、もう 1 方のスレッドの実行速度が 2 ー 10 倍になることがあります。これは、Irix スレッドに関する不明点の多い問題です。この問題が修正できるまで間に合わせの解決法を見つけなくてはなりません。

gcc を使用してコンパイルしている場合は、以下の configure コマンドを使用できます。

```
CC=gcc CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql --enable-thread-safe-client \
--with-named-thread-libs=-lpthread
```

Irix 6.5.11 上でネイティブの Irix C コンパイラおよび C++ コンパイラのバージョン 7.3.1.2 を使用している場合、以下が有効であることが報告されています。

```
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-innodb --with-berkeley-db \
--with-libwrap=/usr/local \
--with-named-curses-libs=/usr/local/lib/libncurses.a
```

2.6.6.9. SCO の注意事項

現行の移植版は、``sco3.2v5.0.5"、``sco3.2v5.0.6"、および ``sco3.2v5.0.7" のシステム上でのみテストされています。``sco 3.2v4.2" への移植版でも大きな進展がありました.

現時点で、OpenServer 上で推奨されるコンパイラは gcc 2.95.2 です。このコンパイラを使用すると、以下のコマンドだけで MySQL をコンパイルできます。

```
CC=gcc CXX=gcc ./configure ... (options)
```

1. OpenServer 5.0.x の場合は、Skunkware の gcc-2.95.2p1 以降を使用する必要がある。gcc-2.95.2p1 を入手するには、http://www.sco.com/skunkware/ でブラウザ版の OpenServer パッケージを選択するか、ftp サイト ftp2.caldera.com のpub/skunkware/osr5/devtools/gcc ディレクトリに ftp 接続する。

2. この製品と開発システムのための GCC 2.5.x の移植が必要である。これらは、このバージョンの SCO Unix 上に存在する必要がある。GCC Dev システムだけを使用することはできない。

3. 最初に、FSU Pthreads パッケージを入手して、それをインストールする必要がある。 FSU Pthreads パッケージは、http://moss.csc.ncsu.edu/~mueller/ftp/pub/PART/pthreads.tar.gz にある。 また、http://www.mysql.com/Downloads/SCO/FSU-threads-3.5c.tar.gz からプリコンパイルされたパッケージを入手することもできる。

4. FSU Pthreads は、SCO Unix 4.2 で tcpip を使用してコンパイルできる。あるいは、GCC 2.5.x ODT または OS 3.0 の適正な移植版を使用して SCO Development System がインストールされた、OpenServer 3.0 または Open Desktop 3.0 (OS 3.0 ODT 3.0) では、GCC 2.5.x の適正な移植版が必要である。適正な移植版を使用しないと、多くの問題が発生する。この製品の移植には、SCO Unix Development system が必要である。これがないと、必要なライブラリとリンカが足りない。

5. FSU Pthreads をシステムにビルドするには、以下の手順で行う。

   a. threads/src ディレクトリで ./configure を実行し、SCO OpenServer オプションを選択する。このコマンドによって、Makefile.SCO5 が Makefile にコピーされる。

   b. make を実行する。

   c. デフォルトの /usr/include ディレクトリにインストールするには、root としてログインして、thread/src ディレク

トリに移動し（cd）、make install を実行する。

6. MySQL をコンパイルするときは、必ず GNU make を使用する。

7. root として mysqld_safe を起動しないと、おそらくプロセス当たり 110 個（デフォルト）のオープンファイルしか取得しない。mysqld は、これに関する注記をログファイルに書き込む。

8. SCO 3.2V5.0.5 では、FSU Pthreads バージョン 3.5c 以降を使用する必要がある。 また、gcc 2.95.2 以降を使用する必要もある。

   以下の configure コマンドが有効である。

   ```
   shell> ./configure --prefix=/usr/local/mysql --disable-shared
   ```

9. SCO 3.2V4.2 では、FSU Pthreads バージョン 3.5c 以降を使用する必要がある。 以下の configure コマンドが有効である。

   ```
   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"
   ```

   インクルードファイルに関する問題が発生することがある。この場合、http://www.mysql.com/Downloads/SCO/SCO-3.2v4.2-includes.tar.gz で、新しい SCO 固有のインクルードファイルを見つける。 MySQL ソースツリーの include ディレクトリにこのファイルをアンパックする。

SCO development の注意事項

- MySQL は、自動的に FSU Pthreads を検出して、-lgthreads -lsocket -lgthreads を使用して mysqld をリンクする。
- SCO development ライブラリは、FSU Pthreads に再入可能である。 SCO は、自社のライブラリの関数は再入可能であるから、FSU Pthreadsでも再入可能であるはずだと公言している。OpenServer 上の FSU Pthreads は、SCO スキームを使用して再入可能なライブラリを作成する。
- FSU Pthreads（http://www.mysql.com/ にあるバージョン以降）は、GNU malloc でリンクされた状態で提供される。メモリ使用率に関する問題が発生した場合は、libgthreads.a と libgthreads.so に gmalloc.o がインクルードされていることを確認する。
- FSU Pthreads で、pthread 対応のシステムコールは、read()、write()、getmsg()、connect()、accept()、select()、および wait() である。
- CSSA-2001-SCO.35.2（このパッチは erg711905-dscr_remap セキュリティパッチ（バージョン 2.0.0）として「custom」にリストされている）は FSU スレッドを破壊し、mysqld を不安定にする。OpenServer 5.0.6 マシン上で mysqld を実行する場合は、このパッチを削除する必要がある。
- SCO は、ftp://ftp.sco.com/pub/openserver5（OpenServer 5.0.x 用）でオペレーティングシステムパッチを提供している。
- SCO は、セキュリティ修正プログラムと libsocket.so.2 を ftp://ftp.sco.com/pub/security/OpenServer および

ftp://ftp.sco.com/pub/security/sse （ OpenServer 5.0.x 用 ）で提供している。

- OSR506 より前のバージョンのセキュリティ修正プログラム。また、OSR506 より前のバージョンのシステムへのインストールに関する説明が含まれた libsocket.so.2 と libresolv.so.1 の両方として、ftp://stage.caldera.com/pub/security/openserver/ または ftp://stage.caldera.com/pub/security/openserver/CSSA-2001-SCO.10/ にある telnetd 修正プログラム。

  MySQL のコンパイルや使用を試みる前に、上記のパッチをインストールすることをお勧めする。

SCO に DBI をインストールする場合は、DBI-xxx と各サブディレクトリに含まれている Makefile を編集する必要があります。

注意: 以下は、gcc 2.95.2 以降を想定したものです。

```
変更前:                          変更後:
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

変更前:
CCCFLAGS = -belf -dy -w0 -U M_XENIX -DPERL_SCO5 -I/usr/local/include

変更後:
CCFLAGS = -U M_XENIX -DPERL_SCO5 -I/usr/local/include
```

icc または cc でコンパイルされた DBI モジュールを Perl dynaloader がロードしないので、このように処理する必要があります。

Perl は、cc でコンパイルされると最もよく機能します。

2.6.6.10. SCO UnixWare バージョン 7.1.x の注意事項

MySQL のバージョン 3.22.13 以降の MySQL のバージョンと UnixWare 7.1.0 のバージョンを使用する必要があります。これは、これらのバージョンで UnixWare 上の移植性と OS に関する問題がいくつか修正されているためです。

UnixWare バージョン 7.1.x 上で以下の configure コマンドを使用して MySQL をコンパイルすることができました。

```
CC=cc CXX=CC ./configure --prefix=/usr/local/mysql
```

gcc を使用する場合は、gcc 2.95.2 以降をインストールする必要があります。

```
CC=gcc CXX=g++ ./configure --prefix=/usr/local/mysql
```

1. SCO は、ftp://ftp.sco.com/pub/unixware7 （ UnixWare 7.1.1 および 7.1.3 ）と ftp://ftp.sco.com/pub/openunix8 （

OpenUNIX 8.0.0) で、オペレーティングシステムパッチを提供している。

2. SCO は、ftp://ftp.sco.com/pub/security/OpenUNIX （ OpenUNIX) および ftp://ftp.sco.com/pub/security/sse （ UnixWare) で、セキュリティ修正プログラムについての情報を提供している。

2.6.7. OS/2 の注意事項

MySQL は、多数のオープンファイルを使用します。そのため、CONFIG.SYS ファイルに以下のような記述を追加する必要があります。

```
SET EMXOPT=-c -n -h1024
```

この処理を行わないと、おそらく以下のエラーが発生します。

```
File 'xxxx' not found (Errcode: 24)
```

OS/2 Warp 3 で MySQL を使用する場合は、FixPack 29 以降が必要です。OS/2 Warp 4 で MySQL を使用する場合は、FixPack 4 以降が必要です。これは、Pthreads ライブラリの要件です。MySQL は、HPFS、FAT32など、長いファイル名をサポートするパーティションにインストールする必要があります。

INSTALL.CMD スクリプトは、OS/2 の CMD.EXE から実行しなければならず、4OS2.EXE などの置換シェルでは機能しないことがあります。

scripts/mysql-install-db スクリプトは、名前が変更されました。現在、install.cmd と呼ばれているこのスクリプトは、REXX スクリプトで、デフォルトの MySQL セキュリティ設定をセットアップし、MySQL 用の ワークプレイスシェルアイコンを作成します。

動的モジュールサポートは、コンパイルはされていますが、完全にはテストされていません。動的モジュールは、Pthreads ランタイムライブラリを使用してコンパイルする必要があります。

```
gcc -Zdll -Zmt -Zcrtdll=pthrdrtl -I../include -I../regex -I.. \
   -o example udf_example.cc -L../lib -lmysqlclient udf_example.def
mv example.dll example.udf
```

注意:OS/2 での制限のために、UDF モジュール名の語幹ステムは 8 文字を超えることはできません。モジュールは、/mysql2/udf ディレクトリに格納されます。safe-mysqld.cmd スクリプトはこのディレクトリを BEGINLIBPATH 環境変数に設定します。UDF モジュールを使用している場合、指定した拡張子は無視されます。拡張子は .udf と仮定されます。たとえば、Unix では、共有モジュールが example.so という名前の場合、以下のようなコマンドでそこから関数をロードします。

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME "example.so";
```

OS/2 では、モジュールが example.udf という名前であっても、ユーザはモジュール拡張子を指定しません。

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME "example";
```

2.6.8. Novell NetWare の注意事項

NetWare への MySQL の移植は、Novell 主導で行われた作業です。Novell のユーザは、NetWare 6.5を実行するすべての

サーバに対する自動的な商用ライセンスを備えた MySQL バイナリが、NetWare 6.5 にバンドルされることを歓迎するはずです。

See 項2.1.4.「NetWare への MySQL のインストール」。

NetWare 用の MySQL は、Metrowerks CodeWarrior for NetWare と特別なクロスコンパイルバージョンの GNU autotool の組み合わせを使用してコンパイルされています。今後、NetWare 用の MySQL をビルドと最適化に関する詳細情報が必要になったら、このセクションをもう一度参照してください。

2.6.9. BeOS の注意事項

当社は、過去に MySQL を BeOS に 80% 移植すると語る BeOS の開発者と話したことがありますが、その後連絡はありません。

2.7. Perl インストールについてのコメント

2.7.1. Unix への Perl のインストール

MySQL は、DBI/DBD クライアントインタフェースという形で Perl をサポートします。See 項11.5.「MySQL Perl API」。 Perl DBD/DBI クライアントコードは、Perl バージョン 5.004 以降を必要とします。これより前のバージョンの Perl を使用している場合は、このインタフェースは正常に機能しません。

MySQL の Perl モジュールの作成には、MySQL クライアントの開発環境(ヘッダファイルや libmysqlclient ライブラリ)をインストールすることが必要です。 RPM ファイルから MySQL をインストールした場合は、開発環境は MySQL-devel RPM に含まれています。MySQL-devel パッケージをインストールしていることを確認してください。

バージョン 3.22.8 以降、Perl のモジュールはメインのMySQL ディストリビューションとは別に配布されています。Perl モジュールをインストールする場合は、http://www.mysql.com/downloads/api-dbi.html から必要なファイルを入手できます。

Perl モジュールのディストリビューションは、tar 形式の圧縮アーカイブとして提供され、MODULE-VERSION.tar.gz などの名前が付いています。この場合、MODULE はモジュール名で、VERSION はバージョン番号です。Data-Dumper、DBI、DBD-mysql の各ディストリビューションを入手して、この順番でそれらをインストールする必要があります。インストール手順については、以下を参照してください。 ここでは、Data-Dumper モジュールの場合の例で説明しますが、3 つのディストリビューションすべてで手順は同じです。

1. ディストリビューションをカレントディレクトリにアンパックする。

   ```
   shell> gunzip < Data-Dumper-VERSION.tar.gz | tar xvf -
   ```

   このコマンドによって、Data-Dumper-VERSION という名前のディレクトリが作成される。

2. アンパックしたディストリビューションの最上位ディレクトリに移動する。

   ```
   shell> cd Data-Dumper-VERSION
   ```

3. ディストリビューションをビルドして、すべてのものをコンパイルする。

   ```
   shell> perl Makefile.PL
   shell> make
   shell> make test
   ```

```
shell> make install
```

make test コマンドは、モジュールが正常に機能することを確認するので重要です。注意: DBD-mysql のインストール時にこのコマンドを実行してインタフェースコードを実行する場合は、MySQL サーバが稼動していなければなりません。稼動していないとテストは失敗します。

新しいリリースの MySQL をインストールするときは常に、DBD-mysql ディストリビューションの再ビルドと再インストールを行うことをお勧めします。MySQL をアップグレードした後にすべての DBI スクリプトが失敗するというような現象に気付いた場合は、特に推奨します。

システムディレクトリに Perl モジュールをインストールする権限がない場合や、ローカルの Perl モジュールをインストールする場合は、以下を参照してください。

```
http://servers.digitaldaze.com/extensions/perl/modules.html#modules
```

Installing New Modules that Require Locally Installed Modules という見出しの下を調べてください。

2.7.2. Windows への ActiveState Perl のインストール

Windows では、以下を実行して、ActiveState Perl が組み込まれた MySQL DBD モジュールをインストールする必要があります。

- http://www.activestate.com/Products/ActivePerl/ から ActiveState Perl を入手してインストールする。
- コンソールウィンドウ (``DOS ウィンドウ") を開く。
- 必要に応じて、HTTP_proxy 変数を設定する。たとえば、以下のコマンドを使用する。

  ```
  set HTTP_proxy=my.proxy.com:3128
  ```

- PPM(Perl Package Module) プログラムを起動する。

  ```
  C:\> c:\perl\bin\ppm.pl
  ```

- DBI をまだインストールしていない場合は、ここでインストールする。

  ```
  ppm> install DBI
  ```

- 上記のコマンドが成功したら、以下のコマンドを実行する。

  ```
  install \
  ftp://ftp.de.uu.net/pub/CPAN/authors/id/JWIED/DBD-mysql-1.2212.x86.ppd
  ```

上記のコマンドは、少なくとも ActiveState Perl バージョン 5.6 の場合は有効です。

このコマンドが正常に機能しない場合は、代わりに MyODBC ドライバをインストールして、ODBC を通じて MySQL サーバに接続してください。

```
use DBI;
```

```
$dbh= DBI->connect("DBI:ODBC:$dsn",$user,$password) ||
 die "Got error $DBI::errstr when connecting to $dsn\n";
```

2.7.3. Perl DBI/DBD インタフェース使用時の問題

../mysql/mysql.so モジュールを検出できないというメッセージが Perl から出力された場合は、Perl が共有ライブラリ libmysqlclient.so を見つけることができなかったことを意味します。

この問題は、以下のいずれかの方法で解決できます。

- perl Makefile.PL ではなく perl Makefile.PL -static -config を使用して、DBD-mysql をコンパイルする。
- 他の共有ライブラリが配置されているディレクトリ（おそらく、/usr/lib または/lib）に libmysqlclient.so をコピーする。
- Linux では、libmysqlclient.so が配置されているディレクトリのパス名を /etc/ld.so.conf ファイルに追加できる。
- libmysqlclient.so が配置されているディレクトリのパス名を LD_RUN_PATH 環境変数に追加する。

DBD-mysql から以下のエラーが出力された場合は、おそらく gcc を使用しています（または gcc でコンパイルされた古いバイナリを使用しています）。

```
/usr/bin/perl: can't resolve symbol '__moddi3'
/usr/bin/perl: can't resolve symbol '__divdi3'
```

mysql.so ライブラリをビルドするときに、リンクコマンドに -L/usr/lib/gcc-lib/... -lgcc を追加します（Perl クライアントをコンパイルするときに、make からの mysql.so に関する出力をチェックしてください）。-L オプションによって、システム内の libgcc.a が配置されているディレクトリのパス名が指定されます。

この問題のもう 1 つの原因は、Perl と MySQL の両方が gcc でコンパイルされていないことです。この場合、両方を gcc でコンパイルすることでこの不一致を解消できます。

テストを実行したときに、DBD-mysql から以下のエラーが出力されることがあります。

```
t/00base.....install_driver(mysql) failed:
Can't load '../blib/arch/auto/DBD/mysql/mysql.so' for module DBD::mysql:
../blib/arch/auto/DBD/mysql/mysql.so: undefined symbol:
uncompress at /usr/lib/perl5/5.00503/i586-linux/DynaLoader.pm line 169.
```

その場合は、-lz 圧縮ライブラリをリンク行に追加する必要があります。これは、lib/DBD/mysql/Install.pm ファイルの以下の行を変更して行います。

```
$sysliblist .= " -lm";
```

この行を以下のように変更します。

```
$sysliblist .= " -lm -lz";
```

この後、必ず make realclean を実行して、最初からインストールを行います。

動的リンク（SCO など）をサポートしていないシステム上で Perl モジュールを使用する場合は、DBI と DBD-mysql が組み込まれた静的バージョンの Perl を作成します。これがうまくいく方法は、DBI コードがリンクされた Perl のバージョンを作成して、現在の Perl の上にインストールすることです。次に、それを使用して、DBD コードが追加でリンクされた Perl のバージョンをビルドして、インストールします。

SCO では、以下の環境変数が設定されている必要があります。

```
shell> LD_LIBRARY_PATH=/lib:/usr/lib:/usr/local/lib:/usr/progressive/lib
or
shell> LD_LIBRARY_PATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:\
/usr/progressive/lib:/usr/skunk/lib
shell> LIBPATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:\
/usr/progressive/lib:/usr/skunk/lib
shell> MANPATH=scohelp:/usr/man:/usr/local1/man:/usr/local/man:\
/usr/skunk/man:
```

最初に、DBI ディストリビューションが配置されているディレクトリで以下のコマンドを実行して、静的にリンクされた DBI モジュールが組み込まれた Perl を作成します。

```
shell> perl Makefile.PL -static -config
shell> make
shell> make install
shell> make perl
```

その後、作成した Perl をインストールする必要があります。make perl の出力に、インストールを行うために実行する必要がある正確な make コマンドが示されます。SCO の場合は、make -f Makefile.aperl inst_perl MAP_TARGET=perl です。

次に、今作成した Perl を使用して、DBD-mysql ディストリビューションが配置されているディレクトリで以下のコマンドを実行して、やはり静的にリンクされた DBD::mysql モジュールが組み込まれた Perl をもう 1 つ作成します。

```
shell> perl Makefile.PL -static -config
shell> make
shell> make install
shell> make perl
```

最後に、この新しい Perl をインストールする必要があります。ここでも、make perl の出力に、使用するコマンドが示されます。

第3章 MySQL チュートリアル

この章は、mysql クライアントプログラムを使用して簡単なデータベースを作成し、使用する方法について説明する MySQL チュートリアルです。 mysql (``ターミナルモニタ" または単に ``モニタ" とも呼ばれる) は、MySQL サーバへの接続、クエリの実行、および結果の表示の各機能を持つ対話式プログラムです。mysql は、バッチモードでも使用できます。クエリを記述したファイルをあらかじめ作成しておき、そのファイルの内容を実行するように mysql に指示します。この章では、mysql のこれら 2 つの使用方法について説明します。

--help オプションを指定して mysql を呼び出すと、そのオプションの一覧が表示されます。

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

この章では、マシンに mysql がインストールされていること、および MySQL サーバに接続できる環境にあることを前提としています。この前提が成立しない場合は、MySQL 管理者にお問い合わせください。ご自分が MySQL 管理者である場合、このマニュアルの別のセクションを参照する必要があります。

この章では、データベースを設定して使用するまでの過程について、最初から最後まで説明します。既存のデータベースに接続する方法に関心がある場合は、データベースおよびそこに含まれるテーブルの作成方法について説明しているセクションは飛ばしてもかまいません。

この章は基本的にチュートリアルとして記述されているので、詳細な説明の多くは省略されています。この章で説明している内容の詳細については、このマニュアルの関連セクションを参照してください。

3.1. サーバへの接続およびサーバからの切断

サーバに接続するには、通常 mysql を呼び出す際に MySQL ユーザ名および、ほとんどの場合、パスワードを入力する必要があります。ログインするマシンとサーバが動作しているマシンが異なる場合は、ホスト名も入力する必要があります。接続する際に必要な接続パラメータ (使用するホスト名、ユーザ名、およびパスワード) については、管理者にお問い合わせください。正しい接続パラメータがわかったら、以下のようにそれらを指定することで、サーバに接続できます。

```
shell> mysql -h host -u user -p
Enter password: ********
```

******** はパスワードを表します。パスワードは、mysql が Enter password: プロンプトを表示してから入力します。

パスワードが正しければ、mysql> プロンプトの後に以下のようなウェルカムメッセージが表示されます。

```
shell> mysql -h host -u user -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 4.0.14-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

プロンプトは、mysql がユーザからのコマンド入力を受け付けられる状態であることを示します。

MySQL のインストール方法によっては、ローカルホスト上で動作するサーバに匿名ユーザ (名前のないユーザ) として接続できます。使用しているマシンにそのような方法で MySQL がインストールされている場合、以下のように何もオプシ

ョンを指定しないで mysql を呼び出しても、MySQL サーバに接続できます。

```
shell> mysql
```

サーバに正常に接続した後は、mysql> プロンプトで QUIT (または \q) と入力することで、いつでもサーバから切断できます。

```
mysql> QUIT
Bye
```

Unix 上では、Ctrl-D キーを押してサーバから切断することもできます。

以下のセクションに示す例のほとんどは、サーバに接続した状態であることを前提としています。mysql> プロンプトが表示されている場合は、サーバに接続していることを示します。

3.2. クエリの入力

前のセクションで説明したように、サーバに接続していることを確認します。サーバに接続していることを確認するだけでは使用するデータベースの選択は行われませんが、ここではそれで十分です。今の時点では、テーブルの作成、テーブルへのデータロード、およびテーブルからのデータ取得を実行する方法よりも、クエリを発行する方法を学ぶことが重要です。このセクションでは、mysql が動作するメカニズムに慣れるためのさまざまなクエリを使用しながら、コマンド入力の基本的な原則について説明します。

以下のコマンドは、サーバにそのバージョン番号と現在の日付を照会する簡単なコマンドです。mysql> プロンプトの後に以下のとおりに入力し、Enter を押してください。

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

このクエリによって、mysql についてのさまざまなことがわかります。

- 通常、コマンドは SQL ステートメントの後にセミコロンを記述する形式で入力する (セミコロンが不要な場合もあり、たとえば前述の QUIT がそれに該当する。ほかの例外については後述する) 。
- コマンドを発行すると、mysql はそれをサーバに送信して実行を依頼し、返された結果を表示する。そして、mysql> プロンプトを再度表示して、次のコマンド入力を受け付けられる状態であることを示す。
- mysql はクエリの出力を表形式 (行と列) で表示する。先頭行には列のラベルが表示される。それ以降の行にはクエリの結果が表示される。通常は、データベースから取得したカラムの名前が、列のラベルとして表示される。テーブルのカラムではなく、式の値を取得した場合 (上述の例の場合) 、mysql は式そのものを列のラベルとして使用する。
- mysql は、返されたレコード数およびクエリの実行に要した時間を表示する。これらの値からサーバのだいたいのパフォーマンスを知ることができる。ただし、これらの値はクエリの実行を開始してから終了するまでに経過した時間であり (CPU 時間やマシン時間ではない) 、サーバの負荷やネットワーク遅延の影響を受けるので、正確な値ではない (

以後、この章の例では ``rows in set'' の行の表示は省略する ）。

キーワードは大文字と小文字のどちらでも入力できます。以下のクエリはすべて同等です。

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

以下に示す例では、mysql を簡単な計算機として使用できることが示されています。

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+------------+---------+
| SIN(PI()/4) | (4+1)*5 |
+------------+---------+
|    0.707107 |      25 |
+------------+---------+
```

これまでの例で使用したクエリは比較的短い単一行ステートメントでしたが、単一行に複数ステートメントを入力することもできます。この場合も、各ステートメントの末尾にセミコロンを入力するだけです。

```
mysql> SELECT VERSION(); SELECT NOW();
+-------------+
| VERSION()   |
+-------------+
| 3.22.20a-log |
+-------------+

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

コマンドは 1 行に収める必要はなく、長いコマンドを複数行にわたって記述しても問題ありません。mysql は、入力行の終わりではなく、セミコロンをステートメントの終わりと判断して処理します。つまり、mysql への入力はフリーフォーマットであり、入力された行は、セミコロンが読み込まれるまで実行されません。

以下に簡単な複数行ステートメントを示します。

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

この例を見ると、クエリの最初の行を入力した後に、プロンプトが mysql> から -> に変わっています。mysql は、このようにプロンプトを変えることによって、途中まで入力されたステートメントの残りの部分の入力を待っていることを示します。プロンプトには有用な情報が表示されます。この情報に注目することで、mysql が現在何を待っているのかを知ることができます。

途中まで入力したコマンドの実行をキャンセルする場合は、\c を入力します。

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

この例でもプロンプトに注目してください。\c を入力するとプロンプトは mysql> に切り替わり、mysql が新しいコマンドの入力を待っていることが示されます。

表示される可能性があるプロンプトと、それが意味する mysql の状態について、以下の表に示します。

プロンプト	意味
mysql>	新しいコマンドの入力待ち
->	複数行コマンドで次の行の入力待ち
'>	単一引用符 (‘’’) で始まる文字列を読み込み中の次の行の入力待ち
">	二重引用符 (‘”’) で始まる文字列を読み込み中の次の行の入力待ち
`>	バッククォート記号 (‘`’) で始まる識別子を読み込み中の次の行の入力待ち

単一行のコマンドを発行するつもりで末尾のセミコロンの入力を忘れたために、複数行ステートメントになってしまう場合がよくあります。この場合、mysql は入力待ちになります。

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

この場合 (ステートメントの入力を完了したつもりなのに -> プロンプトが表示されるだけ) 、たいていは mysql はセミコロンが入力されるのを待っている状態です。表示されるプロンプトの意味に気が付かない場合、対処法がわかるまでしばらくかかるかもしれません。このステートメントを完了するには、セミコロンを入力します。それによって mysql が入力されたステートメントを実行します。

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

'> および "> の 2 つのプロンプトは、文字列の入力中に表示されます。MySQL では、文字列を ‘’’ または ‘”’ で囲んで表記することができます ('hello'、"goodbye" など) 。また、mysql では、複数行にわたる文字列を入力できます。表示されるプロンプトが '> または "> の場合、‘’’ または ‘”’ のどちらかの引用符で始まった文字列を含む行を入力している途中であり、その文字列の終わりを示す引用符がまだ入力されていないことを意味します。この動作は、実際に複数行にわたる文字列を入力している場合は問題になりませんが、実際にそういうケースは多いのでしょうか。いいえ、それほど多くはありません。むしろ、'> および "> のプロンプトが表示されることで、うっかり引用符を入力し忘れたことに気付かされる場合がほとんどです。例を示します。

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

このような SELECT ステートメントを入力してから Enter を押して、結果が表示されるのを待っていても、何も表示されません。この場合、このクエリになぜこんなに時間がかかるのかを不思議に思う前に、"> プロンプトの意味に気付く必要があります。このプロンプトは、mysql が入力途中の文字列の残りの部分が入力されるのを待っていることを示します（このステートメントには、"Smith で始まる文字列の終わりを示す引用符がないというエラーがあります）。

この場合、何をすればよいでしょうか。最も簡単な方法はコマンドをキャンセルすることです。ただし、この場合は \c を入力するだけではキャンセルできません。その入力は、現在途中まで入力されている文字列の一部として mysql に解釈されるためです。代わりに、まず文字列の終わりを示す引用符を入力し（これによって mysql が文字列の終わりを認識する）、それから \c を入力します。

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

プロンプトが mysql> に戻り、mysql が新しいコマンドの入力を待っていることが示されます。

`> プロンプトは、'> および "> の 2 つのプロンプトに似ていますが、バッククォートで囲まれた識別子の入力が完了していないことを示します。

'>、">、および `> の各プロンプトの意味を知ることは重要です。これに気付かないと、文字列の終わりの入力を忘れた場合に、その後入力した内容が、QUIT も含めて、すべて mysql に無視されます。このような状態になると、特に現在のコマンドをキャンセルする前に文字列の終わりを示す引用符を入力する必要があることを知らない場合は、非常に混乱します。

3.3. データベースの作成および使用

コマンドの入力方法がわかったところで、次はデータベースにアクセスします。

自宅でさまざまなペットを飼っていて、それぞれについてさまざまな情報を記録することを考えます。データを格納するためのテーブルを作成し、必要な情報をロードすることで、それを実現できます。テーブルからデータを取得することで、ペットに関するさまざまな質問に答えることもできます。ここでは、以下の作業手順について説明します。

- データベースを作成する。
- テーブルを作成する。
- テーブルにデータをロードする。
- さまざまな方法でテーブルからデータを取得する。
- 複数のテーブルを使用する。

ペットデータベースは単純ですが（意図的）、同じような種類のデータベースが実際に使用される状況はいくらでも考えられます。たとえば、農家が家畜を管理したり、獣医が病歴を記録する際に、同じようなデータベースを使用できます。いくつかのクエリとサンプルデータを含むペットデータベースのディストリビューションは、MySQL Web サイトから入手できます。ディストリビューションは、圧縮 tar 形式（http://downloads.mysql.com/docs/menagerie-db.tar.gz）または Zip 形式（http://downloads.mysql.com/docs/menagerie-db.zip）で提供されています。

SHOW ステートメントを実行すると、現在サーバ上に存在するデータベースの一覧を表示できます。

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

多くの場合、表示されるデータベースの一覧は上の例と実際のマシンでは異なりますが、mysql および test の 2 つのデータベースはどちらにも含まれていると思われます。mysql データベースは、ユーザ特権の定義に使用するので必須です。test データベースは、ユーザがテストに使用する作業場所として提供されています。

注意: SHOW DATABASES 特権がない場合はすべてのデータベースは表示できません。 See 項4.4.1.「GRANT および REVOKE の構文」。

test データベースが存在している場合、アクセスしてみます。

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

注意: USE は、QUIT と同じように、セミコロンを末尾に付ける必要はありません（セミコロンを付けても悪影響はありません）。USE ステートメントは別の意味で特別です。このステートメントは単一行で入力する必要があります。

test データベースを使用して（アクセス権がある場合）、以降の例を試すことができますが、その際にこのデータベースに作成したオブジェクトは、このデータベースにアクセスできるほかのユーザによって削除される可能性があります。そのため、独自のデータベースを使用させてもらえるように MySQL 管理者に依頼する必要があります。ここでは menagerie という名前のデータベースを使用するものとします。管理者は以下のようなコマンドを実行する必要があります。

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

ここで、your_mysql_name は使用する MySQL ユーザ名、your_client_host はサーバに接続する際に使用するホストです。

3.3.1. データベースの作成および選択

管理者が、権限を設定する際にデータベースも作成した場合は、それをすぐに使い始めることができます。そうでなければ、自分でデータベースを作成する必要があります。

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

Unix の場合、データベース名は（SQL キーワードとは異なり）大文字と小文字が区別されます。したがって、上記のコマンドで作成したデータベースを参照する際は必ず menagerie と指定する必要があります。Menagerie や MENAGERIE などを指定しても参照できません。これは、テーブル名でも同じです（Windows の場合はこの制限は適用されないが、1 つのクエリ内で使用するデータベース名およびテーブル名では、大文字と小文字の指定方法を統一する必要があります）。

データベースを作成しただけでは、そのデータベースの選択は行われません。明示的に選択する必要があります。menagerie をカレントデータベースにするには、以下のコマンドを使用します。

```
mysql> USE menagerie
Database changed
```

データベースを作成する必要があるのは 1 回だけですが、データベースの選択は、mysql セッションを開始するたびに実行する必要があります。データベースは、上記の例のように USE ステートメントを発行して選択します。別の方法として、mysql を呼び出すコマンドラインでデータベースを選択することもできます。その場合は、必要な接続パラメータを記述した後にデータベース名を追加するだけです。例を示します。

```
shell> mysql -h host -u user -p menagerie
Enter password: ********
```

注意: 上記の例では menagerie はパスワードではありません。パスワードをコマンドラインの -p オプションの後で指定する場合は、スペースを空けずに記述します (たとえば、-p mypassword ではなく、-pmypassword と指定する) 。ただし、パスワードをコマンドラインに記述すると、同じマシンにログインしているほかのユーザからそのパスワードが参照可能な状態になるので、そのような指定方法は推奨できません。

3.3.2. テーブルの作成

データベースの作成は簡単です。ただし、以下の SHOW TABLES の結果でわかるように、現在は空のままです。

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

難しいのは、データベースの構造、すなわち必要なテーブルとそれぞれのカラムを決めることです。

まず、ペットごとのレコードを持つテーブルを作成します。このテーブルには pet という名前を付けて、少なくとも動物の名前だけは格納する必要があります。名前だけではテーブルを作成する意味があまりないので、ほかの情報も格納する必要があります。たとえば、家族の中の複数の人物がペットを飼っている場合、ペットごとの飼い主を知りたい場合があります。また、種類や性別などの基本情報を記録することもできます。

年齢についてはどうでしょうか。年齢も重要な情報ですが、それをデータベースに格納するのは適切ではありません。年齢は時間の経過とともに変化するので、記録を頻繁に更新する必要があるためです。代わりに、誕生日などの固定値を格納するほうが適切です。そうすれば、必要なときに現在の日付と誕生日との差分を計算して年齢を知ることができます。MySQL には日付計算を行うための関数が用意されているので、これは難しいことではありません。年齢の代わりに誕生日を格納することには、ほかの長所もあります。

- データベースを使用して、ペットの今度の誕生日を知らせる、などの作業を実行できる (注意: ペットの誕生日を知らせるクエリなどばかばかしいと考えるかもしれないが、人間的な行動をするためにコンピュータを使用するという意味では、ビジネスデータベースを使用して誕生日祝いを贈る必要がある顧客を調べる場合のクエリと同じである) 。
- 現在の日付以外の日付に関連した年齢を計算できる。たとえば、データベースに命日を格納すれば、ペットが死んだときの年齢を簡単に計算できる。

pet テーブルに格納すると便利な情報をほかにも思いつくかもしれませんが、これまでに挙げた名前、飼い主、種類、性別、誕生日、命日だけで、現在は十分です。

CREATE TABLE ステートメントを使用してテーブルのレイアウトを指定します。

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

name、owner、および species の各カラムの値は長さが固定ではないので、データ型としては VARCHAR が適しています。これらのカラムの長さはすべて同じである必要はありません。また、20 文字である必要もありません。1 から 255 の範囲で最適と思われる長さを指定できます（指定した長さが適切ではなく、後からもっと長いフィールドが必要なことがわかった場合のために、MySQL には ALTER TABLE ステートメントが用意されている）。

ペットの記録で性別を表すには、"m" と "f"、または "male" と "female" など、さまざまな表記が考えられます。ここは、"m" と "f" の 1 文字の値を使用するのが最も簡単です。

birth および death の各カラムに DATE データ型を使用するのは自明な選択です。

これでテーブルが作成できました。SHOW TABLES を実行すると、以下の情報が出力されます。

```
mysql> SHOW TABLES;
+--------------------+
| Tables in menagerie |
+--------------------+
| pet                |
+--------------------+
```

テーブルが想定どおりに作成されたことを確認するために、DESCRIBE ステートメントを実行します。

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

DESCRIBE は、テーブルのカラムの名前やそのデータ型を忘れた場合など、いつでも使用できます。

3.3.3. テーブルへのデータのロード

テーブルを作成した後は、そこにデータを追加する必要があります。データを追加するには、LOAD DATA ステートメントおよび INSERT ステートメントを使用します。

ペットのレコードが以下のように表せるものとします（MySQL では日付が 'YYYY-MM-DD' 形式であると想定しており、普段使用している形式と異なる場合があります）。

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	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	

Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	

ここでは空のテーブルを使用するので、そこにデータを追加する簡単な方法は、ペットごとの行を記述したテキストファイルを作成し、1 つのステートメントでそのファイルの内容をロードします。

ここでは、CREATE TABLE ステートメントで記述したカラムの順番に合わせて、1 行に 1 レコードの形式でタブで区切った値を記述したテキストファイル pet.txt を作成します。値がない場合 (性別がわからない場合や生存中のペットの命日など) 、代わりに NULL 値を使用できます。テキストファイルでそれを表現するには、\N (バックスラッシュと大文字の N) を使用します。たとえば、鳥の Whistler のレコードは以下のようになります。

name	owner	species	sex	birth	death
Whistler	Gwen	bird	\N	1997-12-09	\N

テキストファイル pet.txt を pet テーブルにロードするには、以下のコマンドを使用します。

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

必要に応じて、カラムの区切り文字および改行コードを、LOAD DATA ステートメントで明示的に指定できます。デフォルトはそれぞれタブと LF です。上記のステートメントでテキストファイル pet.txt を正しく読み込むには、デフォルトで十分です。

このステートメントでエラーが発生する場合、使用している MySQL インストールで、LOCAL INFILE 機能がデフォルトで有効になっていないことが考えられます。これを変更する方法については、項4.3.4.「LOAD DATA LOCAL のセキュリティ関連事項」 を参照してください。

新しいレコードを 1 レコードずつ追加する場合は、INSERT ステートメントを使用します。CREATE TABLE ステートメントで記述したカラムの順番に合わせて各カラムの値を指定するのが、最も簡単な形式です。Diane が Puffball という名前のハムスターを新しく飼うことにした場合を考えます。この新しいレコードを追加するには、以下のように INSERT ステートメントを使用します。

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

注意: ここでは、日付と文字列の値は引用符で囲んだ文字列として指定されています。また、INSERT ステートメントでは NULL を直接記述して不明な値を表現できます。この場合は LOAD DATA で使用した \N は使用しません。

この例から、空のテーブルにデータを追加するときに複数の INSERT ステートメントを使用すると、1 つの LOAD DATA ステートメントを使用するよりもはるかに多くの文字を入力する必要があるということがわかります。

3.3.4. テーブルからの情報の取得

テーブルから情報を取得するには、SELECT ステートメントを使用します。このステートメントの一般的な形式を以下に示します。

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

what_to_select は、取得するカラムを示します。ここにはカラムのリストを指定します。* を指定した場合は ``すべてのカラム" を表します。which_table は、データの取得元のテーブルを示します。WHERE 節は省略可能です。指定する場合は、取得対象となるレコードの満たすべき条件を conditions_to_satisfy に指定します。

3.3.4.1. 全データの SELECT

以下に示す SELECT の最も簡単な形式を実行すると、テーブルの全データを取得します。

```
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    | 1979-08-31 | 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       |
+----------+--------+---------+------+------------+------------+
```

この形式の SELECT を使用する例として、初期データをテーブルにロードした直後に、テーブルの全データを確認する場合があります。たとえば、Bowser の誕生日が正しくないのではないかと考えたとします。血統書を調べたところ、正しい生まれ年は 1979 年ではなく 1989 年であることがわかりました。

この場合、データを修正するには、少なくとも 2 つの方法があります。

- テキストファイル pet.txt を編集して誤りを修正し、DELETE と LOAD DATA を使用して、テーブルを空にしてから再ロードする。

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

  ただし、この方法を実行した場合、Puffball のレコードを再入力する必要がある。

- UPDATE ステートメントを使用して、誤りのあるレコードだけを修正する。

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

  UPDATE によって該当レコードだけが変更され、テーブルを再ロードする必要はない。

3.3.4.2. 特定のレコードの SELECT

前のセクションで示したように、テーブル全体を読み取るのは簡単です。SELECT ステートメントで WHERE 節を省略すれば、テーブル全体を取得できます。しかし、通常は、特に大規模なテーブルの場合は、テーブル全体を読み取ることは

ありません。特定の質問に回答するというはっきりした目的があり、そのために必要な情報を取得するための制約を指定するのが普通です。ここで、ペットに関する質問という観点から、その回答を示すいくつかの SELECT クエリについて説明します。

テーブルから特定のレコードだけを選択することができます。たとえば、Bowser の誕生日が変更されたことを確認するには、以下のように Bowser のレコードを選択します。

```
mysql> SELECT * FROM pet WHERE name = "Bowser";
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
```

この出力から、生まれ年が 1979 年から 1989 年に正しく変更されたことが確認できます。

文字列の比較では、通常は大文字と小文字は区別されません。したがって、"bowser" や "BOWSER" などを指定しても、クエリから返される結果は同じです。

条件は、name 以外のカラムにも指定できます。たとえば、1998 年以後に生まれたペットを調べるには、birth カラムに条件を指定します。

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

条件は組み合わせることができます。たとえば、メスの犬を調べるには以下のように条件を指定します。

```
mysql> SELECT * FROM pet WHERE species = "dog" AND sex = "f";
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

上記のクエリでは、AND 論理演算子を使用しています。そのほかに、OR 演算子もあります。

```
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 と OR を混合して指定することができます。ただし、AND が OR よりも高い優先度で処理されます。両方の演算子を混合して使用する場合は、条件の関係を明示的に指定するためにかっこを使用することを推奨します。

```
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. 特定のカラムの SELECT

テーブルのレコード全体を表示する必要がない場合、表示するカラムだけをカンマで区切って指定します。たとえば、ペットの誕生日だけを取得するには、name カラムと birth カラムを選択します。

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

ペットの飼い主を取得するには、以下のクエリを使用します。

```
mysql> SELECT owner FROM pet;
+--------+
| owner  |
+--------+
| Harold |
| Gwen   |
| Harold |
| Benny  |
| Diane  |
| Gwen   |
| Gwen   |
| Benny  |
| Diane  |
+--------+
```

この場合、クエリが各レコードの owner フィールドの値を単に取得しているので、重複している値があることに注意してください。出力するデータの量を最小限にするには、キーワード DISTINCT を追加して一意なレコードを 1 回だけ取得します。

```
mysql> SELECT DISTINCT owner FROM pet;
+--------+
| owner  |
+--------+
| Benny  |
| Diane  |
```

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

WHERE 節を使用してレコードの選択とカラムの選択を組み合わせることができます。たとえば、犬と猫だけの誕生日を取得するには、以下のクエリを使用します。

```
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. レコードのソート

これまでの例では、クエリから返されたレコードは順不同で表示されています。レコードの表示順序に何らかの意味があれば、クエリの出力を調べるのが簡単になる場合がよくあります。クエリの結果をソートするには、ORDER BY 節を使用します。

ペットの誕生日を日付順で取得するには、以下のクエリを使用します。

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

文字型のカラムでは、ソートは、ほかの比較演算子と同じように、大文字と小文字を区別しない方法で行われるのが普通です。これは、大文字と小文字の区別でしか違いを識別できないようなデータを含むカラムでは、順序が定義されないことを意味します。BINARY キャストを使用することで、大文字と小文字を区別してカラムのデータをソートするように指定できます。 ORDER BY BINARY col_name。

デフォルトのソート順序は昇順で、小さい値が大きい値よりも先に表示されます。逆の順序 (降順) でソートするには、ソートするカラムの名前に DESC キーワードを付けます。

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

複数のカラムでソートすることができ、その際、カラムごとに異なるソート順序を指定できます。たとえば、ペットの種類を昇順にソートし、同じ種類の中では誕生日を降順（若いペットから先に表示）でソートするには、以下のクエリを使用します。

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

注意: DESC キーワードはその直前に記述されたカラム（birth）にのみ適用されます。species カラムのソート順序には影響しません。

3.3.4.5. 日付計算

MySQL では、年齢計算や日付の一部の抽出など、日付計算を実行するためのさまざまな関数が用意されています。

それぞれのペットの年齢を取得するには、現在の日付と誕生日の年の部分の差を求め、現在の日付の月日の部分が誕生日の月日の部分よりも暦年で早い場合は、さらにそこから 1 を引くという計算をします。以下のクエリは、それぞれのペットについて、誕生日、現在の日付、および年齢を返します。

```
mysql> SELECT name, birth, CURDATE(),
    -> (YEAR(CURDATE())-YEAR(birth))
    -> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
    -> AS age
    -> FROM pet;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
```

```
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
+----------+------------+------------+------+
```

ここで、YEAR() は日付の年の部分を抽出します。RIGHT() は日付の MM-DD (暦年) を表す右端の 5 文字を抽出します。MM-DD の値を比較する式は 1 または 0 に評価され、それによって CURDATE() が birth よりも暦年で早いと判断される場合は、年の差からさらに 1 を引きます。出力するカラムのラベルは、式全体だと長すぎるので、エイリアス (age) を使用して、わかりやすくします。

このままでもクエリは動作しますが、結果を何らかの順序でソートすれば、その内容を調べやすくなります。それには、ORDER BY name 節を追加して、名前でソートした結果を出力します。

```
mysql> SELECT name, birth, CURDATE(),
    -> (YEAR(CURDATE())-YEAR(birth))
    -> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
    -> AS age
    -> FROM pet ORDER BY name;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
+----------+------------+------------+------+
```

name ではなく、age を使用して出力をソートするには、別の ORDER BY 節を使用します。

```
mysql> SELECT name, birth, CURDATE(),
    -> (YEAR(CURDATE())-YEAR(birth))
    -> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
    -> AS age
    -> FROM pet ORDER BY age;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
+----------+------------+------------+------+
```

同じようなクエリを使用して、死んだペットの死亡時の年齢を取得できます。死んだペットかどうかは、death カラムの値が NULL かどうかを調べて判断します。それが NULL 以外の値であるペットについて、death カラムと birth カラムの値

の差を計算します。

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

このクエリでは、death <> NULL ではなく、death IS NOT NULL を使用しています。これは、NULL が特別な値であり、通常の比較演算子を使用して比較することができないためです。これについては、後で説明します。 See 項3.3.4.6.「NULL 値の使用」。

来月に誕生日を迎えるペットを取得する場合はどうすればよいでしょうか。この場合の計算では、年と日の部分は必要なく、birth カラムの月の部分だけを抽出する必要があります。MySQL では、YEAR()、MONTH()、および DAYOFMONTH() など、日付の一部を抽出する関数が用意されています。ここでは、MONTH() が適しています。その動作を調べるには、birth および MONTH(birth) の両方の値を表示する簡単なクエリを実行します。

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

来月に誕生日を迎えるペットも、同じように簡単に取得できます。現在が 4 月であるとします。その場合、月の値は 4 であり、以下のように指定して、5 月 (5) に生まれたペットを取得します。

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
+-------+------------+
| name  | birth      |
+-------+------------+
| Buffy | 1989-05-13 |
+-------+------------+
```

現在が 12 月の場合は多少複雑になります。12 月 (12) に単純に 1 を足して翌月 (13) に生まれたペットを取得しようとしても、そのような月は存在しないので成功しません。この場合は 1 月 (1) に生まれたペットを取得する必要があります。

現在がどの月であっても動作するクエリを作成することもできます。この場合、クエリで特定の月数を指定する必要はありません。DATE_ADD() を使用すると、指定された日付に期間を加算できます。CURDATE() の値に 1 ヵ月を加算し、その結果から MONTH() を使用して月の部分を抽出すると、ペットの誕生日を調べる月がわかります。

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

現在の月（剰余関数（MOD）を使用して現在が 12 月の場合は 0 にする）に 1 を加算する方法でも同じ結果が得られます。

```
mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MOD(MONTH(CURDATE()), 12) + 1;
```

注意: MONTH は 1 から 12 の範囲の値を返します。また、MOD(something,12) は 0 から 11 の範囲の値を返します。したがって、加算する前に MOD() を実行する必要があります。そうしないと、11 月（11）の次が 1 月（1）になります。

3.3.4.6. NULL 値の使用

NULL 値については、慣れるまでは戸惑う点があるかもしれません。概念的には、NULL は何もない値または不明な値を意味し、ほかの値とは多少異なる扱い方をします。NULL かどうかを調べる場合、=、<、または <> などの算術比較演算子は使用できません。これを自分で試すには、以下のクエリを実行します。

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+----------+-----------+----------+----------+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+----------+-----------+----------+----------+
|     NULL |      NULL |     NULL |     NULL |
+----------+-----------+----------+----------+
```

これらの比較から意味のある結果が得られないことは明らかです。この場合は、IS NULL および IS NOT NULL の各演算子を使用します。

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----------+---------------+
| 1 IS NULL | 1 IS NOT NULL |
+-----------+---------------+
|         0 |             1 |
+-----------+---------------+
```

注意: MySQL では、0 または NULL は false (偽)を意味し、それ以外は true (真)を意味します。ブール値演算では、true を表すデフォルト値は 1 です。

このように NULL は特別な扱い方をするので、前のセクションではペットが死んでいるかどうかを判断する際に death <> NULL ではなく、death IS NOT NULL を使用しました。

GROUP BY では、2 つの NULL 値は等しいとみなされます。

ORDER BY を処理する場合、NULL 値は、ORDER BY ... ASC では先頭に表示され、ORDER BY ... DESC では最後に表示されます。

注意: MySQL 4.0.2 から 4.0.10 では、NULL 値が正しく処理されていないため、ソートが昇順か降順かに関係なく、常に先頭に表示されます。

3.3.4.7. パターンマッチ

MySQL では、標準の SQL のパターンマッチだけでなく、Unix の vi、grep、および sed などのユーティリティで使用され

ているのと同じような、拡張正規表現に基づくパターンマッチも用意されています。

SQL のパターンマッチでは、任意の 1 文字に一致する '_' や、任意の数 (0 文字も含む) の文字に一致する '%' を使用できます。MySQL では、SQL パターンは、デフォルトでは、大文字と小文字が区別されません。ここではいくつかの例を示します。注意: SQL パターンを使用する場合は = または <> ではなく、LIKE または NOT LIKE の比較演算子を使用します。

'b' で始まる名前を取得するには、以下のクエリを実行します。

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

'fy' で終わる名前を取得するには、以下のクエリを実行します。

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

'w' を含む名前を取得するには、以下のクエリを実行します。

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

ちょうど 5 文字の名前を取得するには、'_' パターン文字を 5 回繰り返したパターンを使用します。

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

MySQL が提供するもう 1 つのパターンマッチでは、拡張正規表現を使用します。この種のパターンが一致しているかどうかを調べるには、REGEXP および NOT REGEXP (またはその同義語の RLIKE および NOT RLIKE) を使用します。

拡張正規表現の特徴の一部を以下に示します。

- '.' は任意の 1 文字に一致する。

- 文字クラス '[...]' は、角かっこ内の任意の文字に一致する。たとえば、'[abc]' は 'a'、'b'、または 'c' に一致する。文字の範囲を指定するには、ダッシュを使用する。'[a-z]' は任意のアルファベットに一致し、'[0-9]' は任意の数字に一致する。
- '*' は、その前に記述された 1 文字分の表現の 0 個以上の繰り返しと一致する。たとえば、'x*' は任意の数の 'x' と一致し、'[0-9]*' は任意の数の数字と一致する。'.*' は任意の数の任意の文字と一致する。
- REGEXP パターンマッチは、そのパターンが、照合する値の任意の場所に現れている場合に成功する（LIKE パターンマッチはこれとは違って、そのパターンが、照合する値と完全に一致する場合にのみ成功する）。
- 照合する値の先頭との一致を調べるには、パターンの先頭に '^' を指定し、末尾との一致を調べるには、パターンの末尾に '$' を指定する。

拡張正規表現の動作を調べるために、前出の LIKE を使用したクエリを、ここでは REGEXP を使用するように書き換えています。

'b' で始まる名前を取得するには、名前の先頭と一致するかどうかを照合させるために '^' を指定します。

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

MySQL バージョン 3.23.4 より前のバージョンでは、REGEXP では大文字と小文字が区別されるので、上記のクエリは何も結果を返しません。この場合、大文字または小文字の 'b' のどちらかと照合するには、以下のクエリを使用します。

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

MySQL 3.23.4 以降のバージョンでは、REGEXP の比較で実際に大文字と小文字を区別させるには、BINARY キーワードを使用していずれかの文字列をバイナリ文字列にします。以下のクエリは、小文字の 'b' で始まる名前とだけ一致します。

```
mysql> SELECT * FROM pet WHERE name REGEXP BINARY "^b";
```

'fy' で終わる名前を取得するには、名前の末尾と一致するかどうかを照合させるために '$' を指定します。

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

'w' を含む名前を取得するには、以下のクエリを実行します。

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

正規表現パターンは、値の任意の位置にパターンがあれば一致するので、ワイルドカードをパターンの両側に記述して、SQL パターンを使用する場合のようにパターンを値全体と一致させる必要はありません。

ちょうど 5 文字の名前を取得するには、'^' と '$' を使用してそれぞれ名前の先頭と末尾に一致させ、その間に '.' を 5 回繰り返して指定します。

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

前のクエリで ``n 回繰り返し" 演算子である '{n}' を使用することもできます。

```
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. レコードのカウント

データベースは ``ある種のデータはテーブルに何回出現するか" という質問に答えるためによく使用されます。たとえば、自分の飼っているペットの数や、飼い主ごとに飼っているペットの数を調べたり、ペットに関するさまざまな個体数調査を実行することができます。

飼っているペットの総数をカウントするのは、``pet テーブルには何件のレコードがあるか" という質問と同じです。これは、このテーブルでは 1 匹のペットについて 1 レコードを使用しているためです。COUNT(*) はレコードの数をカウントします。したがって、飼っているペットの総数をカウントするには、以下のようなクエリを使用します。

```
mysql> SELECT COUNT(*) FROM pet;
+----------+
| COUNT(*) |
+----------+
|        9 |
+----------+
```

前に、ペットの飼い主の名前を取得するクエリを使用しました。COUNT() を使用すると、飼い主ごとに飼っているペットの数をカウントできます。

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Benny  |        2 |
```

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

owner ごとのすべてのレコードをグループ化するために GROUP BY を使用していることに注意してください。これを指定しない場合、以下のエラーメッセージが表示されます。

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

COUNT() と GROUP BY を組み合わせると、さまざまな方法でデータを特徴付けることができます。ペットのさまざまな個体数調査を実行する例を以下に示します。

種類ごとのペット数を取得します。

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
+---------+----------+
| species | COUNT(*) |
+---------+----------+
| bird    |       2 |
| cat     |       2 |
| dog     |       3 |
| hamster |       1 |
| snake   |       1 |
+---------+----------+
```

性別ごとのペット数を取得します。

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
+------+----------+
| sex  | COUNT(*) |
+------+----------+
| NULL |       1 |
| f    |       4 |
| m    |       4 |
+------+----------+
```

（この出力では、NULL は性別が不明であることを示す）

種類および性別の組み合わせごとのペット数を取得します。

```
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 |
```

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

COUNT() を使用する場合、テーブル全体を取得する必要はありません。たとえば、前のクエリで犬と猫だけをカウントする場合、以下のクエリを実行します。

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

性別が判明しているペットについて、性別ごとのペット数を取得するには、以下のクエリを実行します。

```
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. 複数テーブルの使用

pet テーブルでは飼っているペットのデータを管理しています。ペットに関するその他の情報、たとえば獣医に通った回数や出産した日など、その一生に発生するイベントなどを記録する場合は、別のテーブルを作成する必要があります。このテーブルはどのような構成でしょうか。このテーブルに必要なカラムを以下に示します。

- 各イベントがどのペットに関連するものかを示すためのペット名
- イベント発生日時
- イベントについての説明
- イベントをカテゴリ別に分類できるようにする場合は、イベントの種類

これらの検討事項を踏まえ、event テーブルの CREATE TABLE ステートメントは以下のようになります。

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
    -> type VARCHAR(15), remark VARCHAR(255));
```

pet テーブルの場合と同じように、情報を記述したタブ区切りのテキストファイルを作成して初期レコードをロードするのが最も簡単です。

name	date	type	remark
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

これらのレコードをロードします。

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

pet テーブルに対して実行したクエリから学習した内容を参考にすれば、event テーブルからもさまざまなデータを取得できます。基本的な考え方は同じです。ただし、event テーブルだけでは質問に回答できない場合があります。それはどのような場合でしょうか。

それぞれのペットが出産したときの年齢を調べる場合を考えます。すでに 2 つの日付から年齢を計算する方法については説明しました。母親の出産日は event テーブルに格納されていますが、その日の母親の年齢を調べるには母親の誕生日が必要です。この誕生日は pet テーブルに格納されています。このことは、クエリで 2 つのテーブルを参照する必要があることを意味します。

```
mysql> SELECT pet.name,
    -> (YEAR(date)-YEAR(birth)) - (RIGHT(date,5)<RIGHT(birth,5)) AS age,
    -> remark
    -> FROM pet, event
    -> WHERE pet.name = event.name AND type = "litter";
+--------+------+-----------------------------+
| name   | age  | remark                      |
+--------+------+-----------------------------+
| Fluffy |    2 | 4 kittens, 3 female, 1 male |
| Buffy  |    4 | 5 puppies, 2 female, 3 male |
| Buffy  |    5 | 3 puppies, 3 female         |
+--------+------+-----------------------------+
```

このクエリについては、注目すべき点がたくさんあります。

- このクエリは 2 つのテーブルから情報を取得する必要があるので、FROM 節にはそれら 2 つのテーブル名を列挙している。

- 複数のテーブルの情報を組み合わせる（結合する）場合、一方のテーブルのレコードともう一方のテーブルのレコードを一致させる方法を指定する必要がある。どちらのテーブルにも name カラムがあるので、これは簡単に指定できる。このクエリでは、WHERE 節で name の値を使用して 2 つのテーブルのレコードを一致させている。
- name カラムはどちらのテーブルにも存在するので、そのカラムを参照する場合はどちらのテーブルのカラムかを指定する必要がある。テーブルを指定するには、カラム名の前にテーブル名を前置する。

結合を実行する場合、必ずしも 2 つの異なるテーブルを使用する必要はありません。テーブルのレコードを同じテーブルのほかのレコードと比較する場合には、テーブルをそれ自体と結合させると便利です。たとえば、ペットの中で繁殖させる組み合わせを探す場合、pet テーブルをそれ自体と結合させて、同種のオスとメスの組み合わせの候補を取得することができます。

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

このクエリでは、カラムを参照するために、テーブル名にエイリアスを指定して、それぞれのカラムの参照が、どちらのテーブルインスタンスと関連付けられているかをはっきりわかるようにしています。

3.4. データベースおよびテーブルに関する情報の取得

データベースやテーブルの名前を忘れた場合、または特定のテーブルの構造（カラム名など）を忘れた場合、どうすればよいでしょうか。MySQL では、サポートするデータベースおよびテーブルの情報を提供するステートメントを多数用意することで、このような問題を解決します。

SHOW DATABASES は、すでに説明したとおり、サーバが管理するデータベースの一覧を表示します。現在選択されているデータベースを取得するには、DATABASE() 関数を使用します。

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| menagerie  |
+------------+
```

まだどのデータベースも選択していない場合は、NULL が返されます（MySQL 4.1.1 より前のバージョンでは空の文字列）。

カレントデータベースに含まれるテーブルを取得するには（テーブル名がわからない場合など）、以下のコマンドを実行します。

```
mysql> SHOW TABLES;
+--------------------+
| Tables in menagerie |
```

```
+--------------------+
| event              |
| pet                |
+--------------------+
```

テーブルの構造を取得するには、DESCRIBE コマンドを使用します。このコマンドは、テーブルの各カラムに関する情報を表示します。

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

Field はカラム名、Type はカラムのデータ型、NULL はカラムに NULL 値を格納できるかどうか、Key はカラムにインデックスが付加されているかどうか、Default はカラムのデフォルト値を、それぞれ示します。

テーブルにインデックスが付加されている場合、SHOW INDEX FROM tbl_name を実行するとその情報を取得できます。

3.5. バッチモードでの mysql の使用

これまでのセクションでは、mysql を対話式に使用して、クエリを入力し、結果を表示させていました。mysql は、バッチモードでも実行できます。バッチモードで実行するには、実行するコマンドをファイルに記述し、そのファイルから入力を読み込むように mysql に指示します。

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

Windows 上で mysql を実行していて、ファイルに含まれる特殊文字によって問題が発生する場合、以下のように指定します。

```
dos> mysql -e "source batch-file"
```

コマンドラインで接続パラメータを指定する必要がある場合、以下のようなコマンドを実行します。

```
shell> mysql -h host -u user -p < batch-file
Enter password: ********
```

バッチモードで mysql を実行する場合、まずスクリプトファイルを作成してからそれを実行します。

スクリプトファイルに含まれるステートメントの一部でエラーが発生した場合も、その後の処理を継続させる場合は、--force コマンドラインオプションを指定する必要があります。

スクリプトを使用する理由を以下に示します。

- クエリを繰り返し実行する場合（たとえば、毎日または毎週）、それをスクリプトにすることによって、実行するたび

に同じクエリを入力する手間を省くことができる。

- スクリプトファイルをコピーして編集することによって、同じような既存のクエリから新しいクエリを作成できる。
- クエリの開発中は、特に、複数行にわたるコマンドまたは複数ステートメントによる一連のコマンドを使用するクエリの場合、バッチモードで実行できると便利である。どこかに間違いがあっても、すべてを再入力する必要はない。スクリプトファイルを編集して間違いを修正し、mysql にそれを実行するように指示するだけでよい。
- 大量のデータを出力するクエリがある場合、画面上でデータが一気にスクロールアップするのを見るのではなく、出力をページ表示コマンドに渡して 1 ページ単位で表示させることができる。

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

- 出力をファイルに取り込んで、後で使用することができる。

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

- 自分で作成したスクリプトを配布することで、他人にも同じコマンドを実行させることができる。
- たとえば cron ジョブからクエリを実行する場合など、対話式で処理できない場合がある。この場合は、バッチモードを使用する必要がある。

mysql をバッチモードで実行する場合、その出力形式は、対話式に実行する場合とは異なります (より簡潔に出力される)。たとえば、mysql を対話式に実行する場合、SELECT DISTINCT species FROM pet の出力は以下のようになります。

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

バッチモードで実行した場合は、以下のように出力されます。

```
species
bird
cat
dog
hamster
snake
```

バッチモードでも対話式と同じ出力形式でデータを取得するには、mysql -t を使用します。実行しているコマンドを出力にエコーするには、mysql -vvv を使用します。

source コマンドを使用すると、mysql プロンプトからでもスクリプトを実行できます。

```
mysql> source filename;
```

3.6. 一般的なクエリの例

ここでは、MySQL に関する一般的な問題を解決する方法の例を示します。

一部の例では、テーブル shop を使用します。このテーブルには、業者 (ディーラー) の物品 (品番) ごとの価格が格納されます。各業者は物品ごとに 1 つの定価を付けているものとし、(article, dealer) がレコードのプライマリキーになります。

コマンドラインツール mysql を呼び出して、データベースを選択します。

```
shell> mysql your-database-name
```

(ほとんどの MySQL インストールで、データベース名 test を使用できます)

以下のステートメントを実行すると、テーブルを作成し、データを追加できます。

```
mysql> 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));
mysql> 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);
```

上記のステートメントを発行した後、テーブルには以下の内容が格納されています。

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

3.6.1. カラムの最大値

``品番が最も大きい物品は ?''

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

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

3.6.2. 特定のカラムの最大値が格納されているレコード

``最も高価な物品の数、業者、および価格は ?''

SQL-99 （ および MySQL バージョン 4.1 以上 ）では、サブクエリを使用すると簡単にできます。

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

MySQL バージョン 4.1 より前のバージョンでは、2 段階に分けて実行します。

1. SELECT ステートメントを使用して、最高値をテーブルから取得する。

```
mysql> SELECT MAX(price) FROM shop;
+------------+
| MAX(price) |
+------------+
|      19.95 |
+------------+
```

2. 上のクエリで表示された値 19.95 を最高値として使用するクエリを作成し、対応するレコードを検索および表示する。

```
mysql> SELECT article, dealer, price
    -> FROM   shop
    -> WHERE  price=19.95;
+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

これ以外に、すべてのレコードを価格の降順でソートし、MySQL 固有の LIMIT 節を使用して先頭のレコードだけを取得する方法もあります。

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

注意:価格が 19.95 の物品が複数ある場合、LIMIT 節を使用した方法では、その中の 1 つしか取得できません。

3.6.3. グループごとのカラムの最大値

``物品ごとの最高値は ?''

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

3.6.4. 特定のフィールドのグループごとの最大値が格納されているレコード

``物品ごとに最高値を付けている業者（複数可）は ?''

SQL-99（および MySQL バージョン 4.1 以降）では、以下に示すようなサブクエリを使用してこの問題を解決できます。

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

MySQL バージョン 4.1 より前のバージョンでは、複数の段階に分けて実行するのが最適です。

1. （物品、最高値）の組み合わせの一覧を取得する。
2. 物品ごとに、最高値を格納しているレコードを取得する。

これは、テンポラリテーブルと結合を使用すると、簡単に実行できます。

```
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;
```

TEMPORARY テーブルを使用しない場合は、tmp テーブルもロックする必要があります。

``これを 1 つのクエリで実行できますか ?''

できます。ただし、MAX と CONCAT を使用した、非効率的な手段を使用する必要があります。

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

この方法は、クライアント側で連結したカラムの分割を実行すると、多少効率良くできます。

3.6.5. ユーザ変数の使用

MySQL ユーザ変数を使用すると、クライアント側で一時変数を使用せずに結果を記憶することができます。 See 項6.1.4.「ユーザ変数」。

たとえば、最高値および最安値が付けられている物品を取得するには、以下のクエリを実行します。

```
mysql> SELECT @min_price:=MIN(price),@max_price:=MAX(price) FROM shop;
mysql> SELECT * FROM shop WHERE price=@min_price OR price=@max_price;
+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

3.6.6. 外部キーの使用

MySQL バージョン 3.23.44 以降では、InnoDB テーブルで外部キーの制約のチェックをサポートしています。See 項7.5.「InnoDB テーブル」。 項1.8.4.5.「外部キー」 も参照してください。

実際には外部キーを使用しなくても 2 つのテーブルを結合できます。InnoDB 以外の種類のテーブルの場合、MySQL が現在実行しないのは、1）CHECK を実行して、使用しているキーがテーブルまたは参照しているテーブルに実際に存在するかどうかを確認すること、2）外部キーが定義されているテーブルから自動的にレコードを削除すること、の 2 つだけです。キーを使用してテーブルを結合すると、何の問題もなく動作します。

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

CREATE TABLE shirt (
    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 person(id),
    PRIMARY KEY (id)
);


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

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


INSERT INTO person VALUES (NULL, 'Lilliana Angelovska');

INSERT INTO shirt 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 person;
+----+--------------------+
| id | name               |
+----+--------------------+
|  1 | Antonio Paz        |
|  2 | Lilliana Angelovska |
+----+--------------------+

SELECT * FROM shirt;
+----+---------+--------+-------+
| 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 person p, shirt 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 |
+----+-------+--------+-------+
```

3.6.7. 2 つのキーを使用した検索

MySQL では、OR で結合された 2 つの異なるキーを使用した検索は、まだ最適化されていません（1 つのキーを複数の OR の部分で使用する検索は非常に最適化が進んでいる）。

```
SELECT field1_index, field2_index FROM test_table
```

```
WHERE field1_index = '1' OR  field2_index = '1'
```

これは、一般的なケースでそのような検索を効率的に処理する方法を見つけだすだけの時間がないからです (一方、AND の処理は現在では完全に一般化され、極めて効率的に動作しています)。

MySQL 4.0 以降では、2 つの異なる SELECT ステートメントの出力を結合する UNION を使用することで、この問題を効率的に解決できます。See 項6.4.1.2.「UNION 構文」。 それぞれの SELECT ステートメントを使用した検索では 1 つのキーだけが使用されるので、それを最適化できます。

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

MySQL 4.0 より前のバージョンでは、TEMPORARY テーブルおよび異なる SELECT ステートメントを使用することで同じ効果を得られます。このような最適化は、SQL サーバが不適切な順序で最適化を実行してしまうような、非常に複雑なクエリを実行する場合にも最適です。

```
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;
```

この解決方法は、実際には 2 つのクエリの UNION と同じです。

3.6.8. 日ごとの訪問数の計算

ビットグループ関数を使用して、あるユーザが Web ページを訪問した月ごとの日数を計算する方法の例を以下に示します。

```
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),(2000,2,23);
```

このテーブルには、ユーザがページを訪問した日付を表す年月日が格納されます。月ごとの訪問日数を取得するには、以下のクエリを実行します。

```
SELECT year,month,BIT_COUNT(BIT_OR(1<<day)) AS days FROM t1
    GROUP BY year,month;
```

以下の結果が表示されます。

```
+------+-------+------+
| year | month | days |
+------+-------+------+
| 2000 |    01 |    3 |
| 2000 |    02 |    2 |
+------+-------+------+
```

このクエリでは、年月の組み合わせに対して異なる日付が何回出現するかを、自動的に重複データを除去することによって計算しています。

3.6.9. AUTO_INCREMENT の使用

AUTO_INCREMENT 属性を使用すると、新しく追加するレコードを識別するための一意な値を生成できます。

```
CREATE TABLE animals (
        id MEDIUMINT NOT NULL AUTO_INCREMENT,
        name CHAR(30) NOT NULL,
        PRIMARY KEY (id)
        );
INSERT INTO animals (name) VALUES ("dog"),("cat"),("penguin"),
                        ("lax"),("whale"),("ostrich");
SELECT * FROM animals;
```

以下の結果が表示されます。

```
+----+---------+
| id | name    |
+----+---------+
|  1 | dog     |
|  2 | cat     |
|  3 | penguin |
|  4 | lax     |
|  5 | whale   |
|  6 | ostrich |
+----+---------+
```

SQL 関数 LAST_INSERT_ID() または C API 関数 mysql_insert_id() を使用すると、最後に生成した AUTO_INCREMENT の値を取得できます。 注意:複数レコードを同時に挿入する場合、 LAST_INSERT_ID()/mysql_insert_id() は、実際には最初に挿入したレコードの AUTO_INCREMENT の値を返します。これにより、レプリケーション設定の場合に、ほかのサーバでも正しく複数行の挿入を再現できます。

MyISAM テーブルと BDB テーブルでは、複合インデックスの2つめのカラムに AUTO_INCREMENT を指定できます。この場合、AUTO_INCREMENT カラムで生成される値は、MAX(auto_increment_column)+1) WHERE prefix=given-prefix として計算されます。これは、データを順序付きのグループに分割する場合に便利です。

```
CREATE TABLE animals (
        grp ENUM('fish','mammal','bird') NOT NULL,
        id MEDIUMINT NOT NULL AUTO_INCREMENT,
        name CHAR(30) NOT NULL,
        PRIMARY KEY (grp,id)
        );
INSERT INTO animals (grp,name) VALUES("mammal","dog"),("mammal","cat"),
            ("bird","penguin"),("fish","lax"),("mammal","whale"),
            ("bird","ostrich");
SELECT * FROM animals ORDER BY grp,id;
```

以下の結果が表示されます。

```
+--------+----+---------+
| grp    | id | name    |
+--------+----+---------+
```

```
| fish   |  1 | lax     |
| mammal |  1 | dog     |
| mammal |  2 | cat     |
| mammal |  3 | whale   |
| bird   |  1 | penguin |
| bird   |  2 | ostrich |
+--------+----+---------+
```

注意: この場合 (AUTO_INCREMENT カラムが複合インデックスの一部として使用されている)、グループ内で最大の AUTO_INCREMENT 値を持つレコードを削除すると、そのグループで同じ AUTO_INCREMENT 値が再使用されることになります。これは MyISAM テーブルの場合にも発生する可能性があります (通常は AUTO_INCREMENT が再使用されることはない)。

3.7. 双生児研究プロジェクトのクエリ

Analytikerna および Lentus で、当社は大規模な研究プロジェクトのシステム業務および実地業務を担当しています。このプロジェクトは、Institute of Environmental Medicine at Karolinska Institutet Stockholm および Section on Clinical Research in Aging and Psychology at the University of Southern California の協同作業として行われています。

このプロジェクトでは、スウェーデン国内の 66 歳以上のすべての双生児に電話でインタビューする予備審査を行います。一定の条件を満足した双生児は、次のステージに進みます。このステージでは、プロジェクトに参加を希望する双生児を、医者と看護師からなるチームが訪問します。そこで、身体検査、神経心理学的検査、臨床試験、神経画像、精神状態評価、および家系収集などの作業が行われます。さらに、医学的および環境的な危険因子に関するデータも収集されます。

双生児研究の詳細については、http://www.mep.ki.se/twinreg/index_en.html を参照してください。

プロジェクトの後半部分は Perl および MySQL を使用して作成した Web インタフェースによって管理されています。

インタビューから得られたデータはすべて、毎晩 MySQL データベースに格納されます。

3.7.1. 未訪問のすべての双生児の検索

以下のクエリは、プロジェクトの後半部分に進む双生児を決定するために使用します。

```
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 sceening 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.birthday)) / 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;
```

クエリの一部について簡単に説明します。

- CONCAT(p1.id, p1.tvab) + 0 AS tvid

  id と tvab を連結した値を、数値としての比較によってソートする。連結した結果に 0 を加算することで、MySQL はその結果を数値として扱う。

- カラム id

  双生児の組を識別する。これはすべてのテーブルでキーとして使用される。

- カラム tvab

  双生児の組の中の個人を識別する。この値は 1 または 2 のどちらかになる。

- カラム ptvab

  tvab の逆の値。tvab が 1 の場合、この値は 2 であり、その逆も同様である。このカラムは、入力量を少なくするためと、MySQL がクエリを最適化しやすくするために定義されている。

このクエリは、特に、同じテーブルを使用した結合（p1 と p2）でテーブルを参照する方法を示しています。この例では、双生児の一方が 65 歳以前に死亡しているかどうかの確認に使用されています。死亡している場合、そのレコードは返されません。

上で説明したカラムは、双生児関連のすべてのテーブルで定義されています。クエリの実行速度を向上するために、id,tvab の両方（すべてのテーブル）および id,ptvab （person_data）をキーとしています。

当社の実稼働マシン（200MHz UltraSPARC）では、このクエリは 150 〜 200 件のレコードを返し、所要時間は 1 秒未満です。

上記のクエリで使用されている各テーブルの現在のレコード数を以下に示します。

テーブル	レコード数
person_data	71074
lentus	5291
twin_project	5286
twin_data	2012
informant_data	663
harmony	381

postal_groups	100

3.7.2. 双生児の組のステータスをまとめた表の作成

インタビューが終わると、event と呼ばれるステータスコードを割り当てます。以下のクエリは、すべての双生児をイベントでグループ化した表を表示します。この表には、双生児の両方にインタビューが終わった組の数、一方にインタビューが終わったがもう一方には断られた数、などが示されます。

```
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. Apache での MySQL の使用

MySQL データベースを使用してユーザを認証し、ログファイルを MySQL のテーブルに書き込むプログラムがあります。

Apache 設定ファイルに以下の設定を追加することで、MySQL に簡単に読み込めるように Apache のログ形式を変更することができます。

```
LogFormat \
      "\"%h\",%{%Y%m%d%H%M%S}t,%>s,\"%b\",\"%{Content-Type}o\",  \
      \"%U\",\"%{Referer}i\",\"%{User-Agent}i\""
```

このフォーマットのログファイルを MySQL にロードするには、以下のようなステートメントを使用します。

```
LOAD DATA INFILE '/local/access_log' INTO TABLE table_name
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
```

LogFormat 行がログファイルに書き込むデータに対応して、指定するテーブルのカラムを作成する必要があります。

第4章 データベース管理

4.1. MySQL のコンフィギャ

4.1.1. mysqld コマンドラインオプション

通常、mysqld オプションはオプション設定ファイルから管理してください。 See 項4.1.2.「my.cnf オプション設定ファイル」。

mysqld と mysqld.server は mysqld グループと server グループからオプションを読み込みます。 mysqld_safe は mysqld、server、mysqld_safe、safe_mysqld の各グループからオプションを読み込みます。 組み込み MySQL サーバは通常、server、embedded、および xxxxx_SERVER からオプションを読み込みます。ここで、xxxxx はアプリケーションの名前です。

mysqld には、多くのコマンドラインオプションがあります。以下、一般的なコマンドラインオプションについて説明します。 完全なオプション一覧を参照するには、mysqld --help を実行してください。 レプリケーション用のオプションについては、別のセクションで説明します。項4.11.6.「レプリケーションスタートアップオプション」 を参照してください。

- --ansi

  MySQL 構文ではなく、SQL-99 構文を使用する。 See 項1.8.2.「ANSI モードでの MySQL の実行」。

- -b, --basedir=path

  インストールディレクトリのパス。すべてのパスは通常、このパスからの相対パスで解決される。

- --big-tables

  すべてのテンポラリセットをファイルに保存することにより、大きな結果セットを可能にする。これによりほとんどの 'table full' エラーが解決されるが、メモリ内テーブルだけで事足りるクエリの実行速度も遅くなる。MySQL バージョン 3.23.2 以降、小さなテンポラリテーブルの場合はメモリを使用し、必要に応じて自動的にディスクテーブルに切り替わるようになっている。

- --bind-address=IP

  バインドする IP アドレス。

- --console

  --log-error が指定されている場合でも、stderr/stdout にエラーログメッセージを書き込む。このオプションを使用した場合、Windows では mysqld によりコンソール画面が開かれたままになる。

- --character-sets-dir=path

  キャラクタセットが格納されているディレクトリ。 See 項4.7.1.「データおよびソート用キャラクタセット」。

- --chroot=path

  起動時に chroot 環境に mysqld デーモンを配置する。MySQL 4.0 以降の推奨セキュリティ設定 (MySQL 3.23 では、完全に閉じた chroot ジェイルを提供できない) 。 ただし、LOAD DATA INFILE と SELECT ... INTO OUTFILE に制約が

加わる。

- --core-file

  mysqld が異常終了した場合に、コアファイルを作成する。システムによっては、mysqld_safe に --core-file-size を指定することも必要になる。 See 項4.8.2.「mysqld_safe (mysqld のラッパ) 」。 注意: --user オプションも使用している場合、Solaris などシステムではコアファイルを作成できない。

- -h, --datadir=path

  データベースルートのパス。

- --debug[...]=

  MySQL を --with-debug でコンフィギャしている場合、このオプションを使用して mysqld の動作を示すトレースファイルを取得できる。 See 項E.1.2.「トレースファイルの作成」。

- --default-character-set=charset

  デフォルトのキャラクタセットを設定する。 See 項4.7.1.「データおよびソート用キャラクタセット」。

- --default-table-type=type

  デフォルトのテーブル型を設定する。 See 章 7. MySQL のテーブル型。

- --delay-key-write[= OFF | ON | ALL]

  MyISAM で、DELAYED KEYS をどのように使用するか指定する。 See 項5.5.2.「サーバパラメータのチューニング」。

- --delay-key-write-for-all-tables (MySQL 4.0.3 では、--delay-key-write=ALL を使用)

  すべての MyISAM テーブルにおいて、書き込み間のキーバッファをフラッシュしない。 See 項5.5.2.「サーバパラメータのチューニング」。

- --des-key-file=filename

  DES_ENCRYPT() および DES_DECRYPT() が使用するデフォルトキーをこのファイルから読み取る。

- --enable-external-locking (以前は --enable-locking)

  システムロックを有効にする。注意: lockd が完全には機能しないシステム (Linux など) でこのオプションを使用すると、mysqld がデッドロックになる可能性が高くなる。

- --enable-named-pipe

  名前付きパイプのサポートを有効化する (Windows NT/2000/XP のみ) 。

- -T, --exit-info

  mysqld サーバのデバッグに使用できる、複数の異なるフラグのビットマスク。完全に理解していない限り、このオプションは使用しないこと。

- --flush

各 SQL コマンド実行後、ディスクへの変更をすべてフラッシュする。通常、MySQL は各 SQL コマンドの後、ディスクへの変更の書き込みだけを行い、ディスクへの同期処理はオペレーティングシステムに任せる。 See 項A.4.1.「MySQL が何度もクラッシュする場合に行うこと」。

- -?, --help

  ショートヘルプを表示して終了する。

- --init-file=file

  起動時に、このファイルから SQL コマンドを読み取る。

- -L, --language=...

  クライアントエラーメッセージに使用する言語。フルパスで指定できる。 See 項4.7.2.「英語以外のエラーメッセージ」。

- -l, --log[=file]

  接続とクエリをログファイルに記録する。 See 項4.10.2.「一般クエリログ」。

- --log-bin=[file]

  データを変更するクエリをすべてログファイルに記録する。バックアップおよびレプリケーション用に使用する。 See 項4.10.4.「バイナリログ」。

- --log-bin-index[=file]

  バイナリログファイル名のインデックスファイル。 See 項4.10.4.「バイナリログ」。

- --log-error[=file]

  エラーメッセージおよびスタートアップメッセージをこのログファイルに記録する。 See 項4.10.1.「エラーログ」。

- --log-isam[=file]

  ISAM および MyISAM の変更をすべてログファイルに記録する（ISAM および MyISAM のデバッグ時のみ使用）。

- --log-long-format

  追加情報をログファイルに記録する（更新ログ、バイナリ更新ログ、スロークエリログなど、有効化されているすべてのログ）。たとえば、クエリのユーザ名とタイムスタンプが記録される。--log-slow-queries と --log-long-format を使用している場合、インデックスを使用しないクエリも、スロークエリログに記録される。 注意: --log-long-format は MySQL バージョン 4.1 で廃止され、--log-short-format が導入されている（long log format がバージョン 4.1 以降のデフォルト設定になった）。また、MySQL 4.1 から、インデックスを使用しないクエリをスロークエリログに記録するための --log-queries-not-using-indexes オプションが利用可能になっている。

- --log-queries-not-using-indexes

  このオプションを --log-slow-queries と一緒に使用すると、インデックスを使用しないクエリも、スロークエリログに記録される。このオプションは MySQL 4.1 で導入された。 See 項4.10.5.「スロークエリログ」。

- --log-short-format

  ログファイル (更新ログ、バイナリ更新ログ、スロークエリログなど、有効化されているすべてのログ) に記録される情報が少なくなる。たとえば、クエリのユーザ名とタイムスタンプが記録されなくなる。このオプションは MySQL 4.1 で導入。

- --log-slow-queries[=file]

  実行時間が long_query_time 秒を超えたクエリをすべてログファイルに記録する。 注意: 記録されるデフォルトの情報量は MySQL 4.1 で変更された。詳細については、--log-long-format オプションおよび --log-long-format オプションの説明を参照のこと。 See 項4.10.5.「スロークエリログ」。

- --log-update[=file]

  指定がなければ、更新を file.# に記録する。# は一意の番号。 See 項4.10.3.「更新ログ」。 更新ログは MySQL 5.0 で廃止されるので、代わりにバイナリログ (--log-bin) を使用すること。 See 項4.10.4.「バイナリログ」。 バージョン 5.0 からは、--log-update を使用すると単にバイナリログが有効になる。

- --low-priority-updates

  テーブル編集操作 (INSERT、DELETE、UPDATE) の優先順位が選択よりも低くなる。{INSERT | REPLACE | UPDATE | DELETE} LOW_PRIORITY ... を使用して 1 つのクエリだけ優先順位を低くしたり、SET LOW_PRIORITY_UPDATES=1 を使用して 1 つのスレッド内での優先順位を変更することもできる。 See 項5.3.2.「テーブルロック関連の問題」。

- --memlock

  メモリ内の mysqld プロセスをロックする。これは、使用しているシステムが mlockall() システムコールをサポートしている場合 (Solaris など) のみで使用可能。オペレーティングシステムが原因で mysqld のスワップが発生するという問題がある場合、その解決に役立つ。 注意: このオプションを使用するには、サーバを root として起動することが必要になるが、これはセキュリティ上好ましくない。

- --myisam-recover [=option[,option...]]

  オプションは、DEFAULT、BACKUP、FORCE、および QUICK の任意の組み合わせ。このオプションを無効にするには、これを明示的に "" に設定する。このオプションを使用すると、mysqld はテーブルを開くときに、テーブルにクラッシュのマークが付いていないか、つまりテーブルが正しく閉じられているかどうかをチェックする (最後のオプションは、--skip-external-locking を使用している場合だけ有効)。テーブルにクラッシュマークが付いていた場合、mysqld はテーブルをチェックする。テーブルが破損していた場合、mysqld は修復を試みる。

  以下のオプションにより、修復方法が決定される。

オプション	説明
DEFAULT	--myisam-recover でどのオプションも指定しないのと同じ。
BACKUP	修復時にデータテーブルが変更された場合、table_name.MYD データファイルのバックアップを table_name-datetime.BAK として保存する。
FORCE	.MYD ファイルから複数のレコードが失われる場合でも修復を実行する。
QUICK	削除ブロックがない場合、テーブル内のレコードをチェックしない。

テーブルを自動的に修復する前に、MySQL はそのことをエラーログに追加する。ユーザの介入なしにほとんどすべての問題をリカバリできるようにするには、オプション BACKUP,FORCE を使用する。これにより、いくつかのレコードが削除される場合でもテーブルの修復が行われるが、古いデータファイルがバックアップとして残るので後で調べることができる。

- --new

  バージョン 4.0.12 から使用可能となったオプション。--new オプションを使用すると、特定の局面でサーバを 4.1 のように動作させることができ、簡単に 4.0 から 4.1 にアップグレードできる。

  - TIMESTAMP が、'YYYY-MM-DD HH:MM:SS' 形式の文字列として返る。 See 項6.2.「カラム型」。

- --pid-file=path

  mysqld_safe によって使用される PID ファイルのパス。

- -P, --port=...

  TCP/IP 接続をリッスンするポート番号。

- -o, --old-protocol

  非常に古いクライアントとの互換性のために 3.20 プロトコルを使用する。 See 項2.5.5.「バージョン 3.20 から 3.21 へのアップグレード」。

- --one-thread

  1 スレッドのみ使用する (Linux でのデバッグ用) 。 このオプションは、サーバのデバッグが有効になっている場合のみ使用可能。 See 項E.1.「MySQL サーバのデバッグ」。

- --open-files-limit=

  mysqld で使用可能なファイル記述子の数を変更できる。 これが設定されていないか、または 0 に設定されている場合、mysqld は、setrlimit() で使用する値を使用してファイル記述子を予約する。この値が 0 の場合、mysqld は max_connections*5 または max_connections + table_cache*2 のいずれか大きい値をファイル数として予約する。mysqld で ' Too many open files ' のエラーが出る場合、この値を大きくする。

- -O, --set-variable=name=value

  変数に値を設定する。--help により変数一覧を表示できる。すべての変数の詳細については、このマニュアルの SHOW VARIABLES セクションを参照のこと。 See 項4.6.8.4.「SHOW VARIABLES」。 これら変数を最適化する方法については、「サーバパラメータのチューニング」セクションを参照のこと。 注意: --set-variable=変数名=値 および -O 変数名=値 構文は、MySQL 4.0 で廃止。代わりに --変数名=値 を使用すること。 See 項5.5.2.「サーバパラメータのチューニング」。

  MySQL 4.0.2 では変数を --変数名=値 で直接設定できるので、set-variable は必要なくなった。

  SET で設定可能なスタートアップオプションの最大値を制限するには、--maximum-variable-name コマンドラインオプションを使用して定義する。 See 項5.5.6.「SET 構文」。

  注意: 変数に値を設定すると、その値が設定可能範囲に収まるように、また使用アルゴリズムに合うように自動的に変数値が修正される。

- --safe-mode

  いくつかの最適化ステージをスキップする。

- --safe-show-database

  このオプションを使用して SHOW DATABASES コマンドを実行すると、そのユーザが何らかの権限を持っているデータベースのみが戻り値として返る。 バージョン 4.0.2 からすべてのユーザが SHOW DATABASES 権限を持つようになったので、このオプションは廃止され、何もしない（このオプションはデフォルトで有効）。 See 項4.4.1.「GRANT および REVOKE の構文」。

- --safe-user-create

  これを有効にした場合、ユーザに mysql.user テーブルあるいはそのテーブル内のカラムへの INSERT 権限がなければ、そのユーザは GRANT コマンドを使用して新規ユーザを作成できない。

- --skip-bdb

  BDB テーブルの使用を無効にする。メモリの節約および演算処理のスピードアップに役立つ。

- --skip-concurrent-insert

  MyISAM テーブルで SELECT と INSERT を同時に実行できなくする（これは、この機能にバグがあると思われる場合だけ使用すること）。

- --skip-delay-key-write

  MySQL 4.0.3 では、代わりに --delay-key-write=OFF を使用する。 すべてのテーブルの DELAY_KEY_WRITE オプションを無視する。 See 項5.5.2.「サーバパラメータのチューニング」。

- --skip-grant-tables

  サーバが一切の権限システムを使用しないようにする。これによりすべての人が、すべてのデータベースにフルアクセスできるようになる（実行中のサーバに権限テーブルを再び使用させるには、mysqladmin flush-privileges または mysqladmin reload を実行する）。

- --skip-host-cache

  IP への名前解決に、ホスト名キャッシュを使用せず、接続ごとに DNS サーバに対しクエリを発行する。 See 項5.5.5.「MySQL の DNS の使用」。

- --skip-innodb

  Innodb テーブルの使用を無効にする。メモリとディスク領域の節約および演算処理のスピードアップに役立つ。

- --skip-external-locking（以前は --skip-locking）

  システムロックを使用しない。isamchk または myisamchk を使用するには、サーバをシャットダウンする必要がある。 See 項1.2.3.「MySQL の安定性」。 注意: MySQL バージョン 3.23 では、REPAIR および CHECK を使用して MyISAM テーブルを修復したりチェックすることができる。

- --skip-name-resolve

ホスト名を解決しない。権限テーブルの Host カラムの値がすべて IP アドレスまたは localhost であることが必要である。 See 項5.5.5.「MySQL の DNS の使用」。

- --skip-networking

  TCP ポートを一切リッスンしない。mysqld とのやり取りはすべて、名前付きパイプまたは Unix ソケットを介して行う必要がある。ローカルからの接続要求のみが許可されているシステムにおいて、特にこのオプションが推奨される。 See 項5.5.5.「MySQL の DNS の使用」。

- --skip-new

  新しくて間違っている可能性のあるルーチンを使用しない。

- --skip-symlink

  4.0.13 で廃止。代わりに --skip-symbolic-links を使用すること。

- --symbolic-links, --skip-symbolic-links

  シンボリックリンクのサポートを有効または無効にする。このオプションは、Windows と Unix では効果が異なる。

  Windows でシンボリックリンクを有効にすると、実際のディレクトリへのパスが含まれる directory.sym ファイルの作成により、データベースディレクトリへのシンボリックリンクを作成できるようになる。 See 項5.6.1.3.「Windows 上のデータベースに対するシンボリックリンクの使用」。

  Unix でシンボリックリンクを有効にすると、CREATE TABLE ステートメントの INDEX DIRECTORY オプションまたは DATA DIRECTORY オプションで MyISAM のインデックスファイルまたはデータファイルを別ディレクトリにリンクできるようになる。テーブルを削除したり名前を変更すると、そのシンボリックリンクがポイントするファイルも削除または名前変更される。

- --skip-safemalloc

  MySQL を --with-debug=full でコンフィギャしていれば、すべてのプログラムが、メモリの割り当て時と解放時に必ずメモリのオーバーランをチェックする。このチェックには時間がかかるため、このチェックを行わないで済むサーバに対しては、--skip-safemalloc オプションによりチェックをスキップできる。

- --skip-show-database

  ユーザが SHOW DATABASES 権限を持っていない場合に、SHOW DATABASES コマンドを無効にする。

- --skip-stack-trace

  スタックトレースを書き込まない。このオプションは、デバッガで mysqld を実行するときに役立つ。システムによっては、コアファイルを取得するためにこのオプションの使用が必要な場合もある。 See 項E.1.「MySQL サーバのデバッグ」。

- --skip-thread-priority

  応答時間を短くするため、スレッド優先度の使用を無効にする。

- --socket=path

Unix では、ローカル接続に使用するソケットファイル（デフォルトでは /tmp/mysql.sock ）。 Windows では、名前付きパイプを使用するローカル接続用パイプ名（デフォルトでは MySQL ）。

- --sql-mode=value[,value[,value...]]

  オプション値として、次の任意の組み合わせを設定できる。 REAL_AS_FLOAT、PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、ONLY_FULL_GROUP_BY、NO_UNSIGNED_SUBTRACTION、NO_AUTO_VALUE_ON_ZERO、NO_TABLE_OPTIONS、NO_FIELD_OPTIONS、NO_KEY_OPTIONS、NO_DIR_IN_CREATE、MYSQL323、MYSQL40、DB2、MAXDB、MSSQL、ORACLE、POSTGRESQL、および ANSI 。 リセットするには、値を空白にする（ --sql-mode="" ）。

  NO_AUTO_VALUE_ON_ZERO は、AUTO_INCREMENT カラムの処理に影響を与える。通常、NULL または 0 のいずれかをカラムに挿入することにより、カラムの次のシーケンス番号を生成する。 NO_AUTO_VALUE_ON_ZERO を指定すると、0 のこの働きが抑制されるため、NULL だけが次のシーケンス番号を生成することになる。このモードは、0 がテーブルの AUTO_INCREMENT カラムに保存されている場合に役に立つ（これは推奨されている方法ではないが）。たとえば、mysqldump でテーブルをダンプしてから再読み込みした場合、MySQL は通常、0 値に遭遇したときに新規シーケンス番号を生成するため、ダンプされたテーブルと再読み込みしたテーブルの内容が異なる結果になる。この場合、ダンプしたファイルを再読み込みする前に NO_AUTO_VALUE_ON_ZERO を有効にするとこの問題が解決する（このオプションが使用可能になった MySQL 4.1.1 以降、mysqldump によるダンプ出力には、自動的に NO_AUTO_VALUE_ON_ZERO を有効にするためのステートメントが含まれている）。

  他のサーバとの互換性のために使用するオプション値もある。 これらを指定することにより、SHOW CREATE TABLE の実行結果から、以前のバージョンの MySQL または他のデータベースサーバが理解できない出力が除外される。 これらのオプション値を使用すると、CREATE TABLE ステートメントの他のサーバへの移植性が高まる。

  - NO_TABLE_OPTIONS、NO_FIELD_OPTIONS、NO_DIR_IN_CREATE、および NO_KEY_OPTIONS を使用すると、テーブルオプションや、カラムまたはインデックス定義に関するオプションが除外される。
  - MYSQL323 と MYSQL40 は、MySQL 3.23 と MySQL 4.0 との互換用。
  - 他のサーバとの互換性を維持するための値は、DB2、MAXDB、MSSQL、ORACLE、および POSTGRESQL。

  mysqldump は SHOW CREATE TABLE を使用して、ダンプ出力に含むテーブル作成ステートメントを取得するため、これらのオプションは mysqldump の出力にも影響する。

  オプション値のいくつかは、値のセットまたはグループの略称なので、影響は複雑になる。 たとえば、--sql-mode=ANSI（または --ansi ）オプションを使用して、サーバに ANSI モードで実行するように命令できる。これは以下のコマンドラインオプションを両方指定するのと同じ。

```
--sql-mode=REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ONLY_FULL_GROUP_BY
--transaction-isolation=SERIALIZABLE
```

  注意: この方法で ANSI モードを指定すると、トランザクション分離レベルを設定することにもなる。 ANSI モードでのサーバの実行については、項1.8.2.「ANSI モードでの MySQL の実行」 を参照のこと。

他の ``グループ" 値は、DB2、MAXDB、MSSQL、ORACLE、および POSTGRESQL。 これらのいずれの値を指定しても、PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、NO_TABLE_OPTIONS、NO_FIELD_OPTIONS、および NO_KEY_OPTIONS 値が有効になる。

--sql-mode オプションは MySQL 3.23.41 で追加された。 NO_UNSIGNED_SUBTRACTION 値は 4.0.0 で追加された。 NO_DIR_IN_CREATE は 4.0.15 で追加された。 NO_AUTO_VALUE_ON_ZERO、NO_TABLE_OPTIONS、NO_FIELD_OPTIONS、NO_KEY_OPTIONS、MYSQL323、MYSQL40、DB2、MAXDB、MSSQL、ORACLE、POSTGRESQL、および ANSI は 4.1.1 で追加された。

- --temp-pool

  このオプションを使用すると、サーバによって作成される大部分のテンポラリファイルが、それぞれ一意の名前ではなく、小さな名前のセットを使用する。これは、多くの新規ファイルが異なる名前で作成され、それを Linux カーネルが処理する問題を回避するためである。Linux では、メモリがディスクキャッシュではなく、ディレクトリエントリキャッシュに割り当てられるため、以前の動作ではメモリの ``リーク" が発生しやすい。

- --transaction-isolation={ READ-UNCOMMITTED | READ-COMMITTED | REPEATABLE-READ | SERIALIZABLE }

  デフォルトのトランザクション分離レベルを設定する。 See 項6.7.6.「SET TRANSACTION 構文」。

- -t, --tmpdir=path

  テンポラリファイルの作成に使用されるディレクトリのパス。テンポラリファイルを保存するには小さすぎるパーティション上にデフォルトの /tmp ディレクトリがある場合、このオプションが役に立つ。 MySQL 4.1 以降、複数のパスの指定が可能になっている。これらのパスはラウンドロビン方式で使用される。Unix ではコロン (':') を、Windows ではセミコロン (';') を使用してパスを区切る。 メモリベースのファイルシステムを指すように tmpdir を設定することは可能。ただし、MySQL サーバがスレーブの場合はできない。スレーブの場合、マシンがリブートしてもテンポラリテーブルのレプリケーションまたは LOAD DATA INFILE のレプリケーション処理を続行するためのテンポラリファイルが必要となる。そのため、マシンのリブートで消去されるメモリベースの tmpdir は適しない。ディスクベースの tmpdir が必要。

- -u, --user={user_name | user_id}

  mysqld サーバを、ユーザ名 user_name またはユーザ ID user_id を持つユーザとして実行する (ここでの ``ユーザ" は、権限テーブルにリストされた MySQL ユーザではなく、システムログインアカウントを指す)。

  このオプションは、mysqld を root アカウントで起動する場合、必須である。 起動シーケンス中にサーバがそのユーザ ID を変更し、root ではなく、その特定のユーザとして実行する。 See 項4.3.2.「MySQL のクラッカー対策」。

  MySQL 3.23.56 および 4.0.12 以降では、 ユーザが --user=root オプションを my.cnf ファイルに追加するという (結果、サーバは root として稼動) セキュリティホールを回避するため、mysqld は指定された最初の --user オプションだけを使用し、複数の --user オプションがあった場合は警告を出力する。/etc/my.cnf および datadir/my.cnf 内のオプションは、コマンドラインオプションの前に処理されるため、--user オプションを /etc/my.cnf に含めて root 以外の値を指定することを推奨する。/etc/my.cnf 内のオプションは他の --user オプションより先に検出され、サーバは確実に root 以外のユーザとして実行され、他の --user オプションが検出されると警告が出力される。

- -V, --version

  バージョン情報を表示して終了する。

- -W, --log-warnings

  Aborted connection... などの警告を .err ファイルに出力する。レプリケーションを使用する場合は、このオプションを有効にすることを推奨する（ネットワークエラーや再接続に関するメッセージなど、現在何が起こっているかについての情報をより多く取得できる）。 See 項A.2.10.「通信エラー/Aborted connection」。

  このオプションは以前の --warnings。

実行中のサーバに対して、ほとんどの値を SET コマンドで変更できます。 See 項5.5.6.「SET 構文」。

4.1.2. my.cnf オプション設定ファイル

MySQL ではバージョン 3.22 以降、サーバとクライアントのデフォルトスタートアップオプションをオプション設定ファイルから読み取ることができるようになっています。

Windows では、MySQL はデフォルトオプションを以下のファイルから読み取ります。

ファイル名	用途
windows-directory\my.ini	グローバルオプション
C:\my.cnf	グローバルオプション

windows-directory は Windows ディレクトリの保存場所です。

Unix では、MySQL はデフォルトオプションを以下のファイルから読み取ります。

ファイル名	用途
/etc/my.cnf	グローバルオプション
DATADIR/my.cnf	サーバ固有オプション
defaults-extra-file	--defaults-extra-file=path で指定されたファイル
~/.my.cnf	ユーザ固有オプション

DATADIR は MySQL データディレクトリです（通常、バイナリインストールの場合は /usr/local/mysql/data、ソースインストールの場合は /usr/local/var）。 注意: これは、コンフィギャ時に指定されたディレクトリです。mysqld 起動時に --datadir で指定したディレクトリではありません。サーバはコマンドラインの引数を処理する前にオプション設定ファイルを探すため、--datadir による指定は、サーバがオプション設定ファイルを探す場所に影響しません。

注意: Windows では、オプション設定ファイル内のすべてのパスを '\' ではなく、'/' で指定してください。'\' は MySQL のエスケープ文字であるため、'\' を使用する場合は 2 回指定する必要があります。

MySQL は、オプション設定ファイルを上記の順序で読み取ろうとします。複数のオプション設定ファイルが存在する場合、後で読み取られたファイルに指定されているオプションの方が、先に読み取られたファイル内の同一オプションより優先されます。コマンドラインで指定されたオプションは、オプション設定ファイルで指定されたオプションよりも優先されます。オプションによっては、環境変数を使用して指定できるものもあります。 コマンドラインまたはオプション設定ファイルで指定されたオプションの方が、環境変数値よりも優先されます。 See 付録 F. 環境変数。

オプション設定ファイルをサポートするプログラムは、mysql、mysqladmin、mysqld、mysqld_safe、mysql.server、

mysqldump、mysqlimport、mysqlshow、mysqlcheck、myisamchk、および myisampack です。

バージョン 4.0.2 より、loose プリフィックスをコマンドラインオプション（または my.cnf のオプション）に使用できます。オプションの前に loose を付けると、オプションが未知の場合でも、それを読み取ったプログラムはエラー終了せず、以下の警告を出力します。

```
shell> mysql --loose-no-such-option
```

MySQL プログラム実行時にコマンドラインで指定できる長いオプションは、オプション設定ファイルで指定できます（ダッシュ2つを前に付けない）。使用可能なオプションの一覧を表示するには、--help オプション付きでそのプログラムを実行してください。

オプション設定ファイルには、以下の形式の行を含めることができます。

- #comment

  コメント行は、'#' または ';' で始める。MySQL 4.0.14 より、コメントを行の途中からでも開始できるようになった。空白行は無視される。

- [group]

  group は、オプションを設定するプログラムまたはグループの名前。グループ行の後の option行または set-variable 行はすべてそのグループに適用される。これは、オプション設定ファイルの最後、または他のグループ行が指定されるまで有効。

- option

  これは、コマンドラインの --option と同等。

- option=value

  これは、コマンドラインの --option=value と同等。注意: オプションの引数にコメント文字が含まれる場合、引数を二重引用符で囲む必要がある。

- set-variable = name=value

  これは、コマンドラインの --set-variable=name=value と同等。 注意: --set-variable は MySQL 4.0 で廃止された。MySQL 4.0 では、プログラム変数名をオプション名として使用できる。コマンドラインでは、--name=value を使用する。オプション設定ファイルでは、name=value を使用する。

[client] グループにより、すべての MySQL クライアント（mysqld ではなく）に適用されるオプションを指定できます。これは、サーバに接続する際に使用するパスワードを指定するための理想的なグループです（ただし、管理者以外のユーザがオプション設定ファイルを読み書きできないようにしてください）。

特定のバージョンの mysqld サーバだけが読み取れるオプションを作成するには、[mysqld-4.0]、[mysqld-4.1] などを使用します。

```
[mysqld-4.0]
new
```

上記の new オプションは、MySQL サーババージョン 4.0.x でのみ使用されます。

注意: オプションおよび値に対して、その前後にある空白は自動的に削除されます。エスケープシーケンス '\b'、'\t'、'\n'、'\r'、'\\'、および '\s' を値文字列に使用することができます ('\s' == blank) 。

次に、一般的なグローバルオプション設定ファイルの例を示します。

```
[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
```

次に、一般的なユーザオプション設定ファイルの例を示します。

```
[client]
# 以下のパスワードは、標準の MySQL クライアント全てに使用されます
password="my_password"

[mysql]
no-auto-rehash
set-variable = connect_timeout=2

[mysqlhotcopy]
interactive-timeout
```

ソースディストリビューションがあれば、my-xxxx.cnf という名前の設定ファイルのサンプルが support-files ディレクトリに含まれています。 バイナリディストリビューションの場合は、DIR/support-files ディレクトリにあります。ここで、DIR は MySQL インストールディレクトリのパスです (通常、C:\mysql または /usr/local/mysql) 。現在、小、中、大、および特大システム用のサンプル設定ファイルが用意されています。my-xxxx.cnf を自分のホームディレクトリにコピーして、名前を .my.cnf に変更し、このファイルを使用してみてください。

オプション設定ファイルをサポートする MySQL プログラムはすべて、以下のオプションをサポートします。

オプション	説明
--no-defaults	オプション設定ファイルを一切読み取らない。
--print-defaults	プログラム名と、取得するすべてのオプションを出力する。
--defaults-file=full-path-to-default-file	指定された設定ファイルだけを使用する。
--defaults-extra-file=full-path-to-default-file	グローバル設定ファイルを読み取った後、ユーザ設定ファイルの前にこの設定ファイルを読み取る。

注意: これらのオプションは、コマンド行の最初に置く必要があります。ただし、--print-defaults だけは、--defaults-file または --defaults-extra-file の直後に置くことができます。

開発者向け注意: オプション設定ファイルの処理としては、コマンドライン引数を処理する前に、オプション設定ファイル内のすべての合致するオプション（該当グループのオプション）が処理されるようになっています。複数回指定されているオプションの最後のインスタンスを使用するプログラムにとっては、この処理で問題ありません。複数回指定されているオプションを処理するが、オプション設定ファイルは読み取らない旧式のプログラムについては、2 行追加するだけでその機能を装備できます。 その方法については、標準 MySQL クライアントのいずれかのソースコードを確認してください。

シェルスクリプトで、my_print_defaults コマンドを使用してオプション設定ファイルを解析することができます。以下の例は、[client] グループと [mysql] グループに属すオプションの表示要求があった場合に、my_print_defaults が生成する可能性のある出力です。

```
shell> my_print_defaults client mysql
--port=3306
--socket=/tmp/mysql.sock
--no-auto-rehash
```

4.2. 同じマシン上で複数の MySQL サーバを実行する

場合によっては、同じマシン上で複数の mysqld サーバを実行することが必要になります。たとえば、既存の稼働環境のままにして、新しい MySQL リリースをテストしたい場合が考えられます。 また、ユーザごとに異なる mysqld サーバへのアクセス権を与える場合などもあります（たとえば、顧客ごとに独立した MySQL インストールを提供するインターネットサービスプロバイダなど）。

単一のマシン上で複数のサーバを実行するには、いくつかのパラメータでサーバ固有の値を設定する必要があります。これらはコマンドラインまたはオプション設定ファイルで設定できます。 項4.1.1.「mysqld コマンドラインオプション」 および 項4.1.2.「my.cnf オプション設定ファイル」 を参照してください。

少なくとも以下のオプションはサーバごとに異なります。

- --port=port_num
- --socket=path
- --shared-memory-base-name=name（Windows のみ、MySQL 4.1 で導入）
- --pid-file=path（Unix のみ）

--port は、TCP/IP 接続のポート番号を制御します。 --socket は、Unix ではソケットファイルパスを、Windows では名前付きパイプの名前を制御します（名前付きパイプ接続をサポートしているサーバに対してのみ、Windows 上で一意のパイプ名を指定する必要があります）。 --shared-memory-base-name は、Windows サーバが使用する共有メモリ名を指定します。これにより、クライアントはその共有メモリを介して接続できるようになります。 --pid-file は、Unix サーバがプロセス ID を書き込むファイルの名前を示します。

以下のオプションを使用する場合、サーバごとに異なる値を設定する必要があります。

- --log=path
- --log-bin=path

- --log-update=path
- --log-error=path
- --log-isam=path
- --bdb-logdir=path

パフォーマンスを高めるには、以下のオプションをサーバごとに個別に設定し、負荷を複数のディスクに分散します。

- --tmpdir=path
- --bdb-tmpdir=path

複数のテンポラリディレクトリを上記のように設定し、どの MySQL サーバにどのテンポラリファイルが属するのかわかりやすくしておくことを推奨します。

一般的に、データディレクトリについても、各サーバが異なるディレクトリを使用するようにします。これは --datadir=path オプションで指定します。

警告: 2 つのサーバから同じデータベースのデータを更新しないようにしてください。使用しているオペレーティングシステムが障害からの保護をおこなうようなシステムロックをサポートしていない場合、予期しない事態が発生する可能性があります。また、複数のサーバが同じデータディレクトリを使用し、ログが有効になっている場合、適切なオプションを使用して各サーバに異なるログファイル名を指定する必要があります。そうしないと、サーバは同じファイルにログしてしまいます。

サーバ間でのデータディレクトリ共有に関するこの警告は、NFS 環境にも当てはまります。NFS 環境で複数の MySQL サーバに同じデータディレクトリへのアクセスを認めることは避けてください。

- 主要な問題は、NFS が速度のボトルネックになること。 NFS はそのような使用を考慮していない。
- 2 つ以上のサーバが互いに干渉しないようにすることも困難。通常、NFS ファイルロックは lockd デーモンによって処理されるが、現在のところ、どのような状況でも 100% の信頼性でロックを実行できるプラットフォームは存在しない。

簡単な方法を選択してください。NFS で複数のサーバにデータディレクトリを共有させるアイデアは良いアイデアではありません。 また、複数の CPU を持つ 1 台のコンピュータを用意し、スレッドを効率的に処理するオペレーティングシステムを使用することを推奨します。

複数の MySQL インストールを異なるロケーションで行う場合、--basedir=path オプションを使用して各サーバに対してベースディレクトリを指定し、各サーバがそれぞれ別のデータディレクトリ、ログファイル、および PID ファイルを使用するようにできます (これらの値のデフォルトは、ベースディレクトリに相対して決定されます) 。その場合、他に指定する必要があるオプションは --socket と --port だけです。たとえば、.tar ファイルバイナリディストリビューションを使用して MySQL の複数のバージョンをインストールするとします。これらは別々のロケーションにインストールされるので、対応するベースディレクトリ以下で ./bin/mysqld_safe コマンドを使用して、各インストールのサーバを起動することができます。 mysqld_safe が、mysqld に渡す適切な --basedir オプションを特定するので、--socket オプションと --port オプションを mysqld_safe に設定するだけで済みます。

以下のセクションで説明するように、環境変数の設定または適切なコマンドラインオプションの指定により、追加サーバを起動することが可能です。ただし、より永続的に複数のサーバを実行する必要がある場合には、オプション設定ファイルを使用して各サーバ固有のオプション値を指定する方法が便利です。

4.2.1. Windows 上で複数のサーバを実行する

適切なパラメータを使用し、コマンドラインからサーバを手動で起動することにより、Windows 上で複数のサーバを実行することができます。Windows NT ベースのシステムでは、複数のサーバを Windows サービスとしてインストールし、Windows サービスとして実行する方法もあります。コマンドラインから、またはサービスとして MySQL サーバを実行する方法についての一般的な説明は、項2.6.1.「Windows の注意事項」 に記載されています。このセクションでは、データディレクトリなど、サーバ固有であることが必要なスタートアップオプション値をサーバごとに設定してサーバを起動する方法について説明します（これらのオプションについては、項4.2.「同じマシン上で複数の MySQL サーバを実行する」 を参照してください）。

4.2.1.1. コマンドラインから複数の Windows サーバを起動する

コマンドラインから手動で複数のサーバを起動するには、コマンドラインまたはオプション設定ファイルで必要なオプションを指定します。オプション設定ファイルでオプションを設定する方が便利ですが、各サーバに独自のオプションセットを確実に設定する必要があります。これを行うには各サーバ用のオプション設定ファイルを作成し、サーバ起動時に --defaults-file を使用してそのサーバにファイル名を指定します。

たとえば、mysqld を、データディレクトリ C:\mydata1 を使用してポート 3307 で実行し、mysqld-max を、データディレクトリ C:\mydata2 を使用してポート 3308 で実行するとします。このためには、2 つのオプション設定ファイルを作成します。たとえば、以下のような C:\my-opts1.cnf ファイルを作成します。

```
[mysqld]
datadir = C:/mydata1
port = 3307
```

次に、以下のような 2 つ目のファイル C:\my-opts2.cnf を作成します。

```
[mysqld]
datadir = C:/mydata2
port = 3308
```

そして、それぞれのオプション設定ファイルを使って各サーバを起動します。

```
shell> mysqld --defaults-file=C:\my-opts1.cnf
shell> mysqld-max --defaults-file=C:\my-opts2.cnf
```

Windows NT では、サーバがフォアグラウンドで起動するため、2 つのコマンドを別々のコンソールウィンドウで実行する必要があります。

サーバをシャットダウンするには、該当するポート番号に接続する必要があります。

```
shell> mysqladmin --port=3307 shutdown
shell> mysqladmin --port=3308 shutdown
```

この例のように設定されているサーバは、クライアントに対して TCP/IP 接続を許可します。名前付きパイプ接続も可能にするには、mysqld-nt サーバまたは mysqld-max-nt サーバを使用し、名前付きパイプを有効にするオプションを指定し

、その名前を指定します（名前付きパイプ接続をサポートするサーバは、それぞれ固有のパイプ名を使用することが必要です）。たとえば、C:\my-opts1.cnf ファイルを以下のように修正します。

```
[mysqld]
datadir = C:/mydata1
port = 3307
enable-named-pipe
socket = mypipe1
```

そして、サーバを次のように起動します。

```
shell> mysqld-nt --defaults-file=C:\my-opts1.cnf
```

2 つ目のサーバで使用する C:\my-opts2.cnf も同様に修正します。

4.2.1.2. 複数の Windows サーバをサービスとして起動する

Windows NT ベースのシステムでは、MySQL サーバを Windows サービスとして実行できます。単一の MySQL サービスのインストール、制御、および削除の手順については、項2.1.1.7.「Windows NT、2000、または XP での MySQL の起動」を参照してください。

MySQL 4.0.2 より、複数のサーバをサービスとしてインストールできるようになっています。 その場合、各サーバが異なるサービス名を使用することが必要です。また、サーバごとに一意であることが必要なその他のパラメータにも注意が必要です。

以下の手順は、C:\mysql-4.0.8 および C:\mysql-4.0.17 にインストールされた 2 つの異なる MySQL バージョンから mysqld-nt サーバを実行する場合を想定しています（たとえば、本稼働サーバとして 4.0.8 を実行しているときに、アップグレード前に 4.0.17 をテストする場合など）。

--install オプションで MySQL サービスをインストールする場合、以下の原則が当てはまります。

- サービス名を指定しないと、サーバは MySQL のデフォルトサービス名を使用し、標準オプション設定ファイルの [mysqld] グループからオプションを読み取る。
- --install オプションの後でサービス名を指定すると、サーバは [mysqld] オプショングループを無視し、サービスと同じ名前のグループからオプションを読み取る。サーバは、標準オプション設定ファイルからオプションを読み取る。
- サービス名の後に --defaults-file オプションを指定すると、サーバは標準オプション設定ファイルを無視し、指定ファイルの [mysqld] グループからのみオプションを読み取る。

これらの原則は、--install-manual オプションを使用してサーバをインストールした場合にも当てはまります。

以上の情報に基づいて、複数のサービスをセットアップする方法はいくつかあります。 次に、いくつかの例を示します。いずれを試す場合でも、最初に既存の MySQL サービスをシャットダウンして削除しておくことが必要です。

- 標準オプション設定ファイルの 1 つで、すべてのサービスにオプションを指定する。 これを行う場合は、各サーバに異なる名前を使用する。 たとえば、4.0.8 mysqld-nt を mysqld1 のサービス名で実行し、4.0.17 mysqld-nt を mysqld2 のサービス名で実行する場合、 4.0.8 には [mysqld1] グループを、4.0.17 には [mysqld2] グループを使用できる。 たとえば、以下のように C:\my.cnf をセットアップできる。

```
# options for mysqld1 service
[mysqld1]
basedir = C:/mysql-4.0.8
port = 3307
enable-named-pipe
socket = mypipe1

# options for mysqld2 service
[mysqld2]
basedir = C:/mysql-4.0.17
port = 3308
enable-named-pipe
socket = mypipe2
```

Windows が各サービス用の正しい実行可能プログラムを登録できるように、サーバのフルパス名を指定して、以下のようにサービスをインストールする。

```
shell> C:\mysql-4.0.8\bin\mysqld-nt --install mysqld1
shell> C:\mysql-4.0.17\bin\mysqld-nt --install mysqld2
```

サービスを開始するには、サービスマネージャを使用するか、適切なサービス名で NET START を使用する。

```
shell> NET START mysqld1
shell> NET START mysqld2
```

サービスを停止するには、サービスマネージャを使用するか、適切なサービス名で NET STOP を使用する。

```
shell> NET STOP mysqld1
shell> NET STOP mysqld2
```

注意: MySQL 4.0.17 より前のバージョンでは、デフォルトサービス名 (MySQL) を使用してインストールされたサーバか、mysqld のサービス名で明示的にインストールされたサーバだけが、標準オプション設定ファイルの [mysqld] グループを読み取ります。4.0.17 以降では、他のサービス名を使用してインストールされたサーバでも、標準オプション設定ファイルを読み取るサーバであればすべて、[mysqld] グループを読み取ります。これにより、すべての MySQL サービスに適用するオプション用に [mysqld] グループを使用し、各サービスの名前が付けられたオプショングループを、そのサービス名でインストールされたサーバ用に使用できます。

- 個別のファイルで各サーバにオプションを指定し、サービスのインストール時に --defaults-file を使用して、各サーバにどのファイルを使用するか指定する。このとき、各ファイルに [mysqld] グループを使用してオプションをリストアップすることが必要。

  この方法で 4.0.8 mysqld-nt のオプションを指定するには、以下のような C:\my-opts1.cnf ファイルを作成する。

  ```
  [mysqld]
  basedir = C:/mysql-4.0.8
  port = 3307
  enable-named-pipe
  socket = mypipe1
  ```

  4.0.17 mysqld-nt に対しては、以下のような C:\my-opts2.cnf ファイルを作成する。

  ```
  [mysqld]
  ```

```
basedir = C:/mysql-4.0.17
port = 3308
enable-named-pipe
socket = mypipe2
```

以下のようにサービスをインストールする (各コマンドは 1 行で入力) 。

```
shell> C:\mysql-4.0.8\bin\mysqld-nt --install mysqld1
        --defaults-file=C:\my-opts1.cnf
shell> C:\mysql-4.0.17\bin\mysqld-nt --install mysqld2
        --defaults-file=C:\my-opts2.cnf
```

MySQL サーバをサービスとしてインストールする際に --defaults-file オプションを使用するには、オプションの前にサービス名を付ける必要がある。

サービスのインストール後、起動と停止は前の例と同じように行う。

複数のサービスを削除するには、mysqld --remove を使用して 1 つずつ削除します。削除するサービスがデフォルト名でない場合は、--remove オプションの後にサービス名を指定します。

4.2.2. Unix 上で複数のサーバを実行する

複数のサーバを Unix 上で実行する最も簡単な方法は、異なる TCP/IP ポートとソケットファイルを使用するようにしてサーバをコンパイルし、それぞれが別のネットワークインタフェースで接続するようにすることです。また、各インストールを別々のベースディレクトリでコンパイルすることにより、データディレクトリ、ログファイル、および PID ファイルの場所をサーバ別にすることができます。

既存サーバがデフォルトのポート番号とソケットファイルでコンフィギャされているとします。新しいサーバを別のパラメータでコンフィギャするには、以下のように configure コマンドを使用します。

```
shell> ./configure --with-tcp-port=port_number \
         --with-unix-socket-path=file_name \
         --prefix=/usr/local/mysql-4.0.17
```

ここで、port_number と file_name に、デフォルト以外のポート番号とソケットファイルのパス名を指定します。および、--prefix の値も、既存の MySQL インストールの場所とは別のインストールディレクトリを指定します。

MySQL サーバが特定のポート番号をリッスンしている場合、以下のコマンドを使用して、ベースディレクトリやソケット名など、重要な変数の値を確認できます。

```
shell> mysqladmin --host=host_name --port=port_number variables
```

このコマンドで表示される情報により、追加サーバを設定するときにどのオプション値を使用してはいけないかがわかります。

注意: ``localhost'' をホスト名として指定すると、mysqladmin は TCP/IP ではなく、Unix ソケット接続をデフォルト使用します。 MySQL 4.1 では、--protocol={TCP | SOCKET | PIPE | MEMORY} オプションを使用して、使用する接続プロトコルを明示的に指定できます。

別のソケットファイルおよび TCP/IP ポート番号で起動するために、新たに MySQL サーバをコンパイルする必要はありま

せん。実行時に値を指定することも可能です。これを行う 1 つの方法として、コマンドラインオプションの使用があります。

```
shell> /path/to/mysqld_safe --socket=file_name --port=port_number
```

2 番目のサーバに別のデータベースディレクトリを使用するため、--datadir=path オプションを mysqld_safe に渡します。

同様の効果を得る別の方法として、環境変数を使用してソケット名とポート番号を設定することもできます。

```
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/mysqld_safe &
```

この方法を使用すると、テスト用に 2 つ目のサーバをすばやく起動できます。この方法の長所は、上記のシェルから呼び出したどのクライアントプログラムにもこの環境変数が適用される点です。そのため、これらのクライアントの接続は自動的に 2 つ目のサーバにダイレクトされます。

付録 F. 環境変数 に、mysqld を制御する他の環境変数の一覧が記載されています。

自動サーバ実行では、ブート時に実行されるスタートアップスクリプトにより、サーバごとに以下のコマンドが 1 回ずつ実行される必要があります。その際、各コマンドに対して適切なオプション設定ファイルのパスを設定します。

```
mysqld_safe --defaults-file=path-to-option-file
```

各オプション設定ファイルには、特定のサーバに固有のオプション値が含まれていることが必要です。

Unix では、mysqld_multi スクリプトを使用して複数のサーバを開始することもできます。 See 項4.8.3.「mysqld_multi (複数の MySQL サーバを管理するプログラム) 」。

4.2.3. 複数サーバ環境でクライアントプログラムを使用する

クライアントにコンパイルされたネットワークインタフェース以外のネットワークインタフェースをリッスンしている MySQL サーバにそのクライアントから接続するには、以下のいずれかの方法を実行します。

- TCP/IP 経由でリモートホストに接続する場合は --host=host_name --port=port_number でクライアントを開始する。Unix ソケットまたは Windows の名前付きパイプを使用してローカルホストに接続する場合は --host=localhost --socket=file_name でクライアントを開始する。
- MySQL 4.1 では、TCP/IP 経由で接続する場合は --protocol=tcp を、Unix ソケットを使用する場合は --protocol=socket を、名前付きパイプを使用する場合は --protocol=pipe を、共有メモリを使用する場合には --protocol=memory を指定してクライアントを開始する。TCP/IP 接続では、--host オプションと --port オプションを指定することが必要な場合もある。他の接続タイプでは、場合によっては、--socket オプションでソケットまたは名前付きパイプ名を指定したり、--shared-memory-base-name オプションで共有メモリ名を指定することが必要になる。
- Unix では、クライアントを起動する前に、MYSQL_UNIX_PORT 環境変数と MYSQL_TCP_PORT 環境変数を設定して Unix ソケットおよび TCP/IP ポート番号を指定する。 特定のソケットまたはポートを永続的に使用する場合、これらの環境変数を設定するコマンドを .login ファイルに置いて、ログインのたびにこれらの環境変数が適用されるようにできる。 See 付録 F. 環境変数。

- オプション設定ファイルの [client] グループにデフォルトソケットと TCP/IP ポートを指定する。たとえば、Windows では C:\my.cnf を、Unix ではホームディレクトリの .my.cnf ファイルを使用できる。 See 項4.1.2.「my.cnf オプション設定ファイル」。
- C プログラムでは、ポートまたはソケット引数を mysql_real_connect() の呼び出しで指定できる。また、mysql_options() を呼び出して、プログラムにオプション設定ファイルを読み取らせることもできる。 See 項11.1.3.「C API 関数の説明」。
- Perl DBD::mysql モジュールを使用している場合、MySQL オプション設定ファイルからオプションを読み取ることができる。次に例を示す。

```
$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);
```

  See 項11.5.2.「DBI インタフェース」。

4.3. 一般的なセキュリティ関連事項と MySQL アクセス制御システム

MySQL には、非標準ですが、上級のセキュリティおよび権限システムがあります。このセクションでは、その機能について説明します。

4.3.1. 一般的なセキュリティガイドライン

インターネットに接続されたコンピュータで MySQL を使用するユーザはすべて、このセクションを読んで、頻発しているセキュリティ侵害を回避するようにしてください。

盗聴傍受、改ざん、再生、サービス妨害など、あらゆるタイプの攻撃に対して、MySQL サーバだけでなく、サーバホスト全体を、最大限の努力をもって保護することが必要です。ここでは、可用性および耐障害性のすべてについてはカバーしていません。

MySQL では、すべての接続、クエリなど、ユーザが実行する可能性のある操作に対して、そのセキュリティを、アクセス制御リスト（ACL）をベースにして保護しています。MySQL クライアントとサーバ間の SSL 暗号化接続もサポートしています。ここで説明する概念の多くは、MySQL 固有ではありません。ほとんどすべてのアプリケーションに当てはまります。

MySQL を実行する際には、可能な限り以下のガイドラインに従ってください。

- mysql root ユーザ以外のだれにも mysql データベースの user テーブルへのアクセス権を与えないこと。これは非常に重要である。 MySQL では、暗号化されたパスワードが実際のパスワードである。 user テーブル内のリストに含まれているパスワードを知り得、そのアカウントのリストに含まれているホストに対するアクセス権を持つ者はだれでも、簡単にそのユーザとしてログインできる。
- MySQL アクセス権限システムを理解すること。MySQL へのアクセスを制御するには GRANT コマンドと REVOKE コマンドを使用する。必要以上の権限をユーザに設定しないこと。すべてのホストに対する権限は決して与えないこと。

  チェックリスト

  - mysql -u root を試す。パスワードなしでサーバに正常に接続できるようだと問題がある。この場合、すべての権限

を持つ root ユーザとして、MySQL サーバに接続できるということである。 特に root パスワードの設定に関する項目に注意して、MySQL インストール手順を見直すこと。

  - SHOW GRANTS コマンドを使用して、だれが何にアクセスできるかをチェックする。REVOKE コマンドを使用して不要な権限を削除する。

- データベースには、テキスト形式のパスワードを保存しないこと。コンピュータがクラックされたとき、パスワードの完全なリストが奪われて不正使用される結果になる。MD5()、SHA1()、または別の一方向ハッシング関数を使用する。

- 辞書を使用してパスワードを選択しない。特殊プログラムで解読されてしまう。``xfish98" のようなパスワードも良くない。標準 QWERTY キーボードで ``fish" を 1 列左でタイプした ``duag98" などを推奨。また、``Mary had a little lamb" の頭文字を取って ``Mhall" とするのも良い。 この方法だと覚えやすく、また部外者が類推することも困難。

- ファイアウォールに投資する。これにより、少なくとも 50% の攻撃を防ぐことができる。MySQL をファイアウォール内つまり非武装地帯に配置する。

  チェックリスト

  - nmap などのツールを使用して、インターネットから自分のポートをスキャンしてみる。MySQL はデフォルトでポート 3306 を使用する。このポートは、信頼されていないホストからアクセスできてはいけない。他にも、MySQL ポートが開いているかどうか確かめる簡単な方法として、リモートコンピュータから以下のコマンドを実行するという方法がある。ここで、server_host は MySQL サーバのホスト名である。

    ```
    shell> telnet server_host 3306
    ```

    接続できて意味不明な文字が返ればポートが開いている。開いておく正当な理由がない限り、ファイアウォールまたはルータで閉じておく。telnet がハングするか、接続が拒否されれば正常である。つまり、ポートは閉じている。

- ユーザが入力するデータを信頼しないこと。Web フォーム、URL、その他のアプリケーションから特殊文字または エスケープ文字シーケンスが入力され、不正操作が行われる可能性がある。ユーザが ``; DROP DATABASE mysql;" などのような入力をしても、アプリケーションが安全に保たれるようにしなければならない。これは極端な例であるが予防策を講じておかないと、クラッカーによる同様の手段によって、重大なセキュリティリークやデータの紛失が発生する可能性がある。

  また、数値データもチェックすること。文字列だけを保護するのは、よくあるミスである。公開されているデータのみのデータベースは保護する必要がないという考えもよくある考えだが、これも間違っている。そのようなデータベースにも、サービス妨害タイプの攻撃が可能である。このタイプの攻撃を防ぐ最もシンプルな方法は、数値定数をアポストロフィで囲むこと。SELECT * FROM table WHERE ID=234 ではなく、SELECT * FROM table WHERE ID='234' のようにする。 MySQL は自動的のこの文字列を数値に変換し、非数値記号を削除する。

  チェックリスト

  - すべての Web アプリケーション

    - すべての Web フォームで ‘’’ および ‘”’ を入力してみる。何らかの MySQL エラーが発生した場合は、すぐにその問題を調べる。

    - URL で %22 (‘”’)、%23 (‘#’)、および %27 (‘’’) を追加して動的 URL の修正を試みる。

    - 前の例の文字を含む動的 URL のデータ型を数値型から文字型に変更する。このような攻撃があってもアプリケーションが影響されないようにする。
    - 数値フィールドに数値ではなく、文字、スペース、および特殊記号を入力してみる。MySQL に渡すことなくアプリケーションがそれらの値を削除するか、エラーを生成することが必要である。値がチェックされずに MySQL に渡ると非常に危険である。
    - MySQL に渡す前にデータサイズをチェックする。
    - 管理目的で使用するユーザ名とは別のユーザ名で、アプリケーションをデータベースに接続させる。アプリケーションに、必要以上のアクセス権を設定しない。
  - PHP のユーザ
    - addslashes() 関数をチェックする。 PHP 4.0.3 より、mysql_escape_string() 関数が利用可能になっている。これは、MySQL C API の同じ名前の関数をベースにしている。
  - MySQL C API のユーザ
    - mysql_real_escape_string() API 呼び出しをチェックする。
  - MySQL++ のユーザ
    - クエリストリームの escape および quote の修飾子をチェックする。
  - Perl DBI のユーザ
    - quote() メソッドをチェックするか、プレースホルダを使用する。
  - Java JDBC のユーザ
    - PreparedStatement オブジェクトおよびプレースホルダを使用する。
- インターネット上で暗号化されていないデータを送信しないこと。これらのデータは、悪意のある第三者によって盗聴され、悪用される可能性がある。SSL や SSH などの暗号化プロトコルを使用する。MySQL では、バージョン 4.0.0 より内部 SSL 接続をサポートしている。 SSH ポート転送を使用して、暗号化 (および圧縮) された通信トンネルを作成できる。
- tcpdump ユーティリティおよび strings ユーティリティの使用法を理解すること。ほとんどの場合、以下のようなコマンドで、MySQL データストリームが暗号化されているかどうかをチェックできる。

```
shell> tcpdump -l -i eth0 -w - src or dst port 3306 | strings
```

  これは、Linux システムで有効。他のシステムでも、少し修正するだけで使用できる。 警告: データを見ることができなくても、必ずしも実際に暗号化されているとは限らない。高度なセキュリティが必要な場合は、専門家に相談すること。

4.3.2. MySQL のクラッカー対策

MySQL サーバに接続するとき、通常はパスワードを使用します。パスワードはテキスト形式で送信されるわけではありま

せんが、暗号化アルゴリズムはそれほど強力なものではありません。クライアントとサーバ間のトラフィックを盗聴できれば、クラッカーは少しの努力でパスワードを探り当てることができます。クライアントとサーバ間の接続が信頼されていないネットワークを通る場合には、SSH トンネルを使用して通信を暗号化してください。

その他の情報はすべてテキストとして転送されるので、その接続を見ることができる人はだれでもそれらの情報を読むことができます。これに不安を感じる場合は、圧縮プロトコル (MySQL バージョン 3.22 以降) を使用してトラフィックの解読をより困難にすることができます。セキュリティをさらに高めるには、ssh を使用してください。オープンソース の ssh クライアントは http://www.openssh.org/ に、商用 ssh クライアントは http://www.ssh.com/ にあります。これを使用すると、MySQL サーバと MySQL クライアント間で暗号化 TCP/IP 接続を利用できます。

MySQL 4.0 を使用している場合、内部 OpenSSL サポートも利用できます。 See 項4.4.10.「安全な接続の使用」。

MySQL システムのセキュリティを確保するためには、以下の事項を強く推奨します。

- すべての MySQL ユーザにパスワードを使用する。other_user にパスワードが設定されていなければ、だれでも mysql -u other_user db_name として簡単に他人になりすましてログインできる。クライアント/サーバ型のアプリケーションでは、クライアント側で任意のユーザ名を指定できるのが一般的。mysql_install_db スクリプトを実行前に編集することにより、すべてのユーザのパスワードを変更できる。また、MySQL root ユーザのパスワードのみ変更するには、以下のように行う。

```
shell> mysql -u root mysql
mysql> UPDATE user SET Password=PASSWORD('new_password')
    ->           WHERE user='root';
mysql> FLUSH PRIVILEGES;
```

- MySQL デーモンを Unix root アカウントで実行しないこと。このことは、FILE 権限のあるユーザであればだれでも root (たとえば ~root/.bashrc) としてファイルを作成できてしまうので、非常に危険である。これを防ぐため、--user=root オプションを使用して直接指定された場合を除き、mysqld は root として実行することを拒否するようになっている。

  mysqld は、普通の権限なしユーザとして実行できる。 新しい Unix アカウント mysql を作成してさらにセキュリティを高めることもできる。別の Unix アカウントで mysqld を実行する場合、user テーブル内の root ユーザ名を変更する必要はない。MySQL ユーザ名と Unix アカウント名はお互い関係ない。別の Unix アカウントで mysqld を開始するには、サーバのデータディレクトリにある my.cnf または /etc/my.cnf のオプション設定ファイルの [mysqld] グループに、アカウント名を指定する user 行を追加する。次に例を示す。

```
[mysqld]
user=mysql
```

  これで、サーバを手動で起動した場合も、mysqld_safe または mysql.server を使用して起動した場合でも、指定のアカウントでサーバが起動する。 詳細については、項A.3.2.「一般ユーザで MySQL を実行する方法」 を参照のこと。

- テーブルへのシンボリックリンクをサポートしない (これは --skip-symlink オプションで無効にできる) 。このことは、root で mysqld を実行する場合、mysqld データディレクトリへの書き込み権限があるすべてのユーザが、システムのすべてのファイルを削除できることになるので、特に重要である。 See 項5.6.1.2.「Unix 上のテーブルに対するシンボリックリンクの使用」。

- mysqld を実行する Unix アカウントだけに、データベースディレクトリの読み取り権限と書き込み権限があることを確認する。

- PROCESS 権限をすべてのユーザには与えない。mysqladmin processlist の出力には、現在実行中のクエリのテキストが表示される。そのため、このコマンドを実行できるユーザは、UPDATE user SET password=PASSWORD('not_secure') クエリを実行する他のユーザを特定できる可能性がある。

  mysqld は、PROCESS 権限を持つユーザ用に特別接続枠を予約しているので、すべての通常接続が使用中の場合でも、MySQL root ユーザはログインしてサーバの状態をチェックできる。

- FILE 権限をすべてのユーザには与えない。この権限を持つユーザは、mysqld デーモンの権限でファイルシステムのどの場所にでもファイルを書き込める。安全対策として、SELECT ... INTO OUTFILE で生成されるファイルについてはだれでも書き込み可能だが、既存のファイルには書き込めないようになっている。

  FILE 権限は、サーバを実行している Unix ユーザがアクセスできるすべての読み取り可能ファイルを読む場合にも使用される。また、ユーザが何らかの権限を持っているカレントデータベースに任意のファイルを読み込むことができる。これは、不正使用される可能性がある。たとえば、LOAD DATA を使用して /etc/passwd をテーブルにロードし、SELECT で読むことができる。

- DNS を信頼しない場合、権限テーブルで、ホスト名ではなく IP アドレスを使用すること。いずれにしても、ワイルドカードを含むホスト名を使用して権限テーブルを作成する際には特に注意が必要である。

- 単一ユーザの接続数を制限する場合は、mysqld で max_user_connections 変数を設定する。

4.3.3. セキュリティ関連の mysqld スタートアップオプション

セキュリティに影響する mysqld オプションは以下のとおりです。

- --local-infile[=(0|1)]

  --local-infile=0 を使用すると、LOAD DATA LOCAL INFILE を使用できなくなる。

- --safe-show-database

  このオプションを使用すると、SHOW DATABASES コマンドで、そのユーザが何らかの権限を持っているデータベースのみが返される。 現在は、SHOW DATABASES 権限が存在するため、バージョン 4.0.2 以降、このオプションは廃止されており、何もしない (デフォルトで有効になっている) 。 See 項4.4.1.「GRANT および REVOKE の構文」。

- --safe-user-create

  このオプションが有効になっている場合、ユーザに mysql.user テーブルへの INSERT 権限がなければ、そのユーザは GRANT コマンドを使用して新規ユーザを作成できない。ユーザが事前設定された権限で新規ユーザを作成できるようにするには、そのユーザに以下の権限を設定することが必要である。

```
mysql> GRANT INSERT(user) ON mysql.user TO 'user'@'hostname';
```

  これで、ユーザは権限カラムを直接変更できないが、GRANT コマンドを使用して他のユーザに権限を与えることができるようになる。

- --skip-grant-tables

  サーバが権限システムをまったく使用しないようにする。これによりすべての人が、すべてのデータベースにフルアク

セスできるようになる。実行中のサーバに権限テーブルを再び使用させるためには、mysqladmin flush-privileges 、mysqladmin reload または FLUSH PRIVILEGES; を実行すればよい。

- --skip-name-resolve

  ホスト名を解決しない。権限テーブルの Host カラム値はすべて IP アドレスか localhost でなければならない。

- --skip-networking

  TCP/IP 経由の接続を認めない。mysqld への接続をすべて Unix ソケットで行う。 このオプションは、3.23.27 より前の MySQL バージョンで MIT-pthreads パッケージを利用しているものには適さない。その当時、Unix ソケットは MIT-pthreads によってサポートされていなかったからである。

- --skip-show-database

  SHOW DATABASES 権限のないユーザに SHOW DATABASES コマンドの使用を認めない。バージョン 4.0.2 から、このオプションは必要なくなった。現在では、SHOW DATABASES 権限が割り当てられることでコマンドを使用できるようになっている。

4.3.4. LOAD DATA LOCAL のセキュリティ関連事項

MySQL 3.23.49 および MySQL 4.0.2 (Windows では 4.0.13) で、LOAD DATA LOCAL に関するセキュリティ対策としていくつかの新しいオプションが追加されました。

このコマンドには次のような潜在的な問題が 2 つあります。

ファイルの読み取りがサーバ側から開始されるため、理論的には、改悪した MySQL サーバを作成しておけば、クライアントがテーブルに対してクエリを実行した時に、そのクライアントコンピュータ上に存在する全てのファイル(カレントユーザが読み取り権を持つ)を、改悪した MySQL サーバが読み取れるということになります。

クライアントが Web サーバから接続する Web 環境では、ユーザは LOAD DATA LOCAL を使用して、Web サーバプロセスが読み取りアクセス権を持つどのファイルでも読み取ることができます (ユーザが SQL サーバに対してすべてのコマンドを実行できる場合) 。

これには 2 つの解決方法があります。

MySQL を --enable-local-infile でコンフィギャしていなければ、mysql_options(... MYSQL_OPT_LOCAL_INFILE, 0) を呼び出さないクライアントは LOAD DATA LOCAL を使用できません。 See 項11.1.3.40.「mysql_options()」。

mysql コマンドラインクライアントに対しては、--local-infile[=1] オプションで LOAD DATA LOCAL を有効にでき、--local-infile=0 オプションで無効にできます。

デフォルトでは、すべての MySQL クライアントとライブラリが --enable-local-infile でコンパイルされ、MySQL 3.23.48 以前との互換性が保たれるようになっています。

MySQL サーバですべての LOAD DATA LOCAL コマンドを無効にするには、mysqld を --local-infile=0 で開始します。

LOAD DATA LOCAL INFILE がサーバまたはクライアントで無効になっている場合、次のエラーメッセージ (1148) が表示されます。

```
The used command is not allowed with this MySQL version
```

4.3.5. 権限システムが行うこと

MySQL 権限システムは、主に、任意のホストから接続したユーザを認証し、そのユーザを、データベース上に登録されている権限 (SELECT、INSERT、UPDATE、DELETE) と関連付けます。

さらに、匿名ユーザを作成し、LOAD DATA INFILE や管理操作機能など MySQL 固有の機能を実行するための権限を設定します。

4.3.6. 権限システムはどのように機能するか

MySQL 権限システムは、すべてのユーザがそれぞれ許可された操作だけを実行できるようにします。MySQL サーバに接続すると、ユーザの ID は接続元のホストおよび指定したユーザ名によって特定されます。権限システムは、ユーザ ID と行いたい操作に応じて権限を設定します。

MySQL では、ホスト名とユーザ名を使用してユーザを認証します。1 つのユーザ名がインターネット上のどこででも同じユーザを示しているという保証がないためです。たとえば、office.com から接続したユーザ joe は、elsewhere.com から接続した joe と同一人物とは限りません。 MySQL では、同じユーザ名でも異なるホストから接続するユーザ間で区別することにより、このことを処理しています。joe に対して、office.com から接続した場合の権限セットと、elsewhere.com から接続した場合の権限セットを別々に設定できます。

MySQL のアクセス制御には 2 段階があります。

- 段階 1: ユーザに接続する権限があるかどうかサーバがチェックする。
- 段階 2: 接続できた場合、要求ごとにそれを実行できる権限があるかどうかサーバがチェックする。たとえば、データベースのあるテーブルからレコードを SELECT したり、データベースのテーブルを DROP しようとすると、ユーザにそのテーブルの SELECT 権限があるかどうか、あるいはデータベースの DROP 権限があるかどうかをサーバがチェックする。

注意: 接続中に権限が変更された場合 (ユーザ自身または第三者によって)、必ずしもその変更は次のクエリに反映されません。詳細については、項4.4.3.「権限の変更はいつ反映されるか」 を参照してください。

サーバはアクセス制御の両方の段階で、mysql データベースの user、db、host の各テーブルを使用します。これらの権限テーブルのフィールドを以下に示します。

テーブル名	user	db	host
スコープフィールド	Host	Host	Host
	User	Db	Db
	Password	User	
権限フィールド	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	References_priv	References_priv
	Reload_priv		
	Shutdown_priv		
	Process_priv		
	File_priv		
	Show_db_priv		
	Super_priv		
	Create_tmp_table_priv	Create_tmp_table_priv	Create_tmp_table_priv
	Lock_tables_priv	Lock_tables_priv	Lock_tables_priv
	Execute_priv		
	Repl_slave_priv		
	Repl_client_priv		
	ssl_type		
	ssl_cypher		
	x509_issuer		
	x509_cubject		
	max_questions		
	max_updates		
	max_connections		

アクセス制御の 2 段階目（要求確認）で、要求がテーブルに関連するものである場合、サーバがさらに tables_priv テーブルおよび columns_priv テーブルを参照することがあります。これらのテーブルのフィールドを以下に示します。

テーブル名	tables_priv	columns_priv
スコープフィールド	Host	Host
	Db	Db
	User	User
	Table_name	Table_name
		Column_name
権限フィールド	Table_priv	Column_priv
	Column_priv	
その他のフィールド	Timestamp	Timestamp

	Grantor	

各権限テーブルには、スコープフィールドと権限フィールドがあります。

スコープフィールドは、テーブルの各登録の範囲を特定します。たとえば、Host と User の値が 'thomas.loc.gov' および 'bob' である user テーブルエントリは、thomas.loc.gov ホストからサーバに接続しようとする bob を認証します。同様に、Host、User、Db の各フィールドの値が 'thomas.loc.gov'、'bob'、および 'reports' である db テーブルエントリは、thomas.loc.gov ホストから reports データベースに接続しようとする bob を認証します。tables_priv テーブルおよび columns_priv テーブルには、各エントリに許可されているテーブルまたはテーブルとカラムの組み合わせを示すスコープフィールドが含まれています。

アクセスをチェックする目的において、Host 値は大文字と小文字の区別がありません。User、Password、Db、および Table_name の値については大文字と小文字が区別されます。 Column_name 値は、MySQL バージョン 3.22.12 以降では大文字と小文字が区別されなくなっています。

権限フィールドは、テーブル内のエントリごとに設定されている権限、つまり何の操作を実行できるかを示します。サーバはさまざまな権限テーブルの情報を組み合わせて、ユーザの権限についての完全な記述を生成します。 この動作に適用されるルールについては、項4.3.10.「アクセス制御の段階 2: 要求確認」 を参照してください。

スコープフィールドは文字列です。ここで示されているように、それぞれのデフォルト値は空の文字列です。

フィールド名	型	注意
Host	CHAR(60)	
User	CHAR(16)	
Password	CHAR(16)	
Db	CHAR(64)	(tables_priv テーブルおよび columns_priv テーブルでは CHAR(60))
Table_name	CHAR(60)	
Column_name	CHAR(60)	

user、db、host の各テーブルでは、すべての権限フィールドが ENUM('N','Y') として宣言されます。それぞれの値は 'N' または 'Y' で、デフォルト値は 'N' です。

tables_priv テーブルおよび columns_priv テーブルでは、権限フィールドは SET フィールドとして宣言されます。

テーブル名	フィールド名	設定可能な要素
tables_priv	Table_priv	'Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter'
tables_priv	Column_priv	'Select', 'Insert', 'Update', 'References'
columns_priv	Column_priv	'Select', 'Insert', 'Update', 'References'

以下、サーバが権限テーブルをどのように使用するか簡潔に説明します。

- user テーブルのスコープフィールドにより、着信した接続を許可するか拒否するか決定する。許可された接続につい

て、user テーブルに設定されている権限はいずれも、そのユーザのグローバル (スーパーユーザ) 権限になる。 これらの権限は、サーバのすべてのデータベースに適用される。

- db テーブルと host テーブルは一緒に使用される。
  - db テーブルのスコープフィールドにより、どのユーザがどのホストからどのデータベースにアクセスできるかを決定する。権限フィールドから、何の操作が許可されているか判断する。
  - host テーブルは、任意の db テーブルエントリをいくつかのホストに適用する場合、db テーブルを補完するものとして使用される。たとえば、1 人のユーザにネットワーク内のいくつかのホストからデータベースへアクセスすることを許可する場合、ユーザの db テーブルエントリの Host 値を空白のままにしておき、それらのホストの各エントリを host テーブルに入力する。このメカニズムの詳細については、項4.3.10.「アクセス制御の段階 2: 要求確認」を参照のこと。
- tables_priv テーブルおよび columns_priv テーブルは db テーブルに似ているが、より詳細な設定が可能。データベースレベルではなく、テーブルおよびカラムレベルで適用される。

注意: 管理者権限 (RELOAD、SHUTDOWN など) は、user テーブルでのみ指定します。管理操作はデータベース固有ではなく、サーバそのもので行う操作なので、この権限を他の権限テーブルで設定する必要はありません。実際、管理操作を実行できるかどうか決定する際には、user テーブルを参照するだけで済みます。

FILE 権限も user テーブルだけで指定します。 これは管理者権限ではありませんが、サーバホスト上でのファイルの読み書きは、アクセスしているデータベースに依存しません。

mysqld サーバは権限テーブルの内容を、起動時に 1 回読み取ります。権限テーブルへの変更の反映については、項4.4.3.「権限の変更はいつ反映されるか」 を参照してください。

権限テーブルの内容を変更する場合、その変更によって、目的の権限が適切に設定されていることを確認してください。問題診断のヘルプについては、項4.3.12.「Access denied エラーの原因」 を参照してください。セキュリティに関するアドバイスについては、項4.3.2.「MySQL のクラッカー対策」 を参照してください。

便利な診断ツールとして、mysqlaccess スクリプトがあります。これは、Yves Carlier が MySQL ディストリビューション用に提供したものです。このツールの動作を確認したい場合には、mysqlaccess を --help オプションで起動してください。 注意: mysqlaccess は user、db、および host テーブルだけを使用してアクセスをチェックします。テーブルまたはカラムレベルの権限はチェックしません。

4.3.7. MySQL が提供する権限

ユーザ権限に関する情報は、mysql データベース (mysql という名前のデータベース) の user、db、host、tables_priv、columns_priv の各テーブルに保存されています。MySQL サーバは起動時と、項4.4.3.「権限の変更はいつ反映されるか」で説明されている状況下において、これらのテーブルを読み取ります。

以下、このマニュアルで使用されている MySQL バージョン 4.0.2 提供の権限名を、それに関連付けられているテーブルカラム名およびその適用範囲とともに示します。各権限の意味については、項4.4.1.「GRANT および REVOKE の構文」 を参照してください。

権限	カラム	適用範囲
ALTER	Alter_priv	テーブル
DELETE	Delete_priv	テーブル

INDEX	Index_priv	テーブル
INSERT	Insert_priv	テーブル
SELECT	Select_priv	テーブル
UPDATE	Update_priv	テーブル
CREATE	Create_priv	データベース、テーブル、またはインデックス
DROP	Drop_priv	データベースまたはテーブル
GRANT	Grant_priv	データベースまたはテーブル
REFERENCES	References_priv	データベースまたはテーブル
CREATE TEMPORARY TABLES	Create_tmp_table_priv	サーバ管理
EXECUTE	Execute_priv	サーバ管理
FILE	File_priv	サーバ上のファイルアクセス
LOCK TABLES	Lock_tables_priv	サーバ管理
PROCESS	Process_priv	サーバ管理
RELOAD	Reload_priv	サーバ管理
REPLICATION CLIENT	Repl_client_priv	サーバ管理
REPLICATION SLAVE	Repl_slave_priv	サーバ管理
SHOW DATABASES	Show_db_priv	サーバ管理
SHUTDOWN	Shutdown_priv	サーバ管理
SUPER	Super_priv	サーバ管理

SELECT、INSERT、UPDATE、DELETE の各権限は、データベース上の既存テーブルのレコードに対する各操作を許可するものです。

SELECT ステートメントは、テーブルからレコードを実際に取り出す場合のみ、SELECT 権限が必要となります。サーバのデータベースへのアクセス権がなくても実行できる SELECT ステートメントもあります。たとえば、mysql クライアントをシンプルな計算機として使用するような場合です。

```
mysql> SELECT 1+1;
mysql> SELECT PI()*2;
```

INDEX 権限では、インデックスを作成または破棄 (削除) できます。

ALTER 権限では、ALTER TABLE を使用できます。

CREATE および DROP 権限では、新規データベースおよびテーブルの作成、既存データベースおよびテーブルの破棄 (削除) を行えます。

注意: ユーザに mysql データベースに対する DROP 権限を与えると、そのユーザは MySQL アクセス権限が保存されているデータベースを破棄できるということになるので注意が必要です。

GRANT 権限では、ユーザが所有している権限を他のユーザに与えることができます。

FILE 権限では、LOAD DATA INFILE および SELECT ... INTO OUTFILE ステートメントを使用してサーバ上でファイルの読み書きを行えます。この権限を与えられたユーザはだれでも、MySQL サーバによってアクセス可能な読み取り可能ファイルをすべて読むことができ、また MySQL サーバによって書き込めるどのディレクトリにも新規読み取り可能ファイルを作成できます。 このユーザはまた、カレントデータベースディレクトリ内のすべてのファイルを読むことができます。ただし、既存ファイルの変更はできません。

これら以外の権限は、mysqladmin プログラムを使用して実行される管理操作用に使用されます。以下の表で、どの管理権限でどの mysqladmin コマンドを実行できるかを示します。

権限	実行可能なコマンド
RELOAD	reload、refresh、flush-privileges、flush-hosts、flush-logs、flush-tables
SHUTDOWN	shutdown
PROCESS	processlist
SUPER	kill

reload コマンドは、権限テーブルを再読み込みするようにサーバに命令します。refresh コマンドは、すべてのテーブルをフラッシュし、ログファイルを開いて閉じます。flush-privileges は reload のシノニムです。他の flush-* コマンドも refresh と同様の機能を果たしますが、範囲が限られており、状況によって使い分けてください。たとえば、ログファイルだけをフラッシュするには、refresh の代わりに flush-logs を使用します。

shutdown コマンドはサーバをシャットダウンします。

processlist コマンドは、サーバで実行中のスレッドに関する情報を表示します。kill コマンドはサーバスレッドを強制終了します。ユーザはいつでも自分のスレッドを表示および強制終了することができますが、他のユーザによって開始されたスレッドを表示するには PROCESS 権限が、強制終了するには SUPER 権限が必要です。 See 項4.6.7.「KILL 構文」。

一般的な指針として、ユーザにはそのユーザが必要とする権限だけを与えてください。また、以下の権限を与える際には注意が必要です。

- GRANT 権限は、ユーザが自分の権限を他のユーザに与えることができる。2 人のユーザが異なる権限を持ち、両方とも GRANT 権限がある場合、お互いの権限を組み合わせることができる。
- ALTER 権限は、テーブルの名前を変更することにより、権限システムを崩壊させることができる。
- FILE 権限を悪用すれば、サーバの読み取り可能ファイルまたはカレントデータベースディレクトリのファイルをデータベーステーブルに読み込んで、SELECT を使用してその内容にアクセスできる。
- SHUTDOWN 権限を悪用すれば、サーバを終了して、他のユーザへのサービス妨害を行うことができる。
- PROCESS 権限は、パスワードの設定や変更を行うクエリなども含め、現在実行中のクエリのプレーンテキストを表示することができる。
- mysql データベースの権限があれば、パスワードおよびその他のアクセス権限情報を変更することができる（パスワードは暗号化されて保存されているため、悪意のあるユーザが単に読み取ってもテキスト形式のパスワードを知ることはできない）。mysql.user パスワードカラムにアクセスできれば、それを使用して特定ユーザの MySQL サーバにログインできる（必要とされる権限を持っていれば、そのユーザはパスワードを書き換えることもできる）。

MySQL 権限システムでは達成できないこともいくつかあります。

- 特定ユーザからのアクセスを拒否するよう明示的に指定できない。 つまり、ユーザを特定して、その接続を拒否することはできない。
- データベース内のテーブルの作成および破棄の権限をユーザに与え、それと同時にデータベースの作成および破棄をできないようには指定できない。

4.3.8. MySQL サーバへの接続

MySQL クライアントでは通常、MySQL サーバにアクセスする際、接続先のホスト、ユーザ名、パスワードといった接続パラメータを指定する必要があります。たとえば、mysql クライアントは以下のように開始することができます (オプション引数は '[' と ']' で囲んであります) 。

```
shell> mysql [-h host_name] [-u user_name] [-pyour_pass]
```

-h、-u、および -p オプションの別の形として、--host=host_name、--user=user_name、および --password=your_pass があります。注意: -p または --password= とそれに続くパスワードの間にスペースは入りません。

注意: コマンドラインでパスワードを指定するのは安全ではありません。 同じシステム内のどのユーザでも、ps auxww のようなコマンドを使用してパスワードを知ることができてしまいます。 See 項4.1.2.「my.cnf オプション設定ファイル」。

mysql は、コマンドラインで指定されなかった接続パラメータに以下のデフォルト値を使用します。

- デフォルトホスト名は localhost。
- デフォルトユーザ名はユーザの Unix ログイン名。
- -p が指定されなければ、パスワードは無しになる。

したがって、Unix ユーザ joe では、以下のコマンドがいずれも同じ意味になります。

```
shell> mysql -h localhost -u joe
shell> mysql -h localhost
shell> mysql -u joe
shell> mysql
```

他の MySQL クライアントでも同様です。

Unix システムでは、接続時に別のデフォルト値を使用するように設定することができます。これにより、クライアントプログラムを起動するたびにコマンドラインで指定する必要がなくなります。設定方法は 2 つあります。

- ホームディレクトリ内にある .my.cnf 設定ファイルの [client] セクションで接続パラメータを指定する。このセクションは以下のようになる。

  ```
  [client]
  host=host_name
  user=user_name
  ```

```
password=your_pass
```

See 項4.1.2. 「my.cnf オプション設定ファイル」。

- 環境変数を使用して接続パラメータを指定する。mysql のホストは、MYSQL_HOST を使用して指定できる。MySQL ユーザ名は、USER を使用して指定できる (Windows のみ) 。パスワードは、MYSQL_PWD を使用して指定できる (ただし、これは安全ではない。次のセクションを参照のこと) 。 See 付録 F. 環境変数。

4.3.9. アクセス制御の段階 1: 接続確認

ユーザが MySQL サーバに接続しようとした場合、そのユーザ ID、およびそれを正しいパスワードで裏付けできるかどうかによって、サーバは接続の許可または拒否を行います。パスワードが正しくない場合、サーバはアクセスを完全に拒否します。正しければ、サーバは接続を許可し、段階 2 に進んで要求を待ちます。

ユーザ ID は、2 つの情報に基づきます。

- 接続元のホスト
- MySQL ユーザ名

ID チェックには、user テーブルの 3 つのスコープフィールド (Host、User、Password) が使用されます。user テーブルエントリが、指定されたホスト名とユーザ名に一致し、入力したパスワードが正しかった場合のみ、接続が許可されます。

user テーブルのスコープフィールドには次の方法で値を指定できます。

- Host 値にはホスト名または IP アドレスを使用できる。ローカルホストを指定する場合は 'localhost' を使用する。
- ワイルドカード文字 '%' および '_' を Host フィールドに使用できる。
- Host 値としての '%' は、すべてのホスト名を意味する。
- 空白の Host 値は、該当するホスト名に一致する host テーブルによって決定することを意味する。 詳細については、次の章を参照のこと。
- MySQL バージョン 3.23 より、IP アドレスとして指定する Host 値に、ネットワークアドレスを示すためのネットマスクを使用できるようになっている。次に例を示す。

```
mysql> GRANT ALL PRIVILEGES ON db.*
    -> TO david@'192.58.197.0/255.255.255.0';
```

これにより、以下の条件に当てはまる IP からだれでも接続できるようになる。

```
user_ip & netmask = host_ip.
```

上の例では、192.58.197.0 〜 192.58.197.255 間の IP すべてが MySQL サーバに接続できる。

- User フィールドにワイルドカード文字は使用できないが、空白の値は使用でき、これは任意のユーザ名と一致する。

user テーブルエントリに空白のユーザ名があり、接続が空白ユーザーにマッチした場合、そのユーザはクライアントが実際に指定した名前のユーザではなく、匿名ユーザ (名前なしのユーザ) と見なされる。この場合、接続中 (段階 2 の間) に行われる後続のアクセスチェックはすべて、空白のユーザ名で行われる。

- Password フィールドは空白にできる。これはどのパスワードにも一致するという意味ではなく、そのユーザがパスワード指定せずに接続する必要があるという意味である。

空白ではない Password 値は、暗号化パスワードを表します。 MySQL では、パスワードを平文テキストでは保存しません。接続しようとするユーザが入力したパスワードは、暗号化されます (PASSWORD() 関数を使用) 。クライアントおよびサーバがパスワードの確認を行う際、その暗号化されたパスワードが使用されます (このとき、その接続を使って、暗号化されたパスワードが転送されることはありません) 。注意: MySQL から見ると暗号化されたパスワードが実際のパスワードなので、暗号化パスワードにだれにもアクセスできないようにしてください。特に、一般のユーザに mysql データベース内のテーブルの読み取りアクセス権を与えないでください。 バージョン 4.1 より、MySQL は従来とは異なるパスワードおよびログインメカニズムを採用しており、TCP/IP パケットが盗み見されたり、mysql データベースがキャプチャされた場合でも安全になっています。

以下の例は、user テーブルエントリの Host 値および User 値のさまざまな組み合わせがどのように着信の接続に適用されるかを示したものです。

Host 値	User 値	エントリによって許可される接続
'thomas.loc.gov'	'fred'	thomas.loc.gov から接続する fred
'thomas.loc.gov'	''	thomas.loc.gov から接続するすべてのユーザ
'%'	'fred'	任意のホストから接続する fred
'%'	''	任意のホストから接続するすべてのユーザ
'%.loc.gov'	'fred'	loc.gov ドメイン内の任意のホストから接続する fred
'x.y.%'	'fred'	x.y.net、x.y.com、x.y.edu などから接続する fred (これは実用的ではない)
'144.155.166.177'	'fred'	IP アドレス 144.155.166.177 のホストから接続する fred
'144.155.166.%'	'fred'	144.155.166 クラス C サブネットの任意のホストから接続する fred
'144.155.166.0/255.255.255.0'	'fred'	1 つ上の例と同じ

Host フィールドでは IP のワイルドカード値を使用できるので (たとえば、'144.155.166.%' で、サブネットのすべてのホストに一致) 、ホストを 144.155.166.somewhere.com などと名付けてこの機能が悪用される可能性があります。これを防ぐため、MySQL は数値やドットで始まるホスト名を認めません。したがって、1.2.foo.com などのホスト名は、権限テーブルの Host カラムと決して一致しません。IP アドレスのみ IP ワイルドカード値との突き合わせが可能です。

ある接続が、user テーブルの複数のエントリと一致することがあります。たとえば、thomas.loc.gov からの fred による接続は、上記の表のいくつかのエントリと一致します。複数のエントリが一致した場合、サーバは次のようにエントリを選択します。起動時に user テーブルを読み込んだ後、それをソートします。ユーザが接続しようとしたとき、そのソート順でエントリとの突き合わせを行います。そして最初に一致したエントリを使用します。

user テーブルのソートは以下のように行われます。user テーブルが以下の内容であるとします。

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

```
| Host      | User     | ...
+-----------+----------+-
| %         | root     | ...
| %         | jeffrey  | ...
| localhost | root     | ...
| localhost |          | ...
+-----------+----------+-
```

サーバがテーブルを読み込むと、最も具体的な Host 値を最初にもってきます (Host カラムの '%' は ``任意のホスト" を意味し、具体性が低いとなります ) 。同じ Host 値のエントリ間で、最も具体的な User 値を最初にもってきます ( 空白の User 値は ``任意のユーザ" を意味し、具体性が低いとなります) 。ソートされた user テーブルは以下のようになります。

```
+-----------+----------+-
| Host      | User     | ...
+-----------+----------+-
| localhost | root     | ...
| localhost |          | ...
| %         | jeffrey  | ...
| %         | root     | ...
+-----------+----------+-
```

接続が試みられると、サーバはソートされたエントリで突き合わせを行い、最初に一致したものを使用します。localhost からの jeffrey による接続に対しては、Host カラムが 'localhost' のエントリが最初に一致します。これらのうち、空白ユーザ名のエントリが接続ホスト名とユーザ名の両方で一致します ('%'/'jeffrey' エントリも一致しますが、これはテーブル内での最初の突き合わせにはなりません) 。

もう 1 つ例を示します。user テーブルが以下の内容であるとします。

```
+----------------+----------+-
| Host           | User     | ...
+----------------+----------+-
| %              | jeffrey  | ...
| thomas.loc.gov |          | ...
+----------------+----------+-
```

ソートされたテーブルは以下のようになります。

```
+----------------+----------+-
| Host           | User     | ...
+----------------+----------+-
| thomas.loc.gov |          | ...
| %              | jeffrey  | ...
+----------------+----------+-
```

thomas.loc.gov からの jeffrey による接続は最初のエントリと一致し、whitehouse.gov からの jeffrey による接続は 2 番目のエントリと一致します。

サーバが接続の突き合わせを行うとき、指定されたユーザ名を明示的に示すすべてのエントリが最初に使用されると誤解しがちです。これは間違っています。上記の例でもわかるように、thomas.loc.gov からの jeffrey による接続は、'jeffrey' が User フィールド値であるエントリではなく、ユーザ名なしのエントリと最初に一致します。

サーバへの接続がうまくいかない場合、user テーブルを出力して、最初の突き合わせがどれか確認してください。 正常に接続が行われても、権限が予期したものと違う場合、CURRENT_USER() 関数 (バージョン 4.0.6 で導入) を使用して、

その接続に実際に一致しているユーザとホストの組み合わせを確認できます。 See 項6.3.6.2.「その他の各種関数」。

4.3.10. アクセス制御の段階 2: 要求確認

接続が確立すると、サーバは段階 2 に移行します。その接続での要求ごとに、サーバはユーザにそれを実行できる権限があるかどうかを、操作タイプに基づいてチェックします。ここで、権限テーブルの権限フィールドが関係してきます。権限は、user、db、host、tables_priv、columns_priv のテーブルで設定できます。権限テーブルの設定は、GRANT コマンドと REVOKE コマンドで行います。 See 項4.4.1.「GRANT および REVOKE の構文」。 各権限テーブルのフィールドのリストについては、項4.3.6.「権限システムはどのように機能するか」 を参照してください。

user テーブルは、カレントデータベースに関係なく、ユーザに対してグローバルに権限を設定します。たとえば、user テーブルで DELETE 権限が設定されたユーザは、そのサーバホストのどのデータベースのレコードでも削除できます。つまり、user テーブルで許可された権限はスーパーユーザ権限です。user テーブルで権限を設定する対象は、サーバ管理者やデータベース管理者などのスーパーユーザだけにしておくのが賢明です。他のユーザについては、user テーブルの権限を 'N' に設定しておき、db テーブルおよび host テーブルを使用してデータベース依存で権限を設定してください。

db テーブルおよび host テーブルは、データベース依存の権限を設定します。 スコープフィールドには次の方法で値を指定できます。

- ワイルドカード文字の '%' および '_' を、両テーブルの Host フィールドと Db フィールドで使用できる。たとえば、'_' 文字をデータベース名の一部として使用するには、GRANT コマンドで '_' として指定する。
- db テーブルの '%' Host 値は、``任意のホスト" を意味する。db テーブルの空白の Host 値は、``host テーブルを参照すること" を意味する。
- host テーブルの '%' または空白の Host 値は、``任意のホスト" を意味する。
- 両テーブルにおいて '%' または空白の Db 値は、``任意のデータベース" を意味する。
- 両テーブルにおいて空白の User 値は、匿名ユーザを意味する。

db テーブルおよび host テーブルは、サーバ起動時に読み取られてソートされます (user テーブル読み込みと同時) 。db テーブルは Host、Db、および User のスコープフィールドでソートされ、host テーブルは Host と Db のスコープフィールドでソートされます。user テーブルでは、最も具体的な値が最初に、最も抽象的な値が最後にソートされ、サーバによるエントリの突き合わせでは、最初に一致したエントリが使用されます。

tables_priv テーブルと columns_priv テーブルはそれぞれ、テーブル依存およびカラム依存の権限を設定します。スコープフィールドには次の方法で値を指定できます。

- ワイルドカード文字の '%' および '_' を両テーブルの Host フィールドで使用できる。
- 両テーブルにおいて '%' または空白の Host 値は、``任意のホスト" を意味する。
- 両テーブルの Db、Table_name、Column_name の各フィールドで、ワイルドカードおよび空白を使用できない。

tables_priv テーブルおよび columns_priv テーブルは、Host、Db、および User のフィールドでソートされます。これは db テーブルのソートと同様ですが、ワイルドカードの使用が Host フィールドだけに限定されるので、よりシンプルです。

以下、要求確認プロセスについて説明します。アクセスチェックのソースコードを知っている読者は、ここでの説明とコードで使用されるアルゴリズムに少し違いがあることに気付かれるでしょうが、説明はコードが実際に行うことと同じで、違いは説明をシンプルにするためのものです。

管理要求（SHUTDOWN、RELOAD など）に対して、サーバは user テーブルエントリだけをチェックします。管理権限を指定するのはこのテーブルのみであるためです。そのエントリが要求された操作を許可している場合はアクセスが認められ、そうでない場合は拒否されます。たとえば、mysqladmin shutdown を実行しようとしても、user テーブルエントリで SHUTDOWN 権限が設定されていなければ、db テーブルや host テーブルをチェックすることなくアクセスは拒否されます（これらのテーブルには Shutdown_priv カラムがなく、チェックの必要性がありません）。

データベース関連の要求（INSERT、UPDATE など）に対しては、サーバは最初に、user テーブルエントリでユーザのグローバル（スーパーユーザ）権限をチェックします。ここで、要求されている操作が許可されていれば、アクセスが認められます。user テーブルのグローバル権限では十分でない場合、サーバはユーザのデータベースに関する権限を db テーブルおよび host テーブルでチェックします。

1. サーバは、db テーブルの Host、Db、User の各フィールドをチェックする。Host フィールドと User フィールドでは、接続ユーザのホスト名と MySQL ユーザ名がチェックされる。Db フィールドでは、ユーザがアクセスしようとしているデータベースがチェックされる。Host および User に該当するエントリがない場合、アクセスは拒否される。

2. db テーブルに、マッチするエントリがあり、その Host フィールドが空白でなければ、そのエントリが、ユーザのデータベースに対する権限を定義する。

3. db テーブルにある、マッチするエントリの Host フィールドが空白の場合、これは host テーブルが、データベースにアクセスできるホストを指定することを意味する。この場合、host テーブルの Host フィールドと Db フィールドがチェックされる。 host テーブル内にマッチするエントリがなければ、アクセスは拒否される。マッチするエントリがあれば、ユーザのデータベースに対する権限が、db および host テーブルエントリの（和集合ではなく）共通集合として計算される。これは、両方のエントリで 'Y' に設定されている権限を指す。このように、db テーブルエントリで全般的な権限を設定し、host テーブルエントリでホストごとに権限を制限することができる。

サーバは、db および host テーブルエントリからデータベースに対する権限を特定したら、それらの権限を user テーブルで設定されているグローバル権限に足し合わせます。その結果、要求されている操作が許可されていれば、アクセスが認められます。そうでない場合、サーバは tables_priv テーブルと columns_priv テーブルでユーザのテーブル権限とカラム権限をチェックし、これらをユーザの権限に追加します。その結果に基づき、アクセスが許可または拒否されます。

ブール値で表すと、上記のユーザ権限算出法は以下のようにまとめることができます。

```
グローバル権限
OR (database アクセス権 AND host アクセス権)
OR table アクセス権
OR column アクセス権
```

user 内のグローバル権限が十分ではない場合に、サーバがその不十分なグローバル権限を、データベース、テーブル、およびカラム権限と足し合わせる理由はわかりにくいかもしてません。 その理由は、要求に複数の権限が必要な場合があるからです。たとえば、INSERT ... SELECT ステートメントを実行するには、INSERT 権限および SELECT 権限の両方が必要です。 そのうち 1 つの権限が user テーブル内のエントリにあり、もう 1 つの権限が db テーブル内のエントリにある場合があります。その場合、要求を実行するのに必要な権限をユーザは持っていますが、サーバはどちらか一方のテーブルだけでは判断できません。この場合、両テーブルのエントリにある権限を組み合わせる必要があります。

host テーブルを使用して、セキュリティで保護されたサーバのリストを管理することができます。

TcX DataKonsult AB では、host テーブルに、ローカルネットワーク上のすべてのコンピュータの一覧を入れています。これらにすべての権限を設定しています。

host テーブルを使用して、セキュリティで保護されていないホストを示すこともできます。 セキュリティで保護されていない公開エリア内に、コンピュータ public.your.domain があるとします。以下のように、host テーブルのエントリを使用して、ネットワーク内のそのコンピュータ以外の全ホストにアクセスを許可することができます。

```
+--------------------+----+-
| Host               | Db | ...
+--------------------+----+-
| public.your.domain | %  | ... (全権限を 'N' にセット)
| %.your.domain      | %  | ... (全権限を 'Y' にセット)
+--------------------+----+-
```

当然、権限テーブルは常にテストして (mysqlaccess などを使用) 、アクセス権限が目的どおりに設定されていることを確認してください。

4.3.11. MySQL 4.1 のパスワードハッシュ

MySQL ユーザアカウントは、mysql データベースの user テーブルにリストアップされています。各 MySQL アカウントにはパスワードが割り当てられますが、user テーブルの Password カラムに保存されるのは平文テキストのパスワードではなく、パスワードから計算されたハッシュ値です。パスワードハッシュ値は、PASSWORD() 関数によって計算されます。

MySQL は、クライアントとサーバ間通信の 2 つのフェーズでパスワードを使用します。

- まず、クライアントがサーバに接続しようとするとき、初期認証ステップとして、クライアントはパスワードを提示する。このパスワードは、クライアントが使用を希望するアカウントの、ユーザテーブルに保存されたハッシュ値と一致している必要がある。
- 次に、クライアントが接続した後、クライアントは、user テーブルに含まれているパスワードハッシュを設定または変更することができる (適切な権限がある場合) 。このためには、クライアントは、PASSWORD() 関数を使用してパスワードハッシュを生成するか、GRANT または SET PASSWORD ステートメントを使用する。

つまり、クライアントが最初に接続しようとする際、サーバが認証するのにハッシュ値を使用します。接続したクライアントが PASSWORD() 関数を実行したり、GRANT または SET PASSWORD ステートメントを使用してパスワードの設定または変更を行うと、サーバがハッシュ値を生成します。

パスワードハッシュメカニズムは MySQL 4.1 で更新され、セキュリティが向上し、パスワード盗難の危険性が少なくなっています。 ただし、この新しいメカニズムは 4.1 サーバと 4.1 クライアントしか理解できないため、互換性の問題があります。 4.1 クライアントはパスワードハッシュの新旧メカニズムの両方を理解できるので、4.1 より前のサーバにでも接続できます。しかし、4.1 より前のクライアントが 4.1 サーバに接続しようとすると、問題が発生する可能性があります。たとえば、4.0 mysql クライアントが 4.1 サーバに接続しようとすると、以下のエラーメッセージが表示される可能性があります。

```
shell> mysql
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

以下、以前のパスワードメカニズムと新しいパスワードメカニズムとの違い、およびサーバを 4.1 にアップグレードしたときに 4.1 より前のクライアントと互換性を保つ方法について説明します。

注意: ここでの説明は 4.1 とそれ以前の動作を対比するものですが、ここで説明する 4.1 の動作は実際には 4.1.1 からのものです。MySQL 4.1.0 の動作は、4.1.1 以降で導入されたメカニズムと少し異なります。4.1.0 とそれ以降のバージョンとの違いについては、後で説明します。

MySQL 4.1 より前は、PASSWORD() 関数によって計算されるパスワードハッシュは 16 バイト長でした。次に例を示します。

```
mysql> SELECT PASSWORD('mypass');
+-------------------+
| PASSWORD('mypass') |
+-------------------+
| 6f8c114b58f2ce9e   |
+-------------------+
```

MySQL 4.1 より前は、user テーブルの Password カラム (ハッシュが保存される場所) も 16 バイト長でした。

MySQL 4.1 から、PASSWORD() 関数が、より長い 41 バイトのハッシュ値を生成するようになっています。

```
mysql> SELECT PASSWORD('mypass');
+---------------------------------------------+
| PASSWORD('mypass')                          |
+---------------------------------------------+
| *43c8aa34cdc98eddd3de1fe9a9c2c2a9f92bb2098d75 |
+---------------------------------------------+
```

したがって、この長さの値を保存するため、user テーブルの Password カラムも 41 バイト長であることが必要です。

- MySQL 4.1 の新規インストールを実行すると、Password カラムは自動的に 41 バイト長になる。
- 旧バージョンを 4.1 にアップグレードした場合、mysql_fix_privilege_tables スクリプトを実行して Password カラムを 16 バイト長から 41 バイト長に変更する必要がある。このスクリプトは既存のパスワード値を変更しない。既存のものは 16 バイト長のままである。

拡張された Password カラムは、新旧どちらの形式のパスワードハッシュも保存できます。与えられたパスワードハッシュ値の形式は次の 2 つの違いにより判断されます。

- 16 バイトと 41 バイトという長さ自体が違う。
- 2 つ目の違いとしては、新しい形式のパスワードハッシュは常に '*' 文字で始まり、旧形式のパスワードはそれ以外の文字で始まる。

長いパスワードハッシュ形式の方が暗号化の面で優れており、クライアント認証において旧形式の短いハッシュよりもセキュリティが高いと言えます。

パスワードハッシュの長短の違いは、認証時にサーバがパスワードを使用する方法と、パスワードを変更する接続クライアントに対してサーバがパスワードハッシュを生成する方法に関係してきます。

認証時にサーバがパスワードを使用する方法は、Password カラムの幅により次のような影響を受けます。

- カラムが短い場合、短いハッシュ認証だけが使用される。
- カラムが長い場合、長短のハッシュどちらでも保存でき、サーバはどちらの形式でも使用できる。
  - 4.1 より前のクライアントは接続はできても、旧ハッシュメカニズムしか理解できないため、短いハッシュのアカウントしか認証できない。
  - 4.1 クライアントは、長短どちらのハッシュのアカウントでも認証できる。

短いハッシュのアカウントでも、4.1 クライアントの方が旧クライアントよりも認証プロセスのセキュリティが少し向上しています。認証のセキュリティは以下の順で高くなります。

- 短いパスワードハッシュのアカウントに対する 4.1 より前のクライアント認証
- 短いパスワードハッシュのアカウントに対する 4.1 クライアント認証
- 長いパスワードハッシュのアカウントに対する 4.1 クライアント認証

接続クライアントに対してサーバがパスワードハッシュを生成する方法は、Password カラムの幅と --old-passwords オプションにより影響を受けます。4.1 サーバは、次の条件が満たされた場合だけ、長いハッシュを生成します。 Password カラムが長いハッシュ値を保存できる長さであること、および --old-passwords オプションが指定されていないことが必要です。 これらの条件は以下のように適用されます。

- Password カラムが長いハッシュを保存できる長さ（41 バイト）であること。 カラムが更新されておらず、4.1 より前の長さ（16 バイト）のままだと、クライアントが PASSWORD()、GRANT、または SET PASSWORD を使用してパスワード変更操作を行ったとき、長いハッシュが収まらないことをサーバが認識し、短いハッシュを生成する（4.1 にアップグレードしたが、mysql_fix_privilege_tables スクリプトを実行せず、Password カラムが長くなっていない場合にこのようになる）。
- Password カラムが長ければ、長短どちらのパスワードハッシュも保存できる。この場合、サーバが --old-passwords オプションで起動されていなければ、PASSWORD()、GRANT、および SET PASSWORD により長いハッシュが生成される。 このオプションが指定されていると、強制的に短いパスワードハッシュが生成される。

--old-passwords オプションの目的は、サーバが長いパスワードハッシュを生成する環境において、4.1 より前のクライアントと下位互換性を保てるようにすることです。これは認証には影響しませんが（4.1 クライアントは長いパスワードハッシュのアカウントを使用できる）、パスワード変更操作の結果として user テーブルで長いパスワードハッシュが生成されることを防ぎます。長いパスワードハッシュが生成されると、そのアカウントは 4.1 より前のクライアントでは使用できなくなります。--old-passwords オプションを指定しない場合、以下のシナリオが想定されます。

- 旧クライアントが、短いパスワードハッシュのアカウントに接続する。
- クライアントがアカウントのパスワードを変更する。--old-passwords が指定されていない場合、アカウントに対して長いパスワードハッシュが生成される。
- 次回、旧クライアントがこのアカウントに接続しようとしても、認証時にアカウントが新規ハッシュメカニズムを必要

とするため、接続できない。user テーブルでアカウントに対して長いパスワードハッシュが生成された場合、4.1 クライアントのみそれを認証できる。4.1 より前のクライアントでは長いハッシュを理解できない。

このシナリオでは、4.1 より前の旧クライアントをサポートする必要がある場合、--old-passwords オプションを使用せずに 4.1 サーバを起動するのは危険であることを示しています。--old-passwords を指定してサーバを起動することにより、パスワード変更操作を行っても長いパスワードハッシュは生成されません。したがって、旧クライアントがアカウントにアクセスできなくなることがありません（旧クライアントが、パスワード変更時、偶発的に長いパスワードハッシュを生成して自らをロックアウトすることはありません）。

--old-passwords オプションの欠点は、4.1 クライアントでも、パスワード作成または変更時に短いハッシュを使用するということです。そのため、長いパスワードハッシュによる高いセキュリティを活用できません。4.1 クライアント用などに長いハッシュのアカウントを作成したい場合には、--old-passwords なしでサーバを実行する必要があります。

4.1 サーバの実行には以下のシナリオが想定されます。

シナリオ 1. user テーブルの短い Password カラム

- Password カラムには短いハッシュのみ保存できる。
- サーバはクライアント認証時、短いハッシュだけを使用する。
- 接続クライアントは、PASSWORD()、GRANT、または SET PASSWORD を使用するパスワードハッシュ生成操作で短いハッシュだけを使用する。アカウントのパスワードを変更すると、必ず短いパスワードハッシュになる。
- --old-passwords オプションは使用できるが、意味がない。短い Password カラムでは、サーバは短いパスワードハッシュしか生成しない。

シナリオ 2. 長い Password カラム、サーバを --old-passwords オプションなしで起動

- Password カラムには長短いずれのハッシュも保存できる。
- 4.1 クライアントは、長短どちらのハッシュのアカウントでも認証できる。
- 4.1 より前のクライアントは、短いハッシュのアカウントのみ認証できる。
- 接続クライアントは、PASSWORD()、GRANT、または SET PASSWORD を使用するパスワードハッシュ生成操作で長いハッシュだけを使用する。アカウントのパスワードを変更すると、必ず長いパスワードハッシュになる。
- OLD_PASSWORD() を使用して、明示的に短いハッシュを生成するようにできる。たとえば、アカウントに短いパスワードを割り当てるには、以下のように UPDATE を使用する。

```
mysql> UPDATE user SET Password = OLD_PASSWORD('mypass')
    -> WHERE Host = 'some_host' AND User = 'some_user';
mysql> FLUSH PRIVILEGES;
```

上述したように、このシナリオでは、短いパスワードハッシュのアカウントに、4.1 より前のクライアントがアクセスできなくなる危険性があります。GRANT、SET PASSWORD、または PASSWORD() を使用してアカウントのパスワードを変更すると、新規パスワードハッシュが長くなるため、4.1 より前のクライアントは、4.1 にアップグレードしないとそのア

カウントを認証できなくなります。

シナリオ 3. 長い Password カラム、サーバを --old-passwords オプションで起動

- Password カラムには長短いずれのハッシュも保存できる。
- 4.1 クライアントは長短いずれのハッシュのアカウントでも認証できる (注意: 長いハッシュの作成は、サーバを --old-passwords なしで起動したときだけ可能) 。
- 4.1 より前のクライアントは、短いハッシュのアカウントのみ認証できる。
- 接続クライアントは、PASSWORD()、GRANT、または SET PASSWORD を使用するパスワードハッシュ生成操作で短いハッシュだけを使用する。アカウントのパスワードを変更すると、必ず短いパスワードハッシュになる。

このシナリオでは、--old-passwords によって長いハッシュが生成されないため、長いパスワードハッシュのアカウントを作成できません。また、--old-passwords オプションを使用する前に長いハッシュのアカウントを作成した場合でも、--old-passwords の有効時にアカウントのパスワードを変更すると短いパスワードになってしまうので、長いハッシュが持つセキュリティ上の利点が失われます。

これらのシナリオの欠点はそれぞれ以下のように要約できます。

シナリオ 1. セキュリティの高い認証を提供する長いハッシュを活用できない。

シナリオ 2. OLD_PASSWORD() を使用せずにパスワードを変更すると、短いハッシュのアカウントに 4.1 より前のクライアントがアクセスできなくなる。

シナリオ 3. --old-passwords によって、短いハッシュのアカウントがアクセス不能になることは防がれるが、長いハッシュをパスワード変更すると短いハッシュになってしまい、--old-passwords が有効な間はそれを元の長さに戻せない。

アプリケーションプログラムに対するパスワードハッシュ変更の影響

MySQL 4.1 にアップグレードすると、独自の目的で PASSWORD() を使用してパスワードを生成するアプリケーションとの互換性に問題が生じます。本来、PASSWORD() は MySQL ユーザのパスワード管理専用なので、アプリケーションでこれを実行すべきではありません。しかし現状では、いくつかのアプリケーションが独自の目的で PASSWORD() を使用しています。4.1 にアップグレードし、長いパスワードハッシュが生成される状態でサーバを実行すると、独自の目的で PASSWORD() を使用するアプリケーションは壊れます。推奨される対処法は、アプリケーションを修正して SHA1() または MD5() など、別の関数を使用してハッシュ値を生成するように設定することです。 それが可能でなければ、OLD_PASSWORD() 関数を使用することができます。これは、旧形式の短いハッシュを生成するためのものです (注意: ただし、OLD_PASSWORD() は将来サポートされなくなる可能性があります) 。

短いハッシュを生成する状態でサーバが実行中の場合、OLD_PASSWORD() は利用可能ですが、PASSWORD() と同じになります。

MySQL 4.1.0 のパスワードハッシュは 4.1.1 以降のパスワードハッシュと異なります。 4.1.0 の違いは以下のとおりです。

- パスワードハッシュは 41 バイトではなく、45 バイト長である。
- PASSWORD() 関数は繰り返し不可。つまり、特定の引数 X で、連続して PASSWORD(X) を呼び出すと異なる結果が生成される。

4.3.12. Access denied エラーの原因

MySQL サーバに接続しようとして Access denied エラーが発生した場合の対処法を、以下に示します。

- MySQL のインストール後、mysql_install_db スクリプトを実行して権限テーブルを初期設定していなければ、これを行う。 See 項4.4.4.「MySQL 権限の初期設定」。 以下のコマンドを実行して初期権限をテストする。

  ```
  shell> mysql -u root test
  ```

  エラーが発生することなく接続できるはずである。また、MySQL データベースディレクトリに user.MYD ファイルがあることを確認する。 通常、これは PATH/var/mysql/user.MYD である。ここで PATH は、MySQL インストール先ディレクトリのパス名である。

- 初期インストール後、サーバに接続してユーザとそのアクセス権をセットアップする。

  ```
  shell> mysql -u root mysql
  ```

  MySQL root ユーザには、最初パスワードがないため、問題なく接続できるはずである。これはセキュリティ上のリスクでもあるため、他の MySQL ユーザをセットアップするとき、同時に root パスワードも設定すべきである。

  root として接続しようとして以下のエラーが表示された場合

  ```
  Access denied for user: '@unknown' to database mysql
  ```

  これは、user テーブルの User カラムに 'root' 値がなく、mysqld がクライアントのホスト名を解決できないことを意味する。この場合、--skip-grant-tables オプションでサーバを再起動し、/etc/hosts ファイルまたは \windows\hosts ファイルを編集してホストのエントリを追加する必要がある。

- 以下のエラーが発生した場合

  ```
  shell> mysqladmin -u root -pxxxx ver
  Access denied for user: 'root@localhost' (Using password: YES)
  ```

  入力したパスワードが正しくないことを意味する。 See 項4.4.8.「パスワードの設定」。

  root のパスワードを忘れた場合、--skip-grant-tables オプションで mysqld を再起動してパスワードを変更できる。 See 項A.4.2.「忘れたルートパスワードをリセットする方法」。

  パスワードを指定していないにもかかわらず上記のエラーが発生する場合、何らかの my.cnf ファイルに正しくないパスワードが存在していることを意味する。See 項4.1.2.「my.cnf オプション設定ファイル」。 以下のように --no-defaults オプションを使用すると、オプション設定ファイルの使用を避けることができる。

  ```
  shell> mysqladmin --no-defaults -u root ver
  ```

- 既存の MySQL インストールを、バージョン 3.22.11 より前からバージョン 3.22.11 以降にアップグレードした後で、mysql_fix_privilege_tables スクリプトを実行していなければ、これを実行する。GRANT ステートメントが導入された MySQL バージョン 3.22.11 では、権限テーブルの構造が変更されている。 See 項2.5.6.「権限テーブルのアップグレード」。

- セッション途中で権限が変更されているようであれば、スーパーユーザが変更した可能性がある。権限テーブルを再読

み込みすると、新しいクライアント接続に影響し、また、項4.4.3.「権限の変更はいつ反映されるか」 で説明されているように、既存の接続にも影響する。

- パスワードが機能しない場合に INSERT、UPDATE、または SET PASSWORD ステートメントを使用してパスワードを設定する際には PASSWORD() 関数を使用することが必要である。GRANT ... IDENTIFIED BY ステートメントまたは mysqladmin password コマンドを使用してパスワードを指定する場合には、PASSWORD() 関数は不要である。See 項4.4.8.「パスワードの設定」。

- localhost は、ユーザのローカルホスト名のシノニムである。また、ホストを明示的に指定せずにクライアントが接続しようとした場合のデフォルトホストでもある。ただし、MIT-pthread を使用するバージョン 3.23.27 より前の MySQL を使用している場合、localhost への接続は行われない (localhost 接続は Unix ソケットを使用して行われるが、これは当時の MIT-pthread にはサポートされていなかった)。このようなシステムでこの問題を回避するには、--host オプションを使用してサーバホストを明示的に指定する。これにより、mysqld サーバへの TCP/IP 接続が行われる。この場合、サーバホストの user テーブルエントリに実際のホスト名があることが必要である (これは、サーバと同じホスト上でクライアントプログラムを実行している場合も同様である)。

- mysql -u user_name db_name でデータベースに接続しようとして Access denied エラーが発生する場合、user テーブルに問題がある可能性がある。mysql -u root mysql を実行し、以下の SQL ステートメントを実行して、これをチェックする。

```
mysql> SELECT * FROM user;
```

この結果に、使用コンピュータのホスト名および MySQL ユーザ名とマッチする Host カラムおよび User カラムのエントリが含まれていなければならない。

- Access denied エラーメッセージには、ログインしようとしているユーザ名、接続元のホスト、およびパスワードを使用したかどうかが示される。通常、エラーメッセージで表示されたホスト名とユーザ名に完全にマッチするエントリが 1 つ、user テーブルに存在していなければならない。Using password: NO を含むエラーメッセージが表示された場合、パスワードなしでログインしようとしたことを意味する。

- MySQL サーバを実行しているホスト以外のホストから接続しようとして以下のエラーが発生する場合、そのホストと一致するレコードが user テーブルにないということである。

```
Host ... is not allowed to connect to this MySQL server
```

これを解決するには、コマンドラインツール mysql をサーバホスト上で使用して、user、db、または host テーブルに、接続しようとしているユーザとホスト名の組み合わせのレコードを追加し、mysqladmin flush-privileges を実行する。 実行している MySQL がバージョン 3.22 ではなく、接続しようとしているコンピュータの IP アドレスまたはホスト名がわからない場合、user テーブルの Host カラムに '%' を設定し、サーバマシン上で --log オプションを使用して mysqld を再起動する。クライアントマシンから接続を試行すると、MySQL ログの情報により、実際の接続がどのように行われたかわかる。次に、user テーブルエントリの '%' の値を、ログで表示された実際のホスト名に置き換える。そうしないと、システムのセキュリティを保てない。

Linux 上でこの問題が発生した場合、考えられる他の原因としては、使用している glibc とは違うバージョンの glibc でコンパイルされたバイナリ MySQL バージョンを使用していることが挙げられる。この場合、OS/glibc をアップグレードするか、ソース MySQL バージョンをダウンロードして自分でコンパイルする。ソース RPM は通常、コンパイルおよびインストールが簡単なので、これは大きな問題ではない。

- ホスト名で接続しようとしたにもかかわらず、エラーメッセージにホスト名が表示されない、またはホスト名が IP で

表示される場合

```
shell> mysqladmin -u root -pxxxx -h some-hostname ver
Access denied for user: 'root@' (Using password: YES)
```

これは、IP からホスト名に逆引きしようとしたときに MySQL でエラーが発生したことを意味する。この場合、mysqladmin flush-hosts を実行して内部 DNS キャッシュをリセットすることができる。 See 項5.5.5.「MySQL の DNS の使用」。

恒久的な解決方法

- DNS サーバでの問題点を見つけ、それを修正する。
- MySQL 権限テーブルで、ホスト名の代わりに IP を指定する。
- --skip-name-resolve オプションで mysqld を開始する。
- --skip-host-cache オプションで mysqld を開始する。
- 同じマシン上でサーバとクライアントを実行している場合は、localhost に接続する。
- /etc/hosts にクライアントマシン名を記入する。

- mysql -u root test は正常終了するのに mysql -h your_hostname -u root test では Access denied となる場合、user テーブルに正しいホスト名がない可能性がある。 通常、このような場合、user テーブルエントリの Host 値が完全修飾されていないホスト名を指定し、システムの名前解決ルーチンが完全修飾されたドメイン名を返している（またはその逆）。 たとえば、user テーブルにホスト 'tcx' のエントリがあるが、DNS に MySQL にホスト名として 'tcx.subnet.se' を指示した場合、これは機能しない。この場合、Host カラム値としてホストの IP アドレスを持つ user テーブルエントリを追加してみる。または、Host 値として 'tcx.%' などのワイルドカードを含む user テーブルエントリを追加することもできる。ただし、'%' で終わるホスト名の使用は、安全ではないので、推奨できない。
- mysql -u user_name test は正常終了するのに、mysql -u user_name other_db_name ではエラーが発生する場合、db テーブルに other_db_name のエントリがない。
- サーバマシンでは mysql -u user_name db_name が動作するのに、別のクライアントマシンでは mysql -h host_name -u user_name db_name が動作しない場合、user テーブルまたは db テーブルにクライアントマシンが登録されていない。
- Access denied の原因がわからない場合は、user テーブルから Host 値にワイルドカードが含まれているエントリ（'%' または '_' を含むエントリ）をすべて削除する。 よくある間違いは、Host='%' および User='some user' の新規エントリを挿入し、これで、同一マシンから接続する際には localhost を指定できると考えていることである。これがうまくいかない理由は、デフォルト権限として、Host='localhost' および User='' のエントリが含まれているためである。Host 値が 'localhost' の場合、'%' よりも具体的なため、localhost からの接続時に新しいエントリよりもデフォルト権限が優先される。正しい指定の方法は、Host='localhost' および User='some_user' の 2 つ目のエントリを挿入するか、Host='localhost' および User='' のエントリを削除することである。
- 以下のエラーが発生した場合、db テーブルまたは host テーブルに問題がある可能性がある。

```
Access to database denied
```

db テーブルの Host カラムに空白値がある場合、その db テーブルエントリに対してホストを指定するエントリが 1 つ

以上、host テーブルに存在していることを確認する。

SQL コマンド SELECT ...INTO OUTFILE または LOAD DATA INFILE 使用時にエラーが発生する場合、user テーブルの該当エントリで FILE 権限が有効化されていないと考えられる。

- クライアントプログラムは、オプション設定ファイルまたは環境変数で指定された接続パラメータを使用する。See 付録 F. 環境変数。 コマンドラインで接続パラメータを指定しなかったときに、クライアントが間違ったデフォルトの接続パラメータを送っているようであれば、ホームディレクトリの .my.cnf ファイルおよび環境変数を確認する。システムレベルの MySQL オプション設定ファイルも確認できるが、ここでクライアント接続パラメータが指定されている可能性は低い。See 項4.1.2.「my.cnf オプション設定ファイル」。 クライアントをオプションなしで実行して Access denied が発生する場合、オプション設定ファイルに古いパスワードが指定されていないか確認する。 See 項4.1.2.「my.cnf オプション設定ファイル」。

- 権限テーブルを（INSERT または UPDATE ステートメントを使用して）直接変更した場合、その変更は無視されたように見えることがある。サーバに権限テーブルを再読み込みさせるには、FLUSH PRIVILEGES ステートメントを発行するか mysqladmin flush-privileges コマンドを実行する必要がある。このことを行わない場合、変更は次回のサーバ再起動時まで有効とならない。UPDATE コマンドで root パスワードを設定しても、権限をフラッシュするまでは、パスワードが変更されたことをサーバがまだ認識しないため、そのパスワードを指定する必要はない。

- Perl、PHP、Ruby、Python、または ODBC プログラムでアクセスに問題があった場合、mysql -u user_name db_name または mysql -u user_name -pyour_pass db_name を使用して接続してみる。mysql クライアントを使用して接続できる場合、問題はプログラムにあり、アクセス権限にはない。 注意: -p とパスワードの間にスペースは入れない。また、--password=your_pass 構文を使用してパスワードを指定することもできる。-p オプションだけを使用した場合、パスワードが要求される。

- テスト用に、--skip-grant-tables オプションで mysqld デーモンを開始する。次に MySQL 権限テーブルを変更し、変更によって意図した結果が得られているかどうか mysqlaccess スクリプトを使用してチェックできる。変更が意図したとおりであれば、mysqladmin flush-privileges を実行し、新しい権限テーブルを使用して起動するように mysqld サーバに指示する。注意: 権限テーブルを再読み込みすると、--skip-grant-tables オプションが無効になる。これにより、サーバをシャットダウンして再起動することなく、権限テーブルの使用を始めることができる。

- 上記のいずれでもうまくいかない場合は、デバッグオプション（--debug=d,general,query など）で mysqld デーモンを開始する。試された接続に関するホスト情報、ユーザ情報、および発行された各コマンドに関する情報が出力される。See 項E.1.2.「トレースファイルの作成」。

- MySQL 権限テーブルに関するその他の問題が発生し、その旨メーリングリストに投稿する場合は、MySQL 権限テーブルのダンプも必ず提供すること。mysqldump mysql コマンドを使用すれば権限テーブルをダンプできる。問題を投稿するときは、必ず mysqlbug スクリプトを使用すること。See 項1.7.1.3.「バグまたは問題を報告する方法」。mysqldump を実行する際、--skip-grant-tables で mysqld を再起動することが必要な場合もある。

4.4. MySQL のユーザ管理

4.4.1. GRANT および REVOKE の構文

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)] ...]
  ON {tbl_name | * | *.* | db_name.*}
  TO user_name [IDENTIFIED BY [PASSWORD] 'password']
    [, user_name [IDENTIFIED BY [PASSWORD] 'password'] ...]
```

```
  [REQUIRE
     NONE |
     [{SSL| X509}]
     [CIPHER cipher [AND]]
     [ISSUER issuer [AND]]
     [SUBJECT subject]]
  [WITH [GRANT OPTION | MAX_QUERIES_PER_HOUR # |
                MAX_UPDATES_PER_HOUR # |
                MAX_CONNECTIONS_PER_HOUR #]]

REVOKE priv_type [(column_list)] [, priv_type [(column_list)] ...]
  ON {tbl_name | * | *.* | db_name.*}
  FROM user_name [, user_name ...]
```

GRANT は、MySQL バージョン 3.22.11 以降で実装されています。それより前の MySQL バージョンでは、GRANT ステートメントは何もしません。

システム管理者は、GRANT コマンドと REVOKE コマンドを使用して、ユーザを作成し、MySQL ユーザに対して次の 4 つのレベルの権限を与えたり取り消すことができます。

- グローバルレベル

  グローバル権限は特定のサーバ上のすべてのデータベースに適用される。これらの権限は、mysql.user テーブルに保存される。 GRANT ALL ON *.* および REVOKE ALL ON *.* は、グローバル権限の付与および取り消しのみを行う。

- データベースレベル

  データベース権限は、指定したデータベースのすべてのテーブルに適用される。これらの権限は、mysql.db テーブルおよび mysql.host テーブルに保存される。 GRANT ALL ON db.* および REVOKE ALL ON db.* は、データベース権限の付与および取り消しのみを行う。

- テーブルレベル

  テーブル権限は、指定したテーブルのすべてのカラムに適用される。これらの権限は、mysql.tables_priv テーブルに保存される。 GRANT ALL ON db.table および REVOKE ALL ON db.table は、テーブル権限の付与および取り消しのみを行う。

- カラムレベル

  カラム権限は、指定したテーブルの一つのカラムに適用される。これらの権限は、mysql.columns_priv テーブルに保存される。 REVOKE を使用する際は、対象となるカラムを指定することが必要である。

GRANT および REVOKE ステートメントの priv_type には以下のいずれかを指定できます。

ALL [PRIVILEGES]	WITH GRANT OPTION 以外のすべての権限を設定
ALTER	ALTER TABLE の使用を許可
CREATE	CREATE TABLE の使用を許可
CREATE TEMPORARY TABLES	CREATE TEMPORARY TABLE の使用を許可
DELETE	DELETE の使用を許可

DROP	DROP TABLE の使用を許可
EXECUTE	ストアドプロシージャの使用を許可（MySQL 5.0）
FILE	SELECT ... INTO OUTFILE および LOAD DATA INFILE の使用を許可
INDEX	CREATE INDEX および DROP INDEX の使用を許可
INSERT	INSERT の使用を許可
LOCK TABLES	SELECT 権限を持つテーブルで LOCK TABLES の使用を許可
PROCESS	SHOW FULL PROCESSLIST の使用を許可
REFERENCES	将来のために予約
RELOAD	FLUSH の使用を許可
REPLICATION CLIENT	スレーブおよびマスタのサーバーを知る権利を付与
REPLICATION SLAVE	レプリケーションのスレーブに必要（マスタからバイナリログを読み取るため）
SELECT	SELECT の使用を許可
SHOW DATABASES	SHOW DATABASES によりすべてのデータベースが表示される
SHUTDOWN	mysqladmin shutdown の使用を許可
SUPER	最大接続数に達していても接続を 1 つだけ許可し、コマンド CHANGE MASTER、KILL thread、mysqladmin debug、PURGE MASTER LOGS、および SET GLOBAL の実行を許可
UPDATE	UPDATE の使用を許可
USAGE	``権限なし'' のシノニム
GRANT OPTION	WITH GRANT OPTION のシノニム

USAGE を使用すると、権限なしのユーザを作成できます。

CREATE TEMPORARY TABLES、EXECUTE、LOCK TABLES、REPLICATION ...、SHOW DATABASES、および SUPER は、バージョン 4.0.2 で新しく導入された権限です。4.0.2 にアップグレードした後でこれらの権限を使用するには、mysql_fix_privilege_tables スクリプトを実行する必要があります。 See 項2.5.6.「権限テーブルのアップグレード」。

古い MySQL バージョンでは、PROCESS 権限が新しい SUPER 権限と同じ働きをします。

ユーザの GRANT 権限を取り消すには、次のように priv_type 値として GRANT OPTION を使用します。

```
mysql> REVOKE GRANT OPTION ON ... FROM ...;
```

テーブルに対して指定できる priv_type 値は、SELECT、INSERT、UPDATE、DELETE、CREATE、DROP、GRANT OPTION、INDEX、および ALTER だけです。

カラムに対して指定できる（つまり、column_list 節を使用する場合の）priv_type 値は、SELECT、INSERT、および UPDATE だけです。

MySQL では、データベースが存在しない場合でもデータベースレベルの権限を作成でき、データベース使用の準備を行えます。 ただし現在のところ、テーブルが存在しない場合にはテーブルレベルの権限は作成できません。テーブルやデータ

ベースを破棄した場合でも、MySQL が自動的に権限を取り消すことはありません。

グローバル権限は ON *.* 構文を使用して設定できます。データベース権限は ON db_name.* 構文を使用して設定できます。ON * を指定したときにカレントデータベースがあれば、そのデータベースの権限が設定されます。警告: ON * を指定したときにカレントデータベースがなければ、グローバル権限に影響します。

注意: GRANT コマンドでデータベース名を指定する際、'_' および '%' ワイルドカードを使用できます。データベース名の一部として、たとえば '_' 文字を使用したい場合、GRANT コマンドでは GRANT ... ON `foo\_bar`.* TO ... などのように、'_' として指定するようしてください。そうしないと、ワイルドカードパターンに一致する別のデータベースにもユーザがアクセスできるようになります。

MySQL では、任意のホストのユーザに権限を付与できるように、user_name 値を user@host 形式で指定できるようになっています。特殊文字 ('-' など) を含む user 文字列を指定したり、特殊文字またはワイルドカード文字 ('%' など) を含む host 文字列を指定したい場合、ユーザ名またはホスト名を引用符で囲むことができます (たとえば、'test-user'@'test-hostname') 。

ホスト名にワイルドカードを使用できます。たとえば、user@'%.loc.gov' は loc.gov ドメインのどのホストの user にも当てはまり、user@'144.155.166.%' は 144.155.166 クラス C のどのホストの user にも当てはまります。

シンプルな形式 user は user@"%" のシノニムです。

MySQL では、ユーザ名でのワイルドカード使用をサポートしていません。匿名ユーザの定義には、User='' のエントリを mysql.user テーブルに挿入するか、GRANT コマンドで空白名のユーザを作成します。

注意: 匿名ユーザが MySQL サーバに接続できるようにする場合、すべてのローカルユーザに対して、user@localhost として権限を設定する必要があります。そうしないと、ユーザがローカルマシンから MySQL サーバにログインしようとしたときに、mysql.user テーブルのローカルホストの匿名ユーザエントリが使用されるからです。

これを確認するには、以下のクエリを実行します。

```
mysql> SELECT Host,User FROM mysql.user WHERE User='';
```

現在のところ、GRANT ではホスト、テーブル、データベース、およびカラムの名前に最大 60 文字まで使用できます。ユーザ名は最大 16 文字までです。

テーブルまたはカラムに対する権限は、4 つのレベルの権限の論理和 (OR) で形成されます。たとえば、mysql.user テーブルでユーザにグローバル SELECT 権限が設定されている場合、これがデータベース、テーブル、またはカラムレベルのエントリで拒否されることはありません。

カラムに対する権限は以下のように計算されます。

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
```

多くの場合、ユーザへの権限設定には権限レベルを 1 つだけしか使用しないので、上述のように複雑にはなりません。権限チェックプロシージャの詳細については、項4.3.「一般的なセキュリティ関連事項と MySQL アクセス制御システム」を参照してください。

mysql.user テーブルに存在しないユーザとホスト名の組み合わせに権限を付与すると、そのエントリはテーブルに追加さ

れ、DELETE コマンドで削除されるまでそこに残ります。つまり、GRANT で user テーブルエントリを作成できますが、REVOKE ではそのエントリを削除できません。削除するには明示的に DELETE を使用する必要があります。

MySQL バージョン 3.22.12 以降では、新規ユーザが作成された場合、またはグローバル権限を付与する場合、IDENTIFIED BY 節でパスワードが指定されていれば、それがそのユーザのパスワードになります。すでにユーザにパスワードがある場合は、新しいパスワードに置き換えられます。

テキスト形式でパスワードを送信するのが望ましくなければ、PASSWORD オプションを使用して、SQL 関数 PASSWORD() または C API 関数 make_scrambled_password(char *to, const char *password) で生成した暗号化パスワードを設定できます。

警告: 新規ユーザを作成した際、IDENTIFIED BY 節を指定しなければそのユーザはパスワードなしになります。つまり、危険です。

パスワードの設定には、SET PASSWORD コマンドを使用することもできます。 See 項5.5.6.「SET 構文」。

データベースに対する権限を設定する場合、必要に応じて mysql.db テーブルにエントリが作成されます。REVOKE でそのデータベースのすべての権限が削除されると、このエントリも削除されます。

ユーザがテーブルに何の権限も持っていない場合、SHOW TABLES ステートメントなどでテーブルのリストを要求しても、テーブルは表示されません。SHOW DATABASES でも同様です。

WITH GRANT OPTION 節を使用すると、ユーザは自分が所有する権限を他のユーザに与えることができます。 GRANT 権限をユーザに設定する際は注意が必要です。2 人のユーザがお互いの権限を組み合わせることが可能になるからです。

MAX_QUERIES_PER_HOUR #、MAX_UPDATES_PER_HOUR #、および MAX_CONNECTIONS_PER_HOUR # は MySQL バージョン 4.0.2 で新しく導入されました。 これらのオプションは、1 時間でユーザが実行できるクエリ、更新、および口グインの回数を制限します。# が 0 (デフォルト) の場合、そのユーザに制限はないということです。 See 項4.4.7.「ユーザリソースの制限」。 注意: 既存ユーザに追加権限を設定することなくこれらのオプションを指定するには、GRANT USAGE ON *.* ... WITH MAX_... を使用します。

自分自身が持っていない権限を他のユーザに与えることはできません。GRANT 権限では、自分が所有する権限だけを他のユーザに与えることができます。

あるレベルの GRANT 権限をユーザに設定すると、そのユーザがそのレベルで所有する (および将来所有する) すべての権限を、そのユーザは他のユーザに設定できるということに注意してください。 あるユーザにデータベースに対する INSERT 権限を付与したと仮定します。次にデータベースの SELECT 権限を付与する際 WITH GRANT OPTION を指定した場合、そのユーザは SELECT 権限だけでなく INSERT 権限も他のユーザに与えることができます。さらにデータベースの UPDATE 権限をそのユーザに与えると、そのユーザは INSERT、SELECT、および UPDATE 権限を他のユーザに与えることができるようになります。

ALTER 権限は一般ユーザには付与しないでください。設定すると、そのユーザはテーブルの名前を変更して、権限システムを破ることができるようになります。

注意: 1 人のユーザだけにテーブル権限またはカラム権限を設定した場合でも、サーバはすべてのユーザのテーブル権限とカラム権限を調べるため、MySQL の処理速度は少し低下します。

mysqld の起動時、すべての権限がメモリに読み込まれます。 データベース権限、テーブル権限、およびカラム権限はすぐに反映されますが、ユーザレベルの権限はユーザが次回接続したときに有効となります。 GRANT または REVOKE によって行われた権限テーブルへの変更については、サーバは即座に認識します。 INSERT、UPDATE などを使って手動で権限テーブルを変更した場合、FLUSH PRIVILEGES ステートメントまたは mysqladmin flush-privileges を実行してサーバ

に権限テーブルを再読み込みさせる必要があります。 See 項4.4.3.「権限の変更はいつ反映されるか」。

GRANT の標準 SQL のバージョンと MySQL バージョンでの最大の違いは以下のとおりです。

- MySQL の権限は、ユーザ名のみではなく、ユーザ名とホスト名の組み合わせで指定される。
- SQL-99 にはグローバル権限およびデータベースレベルの権限がなく、MySQL がサポートするタイプの権限すべてをサポートしない。 MySQL は、SQL-99 の TRIGGER 権限と UNDER 権限をサポートしない。
- SQL-99 権限は、階層的に構成されている。あるユーザを削除すると、そのユーザに設定されているすべての権限が取り消される。MySQL では、設定された権限が自動的に取り消されることはなく、必要な場合は自分で取り消す必要がある。
- MySQL では、テーブルの一部のカラムで INSERT 権限が設定されている場合、そのテーブルに対して INSERT ステートメントを実行できる。この場合、INSERT 権限のないカラムにはデフォルト値が設定される。SQL-99 では、すべてのカラムで INSERT 権限が必要となる。
- SQL99 でテーブルを破棄すると、そのテーブルのすべての権限が取り消される。 SQL-99 では、1 つの権限を取り消すと、その権限をベースとするすべての権限も取り消される。MySQL で権限を破棄するには、明示的に REVOKE コマンドを使用するか MySQL 権限テーブルを操作することが必要である。

REQUIRE の使用法については、項4.4.10.「安全な接続の使用」 を参照してください。

4.4.2. MySQL のユーザ名とパスワード

MySQL でのユーザ名およびパスワードの使用法と、Unix および Windows での使用法にはいくつかの相違点があります。

- MySQL で認証目的に使用するパスワードは、Unix ユーザ名 (ログイン名) や Windows ユーザ名とは関係ない。ほとんどの MySQL クライアントはデフォルトで、現在の Unix ユーザ名を MySQL ユーザ名として ログインしようとするが、これは利便性のためだけである。 クライアントプログラムで、-u オプションまたは --user オプションにより別の名前を指定することが可能である。これは、すべての MySQL ユーザ名にパスワードを設定しないと、データベースを安全に保てないことを意味する。パスワードを設定しなければ、だれでも任意の名前でサーバへの接続を試みることができ、パスワードのない名前を指定すれば接続に成功してしまう。
- MySQL ユーザ名には最大 16 文字まで使用できるが、Unix ユーザ名は通常 8 文字まで。
- MySQL パスワードは、Unix パスワードと関係ない。Unix マシンにログインするパスワードと、そのマシンでデータベースにアクセスするためのパスワードには関連性がない。
- MySQL では、Unix ログインプロセスで使用されるアルゴリズムとは別のアルゴリズムで、パスワードを暗号化する。PASSWORD() 関数および ENCRYPT() 関数の詳細については、項6.3.6.2.「その他の各種関数」 を参照のこと。注意: パスワードが '暗号化' されて格納されていても、その '暗号化' されたパスワードを知るだけで MySQL サーバに接続できる。 バージョン 4.1 より、MySQL は従来とは異なるパスワードとログインメカニズムを採用しており、TCP/IP パケットがスニフされたり、mysql データベースがキャプチャされた場合でもセキュリティを保持できるようになっている。

MySQL ユーザおよびその権限は通常、GRANT コマンドで作成します。 See 項4.4.1.「GRANT および REVOKE の構文」。

MySQL サーバにコマンドラインクライアントでログインする場合は、--password=your-password でパスワードを指定してください。 See 項4.3.8.「MySQL サーバへの接続」。

```
mysql --user=monty --password=guess database_name
```

クライアントにパスワードを要求させたい場合には、引数なしで --password を使用します。

```
mysql --user=monty --password database_name
```

または次の短い形式

```
mysql -u monty -p database_name
```

注意: 上記の例で、パスワードが 'database_name' なわけではありません。

-p オプションを使用してパスワードを指定するには、以下のように行います。

```
mysql -u monty -pguess database_name
```

システムによっては、MySQL がパスワードを要求する際のライブラリコールにより、自動的にパスワードが 8 文字に切り詰められます。MySQL の内部では、パスワード長に制限はありません。

4.4.3. 権限の変更はいつ反映されるか

mysqld の起動時、すべての権限テーブルがメモリに読み込まれ、この時点で有効となります。

GRANT、REVOKE、または SET PASSWORD を使用して行った権限テーブルの変更は、すぐにサーバに認識されます。

INSERT、UPDATE などを使用して手動で権限テーブルを変更した場合、FLUSH PRIVILEGES ステートメント、mysqladmin flush-privileges、または mysqladmin reload を実行してサーバに権限テーブルを再読み込みさせる必要があります。そうしなければ、サーバを再起動するまで、変更は反映されません。権限テーブルを手動で変更した後、権限の再読み込みを忘れると、当然変更は反映されません。

サーバが権限テーブルの変更を認識すると、既存のクライアント接続は以下のような影響を受けます。

- テーブル権限とカラム権限の変更は、クライアントの次回の要求から反映される。
- データベース権限の変更は、次回の USE db_name コマンドから反映される。
- グローバル権限の変更およびパスワードの変更は、クライアントの次回の接続から反映される。

4.4.4. MySQL 権限の初期設定

MySQL のインストール後、scripts/mysql_install_db を実行してアクセス権限を初期設定します。 See 項2.3.1.「クイックインストールの概要」。 mysql_install_db スクリプトは mysqld サーバを起動し、以下の権限を含むように権限テーブルを初期化します。

- MySQL root ユーザが、すべての操作を実行できるスーパーユーザとして作成される。接続は、ローカルホストから行

う必要がある。

注意: 初期状態では、root パスワードは空白のため、だれでも root ユーザとしてパスワードなしで接続でき、すべての権限を得られる。

- 'test' という名前のデータベース、または 'test_' で始まる名前のデータベースにおいてすべての操作を実行できる匿名ユーザが作成される。接続は、ローカルホストから行う必要がある。どのローカルユーザでもパスワードなしで接続でき、その場合、匿名ユーザとして扱われることを意味する。
- その他の権限は拒否される。たとえば、一般ユーザは mysqladmin shutdown や mysqladmin processlist を使用できない。

注意: Windows のデフォルト権限とは異なります。 See 項2.1.1.8. 「Windows での MySQL の実行」。

初期インストール時はアクセスが開放されている状態なので、インストール後はまず、MySQL root ユーザにパスワードを設定してください。これは以下のように行います (注意: PASSWORD() 関数を使用してパスワードを指定します) 。

```
shell> mysql -u root mysql
mysql> SET PASSWORD FOR root@localhost=PASSWORD('new_password');
```

'new_password' に、使用するパスワードを指定します。

方法を理解していれば、権限テーブルを直接操作することもできます。

```
shell> mysql -u root mysql
mysql> UPDATE user SET Password=PASSWORD('new_password')
    ->     WHERE user='root';
mysql> FLUSH PRIVILEGES;
```

mysqladmin コマンドを使用しても、パスワードを設定できます。

```
shell> mysqladmin -u root password new_password
```

mysql データベースの書き込み権限および更新権限のあるユーザだけが、他のユーザのパスワードを変更できます。すべての一般ユーザ (匿名ユーザ以外) は、上記のコマンドまたは SET PASSWORD=PASSWORD('new_password') を使用して自分のパスワードだけを変更できます。

注意: UPDATE を使用して user テーブルで直接パスワードを更新する場合、FLUSH PRIVILEGES を使用してサーバに権限テーブルを再読み込みさせる必要があります。そうしないと変更が認識されません。

root パスワードをいったん設定するとそれ以降、root としてサーバに接続する際は常にそのパスワードを入力する必要があります。

追加設定やテストを行っている最中にパスワードを入力しないで済むように、root パスワードを空白のままにしたい場合もあるでしょう。しかし、実稼動で使用する前には必ずパスワードを設定してください。

デフォルト権限の設定については、scripts/mysql_install_db スクリプトを参照してください。このスクリプトは、他のユーザを追加する際にその基本として使用できます。

上記の初期権限とは違う初期権限を設定したい場合には、mysql_install_db を編集してから実行してください。

権限テーブルを完全に再生成するには、mysql データベースが含まれるディレクトリ内の .frm、.MYI、および .MYD ファイルをすべて削除します（このディレクトリは、データディレクトリの下にある mysql という名前のディレクトリです。データディレクトリは mysqld --help を実行すると表示されます）。そして、必要に応じて権限を編集してから、mysql_install_db スクリプトを実行します。

注意: MySQL バージョン 3.22.10 より前では、.frm ファイルを削除しないでください。うっかり削除してしまった場合は、mysql_install_db を実行する前に、MySQL ディストリビューションからコピーし直してください。

4.4.5. MySQL への新規ユーザの追加

ユーザの追加には、2 つの方法があります。GRANT ステートメントを使用する方法と、MySQL 権限テーブルを直接操作する方法です。正確でエラーが発生しにくい GRANT ステートメントの使用を推奨します。 See 項4.4.1.「GRANT および REVOKE の構文」。

ユーザの作成および管理に使用できるコントリビューションプログラムもいくつかあります（phpMyAdmin など）。

以下の例では、mysql クライアントを使用して新規ユーザを設定する方法を示します。この例では、前のセクションで説明したデフォルト設定に従って権限がセットアップされていることを前提にしています。そのため変更を行うには、mysqld を実行しているマシンにログインしていること、MySQL root ユーザとして接続していること、root ユーザに mysql データベースに対する INSERT 権限と RELOAD 管理者権限があることが必要条件となります。また、root ユーザパスワードを変更している場合、ここで、mysql コマンドを使用してそのパスワードを指定する必要があります。

まず、mysql プログラムを使用してサーバに MySQL root ユーザとして接続します。

```
shell> mysql --user=root mysql
```

次に、GRANT ステートメントを実行して新規ユーザを追加できます。

```
mysql> GRANT ALL PRIVILEGES ON *.* TO monty@localhost
    ->     IDENTIFIED BY 'some_pass' WITH GRANT OPTION;
mysql> GRANT ALL PRIVILEGES ON *.* TO monty@'%'
    ->     IDENTIFIED BY 'some_pass' WITH GRANT OPTION;
mysql> GRANT RELOAD,PROCESS ON *.* TO admin@localhost;
mysql> GRANT USAGE ON *.* TO dummy@localhost;
```

これらの GRANT ステートメントでは、以下の 3 人の新規ユーザが追加されます。

- monty

  どこからでもサーバに接続できる完全なスーパーユーザ。ただし、パスワード 'some_pass' の使用が必要。注意: GRANT ステートメントを monty@localhost と monty@"%" の両方に対して発行することが必要である。localhost のエントリを追加しないと、ローカルホストから接続したとき、mysql_install_db によって作成された localhost のエントリの方が、Host フィールド値がより具体的であり、user テーブルのソート順序で上位になるため優先されることになる。

- admin

  localhost からパスワードなしで接続でき、RELOAD および PROCESS の管理権限のあるユーザ。このユーザは、mysqladmin reload、mysqladmin refresh、mysqladmin flush-* の各コマンド、および mysqladmin processlist を実行できる。データベースレベルの権限は設定されていない（GRANT ステートメントを実行して後で追加できる）。

- dummy

  パスワードなしでローカルホストからのみ接続できるユーザ。権限は設定されていない。USAGE 権限タイプにより、このような権限なしのユーザを作成できる。これは、すべてのグローバル権限を 'N' に設定するのと同じことである。これは、後でこのアカウントに固有の権限を設定することを想定している。

INSERT ステートメントを発行してサーバに権限テーブルを再度読み込ませることにより、同じユーザアクセス情報を直接追加することもできます。

```
shell> mysql --user=root mysql
mysql> INSERT INTO user VALUES('localhost','monty',PASSWORD('some_pass'),
    ->        'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user VALUES('%','monty',PASSWORD('some_pass'),
    ->        '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;
```

MySQL バージョンによっては、上記 'Y' 値の数が違う場合があります (3.22.11 より前のバージョンでは権限カラムが少なく、4.0.2 以降では多くなります)。admin ユーザに対しては、バージョン 3.22.11 以降で利用可能になった SET を使用する拡張 INSERT 構文が使用されています。

注意: スーパーユーザを設定する場合は、権限フィールドを 'Y' に設定した user テーブルエントリを作成するだけで済みます。db テーブルや host テーブルのエントリは必要ありません。

最後の INSERT ステートメント (dummy ユーザ) では、user テーブルレコードの Host、User、および Password カラムだけが値を割り当てられています。権限カラムはどれも明示的に設定されていないため、MySQL はこれらすべてにデフォルト値の 'N' を設定します。これは、GRANT USAGE と同じ機能です。

以下の例では、ユーザ custom を追加します。このユーザは bankaccount データベースにはローカルホストからのみ、expenses データベースにはホスト whitehouse.gov からのみ、customer データベースにはホスト server.domain からのみアクセスできるとします。3 つのホストすべてでパスワード obscure を使用します。

GRANT ステートメントを使用してこのユーザの権限を設定するには、以下のコマンドを実行します。

```
shell> mysql --user=root mysql
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ->    ON bankaccount.*
    ->    TO custom@localhost
    ->    IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ->    ON expenses.*
    ->    TO custom@'whitehouse.gov'
    ->    IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ->    ON customer.*
    ->    TO custom@'server.domain'
    ->    IDENTIFIED BY 'obscure';
```

権限テーブルを直接変更してユーザの権限を設定するには、以下のコマンドを実行します (注意: 最後の FLUSH

PRIVILEGES に注目）。

```
shell> mysql --user=root mysql
mysql> INSERT INTO user (Host,User,Password)
    -> VALUES('localhost','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
    -> VALUES('whitehouse.gov','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
    -> VALUES('server.domain','custom',PASSWORD('obscure'));
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('server.domain','customer','custom','Y','Y','Y','Y','Y','Y');
mysql> FLUSH PRIVILEGES;
```

INSERT ステートメントを使用した前の例と同様、MySQL のバージョンによっては 'Y' の値の数が異なります。

最初の 3 つの INSERT ステートメントは user テーブルエントリを追加し、ユーザ custom がさまざまなホストから指定のパスワードで接続できるようにしますが、権限については何も設定していません（権限はすべて、デフォルト値の 'N' に設定されています）。次の 3 つの INSERT ステートメントは、db テーブルエントリを追加し、custom に bankaccount、expenses、customer の各データベースに対する権限を設定します。これらの権限は、適切なホストからアクセスしたときにだけ適用されるものです。権限テーブルを直接変更した場合、FLUSH PRIVILEGES を使用してサーバに権限テーブルを再度読み込ませ、権限の変更を反映させる必要があります。

特定のユーザに、あるドメイン（たとえば mydomain.com）のすべてのマシンからのアクセスを許可するには、以下のような GRANT ステートメントを実行します。

```
mysql> GRANT ...
    ->     ON *.*
    ->     TO myusername@'%.mydomain.com'
    ->     IDENTIFIED BY 'mypassword';
```

権限テーブルを直接変更して同じことを行うには、以下を実行します。

```
mysql> INSERT INTO user VALUES ('%.mydomain.com', 'myusername',
    ->            PASSWORD('mypassword'),...);
mysql> FLUSH PRIVILEGES;
```

4.4.6. MySQL ユーザの削除

```
DROP USER user_name
```

このコマンドは MySQL 4.1.1 で追加されたコマンドです。

このコマンドにより、権限を持たないユーザが削除されます。

MySQL からユーザを削除するには、以下の手順に従います。

1. SHOW PRIVILEGES を使用してユーザの権限を確認する。 See 項4.6.8.11.「SHOW PRIVILEGES」。
2. REVOKE を使用してユーザから権限をすべて削除する。 See 項4.4.1.「GRANT および REVOKE の構文」。
3. DROP USER を使用してユーザを削除する。

古い MySQL バージョンを使用している場合は、まず権限を取り消してからユーザを削除します。

```
mysql> DELETE FROM mysql.user WHERE user='username' and host='hostname';
mysql> FLUSH PRIVILEGES;
```

4.4.7. ユーザリソースの制限

MySQL 4.0.2 以降、特定のリソースをユーザごとに制限できるようになりました。

今までは、MySQL サーバリソースの使用を制限するには、スタートアップ変数の max_user_connections をゼロ以外の値に設定するしかありませんでした。しかしこの方法はグローバルに適用されるため、インターネットサービスプロバイダが関心のある個別ユーザの管理には使用できませんでした。

そのため、個別ユーザレベルで 3 つのリソースを管理する方法が導入されました。

- 時間単位の全クエリ数: 1 ユーザが実行できるクエリ。
- 時間単位の全更新数: テーブルまたはデータベースを変更するクエリ。
- 時間単位の接続数: 1 時間に新しく開かれる接続。

上記コンテキストの 1 ユーザとは user テーブルの 1 エントリです。このエントリは user カラムと host カラムによって識別されます。

制限を設定しない限り、すべてのユーザはデフォルトで、上記リソースを制限なく使用できます。これらの制限は、以下の構文を使用してグローバルな GRANT (*.*) でのみ設定できます。

```
GRANT ... WITH MAX_QUERIES_PER_HOUR N1
          MAX_UPDATES_PER_HOUR N2
          MAX_CONNECTIONS_PER_HOUR N3;
```

上記のリソースはどのような組み合わせでも指定可能です。 N1、N2、および N3 は整数で、時間単位のカウント数を表します。

ユーザが時間単位の接続数に達すると、その 1 時間が経過するまで、以降の接続は拒否されます。同様に、ユーザがクエリまたは更新の制限回数に達すると、その 1 時間が経過するまで、以降のクエリまたは更新は拒否されます。いずれの場合も、適切なエラーメッセージが表示されます。

特定ユーザの現在の使用値をフラッシュ（ゼロに設定）するには、上記の任意の節の GRANT ステートメントを発行しま

す。その際 GRANT ステートメントの値を現在の値にします。

また、すべてのユーザの現在の値をフラッシュするには、権限を再読み込みするか（サーバ内部で行うか、mysqladmin reload を使用）、FLUSH USER_RESOURCES コマンドを実行します。

リソース制限機能は、単一ユーザにリソースを制限する GRANT 節が設定されるとすぐに有効になります。

この機能を有効化する前提条件として、scripts サブディレクトリにあるテーブル作成スクリプト mysql_install_db および mysql_install_db.sh で定義されている追加カラムが、mysql データベースの user テーブルに含まれていることが必要です。

4.4.8. パスワードの設定

ほとんどの場合において、ユーザとパスワードの設定には GRANT を使用します。以下の方法は上級ユーザ対象です。
See 項4.4.1.「GRANT および REVOKE の構文」。

前のセクションでは、例を用いて重要な原則を示しました。INSERT または UPDATE ステートメントを使用して空白でないパスワードを保存する際には、PASSWORD() 関数を使用して暗号化しなければならないということです。これは、user テーブルがパスワードを平文テキストではなく、暗号化された形式で保存するためです。たとえば、この原則を忘れて、以下のようにパスワードを設定してしまったとします。

```
shell> mysql -u root mysql
mysql> INSERT INTO user (Host,User,Password)
    -> VALUES('%','jeffrey','biscuit');
mysql> FLUSH PRIVILEGES;
```

この場合、平文テキストの 'biscuit' が、パスワードとして user テーブルに保存されます。ユーザ jeffrey がこのパスワードを使用してサーバに接続しようとすると、mysql クライアントは PASSWORD() でそれを暗号化し、暗号化されたパスワードと、サーバから取得したランダム番号に基づいて認証ベクトルを生成し、それをサーバに送信します。 サーバは user テーブルの password 値（この場合、暗号化されていない 'biscuit'）を使用して同じ計算を実行し、結果を比較します。 比較は失敗し、サーバは接続を拒否します。

```
shell> mysql -u jeffrey -pbiscuit test
Access denied
```

user テーブルにパスワードを挿入するとき、パスワードは暗号化しておく必要があります。したがって、INSERT ステートメントを以下のように指定します。

```
mysql> INSERT INTO user (Host,User,Password)
    -> VALUES('%','jeffrey',PASSWORD('biscuit'));
```

SET PASSWORD ステートメントを使用するときも、PASSWORD() 関数を使用する必要があります。

```
mysql> SET PASSWORD FOR jeffrey@"%" = PASSWORD('biscuit');
```

GRANT ... IDENTIFIED BY ステートメントまたは mysqladmin password コマンドを使用してパスワードを設定する場合、PASSWORD() 関数は必要ありません。両方ともパスワードを暗号化するので、以下のようにパスワードを 'biscuit' と指定します。

```
mysql> GRANT USAGE ON *.* TO jeffrey@"%" IDENTIFIED BY 'biscuit';
```

または

```
shell> mysqladmin -u jeffrey password biscuit
```

注意: PASSWORD() は、Unix のパスワード暗号化とは異なります。 See 項4.4.2.「MySQL のユーザ名とパスワード」。

4.4.9. パスワードのセキュリティ

自分のパスワードを他のユーザがわかるような方法で指定することは危険です。 クライアントプログラムを実行する際、パスワードの指定に使用できる方法を以下に示します。それぞれの危険度についても示します。

- 一般ユーザに mysql.user テーブルへのアクセス権を与えてはいけない。暗号化されているパスワードがわかっただけで、そのユーザとしてログインすることが可能になる。パスワードの暗号化は、本当のパスワードを他人に見られないようにするためだけのものである（他のアプリケーションで似たパスワードを使用する場合に備えて）。
- -pyour_pass オプションまたは --password=your_pass オプションをコマンドラインで使用する。これは便利だが安全ではない。他のユーザがコマンドラインを表示するため起動する ps などのシステムステータスプログラムにはパスワードが見えてしまう（MySQL クライアントは通常、初期化シーケンスにおいてコマンドライン引数をゼロで上書きするが、短いインターバルがあってその間は見えてしまう）。
- -p オプションまたは --password オプションを使用する（your_pass 値を指定しない）。この場合、クライアントプログラムは端末からのパスワード入力を求める。

  ```
  shell> mysql -u user_name -p
  Enter password: ********
  ```

  '*' 文字はパスワードを表す。

  この方法だと他のユーザには見えないため、コマンドラインでパスワードを指定するよりも安全である。ただし、このパスワード入力方法は、対話式で実行するプログラムにのみ適している。非対話式で実行するスクリプトからクライアントを起動する場合、端末からパスワードを入力する機会がない。システムによっては、スクリプトの最初の行を間違ってパスワードと解釈してしまうものもある。
- 設定ファイルにパスワードを保存する。たとえば、自分のホームディレクトリにある .my.cnf ファイルの [client] セクションにパスワードを記入できる。

  ```
  [client]
  password=your_pass
  ```

  パスワードを .my.cnf に保存する場合、だれもそのファイルを読み書きできないようにしておくことが必要である。ファイルのアクセスモードは 400 または 600 にしておくこと。 See 項4.1.2.「my.cnf オプション設定ファイル」。
- MYSQL_PWD 環境変数にパスワードを保存できる。しかしこの方法は非常に危険なので、使用すべきではない。 ps のバージョンによっては、実行中のプロセス環境を表示するオプションがある。MYSQL_PWD を設定していればだれにでもパスワードを見られてしまう。そのようなバージョンの ps がないシステムでも、プロセス環境を表示する他の方法がないと想定するのは危険である。 See 付録 F. 環境変数。

以上のことを総括すると、最も安全な方法は、クライアントプログラムにパスワードを要求させるか、適切に保護された .my.cnf ファイルにパスワードを指定することです。

4.4.10. 安全な接続の使用

4.4.10.1. 基本概念

バージョン 4.0.0 より、MySQL は SSL 暗号化接続をサポートしています。MySQL がどのように SSL を使用するか理解するには、SSL と X509 の基本概念が欠かせません。基本概念を理解している読者は、このセクションを読み飛ばしてください。

デフォルトでは、MySQL はクライアントとサーバ間で暗号化されていない接続を使用します。つまり、他人がすべてのトラフィックおよび送受信データを盗み見ることが可能な接続です。また、クライアントとサーバ間で転送中のデータを改ざんすることも可能です。公開ネットワークでの情報のやり取りでも安全性が求められる場合があります。そのような場合、暗号化されていない接続は適しません。

SSL とは、複数の異なる暗号化アルゴリズムを使用して、公開ネットワークで受信するデータの信頼性を保証するためのプロトコルです。データの変更、消失、および再生を検知するメカニズムがあります。SSL にはまた、X509 規格を使用する ID 認証を認識および提供するためのアルゴリズムも組み込まれています。

暗号化とは、データを読み取り不可能にする技術です。実際、今日の社会では、暗号化アルゴリズムによるさらなるセキュリティが求められています。暗号化メッセージの順番を変更したり同じデータを 2 回再生するなど、さまざまな種類の攻撃に対する耐性が必要となっています。

X509 とは、インターネット上で ID 認証を可能にする規格です。これは、電子商取引アプリケーションで最も一般的に使用されています。基本的には、``認証局" と呼ばれる会社が電子証明書を必要とする者に割り当てるという方法を取ります。証明書は、2 つの暗号化キー (公開キーと秘密キー) がある非対称暗号化アルゴリズムを使用しています。証明書の所有者は、他者に証明書を提示して自分の ID を証明します。証明書には、所有者の公開キーが含まれています。この公開キーで暗号化されたデータはすべて、対応する秘密キーがなければ解読できません。秘密キーは証明書の所有者が保持しています。

MySQL はデフォルトでは暗号化接続を使用しません。使用すると、クライアントとサーバのプロトコルがかなり遅くなるからです。どのような機能を追加した場合でもコンピュータには負荷がかかります。データの暗号化も例外ではなく、CPU を大幅に消費して時間がかかり、MySQL の主要タスクを遅らせる原因となります。そのためデフォルトでは、MySQL は速度が優先されています。

SSL、X509、および暗号化についてさらに詳細を知るには、インターネット検索エンジンを利用して必要な情報を検索してください。

4.4.10.2. 必要条件

安全な接続を MySQL で使用するには、以下を実行する必要があります。

1. OpenSSL ライブラリをインストールする。OpenSSL 0.9.6 をインストールした MySQL はテスト済み。http://www.openssl.org/。
2. MySQL を --with-vio --with-openssl でコンフィギャする。
3. 古い MySQL バージョンを使用している場合、mysql.user テーブルに新しい SSL 関連カラムを追加する必要がある。この処理は、権限テーブルが MySQL 4.0.0 より前のバージョンの場合に必要となる。手順については 項2.5.6.「権限テーブルのアップグレード」 を参照のこと。
4. 実行中の mysqld サーバが OpenSSL をサポートしているかチェックするには、SHOW VARIABLES LIKE

'have_openssl' が YES を返すかどうかを確認する。

4.4.10.3. MySQL での SSL 証明書の設定

以下、MySQL での SSL 証明書の設定例です。

```
DIR=`pwd`/openssl
PRIV=$DIR/private

mkdir $DIR $PRIV $DIR/newcerts
cp /usr/share/ssl/openssl.cnf $DIR
replace ./demoCA $DIR -- $DIR/openssl.cnf

# Create necessary files: $database, $serial and $new_certs_dir
# directory (optional)

touch $DIR/index.txt
echo "01" > $DIR/serial

#
# Generation of Certificate Authority(CA)
#

openssl req -new -x509 -keyout $PRIV/cakey.pem -out $DIR/cacert.pem \
   -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# .............++++++
# .........++++++
# writing new private key to '/home/monty/openssl/private/cakey.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be incorporated
# into your certificate request.
# What you are about to enter is what is called a Distinguished Name or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL admin
# Email Address []:

#
# Create server request and key
#
openssl req -new -keyout $DIR/server-key.pem -out \
   $DIR/server-req.pem -days 3600 -config $DIR/openssl.cnf
```

```
# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ..++++++
# .........++++++
# writing new private key to '/home/monty/openssl/server-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be incorporated
# into your certificate request.
# What you are about to enter is what is called a Distinguished Name or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL server
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove the passphrase from the key (optional)
#

openssl rsa -in $DIR/server-key.pem -out $DIR/server-key.pem

#
# Sign server cert
#
openssl ca  -policy policy_anything -out $DIR/server-cert.pem \
   -config $DIR/openssl.cnf -infiles $DIR/server-req.pem

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName           :PRINTABLE:'FI'
# organizationName      :PRINTABLE:'MySQL AB'
# commonName            :PRINTABLE:'MySQL admin'
# Certificate is to be certified until Sep 13 14:22:46 2003 GMT (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated
```

```
#
# Create client request and key
#
openssl req -new -keyout $DIR/client-key.pem -out \
   $DIR/client-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# .....++++++
# .....++++++
# writing new private key to '/home/monty/openssl/client-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be incorporated
# into your certificate request.
# What you are about to enter is what is called a Distinguished Name or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL user
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove a passphrase from the key (optional)
#
openssl rsa -in $DIR/client-key.pem -out $DIR/client-key.pem

#
# Sign client cert
#

openssl ca  -policy policy_anything -out $DIR/client-cert.pem \
   -config $DIR/openssl.cnf -infiles $DIR/client-req.pem

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName           :PRINTABLE:'FI'
# organizationName      :PRINTABLE:'MySQL AB'
# commonName            :PRINTABLE:'MySQL user'
# Certificate is to be certified until Sep 13 16:45:17 2003 GMT (365 days)
# Sign the certificate? [y/n]:y
```

```
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create a my.cnf file that you can use to test the certificates
#

cnf=""
cnf="$cnf [client]"
cnf="$cnf ssl-ca=$DIR/cacert.pem"
cnf="$cnf ssl-cert=$DIR/client-cert.pem"
cnf="$cnf ssl-key=$DIR/client-key.pem"
cnf="$cnf [mysqld]"
cnf="$cnf ssl-ca=$DIR/cacert.pem"
cnf="$cnf ssl-cert=$DIR/server-cert.pem"
cnf="$cnf ssl-key=$DIR/server-key.pem"
echo $cnf | replace " " '
' > $DIR/my.cnf

#
# To test MySQL

mysqld --defaults-file=$DIR/my.cnf &

mysql --defaults-file=$DIR/my.cnf
```

mysql-source-dist/SSL ディレクトリにあるデモ証明書を参照するように上記の my.cnf ファイルを修正することによっても、設定をテストできます。

4.4.10.4. SSL GRANT オプション

MySQL は、通常のユーザ名とパスワードのスキームに加え、X509 証明書属性をチェックすることができます。通常のオプションもすべて必要です (ユーザ名、パスワード、IP アドレスマスク、データベース/テーブル名) 。

接続の制限にはいくつかのシナリオがあります。

- SSL または X509 オプションがない場合、ユーザ名とパスワードが正しければ、暗号化および非暗号化すべての接続が許可される。
- REQUIRE SSL オプションは、サーバが SSL 暗号化接続のみを許可するように制限する。 注意: 非 SSL 接続を許可する ACL レコードがある場合、このオプションは除外できる。

```
mysql> GRANT ALL PRIVILEGES ON test.* TO root@localhost
    -> IDENTIFIED BY 'goodsecret' REQUIRE SSL;
```

- REQUIRE X509 は、クライアントに有効な証明書が必要なことを意味するが、どの証明書、発行者、およびサブジェクトでなければいけないという指定はない。 CA 証明書の署名を確認できれば、それで十分である。

```
mysql> GRANT ALL PRIVILEGES ON test.* TO root@localhost
    -> IDENTIFIED BY 'goodsecret' REQUIRE X509;
```

- REQUIRE ISSUER 'issuer' は、接続試行に制限を加える。 クライアントは、CA 'issuer' によって発行された有効な X509 証明書を提示する必要がある。 X509 証明書を使用することは、暗号化の使用を意味しており、SSL オプションは不要である。

```
mysql> GRANT ALL PRIVILEGES ON test.* TO root@localhost
    -> IDENTIFIED BY 'goodsecret'
    -> REQUIRE ISSUER 'C=FI, ST=Some-State, L=Helsinki,
    '> O=MySQL Finland AB, CN=Tonu Samuel/Email=tonu@mysql.com';
```

- REQUIRE SUBJECT 'subject' では、サブジェクト 'subject' が有効な X509 証明書がクライアントに必要となる。クライアントが有効な証明書を提示しても、'subject' が異なる場合、接続は拒否される。

```
mysql> GRANT ALL PRIVILEGES ON test.* TO root@localhost
    -> IDENTIFIED BY 'goodsecret'
    -> REQUIRE SUBJECT 'C=EE, ST=Some-State, L=Tallinn,
    '> O=MySQL demo client certificate,
    '> CN=Tonu Samuel/Email=tonu@mysql.com';
```

- REQUIRE CIPHER 'cipher' は、強力な暗号とキー長の使用を義務付ける。短い暗号化キーの古いアルゴリズムを使用した場合、SSL 自体も強力ではなくなる。このオプションを使用することにより、接続許可の条件として特定の暗号化を指定できる。

```
mysql> GRANT ALL PRIVILEGES ON test.* TO root@localhost
    -> IDENTIFIED BY 'goodsecret'
    -> REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

SUBJECT、ISSUER、および CIPHER のオプションは、以下のように REQUIRE 節で組み合わせることができる。

```
mysql> GRANT ALL PRIVILEGES ON test.* TO root@localhost
    -> IDENTIFIED BY 'goodsecret'
    -> REQUIRE SUBJECT 'C=EE, ST=Some-State, L=Tallinn,
    '> O=MySQL demo client certificate,
    '> CN=Tonu Samuel/Email=tonu@mysql.com'
    -> AND ISSUER 'C=FI, ST=Some-State, L=Helsinki,
    '> O=MySQL Finland AB, CN=Tonu Samuel/Email=tonu@mysql.com'
    -> AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

MySQL 4.0.4 以降、REQUIRE オプション間の AND キーワードは任意になっている。

オプションはどんな順序でも指定できるが、同じオプションを 2 回指定することはできない。

4.4.10.5. SSL コマンドラインオプション

以下の表は、SSL、証明書ファイル、およびキーファイルの使用を指定するためのオプションをまとめたものです。 これらのオプションは MySQL 4.0 から導入されたものです。これらのオプションは、コマンドラインでもオプション設定ファイルでも使用できます。

- --ssl

  サーバに対しては、サーバが SSL 接続を許可するように指定する。 クライアントプログラムに対しては、SSL を使用してサーバに接続することを許可する。 このオプションだけでは SSL 接続の使用は保証されない。 --ssl-ca、--ssl-cert

、および --ssl-key オプションも指定する必要がある。

注意: このオプションでは、SSL 接続が必須とならない。 たとえば、サーバまたはクライアントが SSL サポートなしでコンパイルされている場合、通常の暗号化されていない接続が使用される。

SSL 接続だけの使用を確実に保証するには、まず、GRANT ステートメントで REQUIRE SSL 節を含むアカウントをサーバに作成することが必要。 そしてこのアカウントを使用してサーバに接続する。このとき、サーバとクライアントの両方で SSL サポートが有効になっていることが必要である。

このオプションを使用して、接続に SSL を使用しないように指定することができる。 この場合、オプションを --skip-ssl または --ssl=0 として指定する。

- --ssl-ca=file_name

  信頼された SSL 認証局の一覧が含まれるファイルのパス。

- --ssl-capath=directory_name

  PEM 形式の信頼された CA 証明書が保存されているディレクトリのパス。

- --ssl-cert=file_name

  安全な接続を確立するために使用する SSL 証明書ファイルの名前。

- --ssl-cipher=cipher_list

  SSL 暗号化に使用できる暗号の一覧。 cipher_list は、openssl ciphers コマンドと同じ形式である。

  例: --ssl-cipher=ALL:-AES:-EXP

- --ssl-key=file_name

  安全な接続を確立するために使用する SSL キーファイルの名前。

4.5. 障害の予防とリカバリ

4.5.1. データベースのバックアップ

MySQL テーブルはファイルとして保存されるため、バックアップが簡単です。一貫したバックアップを行うには、そのテーブルに対して LOCK TABLES を実行してから、FLUSH TABLES を実行します。 See 項6.7.5.「LOCK TABLES および UNLOCK TABLES 構文」。 See 項4.6.4.「FLUSH 構文」。 読み取りロックだけが必要なので、データベースディレクトリのファイルをコピーしている間も、他のスレッドはテーブルに対してクエリを続行できます。FLUSH TABLE は、バックアップを開始する前に、キャッシュされているページをすべてディスクに書き込むために必要です。

3.23.56 から 4.0.12 では、セキュリティ上のリスクがあるため、BACKUP TABLE で既存ファイルの上書きはできないようになっています。

テーブルを SQL レベルでバックアップするには、SELECT INTO OUTFILE または BACKUP TABLE を使用します。 See 項6.4.1.「SELECT 構文」。 See 項4.5.2.「BACKUP TABLE 構文」。

データベースのバックアップには、mysqldump プログラムまたは mysqlhotcopy script を使用することもできます。 See

項4.9.7.「mysqldump（テーブル構造とデータのダンプ）」。 See 項4.9.8.「mysqlhotcopy（MySQL のデータベースとテーブルのコピー）」。

1. データベースをフルバックアップする。

```
shell> mysqldump --tab=/path/to/some/dir --opt db_name

または

shell> mysqlhotcopy db_name /path/to/some/dir
```

   サーバが何かを更新中でなければ、すべてのファイル（*.frm、*.MYD、および *.MYI）を単にコピーすることもできる。 スクリプト mysqlhotcopy はこの方法を使用している。 注意: データベースに InnoDB テーブルが含まれている場合、InnoDB はテーブルコンテンツをデータベースディレクトリに保存しないため、この方法は使用できない。 mysqlhotcopy は MyISAM テーブルと ISAM テーブルにのみ有効。

2. mysqld が実行中であればいったん停止し、--log-bin[=file_name] オプションで再起動する。See 項4.10.4.「バイナリログ」。 mysqldump 実行後に行われたデータの変更をレプリケートするための情報が、バイナリログファイルにより提供される。

MySQL サーバがスレーブの場合、どのバックアップ方法を選択しても、スレーブのデータをバックアップするとき、master.info ファイルと relay-log.info ファイルもバックアップしてください。これらのファイルは、スレーブのデータをリストア後、レプリケーションを再開するときに必要です。スレーブが、レプリケーションを行う LOAD DATA INFILE コマンドの対象となっていた場合、SQL_LOAD-* ファイルもバックアップしてください。このファイルは、--slave-load-tmpdir オプションによって指定されているディレクトリにあります（このファイルの場所が指定されていない場合、デフォルトで、この場所は tmpdir 変数の値になります）。中断されていた LOAD DATA INFILE 処理のレプリケーションを再開するためにスレーブはこれらのファイルを必要とします。

何かをリストアする必要がある場合、最初に REPAIR TABLE または myisamchk -r を使用してテーブルのリカバリを試みてください。ほとんどの場合、これで成功するはずです。myisamchk でリカバリできなかった場合、以下を実行します。これは、--log-bin で MySQL を起動している場合にのみ可能な方法です。項4.10.4.「バイナリログ」 を参照してください。

1. オリジナルの mysqldump バックアップ、またはバイナリバックアップをリストアする。
2. 以下のコマンドを実行して、バイナリログ内の更新を再実行する。

```
shell> mysqlbinlog hostname-bin.[0-9]* | mysql
```

   場合に応じて、特定の位置から後のバイナリログだけを再実行する（通常は、リストアしたバックアップの日付以降の、不正なクエリ以外のすべてのバイナリログを再実行する）。 mysqlbinlog ユーティリティおよびその使用法の詳細については、項4.9.5.「mysqlbinlog（バイナリログからクエリを実行する）」 を参照のこと。

   更新ログ（MySQL 5.0 で廃止）を使用している場合、更新ログの内容を以下のように実行できる。

```
shell> ls -1 -t -r hostname.[0-9]* | xargs cat | mysql
```

   ls は、すべての更新ログファイルを正しい順序で取得するために使用します。

SELECT * INTO OUTFILE 'file_name' FROM tbl_name で選択的バックアップを行い、LOAD DATA INFILE 'file_name' REPLACE ... でリストアすることもできます。重複テーブルを避けるため、テーブルに PRIMARY KEY または UNIQUE キーが必要です。REPLACE キーワードを使用すると、ユニークキーが新規レコードと旧レコードで重複していた場合に、旧レコードが新規レコードで置き換えられます。

バックアップに伴いパフォーマンス上の問題が発生する場合、レプリケーションをセットアップして、マスタではなくスレーブ上でバックアップを実行することによりこの問題を解決できる。 See 項4.11.1.「はじめに」。

Veritas ファイルシステムを使用している場合、以下を実行できます。

1. クライアント (または Perl) から、FLUSH TABLES WITH READ LOCK を実行する。
2. 別のシェルから、mount vxfs snapshot を実行する。
3. 最初のクライアントから、UNLOCK TABLES を実行する。
4. スナップショットからファイルをコピーする。
5. スナップショットのマウントを解除する。

4.5.2. BACKUP TABLE 構文

```
BACKUP TABLE tbl_name[,tbl_name...] TO '/path/to/backup/directory'
```

ディスクにバッファされている変更をフラッシュした後で、テーブルをリストアするのに最低限必要な数のテーブルファイルをバックアップディレクトリにコピーします。現在のところ、このコマンドは MyISAM テーブルにのみ有効です。MyISAM テーブルに対して、.frm 定義ファイルと .MYD データファイルをコピーします。これら 2 つのファイルからインデックスファイルを再度ビルドできます。

このコマンドを使用する前に、項4.5.1.「データベースのバックアップ」 を参照してください。

バックアップ中、各テーブルが処理されている間そのテーブルにのみ読み取りロックがかかります。複数のテーブルを 1 回のスナップショットでバックアップする場合は、まず LOCK TABLES を実行して、そのグループ内の各テーブルに対して読み取りロックをかけておく必要があります。

このコマンドは、以下のカラムで構成されるテーブルを返します。

カラム	値
Table	テーブル名
Op	常に backup
Msg_type	status、error、info、warning のいずれか
Msg_text	メッセージ

注意: BACKUP TABLE は、MySQL バージョン 3.23.25 以降でのみ使用できます。

4.5.3. RESTORE TABLE 構文

```
RESTORE TABLE tbl_name[,tbl_name...] FROM '/path/to/backup/directory'
```

BACKUP TABLE で作成されたバックアップから、1 つまたは複数のテーブルをリストアします。既存テーブルには上書きされません。既存テーブルの上にリストアしようとすると、エラーになります。インデックスを再度ビルドする必要があるため、リストアはバックアップよりも時間がかかります。キーが多ければ多いほど、時間がかかります。BACKUP TABLE と同じく、RESTORE TABLE は現在のところ、MyISAM テーブルにのみ有効です。

このコマンドは、以下のカラムで構成されるテーブルを返します。

カラム	値
Table	テーブル名
Op	常に restore
Msg_type	status、error、info、warning のいずれか
Msg_text	メッセージ

4.5.4. CHECK TABLE 構文

```
CHECK TABLE tbl_name[,tbl_name...] [option [option...]]

option = QUICK | FAST | MEDIUM | EXTENDED | CHANGED
```

CHECK TABLE は、MyISAM テーブルと InnoDB テーブルにのみ有効です。MyISAM テーブルの場合、このテーブルで myisamchk --medium-check table_name を実行するのと同じです。

オプションを何も指定しなければ、MEDIUM が使用されます。

1 つまたは複数のテーブルのエラーをチェックします。MyISAM テーブルでは、キー統計が更新されます。このコマンドは、以下のカラムで構成されるテーブルを返します。

カラム	値
Table	テーブル名
Op	常に check
Msg_type	status、error、info、warning のいずれか
Msg_text	メッセージ

注意: このステートメントは、チェックした各テーブルに関する多くの情報レコードを生成します。 正常な場合、Msg_type は status で、Msg_text は通常 OK になります。 OK または Table is already up to date が得られなければ、テーブルの修復が必要と考えられます。See 項4.5.6.「myisamchk を使用したテーブルの保守とクラッシュのリカバリ」。 Table is already up to date は、テーブルのストレージマネージャが、そのテーブルにチェックが必要ないと判断したことを意味します。

以下のチェックタイプがあります。

タイプ	意味

QUICK	不正リンクをチェックするためのレコードスキャンを実行しない。
FAST	正しく閉じられなかったテーブルだけをチェックする。
CHANGED	前回のチェック後に変更されたテーブルと、正しく閉じられなかったテーブルだけをチェックする。
MEDIUM	レコードをスキャンして、削除されたリンクが問題なかったことを確認する。また、レコードのキーチェックサムを計算し、キーの計算されたチェックサムを使って確認する。
EXTENDED	各レコードですべてのキーの完全キールックアップを実行する。テーブルの整合性が 100% 保証されるが、時間がかかる。

動的サイズの MyISAM テーブルでは、チェックが開始された場合、常に MEDIUM チェックが実行されます。静的サイズのレコードは破損していることが非常にまれなので、QUICK および FAST でレコードスキャンをスキップします。

チェックオプションは以下のように組み合わせることができます。以下の例では、テーブルが正しく閉じられたかどうかを確認するクイックチェックを行います。

```
CHECK TABLE test_table FAST QUICK;
```

注意: 場合によっては CHECK TABLE はテーブルを変更します。これは、テーブルに 'corrupted' または 'not closed properly' というマークがあるのに、CHECK TABLE がテーブル内に問題を発見しなかった場合に発生します。この場合、CHECK TABLE はテーブルを OK とマークします。

テーブルが破損している場合、データ部分よりもインデックスに問題のある方が可能性が高いです。上記のチェックタイプはいずれも、インデックスを完全にチェックするのでほとんどのエラーは発見できます。

問題がないと思われるテーブルをチェックする場合、チェックオプションを使用しないか、または QUICK オプションを使用します。QUICK は、急いでいるとき、およびデータファイルのエラーが見逃されるかもしれないという非常に小さな危険性を無視できる場合に使用します。ほとんどの場合、MySQL は通常どおりに使用していれば、データファイルのエラーを発見します。発見した場合、そのテーブルには 'corrupted' のマークが付き、修復するまでそのテーブルは使用できません。

FAST および CHANGED は、テーブルをときどきチェックしたい場合に、スクリプトから (たとえば cron から) 実行されることを想定しています。通常は、CHANGED よりも FAST の方を使用してください (これが当てはまらないのは、MyISAM コード内にバグが疑われる場合のみです)。

EXTENDED は、通常のチェック後に、MySQL がレコードを更新したりキーでレコードを検索しようとした際に、不自然なエラーが発生する場合にのみ使用します (通常のチェックが正常に終了した後でエラーが発生するのは非常にまれなことです)。

CHECK TABLE で報告される問題の中には、自動的に修正されないものがあります。

- Found row where the auto_increment column has the value 0。

  これは、AUTO_INCREMENT インデックスカラムの値が 0 になっているレコードがテーブルにあることを意味する (UPDATE ステートメントで明示的に AUTO_INCREMENT カラム値を 0 に設定すれば、このカラムが 0 のレコードを作成することは可能である)。

  これ自体はエラーではないが、テーブルをダンプしてリストアしようとしたり、テーブルで ALTER TABLE を実行しようとすると問題が発生する。この場合、AUTO_INCREMENT カラムは AUTO_INCREMENT カラムのルールに基づ

いて値を変更するため、重複キーエラーなどの原因となる。

警告を消すには、UPDATE ステートメントを実行してこのカラム値を 0 以外の値に設定する。

4.5.5. REPAIR TABLE 構文

```
REPAIR [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name[,tbl_name...] [QUICK] [EXTENDED] [USE_FRM]
```

REPAIR TABLE は MyISAM テーブルにのみ有効であり、このテーブルで myisamchk -r table_name を実行するのと同じです。

通常であれば、このコマンドを使用することはありませんが、障害が発生した場合、REPAIR TABLE を使用すれば、ほとんどの場合、MyISAM テーブルのすべてのデータを取り戻せます。テーブルが頻繁に破損するようであれば、その原因を突き止めて、REPAIR TABLE を使用する必要がなくなるようにしてください。 See 項A.4.1.「MySQL が何度もクラッシュする場合に行うこと」。 See 項7.1.3.「MyISAM テーブルの問題」。

REPAIR TABLE は、破損した可能性のあるテーブルを修復します。このコマンドは、以下のカラムで構成されるテーブルを返します。

カラム	値
Table	テーブル名
Op	常に repair
Msg_type	status、error、info、warning のいずれか
Msg_text	メッセージ

注意: このステートメントは、修復した各テーブルに関する多くの情報レコードを生成します。 正常な場合、Msg_type は status で、Msg_text は通常 OK になります。OK を得られない場合は、myisamchk --safe-recover でテーブルの修復を試みてください。REPAIR TABLE ではまだ myisamchk のすべてのオプションをカバーしていません。将来は、このコマンドをより柔軟性の高いものにする予定です。

QUICK を指定した場合、REPAIR TABLE はインデックスツリーだけを修復しようとします。

EXTENDED を使用すると、MySQL はソートごとにインデックスを生成するのではなく、レコードごとにインデックスを生成します。確実に圧縮される長い CHAR キーを使用している場合など、固定長キーをソートするよりこの方法の方が適しています。このタイプの修復は、myisamchk --safe-recover で実行される修復とほぼ同じです。

MySQL 4.0.2 から、REPAIR に USE_FRM モードが導入されています。 .MYI ファイルがない、またはそのヘッダが破損している場合にこのモードを使用します。 このモードでは、.frm ファイルの情報を使用してテーブルが再度作成されます。このような修復は、myisamchk ではできません。

警告: REPAIR TABLE を実行中に mysqld が終了してしまった場合、他のコマンドを実行する前にもう 1 回 REPAIR を必ず実行してください (もちろん、常にバックアップから開始することを推奨します)。最悪の場合、データファイルに関する情報がない新規のクリーンなインデックスファイルができることがあり、次のコマンドでデータファイルが上書きされてしまう可能性があります。これはあまりないことではありますが、可能性としてはあり得るので、注意が必要です。

MySQL 4.1.1 より前では、REPAIR コマンドはバイナリログに書き込まれません。MySQL 4.1.1 以降、任意の NO_WRITE_TO_BINLOG キーワード (またはその エイリアスの LOCAL) を使用しない限り、このコマンドはバイナリロ

グに書き込まれます。

4.5.6. myisamchk を使用したテーブルの保守とクラッシュのリカバリ

MySQL バージョン 3.23.13 以降、CHECK TABLE コマンドを使用して MyISAM テーブルをチェックできるようになりました。See 項4.5.4.「CHECK TABLE 構文」。 テーブルの修復には REPAIR TABLE コマンドを使用できます。 See 項4.5.5.「REPAIR TABLE 構文」。

MyISAM テーブル（.MYI および .MYD）のチェックおよび修復には myisamchk ユーティリティを使用してください。ISAM テーブル（.ISM および .ISD）のチェックおよび修復には isamchk ユーティリティを使用します。 See 章 7. MySQL のテーブル型。

以下の説明は myisamchk についてですが、旧 isamchk にもすべて当てはまります。

myisamchk ユーティリティを使用して、データベーステーブルの情報を取得したり、テーブルのチェック、修復、および最適化を実行できます。以下のセクションでは、myisamchk の起動方法（オプションの説明も含む）、テーブル保守のスケジュール、およびさまざまな myisamchk 機能について説明します。

多くの場合、OPTIMIZE TABLES コマンドを使用してテーブルの最適化と修復を行うこともできますが、myisamchk に比べると時間がかかり、重大エラーの場合は信頼性も高くありません。その反面、OPTIMIZE TABLE は使用が簡単で、テーブルのフラッシュを心配する必要がありません。 See 項4.6.1.「OPTIMIZE TABLE 構文」。

myisamchk による修復は信頼性に優れていますが、修復を実行する（つまりテーブルに多くの変更を加える）前に、バックアップを作成しておくことが必要です。

4.5.6.1. myisamchk 起動構文

myisamchk は以下のように起動します。

```
shell> myisamchk [options] tbl_name
```

options は、myisamchk が実行することを指定します。それらについてはここで説明します（myisamchk --help を実行すると、オプションの一覧を取得できます）。オプションを指定しなければ、myisamchk はテーブルのチェックだけを行います。詳細情報を取得したり、myisamchk に修正を実行させるには、ここで説明するオプションを指定することが必要です。

tbl_name は、チェックおよび修復対象となるデータベーステーブル名です。データベースディレクトリ以外の場所で myisamchk を実行する場合、myisamchk に対してファイルのパスを指定する必要があります。実際、myisamchk は処理対象のファイルがデータベースディレクトリにあるかどうかを問題にしません。データベーステーブルのファイルを他の場所にコピーし、そこでリカバリ操作を実行することもできます。

myisamchk コマンドラインに複数のテーブル名を指定することもできます。また、インデックスファイル名（.MYI サフィックス付き）を指定することもできます。*.MYI パターンを使用すれば 1 つのディレクトリ内のすべてのテーブルを指定することができます。 たとえば、カレントディレクトリがデータベースディレクトリである場合、以下のようにすればディレクトリ内の全テーブルをチェックできます。

```
shell> myisamchk *.MYI
```

カレントディレクトリがデータベースディレクトリでない場合、以下のようにディレクトリのパスを指定することにより、その全テーブルをチェックできます。

```
shell> myisamchk /path/to/database_dir/*.MYI
```

MySQL データディレクトリのパスにワイルドカードを使用することにより、全データベースのすべてのテーブルをチェックすることもできます。

```
shell> myisamchk /path/to/datadir/*/*.MYI
```

すべてのテーブルに対して簡単にチェックを行う場合、以下の方法を推奨します。

```
myisamchk --silent --fast /path/to/datadir/*/*.MYI
isamchk --silent /path/to/datadir/*/*.ISM
```

すべてのテーブルをチェックし、破損していたテーブルをすべて修復するには、以下を実行します。

```
myisamchk --silent --force --fast --update-state -O key_buffer=64M \
        -O sort_buffer=64M -O read_buffer=1M -O write_buffer=1M \
        /path/to/datadir/*/*.MYI
isamchk --silent --force -O key_buffer=64M -O sort_buffer=64M \
       -O read_buffer=1M -O write_buffer=1M /path/to/datadir/*/*.ISM
```

この例では、64 メガバイト以上の空き容量があることを前提にしています。

注意: 以下のエラーが発生する場合があります。

```
myisamchk: warning: 1 clients is using or hasn't closed the table properly
```

これは、他のプログラム (mysqld サーバなど) によって更新されてまだ閉じられていないファイル、または正しく閉じられていないファイルのテーブルをチェックしようとしているということです。

mysqld を実行中であれば、FLUSH TABLES ですべてのテーブルの同期とクローズを強制的に実行し、myisamchk を実行する間は他のだれにもテーブルを使用させないようにしてください。MySQL バージョン 3.23 で、この問題を回避する最も簡単な方法は、myisamchk の代わりに CHECK TABLE を使用してテーブルをチェックすることです。

4.5.6.2. myisamchk の一般的なオプション

myisamchk は、以下のオプションをサポートします。

- -# または --debug=debug_options

  デバッグログを出力する。debug_options 文字列には、'd:t:o,filename' がよく使用される。

- -? または --help

  ヘルプメッセージを表示して終了する。

- -O name=value, --set-variable=name=value

  変数の値を指定する。 注意: --set-variable=name=value および -O name=value 構文は MySQL 4.0 で廃止。代わりに --name=value を使用すること。 myisamchk で使用できる変数とそのデフォルト値は、myisamchk --help で確認できる。

変数	値
key_buffer_size	523264
read_buffer_size	262136
write_buffer_size	262136
sort_buffer_size	2097144
sort_key_blocks	16
decode_bits	9

sort_buffer_size は、キーのソートによってキーが修復された場合に使用する。これは、--recover を使用すると通常発生する状態。

key_buffer_size は、--extended-check を使用してテーブルをチェックしている場合や、通常の挿入と同様、キーをテーブルのレコードごとに挿入することによりキーが修復される場合に使用する。以下の場合に、キーバッファによる修復が使用される。

- --safe-recover を使用する場合。
- キーファイルを直接作成した場合よりキーをソートした場合の方が、必要なテンポラリファイルが 2 倍以上の大きさになる場合。ソート中はキー全体を保存しておく必要があるため、CHAR キー、VARCHAR キー、または TEXT キーのサイズが大きい場合にこのケースが発生する。テンポラリ領域が豊富にあり、ソートによる修復を myisamchk に実行させることができる場合は、--sort-recover オプションを使用できる。

キーバッファによる修復は、ソートによる修復よりディスク領域は少なくて済むが、処理速度は遅くなる。

速い修復を望む場合は、上記の変数を利用可能メモリの 1/4 に設定する。上記バッファは 1 度に 1 つだけ使用されるため、両方の変数を大きな値に設定できる。

- -s または --silent

  サイレントモード。エラー発生時のみ出力する。-s を 2 回指定すると（-ss）、myisamchk はさらに出力が少なくなる。

- -v または --verbose

  冗長モード。詳細情報を出力する。-d および -e と併用できる。-v を複数回指定すると（-vv、-vvv など）、より冗長になる。

- -V または --version

  myisamchk のバージョン情報を出力して終了する。

- -w または --wait

  テーブルがロックされている場合にエラーを出さず、テーブルのロックが解除されるまで待機する。注意: そのテーブルに対して --skip-external-locking オプション付きの mysqld が実行している場合、そのテーブルは別の myisamchk コマンドによってのみロックされる。

4.5.6.3. myisamchk のチェックオプション

- -c または --check

  テーブルのエラーをチェックする。myisamchk に対して明示的にこのオプションを上書きするオプションを指定しなければ、これがデフォルト処理になる。

- -e または --extend-check

  テーブルを完全にチェックする（インデックスが多い場合、処理時間がかかる）。このオプションは、極端な場合にだけ使用する。通常、myisamchk または myisamchk --medium-check でテーブルのエラーを検出できる。

  --extended-check を使用し、メモリが豊富にある場合、key_buffer_size の値を大きく設定すべきである。

- -F または --fast

  正しく閉じられなかったテーブルだけをチェックする。

- -C または --check-only-changed

  前回のチェック以降に変更されたテーブルだけをチェックする。

- -f または --force

  myisamchk でテーブルにエラーが検出された場合、myisamchk を -r（修復）で再起動する。

- -i または --information

  チェックしたテーブルの統計を出力する。

- -m または --medium-check

  extended-check よりも速いが、すべてのエラーは見つけられない。しかし、ほとんどの場合はこれで十分である。

- -U または --update-state

  テーブルをいつチェックしたか、その際テーブルが破損していたかどうかを .MYI ファイルに記録する。このオプションは、--check-only-changed オプションを最大限有効に活用するために使用する。ただし、mysqld サーバがそのテーブルを使用しており、mysqld を --skip-external-locking で実行している場合は、このオプションを使用してはいけない。

- -T または --read-only

  テーブルにチェック済みのマーキングをしない。mysqld --skip-external-locking などのような、ロックを使用しない他のアプリケーションによって使用されているテーブルを、myisamchk を使用してチェックする場合、このオプションが便利である。

4.5.6.4. myisamchk の修復オプション

以下のオプションは、myisamchk を -r または -o で起動した場合に使用できます。

- -B または --backup

  .MYD ファイルのバックアップを filename-time.BAK として作成する。

- --correct-checksum

  テーブルのチェックサム情報を修正する。

- -D # または --data-file-length=#

  データファイルの最大長（データファイルが 'full' で再生成されるとき）。

- -e または --extend-check

  データファイルから可能なすべてのレコードのリカバリを試みる。 通常、このオプションではガーベジレコードも多く見つかる。他に手がないとき以外は、このオプションは使用しないようにする。

- -f または --force

  停止しないで、古いテンポラリファイル（table_name.TMD）を上書きする。

- -k # または --keys-used=#

  ISAM を使用している場合、ISAM ストレージエンジンが最初の # インデックスだけを更新するように指定する。 MyISAM を使用している場合は、各バイナリビットが 1 つのキーに対応しているので、どのキーを使用するかはビット計算をして指定する。 このオプションを使用して、挿入を高速化できる。無効化されたインデックスを再び有効にするには、myisamchk -r を使用する。

- -l または --no-symlinks

  シンボリックリンクを使用しない。通常、myisamchk はシンボリックリンクが指すテーブルを修復する。このオプションは MySQL 4.0 にはない。MySQL 4.0 では修復中にシンボリックリンクが削除されないためである。

- -p または --parallel-recover

  -r および -n と同じテクニックを使用するが、すべてのキーを複数のスレッドで並列処理する。 このオプションは MySQL 4.0.2 で導入。 これは、アルファコードである。使用にはリスクが伴う。

- -r または --recover

  ほとんどのエラーを修復できる。ただし、ユニークキーが一意でないエラーの場合には対応できない（ISAM テーブルまたは MyISAM テーブルでは非常に珍しいエラーである）。 テーブルをリカバリする場合は、まずこのオプションを試す。-r ではテーブルをリカバリできないと myisamchk により報告された場合のみ、-o を実行する。注意: めったにないことではあるが -r が失敗した場合でも、データファイルは損傷を受けない。 メモリが豊富にある場合は、sort_buffer_size のサイズを大きくすべきである。

- -o または --safe-recover

  古いリカバリ方式を使用する（すべてのレコードを順番どおりに読み取り、検出されたレコードに基づいてすべてのインデックスツリーを更新する）。これは -r よりもかなり遅いが、-r で処理できない場合にも対応する。このリカバリ方式は、-r よりもディスク容量の使用が少なくて済む。通常は最初に -r で修復を試みて、それに失敗した場合のみ -o を使用する。

メモリが豊富にある場合は、key_buffer_size のサイズを大きくすべきである。

- -n または --sort-recover

  テンポラリファイルが非常に大きい場合でも、キー解決のために myisamchk にソートを強制する。

- --character-sets-dir=...

  キャラクタセットが格納されているディレクトリ。

- --set-character-set=name

  インデックスによって使用されるキャラクタセットを変更する。

- -t または --tmpdir=path

  テンポラリファイルを保存するパス。これを設定しない場合、myisamchk はこれに環境変数 TMPDIR を使用する。MySQL 4.1 以降、tmpdir で、複数のパスをコロン : で区切って設定できるようになっている (Windows ではセミコロン ;) 。これは、ラウンドロビン方式で使用される。

- -q または --quick

  データファイルを修正しないことにより、修復を速く行う。-q を 2 回指定すると、重複キーの場合に、myisamchk にオリジナルのデータファイルの修正を強制する。

- -u または --unpack

  myisampack でパックされたファイルをアンパックする。

4.5.6.5. myisamchk のその他のオプション

以下は、myisamchk で実行できる、テーブルの修復およびチェック以外の操作です。

- -a または --analyze

  キーの分布を分析する。このオプションを使用することにより、結合オプティマイザが、テーブルの結合順序や使用すべきキーをより的確に判断できるようになるため、結合パフォーマンスが向上する。 myisamchk --describe --verbose table_name' または MySQL で SHOW KEYS を使用することにより、キーの分布を確認できる。

- -d または --description

  テーブルに関する情報を出力する。

- -A または --set-auto-increment[=value]

  指定の値以上で AUTO_INCREMENT を開始する。ここで値を指定しなければ、次の AUTO_INCREMENT 値は、オートキーに使用された最高値に 1 を加えた値になる。

- -S または --sort-index

  インデックスツリーブロックを降順にソートする。 検索が最適化され、キーによるテーブルスキャンが速くなる。

- -R または --sort-records=#

  インデックスに従ってレコードをソートする。このソートにより、データが整理され、このインデックスでの SELECT や ORDER BY の処理速度が速くなる（最初のソートは非常に遅くなる）。 テーブルのインデックス番号を調べるには SHOW INDEX を使用する。テーブルのインデックスが myisamchk が検出する順序で表示される。インデックス番号は 1 から始まる。

4.5.6.6. myisamchk のメモリ使用

myisamchk を実行する場合、メモリの配分が重要な問題になります。 myisamchk は、-O オプションで指定したサイズ以上のメモリは使用しません。非常に大きなファイルに対して myisamchk を使用する場合、メモリの使用量を最初に決めておいてください。修復処理に使用するメモリのデフォルト値は約 3 メガバイトです。大きな値を使用すれば、myisamchk の処理を高速化できます。たとえば、32 メガバイト以上の RAM がある場合、以下のようにオプションを指定できます（必要に応じて他のオプションも追加します）。

```
shell> myisamchk -O sort=16M -O key=16M -O read=1M -O write=1M ...
```

ほとんどの場合、-O sort=16M で十分です。

myisamchk は、TMPDIR のテンポラリファイルを使用することに注意してください。TMPDIR がメモリファイルシステムをポイントしていれば、すぐにメモリ不足エラーになります。このエラーが発生する場合には、TMPDIR を、容量の大きいディレクトリを指すように設定し、myisamchk を再起動します。

修復時も、myisamchk は大きなディスク容量を必要とします。

- レコードファイルサイズの 2 倍（オリジナルとコピーの分）。--quick で修復を実行する場合、インデックスファイルだけが再生成されるため、この容量は不要である。オリジナルのレコードファイルと同じディスク上にこの容量が必要である。
- 古いインデックスファイルと置き換えられる新しいインデックスファイルの容量。古いインデックスファイルは最初に切り捨てられるため、通常、この容量は無視される。 オリジナルのインデックスファイルと同じディスク上にこの容量が必要である。
- --recover または --sort-recover を使用する場合（--safe-recover を使用する場合は含まれない）、ソートバッファ用の容量として (largest_key + row_pointer_length)*number_of_rows * 2 が必要である。 キーの長さおよびレコードポインタの長さを確認するには、myisamchk -dv table を使用する。 この容量は、テンポラリディスクに割り当てられる（TMPDIR または --tmpdir=# で指定）。

修復中にディスク容量に関する問題が発生した場合は、--recover の代わりに --safe-recover を使用できます。

4.5.6.7. myisamchk を使用したクラッシュのリカバリ

mysqld を --skip-external-locking で実行している場合（Linux などのシステムではデフォルト）、mysqld が使用している同じテーブルを myisamchk でチェックしても信頼できません。myisamchk の実行中、mysqld からテーブルにアクセスするユーザが確実にない場合のみ、mysqladmin flush-tables を実行してからテーブルをチェックします。テーブルにアクセスするユーザがいるかどうか確信できない場合には、テーブルをチェックする間、mysqld を停止する必要があります。mysqld がテーブルを更新しているときに myisamchk を実行すると、テーブルが破損していない場合でも破損の警告が出

る可能性があります。

--skip-external-locking を使用していなければ、いつでも myisamchk を使用してテーブルをチェックできます。この間、テーブルを更新しようとするクライアントはすべて、myisamchk の準備が整うのを待ってから実行します。

myisamchk を使用してテーブルを修復または最適化する場合、必ず、mysqld サーバがそのテーブルを使用していないことを確認してください (--skip-external-locking を使用している場合も同様です)。 mysqld を停止しない場合は、少なくとも mysqladmin flush-tables を myisamchk の前に実行してください。 サーバと myisamchk が同時にテーブルにアクセスすると、テーブルが破損するおそれがあります。

この章では、MySQL データベースのデータ破損のチェック方法およびその対処方法について説明します。テーブルが頻繁に破損する場合は、原因を突き止める必要があります。 See 項A.4.1.「MySQL が何度もクラッシュする場合に行うこと」。

MyISAM テーブルのセクションで、テーブルが破損する原因を示しています。 See 項7.1.3.「MyISAM テーブルの問題」。

クラッシュをリカバリする際には、データベース内のテーブル tbl_name のそれぞれがデータベースディレクトリ内の次の 3 つのファイルに対応していることを理解しておく必要があります。

ファイル	用途
tbl_name.frm	テーブル定義ファイル
tbl_name.MYD	データファイル
tbl_name.MYI	インデックスファイル

これら 3 つのファイルタイプはいずれもさまざまな形で破損する可能性がありますが、特にデータファイルとインデックスファイルに問題がよく発生します。

myisamchk は、.MYD (データ) ファイルのコピーをレコードごとに生成します。修復の最終段階で古い .MYD ファイルを削除し、新規ファイルをオリジナルの名前に変更します。--quick を使用している場合、myisamchk はテンポラリ .MYD ファイルを生成しませんが、代わりに .MYD ファイルが正常であるとみなし、.MYD ファイルに手を加えずに新規インデックスファイルだけを生成します。.MYD ファイルに問題があった場合は myisamchk が自動的に検知して修復を中止するため、この方法も安全です。2 つの --quick オプションを myisamchk に設定することもできます。この場合、myisamchk はいくつかのエラー (重複キーなど) でも中止せず、.MYD ファイルを修正して解決しようとします。通常の修復処理にディスクの空き容量では足りない場合にのみ、2 つの --quick オプション指定が役立ちます。この場合、少なくとも myisamchk を実行する前にバックアップを作成しておいてください。

4.5.6.8. テーブルのエラーチェック方法

MyISAM テーブルをチェックするには、以下のコマンドを使用します。

- myisamchk tbl_name

  これで、エラーのほとんどは発見できる。これで発見できないエラーは、データファイルだけに関連する破損である (これは非常にまれ)。テーブルをチェックする場合、通常、myisamchk をオプションなしか、-s または --silent のオプションで実行する。

- myisamchk -m tbl_name

これで、エラーのほとんどは発見できる。このコマンドを実行すると、最初にすべてのインデックスエントリのエラーがチェックされ、次にすべてのレコードが読み込まれる。レコードのすべてのキーのチェックサムが計算され、その結果がインデックスツリーのキーのチェックサムと一致するかどうか確認される。

- myisamchk -e tbl_name

  このコマンドを実行すると、すべてのデータの完全なチェックが行われる（ -e は ``extended check'' のこと）。各レコードのすべてのキーが読み取られ、チェックされ、正しいレコードが指されているか確認される。この操作は、多くのキーがある大きなテーブルでは時間がかかる。myisamchk は通常、最初のエラーが見つかった時点で停止する。詳細情報を得るには、--verbose（ -v ）オプションを追加する。これにより、myisamchk は最大 20 のエラーが検出されるまで処理を続行する。 通常は、テーブル名以外の引数を指定しない myisamchk だけで十分である。

- myisamchk -e -i tbl_name

  前のコマンドと似ているが、-i オプションによって myisamchk が統計情報も出力する。

4.5.6.9. テーブルの修復方法

このセクションでは、MyISAM テーブル（ .MYI および .MYD の拡張子）に対して myisamchk を使用する方法について説明します。ISAM テーブル（ .ISM および .ISD の拡張子）に対しては、isamchk を使用してください。

MySQL バージョン 3.23.14 以降、REPAIR TABLE コマンドで MyISAM テーブルを修復できるようになっています。 See 項4.5.5.「REPAIR TABLE 構文」。

テーブル破損の症状としては、クエリが予期せず中断したり、以下のようなエラーが発生します。

- tbl_name.frm is locked against change
- Can't find file tbl_name.MYI (Errcode: ###)
- Unexpected end of file
- Record file is crashed
- Got error ### from table handler

  perror ### を実行すると、エラーの詳細情報を取得できる。以下は、テーブルに問題があることを示す一般的なエラーである。

```
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
```

  注意: エラー 135（ no more room in record file ）は、単純な修復で直せるエラーではない。この場合、以下のコマンド

を実行することが必要。

```
mysql> ALTER TABLE table MAX_ROWS=xxx AVG_ROW_LENGTH=yyy;
```

この手法は、エラー 136 (no more room in index file) に対しても使用できる。

他の場合についてはテーブルを修復する必要があります。通常は、myisamchk でほとんどのエラーを検出して修復できます。

修復プロセスは、以下説明するように 4 段階で構成されます。開始する前に、データベースディレクトリに移動 (cd) してテーブルファイルのアクセス権を確認してください。mysqld を実行する Unix ユーザに読み取り権限があることを確認します (この確認を行うユーザにもこれらのファイルへのアクセス権が必要)。ファイルを修正する必要がある場合は、書き込み権限も必要です。

MySQL バージョン 3.23.16 以降を使用している場合は、CHECK コマンドと REPAIR コマンドを使用して MyISAM テーブルのチェックおよび修復を実行できます (できるだけ、これらのコマンドを使用してください)。 See 項4.5.4.「CHECK TABLE 構文」。 See 項4.5.5.「REPAIR TABLE 構文」。

テーブル保守に関するセクションで、isamchk および myisamchk のオプションを説明しています。 See 項4.5.6.「myisamchk を使用したテーブルの保守とクラッシュのリカバリ」。

これらのコマンドで失敗した場合、または isamchk および myisamchk の拡張機能を使用する場合につき、以下の説明を参考にして修復作業を行ってください。

コマンドラインを使用してテーブルを修復する場合、まず、mysqld サーバをシャットダウンする必要があります。注意: mysqladmin shutdown をリモートサーバから実行した場合、mysqladmin が戻ってもしばらくの間、mysqld サーバはシャットダウンしません。すべてのクエリが停止してすべてのキーがディスクにフラッシュされた時点でシャットダウンします。

段階 1: テーブルのチェック

myisamchk *.MYI または時間に余裕があれば myisamchk -e *.MYI を実行します。-s (サイレント) オプションを使用すると、不要な情報は出力されません。

mysqld サーバが終了している場合、--update オプションを使用して myisamchk がテーブルに 'checked' のマークを付けるようにします。

myisamchk がエラーを返したテーブルだけ、修復する必要があります。そのようなテーブルについては、段階 2 へ進みます。

チェック時に複雑なエラー (out of memory エラーなど) が発生した場合、あるいは myisamchk がクラッシュした場合は段階 3 に進みます。

段階 2: 簡単で安全な修復

注意: 修復を速く行うには、-O sort_buffer=# -O key_buffer=# (# は利用可能メモリの約 4 分の 1) をすべての isamchk/myisamchk コマンドに追加します。

最初に myisamchk -r -q tbl_name (-r -q は ``クイックリカバリモード" の意) を実行します。これにより、データファイルを変更せずにインデックスファイルの修復だけが行われます。データファイルにあるべきものがすべてあり、削除リンク

がデータファイル内の正しい場所を指していれば、テーブルは正常に修復されます。成功したら、次のテーブルの修復を開始します。失敗した場合は以下の手順を実行します。

1. まず、データファイルのバックアップを作成する。
2. myisamchk -r tbl_name (-r は ``リカバリモード" の意) を使用する。これにより、不正なレコードと削除されたレコードがデータファイルから削除され、インデックスファイルが再構築される。
3. 前のステップが失敗した場合、myisamchk --safe-recover tbl_name を使用する。 セーフリカバリモードにより、古いリカバリ形式が使用され、通常のリカバリモードでは処理できない修復が行われる。ただし、時間がかかる。

修復時に複雑なエラー (out of memory エラーなど) が発生した場合、あるいは myisamchk がクラッシュした場合は段階 3 に進みます。

段階 3:困難な修復

インデックスファイルの最初の 16K ブロックが破損または不正な情報を含む場合、あるいはインデックスファイルがない場合だけ、この段階に進みます。この段階では、新しいインデックスファイルを作成する必要があります。以下を実行してください。

1. データファイルを安全な場所に移動する。
2. テーブル記述ファイルを使用して、新しい (空白の) データとインデックスファイルを作成する。

   ```
   shell> mysql db_name
   mysql> SET AUTOCOMMIT=1;
   mysql> TRUNCATE TABLE table_name;
   mysql> quit
   ```

   使用している SQL バージョンに TRUNCATE TABLE がない場合は、代わりに DELETE FROM table_name を使用する。
3. 古いデータファイルを、新しく作成したデータファイルにコピーする (単に古いファイルを新しいファイルに移動するのではなく、万一に備えて元の場所にも残しておく)。

段階 2 に戻ります。これで、myisamchk -r -q が機能するはずです (無限ループにならないはずです)。

MySQL 4.0.2 より、この手順全体を自動で実行する REPAIR ... USE_FRM も使用できるようになりました。

段階 4: 非常に困難な修復

記述ファイルもクラッシュしている場合にのみこの段階に進みます。ただし、テーブル作成後に記述ファイルが変更されることはないので、通常は発生しない状況です。

1. バックアップから記述ファイルをリストアし、段階 3 に戻る。インデックスファイルをリストアして段階 2 に戻ることもできる。後者の場合、myisamchk -r を実行する。
2. バックアップがない場合でも、テーブルがどのように作成されたか正確にわかっていれば、別のデータベースにテーブルのコピーを作成する。テーブルのコピーを作成したデータベースから、新しいデータファイルを削除し、記述フ

ァイルとインデックスファイルを、クラッシュしたデータベースに移動する。これで新しい記述ファイルとインデックスファイルができ、データファイルは前のものがそのまま残る。段階 2 に戻り、インデックスファイルを再構築する。

4.5.6.10. テーブルの最適化

断片化したレコードを結合したり、レコードの削除または更新によって発生した無駄なスペースを除去するには、myisamchk をリカバリモードで実行します。

```
shell> myisamchk -r tbl_name
```

同様に、SQL の OPTIMIZE TABLE ステートメントを使用して、テーブルを最適化することもできます。OPTIMIZE TABLE はテーブルの修復とキー分析を行い、さらにインデックスツリーをソートして、キー走査の処理速度を上げます。また、OPTIMIZE TABLE を使用した場合、サーバ側ですべての処理が行われるため、ユーティリティとサーバ間で不要なやり取りが発生しません。 See 項4.6.1.「OPTIMIZE TABLE 構文」。

myisamchk には、テーブルのパフォーマンスを向上させるためのオプションが数多く用意されています。

- -S, --sort-index
- -R index_num, --sort-records=index_num
- -a, --analyze

オプションの詳細な説明については、See 項4.5.6.1.「myisamchk 起動構文」 を参照してください。

4.5.7. テーブル保守計画

MySQL バージョン 3.23.13 以降、CHECK TABLE コマンドを使用して MyISAM テーブルをチェックできるようになりました。See 項4.5.4.「CHECK TABLE 構文」。 テーブルの修復は REPAIR TABLE コマンドで行えます。 See 項4.5.5.「REPAIR TABLE 構文」。

問題が発生するのを待つより、定期的にテーブルチェックを行うことを推奨します。保守目的では、myisamchk -s を使用してテーブルをチェックできます。-s オプション (--silent の短縮形) を使用すると、サイレントモードで myisamchk が実行され、エラー発生時のみメッセージが出力されます。

また、サーバが起動するたびにテーブルをチェックするという方法もあります。 たとえば、更新の途中でマシンがリブートされた場合、影響を受けた可能性のあるテーブルをすべてチェックする必要があります (つまり、``破損の可能性があるテーブル" をチェックします) 。リブート後に古い .pid (プロセス ID) ファイルが残っていた場合、mysqld_safe にテストを追加して、直前の 24 時間以内に変更されたテーブルのチェックを myisamchk で実行することができます。.pid ファイルは mysqld 起動時に作成され、正常終了時に削除されます。システム起動時に .pid ファイルが存在していれば、mysqld が異常終了したことになります。

より完全なテストを行うには、.pid ファイルの作成後に変更されたすべてのテーブルをチェックするという方法もあります。

通常のシステム運用中にも定期的にテーブルをチェックしてください。MySQL AB では、週に一度 cron ジョブを実行して重要なテーブルをチェックしています。crontab ファイルは以下のようになります。

```
35 0 * * 0 /path/to/myisamchk --fast --silent /path/to/datadir/*/*.MYI
```

これで、クラッシュしたテーブルに関する情報が出力されるので、必要に応じて検査および修復を行えます。

ここ数年、予想外のテーブル破損（ハードウェアトラブル以外による破損）は発生していないので、当社では週に一度のチェックで十分としています。

最初のうちは、myisamchk -s を毎晩実行して、直前の 24 時間以内に更新されたすべてのテーブルをチェックするようにします。しばらくすると、MySQL の信頼性の高さを実感できるはずです。

通常、MySQL テーブルの保守はそれほど頻繁に行う必要はありません。動的サイズのレコード（VARCHAR カラム、BLOB カラム、または TEXT カラム）があるテーブルを変更したり、多くのレコードが削除されたテーブルがある場合には、月に一度程度、最適化を行ってテーブルの領域を解放することを推奨します。

これを行うには、目的のテーブルで OPTIMIZE TABLE を実行するか、mysqld サーバを停止して以下を実行します。

```
isamchk -r --silent --sort-index -O sort_buffer_size=16M */*.ISM
myisamchk -r --silent --sort-index  -O sort_buffer_size=16M */*.MYI
```

4.5.8. テーブル情報の取得

テーブルに関する情報またはその統計を取得するには、以下のコマンドを実行します。これらの情報については後で詳しく説明します。

- myisamchk -d tbl_name myisamchk を ``describe モード" で実行し、テーブル情報を生成する。--skip-external-locking オプションを使用して MySQL サーバを起動した場合、myisamchk は、実行中に更新されたテーブルに対してエラーを報告する場合がある。しかし、myisamchk は describe モードではテーブルを変更しないため、データ破壊の危険性はない。
- myisamchk -d -v tbl_name myisamchk の実行中の処理に関する詳細な情報を取得するには、-v を追加して冗長モードで実行する。
- myisamchk -eis tbl_name テーブルの最重要情報のみ表示する。テーブル全体を読み取ることになるので、時間がかかる。
- myisamchk -eiv tbl_name -eis とほぼ同じであるが、このオプションを使用すると、進行中の処理も表示される。

myisamchk -d による出力例

```
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
```

myisamchk -d -v による出力例

```
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
```

myisamchk -eis による出力例

```
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
```

myisamchk -eiv による出力例

```
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
```

上記の例で使用されたテーブルのデータファイルおよびインデックスファイルのサイズは以下のとおりです。

```
-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.MYI
```

以下、myisamchk が生成する情報のタイプについて説明します。``キーファイル" とはインデックスファイルのことです。``レコード" と ``ロー" はシノニムです。

- ISAM file ISAM (インデックス) ファイルの名前。
- Isam-version ISAM 形式のバージョン。現在は常に 2。
- Creation time データファイルの作成日時。
- Recover time インデックスまたはデータファイルが前回再構築された日時。
- Data records テーブル内のレコード数。
- Deleted blocks 予約済み領域をまだ占有している削除済みのブロック数。 テーブルを最適化して、この領域を最小にできる。 See 項4.5.6.10.「テーブルの最適化」。
- Data file:Parts 動的なレコード形式に対しては、存在するデータブロック数。断片化されたレコードのない最適化されたテーブルに対しては、Data records と同じ。
- Deleted data 領域が解放されていない削除済みデータのバイト数。 テーブルを最適化して、この領域を最小にできる。 See 項4.5.6.10.「テーブルの最適化」。
- Data file pointer データファイルポインタのサイズ (バイト単位)。通常は 2、3、4、または 5 バイト。ほとんどのテーブルは 2 バイトで足りるが、現在のところ、MySQL で制御することはできない。固定テーブルでは、これはレコードアドレス。動的テーブルでは、バイトアドレス。
- Keyfile pointer インデックスファイルポインタのサイズ (バイト単位)。通常は 1、2、または 3 バイト。ほとんどのテーブルは 2 バイトで足りるが、MySQL によって自動的に計算される。常にブロックアドレスである。
- Max datafile length テーブルのデータファイル (.MYD ファイル) の最大長 (バイト単位)。
- Max keyfile length テーブルのキーファイル (.MYI ファイル) の最大長 (バイト単位)。
- Recordlength 各レコードが使用する領域サイズ (バイト単位)。
- Record format テーブルレコードの格納形式。 上記の例では Fixed length を使用。 他に、Compressed および Packed がある。
- table description テーブル内のすべてのキーの一覧。各キーについて、以下の低レベル情報が表示される。
  - Key キー番号。
  - Start インデックス部が始まるレコード内の位置。
  - Len インデックス部の長さ。パックされた数値の場合、これは常にそのカラムの全長となる。文字列の場合、文字列カラムのプリフィックスをインデックスにできるため、インデックス化されたカラムの全長よりも短くなる場合がある。

  - Index unique または multip.（複数）。このインデックスに値の重複が認められているかどうかを示す。
  - Type インデックス部のデータ型。packed、stripped、または empty のいずれかの ISAM データ型。
  - Root ルートインデックスブロックのアドレス。
  - Blocksize 各インデックスブロックのサイズ。デフォルトでは 1024 であるが、コンパイル時に変更可能。
  - Rec/key オプティマイザによって使用される統計値。このキーの値ごとのレコード数を示す。ユニークキーの値は常に 1。これは、myisamchk -a でテーブルがロード（または大きく変更）されると更新される。まったく更新されない場合は、デフォルト値の 30 となる。
- 上の最初の例の 9 番目のキーは 2 つのパートを持つマルチパートキー。
- Keyblocks used 使用されたキーブロックのパーセント。例で使用されたテーブルは myisamchk で再構成されたばかりなので、値は非常に高い（理論的な最大値に非常に近い）。
- Packed MySQL は共通のサフィックスでキーのパックを試みる。これは、CHAR、VARCHAR、DECIMAL の各キーにのみ使用できる。名前のような長い文字列では、パックにより使用領域を大きく減らすことができる。上の 3 番目の例では、4 番目のキーが 10 文字長で、領域が 60 パーセント減少している。
- Max levels このキーの B-tree の深さ。長いキーのある大きなテーブルでは、値が高くなる。
- Records テーブル内のレコード数。
- M.recordlength レコードの平均の長さ。固定長レコードのテーブルでは、これは実際のレコード長となる。
- Packed MySQL は、文字列の最後からスペースを削除する。Packed 値は、これによって節約されたパーセントを示す。
- Recordspace used 使用されているデータファイルのパーセント。
- Empty space 使用されていないデータファイルのパーセント。
- Blocks/Record レコードごとの平均ブロック数（断片化レコードを構成するリンク数）。これは、固定形式テーブルでは常に 1.0。この値は可能な限り、1.0 に近くしておく。大きくなりすぎた場合は、myisamchk で再編成できる。 See 項4.5.6.10.「テーブルの最適化」。
- Recordblocks 使用されているブロック（リンク）数。固定形式では、レコード数と同じになる。
- Deleteblocks 削除されたブロック（リンク）数。
- Recorddata 使用されているデータファイルのバイト数。
- Deleted data 削除された（使用されていない）データファイルのバイト数。
- Lost space レコードの長さを短くして更新した場合、領域がいくらか失われる。その失われた領域の合計バイト数。
- Linkdata 動的テーブル形式では、レコードの断片がポインタでリンクされている（それぞれ 4 〜 7 バイト）。Linkdata は、そのようなポインタすべてに使用されているストレージ量の合計を示す。

テーブルが myisampack で圧縮されている場合、myisamchk -d を実行すると、各テーブルカラムに関する追加情報が出力

されます。これらの情報およびその意味については、項4.8.4.「myisampack（MySQL 圧縮読み取り専用テーブルジェネレータ）」を参照してください。

4.6. データベース管理言語リファレンス

4.6.1. OPTIMIZE TABLE 構文

```
OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name[,tbl_name]...
```

OPTIMIZE TABLE は、テーブルの大部分を削除したり、可変長レコードを持つテーブル（VARCHAR カラム、BLOB カラム、または TEXT カラムを持つテーブル）に多くの変更を加えた場合に使用します。 削除されたレコードはリンクされたリストに保持され、元のレコード位置は、後続の INSERT 操作により再利用されます。OPTIMIZE TABLE を使用して未使用領域を解放し、データファイルを最適化することができます。

ほとんどの場合、OPTIMIZE TABLE を実行する必要はありません。可変長レコードに更新を多く行う場合でも、週または月に一度、特定のテーブルだけに実行するだけで十分です。

現在のところ、OPTIMIZE TABLE は MyISAM テーブル、 BDB およびInnoDBテーブルにのみ有効です。BDB テーブルでは、OPTIMIZE TABLE は現在 ANALYZE TABLE にマップされます。 See 項4.6.2.「ANALYZE TABLE 構文」。

mysqld を --skip-new または --safe-mode で起動することにより、OPTIMIZE TABLE を他のテーブルタイプにでも使用できるようになります。しかしこの場合、OPTIMIZE TABLE は ALTER TABLE にマップされます。

OPTIMIZE TABLE は以下のように動作します。

- テーブルに削除されたレコードまたは分割されたレコードがある場合、テーブルを修復する。
- インデックスページがソートされていない場合、ソートする。
- 統計が最新でなければ（およびインデックスのソートによる修復ができなかった場合）統計を更新する。

注意: OPTIMIZE TABLE の実行中、テーブルはロックされます。

MySQL 4.1.1 より前のバージョンを使用している場合、OPTIMIZE コマンドはバイナリログに記録されません。MySQL 4.1.1 以降を使用している場合は、オプションの NO_WRITE_TO_BINLOG キーワード（またはそのエイリアスの LOCAL）を使用していない限り、バイナリログに記録されます。

4.6.2. ANALYZE TABLE 構文

```
ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name[,tbl_name...]
```

テーブルのキー分布を分析して保存します。分析中、テーブルは読み取りロックされます。このコマンドは、MyISAM テーブルと BDB テーブルに対して使用できます。

これは、テーブルに対して myisamchk -a を実行するのと同じです。

定数を条件としない結合を実行する場合、MySQL は保存されたキー分布を使用して、テーブルの結合順序を決定します。

このコマンドは、以下のカラムで構成されるテーブルを返します。

カラム	値
Table	テーブル名
Op	常に analyze
Msg_type	status、error、info、warning のいずれか
Msg_text	メッセージ

保存されたキー分布をチェックするには、SHOW INDEX コマンドを使用します。 See 項4.6.8.1.「データベース、テーブル、カラム、およびインデックスに関する情報の取得」。

前回の ANALYZE TABLE コマンドを実行した後でテーブルが変更されていない場合、テーブルに対して再度分析は実行されません。

MySQL 4.1.1 より前のバージョンを使用している場合、ANALYZE コマンドはバイナリログに記録されません。MySQL 4.1.1 以降を使用している場合は、オプションの NO_WRITE_TO_BINLOG キーワード (またはそのエイリアスの LOCAL) を使用していない限り、バイナリログに記録されます。

4.6.3. CHECKSUM TABLE 構文

```
CHECKSUM TABLE tbl_name[,tbl_name ...] [ QUICK | EXTENDED ]
```

テーブルのチェックサムを報告します。QUICK を指定した場合、現時点のテーブルのチェックサムが報告されます。または、テーブルがチェックサムをサポートしていない場合は NULL が返されます。この処理時間は非常に短くて済みます。EXTENDED モードでは、テーブル全体がレコードごとに読み取られ、チェックサムが計算されます。大きなテーブルの場合は時間がかかります。デフォルトでは (QUICK でも EXTENDED でもない場合) 、テーブルがチェックサムをサポートしていれば、現時点のチェックサムが返され、サポートしていなければ、テーブルがスキャンされます。

このコマンドは、MySQL 4.1.1 で導入されました。

4.6.4. FLUSH 構文

```
FLUSH [LOCAL | NO_WRITE_TO_BINLOG] flush_option [,flush_option] ...
```

MySQL が使用している内部キャッシュを消去するには、FLUSH コマンドを使用します。FLUSH を実行するには、RELOAD 権限が必要です。

flush_option には、以下のいずれかを指定できます。

オプション	説明
HOSTS	ホストキャッシュテーブルを空にする。ホストが IP アドレスを変更した場合や、エラーメッセージ Host ... is blocked が表示される場合には、ホストテーブルをフラッシュすることが必要である。MySQL サーバとの接続中に、特定のホストの 1 つのレコードで max_connect_errors を超えるエラーが発生した場合、MySQL は問題があると判断し、以後、ホストからの接続要求をブロックする。 ホストテーブルをフラッシュすると、ホストが再び接続可能になる。See 項A.2.5.「Host '...' is blocked エラー」。 mysqld を -O max_connect_errors=999999999 で起動すると、このエラーを回避できる。

DES_KEY_FILE	サーバ起動時に --des-key-file オプションで指定されたファイルから、DES キーを再度読み込む。
LOGS	すべてのログファイルを閉じて、再び開く。 更新ログファイルまたはバイナリログファイルを拡張子なしで指定している場合、ログファイルの拡張子番号は前のファイルに 1 を加算した数になる。ファイル名に拡張子を使用している場合、MySQL は更新ログファイルを閉じて、再び開く。 See 項4.10.3.「更新ログ」。 これは、SIGHUP シグナルを mysqld サーバに送信するのと同じである。
PRIVILEGES	mysql データベースの権限テーブルから権限を再読み込みする。
QUERY CACHE	クエリキャッシュを最適化し、メモリの使用効率を高める。このコマンドは RESET QUERY CACHE とは異なり、キャッシュからクエリを削除しない。
TABLES	開いているテーブルをすべて閉じる。使用中のテーブルも強制的に閉じる。これにより、クエリキャッシュもフラッシュされる。
[TABLE | TABLES] tbl_name [,tbl_name...]	指定したテーブルだけをフラッシュする。
TABLES WITH READ LOCK	開いているテーブルをすべて閉じ、すべてのデータベースの全テーブルを読み取りロックする。このロックは UNLOCK TABLES を実行するまで解除されない。適宜スナップショットを取ることができる Veritas などのファイルシステムを使用している場合には、このコマンドがバックアップの作成に非常に便利である。
STATUS	ほとんどのステータス変数を 0 にリセットする。これは、クエリをデバッグするときにのみ使用する。
USER_RESOURCES	すべてのユーザリソースを 0 にリセットする。これにより、ブロックされていたユーザも再度ログイン可能になる。 See 項4.4.7.「ユーザリソースの制限」。

MySQL 4.1.1 より前のバージョンを使用している場合、FLUSH コマンドはバイナリログに記録されません。MySQL 4.1.1 以降のバージョンを使用している場合には、オプションの NO_WRITE_TO_BINLOG キーワード (またはそのエイリアスの LOCAL) を使用していない限り、バイナリログに記録されます。また、このコマンドに LOGS、MASTER、SLAVE、TABLES WITH READ LOCK のいずれかの引数が含まれる場合も、バイナリログに記録されません。これらの引数はスレーブにレプリケートされると問題が発生する可能性があります。

上記コマンドのうちのいくつかには、mysqladmin ユーティリティで flush-hosts、flush-logs、flush-privileges、 flush-status、flush-tables の各コマンドを使用してアクセスすることもできます。

レプリケーションで使用する RESET コマンドも参照してください。 See 項4.6.5.「RESET 構文」。

4.6.5. RESET 構文

```
RESET reset_option [,reset_option] ...
```

RESET コマンドは設定をリセットします。FLUSH コマンドのより強力なバージョンとしても機能します。 See 項4.6.4.「FLUSH 構文」。

RESET を実行するには、RELOAD 権限が必要です。

オプション	説明

MASTER	インデックスファイル内のすべてのバイナリログを削除し、バイナリログインデックスファイルを空白にリセットする。これは、以前の FLUSH MASTER。 See 項4.11.7.「マスタサーバを制御する SQL ステートメント」。
SLAVE	スレーブから、マスタバイナリログ内の自分のレプリケーション位置を消去する。これは、以前の FLUSH SLAVE。 See 項4.11.8.「スレーブサーバを制御する SQL ステートメント」。
QUERY CACHE	クエリキャッシュからすべてのクエリ結果を削除する。

4.6.6. PURGE MASTER LOGS 構文

```
PURGE {MASTER|BINARY} LOGS TO binlog_name
PURGE {MASTER|BINARY} LOGS BEFORE date
```

このコマンドは、指定したバイナリログまたは指定日より前のバイナリログをすべて削除する場合に使用します。 See 項4.11.7.「マスタサーバを制御する SQL ステートメント」。

PURGE MASTER LOGS のシノニムとして PURGE BINARY LOGS が MySQL 4.1.1 から利用可能になっています。

4.6.7. KILL 構文

```
KILL thread_id
```

mysqld の各接続はスレッドごとに行われます。SHOW PROCESSLIST コマンドで、どのスレッドが使用されているか知ることができます。また、KILL thread_id コマンドで、スレッドを強制終了できます。

PROCESS 権限があれば、すべてのスレッドを照会することができます。 SUPER 権限があれば、すべてのスレッドを強制終了できます。 上記の権限がない場合は、自分のスレッドだけ照会および強制終了できます。

mysqladmin processlist コマンドおよび mysqladmin kill コマンドでも、スレッドを確認および強制終了できます。

注意: 現在のところ、組み込み MySQL サーバライブラリでは KILL を使用できません。組み込みサーバは、ホストアプリケーションのスレッド内部で実行するだけで、独自の接続スレッドは作成しません。

KILL を実行すると、スレッド固有の kill flag がスレッドに設定されます。

キルフラグは一定間隔でチェックされるため、多くの場合、スレッドが終了するまでにしばらく時間がかかります。

- SELECT、ORDER BY、GROUP BY の各ループでは、レコードのブロックが 1 つ読み取られるたびにフラグがチェックされます。キルフラグが設定されていれば、ステートメントは停止します。
- ALTER TABLE を実行した場合、オリジナルのテーブルからレコードの各ブロックが読み取られる前に、キルフラグがチェックされます。キルフラグが設定されていればコマンドは停止し、テンポラリテーブルは削除されます。
- UPDATE または DELETE を実行した場合、各ブロックが読み取られた後、および更新レコードまたは削除レコードの後で、キルフラグがチェックされます。キルフラグが設定されていれば、ステートメントは停止します。注意: トランザクションを使用していない場合、変更はロールバックされません。
- GET_LOCK() は NULL で停止します。

- INSERT DELAYED スレッドは、そのメモリにあるすべてのレコードをすばやくフラッシュし、終了します。
- スレッドがテーブルロックハンドラ内にある場合 (Locked 状態) 、テーブルロックがすばやく解除されます。
- スレッドが write コールで空きディスク容量を待っている場合、disk full のエラーメッセージで書き込みが停止します。

4.6.8. SHOW 構文

```
  SHOW DATABASES [LIKE wild]
か SHOW [OPEN] TABLES [FROM db_name] [LIKE wild]
か SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE wild]
か SHOW INDEX FROM tbl_name [FROM db_name]
か SHOW TABLE STATUS [FROM db_name] [LIKE wild]
か SHOW STATUS [LIKE wild]
か SHOW VARIABLES [LIKE wild]
か SHOW [BDB] LOGS
か SHOW [FULL] PROCESSLIST
か SHOW GRANTS FOR user
か SHOW CREATE TABLE table_name
か SHOW MASTER STATUS
か SHOW MASTER LOGS
か SHOW SLAVE STATUS
か SHOW WARNINGS [LIMIT row_count]
か SHOW ERRORS [LIMIT row_count]
か SHOW TABLE TYPES
```

SHOW を実行すると、データベース情報、テーブル情報、カラム情報、またはサーバのステータス情報が提供されます。LIKE wild を使用した場合、wild 文字列内に SQL の '%' および '_' のワイルドカード文字を使用できます。

4.6.8.1. データベース、テーブル、カラム、およびインデックスに関する情報の取得

tbl_name FROM db_name 構文の代わりに db_name.tbl_name を使用することもできます。以下の 2 つのステートメントは同じです。

```
mysql> SHOW INDEX FROM mytable FROM mydb;
mysql> SHOW INDEX FROM mydb.mytable;
```

SHOW DATABASES は、MySQL サーバホスト上のデータベースを一覧表示します。 mysqlshow コマンドラインツールを使用しても、この一覧を取得できます。 バージョン 4.0.2 では、ユーザにグローバル SHOW DATABASES 権限がない場合、そのユーザが何らかの権限を持つデータベースのみ表示されます。

SHOW TABLES は、指定したデータベースのテーブルを一覧表示します。mysqlshow db_name コマンドを使用しても、この一覧を取得できます。

注意: ユーザにテーブルに対する権限が何もない場合、SHOW TABLES および mysqlshow db_name の出力にそのテーブルは含まれません。

SHOW OPEN TABLES は、テーブルキャッシュで現在開いているテーブルを一覧表示します。See 項5.4.7.「MySQL でのテーブルのオープンとクローズの方法」。 Comment フィールドは、テーブルが何回キャッシャされ、使用されたかを示します。

SHOW COLUMNS は、指定したテーブルのカラムを一覧表示します。FULL オプションを指定した場合、ユーザが各カラムに持つ権限も表示されます。カラム情報で示されるカラムの型は、CREATE TABLE ステートメントで指定したものと異なっている場合があります。これは、カラムの型が MySQL によって自動的に変更されることがあるためです。 See 項6.5.3.1.「カラムの暗黙的な変更」。 MySQL 4.1 から、FULL キーワードにより、カラム別コメントも表示されるようになりました。

DESCRIBE ステートメントでも SHOW COLUMNS と同様の情報が出力されます。 See 項6.6.2.「DESCRIBE 構文（カラムに関する情報の取得）」。

SHOW FIELDS は SHOW COLUMNS のシノニムで、SHOW KEYS は SHOW INDEX のシノニムです。mysqlshow db_name tbl_name および mysqlshow -k db_name tbl_name でも、テーブルのカラムまたはインデックスを一覧表示できます。

SHOW INDEX は、ODBC での SQLStatistics 呼び出しとよく似た形式でインデックス情報を返します。以下のカラムが返されます。

カラム	意味
Table	テーブル名。
Non_unique	インデックスに重複が許されない場合は 0、許される場合には 1。
Key_name	インデックス名。
Seq_in_index	インデックスのカラムシーケンス番号。1 から始まる。
Column_name	カラム名。
Collation	インデックスでのカラムのソート方法。 MySQL ではこれは 'A'（昇順）または NULL（ソートしない）になる。
Cardinality	インデックス内のユニークな値の数。 これは、isamchk -a の実行によって更新される。
Sub_part	カラムが部分的にインデックスになっている場合は、インデックスになっている文字数。 キー全体がインデックスになっている場合は NULL。
Null	カラムに NULL を含めることができれば 'YES'。
Index_type	使用されるインデックス方式。
Comment	さまざまなコメント。現在のところ、MySQL 4.0.2 より前のバージョンでは、インデックスが FULLTEXT か否かを表示。

注意: Cardinality は整数で保存された統計情報に基づいてカウントされるため、小さなテーブルについては正確であると限りません。

Null カラムおよび Index_type カラムは MySQL 4.0.2 で追加されました。

4.6.8.2. SHOW TABLE STATUS

```
SHOW TABLE STATUS [FROM db_name] [LIKE wild]
```

SHOW TABLE STATUS（バージョン 3.23 で導入）は SHOW STATUS と似た働きをしますが、各テーブルに関する多くの情報を提供します。mysqlshow --status db_name コマンドを使用しても、この一覧を取得することができます。 以下のカラムが返されます。

カラム	意味
Name	テーブル名。
Type	テーブル型。 See 章 7. MySQL のテーブル型。
Row_format	レコードの保存形式（Fixed、Dynamic、または Compressed）。
Rows	レコードの数。
Avg_row_length	レコードの平均長。
Data_length	データファイルの長さ。
Max_data_length	データファイルの最大長。 固定レコード形式では、テーブルのレコードの最大数になる。 動的レコード形式では、テーブルに保存できるデータバイト数の合計（データポインタサイズの使用が前提）。
Index_length	インデックスファイルの大きさ。
Data_free	割り当てられているが未使用のバイト数。
Auto_increment	次の自動インクリメント値。
Create_time	テーブル作成時刻。
Update_time	前回のデータファイル更新時刻。
Check_time	前回のテーブルチェック時刻。
Collation	テーブルのキャラクタセットと照合順序（4.1.1 で導入）。
Checksum	チェックサム値（ある場合）（4.1.1 で導入）。
Create_options	CREATE TABLE で使用される拡張オプション。
Comment	テーブル作成時のコメント（または MySQL がテーブル情報にアクセスできない理由）。

InnoDB テーブルでは、テーブルコメントにテーブルの空き領域が報告されます。

4.6.8.3. SHOW STATUS

SHOW STATUS は、サーバのステータス情報を提供します（mysqladmin extended-status と同様）。出力は以下のようになります（形式と数値は場合によって異なります）。

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

上記のステータス変数にはそれぞれ以下のような意味があります。

変数	意味
Aborted_clients	接続を閉じる前にクライアントが終了してしまったために中断された接続数。 See 項A.2.10.「通信エラー/Aborted connection」。
Aborted_connects	MySQL サーバへの接続試行失敗回数。 See 項A.2.10.「通信エラー/Aborted connection」。
Bytes_received	すべてのクライアントから受信したバイト数。
Bytes_sent	すべてのクライアントに送信されたバイト数。
Com_xxx	各 xxx コマンドの実行回数。

Connections	MySQL サーバへの接続試行回数。
Created_tmp_disk_tables	ステートメント実行中に、ディスク上に作成された暗黙的テンポラリテーブルの数。
Created_tmp_tables	ステートメント実行中に、メモリ上に作成された暗黙的テンポラリテーブルの数。
Created_tmp_files	mysqld が作成したテンポラリファイルの数。
Delayed_insert_threads	INSERT DELAYED ハンドラスレッドの数。
Delayed_writes	INSERT DELAYED で書き込まれたレコードの数。
Delayed_errors	エラー発生 (重複キーの可能性が高い) により、INSERT DELAYED で書き込まれたレコードの数。
Flush_commands	FLUSH コマンドの実行回数。
Handler_commit	内部 COMMIT コマンド数。
Handler_delete	テーブルからレコードが削除された回数。
Handler_read_first	最初のエントリがインデックスから読み取られた回数。 この値が大きい場合、サーバが何回もフルインデックススキャンを実行していると考えられる。たとえば、SELECT col1 FROM foo を実行したときに col1 がインデックスになっているとそうなる。
Handler_read_key	キーに基づくレコード読み取り要求の回数。この値が大きい場合、クエリおよびテーブルが適切にインデックス化されていると考えられる。
Handler_read_next	キー順序での次のレコードの読み取り要求の回数。範囲指定をしてインデックスカラムに対してクエリを実行すると、これがインクリメントされる。インデックススキャンを実行してもインクリメントされる。
Handler_read_prev	キー順序での前のレコードの読み取り要求の回数。これは主に、ORDER BY ... DESC を最適化するのに使用される。
Handler_read_rnd	固定位置に基づくレコード読み取り要求の回数。 結果のソートを必要とするクエリを多く実行すると、この値が大きくなる。
Handler_read_rnd_next	データファイルでの次のレコードの読み取り要求の回数。 テーブルスキャンが多く実行されると、この値が大きくなる。この場合、一般的に、テーブルが適切にインデックス化されていないか、クエリがインデックスを有効に利用していないかを意味する。
Handler_rollback	内部 ROLLBACK コマンド数。
Handler_update	テーブル内のレコードの更新要求回数。
Handler_write	テーブルへのレコードの挿入要求回数。
Key_blocks_used	キーキャッシュの使用ブロック数。
Key_read_requests	キャッシュからのキーブロック読み取り要求回数。
Key_reads	ディスクからのキーブロックの物理的読み取り回数。
Key_write_requests	キャッシュへのキーブロックの書き込み要求回数。
Key_writes	ディスクへのキーブロックの物理的書き込み回数。
Max_used_connections	同時使用可能な最大接続数。

Not_flushed_key_blocks	変更されたが、ディスクへのフラッシュはまだされていないキーキャッシュのキーブロック。
Not_flushed_delayed_rows	INSERT DELAY キューで書き込みを待っているレコードの数。
Open_tables	開いているテーブルの数。
Open_files	開いているファイルの数。
Open_streams	開いているストリームの数（主にログ用に使用）。
Opened_tables	開かれたテーブルの数。
Rpl_status	フェールセーフなレプリケーションのステータス（まだ使用されていない）。
Select_full_join	キーを使用しない結合の数（この値が 0 でない場合、テーブルのインデックスをチェックする必要がある）。
Select_full_range_join	参照テーブルで範囲指定の検索を使用した結合の数。
Select_range	最初のテーブルの範囲指定された部分のみを使用した結合の数（通常、この値が大きくても問題にはならない）。
Select_scan	最初のテーブルでフルスキャンを行った結合の数。
Select_range_check	各レコードの後でキー使用をチェックする、キーを使用しない結合の数（この値が 0 でない場合、テーブルのインデックスをチェックする必要がある）。
Questions	サーバに送信されたクエリの数。
Slave_open_temp_tables	スレーブスレッドによって現在開かれているテンポラリテーブルの数。
Slave_running	マスタに接続されているスレーブの場合は ON。
Slow_launch_threads	生成に slow_launch_time より時間がかかったスレッドの数。
Slow_queries	long_query_time 秒より時間がかかったクエリの数。 See 項4.10.5.「スロークエリログ」。
Sort_merge_passes	ソートアルゴリズムで必要だったマージパスの回数。この値が大きければ、sort_buffer の値を大きくすることを考慮すべきである。
Sort_range	範囲指定で行われたソートの回数。
Sort_rows	ソートされたレコードの数。
Sort_scan	テーブルのスキャンによって実行されたソートの回数。
ssl_xxx	SSL によって使用される変数（まだ導入されていない）。
Table_locks_immediate	テーブルロックがすぐに実行された回数。3.23.33 で導入。
Table_locks_waited	テーブルロックがすぐには実行されず、待機が必要だった回数。この値が大きい場合、パフォーマンス上の問題がある。まずクエリを最適化し、次にテーブルを分割するかレプリケーションを使用すべきである。3.23.33 で導入。
Threads_cached	スレッドキャッシュ内のスレッド数。
Threads_connected	現在開いている接続の数。
Threads_created	接続を処理するために作成されたスレッドの数。
Threads_running	スリープ状態になっていないスレッドの数。
Uptime	サーバの稼動秒数。

補足コメント

- Opened_tables 値が大きい場合、table_cache 変数が小さすぎると考えられる。
- Key_reads 値が大きい場合、key_buffer_size 変数が小さすぎると考えられる。キャッシュミスレートは、Key_reads/Key_read_requests で計算される。
- Handler_read_rnd 値が大きい場合、MySQL がテーブル全体をスキャンする必要のあるクエリが多すぎるか、キーを適切に使用しない結合があると考えられる。
- Threads_created 値が大きい場合、thread_cache_size 変数を大きくすることを考慮すべきである。キャッシュヒットレートは、Threads_created/Connections で計算される。
- Created_tmp_disk_tables 値が大きい場合、tmp_table_size 変数を大きくして、ディスクベースではなくメモリベースでテンポラリテーブルを取得できるようにする。

4.6.8.4. SHOW VARIABLES

```
SHOW [GLOBAL | SESSION] VARIABLES [LIKE wild]
```

SHOW VARIABLES は、MySQL システム変数の値を表示します。 GLOBAL オプションおよび SESSION オプションは MySQL 4.0.3 で追加されました。 GLOBAL では、MySQL の新規接続に使用される変数を得ることができます。 SESSION では、現在の接続に有効な値を得ることができます。オプションを特に指定しなければ、SESSION が使用されます。

デフォルト値が適当ではない場合には、mysqld 起動時にコマンドラインオプションを使用して、これらの変数のほとんどを変更できます。 See 項4.1.1.「mysqld コマンドラインオプション」。 SET ステートメントからでも、ほとんどの変数を変更できます。 See 項5.5.6.「SET 構文」。

SHOW VARIABLES の出力は以下のようになります（形式と数値は場合によって異なります）。 mysqladmin variables コマンドを使用しても同じ情報を取得できます。

```
+---------------------------------+-----------------------------+
| Variable_name                   | Value                       |
+---------------------------------+-----------------------------|
| back_log                        | 50                          |
| basedir                         | /usr/local/mysql            |
| bdb_cache_size                  | 8388572                     |
| bdb_log_buffer_size             | 32768                       |
| bdb_home                        | /usr/local/mysql            |
| bdb_max_lock                    | 10000                       |
| bdb_logdir                      |                             |
| bdb_shared_data                 | OFF                         |
| bdb_tmpdir                      | /tmp/                       |
| bdb_version                     | Sleepycat Software: ...     |
| binlog_cache_size               | 32768                       |
| bulk_insert_buffer_size         | 8388608                     |
| character_set                   | latin1                      |
| character_sets                  | latin1 big5 czech euc_kr    |
| concurrent_insert               | ON                          |
| connect_timeout                 | 5                           |
| convert_character_set           |                             |
```

```
| datadir                         | /usr/local/mysql/data/       |
| delay_key_write                 | ON                           |
| delayed_insert_limit            | 100                          |
| delayed_insert_timeout          | 300                          |
| delayed_queue_size              | 1000                         |
| flush                           | OFF                          |
| flush_time                      | 0                            |
| ft_boolean_syntax               | + -><()~*:""&|               |
| ft_min_word_len                 | 4                            |
| ft_max_word_len                 | 84                           |
| ft_query_expansion_limit        | 20                           |
| ft_stopword_file                | (built-in)                   |
| have_bdb                        | YES                          |
| have_innodb                     | YES                          |
| have_isam                       | YES                          |
| have_raid                       | NO                           |
| have_symlink                    | DISABLED                     |
| have_openssl                    | YES                          |
| have_query_cache                | YES                          |
| init_file                       |                              |
| innodb_additional_mem_pool_size | 1048576                      |
| innodb_buffer_pool_size         | 8388608                      |
| innodb_data_file_path           | ibdata1:10M:autoextend       |
| innodb_data_home_dir            |                              |
| innodb_file_io_threads          | 4                            |
| innodb_force_recovery           | 0                            |
| innodb_thread_concurrency       | 8                            |
| innodb_flush_log_at_trx_commit  | 1                            |
| innodb_fast_shutdown            | ON                           |
| innodb_flush_method             |                              |
| innodb_lock_wait_timeout        | 50                           |
| innodb_log_arch_dir             |                              |
| innodb_log_archive              | OFF                          |
| innodb_log_buffer_size          | 1048576                      |
| innodb_log_file_size            | 5242880                      |
| innodb_log_files_in_group       | 2                            |
| innodb_log_group_home_dir       | ./                           |
| innodb_mirrored_log_groups      | 1                            |
| interactive_timeout             | 28800                        |
| join_buffer_size                | 131072                       |
| key_buffer_size                 | 16773120                     |
| language                        | /usr/local/mysql/share/...   |
| large_files_support             | ON                           |
| local_infile                    | ON                           |
| locked_in_memory                | OFF                          |
| log                             | OFF                          |
| log_update                      | OFF                          |
| log_bin                         | OFF                          |
| log_slave_updates               | OFF                          |
| log_slow_queries                | OFF                          |
| log_warnings                    | OFF                          |
| long_query_time                 | 10                           |
| low_priority_updates            | OFF                          |
| lower_case_table_names          | OFF                          |
| max_allowed_packet              | 1047552                      |
| max_binlog_cache_size           | 4294967295                   |
| max_binlog_size                 | 1073741824                   |
| max_connections                 | 100                          |
```

```
| max_connect_errors              | 10                          |
| max_delayed_threads             | 20                          |
| max_heap_table_size             | 16777216                    |
| max_join_size                   | 4294967295                  |
| max_relay_log_size              | 0                           |
| max_sort_length                 | 1024                        |
| max_user_connections            | 0                           |
| max_tmp_tables                  | 32                          |
| max_write_lock_count            | 4294967295                  |
| myisam_max_extra_sort_file_size | 268435456                   |
| myisam_repair_threads           | 1                           |
| myisam_max_sort_file_size       | 2147483647                  |
| myisam_recover_options          | force                       |
| myisam_sort_buffer_size         | 8388608                     |
| net_buffer_length               | 16384                       |
| net_read_timeout                | 30                          |
| net_retry_count                 | 10                          |
| net_write_timeout               | 60                          |
| open_files_limit                | 1024                        |
| pid_file                        | /usr/local/mysql/name.pid   |
| port                            | 3306                        |
| protocol_version                | 10                          |
| query_cache_limit               | 1048576                     |
| query_cache_size                | 0                           |
| query_cache_type                | ON                          |
| read_buffer_size                | 131072                      |
| read_rnd_buffer_size            | 262144                      |
| rpl_recovery_rank               | 0                           |
| safe_show_database              | OFF                         |
| server_id                       | 0                           |
| slave_net_timeout               | 3600                        |
| skip_external_locking           | ON                          |
| skip_networking                 | OFF                         |
| skip_show_database              | OFF                         |
| slow_launch_time                | 2                           |
| socket                          | /tmp/mysql.sock             |
| sort_buffer_size                | 2097116                     |
| sql_mode                        |                             |
| table_cache                     | 64                          |
| table_type                      | MYISAM                      |
| thread_cache_size               | 3                           |
| thread_stack                    | 131072                      |
| tx_isolation                    | READ-COMMITTED              |
| timezone                        | EEST                        |
| tmp_table_size                  | 33554432                    |
| tmpdir                          | /tmp/:/mnt/hd2/tmp/         |
| version                         | 4.0.4-beta                  |
| wait_timeout                    | 28800                       |
+---------------------------------+-----------------------------+
```

以下、各オプションについて説明します。

バッファサイズ、長さ、およびスタックサイズの値はバイト単位です。'K' または 'M' のサフィックスを値に付けると、それぞれキロバイト、メガバイトを示します。たとえば、16M は 16 メガバイトです。サフィックスの大文字と小文字は区別されません。16M と 16m は同じです。

- ansi_mode。 mysqld が --ansi オプションで起動している場合は ON。 See 項1.8.2.「ANSI モードでの MySQL の実行」。
- back_log MySQL で保持できる未解決の接続要求数。これは、非常に短い間にメインの MySQL スレッドに非常に多くの接続要求が集中したときに機能する。それから、メインスレッドが接続をチェックし、新規スレッドを開始するのにいくらかの時間 (ほんのわずかであるが) がかかる。back_log 値は、MySQL が瞬間的に新規要求への回答を停止するまでのこの短時間に、スタック可能な要求の数を示す。短時間に多くの接続要求が予想される場合だけ、この値を大きくする必要がある。

  言い換えると、この値は、TCP/IP 接続のリッスンキューのサイズである。使用しているオペレーティングシステムそのものにも、このキューサイズの制限がある。詳細については、Unix listen(2) システムコールのマニュアルページを参照のこと。この変数の最大値について、OS のドキュメントを参照することが必要である。back_log をオペレーティングシステムの制限値より大きくしても、効果はない。
- basedir --basedir オプションの値。
- bdb_cache_size BDB テーブルのインデックスとレコードをキャッシュするために割り当てられたバッファ。BDB テーブルを使用しない場合、mysqld を --skip-bdb オプションで起動して、このキャッシュにメモリを使用しないようにする。
- bdb_log_buffer_size BDB テーブルのインデックスとレコードをキャッシュするために割り当てられたバッファ。BDB テーブルを使用しない場合、この値を 0 に設定するか、mysqld を --skip-bdb オプションで起動して、このキャッシュにメモリを使用しないようにする。
- bdb_home --bdb-home オプションの値。
- bdb_max_lock BDB テーブルで有効にできるロックの最大数 (デフォルトでは 10,000)。長いトランザクションを実行しているときや、mysqld が大量のレコードを使ってクエリを実行しているときに、bdb: Lock table is out of available locks または Got error 12 from ... というエラーが発生する場合には、この数値を大きくする必要がある。
- bdb_logdir --bdb-logdir オプションの値。
- bdb_shared_data --bdb-shared-data を使用している場合は ON。
- bdb_tmpdir --bdb-tmpdir オプションの値。
- binlog_cache_size。 トランザクション中にバイナリログに対する SQL ステートメントを保持するキャッシュのサイズ。大きなマルチステートメントトランザクションを頻繁に使用する場合、この値を大きくすることによりパフォーマンスを向上できる。 See 項6.7.1.「START TRANSACTION、COMMIT、ROLLBACK の各構文」。
- bulk_insert_buffer_size (以前は myisam_bulk_insert_tree_size) MyISAM は特殊なツリー状のキャッシュを使用して、バルク INSERT (INSERT ... SELECT、INSERT ... VALUES (...), (...), ...、LOAD DATA INFILE) を高速化する。この変数により、スレッドごとのキャッシュツリーのサイズが制限される (バイト単位)。この値を 0 に設定すると、最適化は無効になる。注意: このキャッシュは、空白でないテーブルに値を追加する場合にのみ使用される。デフォルト値は 8 メガバイトである。
- character_set デフォルトのキャラクタセット。
- character_sets サポートされるキャラクタセット。
- concurrent_inserts ONの場合 (デフォルト)、SELECT クエリを MyISAM テーブルで実行するとき、同時に INSERT

も実行できる。このオプションをオフにするには、mysqld を --safe または --skip-new で起動する。

- connect_timeout mysqld サーバが、Bad handshake を返すまでに接続パケットを待つ秒数。
- datadir --datadir オプションの値。
- delay_key_write MyISAM テーブルのオプション。以下のいずれかの値を指定できる。

OFF	CREATE TABLE ... DELAYED_KEY_WRITE はすべて無視される。
ON	(デフォルト) CREATE TABLE の DELAY_KEY_WRITE オプションが採用される。
ALL	新しく開かれるすべてのテーブルを、DELAY_KEY_WRITE オプションで作成されたように扱う。

DELAY_KEY_WRITE が有効の場合、このオプションのテーブルのキーバッファは、インデックスの更新ごとにはフラッシュされず、テーブルを閉じるときだけフラッシュされる。これにより、キーの書き込みはかなり速くなるが、これを使用する場合には、myisamchk --fast --force によりすべてのテーブルを自動的にチェックすべきである。

- delayed_insert_limit delayed_insert_limit レコードを挿入後、INSERT DELAYED ハンドラは、保留中の SELECT ステートメントがないかチェックする。ある場合、ハンドラは処理を続行する前に保留中のステートメントの実行を許可する。
- delayed_insert_timeout INSERT DELAYED スレッドが INSERT ステートメントを待機する時間。
- delayed_queue_size INSERT DELAYED を処理するために割り当てられるキューのサイズ (レコード単位) 。キューがいっぱいになると、INSERT DELAYED を実行するすべてのクライアントは、キューに空きができるまで待機する。
- flush MySQL を --flush オプションで起動した場合は ON。
- flush_time これを 0 以外の値に設定すると、flush_time 秒ごとにすべてのテーブルが閉じられる (リソースの解放と未フラッシュデータのディスクへの同期のため) 。Windows 9x/Me、その他リソースが非常に少ないシステムの場合だけ、このオプションを推奨。
- ft_boolean_syntax MATCH ... AGAINST(... IN BOOLEAN MODE) でサポートされている演算子の一覧。 See 項6.8.「MySQL 全文検索」。
- ft_min_word_len FULLTEXT インデックス内の単語の最短長。 注意: この変数を変更後、FULLTEXT インデックスを再ビルドする必要がある。このオプションは MySQL 4.0 で導入された。
- ft_max_word_len FULLTEXT インデックス内の単語の最大長。 注意: この変数を変更後、FULLTEXT インデックスを再ビルドする必要がある。このオプションは MySQL 4.0 で導入された。
- ft_query_expansion_limit クエリ拡張 (MATCH ... AGAINST (... WITH QUERY EXPANSION)) で使用される最上位マッチ数。このオプションは MySQL 4.1.1 で導入された。
- ft_stopword_file 全文検索のストップワードリストが含まれているファイル。ファイル内のすべてのストップワードが使用される。コメントは使用されない。デフォルトでは、ストップワードの組み込みリストが使用される (myisam/ft_static.c で定義されているリスト) 。このパラメータを空白の文字列 ("") に設定すると、ストップワードのフィルタリングが無効になる。注意: この変数を変更後、FULLTEXT インデックスを再ビルドする必要がある。このオプションは MySQL 4.0.10 で導入された。

- have_innodb mysqld が InnoDB テーブルをサポートする場合は YES。--skip-innodb が使用されている場合は DISABLED。
- have_bdb mysqld が Berkeley DB テーブルをサポートする場合は YES。--skip-bdb が使用されている場合は DISABLED。
- have_raid mysqld が RAID オプションをサポートする場合は YES。
- have_openssl mysqld がクライアントとサーバ間のプロトコルに SSL（暗号化）をサポートする場合は、YES。
- init_file サーバの起動時に --init-file オプションで指定されたファイルの名前。これは、サーバの起動時に、サーバが実行する SQL ステートメントのファイルである。
- interactive_timeout 対話式接続を終了する前に、サーバがアクティビティを待機する秒数。対話型クライアントは、mysql_real_connect() で CLIENT_INTERACTIVE オプションを使用するクライアントとして定義される。wait_timeout も参照のこと。
- join_buffer_size 完全結合（インデックスを使用しない結合）に使用されるバッファのサイズ。2 つのテーブル間の完全結合ごとにバッファが割り当てられる。インデックスを追加できない場合に、完全結合を速くする目的でこの値を大きくする（通常、結合を速くする最良の方法はインデックスの追加である）。
- key_buffer_size インデックスブロックはバッファされ、すべてのスレッドによって共有される。key_buffer_size は、インデックスブロック用に使用されるバッファのサイズである。

  インデックス処理（すべての読み取りと複数書き込み）のパフォーマンスを向上させるため、この値は可能な限り大きくする。主に MySQL を実行する 256M マシンでは、64M が一般的である。ただし、これを大きくしすぎると（たとえば、メモリ合計の 50% 以上など）、システムがページングを開始し、処理速度が非常に遅くなる可能性がある。MySQL ではデータ読み取りをキャッシュしないため、OS ファイルシステムのキャッシュ用に領域を確保しておくことが必要である。

  キーバッファのパフォーマンスを確認するには、SHOW STATUS を実行し、Key_read_requests、Key_reads、Key_write_requests、Key_writes の各変数を調べる。Key_reads/Key_read_request の比率は通常、0.01 より小さい。Key_write/Key_write_requests の比率は、操作がほとんど更新と削除だけの場合、通常は約 1 になる。しかし、同時に多くの影響がある更新を頻繁に行う場合や、DELAY_KEY_WRITE を使用する場合はかなり小さくなることもある。See 項4.6.8.「SHOW 構文」。

  多くのレコードを同時に書き込む場合にさらに速度を上げるには、LOCK TABLES を使用する。See 項6.7.5.「LOCK TABLES および UNLOCK TABLES 構文」。
- language エラーメッセージに使用する言語。
- large_file_support mysqld が、大きなファイルをサポートするオプションでコンパイルされている場合。
- locked_in_memory mysqld が、--memlock でメモリにロックされている場合。
- log すべてのクエリのログを有効にしている場合。
- log_update 更新ログを有効にしている場合。
- log_bin バイナリログを有効にしている場合。
- log_slave_updates スレーブからの更新をログに記録する場合。

- long_query_time クエリがこの値（秒単位）より時間がかかると、Slow_queriesカウンタがインクリメントされる。--log-slow-queries を使用している場合、クエリはスロークエリログファイルに記録される。この値は実際の時間であり、CPU 時間ではない。したがって、負荷の軽いシステムではしきい値より下のクエリも、負荷の重いシステムではしきい値より上になる場合がある。 See 項4.10.5.「スロークエリログ」。

- lower_case_table_names 1 に設定すると、テーブル名が小文字でディスクに格納され、テーブル名の比較で大文字と小文字が区別されなくなる。バージョン 4.0.2 以降、このオプションはデータベース名にも適用されるようになった。4.1.1 からは、テーブルエイリアスにも適用されるようになった。 See 項6.1.3.「名前におけるケース依存」。

- max_allowed_packet 1 パケットの最大サイズ。メッセージバッファは net_buffer_length バイトに初期化されるが、必要に応じて max_allowed_packet バイトまで大きくできる。このデフォルト値は、大きな（おそらく不正）パケットを受けるには小さすぎる。大きな BLOB カラムを使用している場合には、この値を大きくする必要がある。使用する最大の BLOB と同じ大きさにすべきである。max_allowed_packet のプロトコル制限は MySQL 3.23 で 16MB、MySQL 4.0 で 1GB である。

- max_binlog_cache_size トランザクションでこれより多くのメモリが必要になると、"Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage" エラーが発生する。

- max_binlog_size 3.23.33 で導入された。バイナリ（レプリケーション）ログファイルへの書き込みが、この値を超える場合、ログをローテートする。設定可能値は、4096 バイト（MySQL バージョン 4.0.14 より前は 1024）から 1 ギガバイト。デフォルトは 1GB。注意: トランザクションを使用している場合、トランザクションは 1 つのまとまりでバイナリログに書き込まれるため、複数のバイナリログに分割されることはない。したがって、大きなトランザクションがある場合、バイナリログが max_binlog_size より大きくなる可能性がある。max_relay_log_size（MySQL 4.0.14 で導入）が 0 の場合、max_binlog_size がリレーログにも適用される。

- max_connections 認められる同時クライアントの数。この値を大きくすると、mysqld が必要とするファイル記述子の数が多くなる。ファイル記述子の制限については、以下を参照のこと。 See 項A.2.6.「Too many connections エラー」。

- max_connect_errors ホストからの接続中断回数がこの値より大きくなった場合、それ以降、そのホストからの接続はブロックされる。ホストのブロックを解除するには、FLUSH HOSTS コマンドを使用する。

- max_delayed_threads INSERT DELAYED ステートメント処理に、この数より多くのスレッドを開始しない。すべての INSERT DELAYED スレッドが使用された後にデータを新規テーブルに挿入しようとすると、そのレコードは DELAYED 属性が指定されていないものとして挿入される。これを 0 に設定すると、max_delayed スレッドは生成されない。

- max_heap_table_size この変数は、それ以降、新しく生成される HEAP テーブルの最大サイズを設定する。この変数の値を使用して、HEAP テーブルの MAX_ROWS 値が計算される。この変数の設定は、既存の HEAP テーブルには影響しない。ただし、CREATE TABLE、TRUNCATE TABLE などのステートメントで再生成したり、ALTER TABLE で変更した場合は適用される。

- max_join_size max_join_size を超えるレコードを読み取る結合ではエラーを出力する。WHERE 節なしで時間がかかる何百万ものレコードを返す結合を実行するユーザがいる場合に、この値を設定する。

- max_relay_log_size 4.0.14 で導入。リレーログ（レプリケーションスレーブによって使用されるログ。see 項4.11.3.「レプリケーションの実装の詳細」）への書き込みが指定値を超える場合、リレーログをローテートする。この変数により、リレーログとバイナリログに異なるサイズの制約を加えることができる。ただし、この変数を 0 に設定すると、max_binlog_size がバイナリログとリレーログの両方に適用される。max_relay_log_size には 0、または 4097 以上

1GB 未満の値を設定できる。デフォルト値は 0。

- max_seeks_for_key キーによるレコード検索の最大回数を制限する。MySQL オプティマイザは、キースキャンでテーブルからレコードを検索するときに、キーの基数には関係なく、キー検索の回数をこの指定値までと想定する。この値を低く（100 くらいに）設定することにより、MySQL はテーブルスキャンではなくキースキャンを優先的に行うようになる。
- max_sort_length BLOB 値または TEXT 値をソートするときに使用するバイト数（各値の最初の max_sort_length バイトのみが使用され、残りは無視される）。
- max_user_connections 単一ユーザのアクティブ接続の最大数（0 = 制限なし）。
- max_tmp_tables このオプションはまだ利用不可。1 つのクライアントが同時に開いておけるテンポラリテーブルの最大数。
- max_write_lock_count この回数の書き込みロック後、間に読み取りロックを認める。
- myisam_recover_options --myisam-recover オプションの値。
- myisam_sort_buffer_size REPAIR を実行するとき、あるいは CREATE INDEXまたは ALTER TABLE を使用してインデックスを作成するときに、インデックスのソートに割り当てられるバッファ。
- myisam_max_extra_sort_file_size。 インデックスの高速作成で使用されるテンポラリファイルが、キーキャッシュを使用する場合より、指定されているサイズ分大きい場合、キーキャッシュ方式が使用される。これは主に、大きなテーブルの長い文字キーに、処理速度が遅いキーキャッシュ方式を用いてインデックスを作成させる場合に使用する。注意: このパラメータの値は、4.0.3 より前のバージョンではメガバイト単位、このバージョンからはバイト単位。
- myisam_repair_threads。 この値が 1 より大きい場合、Repair by sorting プロセス中、MyISAM テーブルインデックスが、それぞれ独自のスレッドで平行して作成される。注意: マルチスレッド修復は、現在まだ alpha 段階。
- myisam_max_sort_file_size REPAIR、ALTER TABLE、または LOAD DATA INFILE を使用したインデックスの再生成時、MySQL が使用できるテンポラリファイルの最大サイズ。ファイルサイズがこれより大きい場合、インデックスはキーキャッシュによって作成される（時間がかかる）。注意: このパラメータの値は、4.0.3 より前のバージョンではメガバイト単位、このバージョンからはバイト単位。
- net_buffer_length クエリとクエリの間、通信バッファはこのサイズにリセットされる。通常、これは変更すべきでないが、メモリが非常に小さい場合には、予測されるクエリサイズに設定できる。これは、クライアントによって送信される SQL ステートメントの予測される長さ。ステートメントがこの長さを超えた場合、バッファは自動的に max_allowed_packet バイトまで拡大される。
- net_read_timeout 読み取りを停止する前に、接続からのデータを待機する秒数。注意: 接続からのデータが期待されない場合、タイムアウトは write_timeout によって定義される。slave_net_timeout も参照のこと。
- net_retry_count 通信ポートの読み取りが中断した場合に、実行できる再試行回数。FreeBSD では、内部中断がすべてのスレッドに通知されるため、この値を大きくすべきである。
- net_write_timeout 書き込みを停止する前に、ブロックが接続に書き込まれるのを待つ秒数。
- open_files_limit システムが mysqld に開くことを許可するファイルの数。これはシステムに指定されている実際の値であり、起動時のパラメータとして mysqld に指定したものとは異なっている場合がある。MySQL がオープンファイル数を変更できないシステムでは 0 になる。

- pid_file --pid-file オプションの値。
- port --port オプションの値。
- protocol_version MySQL サーバによって使用されるプロトコルバージョン。
- range_alloc_block_size 範囲の最適化で割り当てられるブロックのサイズ。
- read_buffer_size (以前は record_buffer) 順次スキャンを実行する各スレッドは、スキャンする各テーブルにこのサイズのバッファを割り当てる。多くの順次スキャンを実行する場合には、この値を大きくすることが必要である。
- read_rnd_buffer_size (以前は record_rnd_buffer) ソート後、ソートされた順序でレコードを読み取る際、ディスク検索を行わないように、レコードをこのバッファから読み取る。この値を大きく設定すると、ORDER BY を大幅に向上できる。これはスレッド固有の変数であるため、これをグローバルに大きく設定すべきではないが、いくつかの特別大きなクエリを実行するときだけ変更する。
- query_alloc_block_size クエリの解析および実行時に作成されるオブジェクトに割り当てられるメモリ割り当てブロックのサイズ。メモリの断片化に問題がある場合、これを少し大きくすると改善する可能性がある。
- query_cache_limit これより大きい結果はキャッシュしない (デフォルトは 1M) 。
- query_cache_size 古いクエリの結果を保存するために割り当てられたメモリ。この値を 0 にすると、クエリキャッシュは無効になる (デフォルト) 。
- query_cache_type 次のいずれかの値 (数値のみ) を設定できる。

値	エイリアス	コメント
0	OFF	結果のキャッシュおよび読み込みを行わない。
1	ON	SELECT SQL_NO_CACHE ... クエリを除くすべての結果をキャッシュする。
2	DEMAND	SELECT SQL_CACHE ... クエリのみキャッシュする。

- query_prealloc_size クエリの解析および実行時に使用される永続バッファのサイズ。このバッファはクエリとクエリの間でも解放されない。理論的には、この値を ``十分大きく" 設定することにより、MySQL は malloc() 呼び出しを 1 回も実行することなくクエリを実行できる。
- safe_show_database ユーザがデータベース権限またはテーブル権限をまったく持たないデータベースは表示しない。これは、他のユーザが権限を持つデータベースを表示することができなくなるので、セキュリティの向上に役立つ。skip_show_database も参照のこと。
- server_id --server-id オプションの値。
- skip_locking mysqld が外部ロックを使用する場合は OFF。
- skip_networking ローカル (ソケット) 接続のみを認める場合は ON。
- skip_show_database ユーザに PROCESS 権限がない場合、SHOW DATABASES の実行を認めない。これは、他のユーザが権限を持つデータベースを表示できなくなるので、セキュリティの向上に役立つ。safe_show_database も参照のこと。
- slave_net_timeout 読み取りを停止する前に、マスタ/スレーブ接続からのデータを待機する秒数。

- slow_launch_time スレッドの作成にこの値（秒単位）より時間がかかれば、Slow_launch_threads カウンタがインクリメントされる。
- socket サーバによって使用される Unix ソケット。
- sort_buffer_size ソートを実行する必要のある各スレッドがこのサイズのバッファを割り当てる。ORDER BY 操作または GROUP BY 操作を速くするには、この値を大きくする。 See 項A.4.4.「MySQL がテンポラリファイルを格納する場所」。
- table_cache すべてのスレッドでのオープンテーブル数。この値を大きくすると、mysqld が必要とするファイル記述子の数が多くなる。テーブルキャッシュを大きくする必要があるかどうかチェックするには、Opened_tables 変数を調べる。 See 項4.6.8.3.「SHOW STATUS」。 この変数が大きく、FLUSH TABLES（すべてのテーブルを閉じて再び開く）をあまり実行しないのであれば、table_cache 変数の値を大きくすることが必要である。

  テーブルキャッシュの詳細については、項5.4.7.「MySQL でのテーブルのオープンとクローズの方法」 を参照のこと。
- table_type デフォルトのテーブル型。
- thread_cache_size 再利用のためにキャッシュに保持するスレッド数。クライアントが接続を切断したときに、以前のスレッド数が thread_cache_size 以下であれば、そのクライアントのスレッドはキャッシュに入る。新しいスレッドはすべてキャッシュから取り込まれ、キャッシュが空の場合のみ、新しいスレッドが作成される。新しい接続が多く発生する場合、この変数を大きくすることによりパフォーマンスを向上できる（スレッド実装が既に理想的な状態であれば、それほどパフォーマンスは向上しない）。Connections ステータス変数と Threads_created ステータス変数（詳細については、see 項4.6.8.3.「SHOW STATUS」）の差異を調べることにより、スレッドキャッシュの効率性を確認できる。
- thread_concurrency Solaris では、mysqld はこの値で thr_setconcurrency() を呼び出す。 thr_setconcurrency() により、アプリケーションは、同時に実行する理想的なスレッド数をスレッドシステムに指定する。
- thread_stack 各スレッドのスタックサイズ。crash-me テストによって検知される制限の多くがこの値に依存する。通常の操作ではデフォルトで十分である。 See 項5.1.4.「MySQL ベンチマークスィート」。
- timezone サーバのタイムゾーン。
- tmp_table_size メモリ内のテンポラリテーブルがこのサイズを超えると、MySQL は自動的にこれをディスク上の MyISAM テーブルに変換する。詳細な GROUP BY クエリを頻繁に行い、メモリに余裕がある場合は、tmp_table_size 値を大きくする。
- tmpdir テンポラリファイルおよびテンポラリテーブル用ディレクトリ。MySQL 4.1 以降、複数のパスをコロン : (Windows ではセミコロン ;) で区切って設定できるようになっている。これは、ラウンドロビン方式で使用される。この機能は、複数の物理ディスクに負荷を分散する目的で使用できる。メモリベースのファイルシステムを指すように tmpdir を設定することは可能。ただし、MySQL サーバがスレーブの場合はできない。スレーブの場合、マシンがリブートしてもテンポラリテーブルのレプリケーションまたは LOAD DATA INFILE のレプリケーション処理を続行するためのテンポラリファイルが必要となる。そのため、マシンのリブートで消去されるメモリベースの tmpdir は適しない。ディスクベースの tmpdir が必要。
- transaction_alloc_block_size コミットするときに、バイナリログに保存されるトランザクションの一部であるクエリを保存するために割り当てられるメモリ割り当てブロックのサイズ。

- transaction_prealloc_block_size transaction_alloc_blocks の永続的バッファ。クエリとクエリの間でも解放されない。通常のトランザクションのすべてのクエリに対応できるように ``十分に大きく" 設定すれば、malloc() を何度も呼び出さないで済む。
- version サーバのバージョン番号。
- wait_timeout 対話式ではない接続を終了する前に、サーバがアクティビティを待機する秒数。

  スレッド開始時、SESSION.WAIT_TIMEOUT は、GLOBAL.WAIT_TIMEOUT または GLOBAL.INTERACTIVE_TIMEOUT によって初期化される。どちらになるかは、クライアントのタイプ (CLIENT_INTERACTIVE 接続オプションで定義) に依存する。interactive_timeout も参照のこと。

これら変数のチューニング方法については、MySQL のチューニングに関するセクションを参照してください。 See 項 5.5.2.「サーバパラメータのチューニング」。

4.6.8.5. SHOW [BDB] LOGS

SHOW LOGS は、既存のログファイルに関するステータス情報を表示します。現在は、Berkeley DB ログファイルの情報を表示するだけなので、そのエイリアスは (MySQL 4.1.1 で導入) SHOW BDB LOGS です。

- File は、ログファイルのフルパスを示す。
- Type は、ログファイルのタイプ (Berkeley DB ログファイルでは BDB) を示す。
- Status は、ログファイルのステータス (ファイルを削除できる場合は FREE、ファイルがトランザクションサブシステムによって必要とされている場合は IN USE) を示す。

4.6.8.6. SHOW PROCESSLIST

SHOW [FULL] PROCESSLIST は、実行中のスレッドを表示します。 mysqladmin processlist コマンドでもこの情報を取得できます。SUPER 権限があれば、すべてのスレッドを表示できます。この権限がない場合は、自分のスレッドのみ表示できます。 See 項4.6.7.「KILL 構文」。 FULL オプションを使用していない場合、各クエリの最初の 100 文字だけが表示されます。

4.0.12 以降、MySQL は TCP/IP 接続のホスト名を hostname:client_port 形式で報告するので、どのクライアントが何を行っているかわかりやすくなっています。

このコマンドは、'too many connections' エラーメッセージが表示され、原因を知りたいときに便利です。MySQL は、SUPER 権限を持つクライアントのために 1 つの特別接続枠を予約しており、いつでもシステムにログインしてチェックできるようになっています (ユーザ全員にこの権限を与えていないようにしてください)。

mysqladmin processlist で確認できる一般的な状態は以下のとおりです。

- Checking table スレッドがテーブルの自動チェックを実行している。
- Closing tables スレッドが、変更されたテーブルデータをディスクにフラッシュし、使用したテーブルを閉じている。これには通常それほど時間がかからない。時間がかかる場合、ディスクの使用率をチェックする必要がある。
- Connect Out マスタに接続しているスレーブ。

- Copying to tmp table on disk テンポラリ結果セットが tmp_table_size よりも大きく、スレッドがメモリベースのテンポラリテーブルをディスクベースに変更して、メモリの節約を図っている。
- Creating tmp table スレッドは、クエリの結果の一部を保持するためのテンポラリテーブルを作成中。
- deleting from main table 複数テーブルを削除する最初の段階で、最初のテーブルを削除中。
- deleting from reference tables 複数テーブルを削除する 2 番目の段階で、他のテーブルから、一致したレコードを削除中。
- Flushing tables スレッドが FLUSH TABLES を実行中。すべてのスレッドによりそのテーブルが閉じられるのを待っている。
- Killed 誰かがスレッドを強制終了の命令を出したため、次回のキルフラグチェック時に強制終了される。MySQL では大きな各ループでフラグがチェックされるが、それでもスレッド終了には少し時間がかかる場合がある。スレッドが他のスレッドによってロックされている場合、そのロックが解除されたところで強制終了が実行される。
- Sending data スレッドは SELECT ステートメントのレコードを処理中で、かつクライアントにデータを送信中。
- Sorting for group スレッドは、GROUP BY のソートを実行中。
- Sorting for order スレッドは、ORDER BY のソートを実行中。
- Opening tables スレッドがテーブルを開こうとしている。これは、何かが妨害していなければすぐに終わるはずである。たとえば、ALTER TABLE や LOCK TABLE などにより、そのコマンドの終了時までテーブルが開かないことがある。
- Removing duplicates クエリで SELECT DISTINCT が使用されたが、MySQL は初期段階で重複を除外する最適化を実行できなかった。このため、MySQL は結果をクライアントに送信する前に、重複レコードを削除する段階を踏む必要がある。
- Reopen table スレッドはテーブルのロックを取得したが、ロック取得後、下のテーブル構造が変更されていることを認識した。このため、ロックを解除し、テーブルを閉じて、再び開こうとしている。
- Repair by sorting 修復コードがソートを使用してインデックスを作成している。
- Repair with keycache 修復コードが、キーキャッシュにより、キーを 1 つずつ作成している。これは、Repair by sorting よりも大幅に時間がかかる。
- Searching rows for update スレッドがレコード更新の初期段階として、更新対象の一致レコードを検索中である。レコード検索に使用するインデックスを UPDATE が変更すると、この段階が必要となる。
- Sleeping スレッドが、クライアントから新しいコマンドが送信されるのを待っている。
- System lock スレッドが、テーブルの外部システムロックを待っている。同じテーブルにアクセスする複数の mysqld サーバを使用していない場合、--skip-external-locking オプションでシステムロックを無効にできる。
- Upgrading lock INSERT DELAYED ハンドラが、レコード挿入のためにテーブルをロックしようとしている。
- Updating スレッドが更新対象レコードを検索して更新している。
- User Lock スレッドが GET_LOCK() を待っている。

- Waiting for tables 下のテーブル構造が変更されているため、テーブルを開き直して新しい構造を取得する必要があるという通知をスレッドが受け取った。テーブルを開き直すためには、他のすべてのスレッドがそのテーブルを閉じるのを待つ必要がある。

  この通知は、他のスレッドがそのテーブルに対して FLUSH TABLES、FLUSH TABLES table_name、ALTER TABLE、RENAME TABLE、REPAIR TABLE、ANALYZE TABLE、OPTIMIZE TABLE のいずれかを実行している場合に発生する。

- waiting for handler insert INSERT DELAYED ハンドラがすべての挿入処理を完了し、新規の挿入を待機中である。

ほとんどの状態はすぐに終わります。何秒も同じ状態が続く場合は、問題のある可能性があるので、調査が必要です。

他にも説明していない状態がありますが、そのほとんどは mysqld のバグ発見に役立つものです。

4.6.8.7. SHOW GRANTS

SHOW GRANTS FOR user は、ユーザの権限を複製するために発行する必要がある grant コマンドを列挙します。

```
mysql> SHOW GRANTS FOR root@localhost;
+---------------------------------------------------------------------+
| Grants for root@localhost                                           |
+---------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+---------------------------------------------------------------------+
```

現在のセッションの権限を列挙するには、CURRENT_USER() 関数 (バージョン 4.0.6 で導入) を使用して、そのセッションで認証されているユーザを確認できます。 See 項6.3.6.2.「その他の各種関数」。

4.6.8.8. SHOW CREATE TABLE

指定したテーブルを作成する CREATE TABLE ステートメントを表示します。

```
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 で表示されるテーブル名とカラム名は、SQL_QUOTE_SHOW_CREATE オプションの値に従って引用符で囲まれます。 項5.5.6.「SET 構文」。

4.6.8.9. SHOW WARNINGS | ERRORS

```
SHOW WARNINGS [LIMIT row_count]
SHOW ERRORS [LIMIT row_count]
```

このコマンドは、MySQL 4.1.0 で導入されました。

これは、前回のコマンドでのエラー、警告、およびコメントを表示します。エラーおよび警告は、テーブルを使用する新規コマンドごとにリセットされます。

MySQL サーバは、前回のコマンドで発生した警告とエラーの合計数を返します。これは、mysql_warning_count() を呼び出すことによって取得できます。

max_error_count までのメッセージが保存されます (グローバルおよびスレッド固有変数) 。

@error_count でエラー数を、@warning_count で警告数を取得できます。

SHOW WARNINGS では前回のコマンドで発生したエラー、警告、およびコメントがすべて表示され、SHOW ERRORS ではエラーだけが表示されます。

```
mysql> DROP TABLE IF EXISTS no_such_table;
mysql> SHOW WARNINGS;

+-------+------+-----------------------------+
| Level | Code | Message                     |
+-------+------+-----------------------------+
| Note  | 1051 | Unknown table 'no_such_table' |
+-------+------+-----------------------------+
```

注意: MySQL 4.1.0 では警告のフレームワークが追加されただけなので、MySQL コマンドの多くは警告を生成しません。4.1.1 では、LOAD DATA INFILE、および INSERT、UPDATE、ALTER などの DML ステートメントのあらゆる種類の警告をサポートします。

以下、挿入ステートメントの変換警告を生成する簡単な例を示します。

```
mysql> create table t1(a tinyint NOT NULL, b char(4));
Query OK, 0 rows affected (0.00 sec)

mysql> insert into t1 values(10,'mysql'),(NULL,'test'),(300,'open source');
Query OK, 3 rows affected, 4 warnings (0.15 sec)
Records: 3  Duplicates: 0  Warnings: 4

mysql> show warnings;
+---------+------+---------------------------------------------------------------+
| Level   | Code | Message                                                       |
+---------+------+---------------------------------------------------------------+
| Warning | 1263 | Data truncated for column 'b' at row 1                        |
| Warning | 1261 | Data truncated, NULL supplied to NOT NULL column 'a' at row 2 |
| Warning | 1262 | Data truncated, out of range for column 'a' at row 3          |
| Warning | 1263 | Data truncated for column 'b' at row 3                        |
+---------+------+---------------------------------------------------------------+
4 rows in set (0.00 sec)
```

警告の最大数は、サーバ変数 'max_error_count'、SET max_error_count=[count] を使用して指定できます。デフォルトは 64 です。この変数を '0' にリセットすればこの設定を無効できます。max_error_count が 0 の場合でも、警告が何回発生したか表示されますが、メッセージは保存されません。

たとえば、上記の例での以下の ALTER テーブルステートメントの場合、max_error_count=1 に設定すると、警告が 3 回発生しても、返される警告メッセージは 1 つだけになります。

```
mysql> show variables like 'max_error_count';
```

```
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| max_error_count | 64    |
+-----------------+-------+
1 row in set (0.00 sec)

mysql> set max_error_count=1;
Query OK, 0 rows affected (0.00 sec)

mysql> alter table t1 modify b char;
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 3

mysql> show warnings;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1263 | Data truncated for column 'b' at row 1 |
+---------+------+----------------------------------------+
1 row in set (0.00 sec)

mysql>
```

4.6.8.10. SHOW TABLE TYPES

```
SHOW TABLE TYPES
```

このコマンドは、MySQL 4.1.0 で導入されました。

SHOW TABLE TYPES は、テーブル型に関するステータス情報を表示します。このコマンドは、テーブル型がサポートされているかどうか、およびデフォルトのテーブル型を確認するのに使用されます。

```
mysql> SHOW TABLE TYPES;

+--------+---------+------------------------------------------------------------+
| Type   | Support | Comment                                                    |
+--------+---------+------------------------------------------------------------+
| MyISAM | DEFAULT | Default type from 3.23 with great performance              |
| HEAP   | YES     | Hash based, stored in memory, useful for temporary tables |
| MERGE  | YES     | Collection of identical MyISAM tables                      |
| ISAM   | YES     | Obsolete table type; Is replaced by MyISAM                 |
| InnoDB | YES     | Supports transactions, row-level locking and foreign keys |
| BDB    | NO      | Supports transactions and page-level locking               |
+--------+---------+------------------------------------------------------------+
6 rows in set (0.00 sec)
```

'Support' フィールドの DEFAULT は、そのテーブル型がサポートされていることと、テーブル型がデフォルトであることを示します。サーバを --default-table-type=InnoDB で起動した場合、InnoDB の 'Support' フィールド値が DEFAULT になります。

4.6.8.11. SHOW PRIVILEGES

```
SHOW PRIVILEGES
```

このコマンドは、MySQL 4.1.0 で導入されました。

SHOW PRIVILEGES は、下の MySQL サーバがサポートするシステム権限を一覧表示します。

```
mysql> show privileges;
+-----------+-------------------------+-----------------------------------------------------+
| Privilege  | Context                 | Comment                                             |
+-----------+-------------------------+-----------------------------------------------------+
| Select     | Tables                  | To retrieve rows from table                         |
| Insert     | Tables                  | To insert data into tables                          |
| Update     | Tables                  | To update existing rows                             |
| Delete     | Tables                  | To delete existing rows                             |
| Index      | Tables                  | To create or drop indexes                           |
| Alter      | Tables                  | To alter the table                                  |
| Create     | Databases,Tables,Indexes | To create new databases and tables                  |
| Drop       | Databases,Tables        | To drop databases and tables                        |
| Grant      | Databases,Tables        | To give to other users those privileges you possess |
| References | Databases,Tables        | To have references on tables                        |
| Reload     | Server Admin            | To reload or refresh tables, logs and privileges    |
| Shutdown   | Server Admin            | To shutdown the server                              |
| Process    | Server Admin            | To view the plain text of currently executing queries |
| File       | File access on server   | To read and write files on the server               |
+-----------+-------------------------+-----------------------------------------------------+
14 rows in set (0.00 sec)
```

4.7. MySQL のローカライズと国際的な使用

4.7.1. データおよびソート用キャラクタセット

デフォルトでは、MySQL はスウェーデン語およびフィンランド語に従ってソートされる ISO-8859-1 (Latin1) キャラクタセットを使用します。これは、米国と西ヨーロッパに適したキャラクタセットです。

標準 MySQL バイナリは、--with-extra-charsets=complex でコンパイルされます。これにより、すべての標準プログラムにコードが追加され、バイナリ内で latin1 とすべてのマルチバイトキャラクタセットを処理できるようになります。他のキャラクタセットが必要な場合は、キャラクタセット定義ファイルからロードします。

キャラクタセットにより、名前に使用できる文字と、SELECT ステートメントの ORDER BY 節および GROUP BY 節での文字列のソート方法が決定します。

サーバ起動時に --default-character-set オプションでキャラクタセットを変更できます。利用可能なキャラクタセットは、--with-charset=charset オプションと --with-extra-charsets= list-of-charset | complex | all | none オプションが configure になっているかどうか、および SHAREDIR/charsets/Index に含まれているキャラクタセット設定ファイルに依存します。See 項2.3.3.「一般的な configure オプション」。

MySQL の実行中にキャラクタセットを変更した場合 (ソート順序が変わる可能性もあります)、すべてのテーブルで myisamchk -r -q --set-character-set=charset を実行する必要があります。このことを行わない場合、インデックスが正しい順序でなくなる可能性があります。

クライアントが MySQL サーバに接続すると、サーバはデフォルトのキャラクタセットをクライアントに送ります。クライアントは、この接続用としてそのキャラクタセットに切り替えます。

SQL クエリの文字列をエスケープする場合は、mysql_real_escape_string() を使用してください。最初のパラメータとして

MYSQL 接続ハンドルを使用すること以外、mysql_real_escape_string() は旧 mysql_escape_string() 関数と同じです。

サーバのインストールパス以外のパスでクライアントがコンパイルされており、MySQL をコンフィギャしたユーザがすべてのキャラクタセットを MySQL バイナリに組み込んでいない場合、サーバがクライアントとは異なるキャラクタセットで実行しているときに必要となる追加キャラクタセットの場所を、クライアントに指定する必要があります。

これは、MySQL オプション設定ファイルで指定することができます。

```
[client]
character-sets-dir=/usr/local/mysql/share/mysql/charsets
```

ここで、パスは、動的 MySQL キャラクタセットが保存されているディレクトリを指します。

以下のように、クライアントに特定のキャラクタセットの使用を強制することも可能です。

```
[client]
default-character-set=character-set-name
```

しかし、通常は必要ありません。

4.7.1.1. ドイツ語キャラクタセット

ドイツ語のソート順序が必要な場合、mysqld を --default-character-set=latin1_de で起動します。これには、以下のような特徴があります。

文字列のソートおよび比較時、比較の実行前に以下の文字列のマッピングが実行される。

```
ä -> ae
ö -> oe
ü -> ue
ß -> ss
```

アクセント文字はすべて、対応するアクセントなしの大文字に変換されます。また、すべての文字が大文字に変換されます。

文字列を LIKE で比較する際には、1 文字から 2 文字へのマッピングは実行されません。すべての文字が大文字に変換されます。Ü、ü、Ö、ö、Ä、および ä 以外、すべての文字からアクセントが削除されます。

4.7.2. 英語以外のエラーメッセージ

mysqld では、以下の言語でエラーメッセージを出力できます。 チョコ語、デンマーク語、オランダ語、英語 (デフォルト) 、エストニア語、フランス語、ドイツ語、ギリシャ語、ハンガリー語、イタリア語、日本語、韓国語、ノルウェー語、新ノルウェー語、ポーランド語、ポルトガル語、ルーマニア語、ロシア語、スロバキア語、スペイン語、およびスウェーデン語。

特定の言語で mysqld を起動するには、--language=lang または -L lang オプションを使用します。次に例を示します。

```
shell> mysqld --language=swedish
```

または

```
shell> mysqld --language=/usr/local/share/swedish
```

注意: 言語名はすべて小文字で指定します。

言語ファイルは (デフォルトで) mysql_base_dir/share/LANGUAGE/ にあります。

エラーメッセージファイルを更新するには、errmsg.txt ファイルを編集し、以下のコマンドを実行して errmsg.sys ファイルを生成します。

```
shell> comp_err errmsg.txt errmsg.sys
```

MySQL を新しいバージョンにアップグレードした場合、この変更を、新規 errmsg.txt ファイルでも同じように行ってください。

4.7.3. 新しいキャラクタセットの追加

他のキャラクタセットを MySQL に追加するには、以下の手順を実行します。

キャラクタセットがシンプルなものかコンプレックスなものかを判断します。キャラクタセットが、ソートのために特殊文字照合ルーチンを必要とせず、マルチバイト文字のサポートも必要なければ、それはシンプルです。どちらかを必要とする場合、それはコンプレックスです。

たとえば、latin1 と danish はシンプルなキャラクタセットで、big5 や czech はコンプレックスなキャラクタセットです。

以下のセクションでは、キャラクタセットを MYSET という名前にすることが前提です。

シンプルなキャラクタセットの場合、以下を実行します。

1. MYSET を sql/share/charsets/Index ファイルの最後に追加し、これに一意の番号を割り当てる。
2. sql/share/charsets/MYSET.conf ファイルを作成する (sql/share/charsets/latin1.conf をベースとして使用できる)。

   ファイルの構文は、以下のとおり非常にシンプルである。

   - コメントは '#' 文字で始まり、行の最後まで続く。
   - 単語は任意の数のスペースで区切る。
   - キャラクタセットを定義するとき、すべての単語は 16 進数値であることが必要。
   - ctype 配列は、最初の 257 語を占める。to_lower[]、to_upper[]、および sort_order[] 配列はそれぞれその後の 256 語を占める。

   See 項4.7.4.「キャラクタ定義配列」。
3. キャラクタセット名を、configure.in の CHARSETS_AVAILABLE リストおよび COMPILED_CHARSETS リストに追加する。
4. 再コンフィギャして再コンパイルし、テストする。

コンプレックスなキャラクタセットの場合、以下を実行します。

1. MySQL ソースディストリビューションに strings/ctype-MYSET.c ファイルを作成する。
2. MYSET を sql/share/charsets/Index ファイルの最後に追加する。これに一意の番号を割り当てる。
3. strings/ctype-big5.c など、既存の ctype-*.c ファイルの 1 つを見て、何の定義が必要か調べる。注意: ファイル内の配列名は、ctype_MYSET、to_lower_MYSET などであることが必要。これは、シンプルなキャラクタセットの配列に対応する。 See 項4.7.4.「キャラクタ定義配列」。
4. ファイルの先頭付近に、以下のようなコメントを置く。

```
/*
 * 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
 */
```

   configure プログラムはこのコメントを使用して、自動的にキャラクタセットを MySQL ライブラリに組み込む。

   strxfrm_multiply 行および mbmaxlen 行については、後続セクションで説明する。これらはそれぞれ、文字列照合関数またはマルチバイト文字セット関数が必要な場合に組み込む。

5. 次に、以下の関数を作成する。
   - my_strncoll_MYSET()
   - my_strcoll_MYSET()
   - my_strxfrm_MYSET()
   - my_like_range_MYSET()

   See 項4.7.5.「文字列照合サポート」。
6. キャラクタセット名を、configure.in の CHARSETS_AVAILABLE リストおよび COMPILED_CHARSETS リストに追加する。
7. 再コンフィギャして再コンパイルし、テストする。

sql/share/charsets/README ファイルに、より詳細な説明が含まれています。

MySQL ディストリビューションへのキャラクタセットの組み込みを希望する場合には、MySQL internals メーリングリストにパッチをメールしてください。 See 項1.7.1.1.「MySQL メーリングリスト」。

4.7.4. キャラクタ定義配列

to_lower[] と to_upper[] は、キャラクタセットの各メンバに対応する、大文字と小文字が格納されるシンプルな配列です。次に例を示します。

```
to_lower['A'] は 'a' を含む
to_upper['a'] は 'A' を含む
```

sort_order[] は、文字をどのような順番で比較しソートするかを示すマップです。多くの場合（すべてのキャラクタセットについてではありませんが）、これは to_upper[] と同じです（ソートで大文字と小文字が区別されません）。MySQL は、sort_order[character] の値に基づいて文字をソートします。さらに複雑なソートルールに関しては、後述の文字列照合の説明を参照してください。 See 項4.7.5.「文字列照合サポート」。

ctype[] はビット値の配列で、1 文字が 1 要素です。注意: to_lower[]、to_upper[]、および sort_order[] は文字値でインデックス化されますが、ctype[] は文字値 + 1 でインデックス化されます。これは、EOF を処理することが必要だったときの古いレガシです。

m_ctype.h に、以下のビットマスク定義があります。

```
#define _U     01    /* Uppercase */
#define _L     02    /* Lowercase */
#define _N     04    /* Numeral (digit) */
#define _S     010    /* Spacing character */
#define _P     020    /* Punctuation */
#define _C     040    /* Control character */
#define _B     0100    /* Blank */
#define _X     0200    /* heXadecimal digit */
```

各文字の ctype[] エントリは、文字を指定するビットマスク値の組み合わせになっている必要があります。たとえば、'A' は大文字（_U）であると同時に 16 進数（_X）でもあるので、ctype['A'+1] には以下の値が含まれます。

```
_U + _X = 01 + 0200 = 0201
```

4.7.5. 文字列照合サポート

使用する言語のソートルールが、シンプルな sort_order[] テーブルで処理するには複雑すぎる場合、文字列照合を行う必要があります。

現在のところ、これに関する最良のドキュメントは、すでに実装されているキャラクタセットです。たとえば、big5、czech、gbk、sjis、tis160 などのキャラクタセットを見てみてください。

ファイルの先頭の特殊コメントで、strxfrm_multiply_MYSET=N 値を指定する必要があります。N には、my_strxfrm_MYSET で文字列が大きくなれる最大比率（正の整数）を設定してください。

4.7.6. マルチバイト文字サポート

マルチバイト文字を含む新しいキャラクタセットのサポートを追加する場合、マルチバイト文字関数を使用する必要があります。

現在のところ、これに関する最良のドキュメントは、すでに実装されているキャラクタセットです。たとえば、euc_kr、gb2312、gbk、sjis、ujis などのキャラクタセットを参照してください。これらは、strings ディレクトリの ctype-'charset'.c ファイルに実装されています。

ソースファイルの先頭の特殊コメントで mbmaxlen_MYSET=N 値を指定する必要があります。N には、キャラクタセット内の最大文字のバイト数を設定してください。

4.7.7. キャラクタセットに関する問題

使用しているバイナリにコンパイルされていないキャラクタセットを使用すると、いくつかの問題が発生する可能性があります。

- プログラムが認識しているパスに、キャラクタセットが保存されていない (デフォルトは /usr/local/mysql/share/mysql/charsets) 。これは、該当プログラムで --character-sets-dir オプションを使用することによって解決できる。
- キャラクタセットがマルチバイトキャラクタセットであり、動的にロードできない。この場合、そのキャラクタセットをサポートするようにプログラムを再コンパイルする必要がある。
- キャラクタセットが動的なキャラクタセットであるが、その設定ファイルがない。この場合、新しい MySQL ディストリビューションから、そのキャラクタセットの設定ファイルをインストールする必要がある。
- Index ファイルに、キャラクタセットの名前が含まれていない。

```
ERROR 1105: File '/usr/local/share/mysql/charsets/?.conf' not found
(Errcode: 2)
```

この場合、新しい Index ファイルを入手するか、または手動でキャラクタセット名を追加する。

MyISAM テーブルでは、テーブルのキャラクタセットの名前と番号を myisamchk -dvv table_name で確認できます。

4.8. MySQL サーバサイドのスクリプトとユーティリティ

4.8.1. サーバサイドのスクリプトとユーティリティの概要

すべての MySQL プログラムが多くのさまざまなオプションを使用します。しかし、すべての MySQL プログラムに --help オプションが用意されており、このオプションを使用することにより、そのプログラムのすべてのオプションの説明を表示することができます。たとえば、mysql --help を試してみてください。

すべての標準プログラムのデフォルトオプションは、オプション設定ファイルで上書きできます。 項4.1.2.「my.cnf オプション設定ファイル」。

以下の一覧は、サーバサイドの MySQL プログラムを簡単に説明したものです。

- myisamchk

  MySQL テーブルの記述、チェック、最適化、および修復を行うユーティリティ。myisamchk には多くの機能があるため、別の章で説明する。 See 章 4. データベース管理。

- make_binary_distribution

  コンパイルされた MySQL のバイナリリリースを作成する。作成したリリースは、他の MySQL ユーザが利用できるように、FTP 経由で support.mysql.com の /pub/mysql/Incoming に送信することができる。

- mysqlbug

  MySQL バグレポートスクリプト。このスクリプトは、MySQL リストにバグレポートを提出する際、必ず使用する。

- mysqld

  SQL デーモン。常に稼動していることが必要である。

- mysql_install_db

  MySQL 権限テーブルをデフォルト権限で作成する。これは通常、最初に MySQL をシステムにインストールするとき、1 回だけ実行する。

4.8.2. mysqld_safe (mysqld のラッパ)

mysqld_safe は、mysqld デーモンを Unix で起動するときの推奨される方法です。mysqld_safe により、エラー発生時にサーバを再起動したり、ランタイム情報をログファイルに記録するなどのセーフティ機能が加わります。

注意: MySQL 4.0 より前は、mysqld_safe は safe_mysqld という名前でした。下位互換性を維持するため、しばらくの間、MySQL バイナリディストリビューションには mysqld_safe へのシンボリックリンクとして safe_mysqld が含められています。

--mysqld=# または --mysqld-version=# を使用しない場合、mysqld_safe は、mysqld-max という名前の実行可能ファイルがあればこれを使用します。このファイルがなければ、mysqld_safe は mysqld を開始します。これにより、mysqld の代わりに mysqld-max が使用されるかどうかのテストを簡単に実行できます。mysqld-max を mysqld のある場所にコピーするだけで、前者が使用されます。

通常、mysqld_safe スクリプトは編集しないでください。代わりに、my.cnf ファイルの [mysqld_safe] セクションの mysqld_safe にオプションを設定します。mysqld_safe は、オプション設定ファイルの [mysqld]、[server]、[mysqld_safe] の各セクションからすべてのオプションを読み取ります (下位互換性のため、[safe_mysqld] セクションからもオプションが読み取られます)。 See 項4.1.2.「my.cnf オプション設定ファイル」。

注意: mysqld_safe に対するコマンドラインオプションはすべて mysqld に渡されます。mysqld がサポートしていないオプションを mysqld_safe で使用するには、オプション設定ファイルでそのオプションを指定する必要があります。

mysqld_safe のオプションのほとんどは、mysqld のオプションと同じです。 See 項4.1.1.「mysqld コマンドラインオプション」。

mysqld_safe は、以下のオプションをサポートします。

- --basedir=path , --core-file-size=#

  mysqld が作成できるコアファイルのサイズ。ulimit -c に渡される。

- --datadir=path , --defaults-extra-file=path , --defaults-file=path , --err-log=path (4.0 で廃止。代わりに --log-error を使用すること) , --log-error=path

  上記ファイルにエラーログを書き込む。 See 項4.10.1.「エラーログ」。

- --ledir=path

  mysqld のパス。

- --log=path , --mysqld=mysqld-version

ledir ディレクトリ内にある起動する mysqld バージョンの名前。

- --mysqld-version=version

  --mysqld= と同様であるが、ここでは mysqld のサフィックスのみ指定する。 たとえば、--mysqld-version=max を使用すると、mysqld_safe は ledir/mysqld-max バージョンを起動する。--mysqld-version の引数が空白の場合、ledir/mysqld が使用される。

- --nice=#（MySQL 4.0.14 で追加）, --no-defaults , --open-files-limit=#

  mysqld が開くことのできるファイルの数。ulimit -n に渡される。注意: これを正常に機能させるには、mysqld_safe を root として起動する必要がある。

- --pid-file=path , --port=# , --socket=path , --timezone=#

  タイムゾーン（TZ）変数を、このパラメータの値に設定する。

- --user=#

mysqld_safe スクリプトは、MySQL のソースバージョンまたはバイナリバージョンのどちらでインストールされたサーバでも起動できるように記述されています。サーバが多少異なる場所にインストールされていても問題ありません。mysqld_safe は、以下のいずれかの条件が満たされていることを必要とします。

- mysqld_safe を起動するディレクトリとの相対関係でサーバとデータベースを検出できる。mysqld_safe は、その作業ディレクトリ以下で、bin ディレクトリと data ディレクトリ（バイナリディストリビューションの場合）、または libexec ディレクトリと var ディレクトリ（ソースディストリビューションの場合）を探す。MySQL インストールディレクトリ（たとえば、バイナリディストリビューションの場合は /usr/local/mysql ）から mysqld_safe を実行する場合、この条件が満たされていることが必要である。
- サーバおよびデータベースを作業ディレクトリとの相対関係で検出できない場合、mysqld_safe は絶対パスによって検出しようとする。通常、この場所は、/usr/local/libexec や /usr/local/var になる。実際の場所は、ディストリビューションがビルドされたときに決定され、ここから mysqld_safe が起動する。MySQL が標準の場所にインストールされている場合、この絶対パスが正しいことが必要になる。

mysqld_safe はその作業ディレクトリに相対してサーバとデータベースを探すため、MySQL インストールディレクトリから mysqld_safe を開始するのであれば、MySQL のバイナリディストリビューションはどこにでもインストールできます。

```
shell> cd mysql_installation_directory
shell> bin/mysqld_safe &
```

MySQL インストールディレクトリから起動しても mysqld_safe が失敗する場合、システムにとって正しい mysqld のパスとパス名オプションが使用されるように修正します。注意: MySQL をアップグレードすると、mysqld_safe の修正バージョンは上書きされます。再インストールできるように、編集バージョンのコピーを取っておいてください。

4.8.3. mysqld_multi（複数の MySQL サーバを管理するプログラム）

mysqld_multi は、さまざまな Unix ソケットおよび TCP/IP ポートをリッスンする複数の mysqld プロセスを管理するため

のプログラムです。

このプログラムは、[mysqld#] という名前のグループを my.cnf (または --config-file=... オプションで指定されたファイル) から検索します。ここで、# は 1 で始まる任意の正の数です。この番号は、以下の説明ではオプショングループ番号、または GNR と呼びます。グループ番号はオプショングループを識別し、起動したり、停止したり、またはステータス情報を取得するサーバを指定する mysqld_multi の引数として使用します。これらのグループに含まれるオプションは、mysqld の起動に使用される通常の [mysqld] グループと同じです (項2.4.3. 「MySQL を自動的に起動および停止する」 などを参照してください) 。ただし、mysqld_multi では、各グループに、各 mysqld プロセスで使用されるポートやソケットなどを指定するオプションが含まれていることが必要です。

mysqld_multi は、以下の構文で起動します。

```
  mysqld_multi [OPTIONS] {start|stop|report} [GNR,GNR,GNR...]
か mysqld_multi [OPTIONS] {start|stop|report} [GNR-GNR,GNR,GNR-GNR,...]
```

各 GNR は、オプショングループ番号を表します。任意の GNR を開始、停止、または報告でき、複数の GNR を同時に開始、停止、または報告することもできます。オプション設定ファイルの設定例を表示するには、以下のコマンドを実行します。

```
shell> mysqld_multi --example
```

リスト内の GNR の値はコンマで区切ったり、ダッシュ記号で結合できます。結合の場合、GNR1-GNR2 間のすべての GNR が対象になります。GNR 引数を指定しなかった場合、リスト内のすべてのグループが起動、停止、または報告されます。注意: GNR リストには空白スペースを使用できません。空白スペースを挿入すると、スペースの後は無視されます。

mysqld_multi は、以下のオプションをサポートします。

- --config-file=...

  代替設定ファイル。注意: これは、このプログラムのオプション ([mysqld_multi] グループ) には影響せず、[mysqld#] グループにのみ影響する。このオプションがない場合、通常の my.cnf ファイルからすべてが検索される。

- --example

  オプション設定ファイル例を表示する。

- --help

  ヘルプを出力して終了する。

- --log=...

  ログファイル。ログファイルの完全パスと名前。注意: このファイルが存在していれば、すべてがログファイルに記録される。

- --mysqladmin=...

  サーバのシャットダウンに使用する mysqladmin バイナリ。

- --mysqld=...

使用する mysqld バイナリ。注意: このオプションに mysqld_safe を指定することもできる。このオプションは、mysqld に渡される。環境変数 PATH が mysqld を指定していることを確認するか、mysqld_safe を修正する。

- --no-log

  ログをログファイルではなく、stdout に書き込む。デフォルトではログファイルが有効になっている。

- --password=...

  mysqladmin のユーザのパスワード。

- --tcp-ip

  Unix ソケットではなく、TCP/IP ポートで各 MySQL サーバに接続する。これは、サーバの停止と報告に影響する。ソケットファイルがなくてもサーバはまだ実行可能であるが、TCP/IP ポート経由のアクセスに限られる。デフォルトでは、Unix ソケットが使用される。

- --user=...

  mysqladmin の MySQL ユーザ。

- --version

  バージョン番号を出力して終了する。

mysqld_multi に関する補足コメント

- mysqladmin プログラムなどを使用して mysqld サービスを停止する MySQL ユーザのパスワードとユーザ名が、アクセスされるすべてのデータディレクトリ（mysql データベースへのアクセス）用のパスワードとユーザ名と同じであることを確認すること。また、ユーザに SHUTDOWN 権限があることも確認する。多くのデータディレクトリがあり、MySQL root ユーザのパスワードが異なる多くの mysql データベースがある場合、以下のように同じパスワードを使用する共通 multi_admin ユーザを作成できる。以下、その例。

  ```
  shell> mysql -u root -S /tmp/mysql.sock -proot_password -e
  "GRANT SHUTDOWN ON *.* TO multi_admin@localhost IDENTIFIED BY 'multipass'"
  ```

  See 項4.3.6.「権限システムはどのように機能するか」。 各データディレクトリで実行する各 mysqld に対してこのコマンドを実行する必要がある（ソケット -S=... のみ変更する）。

- pid-file は、mysqld_safe を使用して mysqld を起動する場合（たとえば、--mysqld=mysqld_safe）非常に重要である。すべての mysqld に独自の pid-file が必要である。ここで mysqld を直接使用せずに mysqld_safe を使用する利点は、kill -9 で送信されるシグナルや、その他セグメント化エラー（MySQL では発生してはいけないことだが）などによって mysqld プロセスが強制終了した場合でも、mysqld_safe がすべての mysqld プロセスを ``保護" し、再起動することである。注意: mysqld_safe スクリプトは、特定の場所から起動することが必要な場合がある。つまり、mysqld_multi を開始する前に、特定のディレクトリに cd することが必要な場合がある。起動時に問題が発生した場合は、mysqld_safe スクリプトを確認のこと。特に以下の行をチェックする。

  ```
  ----------------------------------------------------------------------
  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 項4.8.2.「mysqld_safe (mysqld のラッパ) 」。

上記のテストがうまくいかなければ、問題が発生する可能性がある。

- 同じデータディレクトリで、複数の mysqld サーバを開始することには危険が伴う。完全に理解していない限り、別々のデータディレクトリを使用すべきである。
- 各 mysqld に対して異なるソケットファイルと TCP/IP ポートが指定されていることが必要である。
- 1 番目と 5 番目の mysqld グループは、例から意図的に除外してある。設定ファイルには '空行' を入れておくことが可能である。これにより、柔軟性が高まる。mysqlds の開始と停止の順序は、設定ファイルでの順序に依存する。
- このプログラムで GNR を使用して特定グループを参照するには、グループ名の最後で番号を指定する。 たとえば、[mysqld17] という名前のグループの GNR は 17 である。
- mysqld で --user オプションを使用するには、mysqld_multi スクリプトを Unix root ユーザとして実行する必要がある。設定ファイルにこのオプションがあっても問題はない。スーパーユーザでないユーザが自分の Unix アカウントで mysqld を開始している場合でも、警告が発生するだけである。重要: 特定の mysqld プロセスを開始しているその Unix ユーザが pid-file およびデータディレクトリの読み取り権限と書き込み権限を (データディレクトリの場合は実行権限も) 持っていることを確認すること。完全に理解していない限り、この目的で Unix root アカウントを使用してはいけない。
- 最重要: mysqld サーバに渡されるオプションの意味、および mysqld プロセスを個別に使用する方が望ましい理由を理解しておくこと。同じデータディレクトリで複数のサーバを起動しても、スレッドシステムではパフォーマンスの向上は望めない。

See 項4.2.「同じマシン上で複数の MySQL サーバを実行する」。

以下、mysqld_multi の設定が含まれる設定ファイルの例です。

```
# This file should probably be in your home dir (~/.my.cnf) or /etc/my.cnf
# Version 2.1 by Jani Tolonen

[mysqld_multi]
mysqld     = /usr/local/bin/mysqld_safe
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 項4.1.2.「my.cnf オプション設定ファイル」。

4.8.4. myisampack（MySQL 圧縮読み取り専用テーブルジェネレータ）

myisampack は MyISAM テーブルの圧縮に、pack_isam は ISAM テーブルの圧縮に使用します。ISAM テーブルは廃止されているため、ここでは myisampack に限って話を進めますが、myisampack について説明することはすべて、pack_isam にも当てはまります。

myisampack は、テーブル内の各カラムを個別に圧縮します。テーブルを開いたとき、カラムを展開するための情報がメモリに読み込まれます。これにより、個々のレコードにアクセスする際のパフォーマンスが向上します。この場合、Stacker を MS-DOS で使用するときのような大きなディスクブロックではなく、1 つのレコードだけを解凍するだけで済みます。 通常、myisampack はデータファイルを 40% 〜 70% 圧縮します。

MySQL は、圧縮テーブルでメモリマッピング（mmap()）を使用し、mmap() が機能しない場合は、通常のファイルの読み取り/書き込み使用法に戻ります。

以下に注意してください。

- パック後、テーブルは読み取り専用になる。これは、CD にパックされたテーブルへのアクセスなど、読み取り専用でいいとの判断からそうしているものである。パックされたテーブルへの書き込み機能の実現も開発計画には入っているが、優先順位は高くない。
- myisampack は、BLOB カラムまたは TEXT カラムもパックできる。 古い pack_isam（ISAM テーブル用）ではパックできない。

myisampack は以下のコマンドで起動します。

```
shell> myisampack [options] filename ...
```

各ファイル名は、インデックス (.MYI) ファイル名になっていることが必要です。データディレクトリがカレントディレクトリでなければ、ファイルのパスを指定してください。.MYI 拡張子は省略可能です。

myisampack は、以下のオプションをサポートします。

- -b, --backup

  テーブルのバックアップを tbl_name.OLD として作成する。

- -#, --debug=debug_options

  デバッグログを出力する。debug_options 文字列には、'd:t:o,filename' がよく使用される。

- -f, --force

  テーブルが大きくなってしまう場合、テンポラリファイルが存在する場合でもテーブルのパックを強制する。myisampack は、テーブルの圧縮中、tbl_name.TMD という名前のテンポラリファイルを生成する。myisampack を強制終了した場合、.TMD ファイルが削除されない場合がある。通常、tbl_name.TMD があれば、myisampack はエラーを出力して終了する。--force を使用すると、テンポラリファイルの有無に関わらず myisampack はテーブルをパックする。

- -?, --help

  ヘルプメッセージを表示して終了する。

- -j big_tbl_name, --join=big_tbl_name

  コマンドラインで指定されたすべてのテーブルを結合して 1 つのテーブル big_tbl_name にする。結合するテーブルはすべて、同一 (同じカラム名、同じ型、同じインデックスなど) のテーブルであることが必要である。

- -p #, --packlength=#

  レコード長保存サイズをバイト単位で指定する。値は 1、2、または 3 であることが必要。myisampack は、すべてのレコードを 1、2、または 3 バイトの長さポインタで保存する。ほとんどの場合、myisampack はファイルをパックする前に、正しい長さを判断できるが、パック中にさらに短くてもよいことを認識する場合がある。この場合、myisampack は、次回同じファイルをパックするときに、レコード長を短くできるというメモを出力する。

- -s, --silent

  サイレントモード。エラー発生時のみ出力する。

- -t, --test

  実際にはテーブルをパックせず、パックテストのみ実行する。

- -T dir_name, --tmp_dir=dir_name

  テンポラリテーブルを書き込むディレクトリを指定する。

- -v, --verbose

  冗長モード。パックの進捗および結果に関する情報を出力する。

- -V, --version

  バージョン情報を表示して終了する。

- -w, --wait

  テーブルが使用中の場合、待機して再試行する。mysqld サーバが --skip-external-locking オプションで起動しいる場合、パッキングプロセス中にテーブルが更新される可能性があれば、myisampack を使用すべきではない。

以下のコマンドシーケンスは、一般的なテーブル圧縮を示しています。

```
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
```

以下、myisampack によって出力される情報について説明します。

- normal

  パックが使用されなかったカラムの数。

- empty-space

  値が空白スペースのみのカラムの数。これらは 1 ビットになる。

- empty-zero

  値がバイナリの 0 のみのカラムの数。これらは 1 ビットになる。

- empty-fill

  整数カラムで、その型の全バイト範囲を占めていないカラムの数。これらは、より小さな型に変換される（たとえば、INTEGER カラムを MEDIUMINT に変更する）。

- pre-space

  数値カラムのうち、値の先頭に空白スペースがあるカラムの数。この場合、各値には先頭スペースのカウントが含まれる。

- end-space

  末尾に空白スペースがあるカラムの数。この場合、各値には末尾スペースのカウントが含まれる。

- table-lookup

  カラムに複数の値が少ししかなく、ハフマン圧縮の前に ENUM に変換された。

- zero

  すべての値がゼロであるカラムの数。

- Original trees

  ハフマンツリーの初期番号。

- After join

領域を節約するためにツリーを結合した後に残った、独立したハフマンツリーの数。

テーブルが圧縮された後で myisamchk -dvv を実行すると、各フィールドについての以下の追加情報が出力されます。

- Type

  フィールド型には以下の記述子が入る。

  - constant

    すべてのレコードが同じ値。

  - no endspace

    エンドスペースを保存しない。

  - no endspace, not_always

    エンドスペースを保存せず、すべての値でエンドスペース圧縮を実行しない。

  - no endspace, no empty

    エンドスペースを保存しない。空白値を保存しない。

  - table-lookup

    カラムが ENUM に変換された。

  - zerofill(n)

    値内の最も重要な n バイトは常に 0 なので、保存しない。

  - no zeros

    ゼロを保存しない。

  - always zero

    0 値を 1 ビットで保存する。

- Huff tree

  フィールドに関連付けられているハフマンツリー。

- Bits

  ハフマンツリーで使用されているビット数。

pack_isam または myisampack を実行後、isamchk または myisamchk を実行してインデックスを再生成する必要があります。このとき、MySQL オプティマイザの効率性を上げるために、インデックスブロックをソートし、統計を作成しておくこともできます。

```
myisamchk -rq --analyze --sort-index table_name.MYI
isamchk   -rq --analyze --sort-index table_name.ISM
```

パックされたテーブルを MySQL データベースディレクトリにインストールした後、mysqladmin flush-tables を実行して、mysqld が新しいテーブルを使用して起動するようにしてください。

パックされたテーブルをアンパックするには、--unpack オプションで isamchk または myisamchk を実行します。

4.8.5. mysqld-max (拡張 mysqld サーバ)

mysqld-max は、以下のオプションでコンフィギャされた MySQL サーバ (mysqld) です。

オプション	コメント
--with-server-suffix=-max	-max サフィックスを mysqld バージョン文字列に追加。
--with-innodb	InnoDB テーブルのサポート (MySQL 3.23 のみ) 。
--with-bdb	Berkeley DB (BDB) テーブルのサポート。
CFLAGS=-DUSE_SYMDIR	Windows でのシンボリックリンクのサポート。

InnoDB サポートを有効化するオプションは、MySQL 3.23 でのみ必要です。MySQL 4 以降では、InnoDB はデフォルトで含まれています。

MySQL-Max バイナリは http://www.mysql.com/downloads/mysql-max-4.0.html にあります。

Windows MySQL バイナリディストリビューションには、標準 mysqld.exe バイナリと mysqld-max.exe バイナリの両方が含まれています。 http://www.mysql.com/downloads/mysql-4.0.html。 See 項2.1.1.「Windows への MySQL のインストール」。

注意: BerkeleyDB (BDB) はすべてのプラットフォームで利用できるわけではなく、BDB をサポートしない Max バイナリもあります。サポートされているテーブル型を確認するには、以下のクエリを実行します。

```
mysql> SHOW VARIABLES LIKE "have_%";
+-----------------+----------+
| Variable_name   | Value    |
+-----------------+----------+
| have_bdb        | NO       |
| have_crypt      | YES      |
| have_innodb     | YES      |
| have_isam       | YES      |
| have_raid       | NO       |
| have_symlink    | DISABLED |
| have_openssl    | NO       |
| have_query_cache | YES     |
+-----------------+----------+
```

2 番目のカラムの値には以下の意味があります。

値	意味
YES	オプションは有効化されており、使用可能である。
NO	MySQL は、このオプションをサポートするようにはコンパイルされていない。

DISABLED	mysqld が --skip-xxxx で起動したか、このオプションを有効化するために必要なオプションが足りない状態で mysqld が起動したため、xxxx オプションが無効になっている。この場合、hostname.err ファイルに、このオプションが無効になっている理由が示される。

注意: InnoDB テーブルを MySQL バージョン 3.23 で作成できるようにするには、少なくとも innodb_data_file_path オプションを含めるようにスタートアップオプションを編集する必要があります。 See 項7.5.2.「MySQL バージョン 3.23 での InnoDB」。

BDB テーブルのパフォーマンスを向上させるには、そのための設定オプションもいくつか追加することが必要です。 See 項7.6.3.「BDB 起動オプション」。

mysqld_safe は自動的に、-max サフィックスの mysqld バイナリを開始しようとします。この機能により、既存のインストールで別の mysqld バイナリを簡単にテストすることができます。必要なオプションで configure を実行し、新規 mysqld バイナリを、古い mysqld バイナリと同じディレクトリに mysqld-max としてインストールします。 See 項4.8.2.「mysqld_safe (mysqld のラッパ) 」。

Linux では、MySQL-Max RPM は上記の mysqld_safe 機能を使用します (これは単に mysqld-max 実行可能ファイルをインストールするだけなので、mysqld_safe の再起動時、mysqld_safe は自動的にこの実行可能ファイルを使用します) 。

以下の表は、MySQL-Max バイナリにどのテーブル型が含まれているかを示したものです。

システム	BDB	InnoDB
Windows/NT	Y	Y
AIX 4.3	N	Y
HP-UX 11.0	N	Y
Linux-Alpha	N	Y
Linux-Intel	Y	Y
Linux-IA-64	N	Y
Solaris-Intel	N	Y
Solaris-SPARC	Y	Y
SCO OSR5	Y	Y
UnixWare	Y	Y
Mac OS X	N	Y

注意: MySQL 4 以降、デフォルトで InnoDB が含まれるようになったので、InnoDB 用の MySQL Max サーバは必要なくなりました。

4.9. MySQL クライアントサイドのスクリプトとユーティリティ

4.9.1. クライアントサイドのスクリプトとユーティリティの概要

mysqlclient ライブラリを使用してサーバと通信する MySQL クライアントはすべて、以下の環境変数を使用します。

名前	説明
MYSQL_UNIX_PORT	デフォルトソケット。localhost との接続に使用。
MYSQL_TCP_PORT	デフォルトの TCP/IP ポート。
MYSQL_PWD	デフォルトのパスワード。
MYSQL_DEBUG	デバッグ時のデバッグトレースオプション。
TMPDIR	テンポラリテーブルまたはテンポラリファイルが作成されるディレクトリ。

MYSQL_PWD を使用することは安全な方法ではありません。 See 項4.3.8.「MySQL サーバへの接続」。

Unix では、mysql クライアントは、MYSQL_HISTFILE 環境変数で指定されたファイルを使用してコマンドライン履歴を保存します。ヒストリファイルのデフォルト値は $HOME/.mysql_history です。ここで、$HOME は HOME 環境変数の値です。 See 付録 F. 環境変数。

クエリ履歴が記録されたファイルを必要としない場合には、まず、.mysql_history が存在する場合はそれを削除し、次に以下のいずれかを実行します。

- MYSQL_HISTFILE 変数を /dev/null に設定する。この設定を口グインごとに有効にするには、この設定をシェルのスタートアップファイルに入力する。
- .mysql_histfile を /dev/null へのシンボリックリンクとして作成する。

  ```
  shell> ln -s /dev/null $HOME/.mysql_history
  ```

  これは、1 度実行するだけでよい。

すべての MySQL プログラムが多くのさまざまなオプションを使用します。しかし、すべての MySQL プログラムに --help オプションが用意されており、このオプションを使用することにより、そのプログラムのすべてのオプションの説明を表示することができます。たとえば、mysql --help を試してみてください。

すべての標準クライアントプログラムのデフォルトオプションは、オプション設定ファイルで上書きできます。 項4.1.2.「my.cnf オプション設定ファイル」。

以下の一覧は、クライアントサイドの MySQL プログラムを簡単に説明したものです。

- msql2mysql

  mSQL プログラムを MySQL に変換するシェルスクリプト。すべてのケースに対応するわけではないが、変換時に最初に試行すべきものである。

- mysql

  対話的にクエリを入力したり、バッチモードでファイルからクエリを実行するためのコマンドラインツール。 See 項4.9.2.「mysql (コマンドラインツール) 」。

- mysqlcc

  このプログラムは、サーバと対話するためのグラフィカルインタフェースを提供する。 See 項4.9.3.「mysqlcc (

MySQL コントロールセンタ）」。

- mysqlaccess

  ホスト、ユーザ、およびデータベースの組み合わせのアクセス権限をチェックするスクリプト。

- mysqladmin

  データベースを作成または破棄したり、権限テーブルを再度読み込んだり、テーブルをディスクにフラッシュしたり、ログファイルを再度開くなどの管理操作を実行するユーティリティ。mysqladmin は、サーバからバージョン情報、プロセス情報、およびステータス情報を取得するためにも使用できる。 See 項4.9.4.「mysqladmin（MySQL サーバの管理）」。

- mysqlbinlog

  バイナリログからクエリを読み取るためのユーティリティ。クラッシュ時、古いバックアップでリカバリするために使用できる。 See 項4.9.5.「mysqlbinlog（バイナリログからクエリを実行する）」。

- mysqldump

  MySQL データベースを SQL ステートメントまたはタブ区切りのテキストファイルとして、ファイルにダンプする。Igor Romanenko 提供の拡張フリーウェア。 See 項4.9.7.「mysqldump（テーブル構造とデータのダンプ）」。

- mysqlimport

  LOAD DATA INFILE を使用して各テーブルにテキストファイルをインポートする。 See 項4.9.9.「mysqlimport（テキストファイルからのデータのインポート）」。

- mysqlshow

  データベース、テーブル、カラム、およびインデックスに関する情報を表示する。

- replace

  msql2mysql によって使用されるユーティリティプログラム。ただし、それ以外にも使用範囲は広い。replace は、ファイルや標準入力内の文字列を変更する。有限ステートマシンを使用して、最初に長い文字列の突き合わせを行う。文字列の入れ替えに使用できる。たとえば以下のコマンドは、指定ファイル内の a と b を入れ替える。

```
shell> replace a b b a -- file1 file2 ...
```

4.9.2. mysql（コマンドラインツール）

mysql はシンプルな SQL シェルです（GNU readline 機能がある）。対話形式と非対話形式をサポートしています。対話形式で使用した場合、クエリ結果は ASCII テーブル形式で表示されます。非対話形式（フィルタなど）で使用した場合、結果はタブ区切り形式で表示されます（出力形式は、コマンドラインオプションで変更できます）。以下のように簡単にスクリプトを実行できます。

```
shell> mysql database < script.sql > output.tab
```

クライアントでメモリ不足による問題が発生した場合、--quick オプションを使用します。これにより、mysql は

mysql_store_result() ではなく、mysql_use_result() を使用して結果セットを取得します。

mysql の使用は非常に簡単です。mysql database または mysql --user=user_name --password=your_password database のように、開始します。SQL ステートメントを入力し、';'、'\g'、または '\G' で終了して Enter キーを押します。

mysql は、以下のオプションをサポートします。

- -?, --help

  ヘルプを表示して終了する。

- -A, --no-auto-rehash

  自動リハッシュを実行しない。テーブルおよびフィールドを完成させるには、'リハッシュ' を実行する必要がある。このオプションにより、mysql の起動が速くなる。

- --prompt=...

  mysql プロンプトを指定の形式に設定する。

- -b, --no-beep

  エラー時のビープ音をオフにする。

- -B, --batch

  結果をタブで区切り、各レコードが 1 行になるように出力する。ヒストリファイルでは使用しないこと。

- --character-sets-dir=...

  キャラクタセットが格納されているディレクトリ。

- -C, --compress

  サーバ/クライアントプロトコルで圧縮を使用する。

- -#, --debug[=...]

  デバッグログ。デフォルトは 'd:t:o,/tmp/mysql.trace'。

- -D, --database=...

  使用するデータベース。このオプションは主に、my.cnf ファイルで使用する。

- --default-character-set=...

  デフォルトのキャラクタセットを設定する。

- -e, --execute=...

  コマンドを実行して終了する（--batch と同じ出力）。

- -E, --vertical

  クエリ（レコード）を縦方向に出力する。このオプションを指定しなくても、ステートメントを \G で終了すれば、同じように出力できる。

- -f, --force

  SQL エラーが発生しても続行する。

- -g, --no-named-commands

  名前付きコマンドが無効になる。* 形式のみ使用する。またはセミコロン（';'）で終わる行の最初でのみ名前付きコマンドを使用する。バージョン 10.9 以降、クライアントは起動時に、デフォルトでこのオプションを有効にするようになっている。ただし、-g オプションでは、ロング形式のコマンドは最初の行から機能する。

- -G, --enable-named-commands

  名前付きコマンドが有効になる。ロング形式のコマンドもショートの * コマンドと同様、有効になる。

- -i, --ignore-space

  関数名の後のスペースを無視する。

- -h, --host=...

  指定のホストに接続する。

- -H, --html

  HTML 出力を生成する。

- -X, --xml

  XML 出力を生成する。

- -L, --skip-line-numbers

  エラーの行番号を書き込まない。これは、エラーメッセージが含まれる結果ファイルを比較する際に使用する。

- --no-pager

ページャーを無効にし、stdout に出力する。対話式ヘルプ (\h) も参照のこと。

- --no-tee

  出力ファイルを無効にする。対話式ヘルプ (\h) も参照のこと。

- -n, --unbuffered

  クエリごとにバッファをフラッシュする。

- -N, --skip-column-names

  結果にカラム名を書き込まない。

- -O, --set-variable=name=value

  変数に値を設定する。--help により、変数が一覧表示される。 注意: --set-variable=name=value および -O name=value 構文は、MySQL 4.0 で廃止されたた。代わりに --name=value を使用すること。

- -o, --one-database

  デフォルトのデータベースだけを更新する。バイナリログ内の他のデータベースを更新しない場合に使用する。

- --pager[=...]

  出力タイプ。デフォルトは ENV 変数の PAGER である。有効なページャーは、less、more、cat [> filename] など。対話式ヘルプ (\h) も参照のこと。このオプションはバッチモードでは無効。ページャーは Unix でのみ動作する。

- -p[password], --password[=...]

  サーバ接続時に使用するパスワード。パスワードをコマンドラインで指定しなかった場合、プロンプトが表示される。注意: ショート形式の -p を使用する場合、オプションとパスワードの間にスペースを入れてはいけない。

- -P port_num, --port=port_num

  接続に使用する TCP/IP ポート番号。

- --protocol=(TCP | SOCKET | PIPE | MEMORY)

  使用する接続プロトコルを指定する。MySQL 4.1 で導入。

- -q, --quick

  結果をキャッシュせず、行ごとに出力する。出力が中断した場合、サーバが遅くなる可能性がある。ヒストリファイルは使用しない。

- -r, --raw

  エスケープ変換なしでカラム値を書き出す。--batch とともに使用。

- --reconnect

  接続が失われた場合、サーバへの再接続を自動的に 1 度だけ試行する。

- -s, --silent

  サイレントモード。

- -S --socket=...

  接続に使用するソケットファイル。

- -t --table

  表形式で出力。これは、非バッチモードでのデフォルトである。

- -T, --debug-info

  終了時にデバッグ情報を出力する。

- --tee=...

  出力ファイルにすべて出力する。対話式ヘルプ (\h) も参照のこと。バッチモードでは無効。

- -u, --user=#

  カレントユーザでない場合のログインユーザ。

- -U, --safe-updates[=#], --i-am-a-dummy[=#]

  キーを使用する UPDATE と DELETE のみ許可する。このオプションについては、詳細を後で説明する。このオプションが my.cnf ファイルに含まれている場合、--safe-updates=0 でリセットできる。

- -v, --verbose

  冗長出力 (-v -v -v で表出力形式となる) 。

- -V, --version

  バージョン情報を出力して終了する。

- -w, --wait

  接続が切断された場合、停止せずに待機して再試行する。

-O または --set-variable には、以下の変数も設定できます。注意: --set-variable=name=value および -O name=value 構文は、MySQL 4.0 で廃止されましたた。代わりに --name=value を使用してください。

変数名	デフォルト	説明
connect_timeout	0	接続タイムアウトの秒数。
local-infile	0	LOAD DATA INFILE の LOCAL 機能の無効化（0）または有効化（1）。
max_allowed_packet	16777216	サーバ間での送受信可能な最大パケット長。
net_buffer_length	16384	TCP/IP およびソケット接続用バッファ。
select_limit	1000	--safe-updates 使用時の SELECT の自動制限。
max_join_size	1000000	--safe-updates 使用時に 1 回の結合で扱うレコードの自動制限。

mysql クライアントがサーバにクエリを送信しているときに接続が切断された場合、クライアントは自動的に再接続を試行し、クエリを再度送信します。注意: 再接続に成功した場合でも、最初の接続が終了した時点で、前回のセッションオブジェクト（テンポラリテーブル、ユーザ変数、セッション変数）はすべて失われています。したがって、上記の動作にはリスクが伴います。たとえば、以下の例の場合、ユーザが気付かない間にサーバがシャットダウンし、再起動しています。

```
mysql> set @a=1;
Query OK, 0 rows affected (0.05 sec)

mysql> insert into t values(@a);
ERROR 2006: MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    1
Current database: test

Query OK, 1 row affected (1.30 sec)

mysql> select * from t;
+------+
| a    |
+------+
| NULL |
+------+
1 row in set (0.05 sec)
```

この場合、@a ユーザ変数は接続の切断と同時に失われ、再接続後は未定義になっています。このリスクを回避するには、mysql クライアントを --disable-reconnect オプションで開始します。

コマンドラインで 'help' を入力すると、mysql がサポートするコマンドを出力できます。

```
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.
delimiter (\d)    Set query delimiter.
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.
prompt    (\R)    Change your mysql prompt.
quit      (\q)    Quit mysql.
rehash    (\#)    Rebuild completion hash.
source    (\.)    Execute an SQL script file.
                  Takes a file name as an argument.
status    (\s)    Get status information from the server.
system    (\!)    Execute a system shell command.
tee       (\T)    Set outfile [to_outfile].
                  Append everything into given outfile.
use       (\u)    Use another database.
                  Takes database name as argument.
```

edit、nopager、pager、system の各コマンドは Unix でのみ動作します。

status コマンドを実行すると、接続情報と、使用しているサーバの情報を取得できます。--safe-updates モードで status を実行すると、クエリに影響する mysql 変数値も出力されます。

初心者に便利なスタートアップオプションとして、--safe-updates（MySQL バージョン 3.23.11 で導入）があります（または、どこかで DELETE FROM table_name を実行しているが WHERE 節を忘れたユーザの場合は --i-am-a-dummy）。このオプションを使用すると、接続時、mysql は以下のコマンドを MySQL サーバに送信します。

```
SET SQL_SAFE_UPDATES=1,SQL_SELECT_LIMIT=#select_limit#,
  SQL_MAX_JOIN_SIZE=#max_join_size#"
```

ここで #select_limit# と #max_join_size# は、mysql コマンドラインから設定できる変数です。 See 項5.5.6.「SET 構文」。

効果は以下のとおりです。

- WHERE 部分にキー制約がなければ、UPDATE ステートメントまたは DELETE ステートメントを実行できない。ただし、LIMIT を使用すれば UPDATE/DELETE を強制実行できる。

  ```
  UPDATE table_name SET not_key_column=# WHERE not_key_column=# LIMIT 1;
  ```

- 大きな結果はすべて、自動的に #select_limit# レコードの制限を受ける。
- #max_join_size 以上のレコードをチェックする必要がありそうな SELECT ステートメントは停止する。

mysql クライアントに関するヒント

データによっては、横方向ではなく、縦方向に表示したほうが見やすい場合があります。たとえば改行を含む長いテキストでは、縦方向に出力した方が読みやすくなります。

```
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>
      sbj: 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)
```

ログには、tee オプションを使用できます。tee は --tee=... オプションで起動できます。または、tee コマンドで対話式にコマンドラインから起動することもできます。画面に表示されるデータはすべて、指定のファイルにも記録されます。これは、デバッグ目的にも非常に役立ちます。tee は、notee でコマンドラインから無効にできます。tee を再び実行すると、ログが再開されます。パラメータなしの場合、前回のファイルが使用されます。注意: tee は、コマンドごとに (次のコマンドを待つコマンドラインが表示される直前に) 、結果をファイルにフラッシュします。

Unix の less、more、その他同様のプログラムの対話式モードで結果を参照したり検索するには、--pager[=...] オプションを使用します。引数がない場合、mysql クライアントは PAGER 環境変数を探して、pager をその値に設定します。 pager は、pager コマンドで対話式コマンドラインから開始でき、nopager コマンドで無効にできます。コマンドは任意の引数を取り、pager がその値に設定されます。pager コマンドは引数なしでも呼び出せますが、--pager オプションが使用されていることが必要になります。そうしなければ、pager はデフォルトで stdout になります。pager は、Windows にはない popen() 関数を使用するため、Unix でのみ有効です。Windows では代わりに tee オプションを使用できます。ただし状況によっては、pager ほど便利ではありません。

pager に関するヒント

- ファイルへの書き込みに使用できる。

  ```
  mysql> pager cat > /tmp/log.txt
  ```

  これで、結果がファイルにのみ出力される。また、使用したいプログラムの任意のオプションを pager で渡すことができる。

  ```
  mysql> pager less -n -i -S
  ```

- 上記例の -S オプションに注目。これは、結果を参照する際に非常に役立つ。このオプションを横方向の表示 (\g または ';' でコマンドを終了) 、および縦方向の表示 (\G でコマンドを終了) で試してみるとわかる。非常に幅の広い結果セットは画面上で見にくい場合があり、-S オプションを less に設定することにより、対話式の less で、画面幅より長い行が次の行まで続くことなく、結果セットを左から右に表示することができる。この方法により、結果セットを非常に見やすくできる。対話式の less で、-S によりこのモードのオンオフを切り替えることができる。less に関する詳細については 'h' を参照のこと。
- 結果の処理には、非常に複雑な方法を組み合わせることができる。以下の例では、結果を、/dr1 と /dr2 にマウントされている 2 つの異なるハードディスク上の 2 つの異なるディレクトリにある 2 つのファイルに記録し、画面上では less で表示する。

```
mysql> pager cat | tee /dr1/tmp/res.txt | \
tee /dr2/tmp/res2.txt | less -n -i -S
```

上記の 2 つの関数を組み合わせることも可能です。tee を有効にし、pager を 'less' に設定することにより、Unix 'less' で結果を参照しながら同時に 1 つのファイルにすべてを記録することができます。pager で使用される Unix の tee と mysql クライアントの組み込み tee がありますが、組み込み tee は tee が利用できない場合でも使用できます。また、組み込み tee は画面で出力されたものすべてをログファイルに記録しますが、pager で使用される Unix tee はそれほど多くのことを記録しません。最後に、対話式 tee の方がオン/オフを切り替えやすいという点があります。ファイルに何かを記録したいのだが、ときどき機能をオフにもしたいという場合に便利です。

MySQL バージョン 4.0.2 から、mysql コマンドラインクライアントでプロンプトを変更できるようになりました。

以下のプロンプトオプションを使用できます。

オプション	説明
\v	mysqld バージョン
\d	使用中のデータベース
\h	接続ホスト
\p	接続ポート
\u	ユーザ名
\U	完全ユーザ名@ホスト
\\	'\'
\n	新規改行
\t	タブ
\	スペース
_	スペース
\R	24 時間形式 (0 〜 23)
\r	標準時間形式 (1 〜 12)
\m	分
\y	2 桁年
\Y	4 桁年

\D	完全日付形式
\s	秒
\w	3 文字形式での曜日（Mon、Tue など）
\P	am/pm
\o	数字による月
\O	3 文字形式での月（Jan、Feb など）
\c	コマンド実行ごとにカウントするカウンタ

'\' にそれ以外の文字が続くと、単にその文字になります。

プロンプトは以下の場所で設定できます。

- 環境変数

  MYSQL_PS1 環境変数をプロンプト文字列に設定できる。次に例を示す。

```
shell> export MYSQL_PS1="(\u@\h) [\d]> "
```

- my.cnf , .my.cnf

  MySQL 設定ファイルの mysql グループで prompt オプションを設定できる。次に例を示す。

```
[mysql]
prompt=(\u@\h) [\d]>\_
```

- コマンドライン

  mysql のコマンドラインで --prompt オプションを設定できる。 次に例を示す。

```
shell> mysql --prompt="(\u@\h) [\d]> "

(user@host) [database]>
```

- 対話式

  prompt（または \R）を使用して対話式にプロンプトを変更することもできる。次に例を示す。

```
mysql> prompt (\u@\h) [\d]>\_
PROMPT set to '(\u@\h) [\d]>\_'
(user@host) [database]>
(user@host) [database]> prompt
Returning to default PROMPT of mysql>
mysql>
```

4.9.3. mysqlcc（MySQL コントロールセンタ）

mysqlcc（MySQL コントロールセンタ）はプラットフォームに依存しないクライアントで、MySQL データベースサーバへのグラフィカルユーザインタフェース（GUI）を提供します。対話使用をサポートし、構文強調やタブ完成の機能も含まれます。データベース、テーブル、およびサーバの管理もできます。

現在、mysqlcc は Windows プラットフォームと Linux プラットフォームで動作します。

mysqlcc は MySQL ディストリビューションには含まれていませんが、http://www.mysql.com/downloads/ からダウンロードできます。

mysqlcc は、以下のオプションをサポートします。

- -?, --help

  ヘルプを表示して終了する。

- -b, --blocking_queries

  クエリのブロックを使用する。

- -C, --compress

  サーバ/クライアントプロトコルで圧縮を使用する。

- -c, --connection_name=name

  --server のシノニム。

- -d, --database=...

  使用するデータベース。これは主に、my.cnf ファイルで使用。

- -H, --history_size=#

  クエリウィンドウのヒストリサイズ。

- -h, --host=...

  指定のホストに接続する。

- -p[password], --password[=...]

  サーバ接続時に使用するパスワード。パスワードをコマンドラインで指定しなかった場合、プロンプトが表示される。注意: ショート形式の -p を使用する場合、オプションとパスワードの間にスペースを入れてはいけない。

- -g, --plugins_path=name

MySQL コントロールセンタのプラグインが置かれているディレクトリのパス。

- -P port_num, --port=port_num

  接続に使用する TCP/IP ポート番号。

- -q, --query

  起動時にクエリウィンドウを開く。

- -r, --register

  起動時に 'Register Server' ダイアログを開く。

- -s, --server=name

  MySQL コントロールセンタの接続名。

- -S --socket=...

  接続に使用するソケットファイル。

- -y, --syntax

  構文強調および完成を有効化。

- -Y, --syntax_file=name

  完成する構文ファイル。

- -T, --translations_path=name

  MySQL コントロールセンタ翻訳の置かれているディレクトリのパス。

- -u, --user=#

  カレントユーザでない場合のログインユーザ。

- -V, --version

  バージョン情報を出力して終了する。

-O または --set-variable には以下の変数も設定できます。 注意: --set-variable=name=value および -O name=value 構文は、MySQL 4.0 で廃止されました。代わりに --name=value を使用してください。

変数名	デフォル	説明

	ト	
connect_timeout	0	接続タイムアウトまでの秒数。
local-infile	0	LOAD DATA INFILE の LOCAL 機能の無効化（0）または有効化（1）。
max_allowed_packet	16777216	サーバ間での送受信可能な最大パケット長。
net_buffer_length	16384	TCP/IP およびソケット接続用バッファ。
select_limit	1000	--safe-updates 使用時の SELECT の自動制限。
max_join_size	1000000	--safe-updates 使用時に 1 回の結合で扱うレコードの自動制限。

4.9.4. mysqladmin（MySQL サーバの管理）

管理操作を実行するためのユーティリティです。構文は以下のとおりです。

```
shell> mysqladmin [OPTIONS] command [command-option] command ...
```

使用しているバージョンの mysqladmin がサポートするオプション一覧を表示するには、mysqladmin --help を実行します。

現在の mysqladmin は以下のコマンドをサポートしています。

- create databasename

  新しいデータベースを作成する。

- drop databasename

  データベースとそのすべてのテーブルを削除する。

- extended-status

  サーバから拡張ステータスメッセージを取得する。

- flush-hosts

  キャッシュされたすべてのホストをフラッシュする。

- flush-logs

  すべてのログをフラッシュする。

- flush-tables

  すべてのテーブルをフラッシュする。

- flush-privileges

  権限テーブルを再読み込みする（reload と同じ）。

- kill id,id,...

mysql スレッドを強制終了する。

- password

  新規パスワードを設定する。旧パスワードを新規パスワードに変更する。

- ping

  mysqld が稼動中かどうかを確認する。

- processlist

  サーバでアクティブなスレッドを一覧表示する。SHOW PROCESSLIST ステートメントと同じ。--verbose オプションが指定されている場合、SHOW FULL PROCESSLIST の出力に似た結果が出力される。

- reload

  権限テーブルを再度読み込む。

- refresh

  すべてのテーブルをフラッシュし、ログファイルを閉じてまた開く。

- shutdown

  サーバをシャットダウンする。

- slave-start

  スレーブレプリケーションスレッドを開始する。

- slave-stop

  スレーブレプリケーションスレッドを停止する。

- status

  サーバから短いステータスメッセージを取得する。

- variables

  利用可能な変数を出力する。

- version

  サーバからバージョン情報を取得する。

コマンドはすべて、それぞれ独自のプレフィックスで短縮可能です。次に例を示します。

```
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  Open tables: 2
Memory in use: 1092K  Max memory used: 1116K
```

mysqladmin status コマンドの結果には、以下のカラムが含まれます。

カラム	説明
Uptime	MySQL サーバが稼動している秒数。
Threads	アクティブなスレッド（クライアント）の数。
Questions	mysqld が起動してからクライアントから発行された質問の数。
Slow queries	long_query_time 秒より時間がかかったクエリ。 See 項4.10.5.「スロークエリログ」。
Opens	mysqld が開いたテーブルの数。
Flush tables	flush ...、refresh、reload の各コマンドの回数。
Open tables	現在開いているテーブルの数。
Memory in use	mysqld コードによって直接割り当てられているメモリ（MySQL が --with-debug=full でコンパイルされている場合にのみ利用可能）。
Max memory used	mysqld コードによって直接割り当てられている最大メモリ（MySQL が --with-debug=full でコンパイルされている場合にのみ利用可能）。

ソケットで（つまり mysqld が実行しているコンピュータで）mysqladmin shutdown を実行すると、mysqld の正常終了を確認するため、mysqladmin は MySQL pid-file が削除されるのを待ちます。

4.9.5. mysqlbinlog（バイナリログからクエリを実行する）

mysqlbinlog ユーティリティを使用して、バイナリログファイル（see 項4.10.4.「バイナリログ」）を調べることができます。

```
shell> mysqlbinlog hostname-bin.001
```

上記の例では、バイナリログ hostname-bin.001 に含まれるすべてのクエリが、クエリにかかった時間、クエリを発行したスレッドの ID、クエリ発行時のタイムスタンプなどの情報とともに出力されます。

mysqlbinlog の出力を mysql クライアントに送信できます。これは、古いバックアップでリカバリするときに使用します（see 項4.5.1.「データベースのバックアップ」）。

```
shell> mysqlbinlog hostname-bin.001 | mysql
```

または

```
shell> mysqlbinlog hostname-bin.[0-9]* | mysql
```

また、mysqlbinlog の出力をテキストファイルにリダイレクトし、そのテキストファイルを編集（何らかの理由で実行したくないクエリを削除）し、テキストファイルから mysql に対してクエリを実行することもできます。

mysqlbinlog には position=# オプションがあり、バイナリログでのオフセットが # 以上のクエリだけを出力することがで

きます。

MySQL サーバで実行するバイナリログが複数ある場合、単一の MySQL 接続で実行する方が安全です。以下は、安全でない例です。

```
shell> mysqlbinlog hostname-bin.001 | mysql # DANGER!!
shell> mysqlbinlog hostname-bin.002 | mysql # DANGER!!
```

最初のバイナリログに CREATE TEMPORARY TABLE が含まれ、2 番目のバイナリログにこのテンポラリテーブルを使用するクエリが含まれていると、問題が発生します。最初の mysql が終了すると、テンポラリテーブルが破棄されるので、2 番目の mysql は ``unknown table'' を報告することになります。これが、すべてのバイナリログを単一の接続で実行すべき理由です。テンポラリテーブルを使用する場合には特に気を付けてください。2 つの方法があります。

```
shell> mysqlbinlog hostname-bin.001 hostname-bin.002 | mysql
```

```
shell> mysqlbinlog hostname-bin.001 >  /tmp/queries.sql
shell> mysqlbinlog hostname-bin.002 >> /tmp/queries.sql
shell> mysql -e "source /tmp/queries.sql"
```

MySQL 4.0.14 以降、mysqlbinlog は、バイナリログから LOAD DATA INFILE を実行するための mysql に対する適切な入力を提供できるようになっています。ロードするデータがバイナリログに含まれるため（これは MySQL 4.0 の場合です。MySQL 3.23 ではロードされたデータがバイナリログに書き込まれないため、バイナリログの内容を実行するにはオリジナルファイルが必要でした）、mysqlbinlog はこのデータをテンポラリファイルにコピーして LOAD DATA INFILE コマンドを出力し、mysql にこのテンポラリファイルをロードさせます。テンポラリファイルが作成されるデフォルトのディレクトリはテンポラリディレクトリです。これは、mysqlbinlog の local-load オプションで変更できます。

MySQL 4.1 より前は、同じ名前のテンポラリテーブルを使用する複数の異なるスレッドのクエリがバイナリログに含まれていました。そのため、これらのクエリが混ざっている場合、mysqlbinlog は mysql に対して適切な出力を提供できませんでした。これは、MySQL 4.1 で解決されています。

また、mysqlbinlog --read-from-remote-server を使用して、リモート MySQL サーバから直接バイナリログを読み取ることも可能です。ただし、より簡単に、稼動中の MySQL サーバにバイナリログを適用できるようにするため、これはいずれ廃止する予定です。

詳細については、mysqlbinlog --help を実行してください。

4.9.6. mysqlcheck を使用したテーブルの保守とクラッシュのリカバリ

MySQL バージョン 3.23.38 以降、MyISAM テーブルのチェックおよび修復のために新しいツールを利用できるようになっています。myisamchk との違いは、myisamchk が、サーバが実行していないときに使用するのに対し、mysqlcheck は mysqld サーバが実行しているときに使用します。テーブルのチェックまたは修復のたびにサーバを停止する必要がなくなりました。

mysqlcheck は、MySQL サーバコマンド CHECK、REPAIR、ANALYZE、および OPTIMIZE を利用して、便利なテーブルの保守およびクラッシュからのリカバリ方法を提供します。

mysqlcheck を開始する方法は 3 つあります。

```
shell> mysqlcheck [OPTIONS] database [tables]
shell> mysqlcheck [OPTIONS] --databases DB1 [DB2 DB3...]
shell> mysqlcheck [OPTIONS] --all-databases
```

データベースおよびテーブルの選択に関しては、mysqldump と似た方法になります。

mysqlcheck には、他のクライアントにはない特殊機能があります。デフォルトの動作であるテーブルチェック (-c) を、バイナリの名前を変更することによって変更できます。デフォルトでテーブルを修復するツールが必要であれば、mysqlcheck を mysqlrepair という新しい名前でハードドライブにコピーするか、mysqlrepair へのシンボリックリンクを作成し、そのシンボリックリンクの名前を mysqlrepair にします。mysqlrepair を起動すると、デフォルトでテーブルが修復されます。

mysqlcheck のデフォルト動作を変更するために使用できる名前は以下のとおりです。

```
mysqlrepair:   The default option will be -r
mysqlanalyze:  The default option will be -a
mysqloptimize: The default option will be -o
```

以下、mysqlcheck で使用可能なオプションの一覧です。使用しているバージョンでのサポート状態を確認するには、mysqlcheck --help を実行してください。

- -A, --all-databases

  すべてのデータベースをチェックする。これは、すべてのデータベースを選択した状態で --databases を実行するのと同じである。

- -1, --all-in-1

  テーブルごとにクエリを 1 つずつ作成するのではなく、データベースごとにまとめてすべてのクエリを実行する。テーブル名はカンマで区切る。

- -a, --analyze

  指定したテーブルを分析する。

- --auto-repair

  チェックしたテーブルが破損していた場合、自動的に修復する。破損したテーブルがあった場合、すべてのテーブルのチェックが終わってから修復が行われる。

- -#, --debug=...

  デバッグログを出力する。これは、'd:t:o,filename' であることが多い。

- --character-sets-dir=...

  キャラクタセットが格納されているディレクトリ。

- -c, --check

  テーブルのエラーをチェックする。

- -C, --check-only-changed

  前回のチェック以降に変更されたテーブルと、正しく閉じられなかったテーブルだけをチェックする。

- --compress

  サーバ/クライアントプロトコルで圧縮を使用する。

- -?, --help

  ヘルプメッセージを表示して終了する。

- -B, --databases

  いくつかのデータベースをチェックする。注意: ここでは、テーブルを指定しない。名前の引数はすべて、データベース名として扱われる。

- --default-character-set=...

  デフォルトのキャラクタセットを設定する。

- -F, --fast

  正しく閉じられなかったテーブルだけをチェックする。

- -f, --force

  SQL エラーが発生しても続行する。

- -e, --extended

  このオプションを CHECK TABLE とともに使用すれば、テーブルの整合性が 100 パーセント保証されるが、時間がかかる。

  このオプションを REPAIR TABLE とともに使用した場合、テーブルの拡張修復が行われる。ただし、時間がかかるだけでなく、無駄なレコードも多く生成される可能性がある。

- -h, --host=...

  ホストに接続する。

- -m, --medium-check

  extended-check よりも速いが、全エラーの 99.99% しか見つけられない。しかし、ほとんどの場合これで十分である。

- -o, --optimize

  テーブルを最適化する。

- -p, --password[=...]

  サーバ接続時に使用するパスワード。パスワードを指定しなければ、プロンプトが表示される。

- -P, --port=...

  TCP/IP 接続に使用するポート番号。

- --protocol=(TCP | SOCKET | PIPE | MEMORY)

使用する接続プロトコルを指定する。MySQL 4.1 で導入。

- -q, --quick

  このオプションを CHECK TABLE とともに使用すれば、不正リンクをチェックするレコードスキャンが行われない。これが最速のチェックである。

  このオプションを REPAIR TABLE とともに使用すると、インデックスツリーの修復だけが実行される。これが、最速のテーブルの修復である。

- -r, --repair

  ほとんどのエラーを修復できる、ただし、ユニークキーが一意でないエラーには対応できない。

- -s, --silent

  エラーメッセージだけを出力する。

- -S, --socket=...

  接続に使用するソケットファイル。

- --tables

  --databases (-B) オプションを上書きする。このオプションの後ろの引数はすべて、テーブル名と見なされる。

- -u, --user=#

  カレントユーザでない場合のログインユーザ。

- -v, --verbose

  さまざまな段階の情報を出力する。

- -V, --version

  バージョン情報を出力して終了する。

4.9.7. mysqldump (テーブル構造とデータのダンプ)

バックアップ用のデータベースまたはデータベースの集合をダンプしたり、他の SQL サーバ (MySQL サーバである必要はない) にデータを移動するためのユーティリティです。ダンプには、テーブル作成や入力のための SQL ステートメントが含まれます。

同じサーバ上でバックアップを行う場合には、mysqlhotcopy の方の使用を考慮してください。 See 項4.9.8. 「mysqlhotcopy (MySQL のデータベースとテーブルのコピー) 」。

```
  mysqldump [OPTIONS] database [tables]
か mysqldump [OPTIONS] --databases [OPTIONS] DB1 [DB2 DB3...]
か mysqldump [OPTIONS] --all-databases [OPTIONS]
```

テーブルを指定しなかったり、--databases オプションまたは --all-databases オプションを使用すると、データベース全

体がダンプされます。

使用しているバージョンの mysqldump がサポートするオプションの一覧を照会するには、mysqldump --help を実行します。

注意: mysqldump を --quick または --opt なしで実行すると、mysqldump は結果をダンプする前に、結果セット全体をメモリにロードします。これは、大きなデータベースをダンプする際、問題になる可能性があります。

注意: mysqldump プログラムの新しいコピーを使用している場合で、非常に古い MySQL サーバに読み込むダンプを行うときには、--opt オプションまたは -e オプションは使用しないでください。

mysqldump は、以下のオプションをサポートします。

- --add-locks

  各テーブルダンプの前に LOCK TABLES を追加し、後に UNLOCK TABLE を追加する（MySQL への挿入を速くするため）。

- --add-drop-table

  各作成ステートメントの前に drop table を追加する。

- -A, --all-databases

  すべてのデータベースをダンプする。これは、すべてのデータベースを選択した状態で --databases を実行するのと同じである。

- -a, --all

  MySQL 固有の作成オプションをすべて含める。

- --allow-keywords

  キーワードであるカラム名の作成を認める。各カラム名の先頭にテーブル名を付け加えることが必要。

- -c, --complete-insert

  完全な挿入ステートメント（カラム名も指定）を使用する。

- -C, --compress

  クライアントとサーバの両方が圧縮をサポートする場合、クライアントとサーバ間の情報をすべて圧縮する。

- -B, --databases

  いくつかのデータベースをダンプする。注意:ここでは、テーブル名を指定しない。名前の引数はすべて、データベース名として扱われる。 出力される各新規データベースの前に USE db_name; が追加される。

- --delayed

  INSERT DELAYED コマンドでレコードを挿入する。

- -e, --extended-insert

新しい複数行 INSERT 構文を使用する（さらにコンパクトで速い挿入ステートメントを提供）。

- -#, --debug[=option_string]

  プログラムのトレース使用（デバック目的）。

- --help

  ヘルプメッセージを表示して終了する。

- --fields-terminated-by=... , --fields-enclosed-by=... , --fields-optionally-enclosed-by=... , --fields-escaped-by=... , --lines-terminated-by=...

  これらのオプションは -T オプションとともに使用する。LOAD DATA INFILE の対応する節と同じ意味を持つ。 See 項 6.4.8.「LOAD DATA INFILE 構文」。

- -F, --flush-logs

  ダンプを開始する前に、MySQL サーバ内のログファイルをフラッシュする。注意: このオプションを --all-databases（または -A）オプションと組み合わせて使用した場合、ログは各データベースのダンプごとにフラッシュされる。

- -f, --force,

  テーブルダンプ中にSQL エラーが発生しても続行する。

- -h, --host=..

  指定したホストの MySQL サーバからデータをダンプする。デフォルトのホストは localhost。

- -l, --lock-tables.

  ダンプを開始する前にすべてのテーブルをロックする。テーブルは READ LOCAL でロックされ、MyISAM テーブルの場合は同時挿入が可能になる。

  注意: 複数のデータベースをダンプする場合、--lock-tables は各データベースを個別にロックする。したがって、このオプションを使用した場合、データベース間でのテーブルの論理整合性は保証されない。 異なるデータベースのテーブルは、完全に異なる状態でダンプされる可能性がある。

- -K, --disable-keys

  /*!40000 ALTER TABLE tb_name DISABLE KEYS */; および /*!40000 ALTER TABLE tb_name ENABLE KEYS */; が出力に含まれる。これにより、インデックスが、すべてのデータが挿入それた後に作成されるため、MySQL 4.0 サーバへのデータのロードが速くなる。

- -n, --no-create-db

  CREATE DATABASE /*!32312 IF NOT EXISTS*/ db_name; が出力に含まれない。--databases オプションまたは --all-databases オプションを指定した場合は、上記の行が追加される。

- -t, --no-create-info

  テーブル作成情報（CREATE TABLE ステートメント）を書き込まない。

- -d, --no-data

  テーブルのレコード情報を一切書き込まない。テーブルの構造だけをダンプする場合、非常に便利である。

- --opt

  --quick --add-drop-table --add-locks --extended-insert --lock-tables と同じ。MySQL サーバに読み込むための最速ダンプを提供する。

- -pyour_pass, --password[=your_pass]

  サーバ接続時に使用するパスワード。'=your_pass' 部分を指定しなければ、mysqldump によってパスワードのプロンプトが表示される。

- -P, --port=...

  TCP/IP 接続に使用するポート番号。

- --protocol=(TCP | SOCKET | PIPE | MEMORY)

  使用する接続プロトコルを指定する。MySQL 4.1 で導入。

- -q, --quick

  クエリをバッファせず、stdou に直接ダンプする。これを行うには、mysql_use_result() を使用する。大きなダンプの際に特に便利である。

- -Q, --quote-names

  テーブル名およびカラム名を ‘`’ 文字で囲む。

- -r, --result-file=...

  指定したファイルへの直接出力。このオプションは MSDOS で使用する。改行 '\n' が '\n\r' (改行 + 復帰) に変換されるのを防ぐためである。

- --single-transaction

  このオプションは、サーバからデータをダンプする前に、BEGIN SQL コマンドを発行する。これは、InnoDB テーブルと READ_COMMITTED トランザクション分離レベルで特に役立つ。このモードでは、どのアプリケーションもブロックすることなく、BEGIN が発行されたときのデータベースの整合した状態をダンプできる。

  このオプションを使用する際は、トランザクションテーブルだけが整合状態でダンプされることに注意する。たとえば、このオプションでダンプされた MyISAM テーブルまたは HEAP テーブルは、変更されている可能性がある。

  --single-transaction オプションはバージョン 4.0.2 で追加された。 このオプションは --lock-tables オプションとは相互排他的である。LOCK TABLES は、前のトランザクションをすでにコミットしているためである。

- -S /path/to/socket, --socket=/path/to/socket

  localhost (デフォルトホスト) との接続に使用するソケットファイル。

- --tables

--databases (-B) オプションを上書きする。

- -T, --tab=path-to-some-directory

  各テーブルに対する SQL CREATE コマンドが含まれる table_name.sql ファイル、および各テーブルに対するデータが含まれる table_name.txt ファイルを作成する。.txt ファイルの形式は、--fields-xxx オプションおよび --lines--xxx オプションに基づく。注意: このオプションは、mysqldump が mysqld デーモンと同じマシンで実行している場合のみ有効。FILE 権限のある MySQL アカウントを使用することが必要。また、mysqld を実行しているログインユーザまたはグループ (通常はユーザ mysql、グループ mysql) に、指定した場所でのファイルの作成および書き込み権限が必要。

- -u user_name, --user=user_name

  サーバとの接続に使用する MySQL ユーザ名。デフォルト値はユーザの Unix ログイン名。

- -O name=value, --set-variable=name=value

  変数の値を指定する。指定可能な変数は以下に示す。注意: --set-variable=name=value および -O name=value 構文は、MySQL 4.0 で廃止。代わりに --name=value を使用すること。

- -v, --verbose

  冗長モード。プログラムの実行内容に関する詳細情報を出力する。

- -V, --version

  バージョン情報を出力して終了する。

- -w, --where='where-condition'

  選択したレコードだけをダンプする。注意: 必ず引用符で囲むこと。

  ```
  "--where=user='jimf'" "-wuserid>1" "-wuserid<1"
  ```

- -X, --xml

  1 つのデータベースを整形式の XML としてダンプ。

- -x, --first-slave

  すべてのデータベースのすべてのテーブルをロック。

- --master-data

  --first-slave とほぼ同じだが、CHANGE MASTER TO コマンドを出力する。マスタのこの SQL ダンプを使用してスレーブをセットアップしていた場合、このコマンドにより、後でマスタのバイナリログ内の正しい位置からスレーブを開始させることができる。

- -O net_buffer_length=#, ここで # < 16M

  複数レコード挿入ステートメントを作成したときに (--extended-insert オプションまたは --opt オプションと同様)、mysqldump は net_buffer_length までの長さのレコードを作成する。この変数を大きくする場合には、MySQL サーバの max_allowed_packet 変数が net_buffer_length よりも大きいことを確認する。

一般的に、mysqldump はデータベース全体のバックアップに使用されます。 See 項4.5.1.「データベースのバックアップ」。

```
shell> mysqldump --opt database > backup-file.sql
```

これを、以下のコマンドで MySQL に戻すことができます。

```
shell> mysql database < backup-file.sql
```

または

```
shell> mysql -e "source /path-to-backup/backup-file.sql" database
```

データベースの情報を別の MySQL サーバに移動することもできます。

```
shell> mysqldump --opt database | mysql ---host=remote-host -C database
```

1 つのコマンドで複数のデータベースをダンプすることができます。

```
shell> mysqldump --databases database1 [database2 ...] > my_databases.sql
```

すべてのデータベースを選択するには、以下のようにします。

```
shell> mysqldump --all-databases > all_databases.sql
```

4.9.8. mysqlhotcopy（MySQL のデータベースとテーブルのコピー）

mysqlhotcopy は、LOCK TABLES、FLUSH TABLES、および cp（または scp）を使用して、すばやくデータベースのバックアップを行う Perl スクリプトです。これは、データベースや単一のテーブルのバックアップを行う最速の方法ですが、データベースディレクトリのある同一マシンだけでしか実行できません。 mysqlhotcopy は、Unix のみ、および MyISAM テーブルと ISAM テーブルでのみ使用できます。

```
mysqlhotcopy db_name [/path/to/new_directory]

mysqlhotcopy db_name_1 ... db_name_n /path/to/new_directory

mysqlhotcopy db_name./regex/
```

mysqlhotcopy は、以下のオプションをサポートします。

- -?, --help

  ヘルプ画面を表示して終了する。

- -u, --user=#

  データベースにログインするユーザ。

- -p, --password=#

サーバとの接続に使用するパスワード。

- -P, --port=#

  ローカルサーバとの接続に使用するポート。

- -S, --socket=#

  ローカルサーバとの接続に使用するソケット。

- --allowold

  ターゲットが存在している場合も停止しない（_old を追加して名前変更）。

- --keepold

  完了時、前の（名前を変更した）ターゲットを削除しない。

- --noindices

  バックアップを小さくして処理速度を速くするため、コピーに完全インデックスファイルを含めない。 インデックスは後で、myisamchk -rq. で再構成できる。

- --method=#

  コピー方法（cp または scp）。

- -q, --quiet

  エラー以外は出力しない。

- --debug

  デバッグを有効にする。

- -n, --dryrun

  アクションを実行せずに報告する。

- --regexp=#

  regexp と一致する名前のデータベースをすべてコピーする。

- --suffix=#

  コピーされるデータベース名のサフィックス。

- --checkpoint=#

  指定したデータベーステーブルにチェックポイントエントリを挿入する。

- --flushlog

  すべてのテーブルがロックされた後、ログをフラッシュする。

- --tmpdir=#

  テンポラリディレクトリ（/tmp の代わり）。

perldoc mysqlhotcopy を使用して、mysqlhotcopy のさらに詳しいドキュメントを取得できます。

mysqlhotcopy は、オプション設定ファイルから [client] グループおよび [mysqlhotcopy] グループを読み取ります。

mysqlhotcopy を実行するには、バックアップディレクトリへの書き込みアクセス権と、コピーしようとしているテーブルの SELECT 権限、および MySQL RELOAD 権限（FLUSH TABLES を実行するため）が必要です。

4.9.9. mysqlimport（テキストファイルからのデータのインポート）

mysqlimport は、LOAD DATA INFILE SQL ステートメントのコマンドラインインタフェースを提供します。mysqlimport のほとんどのオプションが、LOAD DATA INFILE のオプションに対応しています。 See 項6.4.8.「LOAD DATA INFILE 構文」。

mysqlimport は以下のようにして起動します。

```
shell> mysqlimport [options] database textfile1 [textfile2 ...]
```

コマンドラインで指定した各テキストファイルについて、mysqlimport は、ファイル名の拡張子を削除し、その結果を使用して、どのテーブルにファイルの内容をインポートするか決定します。たとえば、patient.txt、patient.text、および patient という名前のファイルは、すべて、patient という名前のテーブルにインポートされます。

mysqlimport は、以下のオプションをサポートします。

- -c, --columns=...

  このオプションは、カンマ区切りのフィールド名一覧を引数として取る。 フィールド一覧を使用して適切な LOAD DATA INFILE コマンドを作成し、それを MySQL に渡す。 See 項6.4.8.「LOAD DATA INFILE 構文」。

- -C, --compress

  クライアントとサーバの両方が圧縮をサポートする場合、クライアントとサーバ間の情報をすべて圧縮する。

- -#, --debug[=option_string]

  プログラムのトレース使用（デバック目的）。

- -d, --delete

  テキストファイルをインポートする前にテーブルを空にする。

- --fields-terminated-by=... , --fields-enclosed-by=... , --fields-optionally-enclosed-by=... , --fields-escaped-by=... , --lines-terminated-by=...

  これらのオプションは、LOAD DATA INFILE の対応する節と同じ意味を持つ。 See 項6.4.8.「LOAD DATA INFILE 構文」。

- -f, --force

エラーを無視する。たとえば、テキストファイル用のテーブルが存在しない場合でも、残りのファイルの処理を続行する。--force を指定しない場合、テーブルが存在しなければ mysqlimport は終了する。

- --help

  ヘルプメッセージを表示して終了する。

- -h host_name, --host=host_name

  指定したホストの MySQL サーバにデータをインポートする。デフォルトのホストは localhost。

- -i, --ignore

  --replace オプションの説明を参照のこと。

- --ignore-lines=n

  データファイルの最初の n 行を無視する。

- -l, --lock-tables

  テキストファイルを処理する前に、すべてのテーブルへの書き込みをロックする。これにより、そのサーバ上のすべてのテーブルが同期化される。

- -L, --local

  クライアントから入力ファイルを読み取る。デフォルトでは、localhost (デフォルトホスト) に接続した場合、テキストファイルはサーバにあると想定される。

- -pyour_pass, --password[=your_pass]

  サーバとの接続に使用するパスワード。'=your_pass' 部分を指定しなければ、mysqlimport によってパスワードのプロンプトが表示される。

- -P port_num, --port=port_num

  接続に使用する TCP/IP ポート番号。

- --protocol=(TCP | SOCKET | PIPE | MEMORY)

  使用する接続プロトコルを指定する。MySQL 4.1 で導入。

- -r, --replace

  --replace オプションおよび --ignore オプションは、既存レコードの値と重複するユニークキー値を持つ入力レコードの処理を制御する。--replace を指定した場合、新規レコードが同じユニークキー値を持つ既存レコードを上書きする。IGNORE を指定すると、ユニークキー値が既存のレコードの値と重複する入力レコードは無視される。どちらも指定しない場合、重複キー値が検出されるとエラーになり、テキストファイルの残りの部分が無視される。

- -s, --silent

  サイレントモード。エラー発生時のみ出力する。

- -S /path/to/socket, --socket=/path/to/socket

  localhost (デフォルトホスト) との接続に使用するソケットファイル。

- -u user_name, --user=user_name

  サーバとの接続に使用する MySQL ユーザ名。デフォルト値はユーザの Unix ログイン名。

- -v, --verbose

  冗長モード。プログラムの実行内容に関する詳細を出力する。

- -V, --version

  バージョン情報を出力して終了する。

以下は、mysqlimport 使用の例です。

```
$ 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 imptest(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 |
+------+---------------+
```

4.9.10. mysqlshow (データベース、テーブル、およびカラムの表示)

mysqlshow を使用すると、既存のデータベース、テーブル、およびカラムをすばやく表示できます。

mysql プログラムでは、SHOW コマンドで同じ情報を表示できます。 See 項4.6.8. 「SHOW 構文」。

mysqlshow は以下のようにして起動します。

```
shell> mysqlshow [OPTIONS] [database [table [column]]]
```

- データベースを指定しなければ、一致するすべてのデータベースが表示される。
- テーブルを指定しなければ、データベース内で一致するすべてのデータベースが表示される。
- カラムを指定しなければ、テーブル内で一致するカラムおよびカラム型がすべて表示される。

注意: 新しい MySQL バージョンでは、ユーザが権限を持っているデータベース、テーブル、カラムだけが表示されます。

最後の引数にシェルまたは SQL ワイルドカード (*、?、%、または _) が含まれる場合、ワイルドカードに一致するものだけが表示されます。データベース名にアンダースコアが含まれる場合、テーブルおよびカラムを正しく取得するために、これらをバックラッシュでエスケープ処理してください (Unix シェルによっては 2 つ必要です)。'*' は SQL '%' ワイルドカードに、'?' は SQL '_' ワイルドカードに変換されます。このことは、_ を含むあるテーブルのカラムを表示しようとしたときに、混乱が生じます。この場合、mysqlshow は、このパターンに一致するテーブル名を表示します。 これは、コマンドラインの最後に % を別の引数として追加することで、簡単に解決できます。

4.9.11. mysql_config (クライアントをコンパイルするためのコンパイルオプションの取得)

mysql_config は、MySQL クライアントをコンパイルし、そのクライアントを MySQL に接続する方法についての役立つ情報を提供します。

mysql_config は、以下のオプションをサポートします。

- --cflags

  インクルードファイルを見つけるためのコンパイラフラグ、および重要なコンパイルフラグとコンパイラ定義。これらは libmysqlclient ライブラリをコンパイルするときに使用される。

- --include

  MySQL インクルードファイルを見つけるためのコンパイルオプション (通常はこの代わりに --cflags を使用する)。

- --libs

  MySQL クライアントライブラリにリンクする必要のあるライブラリとオプション。

- --libs_r

  スレッドセーフ MySQL クライアントライブラリにリンクする必要のあるライブラリとオプション。

- --socket

  デフォルトのソケット名。MySQL をコンフィギャするときに定義される。

- --port

  デフォルトのポート番号。MySQL をコンフィギャするときに定義される。

- --version

  MySQL ディストリビューションのバージョン番号とバージョン。

- --libmysqld-libs or --embedded

  MySQL 組み込みサーバにリンクする必要のあるライブラリとオプション。

オプションを指定せずに mysql_config を実行すると、サポートするすべてのオプションとすべてのオプション値が出力されます。

```
shell> mysql_config
Usage: /usr/local/mysql/bin/mysql_config [OPTIONS]
Options:
        --cflags         [-I/usr/local/mysql/include/mysql -mcpu=pentiumpro]
        --include        [-I/usr/local/mysql/include/mysql]
        --libs           [-L/usr/local/mysql/lib/mysql -lmysqlclient -lz -lcrypt -lnsl -lm -L/usr/lib -lssl -lcrypto]
        --libs_r         [-L/usr/local/mysql/lib/mysql -lmysqlclient_r -lpthread -lz -lcrypt -lnsl -lm -lpthread]
        --socket         [/tmp/mysql.sock]
        --port           [3306]
        --version        [4.0.16]
        --libmysqld-libs [-L/usr/local/mysql/lib/mysql -lmysqld -lpthread -lz -lcrypt -lnsl -lm -lpthread -lrt]
```

このスクリプトを使用して以下のように指定することで、MySQL クライアントをコンパイルできます。

```
CFG=/usr/local/mysql/bin/mysql_config
sh -c "gcc -o progname `$CFG --cflags` progname.c `$CFG --libs`"
```

4.9.12. perror（エラーコードの説明）

MySQL は、ほとんどのシステムエラーに対して、内部のテキストメッセージに加え、message ... (errno: #) または message ... (Errcode: #) のいずれかの形式のシステムエラーコードを出力します。

エラーコードの意味を調べるには、システムのドキュメントを参照するか、perror ユーティリティを使用します。

perror は、システムエラーコード、または MyISAM/ISAM ストレージエンジン（テーブルハンドラ）エラーコードの説明を出力します。

perror は以下のようにして起動します。

```
shell> perror [OPTIONS] [ERRORCODE [ERRORCODE...]]

Example:

shell> perror 13 64
Error code  13:  Permission denied
Error code  64:  Machine is not on the network
```

注意: エラーメッセージは、ほとんどがシステム依存です。

4.9.13. テキストファイルから SQL コマンドを実行する方法

mysql クライアントは、通常、以下のように対話的に使用します。

```
shell> mysql database
```

しかし、SQL コマンドをファイルに入力して、そのファイルから mysql に読み取らせることも可能です。これを行うには、実行するコマンドを含むテキストファイル text_file を作成します。 そして、以下のように mysql を起動します。

```
shell> mysql database < text_file
```

テキストファイルを USE db_name ステートメントで開始することもできます。この場合、コマンドラインでデータベース名を指定する必要はありません。

```
shell> mysql < text_file
```

すでに mysql を実行中の場合、source コマンドを使用して SQL スクリプトを実行できます。

```
mysql> source filename;
```

バッチモードの詳細については、項3.5.「バッチモードでの mysql の使用」 を参照してください。

4.10. MySQL ログファイル

MySQL にはいくつかの種類のログファイルがあり、mysqld 内で何が発生しているか調べることができます。

ログファイル	説明
エラーログ	mysqld の起動、実行、および停止で発生した問題。
isam ログ	ISAM テーブルへの変更がすべて記録。isam コードのデバッグのみに使用。
一般クエリログ	接続の情報と実行されたクエリ。
更新ログ	廃止。データを変更したすべてのステートメントを保存。
バイナリログ	何かを変更したすべてのステートメントを保存。レプリケーションにも使用。
スローログ	long_query_time 秒より時間のかかったクエリ、またはインデックスを使用しなかったクエリをすべて保存。

ログはすべて、mysqld データディレクトリにあります。mysqld でログファイルを再度開く (あるいは新しいログに切り替える) には、FLUSH LOGS を実行します。 See 項4.6.4.「FLUSH 構文」。

4.10.1. エラーログ

エラーログファイルには、mysqld の起動時刻と停止時刻、および実行中に発生したエラーに関する情報が記録されます。

mysqld が異常終了して mysqld_safe が mysqld を再起動する必要がある場合、mysqld_safe はこのファイルに restarted mysqld 行を書き込みます。自動チェックまたは自動修復が必要なテーブルを mysqld が見つけた場合、警告も記録されます。

一部のオペレーティングシステムでは、mysqld が異常終了した箇所のスタックトレースがエラーログに記録されます。これを使用すると、mysqld がどの時点で異常終了したかわかります。 See 項E.1.4.「スタックトレースの使用」。

MySQL 4.0.10 以降、mysqld がエラーログファイルを保存する場所を --log-error[=filename] オプションで指定できるようになっています。ファイル名を指定しなければ、mysqld は Unix では mysql-data-dir/'hostname'.err を、Windows では \mysql\data\mysql.err を使用します。 flush logs を実行すると、古いファイルには --old のプリフィックスが付き、mysqld

が空の新規ログファイルを作成します。

旧 MySQL バージョンでは、エラーログ処理は mysqld_safe によって行われ、エラーファイルは 'hostname'.err にリダイレクトされます。このファイル名は、--err-log=filename オプションで変更できます。

--log-error を指定しない場合、または --console オプションを使用した場合には、エラーは stderr (端末) に書き込まれます。

Windows では、--console を指定しなければ、出力は常に .err ファイルに書き込まれます。

4.10.2. 一般クエリログ

mysqld 内で何が発生しているか確認したい場合には、--log[=file] オプションを使用して mysqld を起動します。これにより、すべての接続とクエリがログファイル (デフォルトは 'hostname'.log) に記録されます。このログは、クライアント側にエラーの原因があると考えられ、mysqld に対してクライアントが何を送信したのか調べるときに非常に役立ちます。

古いバージョン (MySQL 3.23.4 から 3.23.8) の mysql.server スクリプトは、safe_mysqld に、一般クエリログを有効化する --log オプションを渡します。実稼動環境で MySQL の起動時のパフォーマンスを向上させる必要がある場合には、--log オプションを mysql.server から削除するか、--log-bin に変更します。 See 項4.10.4.「バイナリログ」。

このログのエントリは、mysqld がクエリを受けた時点で書き込まれます。これは、ステートメントが実行された順序とは異なる場合があります。また、更新ログおよびバイナリログとも対比的です。更新ログおよびバイナリログは、クエリが実行された後、ロックが解除される前に書き込まれます。

4.10.3. 更新ログ

注意: 更新ログは廃止され、代わりにバイナリログが使用されます。See 項4.10.4.「バイナリログ」。 バイナリログは、以前の更新ログの機能をすべて備え、加えて別の機能も追加されています。MySQL 5.0 で更新ログはなくなります。

mysqld を --log-update[=file_name] オプションで起動すると、データを更新するすべての SQL コマンドがログファイルに記録されます。ファイル名を指定しなければ、ホストマシンの名前がデフォルト名になります。ファイル名を指定し、パスを指定しない場合、ファイルはデータディレクトリに作成されます。file_name に拡張子がない場合、mysqld は file_name.### の形式でログファイル名を生成します。ここで ### は、mysqladmin refresh、mysqladmin flush-logs、および FLUSH LOGS ステートメントが実行されるたびに、およびサーバが再起動するたびにインクリメントされる番号です。

注意: 上記スキームが有効であるために、更新ログが使用するディレクトリ内では、更新ログ + 何らかの拡張子など、番号と誤認されるようなファイル名のファイルを作成してはいけません。

--log オプションまたは -l オプションを使用した場合、mysqld は hostname.log という名前の一般ログファイルを作成します。再起動やリフレッシュを行っても新規ログファイルは生成されません (いったん閉じて、再度開くだけです)。この場合、以下のようにして (Unix 上) ログファイルをコピーすることができます。

```
mv hostname.log hostname-old.log
mysqladmin flush-logs
cp hostname-old.log to-backup-directory
rm hostname-old.log
```

更新ログは、データを実際に更新したステートメントだけを記録します。UPDATE または DELETE の WHERE 指定でレコードが見つからなかった場合、そのステートメントはログに書き込まれません。カラムの値を、すでにある同じ値で更新した UPDATE ステートメントもスキップされます。

更新ログの書き込みは、クエリが完了した直後でロックの解除前に、またはコミットの前に行われます。このため、ログは実行順序どおりに書き込まれます。

更新ログファイルからデータベースを更新するには、以下を実行します。ここでは、更新ログ名の形式を file_name.### と仮定しています。

```
shell> ls -1 -t -r file_name.[0-9]* | xargs cat | mysql
```

ls は、すべてのログファイルを正しい順序で取得するために使用します。

これは、クラッシュ後にバックアップファイルの状態にまで戻す、あるいはバックアップ後からクラッシュまでの間の更新を繰り返す場合などに役立ちます。

4.10.4. バイナリログ

バイナリログは、以前の更新ログの代わりになるものです。更新ログは MySQL 5.0 でなくなります。バイナリログには、更新ログで利用可能だった情報がすべて、より効率的かつトランザクションセーフな方法で含まれます。

バイナリログは以前の更新ログと同様、データを実際に更新するステートメントだけを記録します。UPDATE または DELETE の WHERE 指定でレコードが見つからなかった場合、そのステートメントはログに書き込まれません。カラムの値を、すでにある同じ値で更新した UPDATE ステートメントもスキップされます。

バイナリログには、バックアップ後の更新がすべて記録されます。バイナリログの主な目的は、リストア時に、データベースに対して可能な限りバックアップ後の更新を実行できるようにすることです。

バイナリログは、マスタからスレーブにレプリケーションを行うときにも使用します。 See 項4.11.「MySQL のレプリケーション」。

バイナリログには、データベースを更新した各クエリにかかった時間の情報も含まれます。データを変更しなかったクエリは含まれません。問題のあるクエリを見つけるなどの目的ですべてのクエリを記録したい場合には、一般クエリログを使用してください。 See 項4.10.2.「一般クエリログ」。

mysqld を --log-bin[=file_name] オプションで起動すると、データを更新したすべての SQL コマンドがログファイルに記録されます。ファイル名を指定しなければ、ホストマシンの名前に -bin を付けたものがデフォルト名になります。ファイル名を指定し、パスを指定していない場合には、ファイルはデータディレクトリに作成されます。

--log-bin=filename.extension で拡張子を指定した場合、拡張子は削除されます。

mysqld は、バイナリログファイル名に数字の拡張子を追加します。この番号は、mysqladmin refresh、mysqladmin flush-logs、および FLUSH LOGS ステートメントが実行されるたびに、またはサーバが再起動するたびにインクリメントされます。バイナリログのサイズが max_binlog_size に達すると、新しいバイナリログが自動的に作成されます。注意: トランザクションを使用している場合、トランザクションは 1 つのまとまりでバイナリログに書き込まれるため、複数のバイナリログに分割されることはありません。したがって、大きなトランザクションがある場合、バイナリログが max_binlog_size より大きくなる可能性があります。

すべてのバイナリログファイルを削除するには、RESET MASTER コマンド (see 項4.6.5.「RESET 構文」) を使用します。一部のファイルを削除するには、PURGE MASTER LOGS (see 項4.11.7.「マスタサーバを制御する SQL ステートメント」) を使用します。

mysqld で以下のオプションを使用し、何をバイナリログに記録するか制御できます (表の後の説明を必ず読んでください

）。

オプション	説明
binlog-do-db=database_name	カレントデータベース（USE によって選択されたデータベース）が 'database_name' の場合、更新をバイナリログに記録するようにマスタに指示する。それ以外の明示的に指定されていないデータベースはすべて無視する。注意: カレントデータベース内でのみ更新を行うことが確実な場合に、このオプションを使用すること（例: binlog-do-db=some_database）。 予想しづらい動作例: サーバを binlog-do-db=sales で起動し、USE prices; UPDATE sales.january SET amount=amount+1000; を実行した場合、このクエリはバイナリログには書き込まれない。
binlog-ignore-db=database_name	カレントデータベース（USE によって選択されたデータベース）が 'database_name' の場合、更新をバイナリログに記録しないようにマスタに指示する。注意: カレントデータベース内でのみ更新を行うことが確実な場合に、このオプションを使用すること（例: binlog-ignore-db=some_database）。 予想しづらい動作例: サーバを binlog-ignore-db=sales で起動し、USE prices; UPDATE sales.january SET amount=amount+1000; を実行した場合、このクエリはバイナリログに書き込まれる。

クエリをバイナリログに書き込むかどうかの評価は、以下の順序で行われます。

1. binlog-do-db ルールまたは binlog-ignore-db ルールがあるか。
   - No: クエリをバイナリログに書き込んで終了する。
   - Yes: 次のステップに移る。
2. ルール（binlog-do-db または binlog-ignore-db、あるいはその両方）が存在する。カレントデータベースがあるか（USE が選択しているデータベースがあるか）。
   - No: クエリを書き込まないで終了する。
   - Yes: 次のステップに移る。
3. カレントデータベースがある。binlog-do-db ルールはあるか。
   - Yes: カレントデータベースが binlog-do-db ルールのいずれかに一致しているか。
     - Yes: クエリを書き込んで終了する。
     - No: クエリを書き込まないで終了する。
   - No: 次のステップに移る。
4. binlog-ignore-db ルールが存在する。 カレントデータベースが binlog-ignore-db ルールのいずれかに一致しているか。
   - Yes: クエリを書き込まないで終了する。
   - No: クエリを書き込んで終了する。

したがって、たとえば binlog-do-db=sales だけで実行中のスレーブは、カレントデータベースが sales ではないクエリを一切バイナリログに書き込みません (つまり、binlog-do-db は ``他のデータベースを無視する" ということにもなります) 。

どのバイナリログファイルが使用されたかわかるように、mysqld はバイナリログインデックスファイルも作成します。このファイルに、使用されたバイナリログファイルすべての名前が含まれます。デフォルトでは、このファイルの名前はバイナリログファイル名に拡張子 '.index' を付けたものとなります。 バイナリログインデックスファイルの名前を変更するには、--log-bin-index=[filename] オプションを使用します。 mysqld の実行中は、このファイルを手動で編集しないでください。mysqld が混乱する原因となります。

レプリケーションを使用している場合、スレーブが必要としている間は、古いバイナリログファイルを削除しないでください。 たとえば、mysqladmin flush-logs を毎日実行する場合、3 日たったログをすべて削除するようにします。これらのログは手動で削除することもできますが、バイナリログインデックスファイルも安全に更新できる PURGE MASTER LOGS (see 項4.11.7.「マスタサーバを制御する SQL ステートメント」) を使用して削除することを推奨します。このコマンドは、MySQL 4.1 から日付引数を指定できるようになっています。

SUPER 権限での接続では、SET SQL_LOG_BIN=0 を使用して、クエリのバイナリログを無効にできます。 See 項4.11.7.「マスタサーバを制御する SQL ステートメント」。

バイナリログファイルを調べるには、mysqlbinlog ユーティリティを使用します。 たとえば、以下のようにバイナリログから MySQL サーバを更新することができます。

```
shell> mysqlbinlog log-file | mysql -h server_name
```

mysqlbinlog ユーティリティおよびその使用法に関する詳細については、項4.9.5.「mysqlbinlog (バイナリログからクエリを実行する) 」 を参照してください。

BEGIN [WORK] または SET AUTOCOMMIT=0 を使用している場合は、以前の更新ログではなく MySQL バイナリログをバックアップ用に使用してください。更新ログは MySQL 5.0 でなくなります。

バイナリログの書き込みは、クエリが完了した直後でロックの解除前に、またはコミットの前に行われます。このため、ログは実行順序どおりに書き込まれます。

非トランザクションテーブルの更新は実行直後にバイナリログに保存されます。BDB テーブルや InnoDB テーブルなどのトランザクションテーブルでは、テーブルを変更するすべての更新 (UPDATE、DELETE、または INSERT) は、COMMIT コマンドがサーバに送信されるまでキャッシュされます。mysqld は、COMMIT が実行される前にトランザクション全体をバイナリログに書き込みます。 すべてのスレッドが、最初に、クエリをバッファする binlog_cache_size のバッファを割り当てます。クエリがこれより大きい場合、スレッドはテンポラリファイルを開いてトランザクションを保存します。スレッドが終了するとテンポラリファイルは削除されます。

max_binlog_cache_size (デフォルトは 4G) を使用して、複数クエリトランザクションのキャッシュに使用するトータルサイズを制限できます。トランザクションがこれより大きい場合、エラーとなってロールバックします。

更新ログまたはバイナリログを使用している場合に、CREATE ... SELECT または INSERT ... SELECT を使用すると、同時挿入が通常の挿入に変換されます。 これは、バックアップにログを適用したときに、テーブルの正確なコピーを作成するための措置です。

4.10.5. スロークエリログ

mysqld を --log-slow-queries[=file_name] オプションで起動すると、実行に long_query_time 秒より長くかかった SQL コ

マンドがすべてログファイルに書き込まれます。最初のテーブルロックにかかった時間は実行時間に含まれません。

スロークエリログは、クエリ実行後、すべてのロックが解除された後に書き込まれます。これは、ステートメントが実行された順序とは異なる場合があります。

ファイル名を指定しなければ、ホストマシンの名前に -slow.log を付けたものがデフォルト名になります。ファイル名を指定し、パスを指定しない場合、ファイルはデータディレクトリに作成されます。

スロークエリログは、実行に時間がかかり、最適化の対象となるクエリを見つけるために使用できます。大きなログでは、これは難しい作業になります。mysqldumpslow コマンドを使用すれば、ログのクエリサマリを取得できます。

--log-long-format を使用すれば、インデックスを使用しなかったクエリも出力されます。 See 項4.1.1.「mysqld コマンドラインオプション」。

4.10.6. ログファイルの保守

MySQL サーバではいくつかの種類のログファイルが生成され、実行内容を簡単に確認することができます。See 項4.10.「MySQL ログファイル」。 ただし、ログがディスク領域を占有しすぎないようにこれらのファイルを定期的にクリーンアップする必要があります。

MySQL をログファイルとともに使用する場合、定期的に古いファイルを削除またはバックアップし、新しいファイルでログを開始するようにしてください。 See 項4.5.1.「データベースのバックアップ」。

Linux (Red Hat) インストールでは、mysql-log-rotate スクリプトを使用して、古いログファイルの処理を自動的に行うことができます。RPM ディストリビューションから MySQL をインストールしていれば、このスクリプトは自動的にインストールされているはずです。注意: レプリケーション用にバイナリログを使用している場合、このスクリプトの使用には注意が必要です。

他のシステムでは、ログファイルを処理するための短いスクリプトを自分でインストールする必要があります。このスクリプトは、cron から開始します。

新しいログファイルの使用を MySQL に強制するには、mysqladmin flush-logs を使用するか、SQL コマンド FLUSH LOGS を使用します。 MySQL バージョン 3.21 を使用している場合、mysqladmin refresh を使用する必要があります。

上記コマンドは以下を実行します。

- 一般ログ (--log) またはスロークエリログ (--log-slow-queries) が使用されている場合、ログファイル (デフォルトでは mysql.log と `hostname`-slow.log) を閉じて再び開く。
- 更新ログ (--log-update) が使用されている場合、更新ログを閉じて、1 つ番号の大きい新規ログファイルを開く。

更新ログだけを使用している場合は、ログをフラッシュして、古い更新ログファイルをバックアップに移動するだけで済みます。 通常のログを使用している場合は、以下のように行います。

```
shell> cd mysql-data-directory
shell> mv mysql.log mysql.old
shell> mysqladmin flush-logs
```

次に、バックアップを作成し、mysql.old を削除します。

4.11. MySQL のレプリケーション

1 つの MySQL サーバにあるデータベースを別のサーバに複製できるレプリケーション機能は、MySQL バージョン 3.23.15 で導入されました。このセクションでは、MySQL のさまざまなレプリケーション機能について説明します。このセクションは、レプリケーションで利用可能なオプションのリファレンスとしても役立ちます。まずレプリケーションについて説明し、その実装方法を解説します。後半では、FAQ、および発生し得る問題の説明とその解決方法を記述します。

当社の Web サイト http://www.mysql.com/ を定期的にアクセスし、このセクションの最新版に目を通しておくことを推奨します。レプリケーションは常に改良されており、頻繁にマニュアルを最新の情報で更新しています。

4.11.1. はじめに

バージョン 3.23.15 から、MySQL は一方向レプリケーションを内部的にサポートしています。1 つのサーバがマスタとして機能し、別の 1 つまたは複数のサーバがスレーブとなります。マスタサーバが更新のバイナリログを保持します (see 項4.10.4.「バイナリログ」)。 また、バイナリログのインデックスファイルも保持し、ログのローテーションを記録します。各スレーブは接続時、前回レプリケーションに成功にしている更新の位置をマスタに通知し、それ以降の更新を実行し、その後ブロックして、マスタからの新規更新の通知を待ちます。

レプリケーションサーバをチェーン状に構成すれば、スレーブは他のサーバに対するマスタとしても機能できます。

注意: レプリケーションを使用する場合、テーブルの更新はすべてマスタサーバで実行する必要があります。そうでない場合、ユーザがマスタのテーブルで行った更新とスレーブのテーブルで行った更新との間で競合が発生します。

一方向レプリケーションは、堅牢性、速度、およびシステム管理の面で優れています。

- マスタ/スレーブセットアップにより堅牢性が向上する。 マスタで問題があった場合、バックアップとしてスレーブに切り替えることができる。
- クライアントクエリの負荷をマスタとスレーブに分散することによって速度を向上でき、その結果クライアントの応答時間が短くなる。 SELECT クエリをスレーブに送信することにより、マスタのクエリ処理負荷を軽減できる。データを変更するクエリは、マスタとスレーブの整合性を保つため、マスタに送信する必要がある。負荷分散戦略は、更新以外のクエリが多い場合のみ効果がある。通常、この場合が多い。
- その他にも、レプリケーションを使用すると、マスタではなくスレーブでバックアップを行えるという利点がある。 See 項4.5.1.「データベースのバックアップ」。

4.11.2. レプリケーション実装の概要

MySQL レプリケーションは、マスタサーバが、データベースに対するすべての変更 (更新、削除など) をバイナリログ (see 項4.10.4.「バイナリログ」) に記録することを基本としています。各スレーブサーバは、マスタがそのバイナリログに記録したクエリをマスタから受け取り、データのコピー上で同じクエリを実行できます。

バイナリログは、単に、ある特定の時刻 (バイナリログを有効にした瞬間) からの記録であることを理解することが重要です。スレーブにはすべて、マスタでバイナリログを有効にした瞬間のマスタデータベースのコピーが必要です。バイナリログが開始されたときのマスタ上のデータとは違うデータでスレーブを開始すると、そのスレーブは失敗します。

以下の表は、MySQL の異なるバージョン間でのマスタ/スレーブレプリケーションの互換性を示したものです。

		マスタ	マスタ	マスタ	マスタ
		3.23.33 以降	4.0.0	4.0.1	4.0.3 以降
スレーブ	3.23.33 以降	yes	no	no	no
スレーブ	4.0.0	no	yes	no	no
スレーブ	4.0.1	yes	no	yes	no
スレーブ	4.0.3 以降	yes	no	no	yes

レプリケーション機能は常により強力な機能へと変化しているので、最新の MySQL バージョンを常に使用することを推奨します。バージョン 4.0 については、マスタとスレーブの両方に同じバージョンの使用を推奨します。例外として、MySQL 4.0.2 はレプリケーション目的には適しません。

注意: マスタを MySQL 3.23 から MySQL 4.0 (または 4.1) にアップグレードする場合、古い 3.23 バイナリログを使用してレプリケーションを再開しないでください。4.0 スレーブが混乱する原因となります。アップグレードするマスタが 3.23 で、スレーブが 4.0 の場合は以下の方法で安全にアップグレードできます。

1. マスタでの更新をすべてブロックする (FLUSH TABLES WITH READ LOCK) 。
2. すべてのスレーブがマスタの更新に追いつくのを待つ (マスタで SHOW MASTER STATUS を使用し、スレーブで SELECT MASTER_POS_WAIT() を使用する) 。そして、スレーブで STOP SLAVE を実行する。
3. マスタの MySQL をシャットダウンし、マスタを MySQL 4.0 にアップグレードする。
4. マスタで MySQL を再起動する。マスタの新しく作成しされたバイナリログの名前 <name> を記録する。ファイルの名前を取得するには、マスタで SHOW MASTER STATUS を実行する。各スレーブに以下のコマンドを実行する。

```
mysql> CHANGE MASTER TO MASTER_LOG_FILE='<name>', MASTER_LOG_POS=4;
mysql> START SLAVE;
```

スレーブも 3.23 から 4.0 にアップグレードする必要がある場合は、最初にスレーブをアップグレードしてください。それぞれをシャットダウンし、アップグレードしてから再起動します。 それから、上記の手順でマスタをアップグレードします。

4.0.0 以降、LOAD DATA FROM MASTER を使用してスレーブをセットアップできるようになっています。現在のところ、LOAD DATA FROM MASTER が動作するのは、マスタのテーブルがすべて MyISAM 型の場合だけです。また、このステートメントではグローバルに READ ロックが掛かるため、テーブルをマスタから転送している間は書き込みできなくなります。MySQL 5.0 でロックフリーのホットテーブルバックアップが導入されると、グローバルの READ ロックは必要なくなります。

これらの制限のため、現時点では、マスタのデータセットが比較的小さい場合、またはマスタの READ ロックが長くなってもかまわない場合だけ、LOAD DATA FROM MASTER を使用してください。LOAD DATA FROM MASTER の実際の速度はシステムによって異なりますが、だいたいの目安としては、データファイル 1 メガバイトあたり 1 秒と考えてください。これは、マスタとスレーブが同等の 700 MHz Pentium で、100 MBit/秒 ネットワークの接続を想定した基準です。あくまでおおよその目安です。

いったんスレーブを適切に設定して実行すれば、スレーブはマスタに接続して更新処理を待ちます。マスタとの接続が切れた場合、再接続できるまで定期的に接続を再試行し、接続できたら更新のリッスンを再開します。再試行の間隔は、-

-master-connect-retry オプションで制御します。デフォルトは 60 秒です。

各スレーブは、終了時点を記録しています。マスタサーバ側では、いくつのスレーブが存在し、どれがいつ更新されたかを記録していません。

4.11.3. レプリケーションの実装の詳細

レプリケーションには 3 つのスレッドが関連しています。1 つがマスタ、2 つがスレーブのスレッドです。 START SLAVE が発行されると、I/O スレッドがスレーブに作成されます。これはマスタに接続し、マスタのバイナリログに記録されているクエリの送信を要求します。 次に、これらのバイナリログを送信するためのスレッドがマスタに作成されます。このスレッドは、マスタの SHOW PROCESSLIST 出力で Binlog Dump として識別できます。 I/O スレッドは、マスタ Binlog Dump スレッドが送信したものを読み取り、スレーブ内のリレーログと呼ばれるデータディレクトリのローカルファイルにコピーします。 最後に、SQL スレッドがスレーブに作成されます。これはリレーログを読み取り、それに含まれるクエリを実行します。

注意: マスタには、接続しているスレーブサーバ 1 つにつき、1 つのスレッドがあります。

SHOW PROCESSLIST を使用すれば、レプリケーションに関するマスタおよびスレーブでの処理内容を確認できます。

以下の例で、3 つのスレッドが SHOW PROCESSLIST でどのように表示されるか示します。出力形式は、MySQL バージョン 4.0.15 の SHOW PROCESSLIST のものです。旧バージョンに比べて State カラムの内容が充実しています。

マスタサーバでは、以下のような出力になります。

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 2
   User: root
   Host: localhost:32931
     db: NULL
Command: Binlog Dump
   Time: 94
  State: Has sent all binlog to slave; waiting for binlog to be updated
   Info: NULL
```

スレーブサーバでは、以下のような出力になります。

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 10
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 11
  State: Waiting for master to send event
   Info: NULL
*************************** 2. row ***************************
     Id: 11
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 11
  State: Has read all relay log; waiting for the slave I/O thread to update it
```

Info: NULL

ここでは、スレッド Id 2 がマスタです。スレッド Id 10 はスレーブの I/O スレッドです。スレッド Id 11 はスレーブの SQL スレッドです。Time カラムの値により、スレーブがマスタと比較してどれくらい遅れているかわかります (see 項 4.11.9.「レプリケーション FAQ」) 。

以下の一覧は、マスタの Binlog Dump スレッドの State カラムに表示される一般的な状態をまとめたものです。このスレッドがマスタサーバになければ、レプリケーションは実行されていないということです。

- Sending binlog event to slave

  バイナリログはイベントで構成される。通常、イベントとは、クエリとその他の情報が組み合わさったものである。このステータスは、スレッドが、バイナリからイベントを読み取り、それをスレーブに送信中ということである。

- Finished reading one binlog; switching to next binlog

  スレッドは 1 つのバイナリログファイルの読み取りを完了し、スレーブに送信するために次のファイルを開いている途中である。

- Has sent all binlog to slave; waiting for binlog to be updated

  スレッドはすべてのバイナリログファイルの読み取りを完了し、アイドル状態である。スレッドは、マスタで実行される新しい更新クエリの結果として、バイナリログに新しいイベントが書き込まれるのを待っている。

- Waiting to finalize termination

  スレッド停止中に瞬間的に発生する状態。

以下は、スレーブサーバの I/O スレッドの State カラムに表示される一般的な状態です。MySQL 4.1.1 以降、この状態は SHOW SLAVE STATUS 出力の Slave_IO_State カラムにも表示されます。そのため、SHOW SLAVE STATUS を使用するだけで現状を把握できます。

- Connecting to master

  スレッドは、マスタへの接続を試行中である。

- Checking master version

  マスタへの接続が確立した直後、瞬間的に発生する状態。

- Registering slave on master

  マスタへの接続が確立した直後、瞬間的に発生する状態。

- Requesting binlog dump

  マスタへの接続が確立した直後、瞬間的に発生する状態。スレッドは、バイナリログの内容をマスタに要求する。バイナリログファイルの名前と場所が最初にマスタに送信される。

- Waiting to reconnect after a failed binlog dump request

バイナリログダンプ要求が失敗した（接続切断によって）場合、スレッドはスリープ中この状態になる。スレッドは再試行する前に、master-connect-retry 秒間スリープ状態となる。

- Reconnecting after a failed binlog dump request

  次に、スレッドはマスタへの再接続を試行する。

- Waiting for master to send event

  スレッドは接続を完了し、バイナリログイベントの到着を待っている。マスタがアイドル状態であれば、この状態が長く続く場合がある。待機が slave_read_timeout 秒間続くと、タイムアウトが発生する。その時点でスレッドは接続が切断したと見なし、再接続を試行する。

- Queueing master event to the relay log

  スレッドがイベントの読み取りを完了し、SQL スレッドがそのイベントを処理できるようにリレーログにコピー中である。

- Waiting to reconnect after a failed master event read

  読み取り中にエラーが発生した（接続切断のため）。スレッドは接続を再試行する前に、master-connect-retry 秒間スリープ状態になる。

- Reconnecting after a failed master event read

  次に、スレッドは再接続を試行する。接続が再度確立すると、状態は Waiting for master to send event になる。

- Waiting for the slave SQL thread to free enough relay log space

  relay_log_space_limit の値が 0 以外に設定されているが、リレーログが大きくなりすぎて、その合計サイズがこの値を超えた。SQL スレッドがリレーログの内容を処理してリレーログファイルを削除し、領域に余裕ができるのを I/O スレッドは待っている。

- Waiting for slave mutex on exit

  スレッド停止中に瞬間的に発生する状態。

以下は、スレーブサーバの SQL スレッドの State カラムで表示される一般的な状態です。

- Reading event from the relay log

  スレッドは、イベントを処理するため、リレーログからイベントを読み取っている。

- Has read all relay log; waiting for the slave I/O thread to update it

  スレッドは、リレーログファイルのイベントをすべて処理し、I/O スレッドがリレーログに新しいイベントを書き込むのを待っている。

- Waiting for slave mutex on exit

  スレッド停止中に瞬間的に発生する状態。

I/O スレッドの State カラムには、クエリ文字列が表示される場合もあります。これは、スレッドがリレーログからイベントを読み取り、クエリを抽出して、そのクエリを実行中であることを示します。

MySQL 4.0.2 より前は、スレーブの I/O スレッドと SQL スレッドは 1 つのスレッドにまとめられており、リレーログファイルは使用されていませんでした。 2 つのスレッドを使用する利点は、クエリの読み取りとクエリの実行を 2 つの独立したタスクに分けられることです。そのため、クエリ実行が遅くてもクエリ読み取りが遅くなることはありません。たとえば、スレーブサーバがしばらくの間実行していなかった場合でも、スレーブが起動すると、I/O スレッドがバイナリログのすべての内容をすばやく読み出します。SQL スレッドがかなり遅れていて、追いつくのに数時間かかるとしても影響を受けません。SQL スレッドが読み出したクエリをすべて実行する前にスレーブが停止しても、少なくとも I/O スレッドがクエリの読み出しを完了しているので、クエリのコピーはスレーブのリレーログにローカルで安全に保存されており、次回スレーブが起動したときに実行することができます。このため、マスタからはバイナリログを削除できます。スレーブがその内容を読み出すのを待つ必要はないからです。

デフォルトでは、リレーログの名前は host_name-relay-bin.nnn 形式になります。ここで、host_name はスレーブサーバのホスト名、nnn はシーケンス番号です。 連続したリレーログファイルでは、シーケンス番号は 001 から始まる連続した番号になります。 スレーブは、現在使用中のリレーログの記録をインデックスファイルに保持します。デフォルトのリレーログインデックスファイル名は host_name-relay-bin.index です。 デフォルトでは、これらのファイルはスレーブのデータディレクトリに作成されます。デフォルトのファイル名は、--relay-log および --relay-log-index の各サーバオプションで上書きできます。

リレーログはバイナリログと同じ形式なので、mysqlbinlog で読み取ることができます。リレーログは必要がなくなると（そのイベントがすべて実行されたら）、SQL スレッドによって自動的に削除されます。SQL スレッドが自動的に行うため、リレーログを削除するコマンドはありません。ただし、MySQL 4.0.14 からは FLUSH LOGS がリレーログをローテートするため、SQL が削除するタイミングに影響します。

新しいリレーログは、以下の条件で作成されます。

- スレーブサーバの起動後、最初に I/O スレッドが開始されたとき（MySQL 5.0 では、最初の I/O スレッド開始時だけでなく、I/O スレッドが開始されるたびに新規リレーログが作成される）。
- FLUSH LOGS ステートメントの実行時（4.0.14 以降）。
- 現在のリレーログファイルのサイズが大きくなりすぎたとき。``大きすぎる" かどうかは以下の変数で決定される。
  - max_relay_log_size（max_relay_log_size > 0 の場合）
  - max_binlog_size（max_relay_log_size = 0 または MySQL バージョンが 4.0.14 よりも古い場合）

スレーブレプリケーションサーバは、2 つの小さなファイルをデータディレクトリに作成します。これらのファイルの名前はデフォルトで、master.info および relay-log.info になります。これらのファイルには、SHOW SLAVE STATUS ステートメント（このコマンドの説明については、see 項4.11.8.「スレーブサーバを制御する SQL ステートメント」）の出力に含まれるような情報が保存されます。ディスクファイルとして、スレーブがシャットダウンしても保持されます。次回スレーブが起動したとき、マスタからのバイナリログ読み取りとリレーログの処理がどこまで進んでいるかの情報を、スレーブはこれらのファイルから読み取ることができます。

master.info ファイルは I/O スレッドによって更新されます。このファイル内の行と SHOW SLAVE STATUS の出力結果内のカラムとの対応は以下のとおりです。

行	説明

1	Master_Log_File
2	Read_Master_Log_Pos
3	Master_Host
4	Master_User
5	パスワード (SHOW SLAVE STATUS では表示されない)
6	Master_Port
7	Connect_Retry

relay-log.info ファイルは SQL スレッドによって更新されます。ファイル内の行と SHOW SLAVE STATUS の出力結果内のカラムとの対応は以下のとおりです。

行	説明
1	Relay_Log_File
2	Relay_Log_Pos
3	Relay_Master_Log_File
4	Exec_Master_Log_Pos

スレーブのデータをバックアップするとき、リレーログファイルとともにこれら 2 つの小さなファイルもバックアップしてください。スレーブのデータをリストア後、レプリケーションを再会するのに必要となります。リレーログを失った場合でも relay-log.info ファイルがあれば、SQL スレッドがマスタバイナリログをどこまで実行したか調べることができます。そうすれば、CHANGE MASTER TO を MASTER_RELAY_LOG オプションおよび MASTER_RELAY_POS オプションとともに使用して、その箇所からバイナリログを再読み取りするようにスレーブに指示できます。これはもちろん、バイナリログがまだマスタサーバに存在することが前提となります。

スレーブが、LOAD DATA INFILE ステートメントもレプリケーションしなくてはならない場合、SQL_LOAD-* ファイルもバックアップしてください。このファイルは、この目的でスレーブが使用するディレクトリ内にあります。LOAD DATA INFILE ステートメントが中断されていた場合、レプリケーションを再開するためにスレーブはこれらのファイルを必要とします。ディレクトリの場所は、--slave-load-tmpdir オプションを使用して指定します。指定しなければ、デフォルト値は tmpdir 変数の値となります。

4.11.4. レプリケーションのセットアップ方法

以下、バージョン 4.1 の MySQL サーバに対する完全なレプリケーションのセットアップ方法について簡単に説明します。 すべてのデータベースのレプリケーションを希望し、過去にレプリケーションの設定をまったく行っていない場合を前提としています。 以下の手順を実行するには、マスタサーバをいったんシャットダウンする必要があります。

以下手順では、1 つのスレーブをセットアップしているだけですが、同様に複数のスレーブをセットアップできます。

この方法が最も簡単でわかりやすいですが、唯一の方法ではありません。たとえば、マスタのデータのスナップショットがすでにあり、マスタにサーバ ID が設定されてバイナリログが有効になっていれば、マスタをシャットダウンしたり更新をブロックしなくてもスレーブをセットアップできます。詳細については、項4.11.9.「レプリケーション FAQ」 を参照してください。

MySQL レプリケーションのセットアップを管理する場合は、この章全体をよく読み、項4.11.7.「マスタサーバを制御する SQL ステートメント」 および 項4.11.8.「スレーブサーバを制御する SQL ステートメント」 で説明されているすべてのコマンドを試してみてください。また、項4.11.6.「レプリケーションスタートアップオプション」 で説明されている my.cnf のレプリケーションスタートアップオプションにも精通してください。

注意: ここでの手順および後続セクションで説明するレプリケーション SQL ステートメントの実行には、SUPER 権限が必要です。MySQL 4.0.2 より前では、代わりに PROCESS 権限を使用します。

1. マスタとスレーブに MySQL の最近のバージョンがインストールされていることを確認する。また、項4.11.2.「レプリケーション実装の概要」 の表で、それらのバージョンに互換性があることを確認する。

   バグについては、その問題が最新のリリースで存在することを確認してから報告すること。

2. スレーブサーバが接続に使用する MySQL ユーザーをマスタサーバに設定する。このユーザーには、REPLICATION SLAVE 権限を与える必要がある（MySQL バージョンが 4.0.2 より古い場合は、代わりに FILE 権限を与える）。MySQL ユーザーがレプリケーション目的のみのものであれば（推奨）、他に権限を設定する必要はない。

   スレーブのホスト名を、各スレーブサーバがそのユーザーを使用してマスタに接続できるように設定する。たとえば、どのホストからでもマスタにアクセスできる repl という名前のユーザを作成するには、以下のコマンドを使用する。

   ```
   mysql> GRANT REPLICATION SLAVE ON *.* TO repl@'%' IDENTIFIED BY '<password>';
   ```

   MySQL バージョンが 4.0.2 より古い場合は、代わりに以下のコマンドを使用する。

   ```
   mysql> GRANT FILE ON *.* TO repl@'%' IDENTIFIED BY '<password>';
   ```

   LOAD TABLE FROM MASTER ステートメントまたは LOAD DATA FROM MASTER ステートメントをスレーブホストから使用する予定であれば、この MySQL ユーザーにさらに権限を設定する必要がある。

   - MySQL ユーザーに SUPER および RELOAD の各グローバル権限を設定する。
   - ロード対象のすべてのテーブルに対する SELECT 権限を設定する。MySQL ユーザーが SELECT を実行できないマスタテーブルは、LOAD DATA FROM MASTER で無視される。

3. MyISAM テーブルを使用する場合、FLUSH TABLES WITH READ LOCK コマンドを実行してすべてのテーブルをフラッシュし、クエリ書き込みをブロックする。

   ```
   mysql> FLUSH TABLES WITH READ LOCK;
   ```

   そして、マスタサーバのデータのスナップショットを撮る。

   スナップショットを作成する最も簡単な方法は、アーカイブプログラム（Unix では tar、Windows では PowerArchiver、WinRAR、WinZip など）を使用して、マスタのデータディレクトリにデータベースのアーカイブを作成することである。 たとえば、tar を使用してすべてのデータベースを含むアーカイブを作成するには、カレントディレクトリをマスタサーバのデータディレクトリに変更して、以下のコマンドを実行する。

   ```
   shell> tar -cvf /tmp/mysql-snapshot.tar .
   ```

   this_db という名前のデータベースだけをアーカイブに含める場合は、以下のコマンドを使用する。

```
shell> tar -cvf /tmp/mysql-snapshot.tar ./this_db
```

そしてアーカイブファイルを、スレーブサーバホストの /tmp ディレクトリにコピーする。そのマシンで、カレントディレクトリをスレーブのデータディレクトリに変更し、以下のコマンドを使用してアーカイブファイルをアンパックする。

```
shell> tar -xvf /tmp/mysql-snapshot.tar
```

場合によっては、mysql データベースのレプリケーションが必要でないこともある。その場合は、アーカイブからこのデータベースを除外できる。また、アーカイブに、ログファイル、つまり master.info ファイルまたは relay-log.info ファイルを含める必要はない。

FLUSH TABLES WITH READ LOCK によって掛けられた読み取りロックが有効な間に、マスタ上の現在のバイナリログ名とオフセット値を読み取る。

```
mysql > SHOW MASTER STATUS;
+---------------+----------+--------------+------------------+
| File          | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+---------------+----------+--------------+------------------+
| mysql-bin.003 | 73       | test,bar     | foo,manual,mysql |
+---------------+----------+--------------+------------------+
1 row in set (0.06 sec)
```

File カラムはログの名前を、Position はオフセットを示す。上記の例では、バイナリログ値は mysql-bin.003 でオフセットは 73 である。値を記録しておく。後でスレーブをセットアップするときに、これらの値が必要となる。

スナップショットを作成してログ名とオフセットを記録したら、マスタの書き込みを再度有効にできる。

```
mysql> UNLOCK TABLES;
```

InnoDB テーブルを使用している場合、InnoDB Hot Backup ツールを使用するのが理想的である。このツールは、MySQL コマーシャルライセンス、サポート、またはバックアップツールそのものを購入することによって使用できる。このツールは、マスタサーバを一切ロックせずに整合したスナップショットを作成し、後でスレーブで使用できるように、スナップショットに対応するログ名とオフセットを記録する。このツールの詳細については、http://www.innodb.com/order.php を参照のこと。

Hot Backup ツールを使用しないで、InnoDB テーブルのスナップショットを作成する最も速い方法は、マスタサーバをシャットダウンし、InnoDB のデータファイルとログ、およびテーブル定義ファイル (.frm) をコピーすることである。現在のログファイルとオフセットを記録するには、サーバをシャットダウンする前に以下を実行する。

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SHOW MASTER STATUS;
```

次に、SHOW MASTER STATUS の出力からログ名とオフセットを控えておく。ログ名とオフセットを控えたら、テーブルのロックを解除せずにサーバをシャットダウンする。ロックを解除しないのは、スナップショットが現在のログファイルとオフセットに対応している状態を維持するためである。

```
shell> mysqladmin -uroot shutdown
```

MyISAM テーブルと InnoDB テーブルの両方に適用できる代替手段として、上記のバイナリコピーの代わりに、マス

タの SQL ダンプを使用できる。これを行うには、マスタで mysqldump --master-data を実行し、後でその SQL ダンプをスレーブに実行する。ただし、これはバイナリコピーよりも時間がかかる。

マスタが、--log-bin を有効にせずに稼動していた場合、SHOW MASTER STATUS または mysqldump で表示されるログ名と位置の値は空白となる。この場合は、ログ名として空白文字列 ('') を、オフセット値として 4 を記録しておく。

4. マスタホストの my.cnf ファイルの [mysqld] セクションに log-bin オプションが含まれていることを確認する。このセクションには、server-id=master_id オプションも含まれている必要がある。ここで、master_id は、1 から 2^32 - 1 までの整数値である。次に例を示す。

```
[mysqld]
log-bin
server-id=1
```

これらのオプションがなければ、追加してから、サーバを再起動する。

5. スレーブサーバとして使用するサーバを停止し、その my.cnf ファイルに以下を追加する。

```
[mysqld]
server-id=slave_id
```

slave_id 値は master_id 値と同様、1 から 2^32 - 1 までの整数値である。さらに、スレーブの ID はマスタの ID と異なっていることが非常に重要である。次に例を示す。

```
[mysqld]
server-id=2
```

複数のスレーブをセットアップする場合、それぞれの server-id がマスタのものと異なり、またスレーブ同士でも異なるようにする。server-id の値は IP アドレスのようなものと考えてよい。これらの ID は、レプリケーションパートナのコミュニティ内で、各サーバインスタンスを一意に識別する。

server-id 値を指定しない場合、master-host を指定していなければ 1 に設定される。それ以外は 2 に設定される。注意: server-id がなければ、マスタはすべてのスレーブからの接続を拒否し、スレーブはマスタへの接続を拒否する。したがって、server-id の省略はバイナリログによるバックアップでのみ正当化される。

6. マスタサーバのデータのバイナリバックアップを作成した場合、スレーブを開始する前に、このバックアップをスレーブサーバのデータディレクトリにコピーする。ファイルおよびディレクトリの権限が正しいことを確認する。MySQL を実行しているユーザには、マスタと同様、それらのファイルおよびディレクトリに対する読み取りおよび書き込み権限が必要である。

mysqldump を使用してバックアップを作成した場合、最初にスレーブを開始する (次のステップを参照) 。

7. スレーブサーバを起動する。レプリケーションを実行していた場合には、--skip-slave-start オプションでスレーブサーバを起動する。また、--log-warnings オプションを使用してスレーブサーバを起動するのもよい。そうすると、問題 (ネットワークや接続関連などの問題) に関するメッセージが多く出力される。

8. mysqldump を使用してマスタサーバのデータのバックアップを作成した場合、ダンプファイルをスレーブサーバにロードする。

```
shell> mysql -u root -p < dump_file.sql
```

9. <> 内の値を実際のシステムの値に置き換えて、スレーブで以下のコマンドを実行する。

```
mysql> CHANGE MASTER TO
    ->     MASTER_HOST='<master host name>',
    ->     MASTER_USER='<replication user name>',
    ->     MASTER_PASSWORD='<replication password>',
    ->     MASTER_LOG_FILE='<recorded log file name>',
    ->     MASTER_LOG_POS=<recorded log offset>;
```

以下の表は、これらの変数の最大文字列長である。

MASTER_HOST	60
MASTER_USER	16
MASTER_PASSWORD	32
MASTER_LOG_FILE	255

10. スレーブスレッドを開始する。

```
mysql> START SLAVE;
```

この手順を完了すると、スレーブはマスタに接続し、スナップショット以後の更新を実行します。

マスタに server-id を設定し忘れた場合、スレーブはマスタに接続できません。

スレーブに server-id を設定し忘れた場合、エラーログに以下のエラーが出力されます。

```
Warning: one should set server_id to a non-0 value if master_host is set.
The server will not act as a slave.
```

他の理由でスレーブがレプリケーションを実行できない場合も、スレーブのエラーファイルにエラーメッセージが表示されます。

スレーブがレプリケーションを開始すると、そのデータディレクトリに master.info ファイルと relay-log.info ファイルが作成されます。スレーブはこれら 2 つのファイルを使用して、マスタのバイナリログの処理進捗を記録します。完全に理解していない限り、これらのファイルを削除したり編集しないでください。十分に理解して行う場合には、CHANGE MASTER TO コマンドの使用を推奨します。

注意: master.info の内容には、コマンドラインまたは my.cnf で指定されるオプションより優先されるものがあります。 詳細については、項4.11.6.「レプリケーションスタートアップオプション」 を参照してください。

スナップショットを一度作成すれば、それを使用して上記手順のスレーブの部分を繰り返して、別のスレーブを設定できます。マスタのスナップショットを新たに作成する必要はありません。

4.11.5. レプリケーション機能と既知の問題

以下、サポートされていることとサポートされていないことについて説明します。

- レプリケーションは、AUTO_INCREMENT、LAST_INSERT_ID()、および TIMESTAMP の値で正しく実行される。
- USER() 関数および LOAD_FILE() 関数は変更なしにレプリケートされるため、スレーブではうまく機能しない。これは、4.1.1 より古いバージョンのスレーブの CONNECTION_ID() にも当てはまる。 MySQL 4.1 の新しい PASSWORD() 関数は、4.1.1 以降のマスタでは正常にレプリケートできるが、スレーブが 4.1.0 以降であることが前提条件となる。スレーブが古く、PASSWORD() を 4.1.x マスタからレプリケートする必要がある場合、--old-password オプションでマスタを起動する。
- SQL_MODE、UNIQUE_CHECKS、SQL_SELECT_LIMIT、SQL_AUTO_IS_NULL、TABLE_TYPE の各変数は現在のところレプリケートされない。FOREIGN_KEY_CHECKS はバージョン 4.0.14 からレプリケートされる。
- マスタとスレーブでは、同じキャラクタセット (--default-character-set) を使用する必要がある。そうしないと、マスタのキャラクタセットで一意と認識されるキーが、スレーブのキャラクタセットでは一意と見なされない場合があるため、スレーブで重複キーエラーが発生する可能性がある。
- マスタでトランザクションテーブルを使用し、スレーブで非トランザクションテーブルを使用 (同一テーブルで) している場合、スレーブが BEGIN/COMMIT ブロックの途中で停止すると、次回スレーブが BEGIN ブロックの始めから開始するという問題が発生する。 この問題は近い将来、解決する予定である。
- ユーザ変数を使用する更新クエリは、3.23 および 4.0 で正常にレプリケートされない。これは 4.1 で解決されている。注意: バージョン 5.0 以降、ユーザ変数名の大文字と小文字は区別されなくなるので、5.0 とその前のバージョンとの間でレプリケーションを設定する際は注意が必要となる。
- マスタとスレーブが両方とも 4.1.1 以降の場合、スレーブは SSL を使用してマスタに接続できる。
- 実際に発生した例を聞いたことはないが、データの変更が非決定論的、つまりクエリオプティマイザの判断で行われるようにクエリが設計されている場合 (レプリケーション外でも、通常、この方法は適切な方法でない)、論理的には、マスタとスレーブのデータが異なる可能性がある。詳細については、項1.8.6.2.「MySQL の未解決のバグと設計上の問題」 を参照のこと。
- MySQL 4.1.1 より前では、FLUSH、ANALYZE、OPTIMIZE、REPAIR の各コマンドはバイナリログに保存されず、したがってスレーブにレプリケートされない。これらのコマンドは何も変更しないため、通常は問題ではない。しかし、GRANT ステートメントを使用せずに MySQL 権限テーブルを直接更新し、mysql 権限テーブルをレプリケートした場合、新しい権限を有効化するには、スレーブで FLUSH PRIVILEGES を実行することが必要となる。また、MERGE テーブルに含まれる MyISAM テーブルの名前を変更するときに FLUSH TABLES を使用した場合、スレーブで FLUSH TABLES を手動で実行する必要がある。 MySQL 4.1.1 以降、NO_WRITE_TO_BINLOG (またはエイリアスの LOCAL) を指定しない限り、これらのコマンドはバイナリログに書き込まれる (FLUSH LOGS、FLUSH MASTER、FLUSH SLAVE、FLUSH TABLES WITH READ LOCK は除く)。 構文例については、項4.6.4.「FLUSH 構文」 を参照のこと。
- MySQL は、1 つのマスタと複数のスレーブ構成のみサポートする。将来は、カレントマスタで問題が発生した場合にマスタを自動変更するアルゴリズムを追加する予定である。また、複数のスレーブに SELECT クエリを送信することによって負荷分散を行う ``エージェント" 処理も導入する予定である。
- テンポラリテーブルもレプリケートされる。ただし、スレーブスレッドだけでなくスレーブサーバもシャットダウンし、その際、まだスレーブで実行していない更新ステートメントで使用されるテンポラリテーブルをレプリケートしていた場合、そのテンポラリテーブルはレプリケートされない。スレーブをシャットダウンすると、次回、スレーブを起動したとき、更新に必要なテンポラリテーブルは利用できない。この問題を回避するには、テンポラリテーブルが開いている間、スレーブをシャットダウンしない。代わりに、以下の手順を実行する。

1. STOP SLAVE ステートメントを実行する。
2. SHOW STATUS を使用して、Slave_open_temp_tables 変数の値をチェックする。
3. 値が 0 であれば、mysqladmin shutdown コマンドを実行してスレーブをシャットダウンする。
4. 値が 0 でなければ、START SLAVE でスレーブスレッドを再開する。
5. 次回も上記手順を繰り返す。

この問題は、近い将来解決する予定である。

- log-slave-updates を有効にすれば、循環的なマスタ/スレーブ関係でサーバを接続しても安全である。注意: ただし、異なるサーバ間で、違った手順で行った更新が引き起こすであろう問題を処理するように、クライアントコードを記述しておかなければ、多くのクエリはこの種のセットアップで正しく機能しない。

  つまり、以下のようなセットアップが可能である。

  ```
  A -> B -> C -> A
  ```

  サーバ ID は、バイナリログイベント内でエンコードされる。A は、A が作成したイベントを認識できるので、A はそのイベントを実行せず、無限ループは発生しない。しかし、この循環セットアップは、テーブル間で競合が生じない更新だけを行っている場合にしか機能しない。つまり、A と C にデータを挿入する場合、C でのレコード挿入と競合するキーのレコードを A に挿入することはできない。また、更新が適用される順序が重要な場合、2 つのサーバで同じ複数のレコードを更新することもできない。

- スレーブでクエリにエラーが発生すると、スレーブの SQL スレッドは強制終了し、スレーブエラーログにメッセージが書き込まれる。この場合、スレーブに手動で接続し、エラーの原因 (テーブルが存在しないなど) を解決し、START SLAVE を実行する。

- マスタへの接続が失われた場合、スレーブはすぐに再接続を試行する。それが失敗した場合、スレーブは master-connect-retry 秒 (デフォルトは 60) ごとに再試行する。このため、マスタをシャットダウンし、しばらく後で再起動しても安全である。スレーブはまた、ネットワーク接続停止にも対応できる。ただし、スレーブがネットワーク停止を認識するのは、マスタから slave_net_timeout 秒間、何のデータも受け取らなかった後である。そのため、ネットワーク停止の時間が短い場合は、slave_net_timeout の値を小さくすべきである。 See 項4.6.8.4.「SHOW VARIABLES」。

- スレーブのシャットダウンをクリーンに行えば、どこでレプリケーションが終了したか記録されるので、安全である。クリーンでないシャットダウンでは、システム終了前にディスクキャッシュが同期化されていない場合は特に、問題が発生する可能性がある。性能の高い無停電電源装置を用意すれば、システムの耐障害性はかなり向上する。

- MyISAM テーブルの非トランザクション性により、テーブルの更新を一部だけ実行してエラーコードを返すクエリが現れる可能性がある。複数レコード挿入で 1 つのレコードがキー制約に違反していた場合や、長い更新クエリが一部のレコードだけを更新した後に強制終了された場合に、発生する。これがマスタで発生した場合、エラーコードが正当で、かつクエリ実行結果も同じエラーコードであったとき以外、スレーブスレッドは終了し、データベース管理者の判断を待つ。このエラーコード確認動作が望ましくない場合は、--slave-skip-errors オプションで一部 (または全部) のエラーを無視できる。このオプションは、MySQL バージョン 3.23.47 より使用可能。

- BEGIN/COMMIT セグメント内部で、非トランザクションテーブルからトランザクションテーブルを更新する場合、トランザクションがコミットする前に非トランザクションテーブルを変更するスレッドがあれば、バイナリログへの更新が非同期になる。これは、トランザクションのバイナリログへの書き込みがコミット後に行われるためである。

- バージョン 4.0.15 より前では、非トランザクションテーブルへの更新は、更新実行後すぐにバイナリログに書き込まれ、トランザクションテーブルへの変更は、COMMIT 時に書き込まれるか、または ROLLBACK を使用した場合はまったく書き込まれない。バックアップまたはレプリケーションでバイナリログを使用する場合、同一トランザクションでトランザクションテーブルと非トランザクションテーブルの更新を行う際は、この点に注意しなければいけない。トランザクションテーブルと非トランザクションテーブルの更新が混在したトランザクションでのログ動作は、バージョン 4.0.15 で変更され、この問題は解決された (バイナリログでのクエリの書き込み順序に問題はなくなり、ROLLBACK の場合でも必要なクエリはすべてバイナリログに書き込まれるようになった)。ただし、最初の接続のトランザクションが完了していない時点で、2 番目の接続が非トランザクションテーブルを更新した場合の問題は残っている (2 番目の接続の更新は実行後すぐ書き込まれるため、順番が混乱する可能性はまだある)。

以下の表は、MySQL 3.23 で問題だったものを MySQL 4.0 で解決した事項です。

- LOAD DATA INFILE は、更新伝達時にデータファイルがマスタサーバにあれば、正しく処理される。
- LOAD LOCAL DATA INFILE はスキップされる。
- 3.23 では、更新内の RAND() は正しくレプリケートされなかった。RAND() が含まれる更新をレプリケートしている場合は、RAND(some_non_rand_expr) を使用する。 たとえば、RAND() への引数に UNIX_TIMESTAMP() を使用できる。これは 4.0 で解決された。

4.11.6. レプリケーションスタートアップオプション

マスタとスレーブの両方で、server-id オプションを使用して各サーバに一意のレプリケーション ID を設定する必要があります。マスタとサーバそれぞれに、1 から 2^32 - 1 までの整数から一意の番号を割り当てます。 たとえば、server-id=3 のように設定します。

マスタサーバで使用できる、バイナリログを制御するオプションについては、項4.10.4.「バイナリログ」 で説明しています。

以下の表は、スレーブサーバで使用できるオプションを説明したものです。 コマンドラインまたはオプション設定ファイルで指定できます。

注意: レプリケーションでは、以下のオプションを特殊な方法で処理します。

- --master-host
- --master-user
- --master-password
- --master-port
- --master-connect-retry

スレーブサーバの起動時に master.info ファイルが存在しない場合、オプション設定ファイルまたはコマンドラインで指定した値が使用されます。これは、サーバをレプリケーションスレーブとして初めて起動した場合や、RESET SLAVE を実行してスレーブサーバをシャットダウンし、再起動した場合などに起こることです。

スレーブサーバの起動時に master.info ファイルが存在する場合は、このファイルの値が使用されます。オプション設定ファイルまたはコマンドラインの対応するオプション値は無視されます。

my.cnf ファイルで次のオプションを指定していると仮定します。

```
[mysqld]
master-host=this_host
```

サーバをレプリケーションスレーブとして初めて起動したとき、サーバは my.cnf ファイルからこのオプションを読み取って使用します。サーバは次にその値を master.info ファイルに記録します。次回サーバを起動したとき、サーバは master.info ファイルからのみマスタホスト値を読み取ります。my.cnf ファイルを変更して別のマスタホストを指定しても、反映されません。マスタホスト値を変更したい場合には、 CHANGE MASTER TO を使用する必要があります。

MySQL 4.1.1 以降、以下のオプションも特殊な方法で処理されます。

- --master-ssl
- --master-ssl-ca
- --master-ssl-capath
- --master-ssl-cert
- --master-ssl-cipher
- --master-ssl-key

これらのオプションに対応する値が master.info ファイルに含まれています。加えて、4.1.1 のファイル形式では、最初の行に、ファイル内の行数が示されます。古いサーバを 4.1.1 にアップグレードすると、新しいサーバの起動時に、master.info は自動的に新しい形式にアップグレードされます (4.1.1 以降のものを 4.1.1 より前のバージョンにダウングレードした場合、古いサーバを最初に起動する前に、ファイルの最初の行を手動で削除する必要があります) 。

サーバは、スタートアップオプションよりも既存の master.info ファイルを優先するので、場合によっては、これらの値にはスタートアップオプションを使用せず、CHANGE MASTER TO ステートメントを使用して指定する方が適しています。項4.11.8.1.「CHANGE MASTER TO」 を参照してください。

以下、スタートアップオプションを使用してスレーブサーバを設定した例です。

```
[mysqld]
server-id=2
master-host=db-master.mycompany.com
master-port=3306
master-user=pertinax
master-password=freitag
master-connect-retry=60
report-host=db-slave.mycompany.com
```

以下の一覧は、レプリケーション制御に使用できるスタートアップオプションの説明です。

- --log-slave-updates

スレーブの SQL スレッドで実行された更新を、スレーブのバイナリログに記録するようにスレーブに指示する。デフォルトではオフ。このオプションを有効にするには、バイナリログを有効にして (--log-bin オプションを使用して) スレーブを起動する必要がある。レプリケーションサーバをチェーン状に構成するには、--log-slave-updates を使用する。たとえば、次のようにセットアップできる。

```
A -> B -> C
```

A はスレーブ B のマスタとして機能し、B はスレーブ C のマスタとして機能する。B がマスタでもあり、スレーブでもあるこの構成では、--log-slave-updates オプションで B を起動する必要がある。A と B は両方とも、バイナリログを有効にして起動する必要がある。

- --log-warnings

  スレーブにより詳細なメッセージを出力させる。たとえば、ネットワークまたは接続が切断された後で再接続に成功したというメッセージや、各スレーブスレッドがどのように開始したかについての情報メッセージを出力することができる。

  このオプションはレプリケーションの使用のみに限定されない。さまざまなサーバアクティビティに関する通知を出力する。

- --master-host=host

  マスタレプリケーションサーバのホスト名または IP アドレスを指定する。このオプションを指定しなければ、スレーブスレッドは開始できない。master.info の値を読み取れる場合は、ファイルに書かれている値が優先される。このオプション名は --bootstrap-master-host などにすべきだったという意見もあるが、現在、そのまま --master-host が使用されている。

- --master-user=username

  マスタへの接続時、スレーブスレッドが認証に使用するアカウントのユーザ名。アカウントには REPLICATION SLAVE 権限が必要 (MySQL 4.0.2 より前では、FILE 権限が必要) 。マスタユーザが設定されていない場合、ユーザ test と想定する。master.info の値を読み取れる場合は、この値が優先される。

- --master-password=password

  マスタへの接続時、スレーブスレッドが認証に使用するアカウントのパスワード。設定しなければ、空白パスワードと見なされる。master.info の値を読み取れる場合は、この値が優先される。

- --master-port=port_number

  マスタがリッスンするポート。設定しなければ、MYSQL_PORT のコンパイルされた設定が採用される。configure オプションを使用していなければ、これは 3306 となる。master.info の値を読み取れる場合、その値が優先される。

- --master-connect-retry=seconds

  マスタがダウンした場合、または接続が失われた場合、スレーブスレッドがマスタへの接続を再試行する前に、スリープ状態で待機する秒数。デフォルトは 60 である。master.info の値を読み取れる場合は、その値が優先される。

- --master-info-file=filename

スレーブがマスタに関する情報を記録するためのファイル名を指定する。デフォルトでは、データディレクトリの mysql.info。

- --master-ssl , --master-ssl-ca=file_name , --master-ssl-capath=directory_name , --master-ssl-cert=file_name , --master-ssl-cipher=cipher_list , --master-ssl-key=file_name

  これらのオプションは、SSL 使用によるマスタサーバへの安全なレプリケーション接続をセットアップするために使用する。これらの意味は、項4.4.10.5.「SSL コマンドラインオプション」 で説明した --ssl、--ssl-ca、--ssl-capath、--ssl-cert、--ssl-cipher、--ssl-key の各オプションと同じである。

  これらのオプションは、MySQL 4.1.1 から使用可能である。

- --max-relay-log-size=#

  リレーログを自動的にローテートする際に使用する。 See 項4.6.8.4.「SHOW VARIABLES」。

- --relay-log=filename

  リレーログの場所と名前を指定する。これを使用して、ホスト名に依存しないリレーログ名を指定できる。また、リレーログが大きくなる傾向があり（および max_relay_log_size の値を小さくしたくない場合で）、リレーログをデータディレクトリとは別の領域に置く必要がある場合にもこのオプションを使用できる。あるいは、複数のディスクに負荷を分散して処理速度を上げる場合にもこのオプションを使用できる。

- --relay-log-index=filename

  リレーログインデックスファイルの場所と名前を指定する。

- --relay-log-info-file=filename

  relay-log.info に別の名前を指定したり、このファイルをデータディレクトリとは別のディレクトリに配置する。

- --relay-log-purge=0|1

  リレーログファイルが不要になったときの自動パージを有効または無効にする。これは、SET GLOBAL RELAY_LOG_PURGE=0|1 で動的に変更できるグローバル変数である。デフォルト値は 1。

  このオプションは MySQL 4.1.1 から利用可能。

- --relay-log-space-limit=#

  スレーブの全リレーログの合計サイズ条件を設定する（0 は ``無制限" という意味）。スレーブマシンのハードディスクが小さい場合に便利である。限度に達すると、SQL スレッドが、クエリを実行し終わって不要になったリレーログを削除するまで、I/O スレッドは一時停止する（マスタのバイナリログを読み取らない）。注意: この制限は絶対値ではない。SQL スレッドがリレーログを削除するためにさらにイベントを必要とする場合があり、その場合は削除が可能になるまで、I/O スレッドは制限を超えて続行する。続行しなければデッドロックが発生する（MySQL 4.0.13 より前では発生していた）。 --relay-log-space-limit は、--max-relay-log-size 値の 2 倍より小さく設定してはいけない。また、--max-relay-log-size が 0 の場合は --max-binlog-size 値の 2 倍より小さく設定してはいけない。小さく設定した場合、--relay-log-space-limit が超過しているために I/O スレッドが待機している間、SQL スレッドにはパージできるリレーログがなく、I/O スレッドは一時的に --relay-log-space-limit を無視することになる。

- --replicate-do-table=db_name.table_name

指定したテーブルに対するクエリのみレプリケーションを限定するようスレーブスレッドに指示する。複数のテーブルを指定するには、コマンドを複数回、各テーブルに 1 回ずつ使用する。これは、--replicate-do-db とは対照的に、データベースをまたがった更新用に使用する。この一覧の後の説明を参照のこと。

- --replicate-ignore-table=db_name.table_name

  指定したテーブルを更新するクエリをレプリケートしないことをスレーブスレッドに指示する（そのコマンドが同時に他のテーブルを更新する場合でも）。無視するテーブルを複数指定するには、コマンドを複数回、各テーブルに 1 回ずつ使用する。これは、--replicate-ignore-db とは対照的に、データベースをまたがった更新用に使用する。この一覧の後の説明を参照のこと。

- --replicate-wild-do-table=db_name.table_name

  指定したワイルドカードパターンに一致するテーブルに対するクエリのみを、レプリケーションすることをスレーブスレッドに指示する。複数のテーブルを指定するには、コマンドを複数回、各テーブルに 1 回ずつ使用する。これは、データベースをまたがった更新用に使用する。この一覧の後の説明を参照のこと。

  例: --replicate-wild-do-table=foo%.bar% は、foo で始まるデータベースの bar で始まるテーブルを使用する更新だけをレプリケートする。

  注意: --replicate-wild-do-table=foo%.% を使用すると、このルールは CREATE DATABASE および DROP DATABASE にも伝播し、データベース名がデータベースパターン（ここでは foo%）に一致した場合、これら 2 つののステートメントがレプリケートされる（これは、% がパターンであることによって引き起こされる）。

  ワイルドカード文字 _ と % のエスケープにも注意が必要である。たとえば、my_own%db データベース（データベースの正確な名前）のすべてのテーブルをレプリケートし、my1ownAABCdb データベースのテーブルはレプリケートしない場合、_ および % をエスケープする。この場合、replicate-wild-do-table=my_own\%db のように実行する。このオプションをコマンドラインから指定した場合、システムによっては \ をエスケープする必要がある（たとえば bash シェルでは、--replicate-wild-do-table=my_own\\%db と入力する必要がある）。

- --replicate-wild-ignore-table=db_name.table_name

  指定したワイルドカードパターンに一致するテーブルに対するクエリをレプリケートしないことをスレーブスレッドに指示する。無視するテーブルを複数指定するには、コマンドを複数回、各テーブルに 1 回ずつ使用する。これは、 データベースをまたがった更新用に使用する。この一覧の後の説明を参照のこと。

  例: --replicate-wild-ignore-table=foo%.bar% は、foo で始まるデータベースの bar で始まるテーブルを使用する更新をレプリケートしない。

  注意: --replicate-wild-ignore-table=foo%.% を実行すると、ルールは CREATE DATABASE および DROP DATABASE にも伝播し、データベース名がデータベースパターン（ここでは foo%）に一致するとこれら 2 つののステートメントがレプリケートされる（これは、% がパターンであることによって引き起こされる）。

  ワイルドカード文字 _ と % のエスケープにも注意が必要である。replicate-wild-do-table の説明を参照のこと。

- --replicate-do-db=database_name

  カレントデータベース（USE によって選択されたデータベース）が database_name の場合に、このデータベースに対するクエリだけをレプリケーションするようスレーブに指示する。 複数のデータベースを指定するには、コマンドを複数回、各データベースに 1 回ずつ使用する。注意: UPDATE some_db.some_table SET foo='bar' など、データベース

をまたがるクエリは、別のデータベースが選択されていたりデータベースが選択されていなければ、レプリケートされない。データベースをまたがる更新が必要な場合は、3.23.28 以降を使用し、--replicate-wild-do-table=db_name.% を使用する。この一覧の後の説明を参照のこと。

予想しづらい動作例: スレーブを --replicate-do-db=sales で起動し、USE prices; UPDATE sales.january SET amount=amount+1000; を実行した場合、このクエリはレプリケートされない。

データベースをまたがる更新が必要な場合は、代わりに --replicate-wild-do-table=db_name.% を使用する。

この ``カレントデータベースのみチェックする" という動作は、複数のデータベースにまたがる複数のテーブルの削除や更新の場合、コマンドからだけではクエリをレプリケートすべきかどうか判別が難しいということが主な理由になる。また、カレントデータベースのみチェックする方が速い。

- --replicate-ignore-db=database_name

  カレントデータベース (USE によって選択されたデータベース) が database_name である場合に、このデータベースに対するクエリをレプリケートしないようスレーブに指示する。無視するデータベースを複数指定するには、コマンドを複数回、各データベースに 1 回ずつ使用する。テーブルをまたがる更新を行い、それらの更新をレプリケートしない場合には、このオプションを使用すべきではない。この一覧の後の説明を参照のこと。

  予想しづらい動作例: スレーブを --replicate-ignore-db=sales で起動し、USE prices; UPDATE sales.january SET amount=amount+1000; を実行した場合、このクエリはレプリケートされる。

  データベースをまたがる更新が必要な場合には、代わりに --replicate-wild-ignore-table=db_name.% を使用する。

- --replicate-rewrite-db=from_name->to_name

  カレントデータベース (USE によって選択されたデータベース) がマスタの from_name である場合、to_name に変換するようスレーブに指示する。テーブルに関連するステートメントだけが影響を受け (CREATE DATABASE、DROP DATABASE には影響しない)、かつ from_name がマスタのカレントデータベースである場合のみ影響がある。これは、データベースをまたがった更新には機能しない。注意: 変換は、--replicate-* ルールがテストされる前に行われる。

  例: replicate-rewrite-db=master_db_name->slave_db_name

- --report-host=host

  スレーブの登録中にマスタに報告されるスレーブのホスト名または IP アドレス。これらは SHOW SLAVE HOSTS の出力に表示される。スレーブに自らをマスタに登録させない場合、設定しないままにしておく。注意: スレーブの接続後、マスタが TCP/IP ソケットからスレーブの IP アドレスを読み取るだけでは十分ではない。NAT および他のルーティングの事情により、この IP は、マスタまたは他のホストからスレーブに接続するのに有効ではない場合がある。

  このオプションは MySQL 4.0.0 から利用可能。

- --report-port=port_number

  スレーブの登録中にマスタに報告するスレーブの接続ポート。スレーブがデフォルト以外のポートをリッスンする場合、または、マスタやその他のクライアントとスレーブ間に特殊なトンネルがある場合だけ、これを設定する。わからない場合は、このオプションは設定しないでおく。

このオプションは MySQL 4.0.0 から利用可能。

- --skip-slave-start

  サーバの起動時にスレーブスレッドを開始しないようスレーブサーバに指示する。ユーザが後で START SLAVE を使用して、スレーブスレッドを開始できる。

- --slave_compressed_protocol=#

  スレーブとクライアントの両方が圧縮をサポートしていれば、値が 1 の場合、そのプロトコルに圧縮が使用される。

- --slave-load-tmpdir=filename

  このオプションはデフォルトでは、tmpdir 変数の値と同じになる。スレーブ SQL スレッドが LOAD DATA INFILE コマンドをレプリケートするとき、ロードするファイルをリレーログからテンポラリファイルに抽出して、それをテーブルにロードする。マスタにロードされたファイルが非常に大きい場合、スレーブのテンポラリファイルも巨大になる。そのため、tmpdir とは別の大きなディスクにテンポラリファイルを置くことが必要になる場合に、このオプションを使用する。その場合、リレーログも大きくなるため、--relay-log オプションも使用できる。--slave-load-tmpdir は、メモリベースではなく、ディスクベースのファイルシステムをポイントすることが必要。これは、LOAD DATA INFILE のレプリケートに使用されたテンポラリファイルが、マシンをリブートしても残っていることがスレーブにとって必要なためである。

- --slave-net-timeout=#

  接続が切断されたと判断して接続を再試行する際、そのために読み取りを中止する前に、マスタからのデータを待つ秒数。最初の再試行は、このタイムアウト直後に行われる。再試行の間隔は、--master-connect-retry オプションで制御する。

- --slave-skip-errors= [err_code1,err_code2,... | all]

  指定したエラーをクエリが返しても、レプリケーションを続行するようスレーブ SQL スレッドに指示する。通常、エラーが発生するとレプリケーションは停止し、ユーザはデータの不整合を手動で解決できる。なぜエラー発生するのか原因を完全に理解していない場合には、このオプションを使用してはいけない。レプリケーションのセットアップに問題がなく、クライアントプログラムにもバグがなく、および MySQL 自体にもバグがなければ、エラーで中止になることはないはずである。このオプションを乱用すると、スレーブがマスタと大きく非同期となり、ユーザには問題発生の原因もわからなくなる。

  エラーコードには、スレーブエラーログのエラーメッセージおよび SHOW SLAVE STATUS の出力に示される番号を使用する必要がある。エラーメッセージの全一覧は、ソースディストリビューション内の Docs/mysqld_error.txt に記載されている。 サーバのエラーコードの一覧は、項12.1.「返されるエラー」 にも記載されている。

  all の値を使用してすべてのエラーメッセージを無視することもできるが、これは使用すべきではない。言うまでもなく、それを使用すればデータの整合性が保証されない。使用した場合、スレーブのデータがマスタのデータと異なってしまう可能性が十分にある。

  例:

```
--slave-skip-errors=1062,1053
--slave-skip-errors=all
```

これらのオプションのうちいくつかは、すべての --replicate-* オプションのように、スレーブサーバの起動時のみ設定でき、実行中は設定できません。これは、将来解決する予定です。

以下は、スレーブがクエリを実行するか無視するかを決定するための r--eplicate-* ルールの評価順序です。

1. --replicate-do-db ルールまたは --replicate-ignore-db ルールがあるか。
   - Yes: --binlog-do-db および --binlog-ignore-db と同じようにそれらをテストする（see 項4.10.4.「バイナリログ」）。その結果を見る。
     - ignore the query: 無視して終了する。
     - execute the query: すぐには実行せず、決定を保留して次のステップに進む。
   - No: 次のステップに移る。
2. --replicate-*-table ルールはあるか。
   - No: クエリを実行して終了する。
   - Yes: 次のステップに移る。更新されるテーブルのみ、ルールと比較される（INSERT INTO sales SELECT * FROM prices では、sales のみルールと比較される）。複数のテーブルが更新される（複数テーブルステートメントの）場合、最初に一致したテーブル（``do" または ``ignore" と一致するテーブル）が対象となる（最初のテーブルがルールと比較され、それで決定できない場合は 2 番目のテーブルがルールと比較される）。
3. --replicate-do-table ルールはあるか。
   - Yes: テーブルがいずれかに一致するか。
     - Yes: クエリを実行して終了する。
     - No: 次のステップに移る。
   - No: 次のステップに移る。
4. --replicate-ignore-table ルールはあるか。
   - Yes: テーブルがいずれかに一致するか
     - Yes: クエリを無視して終了する。
     - No: 次のステップに移る。
   - No: 次のステップに移る。
5. --replicate-wild-do-table ルールはあるか。
   - Yes: テーブルがいずれかに一致するか。
     - Yes: クエリを実行して終了する。
     - No: 次のステップに移る。

- No: 次のステップに移る。

6. --replicate-wild-ignore-table ルールはあるか。
   - Yes: テーブルがいずれかに一致するか。
     - Yes: クエリを無視して終了する。
     - No: 次のステップに移る。
   - No: 次のステップに移る。
7. --replicate-*-table ルールはいずれも一致しなかった。 これらのルールでテストするテーブルは他にあるか。
   - Yes: ループする。
   - No: 更新対象のすべてのテーブルをテストしたが、どのルールにも一致しなかった。 --replicate-do-table または --replicate-wild-do-table ルールがあるか。
     - Yes: クエリを無視して終了する。
     - No: クエリを実行して終了する。

4.11.7. マスタサーバを制御する SQL ステートメント

レプリケーションは SQL インタフェースで制御できます。このセクションでは、マスタレプリケーションサーバを管理するためのステートメントについて説明します。項4.11.8.「スレーブサーバを制御する SQL ステートメント」 では、スレーブサーバを管理するためのステートメントについて説明しています。

4.11.7.1. PURGE MASTER LOGS

```
PURGE {MASTER|BINARY} LOGS TO 'log_name'
PURGE {MASTER|BINARY} LOGS BEFORE 'date'
```

指定したログまたは指定した日付より前の、ログインデックスにリストされたバイナリログをすべて削除します。ログインデックスファイル内に記録されたリストからもログが削除されるので、指定したログが最初になります。

例:

```
PURGE MASTER LOGS TO 'mysql-bin.010';
PURGE MASTER LOGS BEFORE '2003-04-02 22:46:26';
```

BEFORE 変数は MySQL 4.1 から利用可能になっています。その日付引数の形式は 'YYYY-MM-DD hh:mm:ss' です。MASTER と BINARY はシノニムですが、BINARY は MySQL 4.1.1 以降でのみ使用できます。

削除しようとしているログのいずれかを、アクティブなスレーブが読み取り中であれば、このコマンドは何も実行せず、エラーとなります。ただし、スリープ状態のスレーブがあり、それが後で読み取る予定のログをパージしてしまうと、そのスレーブが活動を再開してもレプリケーションを実行できません。スレーブがレプリケーションを実行中にコマンドを実行しても安全です。中止する必要はありません。

最初に SHOW SLAVE STATUS を実行してすべてのスレーブをチェックし、どのログを読み取り中であるか調べ、SHOW MASTER LOGS を実行してマスタのログ一覧を表示します。すべてのスレーブで最初のログを見つけ (全スレーブが最新の状態に更新されていれば、これは一覧の最後のログになります)、削除しようとしているすべてのログをバックアップし (任意)、目的のログまでをパージします。

4.11.7.2. RESET MASTER

```
RESET MASTER
```

インデックスファイルにリストされたすべてのバイナリログを削除し、バイナリログインデックスファイルを空白にリセットします。

このステートメントは MySQL 3.23.26 より前は FLUSH MASTER という名前でした。

4.11.7.3. SET SQL_LOG_BIN

```
SET SQL_LOG_BIN = {0|1}
```

クライアントが SUPER 権限のアカウントを使用して接続している場合、現在の接続 (SQL_LOG_BIN はセッション変数) でのバイナリログの無効/有効を切り替えます。クライアントにその権限がない場合、このステートメントは無視されます。

4.11.7.4. SHOW BINLOG EVENTS

```
SHOW BINLOG EVENTS [ IN 'log_name' ] [ FROM pos ] [ LIMIT [offset,] row_count ]
```

バイナリログ内のイベントを表示します。'log_name' を指定しなければ、最初のバイナリログが表示されます。

このステートメントは MySQL 4.0 から利用可能です。

4.11.7.5. SHOW MASTER STATUS

```
SHOW MASTER STATUS
```

マスタのバイナリログのステータス情報を提供します。

4.11.7.6. SHOW MASTER LOGS

```
SHOW MASTER LOGS
```

マスタのバイナリログを一覧表示します。このコマンドは PURGE MASTER LOGS を使用する前に実行して、どれだけ削除すべきか調べます。

4.11.7.7. SHOW SLAVE HOSTS

```
SHOW SLAVE HOSTS
```

マスタに現在登録されているスレーブの一覧を表示します。 注意: --report-host=slave_name オプションで起動していないスレーブは、この一覧には表示されません。

4.11.8. スレーブサーバを制御する SQL ステートメント

レプリケーションは SQL インタフェースで制御できます。このセクションでは、スレーブレプリケーションサーバを管理するためのステートメントについて説明します。 マスタサーバを管理するステートメントについては、項4.11.7.「マスタサーバを制御する SQL ステートメント」 で説明しています。

4.11.8.1. CHANGE MASTER TO

```
CHANGE MASTER TO master_def [, master_def] ...

master_def =
    MASTER_HOST = 'host_name'
  | MASTER_USER = 'user_name'
  | MASTER_PASSWORD = 'password'
  | MASTER_PORT = port_num
  | MASTER_CONNECT_RETRY = count
  | MASTER_LOG_FILE = 'master_log_name'
  | MASTER_LOG_POS = master_log_pos
  | RELAY_LOG_FILE = 'relay_log_name'
  | RELAY_LOG_POS = relay_log_pos
  | MASTER_SSL = {0|1}
  | MASTER_SSL_CA = 'ca_file_name'
  | MASTER_SSL_CAPATH = 'ca_directory_name'
  | MASTER_SSL_CERT = 'cert_file_name'
  | MASTER_SSL_KEY = 'key_file_name'
  | MASTER_SSL_CIPHER = 'cipher_list'
```

スレーブサーバがマスタサーバとの接続および通信のために使用しているパラメータを変更します。 以下、可能な master_def の値を示します。

リレーログオプション (RELAY_LOG_FILE と RELAY_LOG_POS) は、MySQL 4.0 から利用可能です。

SSL オプション (MASTER_SSL、MASTER_SSL_CA、MASTER_SSL_CAPATH、MASTER_SSL_CERT、MASTER_SSL_KEY、および MASTER_SSL_CIPHER) は、MySQL 4.1.1 から利用可能です。これらのオプションは、SSL サポートなしでコンパイルされているスレーブでも変更できます。これは master.info ファイルに保存されますが、SSL サポートが有効になっているサーバを使用するまでは無視されます。

例:

```
mysql> CHANGE MASTER TO
    ->    MASTER_HOST='master2.mycompany.com',
    ->    MASTER_USER='replication',
    ->    MASTER_PASSWORD='bigs3cret',
    ->    MASTER_PORT=3306,
    ->    MASTER_LOG_FILE='master2-bin.001',
    ->    MASTER_LOG_POS=4,
    ->    MASTER_CONNECT_RETRY=10;
mysql> CHANGE MASTER TO
    ->    RELAY_LOG_FILE='slave-relay-bin.006',
    ->    RELAY_LOG_POS=4025;
```

MASTER_USER、MASTER_PASSWORD、MASTER_SSL、MASTER_SSL_CA、MASTER_SSL_CAPATH、MASTER_SSL_CERT、MASTER_SSL_KEY、および MASTER_SSL_CIPHER は、スレーブがマスタに接続するために必要な情報です。これらの情報を指定しなければ、以前の値がそのまま使用されます。たとえば、MySQL マスタへの接続パ

スワードが変更された場合は、以下を実行し、

```
mysql> STOP SLAVE; -- if replication was running
mysql> CHANGE MASTER TO MASTER_PASSWORD='new3cret';
mysql> START SLAVE; -- if you want to restart replication
```

スレーブに対して新しいパスワードを指定します。変更しない情報（ホスト、ポート、ユーザなど）については指定する必要がありません。

MASTER_HOST と MASTER_PORT は、マスタホストのホスト名または IP アドレスと、その TCP ポートです。注意: MASTER_HOST が localhost と同じ場合、MySQL の他の部分と同様、ポートは無視されることがあります（Unix ソケットを利用できる場合など）。

MASTER_HOST または MASTER_PORT を指定した場合、スレーブは、マスタサーバが以前とは異なると想定します（ホストまたはポートの値を以前と同じに指定した場合でも）。この場合、マスタバイナリログの名前と位置の古い値は適用不可と判断されるため、コマンドで MASTER_LOG_FILE と MASTER_LOG_POS を指定しなければ、MASTER_LOG_FILE='' と MASTER_LOG_POS=4 が自動的に追加されます。

MASTER_LOG_FILE と MASTER_LOG_POS は、スレーブの I/O スレッドが次回開始したときにマスタから読み取りを開始する場所の座標です。これらのいずれかを指定した場合、RELAY_LOG_FILE または RELAY_LOG_POS を指定することはできません。 MASTER_LOG_FILE と MASTER_LOG_POS のいずれも指定しなければ、CHANGE MASTER 発行前のスレーブ SQL スレッドの最後の座標が使用されます。これにより、スレーブ SQL スレッドがスレーブ I/O スレッドより遅い場合でも、たとえばパスワードだけを変更するときなど、レプリケーションに途切れが生じません。この安全な動作は、MySQL 4.0.17 から 4.1.1 で導入されました。これらのバージョンの前は、CHANGE MASTER 発行前のスレーブ I/O スレッドの最後の座標が使用されていました。そのため、SQL スレッドがマスタからのイベントを失い、レプリケーションが破綻することがありました。

CHANGE MASTER はすべてのリレーログを削除します（新規リレーログを開始）。ただし、RELAY_LOG_FILE または RELAY_LOG_POS を指定した場合はこれを実行しません。リレーログは保持されます（MySQL 4.1.1 から、RELAY_LOG_PURGE グローバル変数はサイレントで 0 に設定されます）。CHANGE MASTER TO は、master.info と relay-log.info を更新します。

CHANGE MASTER は、マスタのスナップショットがあり、そのスナップショットが対応するマスタのログとオフセットを記録しているときのスレーブのセットアップに便利です。スナップショットのリストア後、スレーブで CHANGE MASTER TO MASTER_LOG_FILE='log_name_on_master', MASTER_LOG_POS=log_offset_on_master を実行できます。

上記の最初の例（CHANGE MASTER TO MASTER_HOST='master2.mycompany.com' etc）は、マスタとマスタのバイナリログの座標を変更します。これは、スレーブにマスタのレプリケートを実行させる場合です。 2 番目の例は、使用頻度は低いですが、何らかの理由でスレーブのリレーログを再度実行する場合です。この際、マスタに接続する必要はなく、CHANGE MASTER TO を実行して SQL スレッドを開始するだけです（START SLAVE SQL_THREAD）。 これは、レプリケーション以外で、スタンドアロンで非スレーブのサーバのクラッシュ後のリカバリにも使用できます。 サーバがクラッシュし、バックアップをリストアしたとします。 サーバのバイナリログ（リレーログではなく通常のバイナリログ）を再度実行します。このログの名前を myhost-bin.* と仮定します。万が一、以下の手順に忠実に従わず、サーバにバイナリログをパージさせてしまった事態に備えて、まず、これらのバイナリログのバックアップコピーを安全な場所に作成しておきます。 MySQL 4.1.1 以降を使用している場合、さらに安全対策として SET GLOBAL RELAY_LOG_PURGE=0 を実行します。 次に、log-bin なしで、新しい（以前とは別の）サーバ ID、relay-log=myhost-bin（サーバに、通常のバイナリログをリレーログと思い込ませるため）、および skip-slave-start でサーバを起動し、以下のステートメントを発行します。

```
mysql> CHANGE MASTER TO
    ->     RELAY_LOG_FILE='myhost-bin.153',
    ->     RELAY_LOG_POS=410,
```

```
  ->    MASTER_HOST='some_dummy_string';
mysql> START SLAVE SQL_THREAD;
```

サーバは自らのバイナリログを読み取って実行し、クラッシュリカバリを達成します。リカバリが完了したら、STOP SLAVE を実行してサーバをシャットダウンし、master.info と relay-log.info を削除して、元のオプションでサーバを再起動します。 現在のところ、サーバにスレーブであると思い込ませるために、MASTER_HOST の指定 (ダミー値でも) は必須です。また、以前とは異なるサーバ ID の指定も必要です。そうしないと、サーバは自分の ID の付いたイベントを見て循環レプリケーションであると勘違いし、そのイベントをスキップしてしまいます。将来、これらに対処するオプションを追加する予定です。

4.11.8.2. LOAD DATA FROM MASTER

```
LOAD DATA FROM MASTER
```

マスタのスナップショットを撮り、スレーブにコピーします。 スレーブが正しい位置からレプリケーションを開始できるように、MASTER_LOG_FILE と MASTER_LOG_POS の値を更新します。replicate-* オプションで指定したテーブルとデータベースの除外ルールも適用されます。

このステートメントの使用には以下の条件があります。

- これは、MyISAM テーブルに対してのみ有効。
- スナップショットの作成中、グローバル読み取りロックがかかるので、ロード中のマスタでの更新ができなくなる。

将来、このステートメントを InnoDB テーブルでも使用できるようにし、また非ブロックのオンラインバックアップ機能によってグローバル読み取りロックの必要性もなくす予定です。

大きなテーブルをロードするとき、場合によっては、マスタとスレーブの両方で net_read_timeout と net_write_timeout の値を大きくすることが必要になります。項4.6.8.4.「SHOW VARIABLES」 を参照してください。

注意: LOAD DATA FROM MASTER は、mysql データベースのテーブルをコピーしません。これは、マスタとスレーブに対して複数の異なるユーザと権限を設定しやすくするための措置です。

このステートメントでは、マスタへの接続に使用する MySQL ユーザが、マスタに対する RELOAD 権限と SUPER 権限、およびロードするすべてのマスタテーブルに対する SELECT 権限を持っていることが前提となります。ユーザに SELECT 権限のないマスタのテーブルはすべて、LOAD DATA FROM MASTER では無視されます。マスタは、それらのテーブルをそのユーザには表示しないためです。LOAD DATA FROM MASTER は SHOW DATABASES を呼び出してどのマスタデータベースをロードするか調べますが、SHOW DATABASES はユーザに権限のあるデータベースだけを返します。項4.6.8.1.「データベース、テーブル、カラム、およびインデックスに関する情報の取得」 を参照してください。 スレーブ側では、LOAD DATA FROM MASTER を発行するユーザに、該当するデータベースとテーブルを破棄および作成できる権限が必要です。

4.11.8.3. LOAD TABLE tbl_name FROM MASTER

```
LOAD TABLE tbl_name FROM MASTER
```

マスタからスレーブにテーブルのコピーをダウンロードします。このステートメントは主に LOAD DATA FROM MASTER をデバッグするために導入されました。 マスタサーバへの接続に使用する MySQL ユーザに、マスタに対する RELOAD 権限と SUPER 権限、およびロードするマスタテーブルに対する SELECT 権限が必要です。 スレーブ側では、LOAD

TABLE FROM MASTER を発行するユーザに、該当するテーブルを破棄および作成する権限が必要です。 前述の LOAD DATA FROM MASTER 項目のタイムアウトに関する説明を参照してください。ここでも適用されます。また、LOAD DATA FROM MASTER の制約がここでも当てはまるので、説明を参照してください (たとえば、LOAD TABLE FROM MASTER は MyISAM テーブルでのみ使用できます) 。

4.11.8.4. MASTER_POS_WAIT()

```
SELECT MASTER_POS_WAIT('master_log_file', master_log_pos)
```

これは関数であり、コマンドではありません。これは、マスタバイナリログの指定位置までスレーブが達している (読み取りと実行を終了している) ことを確認するために使用します。詳細については、項6.3.6.2.「その他の各種関数」 を参照してください。

4.11.8.5. RESET SLAVE

```
RESET SLAVE
```

スレーブのマスタバイナリログでのレプリケーション位置を、スレーブに忘れさせます。 このステートメントは、クリーンな開始のために使用します。これにより、master.info ファイルと relay-log.info ファイル、およびすべてのリレーログが削除され、新しいリレーログが開始されます。注意: スレーブの SQL スレッドによって完全に実行されていない場合でも、すべてのリレーログが削除されます (負荷の高いレプリケーションサーバや、STOP SLAVE ステートメントを発行している場合にこの状態が発生しやすくなります) 。master.info ファイルに保存されている接続情報は、対応するスタートアップオプションで値が指定されていれば、すぐにその値にリセットされます。 この情報には、マスタホスト、マスタポート、マスタユーザ、およびマスタパスワードなどの値が含まれます。 スレーブ SQL スレッドが停止したときに、テンポラリテーブルのレプリケート中であった場合、RESET SLAVE が発行され、レプリケートされたテンポラリテーブルはスレーブから削除されます。

このステートメントは MySQL 3.23.26 より前は FLUSH SLAVE という名前でした。

4.11.8.6. SET GLOBAL SQL_SLAVE_SKIP_COUNTER

```
SET GLOBAL SQL_SLAVE_SKIP_COUNTER = n
```

マスタの次の n イベントをスキップします。これは、ステートメントによるレプリケーション中断をリカバリするのに役立ちます。

このステートメントは、スレーブスレッドが実行していない場合にのみ有効です。 実行中にはエラーになります。

MySQL 4.0 より前のバージョンを使用している場合には、ステートメントから GLOBAL キーワードを除外してください。

4.11.8.7. SHOW SLAVE STATUS

```
SHOW SLAVE STATUS
```

スレーブスレッドの重要なパラメータのステータス情報を提供します。mysql クライアントを使用してこのステートメントを発行する場合、セミコロンの代わりに \G ステートメント終端記号を使用して、読みやすい縦方向のレイアウトにできます。

```
mysql> SHOW SLAVE STATUS\G
```

```
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: localhost
                  Master_User: root
                  Master_Port: 3306
                Connect_Retry: 3
              Master_Log_File: gbichot-bin.005
          Read_Master_Log_Pos: 79
               Relay_Log_File: gbichot-relay-bin.005
                Relay_Log_Pos: 548
        Relay_Master_Log_File: gbichot-bin.005
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 79
              Relay_Log_Space: 552
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 8
```

MySQL のバージョンによっては、ここで示したフィールドがすべて表示されない場合もあります。特に、MySQL 4.1.1 にのみ存在するフィールドもあります。

SHOW SLAVE STATUS で表示されるフィールドには、以下の意味があります。

- Slave_IO_State

  スレーブ I/O スレッドの SHOW PROCESSLIST 出力の State カラムのコピー。このスレッドがマスタに接続しようとしているのか、マスタからのイベントを待っているのか、マスタに再接続しているのか、などを示す。可能なステータスについては、項4.11.2.「レプリケーション実装の概要」 を参照のこと。スレッドは実行中だが、マスタとの接続がうまくいかないなどの状況では、このカラムの確認が必要となる。接続で問題がある場合に、それを示すことができるのはこのカラムだけである。反対に、SQL スレッドの場合、シンプルであるため、そのステータスはコピーされない。実行中であれば、何も問題はない。SQL スレッドが実行していなければ、Last_Error カラム（後述）にエラーが表示される。

  このフィールドは MySQL 4.1.1 で導入された。

- Master_Host

  現在のマスタホスト。

- Master_User

マスタへの接続に使用する現在のユーザ。

- Master_Port

  現在のマスタポート。

- Connect_Retry

  master-connect-retry の現在値。

- Master_Log_File

  I/O スレッドが現在読み取り中のマスタのバイナリログファイル名。

- Read_Master_Log_Pos

  I/O スレッドがマスタのバイナリログでそれまでに読み取りを完了した位置。

- Relay_Log_File

  SQL スレッドが現在読み取りおよび実行中のリレーログファイル名。

- Relay_Log_Pos

  SQL スレッドがリレーログ内でそれまでに読み取りおよび実行を完了した位置。

- Relay_Master_Log_File

  SQL スレッドによって実行された最後のイベントを含むマスタのバイナリログファイル名。

- Slave_IO_Running

  I/O スレッドが開始されたかどうかを示す。

- Slave_SQL_Running

  SQL スレッドが開始されたかどうかを示す。

- Replicate_Do_DB, Replicate_Ignore_DB

  --replicate-do-db オプションおよび --replicate-ignore-db オプションでデータベースが指定されていれば、その一覧。

- Replicate_Do_Table, Replicate_Ignore_Table, Replicate_Wild_Do_Table, Replicate_Wild_Ignore_Table

  --replicate-do-table、--replicate-ignore-table、--replicate-wild-do-table、--replicate-wild-ignore_table の各オプションでテーブルが指定されていれば、その一覧。

  これらのフィールドは、MySQL 4.1.1 で導入された。

- Last_Errno

  最後に実行したクエリが返したエラー番号。 0 の値は ``エラーがない" ことを意味する。

- Last_Error

最後に実行したクエリが返したエラーメッセージ。 例:

```
Last_Errno: 1051
Last_Error: error 'Unknown table 'z'' on query 'drop table z'
```

このメッセージは、マスタにあったテーブル z が破棄されているが、スレーブには存在しないため、DROP TABLE を実行できなかったということを示す。これは、レプリケーションのセットアップ時にテーブルをスレーブにコピーし忘れた場合に発生する。

空白文字列は ``エラーがない'' ことを意味する。 Last_Error 値が空白でない場合、スレーブのエラーログにもメッセージが表示される。

- Skip_Counter

  SQL_SLAVE_SKIP_COUNTER で最後に使用された値。

- Exec_Master_Log_Pos

  マスタのバイナリログ (Relay_Master_Log_File) 内の、SQL スレッドによって実行された最後のイベントの位置。マスタのバイナリログの (Relay_Master_Log_File,Exec_Master_Log_Pos) は、リレーログの (Relay_Log_File,Relay_Log_Pos) に対応する。

- Relay_Log_Space

  既存リレーログすべての合計サイズ。

- Until_Condition, Until_Log_File, Until_Log_Pos

  START SLAVE ステートメントの UNTIL 節で指定された値。

  Until_Condition には以下の値がある。

  - None UNTIL 節が指定されていない場合。
  - Master スレーブが、マスタのバイナリログの指定位置までを読み取り中である場合。
  - Relay スレーブが、そのリレーログの指定位置までを読み取り中である場合。

  Until_Log_File と Until_Log_Pos は、SQL スレッドが実行を中止するポイントを定義するログファイル名と位置を示す。

  これらのフィールドは、MySQL 4.1.1 で導入された。

- Master_SSL_Allowed, Master_SSL_CA_File, Master_SSL_CA_Path, Master_SSL_Cert, Master_SSL_Cipher, Master_SSL_Key

  これらのフィールドは、スレーブがマスタに接続するために使用する SSL パラメータがある場合、それを示す。

  Master_SSL_Allowed には以下の値がある。

- Yes マスタへの SSL 接続が許可されている場合。
- No マスタへの SSL 接続が許可されていない場合。
- Ignored SSL サポートが有効化されていないスレーブによって SSL 接続が許可されている場合。

他のフィールドの値は、--master-ca、--master-capath、--master-cert、--master-cipher、--master-key の各オプションの値に対応する。

これらのフィールドは、MySQL 4.1.1 で導入された。

- Seconds_Behind_Master

スレーブ SQL スレッドによって実行された最後のマスタイベントのタイムスタンプから経過した秒数。まだイベントがまったく発生していない場合、あるいは CHANGE MASTER および RESET SLAVE からイベントが発生していない場合は NULL 値となる。このカラムにより、スレーブがどれだけ ``遅い" かを知ることができる。マスタとスレーブで同一の時計を使用していなくても問題ない。

このフィールドは MySQL 4.1.1 で導入された。

4.11.8.8. START SLAVE

```
START SLAVE [thread_name [, thread_name] ... ]
START SLAVE [SQL_THREAD] UNTIL
   MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos
START SLAVE [SQL_THREAD] UNTIL
   RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos

thread_name = IO_THREAD | SQL_THREAD
```

START SLAVE はオプションなしで、両方のスレーブスレッドを開始します。I/O スレッドは、マスタサーバからクエリを読み取り、リレーログに保存します。SQL スレッドはリレーログからクエリを読み取り、実行します。 注意: START SLAVE がスレーブスレッドの開始に成功すると、エラーなしで戻ります。ただし、その場合でも、スレーブスレッドが開始した後に停止することがあります (マスタへの接続に失敗したり、バイナリログの読み取りに失敗した場合など) 。START SLAVE は、そのような場合に警告を出しません。エラーログ内で、スレーブスレッドによって生成されるエラーメッセージをチェックするか、SHOW SLAVE STATUS を実行してスレッドが適切に実行しているかどうかをチェックする必要があります。

MySQL 4.0.2 で、ステートメントに IO_THREAD オプションまたは SQL_THREAD オプションを追加して、どちらのスレッドを開始するか指定できるようになりました。

MySQL 4.1.1 以降、UNTIL 節を追加して、マスタバイナリログまたはスレーブリレーログの指定ポイントに SQL スレッドが到達するまでにスレーブを開始するよう指定できます。SQL スレッドがそのポイントに達すると、停止します。ステートメントで SQL_THREAD オプションが指定されていれば、SQL スレッドだけを開始します。そうでない場合は、両方のスレッドを開始します。SQL スレッドがすでに実行中の場合、UNTIL 節は無視され、警告が出力されます。

UNTIL 節では、ログファイル名と位置の両方を指定する必要があります。マスタオプションとリレーログオプションを混同しないでください。

UNTIL 条件は、後続の STOP SLAVE ステートメント、または UNTIL 節を含まない START SLAVE ステートメント、ま

たはサーバの再起動でリセットされます。

UNTIL 節は、レプリケーションのデバッグや、特定のステートメントをスレーブにレプリケートさせたくない場合、その直前までレプリケートするときに使用します。たとえば、マスタで DROP TABLE ステートメントを実行しているが、これが賢明ではなかった場合、UNTIL を使用してその直前までをレプリケートするようスレーブに指示できます。 イベントを調べるには、マスタログまたはリレーログで mysqlbinlog を使用するか、SHOW BINLOG EVENTS ステートメントを使用します。

UNTIL を使用してスレーブにセクションごとのクエリのレプリケーションを実行させる場合は、--skip-slave-start オプションでスレーブを起動し、スレーブの起動と同時に SQL スレッドを開始しないようにすることを推奨します。このオプションは、予期しないサーバの再起動に備えて、コマンドラインではなくオプションファイルで使用するのが良いでしょう。

SHOW SLAVE STATUS ステートメントには、UNTIL 条件の現在の値を表示する出力フィールドが含まれます。

このコマンドは、MySQL 4.0.5 より前は SLAVE START と呼ばれていました。 しばらくは、SLAVE START も下位互換性のために認められますが、将来は廃止されます。

4.11.8.9. STOP SLAVE

```
STOP SLAVE [thread_name [, thread_name] ... ]

thread_name = IO_THREAD | SQL_THREAD
```

スレーブスレッドを停止します。 START SLAVE と同様、このステートメントは IO_THREAD オプションと SQL_THREAD オプションを使用して、どちらのスレッドを停止するか指定できます。

このコマンドは、MySQL 4.0.5 より前は SLAVE STOP と呼ばれていました。 しばらくは、SLAVE STOP も下位互換性のために認められますが、将来は廃止されます。

4.11.9. レプリケーション FAQ

Q:マスタがすでに実行中でそれを停止したくない場合、どのようにスレーブをセットアップしますか。

A:いくつかの方法があります。マスタのバックアップを作成してあり、スナップショットに対応するバイナリログ名とオフセットを記録 (SHOW MASTER STATUS の出力から) してある場合は、以下を実行します。

1. スレーブに一意のサーバ ID が割り当てられていることを確認する。
2. 各パラメータに適切な値を割り当て、以下のステートメントをスレーブで実行する。

```
mysql> CHANGE MASTER TO
    ->     MASTER_HOST='master_host-name',
    ->     MASTER_USER='master_user_name',
    ->     MASTER_PASSWORD='master_pass',
    ->     MASTER_LOG_FILE='recorded_log_name',
    ->     MASTER_LOG_POS=recorded_log_pos;
```

3. スレーブで START SLAVE を実行する。

マスタのバックアップがない場合、以下の方法ですばやく確実にスレーブのセットアップを実行できます。

1. FLUSH TABLES WITH READ LOCK
2. gtar zcf /tmp/backup.tar.gz /var/lib/mysql（または、このバリエーション）
3. SHOW MASTER STATUS（後で必要となるため、出力を必ず記録しておく）
4. UNLOCK TABLES

代替手段として、上記のバイナリコピーの代わりに、マスタの SQL ダンプを使用することもできます。これを行うには、マスタで mysqldump --master-data を実行し、後でその SQL ダンプをスレーブで実行します。ただし、これはバイナリコピーよりも時間がかかります。

2 つの方法のどちらを使用する場合でも、その後、スナップショットがあり、ログ名およびオフセット値を記録する場合の指示に従ってください。同じスナップショットを使用して複数のスレーブをセットアップできます。いったんマスタのスナップショットを作成すれば、マスタのバイナリログが損傷を受けない限り、数日後でも、場合によっては一ヶ月後でもそのスナップショットを使用してスレーブをセットアップできます。理論的には、待機できる時間は無限です。現実的な制約としては、マスタのディスク空き容量が古いログによって減っていくことと、スレーブがマスタに追いつくのに時間がかかることが挙げられます。

LOAD DATA FROM MASTER を使用することもできます。これはスナップショットを撮り、スレーブにそれをリストアして、スレーブ上のログ名とオフセットを調整することを、全部同時に実行できる便利なコマンドです。将来は、LOAD DATA FROM MASTER がスレーブセットアップの推奨方法となります。ただし、このコマンドを使用した場合、読み取りロックが長時間掛かる可能性があります。このコマンドはまだ理想とする効率性では実装されていません。大きなテーブルがある場合、現時点での推奨方法は、FLUSH TABLES WITH READ LOCK を実行後のローカル tar スナップショットです。

Q: スレーブは常にマスタに接続しておく必要がありますか。

A: いいえ、その必要はありません。スレーブは何時間でも、あるいは何日間でもシャットダウンしておいたり、非接続にしておいても、再接続してマスタの更新に追いつくことができます。たとえば、マスタとスレーブの関係をダイヤルアップリンクでセットアップし、リンクアップを散発的かつ短時間に設定できます。この場合、何か特殊な対策をとらないと、任意の時点でスレーブがマスタと同期されている保証はありません。将来、少なくとも 1 つのスレーブが同期するまでマスタをブロックするオプションを追加する予定です。

Q: スレーブがマスタと比較してどれだけ遅れているかを知るにはどうすればいいですか。つまり、スレーブによってレプリケートされた最後のクエリの日付を知る方法はありますか。

A: スレーブが 4.1.1 以降の場合は、SHOW SLAVE STATUS の Seconds_Behind_Master カラムを参照します。これより前のバージョンについては、次のようになります。スレーブ SQL スレッドが存在する場合、つまりSHOW PROCESSLIST で表示される場合（MySQL 3.23 では、スレーブスレッドが存在する場合、つまりスレーブスレッドが SHOW PROCESSLIST で表示される場合）（see 項4.11.3.「レプリケーションの実装の詳細」）、およびそのスレッドがマスタから読み取ったイベントを最低 1 回は実行している場合のみ、スレーブによってレプリケートされた最後のクエリの日付を知ることができます。実際、スレーブ SQL スレッドがマスタから読み取ったイベントを実行すると、このスレッドは自分の時間をそのイベントのタイムスタンプに修正します（これが、TIMESTAMP もレプリケートする理由です）。結果、SHOW PROCESSLIST 出力の Time カラムでスレーブ SQL スレッドに対して表示される秒数は、最後にレプリケートされたイベントのタイムスタンプとスレーブマシンの実際の時刻の差になります。これを使用して、最後にレプリケートされたイベントの日付を特定できます。注意: スレーブがマスタから切断されて 1 時間経過してから再接続した場合、SHOW PROCESSLIST の Time カラムでスレーブ SQL スレッドが 3600 の値を表示する場合があります。これは、スレーブが実行しているクエリが、タイムスタンプから 1 時間経過しているためです。

Q: スレーブが追いつくまでマスタの更新をブロックするにはどうしますか。

A: 以下の手順を実行します。

1. マスタで、以下のコマンドを実行する。

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SHOW MASTER STATUS;
```

   SHOW ステートメントの出力からログ名とオフセットを記録する。

2. スレーブで以下のコマンドを実行する。ここで、MASTER_POS_WAIT() 関数の引数であるレプリケーション座標は、前のステップで記録した値。

```
mysql> SELECT MASTER_POS_WAIT('log_name', log_offset);
```

   指定のログファイルおよびオフセットにスレーブが到達するまで、SELECT ステートメントによってブロックされる。到達した時点でスレーブはマスタと同期しており、ここここでステートメントが戻る。

3. マスタで以下のステートメントを発行し、マスタによる更新処理の再開を許可する。

```
mysql> UNLOCK TABLES;
```

Q: 二方向レプリケーションをセットアップするときに注意すべき点はありますか。

A: MySQL レプリケーションでは、現在のところ、分散（サーバ間の）更新の原子性を保証するためのマスタとスレーブ間のロッキングプロトコルをサポートしていません。つまり、クライアント A が co-master 1 に更新を行い、それを co-master 2 に伝播する前に、クライアント B が co-master 2 に更新を行って、クライアント A の更新が co-master 1 で行ったものとは変わってしまう可能性があります。この場合、co-master 2 からの更新がすべて伝播した後であっても、クライアント A の更新が co-master 2 に伝わったとき、co-master 1 とは異なるテーブルが生成されます。このため、どのような順序でも更新が安全に行われるという確証がある場合、またはクライアントコードで更新順序の不正に対処できる場合以外は、二方向レプリケーションで 2 つのサーバをチェーン状に設定しないでください。

また更新については、二方向レプリケーションはそれほどパフォーマンス向上に役立たないことも認識してください。サーバは両方ともそれぞれ、1 つのサーバのときと同じ量の更新を行います。違いは、別のサーバから発生した更新が 1 つのスレーブスレッドでシリアル化されるため、ロックの競合が少なくなることぐらいです。その利点も、ネットワーク遅延によって相殺されてしまいます。

Q: レプリケーションを使用してシステムのパフォーマンスを改善する方法はありますか。

A: 1 つのサーバをマスタとしてセットアップし、すべての書き込みをそれにダイレクトします。そして予算とスペースが許容する限り多くのスレーブをセットアップし、マスタと複数のスレーブで読み取りを分散します。--skip-bdb、--low-priority-updates、および --delay-key-write=ALL でスレーブを起動すると、スレーブの速度が向上します。この場合、スレーブは速度を上げるため、BDB テーブルの代わりに非トランザクションの MyISAM テーブルを使用します。

Q: パフォーマンス改善レプリケーションを使用するアプリケーションを作成したいのですが、クライアントコードはどうすればいいですか。

A: コード内のデータベースアクセスを担当する部分が適切に抽象化つまりモジュール化されていれば、レプリケートのセ

ットアップで実行できるように変換するのはスムーズかつ簡単です。データベースアクセスの実装部分を変更し、すべての書き込みをマスタに、そして読み取りをマスタかスレーブに送信するようにします。ご使用のコードにこのレベルの抽象化がなされていなければ、レプリケーションシステムを 1 からセットアップします。まず以下の関数で、ラッパライブラリまたはモジュールを作成します。

- safe_writer_connect()
- safe_reader_connect()
- safe_reader_query()
- safe_writer_query()

各関数名の safe_ は、関数がすべてのエラー条件を処理することを意味します。もちろん、関数には別の名前を使用することも可能です。重要なことは、読み取りのための接続、書き込みのための接続、読み取り実行、および書き込み実行で統合インタフェースを持つことです。

次に、ラッパライブラリを使用するようにクライアントコードを変換します。このプロセスは最初は苦痛かもしれませんが、長期的に見ると損はありません。先ほど説明したアプローチを使用するアプリケーションはすべて、マスタ/スレーブ構成の利点を活用できます。これは、スレーブが複数の場合でも同様です。コードは保守が非常に簡単で、トラブルシューティングオプションの追加も手間がかかりません。1 つか 2 つの関数を修正するだけで、各クエリにかかった時間をログしたり、数千のクエリの中からどのクエリがエラーの原因になったかを特定することなどができます。

コード作成の経験が豊富であれば、replace ユーティリティを使用して変換タスクを自動化することもできます。このユーティリティは MySQL の標準ディストリビューションに含まれています。また、独自の Perl スクリプトを作成することもできます。 コードが認識可能なパターンに従っていれば理想的です。そうでない場合は、書き換えるのが良いでしょう。あるいは少なくとも、1 つのパターンになっているようにしてください。

Q: MySQL レプリケーションによって、どのような場合にどれだけシステムのパフォーマンスを改善できますか。

A: MySQL レプリケーションは、読み取り頻度が高く、書き込み頻度が低いシステムで最も威力を発揮します。単一マスタと複数スレーブのセットアップを使用することにより、理論的にはネットワーク帯域幅の限界またはマスタが処理できる更新負荷の限界までスレーブを追加することにより、システムを拡張できます。

追加する利点がなくなるまでいくつのスレーブを追加できるか、またサイトのパフォーマンスをどれだけ改善できるかを判断するには、クエリパターンを知り、実証的に（ベンチマークを使用して）通常のマスタと通常のスレーブの間の読み取り（毎秒ごとの読み取り、または max_reads）と書き込みスループットの関係を調べる必要があります。ここでは、例として、仮想システムのレプリケーションでの単純化した計算を示します。

システム負荷が 10% の書き込みと 90% の読み取りで構成され、max_reads が 1200 - 2 * max_writes であると仮定します。 つまり、書き込みがなければシステムは毎秒 1200 の読み取りを実行できます。書き込みの平均は読み取り平均の 2 倍かかります。そしてその関係は直線的です。 マスタと各スレーブの能力は同じで、1 マスタと N スレーブがあると想定します。各サーバ（マスタまたはスレーブ）は以下のようになります。

reads = 1200 - 2 * writes（ベンチマークから）

reads = 9* writes / (N + 1)（読み取りは分散されるが、書き込みはすべてのサーバで行われる）

9*writes/(N+1) + 2 * writes = 1200

writes = 1200/(2 + 9/(N+1)

この分析により、以下の結論を導きだせます。

- N = 0 (レプリケーションがないことを意味する) の場合、システムは 1200/11、つまり毎秒 109 書き込みを処理できる (アプリケーションの性質上、読み取りが 9 倍になる)。
- N = 1 の場合、毎秒 184 の書き込みまで処理できる。
- N = 8 の場合、毎秒 400 の書き込みまで処理できる。
- N = 17 であれば、480 の書き込みとなる。
- N が無限に近づくと (予算も無限大に膨らむが)、毎秒 600 の書き込みに近くなり、システムスループットは 5.5 倍になる。しかし、8 サーバだけで 4 倍近くになっている。

注意: この計算ではシステム帯域を無限として想定しており、現実の場面では重要となるかもしれない要因をいくつか無視しています。多くの場合、N アプリケーションスレーブを追加した場合の結果を正確に予測する上記のような計算を行うのは難しいと言えます。しかし、以下の質問から、レプリケーションによってシステムのパフォーマンスが改善されるかどうか、またどの程度改善されるか、ある程度判断できるでしょう。

- システムの読み取りと書き込みの比率はどれくらいか。
- 読み取りを減らした場合、1 つのサーバで処理できる書き込み負荷をどの程度増やせるか。
- ネットワークの帯域幅を使用できるスレーブ数の上限はいくつか。

Q: 冗長性と高可用性を実現するようにレプリケーションを使用するにはどうすればよいですか。

A: 現在使用可能な機能で、1 つのマスタと 1 つのスレーブ (または複数のスレーブ) をセットアップする必要があります。そして、マスタの稼働状態を監視し、失敗時にアプリケーションとスレーブにマスタの変更を指示するスクリプトを作成します。以下のガイドラインを参考にしてください。

- スレーブにマスタを変更するよう指示するには、CHANGE MASTER TO コマンドを使用する。
- アプリケーションにマスタの場所を認識させておくためには、マスタに動的 DNS エントリを採用すると良い。bind で、nsupdate を使用して DNS を動的に更新できる。
- --log-bin オプションを指定し、--log-slave-updates なしでスレーブを実行する。STOP SLAVE および RESET MASTER を実行し、他のスレーブで CHANGE MASTER TO を実行することにより、スレーブがマスタになる準備ができる。たとえば、以下のセットアップがあると仮定する。`M" はマスタ、``S" はスレーブ、``WC" はデータベース読み取りおよび書き込みを発行するクライアントを意味する。データベース読み取りだけを発行するクライアントは、切り替えの必要がないため、ここでは考慮されていない。

```
    WC
     \
      v
WC----> M
    / | \
   /  |  \
```

```
 v  v  v
S1  S2 S3
```

S1 (S2 および S3 と同様) は、--log-bin を使用し、--log-slave-updates なしで実行しているスレーブである。S1 で実行される書き込みは M からのレプリケーションのみなので、S1 のバイナリログは空白である (S1 は --log-slave-updates なしで実行されている)。 何らかの理由で M が利用できなくなったため、S1 を新しいマスタにする (すべての WC を S1 にダイレクトし、S2 と S3 が S1 をレプリケートするように設定する)。

すべてのスレーブで、自身のリレーログにあるすべてのクエリを処理したことを確認する。各スレーブで STOP SLAVE IO_THREAD を発行し、SHOW PROCESSLIST の出力で Has read all relay log が表示されることを確認する。すべてのスレーブでこれを確認したら、新しい環境に設定できる。すべてのスレーブで STOP SLAVE を発行し、マスタに昇格するスレーブに RESET MASTER を、他のスレーブには CHANGE MASTER を実行する。

これで、M にアクセスする WC はいなくなる。すべての WC に、クエリを S1 に送信するよう指示する。これ以降、WC によって S1 に送信されたクエリはすべて、S1 のバイナリログに書き込まれる。S1 のバイナリログには、M が終了した時点以降に S1 に送信されたクエリがすべて正確に含まれる。 S2 (および S3) で、STOP SLAVE、CHANGE MASTER TO MASTER_HOST='S1' を実行する ('S1' には S1 の実際のホスト名が入る)。CHANGE MASTER で、S2 または S3 から S1 に接続するための全情報 (ユーザ、パスワード、ポート) を追加する。CHANGE MASTER で、S1 のバイナリログの名前や位置を指定する必要はない。それが最初のバイナリログであり、位置が 4 であることがわかっており、これらの値は CHANGE MASTER のデフォルトとなっているからである。最後に、S2 と S3 で START SLAVE を実行すると、以下のようになる。

```
    WC
   /
   |
WC  | M(unavailable)
\  |
 \ |
  v v
  S1<--S2  S3
   ^     |
   +-------+
```

M が再稼動したら、S2 および S3 と同じように CHANGE MASTER を実行するだけで済む。そうすると M は S1 のスレーブとなり、ダウンしてから実行された WC 書き込みをすべてピックアップする。M をマスタに戻す (それが一番強力なマシンである、という理由などで) には、S1 と M の立場を逆にして前回の手順を繰り返す。このとき、S1、S2、S3 を M のスレーブにする前に、M で RESET MASTER を実行することを忘れてはいけない。そうしないと、M が非稼動になる前の古い WC 書き込みも拾ってしまう。

現在、自動マスタ選択システムを MySQL に統合する方向で進んでいますが、それが実現するまでは、自分で監視ツールを作成してください。

4.11.10. レプリケーションのトラブルシューティング

指示に従って設定したにもかかわらず、レプリケーションセットアップが機能しない場合は、まず、以下の項目をチェックしてください。

- エラーログのメッセージをチェックする。多くのユーザがこれを最初に実行せずに多くの時間を浪費している。

- マスタがバイナリログに記録しているかどうか。SHOW MASTER STATUS でチェックする。記録していれば、Position はゼロ以外の値になっている。そうでない場合、マスタに log-bin オプションを設定していること、および server-id を設定していることを確認する。
- スレーブが実行しているかどうか。SHOW SLAVE STATUS を実行し、Slave_IO_Running と Slave_SQL_Running の値が両方とも Yes であることをチェックする。そうでなければ、スレーブオプションを確認する。
- スレーブが実行している場合、スレーブがマスタとの接続を確立しているかどうか。SHOW PROCESSLIST を実行し、I/O スレッドと SQL スレッドを見つけ (その表示については、see 項4.11.3.「レプリケーションの実装の詳細」) 、その State カラムをチェックする。Connecting to master であれば、レプリケーションユーザのマスタでの権限、マスタホスト名、DNS 設定、マスタが実際に実行しているかどうか、マスタがスレーブからアクセス可能かどうかを調べる。
- スレーブが以前は実行していたが現在は停止している場合、その理由は通常、マスタで成功したクエリがスレーブで失敗したことにある。マスタのスナップショットを適切に作成しており、スレーブスレッド外でスレーブのデータを変更していなければ、これは発生しないはずである。発生した場合はバグと考えられるので、次のバグレポートのセクションを参照のこと。
- マスタで成功したクエリがスレーブでは実行できず、完全データベース再同期 (スレーブのデータベースを削除し、マスタの新規スナップショットをコピーする) の実行が難しい場合、以下を試す。
  - 最初に、スレーブのテーブルがマスタのテーブルと異なっていないか確認する。異なっていた場合、どのようにそれが発生したかを見極める。バグの可能性もある。http://www.mysql.com/documentation でオンライン MySQL マニュアルの変更ログを読み、それが既知のバグかどうか、またすでに解決されているものであるかどうか確認する。 次に、スレーブのテーブルをマスタのテーブルと同じなるように変更して、START SLAVE を実行する。
  - 上記のことを行ってもうまく機能しない場合、または上記のことができない場合は、必要に応じて手動で更新し、マスタからの次の更新を無視しても安全かどうかを判断する。
  - 次のクエリをスキップすることに決定した場合、以下のステートメントを発行する。

    ```
    mysql> SET GLOBAL SQL_SLAVE_SKIP_COUNTER = n;
    mysql> START SLAVE;
    ```

    クエリが AUTO_INCREMENT または LAST_INSERT_ID() を使用しない場合、n の値は 1 にする。使用する場合、値は 2 にする。AUTO_INCREMENT または LAST_INSERT_ID() を使用するクエリに 2 の値を使用する理由は、そのクエリはマスタのバイナリログでイベントを 2 つ使用するからである。
  - 最新のバージョンにアップグレードし、古いバグを確実に回避する。
  - スレーブがマスタと完璧に同期して開始し、スレーブスレッド外で誰もテーブルを更新していないと確信がある場合は、バグを報告する。

4.11.11. レプリケーションバグのレポート

ユーザエラーがないにもかかわらずレプリケーションが機能しない、または不安定であると判断したら、当社にバグレポートを送ってください。バグをトラックダウンするために、できるだけ多くの情報が必要です。ある程度の時間と労力を費やしてバグレポートを準備してください。

バグが再現可能な場合、バグデータベース (http://bugs.mysql.com/) に入力してください。ファントム問題 (意図的に再現できないもの) がある場合、以下の手順に従います。

1. ユーザエラーがないことを確認する。たとえば、スレーブをスレーブスレッド外から更新すると、データの同期が失われ、更新でユニークキー違反となる。この場合、スレーブスレッドは停止し、ユーザが手動でクリーンアップして同期を取り戻すのを待つ。これはレプリケーションの問題ではない。レプリケーションを妨害する外部干渉の問題である。
2. --log-slave-updates オプションおよび --log-bin オプションでスレーブを実行する。これにより、スレーブは受け取った更新を自身のバイナリログに記録する。
3. レプリケーションの状態をリセットする前に、すべての証拠を保存する。情報がなかったり少ない場合には、問題のトラッキングに時間がかかる。以下のような証拠を収集すること。
   - マスタからのすべてのバイナリログ
   - スレーブからのすべてのバイナリログ
   - 問題発見時のマスタでの SHOW MASTER STATUS 出力
   - 問題発見時のマスタでの SHOW SLAVE STATUS 出力
   - マスタおよびスレーブのエラーログ
4. mysqlbinlog を使用してバイナリログを調べる。たとえば以下を実行して、問題のあるクエリを見つけることができる。

   ```
   shell> mysqlbinlog -j pos_from_slave_status /path/to/log_from_slave_status | head
   ```

ファントム問題の証拠を収集したら、まず個別のテストケースに分類する努力をしてください。その後、バグデータベース (http://bugs.mysql.com/) に、その問題を、できるだけ多くの情報とともに入力します。

第5章 MySQL の最適化

最適化は、システム全体の理解が必要であるため、複雑な作業です。システムやアプリケーションに関する知識が豊富でなくても部分的なローカルの最適化は可能ですが、より高度な最適化が必要になるほど求められる知識も高度になります。

この章では、MySQL 最適化の方法説明し、その例もいくつか紹介します。ただし、常にシステムの速度をさらに上げる補足的な方法もありますが、難度も高くなることを覚えておいてください。

5.1. 最適化の概要

言うまでもなく、システムの速度を上げる際に最も重要な要素は基本設計です。また、使用するシステムの用途およびそのボトルネックを認識しておく必要もあります。

最も一般的なボトルネックは下記のとおりです。

- ディスクシーク。 ディスクが 1 つのデータを検索するには時間がかかる。1999 年の最新のディスクでは、通常これにかかる平均時間が 10 ms 未満であるため、理論的には 1 秒間に 100 のシークを実行できることになる。新しいディスクではこの時間の改善が緩やかで、1 つのテーブルの最適化が非常に困難である。これを最適化する方法として、複数のディスクにデータを分散することが挙げられる。
- ディスクの読み取りと書き込み。 ディスクが適切な位置にある場合、データの読み取りが必要になる。1999 年の最新のディスクでは、1 つのディスクで約 10 - 20 MB の読み取りが可能になる。これは、複数のディスクから並行した読み取りが可能であるため、シークに比較して最適化が容易である。
- CPU サイクル。 メインメモリにデータがある場合（またはすでにそこに存在している場合）、結果を得るためには処理が必要になる。メモリと比較してテーブルが小さい場合は、最も一般的な制限要因になる。しかし、テーブルが小さくても、一般に速度上の問題は発生しない。
- メモリ帯域幅。 CPU キャッシュの適正量より多く CPU がデータを必要とする場合、メインメモリの帯域幅がボトルネックになる。これは、ほとんどのシステムに一般的な問題ではないが、認識しておく必要がある。

5.1.1. MySQL の設計上の制約とトレードオフ

MyISAM ストレージエンジンの使用時に、MySQL では非常に高速のテーブルロック（複数リーダ/単一ライタ）が使用されます。このテーブル型の最大の問題は、同じテーブルに対して複数の UPDATE と遅い SELECT が混在する場合に発生します。テーブルでこのような問題が発生した場合は、別のテーブル型を使用してもかまいません。 See 章 7. MySQL のテーブル型。

MySQL はトランザクションテーブル、非トランザクションテーブルの両方で機能します。 非トランザクションテーブル（何らかのエラー発生した場合にロールバックができない）での動作をスムーズにするため、MySQL には次のルールがあります。

- すべてのカラムにデフォルト値がある。
- NOT NULL カラムに対して NULL などの '正しくない' 値を挿入した場合や、数値列の数値が大きすぎる場合、MySQL

では エラーを発生するのではなく、'とりうる可能な値のうちの最適値' に値を設定する。数値の場合は 0 で、可能な最小値か最大値になる。文字列の場合は、空白文字かカラムの最大長さにあわせた文字列になる。

- 計算式はすべて、エラー状態を表示するのではなく、使用可能な値を返す。たとえば、1/0 の場合は、NULL を返す。

この詳細については、See 項1.8.5.「MySQL における制約の処理」 を参照してください。

このことは、フィールド内容のチェックに MySQL を使用するのではなく、このチェックをアプリケーションで実行する必要があることを意味します。

5.1.2. 移植性

SQL サーバは SQL のさまざまな部分を実装しているので、移植可能な SQL アプリケーションの作成が可能となります。非常に単純な SELECT や INSERT は容易ですが、必要なことが増えれば増えるほど、作成が難しくなります。多数のデータベースを使用しながら素早い速度が要求されるアプリケーションの場合は、さらに難度が上がります。

複雑なアプリケーションを移植可能にするには、ともに稼動する必要のある SQL サーバ数を選択する必要があります。

たとえば、Infomix や DB2 の使用を可能にするには、18 文字を超えるカラム名は使用できません。

MySQL ベンチマークと crash-me プログラムはいずれもデータベースへの依存度が非常に低くなっています。これらのプログラムがどのように処理されているかを調べることによって、データベースに依存しないアプリケーションを作成する際に必要なことに関する感覚を得ることができます。ベンチマーク自体は、MySQL ソースディストリビューションの sql-bench ディレクトリにあります。これは Perl - DBI データベースインタフェース (問題のアクセス部分を解決する) で作成されています。

このベンチマークの結果については、http://www.mysql.com/information/benchmarks.html を参照してください。

これらの結果からもわかるように、データベースのすべてに何らかの弱点があります。言い換えると、動作の相違を招くさまざまな設計上の障害があります。

データベースの独立性の獲得を目指す場合は、SQL サーバそれぞれのボトルネックを正しく理解する必要があります。MySQL では、非常に高速にレコードの取り出しと更新が行われますが、1 つのテーブル上に低速のリーダとライタが混在することに問題があります。これとは異なり、Oracle では、更新直後のレコードがディスクに保存される前にそのレコードにアクセスしようとする際に大きな問題があります。一般にトランザクションデータベースの場合、ログテーブルからのサマリテーブルの生成時は行ロックがほとんど役に立たず、問題が生じやすくなっています。

アプリケーションを実際にデータベース非依存にするには、データ操作に使用する簡単な拡張可能インタフェースを定義する必要があります。ほとんどのシステムでは C++ が使用できるため、データベースに C++ クラスインタフェースを使用することは道理にかなっています。

あるデータベースに固有の機能を使用する場合 (MySQL の REPLACE コマンドなど) は、他の SQL サーバでその機能を実装できるようにするメソッドをコード化する必要があります (ただし低速化します)。MySQL を使用すると、/*! */ 構文を使用して MySQL 固有のキーワードをクエリに追加できます。/**/ 内のコードは、その他の SQL サーバのほとんどでコメントとして処理 (無視) されます。

一部の Web アプリケーションのように正確性よりパフォーマンスが重視される場合は、すべての結果をキャッシュするアプリケーションレイヤを作成すると、さらにパフォーマンスを改善できます。一定の期間後に古い結果を '期限切れ' することで、キャッシュを適度に最新の状態に保持できます。これにより、キャッシュを動的に拡大し、通常の状況に戻るまでタイムアウト期限を高速に設定して、高負荷のスパイクを処理するメソッドが提供されます。

この場合、テーブル作成情報にキャッシュの初期サイズと通常時にテーブルがリフレッシュされる頻度に関する情報が組み込まれます。

5.1.3. MySQL 使用実績

MySQL の初期開発当時は、最大顧客に合わせて MySQL の機能が開発されてきました。この機能は、スウェーデンの最大小売商数社向けにデータウェアハウスを処理するものです。

すべての店舗からボーナスカード取引すべてのサマリを毎週取得し、店舗の所有者が顧客に対する広告キャンペーンの効果を調べる際に役立つ情報を提供するように求められています。

このデータは非常に大量（1 か月に約 700 万のサマリ取引）で、ユーザへの提示に必要な 4-10 年間のデータを保有しています。 このデータから新しいレポートに '即時' アクセスできるようにしたいという顧客からの要求が毎週ありました。

1 か月ごとにすべての情報を圧縮 'トランザクション' テーブルに格納することでこの要求を解決しました。トランザクションテーブルからさまざまな基準（製品グループ、顧客 ID、店舗など）によって分類されたサマリテーブルを生成する単純なマクロ（スクリプト）セットを開発しています。レポートは Web ページ形式で、Web ページを解析し、SQL ステートメントを実行して、結果を挿入する、短い Perl スクリプトから動的に生成されます。PHP か mod_perl の使用のほうが適しているとも言えますが、その当時は利用できませんでした。

グラフィカルデータについては、SQL クエリの結果（この結果に処理を加えて）から GIF を生成する簡単なツールを C で作成しました。これも HTML ファイルを解析する Perl スクリプトから動的に実行されます。

ほとんどの場合、既存のスクリプトをコピーし、その SQL クエリを修正することで新規のレポートを簡単に実行することができます。状況によっては、既存のサマリテーブルにフィールドを追加したり、新規のテーブルを生成することが必要な場合もありますが、これもディスク上にすべてのトランザクションテーブルを保存しているため非常に容易なことです（現在、少なくとも 50 G のトランザクションテーブルとその他の 200 G の顧客データを保持しています）。

顧客は、ODBC によってサマリテーブルに直接アクセスすることができ、上級ユーザであれば各自でデータを処理することができます。

非常に適度な規模の Sun Ultra SPARCstation（2x200 Mhz）を使用した処理では何も問題が発生していません。最近サーバの 1 つを 2 CPU 400 Mhz UltraSPARC にアップグレードし、製品レベルでのトランザクション処理の開始を計画しています。この処理ではデータが 10 倍増加することになります。システムにディスクを追加するだけでこれに対応できると考えています。

安価に CPU 能力を増強できるように Intel-Linux でも実験を行っています。現在、バイナリの移植可能データベースフォーマット（バージョン 3.23 の新機能）があり、アプリケーションの一部への使用を開始することになっています。

当初、Linux では低から中程度の負荷でのパフォーマンスに優れ、Solaris ではディスク IO が非常に高いため高負荷を達成しようとする際のパフォーマンスに優れているという感触を得ましたが、現在のところこれに関する結論は出ていません。Linux カーネルの開発者との協議の結果、これは、Linux のバッチジョブに割り当てられるリソースが多すぎると対話的なパフォーマンスが非常に低くなる副作用の可能性もあります。これによって大量のバッチが進行中に非常に低速になり、応答不可の状態が発生します。将来の Linux カーネルではこの処理が改善されるでしょう。

5.1.4. MySQL ベンチマークスィート

このセクションでは、MySQL ベンチマークスィート（および crash-me）の技術的記述を記載する予定ですが、この記述はまだ作成されていません。現状では、MySQL ソースディストリビューションの sql-bench ディレクトリにあるコードと

結果を確認することでベンチマークに関するヒントが得られます。

このベンチマークスィートは、SQL 実装のパフォーマンスを向上または低下させる操作をユーザに示すことを目的としています。

このベンチマークはシングルスレッドであるため、実行される操作の最短時間が測定されていることに注意してください。将来はこのベンチマークスィートにマルチスレッドのテストも多数追加する予定です。

下表は、Windows NT 4.0 コンピュータ上で ODBC を介していくつかのデータベースサーバにアクセスした場合のベンチマーク結果の比較を示しています。

インデックスごとに 2,000,000 レコードの読み取り	秒	秒
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	

350,768 レコードの挿入	秒	秒
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	

最初のテストでは、8M のインデックスキャッシュサイズで MySQL が実行されました。

これ以外にもベンチマーク結果を http://www.mysql.com/information/benchmarks.html のサイトに収集しています。

Oracle は削除の依頼があったため含まれていません。Oracle のベンチマークはすべて、Oracle から提供されます。上記のベンチマークは標準のインストールが 1 クライアントに対して実行できることを示すことを想定しているため、Oracle のベンチマークは非常に偏りがあると確信しています。

ベンチマークスィートを使用するには、以下の要件を満たす必要があります。

- ベンチマークスィートは、MySQL ソースディストリビューションによって提供されるため、ソースディストリビュー

ションが必要である。http://www.mysql.com/downloads/ のサイトからリリースされているディストリビューションをダウンロードするか、現在の開発ツリー（see 項2.3.4.「開発ソースツリーからのインストール」）を使用できる。

- ベンチマークスクリプトは Perl で作成され、データベースサーバへのアクセスには Perl DBI モジュールを使用しているため、DBI のインストールが必要である。テスト対象のサーバのそれぞれにサーバ専用の DBD ドライバも必要である。 たとえば、MySQL、PostgreSQL、DB2 をテストするには、DBD::mysql、DBD::Pg、DBD::DB2 のモジュールをインストールする必要がある。

ベンチマークスィートは、MySQL ソースディストリビューションの sql-bench ディレクトリにあります。 ベンチマークテストを実行するには、ロケーションをそのディレクトリに変更し、run-all-tests スクリプトを実行します。

```
shell> cd sql-bench
shell> perl run-all-tests --server=server_name
```

server_name はサポートされるサーバの 1 つを表します。run-all-tests --help を呼び出すと、すべてのオプションとサポート対象サーバの一覧を取得できます。

crash-me では、データベースがサポートする機能と、実際のクエリを実行した場合の機能と制約の判定が試行されます。たとえば、以下についての判定が行われます。

- サポートされるカラム型
- サポートされるインデックス数
- サポートされる関数
- 使用可能なクエリのサイズ
- 使用可能な VARCHAR カラムのサイズ

多様なデータベースに関する crash-me の結果は、http://www.mysql.com/information/crash-me.php のサイトにあります。

5.1.5. 独自のベンチマークの使用

確実にアプリケーションとデータベースのベンチマークを行い、ボトルネックを検出しておく必要があります。これを修正（または、ボトルネックを ``ダミーモジュール" に置換）することによって、次のボトルネック（など）の確認が容易になります。現在のアプリケーションの総合的なパフォーマンスが許容できるものであっても、実際にパフォーマンスの強化が迫られる場合に備えて、少なくともボトルネックそれぞれに対して計画を立て解決方法を判定しておく必要があります。

移植可能なベンチマークプログラムの例として、MySQL ベンチマークスィートを取り上げます。See 項5.1.4.「MySQL ベンチマークスィート」。このスィートから任意のプログラムを選び、必要に合わせて修正することができます。これによって、それぞれの問題に対して複数の解決方法を試行して、実際に最も高速が得られるのはどれであるかについてテストすることができます。

これ以外の無料のベンチマークスィートに Open Source Database Benchmark があり、これは http://osdb.sourceforge.net/ で入手できます。

一般的には、システムの負荷が非常に高い状況にのみ問題が発生します。負荷の問題が（テスト済の）本稼動のシステム

で発生したと問い合わせてくる顧客が多数いました。ほとんどの場合、パフォーマンスに関わる問題は基本的な設計上の問題（高負荷時のテーブルスキャンの不良）かオペレーティングシステムやライブラリの問題が原因だと判明しています。たいていは、システムがまだ本稼動に入っていない場合のほうが問題の修正がはるかに容易です。

このような問題を回避するには、想定可能な最悪の負荷でアプリケーション全体のベンチマークにある程度力を注ぐ必要があります。これには Super Smack を使用できます。これは、http://www.mysql.com/Downloads/super-smack/super-smack-1.0.tar.gz で入手できます。 その名（Smack = 打ちこわし）のとおり、システムに限界まで負荷をかけることができるため、必ず開発システムでのみ使用するようにしてください。

5.2. SELECT ステートメントおよびその他のクエリの最適化

第 1 にすべてのクエリに影響を及ぼすことが 1 つあります。アクセス権システムのセットアップの複雑性が増すほど、オーバヘッドも増加します。

GRANT ステートメントを何も実行していない場合は、MySQL によってアクセス権チェックが多少最適化されます。大量の処理が必要なときは、GRANT を使用しないことで時間を節約できる場合もあります。GRANT を使用した場合は、アクセス権チェックが多くなり、オーバヘッドが増加します。

明示的な MySQL 関数に関わる問題がある場合は、常に MySQL クライアントでこの関数の計時を行うことができます。

```
mysql> SELECT BENCHMARK(1000000,1+1);
+------------------------+
| BENCHMARK(1000000,1+1) |
+------------------------+
|                      0 |
+------------------------+
1 row in set (0.32 sec)
```

これは、PentiumII 400MHz 上で MySQL によって 1,000,000 の + 式を 0.32 秒間に実行できることを示しています。MySQL 関数はすべて最適化されていますが、例外も若干あります。BENCHMARK(loop_count,expression) はクエリに関数上の問題があるかどうかを調べる際に最適のツールです。

5.2.1. EXPLAIN 構文（SELECT に関する情報の取得）

```
   EXPLAIN tbl_name
か  EXPLAIN SELECT select_options
```

EXPLAIN tbl_name は、DESCRIBE tbl_name または SHOW COLUMNS FROM tbl_name のシノニムです。

キーワード EXPLAIN を SELECT ステートメントの前に置いた場合、MySQL によってテーブルの結合状況と順序に関する情報が提供され、テーブルの SELECT の処理方法が説明されます。

EXPLAIN を利用すると、より速くレコードを検索する SELECT を得るために、どの時テーブルにインデックスを追加しなければならないかを確認できます。

最適化方法の選択に影響を及ぼすキーの、カーディナリティなどのテーブル統計を更新するために、ANALYZE TABLE を定期的に実行する必要があります。 See 項4.6.2.「ANALYZE TABLE 構文」。

また、オプティマイザが、テーブルを最適な順序で結合しているかどうかも確認することができます。 オプティマイザが特定の順番で結合を行うように強制するには、SELECT ステートメントに STRAIGHT_JOIN 節を追加します。

非単純結合の場合、EXPLAIN は SELECT ステートメントで使用される各テーブルに関する情報を返します。 テーブルは、読み取られた順序に従って一覧表示されます。 MySQL は、単一スイープ多結合メソッドを使用してすべての結合を解決します。これは、MySQL が最初のテーブルからレコードを読み取ってから、第 2 のテーブル、第 3 のテーブルといった順序で、一致するレコードの検索を行うことを意味します。 すべてのテーブルの処理が終わると、選択したカラムと、さらに一致レコードがあるテーブルが検索されるまでのテーブル一覧のバックトラックが出力されます。 次のレコードはこのテーブルから読み取られ、処理が次のテーブルから続行されます。

MySQL バージョン 4.1 では、EXPLAIN 出力が変更され、UNION ステートメント、サブクエリ、派生テーブルなどの構造での機能が改善されています。最も重要なことは、id と select_type という 2 つの新しいカラムが追加されたことです。

EXPLAIN の出力は、次のカラムで構成されます。

- id

  SELECT に割り当てられた ID。クエリ内におけるこの SELECT の順序番号。

- select_type

  SELECT 節の種類、次のいずれかが示される。

  - SIMPLE

    単純な SELECT (UNION やサブクエリを使用しない) 。

  - PRIMARY

    最外部の SELECT

  - UNION

    UNION 内の第 2 およびそれ以降の SELECT ステートメント。

  - DEPENDENT UNION

    UNION 内の第 2 およびそれ以降の SELECT ステートメント、外側のサブクエリに依存する。

  - SUBQUERY

    サブクエリ内の第 1 SELECT。

  - DEPENDENT SUBQUERY

    第 1 SELECT、外側のサブクエリに依存する。

  - DERIVED

    派生テーブル SELECT (FROM 節内のサブクエリ) 。

- table

  結果を得るために参照するテーブル。

- type

結合型。各結合型を最適なものから順に紹介する。

- system

  1 レコードのみで構成されるテーブル (= システムテーブル) 。これは、const 結合型の特殊なケースである。

- const

  テーブルに、一致するレコードが最大で 1 つあり、クエリの開始時に読み取られる。レコードが 1 つしかないため、このレコードのカラムの値はオプティマイザによって定数と見なされる。const テーブルは、1 回しか読み取られないため、非常に高速である。

  const は、PRIMARY/UNIQUE キーを定数と比較する場合に使用される。

```
SELECT * FROM const_table WHERE primary_key=1;

SELECT * FROM const_table
WHERE primary_key_part1=1 AND primary_key_part2=2;
```

- eq_ref

  前のテーブルのレコードの組み合わせのそれぞれに対して、このテーブルから 1 レコードずつ読み取られる。これは、const 型以外で最適な結合型である。結合でインデックスのすべての部分が使用され、このインデックスが UNIQUE または PRIMARY KEY である場合に使用される。

  = 演算子と比較されるインデックスの張られたカラムには、eq_ref を使用できる。比較対象のアイテムは定数でも、このテーブル以前に読み取られたテーブルのカラムを使用する式でもかまわない。

  下記の例では、ref_table で eq_ref が使用される。

```
SELECT * FROM ref_table,other_table
WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
WHERE ref_table.key_column_part1=other_table.column
AND ref_table.key_column_part2=1;
```

- ref

  前のテーブルのレコードの組み合わせのそれぞれに対して、インデックス値にマッチするすべてのレコードがこのテーブルから読み取られる。ref は、インデックスの左端の先頭部分のみが結合で使用される場合、またはインデックスが UNIQUE や PRIMARY KEY ではない場合 (すなわち、この結合において、インデックス値から1つのレコードをSELECTできない場合) に使用される。この結合型は、使用されるインデックスと一致するレコードが数レコードしかない場合に適している。

  = 演算子と比較されるインデックスが張られたカラムには、ref が使用される。

  下記の例では、ref_table で ref が示される。

```
SELECT * FROM ref_table WHERE key_column=expr;

SELECT * FROM ref_table,other_table
WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
WHERE ref_table.key_column_part1=other_table.column
AND ref_table.key_column_part2=1;
```

- ref_or_null

  ref と同様だが、NULL を使用したレコードの補足検索も追加で実行される。 See 項5.2.5.「MySQL による IS NULL の最適化」。

```
SELECT * FROM ref_table WHERE key_column=expr OR key_column IS NULL;
```

  この結合型の最適化は、MySQL 4.1.1 の新機能で、主としてサブクエリを解決する場合に使用される。

- range

  インデックスを使用して、一定の範囲にあるレコードのみが取り出される。key カラムに使用されるインデックスが示される。 key_len には使用される最長のインデックス部分が記載される。 この型では、ref カラムが NULL になる。

  range は、インデックスを張っているカラムが =、<>、>、>=、<、<=、IS NULL、<=>、BETWEEN、および IN を使用して定数と比較される場合に使用される。

```
SELECT * FROM range_table WHERE key_column = 10;

SELECT * FROM range_table WHERE key_column BETWEEN 10 and 20;

SELECT * FROM range_table WHERE key_column IN (10,20,30);

SELECT * FROM range_table WHERE key_part1= 10 and key_part2 IN (10,20,30);
```

- index

  これは、インデックスツリーのみがスキャンされる点を除いて ALL と同じである。一般にインデックスファイルはデータファイルより小さいため、通常は ALL より高速である。

  これは、クエリで 1 インデックスの構成部分であるカラムのみが使用される場合にのみ使用される。

- ALL

  前のテーブルのレコードの組み合わせのそれぞれに対して、フルテーブルスキャンが実行される。一般に、テーブルが const の指定がない第 1 テーブルの場合には適さず、その他の場合はすべて非常に不適である。通常は、さらにインデックスを追加することで ALL を回避し、定数値または以前のテーブルのカラム値を基準にレコードを取り出すようにすることができる。

- possible_keys

  possible_keys カラムは、このテーブル内のレコードの検索に MySQL で使用可能なインデックスを示す。このカラムはテーブルの順序にはまったく依存しないことに注意する。すなわち、possible_keys のキーの一部は、生成されたテーブルの順序では事実上使用できないことになる。

  このカラムが NULL の場合は、対応するインデックスがない。この場合は、WHERE 節でインデックス作成に適するカラムを 1 つ以上参照しているかどうかを調べることでクエリのパフォーマンスを改善できる。参照している場合は適切なインデックスを作成し、再度 EXPLAIN を使用してクエリをチェックする。 See 項6.5.4.「ALTER TABLE 構文」。

  テーブルにあるインデックスを調べるには SHOW INDEX FROM tbl_name を使用する。

- key

  key カラムは、MySQL が実際に使用を決定したキー (インデックス) を示す。選択されたインデックスがない場合、このキーは NULL になる。MySQL で possible_keys カラムに記載されたキーが使用されるように強制するには、クエリで USE KEY/IGNORE KEY を使用する。 See 項6.4.1.「SELECT 構文」。

  また、テーブルで myisamchk --analyze (see 項4.5.6.1.「myisamchk 起動構文」) または ANALYZE TABLE (see 項4.6.2.「ANALYZE TABLE 構文」) を実行することも、オプティマイザでより適したインデックスを選択する際に役立つ。

- key_len

  key_len カラムは、MySQL が使用を決定したキーの長さを示す。key が NULL の場合、この長さは NULL になる。これによって、複合キーで MySQL が実際に使用するパート数が示されることに注意する。

- ref

  ref カラムは、テーブルからレコードを選択する際に key とともに使用されるカラムまたは定数を示す。

- rows

  rows カラムは、クエリの実行に際して調べる必要があると MySQL によって判定されたレコードの数を示す。

- Extra

  このカラムには、MySQL でどのようにクエリが解決されるかに関する追加情報が記載される。以下は、このカラムに記載できる各種テキスト文字列の説明である。

  - Distinct

    マッチした最初のレコードが検索されると、MySQL は現在のレコードの組み合わせによるその後のレコード検索を続行しないことを示す。

  - Not exists

    MySQL でクエリに対する LEFT JOIN 最適化が実行でき、LEFT JOIN に一致するレコードが 1 つ検索されると、前のレコードの組み合わせによるその後のテーブルのレコードについては調べないことを示す。

    この例は以下のとおりである。

```
SELECT * FROM t1 LEFT JOIN t2 ON t1.id=t2.id
WHERE t2.id IS NULL;
```

t2.id が NOT NULL で定義されているとする。この場合、MySQL で t1 がスキャンされ、t1.id で t2 内のレコードのルックアップが行われる。MySQL によって t2 内のマッチするレコードが検索されると、t2 は t2.id ではないと認識され、t2 内の同じ id を持つ残りのレコードのスキャンは行われない。言い換えると、t2 にあるマッチするレコードの数に関わらず、MySQL で実行が必要なことは t1 のレコードのそれぞれに対して、t2 のルックアップを 1 回実行することだけである。

- range checked for each record (index map: #)

  MySQL で使用に適した実際のインデックスを検索できなかったことを示す。代替として、先行テーブルのレコードの組み合わせのそれぞれに対して、使用するインデックス (存在する場合) のチェックが実行され、このインデックスがテーブルからのレコードの取り出しに使用される。非常に高速ではないが、インデックスなしの結合と比較すると高速である。

- Using filesort

  レコードをソートして取り出す方法を決定するには、MySQL はパスを余分に実行しなくてはならないことを示す。join type に従ってすべてのレコードをスキャンし、WHERE 条件に一致する全てのレコードに、ソートキー + 行ポインタを格納して、ソートは実行される。 その後キーがソートされる。 最後に、ソートされた順にレコードが取り出される。

- Using index

  インデックスツリーの情報のみを使用してカラム情報がテーブルから取り出され、実際のレコードを読み取るその後の検索を実行する必要がないことを示す。これは、そのテーブルで使用されたカラムがすべて同一インデックスの構成部分である場合に実行できる。

- Using temporary

  クエリの解決に MySQL で結果を保持するテンポラリテーブルの作成が必要であることを示す。これは一般に、GROUP BY を実行したカラムセットと異なるカラムセットに対して ORDER BY を実行した場合に発生する。

- Using where

  次のテーブルとの一致が調べられるレコードまたはクライアントに送信されるレコードの限定に WHERE 節が使用されることを示す。この情報がなく、テーブルの型が ALL または index である場合はクエリが正常に実行されないことがある (テーブルのすべてのレコードの取得や検査を意図していない場合) 。

クエリを最大限高速に実行する必要がある場合は、Using filesort と Using temporary に注意する必要がある。

EXPLAIN 出力の rows カラムのすべての値を掛け算することで、結合がどの程度適しているかを示す指針を取得できます。これは、クエリの実行時に MySQL で調べる必要があるレコード数の概要を示します。この数値は、max_join_size 変数でクエリを制限する際にも使用されます。 See 項5.5.2.「サーバパラメータのチューニング」。

下記の例は、EXPLAIN によって得られた情報を使用して、JOIN を累進的に最適化する方法を示しています。

ここでは、EXPLAIN を使用して、SELECT ステートメントを調べるとします。

```
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;
```

この例では以下のように想定しています。

- 比較対象のカラムは以下のように宣言されます。

テーブル	カラム	カラムの型
tt	ActualPC	CHAR(10)
tt	AssignedPC	CHAR(10)
tt	ClientID	CHAR(10)
et	EMPLOYID	CHAR(15)
do	CUSTNMBR	CHAR(15)

- テーブルには以下のインデックスがあります。

テーブル	インデックス
tt	ActualPC
tt	AssignedPC
tt	ClientID
et	EMPLOYID (主キー)
do	CUSTNMBR (主キー)

- tt.ActualPC 値の分布が均一ではない。

当初、最適化の実行前は、EXPLAIN ステートメントで次の情報が生成されました。

```
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)
```

各テーブルで type が ALL であるため、この出力は MySQL がすべてのテーブルのデカルト積を生成すると示しています。各テーブルのレコードの数の積の分量を調べる必要があるため、これは非常に時間がかかります。この例の場合は、レコードの数が 74 * 2135 * 74 * 3872 = 45,268,558,720 になります。テーブルがこれより大きい場合は、さらに時間がかかると考えられます。

ここでの問題の 1 つは、宣言の方法が異なると MySQL でカラムのインデックスを効率的に使用できないことにあります。この例では、VARCHAR と CHAR が異なる長さで宣言されていなければ同じになります。tt.ActualPC が CHAR(10) として、et.EMPLOYID が CHAR(15) として宣言されているため、長さの不一致が発生します。

カラムの長さの不一致を修正するため、ALTER TABLE を使用して ActualPC を 10 文字から 15 文字にします。

```
mysql> ALTER TABLE tt MODIFY ActualPC VARCHAR(15);
```

これで tt.ActualPC と et.EMPLOYID はいずれも VARCHAR(15) になりました。 ここでまた EXPLAIN を実行してみると、以下の結果が得られました。

```
table type   possible_keys   key     key_len ref        rows    Extra
tt    ALL    AssignedPC,ClientID,ActualPC NULL NULL NULL 3872    Using where
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
```

これも完全ではありませんが、かなり改善されています (rows 値の積が 74 の係数分だけ減少) 。このバージョンの場合実行に数秒かかります。

第 2 の変更を加えると、tt.AssignedPC = et_1.EMPLOYID と tt.ClientID = do.CUSTNMBR の比較でのカラム長の不一致を解消できます。

```
mysql> ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),
    ->              MODIFY ClientID   VARCHAR(15);
```

ここでは、EXPLAIN から以下の出力が生成されます。

```
table type   possible_keys   key     key_len ref          rows Extra
et    ALL    PRIMARY         NULL    NULL    NULL         74
tt    ref    AssignedPC,     ActualPC 15     et.EMPLOYID   52   Using where
             ClientID,
             ActualPC
et_1  eq_ref PRIMARY         PRIMARY  15     tt.AssignedPC 1
do    eq_ref PRIMARY         PRIMARY  15     tt.ClientID   1
```

これでほとんど改善されています。

残りの問題は、MySQL ではデフォルトで tt.ActualPC カラムの値の分布が均一であると想定されますが、tt テーブルはこれにあてはまらないことです。これは容易に MySQL に示すことができます。

```
shell> myisamchk --analyze PATH_TO_MYSQL_DATABASE/tt
shell> mysqladmin refresh
```

これで結合が完全になり、EXPLAIN で以下の結果が生成されます。

```
table type   possible_keys key     key_len ref          rows Extra
tt    ALL    AssignedPC    NULL    NULL    NULL         3872 Using where
           ClientID,
           ActualPC
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
```

EXPLAIN の出力の rows カラムは、MySQL 結合オプティマイザの学習による推測であることに注意してください。クエリを最適化するには、この数値が実際に近いものであるかどうかを確認する必要があります。実際とかけ離れている場合は、SELECT ステートメントで STRAIGHT_JOIN を使用し、FROM 節でテーブルの順序を変えて一覧表示してみるとパフォーマンスを改善できます。

5.2.2. クエリパフォーマンスの推定

ほとんどの場合、ディスクシークをカウントしてパフォーマンスを推定できます。 小さいテーブルの場合は一般に 1 つのディスクシークでレコードを検索できます (インデックスがキャッシュされることが多いため)。大きいテーブルの場合の推定では、 (B++ ツリーインデックスを使用して) log(row_count) / log(index_block_length / 3 * 2 / (index_length + data_pointer_length)) + 1 のシークがレコードの検索に必要になります。

MySQL では、インデックスブロックが通常 1,024 バイトで、データポインタは通常 4 バイトです。インデックスの長さが 3 (中位の整数) の 500,000 レコードのテーブルの場合は以下のようになります。 log(500,000)/log(1024/3*2/(3+4)) + 1 = 4 シーク

上のインデックスでは約 500,000 * 7 * 3/2 = 5.2M が必要になるため (一般的な状況としてインデックスバッファの 2/3 が使用されていると想定)、メモリにインデックスの多くがあり、OS からデータを読み取り、レコードを検索するには、1 回か 2 回の呼び出しで済むと推定されます。

ただし、書き込みについては、上記の例で新規インデックスの配置場所を探し出すのに 4 シークの要求が、また、インデックスの更新とレコードの書き込みに通常 2 シークが必要になります。

このことは、アプリケーションが対数 N の分だけ低速になるという意味ではないことに注意してください。OS または SQL サーバですべてがキャッシュされている限り、テーブルが拡大しても速度の低下はわずかです。データがキャッシュできないほど増加すると、ディスクシーク (対数 N の分だけ増加する) によって最終的にアプリケーションがバインドされるまで大幅に速度の低下が始まります。これを回避するには、データの増加に合わせてインデックスキャッシュも拡大します。 See 項5.5.2.「サーバパラメータのチューニング」。

5.2.3. SELECT クエリの速度

一般に、低速の SELECT ... WHERE の速度を上げる必要がある場合は、まず、インデックスを追加できるかどうかをチェックします。See 項5.4.3.「MySQL でのインデックスの使用」。一般に複数のテーブル間の参照はすべてインデックスを使用して実行する必要があります。EXPLAIN コマンドを使用して、SELECT に使用されるインデックスを判定できます。See 項5.2.1.「EXPLAIN 構文 (SELECT に関する情報の取得)」。

一般的なヒント

- MySQL によるクエリの最適化を容易にするには、関連データをロードした後にテーブルに対して myisamchk --analyze を実行する。これはインデックスのために、同じ値があるレコードの平均値を更新する (ユニークインデックスの場合、これは常に 1 になる)。MySQL はこれを使用して、2 つのテーブルを '非定数式' で接続する際に選択する

インデックスを判定する。 SHOW INDEX FROM table_name を実行し Cardinality カラムを調べると、analyze の実行結果をチェックできる。

- インデックスに従ってインデックスとデータをソートするには myisamchk --sort-index --sort-records=1 (インデックス 1 でソートする場合) を使用する。速度を上げるには、すべてのレコードの読み取りにユニークインデックスを使用し、そのインデックスに従った順序で読み取りを行うように推奨される。ただし、このソートでは書き込みの最適化はできず、テーブルが大きい場合は時間がかかる。

5.2.4. MySQL による WHERE 節の最適化

WHERE の最適化は、ほとんどの場合 SELECT とともに使用されるため、SELECT 部分に適用されますが、DELETE や UPDATE のステートメントの WHERE にも同じ最適化が適用されます。

また、このセクションは完全なものではないため、注意が必要です。MySQL は多様な最適化を実行するため、すべてを文書化するには時間が足りませんでした。

MySQL によって実行される最適化の一部をここに紹介します。

- 不要なかっこの削除。

```
  ((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)
```

- 定数の折りたたみ。

```
  (a<b AND b=c) AND a=5
-> b>5 AND b=c AND a=5
```

- 定数条件の削除 (定数の折りたたみに必要) 。

```
  (B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)
-> B=5 OR B=6
```

- インデックスが使用する定数式が評価されるのは、1 回に限られる。
- WHERE がない単一テーブルの COUNT(*) は、MyISAM と HEAP テーブルのテーブル情報から直接取り出される。 これは、テーブル 1 つのみで使用する場合はすべての NOT NULL 式でも実行される。
- 無効定数式の早期検出。MySQL は実行不可能な SELECT ステートメントがある場合、それを迅速に検出し、結果としてレコードを返さない。
- GROUP BY またはグループ関数 (COUNT()、MIN()。..) を使用しない場合、HAVING は WHERE とマージされる。
- サブ結合のそれぞれに、単純な WHERE が構造化され、サブ結合ごとに迅速に WHERE 評価を取得し、可能な限り迅速にレコードをスキップする。
- クエリ内の他のすべてのテーブルの前に、まず、すべての定数テーブルが読み込まれる。 定数テーブルとは以下のものを指す。
    - 空白テーブルまたは 1 レコードのみのテーブル。

- UNIQUE インデックスまたは PRIMARY KEY を使う WHERE 節とともに使用されるテーブルで、インデックス部分のすべてが定数式とともに使用され、そのインデックス部分が NOT NULL として定義されている場合。

以下のテーブルはすべて定数テーブルとして使用される。

```
mysql> SELECT * FROM t WHERE primary_key=1;
mysql> SELECT * FROM t1,t2
   ->       WHERE t1.primary_key=1 AND t2.primary_key=t1.id;
```

- テーブルを結合する最適な結合の組み合わせは、すべての可能性を試行してみることで発見される。ORDER BY および GROUP BY 内の全てのカラムが 1 つのテーブルに存在する場合、結合を行う時は第一にこのテーブルが選ばれる。
- ORDER BY 節とそれと異なる GROUP BY 節がある場合、あるいは、ORDER BY または GROUP BY に結合キューの第 1 テーブルとは異なるテーブルのカラムが含まれている場合は、テンポラリテーブルが作成される。
- SQL_SMALL_RESULT を使用する場合、MySQL ではメモリ内のテンポラリテーブルが使用される。
- テーブルインデックスごとにクエリが行われ、スパンがレコードの 30% 未満である最適インデックスが使用される。このようなインデックスが検索されない場合は、クイックテーブルスキャンが使用される。
- 状況によっては、MySQL でデータファイルの参照もせずにインデックスからレコードを読み取れる場合もある。インデックスから使用されるカラムのすべてが数値型の場合、クエリの解決にはインデックスツリーのみが使用される。
- レコードのそれぞれが出力される前に、HAVING 節と一致しないレコードはスキップされる。

非常に高速なクエリのサンプルをいくつか紹介します。

```
mysql> SELECT COUNT(*) FROM tbl_name;
mysql> SELECT MIN(key_part1),MAX(key_part1) FROM tbl_name;
mysql> SELECT MAX(key_part2) FROM tbl_name
   ->       WHERE key_part_1=constant;
mysql> SELECT ... FROM tbl_name
   ->       ORDER BY key_part1,key_part2,... LIMIT 10;
mysql> SELECT ... FROM tbl_name
   ->       ORDER BY key_part1 DESC,key_part2 DESC,... LIMIT 10;
```

次のクエリは、インデックスツリーのみを使用して解決されます (インデックスのあるカラムが数値型であると想定) 。

```
mysql> SELECT key_part1,key_part2 FROM tbl_name WHERE key_part1=val;
mysql> SELECT COUNT(*) FROM tbl_name
   ->       WHERE key_part1=val1 AND key_part2=val2;
mysql> SELECT key_part2 FROM tbl_name GROUP BY key_part1;
```

The following queries use indexing to retrieve the rows in sorted order without a separate sorting pass: 次のクエリは、ソートのパスを分けることなく、ソートしたレコードを取り出すためにインデックスを使用します。

```
mysql> SELECT ... FROM tbl_name
   ->          ORDER BY key_part1,key_part2,... ;
mysql> SELECT ... FROM tbl_name
   ->          ORDER BY key_part1 DESC,key_part2 DESC,... ;
```

5.2.5. MySQL による IS NULL の最適化

MySQL では、column = constant_value の場合と同じ最適化を column IS NULL に対しても実行できます。たとえば、MySQL では、インデックスと範囲を使用して、IS NULL で NULL を検索できます。

```
SELECT * FROM table_name WHERE key_col IS NULL;

SELECT * FROM table_name WHERE key_col <=> NULL;

SELECT * FROM table_name WHERE key_col=# OR key_col=# OR key_col IS NULL
```

OUTER JOIN に使用されないテーブル上で、WHERE 節内で column_name IS NULL で定義された物を NOT NULL と使用する場合、その式は消去して最適化されます。

MySQL 4.1.1 では、column = expr AND column IS NULL の組み合わせを最適化する機能が追加されています。この最適化が使用される場合は、EXPLAIN は ref_or_null を表示します。

この最適化は、すべてのキー部分で IS NULL を 1 つ処理できます。

最適されたクエリのサンプルをいくつか紹介します (t2 のキーを (a,b) とします) 。

```
SELECT * FROM t1 WHERE t1.a=expr OR t1.a IS NULL;

SELECT * FROM t1,t2 WHERE t1.a=t2.a OR t2.a IS NULL;

SELECT * FROM t1,t2 WHERE (t1.a=t2.a OR t2.a IS NULL) AND t2.b=t1.b;

SELECT * FROM t1,t2 WHERE t1.a=t2.a AND (t2.b=t1.b OR t2.b IS NULL);

SELECT * FROM t1,t2 WHERE (t1.a=t2.a AND t2.a IS NULL AND ...) OR (t1.a=t2.a AND t2.a IS NULL AND ...);
```

まず、ref_or_null はリファレンスキーの読み取りを行い、その後 NULL キーのあるレコードの検索を実行します。

この最適化では、1 つの IS NULL レベルしか処理できないことに注意が必要です。

```
SELECT * FROM t1,t2 where (t1.a=t2.a AND t2.a IS NULL) OR (t1.b=t2.b AND t2.b IS NULL);
```

この状況で MySQL は (t1.a=t2.a AND t2.a IS NULL) の部分に対してキーのルックアップを実行するのみで、b のキー部分は使用できません。

5.2.6. MySQL による DISTINCT の最適化

DISTINCT が ORDER BY と組み合わされて用いられると、多くの場合はテンポラリテーブルが必要になります。

DISTINCT は GROUP BY をともなう可能性が高いので、SELECT されないフィールドを ORDER BY または HAVING した時に、どのように MySQL が機能するかを認識しておく必要があります。 See 項6.3.7.3.「非表示のフィールドに対する GROUP BY」。

LIMIT row_count を DISTINCT とともに使用した場合、MySQL は一意のレコードを row_count 行検索するとただちに検索を停止します。

使用するテーブル内のカラムを使用しない場合、MySQL は最初にマッチするレコードを検索するとただちに未使用テーブ

ルのスキャンを停止します。

```
SELECT DISTINCT t1.a FROM t1,t2 WHERE t1.a=t2.a;
```

ここでは、t1 が t2 の前に使用され（EXPLAIN によるチェック）、t2 で最初のレコードが検索されると t2からの読み取り（t1 の特定のレコード）を停止します。

5.2.7. MySQL による LEFT JOIN と RIGHT JOIN の最適化

MySQL の A LEFT JOIN B join_condition は以下のように実装されます。

- テーブル B はテーブル A と A が依存するすべてのテーブルに依存するように設定される。
- テーブル A は、LEFT JOIN 条件で使用されるすべてのテーブル（B を除く）に依存するように設定される。
- LEFT JOIN 条件は、テーブル B からのレコードの取り出し方法の判定に使用される（言い換えると、WHERE 節の条件はいずれも使用されない）。
- あるテーブルが全てのテーブルの後に読み取られる場合を除き、通常の最適化全てが行われる。依存関係が循環している場合は、MySQL からエラーが出力される。
- 標準の WHERE 最適化すべてが実行される。
- A に WHERE 節の条件にマッチするレコードがあり、B に ON 条件にマッチするレコードがない場合、B のカラムの値が NULL に設定されたレコードが生成される。
- テーブルのいずれかに存在しないレコードを検索する際に LEFT JOIN を使用していて、かつ、WHERE 節内で、NOT NULL と定義した column_name を column_name IS NULL で評価した場合、 MySQL は LEFT JOIN 条件に一致するレコードを 1 つ検索すると、その後はレコードの検索（特定のキー組み合わせの）を停止する。

RIGHT JOIN の実装は LEFT JOIN と類似しています。

テーブル読み取り順序は LEFT JOIN と STRAIGHT JOIN によって強制されるため、チェック対象のテーブル順列が減少し、結合オプティマイザ（テーブルの結合順序を計算する）の動作の速度がさらに上がります。

上記は、該当する種類のクエリを実行した場合であることに注意してください。

```
SELECT * FROM a,b LEFT JOIN c ON (c.key=a.key) LEFT JOIN d (d.key=a.key)
      WHERE b.key=d.key
```

LEFT JOIN が d の前に読み取るように強制するため、MySQL では b の完全スキャンが実行されます。

この状況はクエリを以下のように変更して修正します。

```
SELECT * FROM b,a LEFT JOIN c ON (c.key=a.key) LEFT JOIN d (d.key=a.key)
      WHERE b.key=d.key
```

4.0.14 以降、MySQL では以下の LEFT JOIN 最適化が行われます。

生成された NULL レコードで WHERE 条件が常に false である場合、LEFT JOIN は通常の結合に変更されます。

たとえば、t2 カラムが NULL であるとすると、以下のクエリの WHERE 節は false になるため、通常の結合に変換しても問題ありません。

```
SELECT * FROM t1 LEFT t2 ON (column) WHERE t2.column2 =5;
->
SELECT * FROM t1,t2 WHERE t2.column2=5 AND t1.column=t2.column;
```

これでクエリが改善できる場合、MySQL がテーブル t1 を読み取る前にテーブル t2 を使用できるようになるためスピードが向上します。テーブルの順序を指定して強制する場合は STRAIGHT JOIN を使用します。

5.2.8. MySQL による ORDER BY の最適化

余分なソートを行わずに ORDER BY または GROUP BY の要求に応じるために、MySQL はインデックスを使用する場合があります。

全ての使用されていないインデックス部分と他の部分が WHERE 節内で定数であるカラムである場合、ORDER BY がインデックスに完全にマッチしない場合でもこのインデックスを使用できます。 次のクエリではインデックスを使用して ORDER BY / GROUP BY 部分を解決します。

```
SELECT * FROM t1 ORDER BY key_part1,key_part2,...
SELECT * FROM t1 WHERE key_part1=constant ORDER BY key_part2
SELECT * FROM t1 WHERE key_part1=constant GROUP BY key_part2
SELECT * FROM t1 ORDER BY key_part1 DESC,key_part2 DESC
SELECT * FROM t1 WHERE key_part1=1 ORDER BY key_part1 DESC,key_part2 DESC
```

MySQL で ORDER BY の解決にインデックスを使用できない場合は以下のとおりです（この場合も MySQL は WHERE 節の条件に一致するレコードの検索にインデックスを使用します）。

- 複数のキーに対して ORDER BY を実行する場合。

  SELECT * FROM t1 ORDER BY key1,key2

- 連続しないキー部分に対して ORDER BY を実行する場合。

  SELECT * FROM t1 WHERE key2=constant ORDER BY key_part2

- ASC と DESC が混在している場合。

  SELECT * FROM t1 ORDER BY key_part1 DESC,key_part2 ASC

- レコードの取り出しに使用されるキーが ORDER BY の実行に使用されるキーと異なる場合。

  SELECT * FROM t1 WHERE key2=constant ORDER BY key1

- ORDER BY で多くのテーブルとカラムを結合していて、それら全てがレコードの取り出しに使用される最初の非 const テーブルではない場合（これは EXPLAIN で出力される最初のテーブルで、かつ、const メソッドを使用していないテーブル）。

- ORDER BY と GROUP BY 式が異なる場合。

- 使用されたテーブルインデックスが、並び順にレコードを格納していないインデックスタイプの場合（HEAP テーブルの HASH インデックスなど）。

MySQL で結果のソートが必要な場合は、以下のアルゴリズムが使用されます。

- キーまたはテーブルスキャンに従ってすべてのレコードが読み取られる。 WHERE 節に一致しないレコードはスキップされる。
- ソートキーがバッファ（サイズ sort_buffer）に格納される。
- バッファが満杯になると、qsort が実行され結果がテンポラリファイルに格納される。ポインタはソートブロックに保存される（すべてのレコードがソートバッファに適合する場合は、テンポラリファイルが作成されない）。
- すべてのレコードが読み取られるまで上記項目が反復される。
- MERGEBUFF（7）領域まで、別のテンポラリファイルの 1 ブロックにマルチマージが実行される。最初のファイルの全ブロックが 2 つめのファイルに配置されるまで反復される。
- 残りが MERGEBUFF2（15）ブロック未満になるまで、以下が反復される。
- 最終マルチマージでは、レコードに対するポインタ（ソートキーの最終部分）のみが結果ファイルに書き込まれる。
- 次に、sql/records.cc のコードが使用され、結果ファイルのポインタによってソートされた順序で読み取りが行われる。これを最適化するためローポインタの大きなブロックを読み込み、そのソートを行ってからソートされた順序でレコードバッファにレコードを読み取る（read_rnd_buffer_size）。

EXPLAIN SELECT ... ORDER BY を使用すると、MySQL でインデックスを使用してクエリを解決できるかどうかをチェックできます。extra カラムに Using filesort が出力された場合は、MySQL で ORDER BY の解決にインデックスを使用できません。 See 項5.2.1.「EXPLAIN 構文（SELECT に関する情報の取得）」。

さらに ORDER BY の速度を上げる必要がある場合はまず、ソートフェーズを実行する必要なく MySQL でインデックスを使用できるかどうかを調べます。これが不可能な場合は、以下を実行できます。

- sort_buffer_size 変数の値を増やす。
- read_rnd_buffer_size 変数の値を増やす。
- tmpdir に空き領域が大量にある専用ディスク上のディレクトリを指定する。 MySQL 4.1 以降を使用している場合、tmpdir に対してコロン : （Windows の場合はセミコロン ;）で区切ったパスの一覧を設定することで、複数の物理ディスク間の負荷を分散させることができる。この物理ディスクは、ラウンドロビン方法で使用される。 注意: これらのパスは、同一ディスクの複数のパーティションではなく、異なる物理ディスクである必要がある。

デフォルトでは、クエリで ORDER BY x,y[,...] と指定した場合と同様に MySQL によってすべての GROUP BY x,y[,...] クエリがソートされます。ORDER BY 節を明示的に記述した場合、ソートは発生するものの、MySQL はスピードを損なうことなくそれを最適化します。 クエリに GROUP BY が含まれていて、もし結果のソートのオーバヘッドを回避したいならば、ORDER BY NULL を指定することでソートを抑止できます。

```
INSERT INTO foo SELECT a,COUNT(*) FROM bar GROUP BY a ORDER BY NULL;
```

5.2.9. MySQL による LIMIT の最適化

HAVING を使用するのではなく LIMIT row_count を使用している場合、MySQL によるクエリの処理方法が異なる場合があります。

- LIMIT を使用して数レコードしか選択していないと、フルテーブルスキャンが行われそうな場合に、MySQL はインデックスを使うことがある。
- ORDER BY とともに LIMIT row_count を使用している場合、MySQL ではすべてのテーブルがソートされるのではなく、最初の row_count レコードの検索が行われた時点でただちにソートを終了する。
- LIMIT row_count を DISTINCT とあわせて使用した場合、MySQL は一意の row_count 行のレコードを検索するとただちに停止する。
- GROUP BY がキーを順番に読む (またはキーのソートを実行して読む) ことで解決でき、キーの値が変わるまで サマリが計算される場合もある。この場合、LIMIT row_count では不要な GROUP BY 値の計算がすべて行われなくなる。
- MySQL が最初の # レコードをクライアントに送信すると、クエリが中止される (SQL_CALC_FOUND_ROWS を使用していない場合) 。
- LIMIT 0 は常に迅速に空のセットを返す。これは、クエリのチェックおよび結果として返るカラムのカラム型の取得に役立つ。
- サーバでテンポラリテーブルを使用してクエリが解決される場合、LIMIT row_count が必要な領域の計算に使用される。

5.2.10. INSERT クエリの速度

レコード挿入の時間構成の概要は次のとおりです。

- 接続: (3)
- サーバへのクエリの送信: (2)
- クエリの解析: (2)
- レコード挿入: (1 x レコードサイズ)
- インデックス挿入: (1 x インデックス数)
- クローズ: (1)

ここに示した数値は、時間全体を比例的に配分したものです。テーブルを開く初期オーバヘッドは算入されていません (これは同時実行クエリのそれぞれで 1 回実行されます) 。

テーブルのサイズによって対数 N の分だけインデックス挿入の速度が低下します (B ツリー) 。

挿入の速度を上げる方法

- 1 つのクライアントから同時に多数のレコードを挿入する場合はマルチプル INSERT ステートメントを使用する。これで独立した INSERT ステートメントの使用時と比較して大幅に (場合によっては数倍) 速度が上がる。空ではないテ

ーブルにデータを追加する場合は、さらに速度を上げるために bulk_insert_buffer_size 変数を調整する。 See 項4.6.8.4.「SHOW VARIABLES」。

- 異なる複数のクライアントから大量のレコードを挿入する場合は、INSERT DELAYED ステートメントを使用すると速度を上げることができる。 See 項6.4.3.「INSERT 構文」。
- MyISAM テーブルでは、テーブルに削除されたレコードがない場合、SELECT の実行と同時にレコードを挿入できることに注意する。
- テキストファイルからテーブルをロードする場合は LOAD DATA INFILE を使用する。通常、これは INSERT ステートメントを多数使用する場合と比較して、20 倍速度が上がる。 See 項6.4.8.「LOAD DATA INFILE 構文」。
- テーブルにインデックスが多数ある場合、操作を少し追加するだけで LOAD DATA INFILE の実行速度をさらに上げることができる。以下の手順を使用する。

  1. CREATE TABLE を使用して、テーブルを作成する。mysql や Perl-DBI などを使用する。
  2. FLUSH TABLES ステートメントまたはシェルコマンド mysqladmin flush-tables を実行する。
  3. myisamchk --keys-used=0 -rq /path/to/db/tbl_name を使用する。これでテーブルからすべてのインデックスの使用が削除される。
  4. LOAD DATA INFILE を使用し、テーブルにデータを挿入する。これはインデックスをまったく更新しないため、非常に高速になる。
  5. テーブルを読み取り専用にする場合は、myisampack を実行してテーブルを小さくする。 See 項7.1.2.3.「圧縮テーブルの特性」。
  6. myisamchk -r -q /path/to/db/tbl_name を使用してインデックスを作成しなおす。これは、ディスクに書き込む前にメモリにインデックスツリーを作成して、ディスクシークを回避するため非常に高速になる。生成されたインデックスツリーは完全にバランスが取られている。
  7. FLUSH TABLES ステートメントまたはシェルコマンド mysqladmin flush-tables を実行する。

  空のテーブルへ挿入する場合は、LOAD DATA INFILE は上記の最適化を実行します。上記手順との主な相違点は、myisamchk にインデックス作成用のテンポラリメモリを大幅に割り当てることができる点です。

  MySQL 4.0 以降は、myisamchk --keys-used=0 -rq /path/to/db/tbl_name の代わりに ALTER TABLE tbl_name DISABLE KEYS を、また myisamchk -r -q /path/to/db/tbl_name の代わりに ALTER TABLE tbl_name ENABLE KEYS を使用することもできます。このようにすると、FLUSH TABLES ステップをスキップすることもできます。
- 複数ステートメントを使用して実行される挿入の速度を、テーブルをロックすることによって上げることができる。

```
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;
```

主な速度の相違点は、すべての INSERT ステートメントの完了後にインデックスバッファが 1 回のみディスクにフラッシュされることである。通常は、INSERT ステートメントの数と同じだけ、インデックスバッファのフラッシュが行われる。すべてのレコードを 1 つのステートメントで挿入できる場合はロックの必要がない。

トランザクションテーブルの場合は、LOCK TABLES ではなく BEGIN/COMMIT を使用して速度の改善を図る。

ロックは複数の同時接続テストの合計時間も短縮するが、一部のスレッドの最大待機時間は長くなる（ロックの際に待機するため）。次の例を参照してください。

```
スレッド 1 は 1000 レコードをインサート
スレッド 2, 3, 4 は 1 レコードをインサート
スレッド 5 は 1000 レコードをインサート
```

ロックを使用しない場合、2、3、4 は 1 と 5 の前に終了する。ロックを使用した場合は、2、3、4 は 1 と 5 の前には終了しない確率が高くなるが、合計時間は約 40% 短縮される。

MySQL では、INSERT、UPDATE、および DELETE の演算が非常に速いため、約 5 つより多い挿入や 1 レコード更新する前にロックを追加すると総合的なパフォーマンスを改善できる。1 行で非常に多数の挿入を実行する場合は、ときどき（約 1,000 レコードごと）LOCK TABLES に UNLOCK TABLES を続けて実行して、他のスレッドからのテーブルへのアクセスを可能にすることができる。これでもパフォーマンの増加が得られる。

言うまでもなく、データのロードには LOAD DATA INFILE のほうが大幅に高速である。

LOAD DATA INFILE と INSERT の両方の速度をさらに改善するには、キーバッファを拡張します。 See 項5.5.2.「サーバパラメータのチューニング」。

5.2.11. UPDATE クエリの速度

更新クエリは、SELECT クエリと同様に最適化されますが、書き込みオーバヘッドが加算されます。書き込みの速度は更新対象のデータのサイズおよび更新対象のインデックス数によって異なります。変更がないインデックスは更新されません。

更新の速度を上げるもう 1 つの方法は、更新を遅延して 1 行で多数の更新を後から行うことです。1 行での多数の更新は、テーブルをロックすると同時に行う場合と比較して大幅に高速に実行できます。

可変長レコードの場合は、合計の長さが今よりも長いものにレコードを更新すると、レコードが分割される場合があることに注意します。このため、頻繁にこれを実行する場合は、ときどき OPTIMIZE TABLE することが重要になります。 See 項4.6.1.「OPTIMIZE TABLE 構文」。

5.2.12. DELETE クエリの速度

テーブル内のすべてのレコードを削除する場合は、TRUNCATE TABLE table_name を使用します。 See 項6.4.6.「TRUNCATE 構文」。

レコード削除に要する時間は、完全にインデックス数に比例します。レコード削除の速度を上げるには、インデックスキャッシュのサイズを拡大します。 See 項5.5.2.「サーバパラメータのチューニング」。

5.2.13. その他の最適化のヒント

システム高速化のためのヒント（順不同）

- 接続オーバヘッドを回避するには、データベースに対して永続的な接続を使用する。永続的な接続を使用せずにデータ

ベースに対して多数の新規接続を実行する場合は、thread_cache_size 変数の値の変更が必要になることがある。 See 項5.5.2.「サーバパラメータのチューニング」。

- 常にすべてのクエリがテーブル内に作成したインデックスを実際に使用していることを確認する。MySQL では、EXPLAIN コマンドでこれを実行できる。 See 項5.2.1.「EXPLAIN 構文（SELECT に関する情報の取得）」。
- 大量に更新された MyISAM テーブルに対して複雑な SELECT クエリを使用しないようにする。これでテーブルロックを回避する。
- 削除されたレコードがない MyISAM テーブルの場合は、別のクエリでそのテーブルからの読み取りが行われるのと同時にレコードを挿入できる。これがあなたにとって重要ならば、レコードの削除が不要なメソッドや、大量のレコード削除後の OPTIMIZE TABLE の実行を検討する。
- expr1,expr2... の順に従って頻繁にレコードを読み取る場合は、ALTER TABLE ... ORDER BY expr1,expr2... を使用する。テーブルが大幅に変更された後にこのオプションを使用すると、パフォーマンスを改善できる。
- 他のカラムの情報を基にした 'ハッシュされた' カラムを導入することが役立つ場合がある。このカラムが短いもので、一意性がある場合は、多数のカラムに大きなインデックスを使用するより大幅に高速化できる。MySQL では、追加カラムの使用が以下のように非常に容易である。 SELECT * FROM table_name WHERE hash=MD5(CONCAT(col1,col2)) AND col_1='constant' AND col_2='constant'
- 大量に変更があるテーブルはすべて VARCHAR や BLOB のカラムを使用しないようにする。VARCHAR または BLOB カラムを 1 つ使用するとレコードがただちに可変長になってしまう。 See 章 7. MySQL のテーブル型。
- 一般に、1 つのテーブルを複数のテーブルに分割することは、レコードが '大きく' なるだけで高速化の役には立たない。レコードにアクセスする際の、最も大きなパフォーマンス要因は、レコードの最初のバイトを見つけるためのディスクシークである。データの検索後、ほとんどの新規ディスクでは、大多数のアプリケーションに十分な速度でレコード全体を読み取ることができる。テーブルの分割が実際に有効な状況は、固定長テーブルへの変更が可能な可変長テーブル（上記参照）の場合か、テーブルのスキャンを非常に頻繁に必要としながらもほとんどのカラムを結果に必要としない場合のみである。 See 章 7. MySQL のテーブル型。
- 多数のレコードの情報から計算する頻度を非常に高くする必要がある場合（カウントの場合など）、新たなテーブルを導入し、リアルタイムでカウンタを更新するほうがはるかに適している。UPDATE table SET count=count+1 WHERE index_column=constant のような更新は非常に高速にできる。

  実際これは、MyISAM や ISAM のようにテーブルロック（複数リーダ/単一ライタ）のみの MySQL テーブル型を使用する場合に非常に重要である。また、このような場合は行ロックマネージャで必要な作業が少なくなるため、ほとんどのデータベースでパフォーマンスが改善される。
- 大きなログテーブルから統計を収集する必要がある場合は、テーブル全体をスキャンするのではなく、サマリテーブルを使用する。サマリの管理は、リアルタイムで統計を実行する場合と比較して非常に高速になる。何らかの変更がある（業務上の決定に応じて）場合は、ログから新規にサマリテーブルを再生成したほうが、実行アプリケーションの変更よりはるかに高速である。
- 可能であれば、レポートをリアルタイムか集計かのいずれかに分類するように推奨する。集計レポートに必要なデータは、サマリテーブルから生成され、サマリテーブルは実データから生成される。
- カラムにデフォルト値がある利点を生かす。挿入対象の値がデフォルト値と相違する場合のみ明示的に値を挿入する。これで、MySQL が要する解析作業が軽減され、挿入の速度が改善される。

- 状況によっては、データを BLOB にパックし、格納したほうが便利である。このような場合は、BLOB へのパックおよびパック解除を行うコードをアプリケーションに追加する必要があるが、あるステージにおける大量のアクセスを省略できることになる。 これは、固定長テーブル構造に準拠しないデータがある場合に実用的である。
- 通常は、すべてのデータが冗長にならないようにする必要がある（データベースセオリの第 3 正規化）が、高速化を図る必要がある場合はデータなどの複製やサマリテーブルの作成をためらうべきではない。
- ストアドプロシージャや UDF（ユーザ定義関数）はパフォーマンスの向上に役立つ手段である。ただし、これをサポートしないデータベースを使用する場合は、常に代替の（速度が遅い）方法も用意する必要がある。
- アプリケーションのクエリと応答をキャッシュすること、および挿入と更新の同時実行を試行することは必ず高速化に役立つ。データベースでロックテーブルがサポートされる場合（MySQL や Oracle など）は、これによって確実にすべての更新後にインデックスキャッシュが 1 回だけフラッシュされるようにできる。
- データの書き込みするタイミングを知る必要がない場合は、INSERT /*! DELAYED */ を使用する。多数のレコードが 1 回のディスクへの書き込みで書き込まれるため、これで高速化が図れる。
- SELECT の優先を上げる場合は、INSERT /*! LOW_PRIORITY */ を使用する。
- SELECT がキューをジャンプするようにする場合は、SELECT /*! HIGH_PRIORITY */ を使用する。言い換えると、書き込み待機中のユーザがいる場合でも、SELECT を実行できるようになる。
- 1 つの SQL コマンドで多数のレコードを格納するには、複数行の INSERT ステートメントを使用する（これは多数の SQL でサポートされている）。
- 大量のデータをロードする場合は LOAD DATA INFILE を使用する。これは通常の挿入より高速になる。
- 一意の値にするには AUTO_INCREMENT カラムを使用する。
- 一定の間隔で OPTIMIZE TABLE を使用して、動的テーブルの断片化を回避する。 See 項4.6.1.「OPTIMIZE TABLE 構文」。
- 可能ならば HEAP を使用して高速化を図れるようにする。 See 章 7. MySQL のテーブル型。
- 通常の Web サーバセットアップを使用する場合は、画像をファイルとして格納する。言い換えると、データベース内にはファイル参照のみを格納する。この主な理由は、通常の Web サーバのほうがデータベースコンテンツと比較してファイルのキャッシュに優れているためである。このため、ファイルを使用したほうがシステムの高速化を容易に図れる。
- 頻繁にアクセスされる非クリティカルデータ（クッキーなしでユーザに最後に表示されたバナーの情報など）にはメモリテーブルを使用する。
- 別のテーブルで同一情報を扱うカラムは、同じ宣言をし、同じ名前を付ける。バージョン 3.23 以前はこのようにしないと結合の速度が遅くなる。

  名前はなるべく単純なものに保持する（カスタマテーブルでは customer_name ではなく name を使用する）。他の SQL サーバに移植可能にすることを考慮するなら、名前を 18 文字未満にする。
- 高速化が大きく必要とされる場合は、複数の SQL サーバがサポートするデータストレージの低レベルインタフェースを調べる必要がある。たとえば、MySQL MyISAM に直接アクセスすることによって、SQL インタフェース使用時と比較して 2-5 倍の速度が得られることもある。 これを実行可能にするには、データをアプリケーションと同じサーバに

配置し、また通常は 1 プロセスのみからアクセスするようにする必要がある (外部ファイルロックが非常に低速なため) 。上記の問題は、MySQL サーバに低レベルの MyISAM コマンドを導入することで解消できる (必要に応じてパフォーマンスを改善する容易な手段の 1 つ) 。データベースインタフェースを慎重に設計することで、この種の最適化を容易にサポートできる。

- 多くの場合、テキストファイルにアクセスするのと比較して、データベースからデータにアクセスしたほうが高速である。この理由は一般にテキストファイル (数値データ使用時) よりデータベースのほうがよりコンパクトで、必要なディスクアクセスが少ないことによる。また、テキストファイルを解析してレコードとカラムの境界を検索する必要がないため、コードも節約できる。
- レプリケーションでも高速化を図ることができる。 See 項4.11.「MySQL のレプリケーション」。
- DELAY_KEY_WRITE=1 オプションでテーブルを定義すると、ファイルが閉じられるまでディスクにログが記録されないためインデックス更新の速度が上がる。 この欠点は、途中で mysqld の強制終了が発生した場合にテーブルに問題がないことを確認するため、mysqld を開始する前に、テーブルに対して myisamchk を実行する必要があるということである。キー情報は常にデータから生成可能であるため、DELAY_KEY_WRITE を使用しても何も消失はしない。

5.3. ロック関連の問題

5.3.1. MySQL のテーブルロック方法

ロックメソッドそれぞれについての説明は付録にあります。 See 項E.4.「ロック方法」。

InnoDB 型と BDB 型のテーブルを除き、MySQL のロックはすべてデッドロックフリーです。 これは、常にクエリの開始時に必要なすべてのロックを要求し、また、常に同じ順序でテーブルをロックすることによって管理されます。

InnoDB 型のテーブルは、行ロックを自動的に取得し、BDB 型のテーブルは、トランザクションの開始時ではなく SQL ステートメントの処理時にページロックを自動取得します。

MySQL は WRITE ロックに以下のロック方法を使用します。

- テーブルにロックがない場合は、書き込みロックを配置する。
- その他の場合は、書き込みロックキューにロック要求を配置する。

MySQL は READ ロックに以下のロック方法を使用します。

- テーブルに書き込みロックがない場合は、読み取りロックを配置する。
- その他の場合は、読み取りロックキューにロック要求を配置する。

ロックが解除されると、まず書き込みロックキューのスレッドでロックが使用可能になり、その後読み取りロックキューのスレッドで利用可能になります。

これは、1 つのテーブルに対して更新が多数ある場合に、更新がすべてなくなるまで SELECT ステートメントが待機することを意味します。

テーブルに対して多数の INSERT および SELECT 操作を行う必要がある場合、このような待機を回避するには、テンポラリテーブルにレコードを挿入し、一定の間隔でテンポラリテーブルからのレコードで実テーブルを更新します。

これは以下のコードで実行できます。

```
mysql> LOCK TABLES real_table WRITE, insert_table WRITE;
mysql> INSERT INTO real_table SELECT * FROM insert_table;
mysql> TRUNCATE TABLE insert_table;
mysql> UNLOCK TABLES;
```

特定の状況で取り出しに優先順位を設定するには、LOW_PRIORITY オプションを INSERT、UPDATE または DELETE に、あるいは HIGH_PRIORITY オプションを SELECT に使用します。また、--low-priority-updates オプションで mysqld を開始しても同じ効果が得られます。

SQL_BUFFER_RESULT の使用もテーブルロックを短縮するのに役立ちます。 See 項6.4.1.「SELECT 構文」。

さらに、1 つのキューを使用するように mysys/thr_lock.c のロックコードを変更することもできます。この場合は、書き込みロックと読み取りロックの優先度が同じになり、アプリケーションによっては高速化に役立ちます。

5.3.2. テーブルロック関連の問題

MySQL のテーブルロックコードはデッドロックフリーです。

MySQL は、InnoDB テーブルと BDB テーブルを除くすべてのテーブル型にテーブルロックを使用して、非常に高速なロックを実現します。大型のテーブルの場合、ほとんどのアプリケーションで行ロックと比較してテーブルロックのほうがはるかに優れていますが、これには危険もあります。

InnoDB テーブルと BDB テーブルの場合は、MySQL で LOCK TABLES によって明示的テーブルをロックした場合のみテーブルロックが使用されます。 InnoDB は自動行レベルロックを使用し、BDB はページレベルロックを使用してトランザクションの独立を確実にするため、これらのテーブル型には、LOCK TABLES をまったく使用しないように推奨します。

MySQL バージョン 3.23.7 以降は、MyISAM テーブルへのレコードの挿入を、他のスレッドが同一テーブルから読み取りを行うのと同時に実行できるようになりました。現在のところ、挿入実行時にテーブルのレコード削除後のホールがない場合にのみ、この機能が使用可能になるため注意が必要です。すべてのホールに新規のデータが入力されると、同時挿入が自動的に再度可能になります。

テーブルロックにより、同時に多数のスレッドがテーブルからの読み取りを行うことができますが、あるスレッドがテーブルへの書き込みを行うときは、まず排他処理をする必要があります。更新時は、特定のテーブルにアクセスしようとする他のすべてのスレッドが、更新の準備ができるまで待機します。

一般にテーブルの更新は SELECT より重要だと見なされるため、テーブルを更新するステートメントはすべて、テーブルから情報を取り出すステートメントより優先度が高くなります。これにより、更新では特定のテーブルに対して大量の重いクエリが使用されるため、更新が '資源枯渇' にさらされないことが確実になります (これは、更新を実行するステートメントを LOW_PRIORITY とともに使用するか、SELECT ステートメントとともに HIGH_PRIORITY を使用することで変更できます) 。

MySQL バージョン 3.23.7 以降は、max_write_lock_count 変数を使用して、テーブルに対する挿入が一定数行われた後に、MySQL によってテーブルの使用を待機している SELECT ステートメントのすべてに高い優先度を強制的に設定できるようになりました。

ただし、テーブルロックは以下のシナリオには適していません。

- クライアントが実行に長時間かかる SELECT を使用する。
- その後、別のクライアントが使用テーブルに対して UPDATE を使用する。このクライアントは SELECT が完了するまで待機が必要になる。
- 別のクライアントが同一テーブルに対してさらに SELECT ステートメントを使用する。UPDATE は SELECT より優先度が高いため、この SELECT は UPDATE が完了するまで待機が必要になる。また、最初の SELECT の完了を待つ必要もある。
- full disk などによってスレッドが待機中の場合、そのテーブルへのアクセスが必要なすべてのスレッドが追加のディスク容量が使用可能になるまで待機状態に置かれる。

この問題に対応する解決策は以下のとおりです。

- SELECT ステートメントの実行の高速化を試行する。これにはサマリテーブルの作成が必要な場合もある。
- --low-priority-updates のオプションで mysqld を開始する。これは、テーブルを更新 (変更)するすべてのステートメントの優先度を SELECT ステートメントの優先度より低くする。この場合、前のシナリオの最後の SELECT ステートメントが INSERT ステートメントより前に実行されることになる。
- LOW_PRIORITY 属性を使用して、特定の INSERT、UPDATE、または DELETE ステートメントの優先度を低く設定できる。
- max_write_lock_count の値を低くして mysqld を開始し、一定数の WRITE ロックの後に READ ロックを設定する。
- SQL コマンド SET LOW_PRIORITY_UPDATES=1 を使用すると、特定のスレッドからの更新すべてが低い優先度で実行されるように指定できる。 See 項5.5.6.「SET 構文」。
- HIGH_PRIORITY 属性を使用すると、特定の SELECT の重要度を高く指定できる。 See 項6.4.1.「SELECT 構文」。
- SELECT と結合した INSERT に問題がある場合は、SELECT ステートメントと INSERT ステートメントの同時サポートが可能になるため、新規の MyISAM テーブルを使用するように切り替える。
- INSERT ステートメントと SELECT ステートメントの混在が多い場合、INSERT の DELAYED 属性によって問題が解決される確率が高い。 See 項6.4.3.「INSERT 構文」。
- SELECT と DELETE に問題がある場合、DELETE に LIMIT オプションを使用すると解決できる場合がある。 See 項6.4.5.「DELETE 構文」。

5.4. データベース構造の最適化

5.4.1. 設計上の選択

MySQL はローデータとインデックスデータを別のファイルに格納します。その他のデータベースの多く (ほとんど) は、ローデータとインデックスデータが同じファイルに混在しています。現在の非常に多くのシステムで MySQL の選択のほうが優れていると確信しています。

ローデータの格納方法には、各カラムの情報を独立した領域に格納する方法もあります（例: SDBM、Focus など）。これは、複数のカラムにアクセスするすべてのクエリでパフォーマンスに影響を及ぼします。パフォーマンスは複数のコラムへのアクセスを開始するとただちに低速化するため、このようなモデルは汎用データベースには適さないと確信しています。

一般的にインデックスとデータが一緒に格納されている場合も多くあります（Oracle、Sybase などの場合）。この場合は、レコード情報をインデックスのリーフページで検索します。このレイアウトで優れている点は、多くの場合インデックスのキャッシュ方法次第でディスクの読み取りを節約できることにあります。このレイアウトの欠点は以下のとおりです。

- データの取得時にインデックス全体を読み取る必要があるため、テーブルスキャンの速度が大幅に下がる。
- クエリでデータを取り出す際にインデックステーブルのみの使用ができない。
- ノードからインデックスを複製する必要があるため（レコードはノードに格納できないことによる）、大量の領域が消費される。
- 削除があるとテーブルの速度が次第に低下する（通常、削除ではノードのインデックスが更新されないため）。
- インデックスデータのみのキャッシュが困難である。

5.4.2. データの小型化

最も基本的な最適化の 1 つにデータ（およびインデックス）が占めるディスク領域を可能な限り少なくすることがあります。これで、ディスクの読み取りが高速化し、使用メモリも一般に減少するため、大幅な改善が図れます。カラムが小さければインデックス作成で消費されるリソースも少なくなります。

MySQL では多様なテーブル型とレコード形式がサポートされます。 適切なテーブル形式を選択することで、パフォーマンスを大幅に改善できます。 See 章 7. MySQL のテーブル型。

ここで紹介する技法を使用すると、テーブルのパフォーマンス改善とストレージ領域の最小化を図ることができます。

- できる限り効率性の高い（最小）の型を使用する。MySQL にはディスク領域とメモリを節約できる専用の型がある。
- 可能な場合は、小さなテーブルの取得には小さな整数型を使用する。たとえば、INT より、MEDIUMINT のほうが適している場合もしばしばある。
- できる限り、カラムに NOT NULL を宣言する。これですべてが高速化され、1 カラム当たり 1 ビットを節約できる。アプリケーションで実際に NULL が必要な場合は、必ず使用する必要があるため、注意が必要である。デフォルトですべてのカラムにこれを設定することは避ける。
- 可変長カラム（VARCHAR、TEXT、BLOB など）がまったくない場合は固定長レコード形式を使用する。これで速度が上がるが、領域の消費も増える。 See 項7.1.2.「MyISAM テーブル形式」
- テーブルのプライマリインデックスを可能な限り短くする。これで、レコードの識別が容易になり効率化が図れる。
- テーブルごとに使用するストレージとインデックスの方法を設定する必要がある。 See 章 7. MySQL のテーブル型。
- インデックスの作成は必要なものだけに限定する。インデックスは取り出しに優れているが、高速保存が必要な場合は

適さない。カラムの組み合わせを使用してテーブルを検索し、テーブルにアクセスする場合がほとんどであれば、インデックスを作成する。インデックスの最初の部分は、最も使用頻度の高いカラムにする必要がある。常に多数のカラムを使用する場合は、より重複しているカラムを先に使用するとインデックスの圧縮を改善できる。

- 文字列の最初の数文字に、一意のプリフィックスがあるカラムが多い場合は、このプリフィックスのみをインデックス化したほうがよい。MySQL は CHAR 型カラムの部分インデックスをサポートする。短いインデックスの速度が速い理由は、占有ディスク領域が小さいことだけではなく、インデックスキャッシュでのヒットが多くなり、所要ディスクシークが少なくなることにもよる。 See 項5.5.2.「サーバパラメータのチューニング」。
- 状況によっては、スキャンの頻度が高いテーブルを 2 つに分割したほうが有利な場合もある。これは特に、動的テーブルで、テーブルスキャンの際に対応するレコードの検索に小さな静的テーブルの使用が可能である場合にあてはまる。

5.4.3. MySQL でのインデックスの使用

インデックスは、カラムが特定の値をもつレコードの迅速な検索に使用されます。インデックスがないと、MySQL がレコードを見つけるために、最初のレコードから開始し、テーブル全体を読み取るとが必要になります。テーブルが大きくなると、これにコストがかかります。クエリ対象のカラムにインデックスがあると、MySQL は全てのデータを探すことなく、データファイルの途中にあるシーク対象ポジションを迅速に取得することができます。テーブルに 1000 レコードある場合、シーケンシャルに読み取る場合と比較して少なくとも 100 倍は高速化できます。1000 レコードのほとんどすべてにアクセスする必要がある場合は、ディスクシークが最小になるため、シーケンシャルに読むほうが速くなることに注意してください。

MySQL インデックスのすべて (PRIMARY KEY、UNIQUE、および INDEX) は、B ツリーに格納されます。文字列の頭にある空白と最後にある空白は自動的に圧縮されます。 See 項6.5.7.「CREATE INDEX 構文」。

インデックスの用途は以下のとおりです。

- WHERE 節の条件に一致するレコードを迅速に検索する。
- 結合実行時に他のテーブルのレコードを取り出す。
- インデックス化された特定のカラムの MAX() 値や MIN() 値を検索する。これは、WHERE key_part_# = すべてのキー部分の定数 < N を使用しているかどうかをチェックするプリプロセッサによって最適化される。この場合、MySQL では単一キーのルックアップを実行し、MIN() 式を定数に置換する。すべての式が定数に置換されると、ただちにクエリの応答が返される。

```
SELECT MIN(key_part2),MAX(key_part2) FROM table_name WHERE key_part1=10
```

- 使用可能キーの左端の先頭部分でソートやグループ化が実行されている場合、テーブルのソートやグループ化を実行する (例: ORDER BY key_part_1,key_part_2) 。すべてのキー部分の後ろに DESC がある場合は、キーが逆の順序で読み取られる。 See 項5.2.8.「MySQL による ORDER BY の最適化」。
- 状況によっては、データファイルを参照せずに値を取り出すようにクエリを最適化できる。あるテーブルで使用カラムのすべてが数値型で、キーの左端の先頭部分を構成している場合は、インデックスツリーから非常に高速に値を取り出すことができる。

```
SELECT key_part3 FROM table_name WHERE key_part1=1
```

次の SELECT ステートメントを指定したとします。

```
mysql> SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;
```

col1 と col2 に複合インデックスが存在する場合、対応するレコードを直接読み取れます。col1 と col2 に独立した単一カラムインデックスが存在する場合、検索されるレコードの少ないインデックスを判定し、そのインデックスをレコードの読み取りに使用して、最も制限性の高いインデックスの検索が試行されます。

テーブルに複合インデックスがある場合、オプティマイザではインデックスの左端の先頭部分のいずれかをレコードの検索に使用できます。たとえば、(col1, col2, col3) に 3 カラムのインデックスがある場合、(col1)、(col1, col2)、および (col1, col2, col3) に対して、インデックスの検索機能を使用できます。

カラムがインデックスの左端の先頭部分を構成していない場合、MySQL では、部分インデックスを使用できなくなります。以下の SELECT ステートメントがあるとします。

```
mysql> SELECT * FROM tbl_name WHERE col1=val1;
mysql> SELECT * FROM tbl_name WHERE col2=val2;
mysql> SELECT * FROM tbl_name WHERE col2=val2 AND col3=val3;
```

インデックスが (col1, col2, col3) に存在する場合、最初のクエリだけがインデックスを使用できます。2 つめと 3 つめのクエリには、インデックス化したカラムが必要ですが、(col2) と (col2, col3) は (col1, col2, col3) の左端のプリフィックスではありません

MySQL は、LIKE の引数がワイルドカード文字で始まらない文字列定数である場合に、LIKE 比較にもインデックスを使用します。たとえば、以下の SELECT ステートメントではインデックスが使用されます。

```
mysql> SELECT * FROM tbl_name WHERE key_col LIKE "Patrick%";
mysql> SELECT * FROM tbl_name WHERE key_col LIKE "Pat%_ck%";
```

最初のステートメントでは "Patrick" <= key_col < "Patricl" のあるレコードだけが考慮されます。2 つめのステートメントでは "Pat" <= key_col < "Pau" のあるレコードだけが考慮されます。

以下の SELECT ステートメントではインデックスが使用されません。

```
mysql> SELECT * FROM tbl_name WHERE key_col LIKE "%Patrick%";
mysql> SELECT * FROM tbl_name WHERE key_col LIKE other_col;
```

最初のステートメントでは LIKE がワイルドカード文字で始まっています。2 つめのステートメントでは LIKE 値が定数ではありません。

MySQL 4.0 ではこれ以外の LIKE の最適化も実行されます。... LIKE "%string%" を使用し、string が 3 文字より長い場合、MySQL は Turbo Boyer-Moore アルゴリズムを使用して、文字列のパターンを初期化してから、このパターンを使用して検索を素早く実行します。

column_name IS NULL を使用した検索では、column_name にインデックスが張られている場合にインデックスが使用されます。

通常 MySQL は、検索するレコードを少なくするために、インデックスを使用します。インデックスは、以下の演算子で比較するカラムに使用されます。 =、>、>=、<、<=、BETWEEN、または、'something%' などのワイルドカード以外のプリフィックスで始まるパターンに対する LIKE。

WHERE 節内の全ての AND にかかっていないインデックスは、クエリの最適化に使用されません。言い換えると、インデックスの使用を可能にするには、インデックスの先頭部分がすべての AND グループで使用されている必要があります。

次の WHERE 節ではインデックスが使用されます。

```
... 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
        /* optimized like "index_part1='hello'" */
... WHERE index1=1 AND index2=2 OR index1=3 AND index3=3;
        /* Can use index on index1 but not on index2 or index 3 */
```

次の WHERE 節ではインデックスが使用されません。

```
... WHERE index_part2=1 AND index_part3=2  /* index_part_1 is not used */
... WHERE index=1 OR A=10                  /* Index is not used in
                                           both AND parts */
... WHERE index_part1=1 OR index_part2=10  /* No index spans all rows  */
```

MySQL では利用可能な場合でもインデックスが使用されない場合があることに注意してください。この一例として、インデックスの使用によって、MySQL がテーブルの 30% を超えるレコードにアクセスする必要が生じる場合が挙げられます（この場合は、必要なシークが大幅に減少するため、テーブルスキャンのほうが高速になる可能性が高くなります）。ただしこのクエリに、レコードの一部のみを取り出す LIMIT が使用されている場合、結果で返される少数のレコードを迅速に検索できるため、MySQL はインデックスを使用します。

5.4.4. カラムインデックス

MySQL の全てのカラム型にはインデックスを張ることができます。SELECT 操作のパフォーマンスの改善には、対応するカラムにインデックスを使用することが最善の方法です。

テーブルあたりの最大インデックス数とインデックスの最大長は、ストレージエンジンごとに定義されます。See 章 7. MySQL のテーブル型。ストレージエンジンのすべてで、1 テーブルあたり 16 以上のインデックスと 256 バイト以上のインデックス長がサポートされます。

CHAR 型および VARCHAR 型のカラムでは、カラムの先頭部分をインデックス化できます。これは、カラム全体をインデックス化する場合と比較して大幅に高速になり、所要ディスク領域も少なくて済みます。カラムの先頭部分をインデックス化する CREATE TABLE ステートメント構文は次のようになります。

```
INDEX index_name (col_name(length))
```

この例では、name カラムの最初の 10 文字のインデックスが作成されます。

```
mysql> CREATE TABLE test (
    ->      name CHAR(200) NOT NULL,
    ->      INDEX index_name (name(10)));
```

BLOB 型および TEXT 型のカラムでは、カラムの先頭部分をインデックス化する必要があります。インデックスが張れる部分の最大長は 255 バイトです。

MySQL バージョン 3.23.23 以降は、特殊な FULLTEXT インデックスも作成できます。これは全文検索に使用されます。FULLTEXT インデックスは、MyISAM テーブル型でのみ、CHAR、VARCHAR、および TEXT カラムに限ってサポートさ

れます。 フルテキストインデックスの作成は常にカラム全体を対象として、先頭部分（プリフィックス）のインデックス化は行われません。詳細については、項6.8.「MySQL 全文検索」 を参照してください。

5.4.5. 複合インデックス

MySQL では複数のカラムに対するインデックスを作成できます。インデックスは最大 16 カラムで構成できます（CHAR および VARCHAR カラムではカラムの先頭部分をインデックスの部分として使用することもできます）。

複数カラムのインデックス（複合インデックス）は、インデックス化されたカラムの値を連結することによって生成された値が含まれ、ソート化された配列と見なすことができます。

MySQL では、WHERE 節内でインデックスの第 1 カラムを指定する場合、他のカラムの値を指定しなくても、クエリが高速化できるように複合インデックスが使用されます。

次のようなテーブルが定義されているとします。

```
mysql> CREATE TABLE test (
    ->       id INT NOT NULL,
    ->       last_name CHAR(30) NOT NULL,
    ->       first_name CHAR(30) NOT NULL,
    ->       PRIMARY KEY (id),
    ->       INDEX name (last_name,first_name));
```

ここで、インデックス name は、last_name と first_name に対するインデックスです。このインデックスは、last_name の範囲、または last_name と first_name の両方の範囲の値を指定するクエリに使用できます。 したがって、name インデックスは次のようなクエリに使用されます。

```
mysql> SELECT * FROM test WHERE last_name="Widenius";

mysql> SELECT * FROM test WHERE last_name="Widenius"
    ->                 AND first_name="Michael";

mysql> SELECT * FROM test WHERE last_name="Widenius"
    ->                 AND (first_name="Michael" OR first_name="Monty");

mysql> SELECT * FROM test WHERE last_name="Widenius"
    ->                 AND first_name >="M" AND first_name < "N";
```

しかし、次のクエリには name インデックスが使用されません。

```
mysql> SELECT * FROM test WHERE first_name="Michael";

mysql> SELECT * FROM test WHERE last_name="Widenius"
    ->                 OR first_name="Michael";
```

MySQL でインデックスを使用してクエリパフォーマンスを改善する方法の詳細については、項5.4.3.「MySQL でのインデックスの使用」を参照してください。

5.4.6. MySQL のオープンテーブルのカウント方法

mysqladmin status を実行すると、以下の出力が表示されます。

```
Uptime: 426 Running threads: 1 Questions: 11082 Reloads: 1 Open tables: 12
```

テーブルが 6 つしかない場合に Open tables 値が 12 と表示されることに、当惑する場合もあります。

MySQL はマルチスレッド化されているため、多数のクライアントが同時に同じものに対してクエリを使用することがあります。2 つのクライアントスレッドで 1 つのファイルに異なるステータスが発生する問題を最小にするため、同時に実行しているスレッドがそれぞれで無関係にテーブルを開きます。これはメモリの消費を増やしますが、一般にパフォーマンスは向上します。ISAM テーブルと MyISAM テーブルの場合は、テーブルを開いたそれぞれのクライアントにデータファイルに対するファイル記述子が必要になります。このテーブル型では、インデックスファイルに対するファイル記述子がすべてのスレッドで共有されます。

次のセクションでもこのトピックについてさらに説明します。 See 項5.4.7.「MySQL でのテーブルのオープンとクローズの方法」。

5.4.7. MySQL でのテーブルのオープンとクローズの方法

table_cache、max_connections、および max_tmp_tables サーバ変数は、サーバが開いた状態で保持できるファイルの最大数に影響します。 これらの値の 1 つ以上を増加すると、OS によって制限されている 1 プロセスが持つことができるファイル記述子の最大数まで実行が可能になります。システムごとに方法は多様ですが、多数のオペレーティングシステムでオープンファイルの制限値を上げることができます。 制限値の拡大が可能かどうかの判定、およびその実行方法については、使用するオペレーティングシステムの文書を参照してください。

table_cache は max_connections と関係します。たとえば、同時接続数が 200 の場合、最低 200 * n のテーブルキャッシュサイズが必要です。この n は結合で使用するテーブル数の最大値を示します。また、テンポラリテーブルとファイル用のファイル記述子も必要です。

あなたのオペレーティングシステムが table_cache の設定に従ったファイル記述子の数を処理できることを確認してください。table_cache の設定が高すぎると、MySQL がファイル記述子を使い果たして接続を拒否し、クエリの実行ができなくなり、信頼性が大幅に低下します。また、MyISAM ストレージエンジンでは1つのテーブルごとに 2 つのファイル記述子が必要であることも考慮に入れる必要があります。--open-files-limit=# スタートアップオプションを使用すると、MySQL で使用可能なファイル記述子数を拡大できます。 See 項A.2.17.「File Not Found エラー」。

オープンテーブルのキャッシュは、table_cache エントリレベルに保持されます。デフォルト値は 64 です。これは、-O table_cache=# オプション mysqld に与えることで変更できます。MySQL は一時的にさらに多くのテーブルを開いてクエリの実行を実現することがあります。

以下の状況では、使用されていないテーブルが閉じられ、テーブルキャッシュから削除されます。

- キャッシュが満杯のときに、キャッシュにないテーブルをスレッドが開こうとした場合。
- キャッシュに table_cache を超えるエントリがあり、あるスレッドがテーブルの使用を終えた場合。
- いずれかのユーザが mysqladmin refresh または mysqladmin flush-tables を実行した場合。
- いずれかのユーザが FLUSH TABLES ステートメントを実行した場合。

テーブルキャッシュが満杯になると、サーバでは以下の手順に従って使用するキャッシュエントリを割り当てます。

- 現在使用中でないテーブルは、最後に使用した時が古いものから順にリリースされる。

- キャッシュが満杯でリリース可能なテーブルがなく、新たにテーブルを開く必要がある場合は、必要に応じてキャッシュが一時的に拡張される。
- キャッシュが一時的に拡張された状況で、使用中のテーブルが使用されなくなったときは、そのテーブルが閉じられ、キャッシュからリリースされる。

テーブルは同時アクセスのそれぞれで開かれます。つまり、2 つのスレッドで同じテーブルにアクセスする場合、または 1 つのスレッドが同一クエリでテーブルに 2 回アクセスする場合 (テーブルを同一テーブルに結合する場合など) は、テーブル を 2 回開く必要があることになります。 いずれかのテーブルを最初に開く際に 2 つのファイル記述子が割り当てられ、その後さらにそのテーブルを使用する場合はファイル記述子が 1 つのみ割り当てられます。最初のオープン時の 2 つめの記述子は、インデックスファイルに使用され、この記述子はすべてのスレッドで共有されます。

HANDLER table_name OPEN ステートメントを使用してテーブルを開く場合は、専用テーブルオブジェクトがスレッドに割り当てられます。 このテーブルオブジェクトは他のスレッドと共有されず、スレッドが HANDLER table_name CLOSE を呼び出すか、スレッドが終了するまで閉じられません。 See 項6.4.9.「HANDLER 構文」。 この場合はテーブルがテーブルキャッシュに戻されます (キャッシュが満杯でない場合)。

テーブルキャッシュが小さすぎるかどうかは、mysqld の Opened_tables 変数のチェックで確認できます。たとえ多くの FLUSH TABLES を実行していない場合でも、この値が非常に大きい場合は、テーブルキャッシュサイズを拡張する必要があります。 See 項4.6.8.3.「SHOW STATUS」。

5.4.8. 1 つのデータベースに大量のテーブルを作成した場合の欠点

ディレクトリにファイルが多数ある場合、オープン、クローズ、および作成の動作が低速になります。多数のテーブルに対して SELECT ステートメントを実行した場合、必要なテーブルを開くごとに、他のテーブルを閉じることが必要になるため、テーブルキャッシュが満杯の場合にオーバヘッドが少し発生します。このオーバヘッドは、テーブルキャッシュを拡大することで軽減できます。

5.5. MySQL サーバの最適化

5.5.1. システム、コンパイル時間およびスタートアップパラメータのチューニング

システムレベルの要素は、その一部を初期段階に決定する必要があるため、この話から始めます。これに該当しない場合は、システムを大きく変えることが重要でないのであれば、このセクションは簡単に目を通せば十分です。ただし、このレベルで変更を行うことでどの程度改善できるのかを自覚しておくことは必ず役に立ちます。

使用するオペレーティングシステムは非常に重要です。複数 CPU のコンピュータを使用するなら、Solaris (スレッド実装機能が優れている) または Linux (2.2 カーネルの SMP サポートが優れている) が良いでしょう。 また、旧バージョンの Linux カーネルのデフォルトには 2G ファイルサイズの制限があります。このカーネルで 2G より大きいファイルがどうしても必要な場合は、ext2 ファイルシステムの LFS (Large File System) パッチを入手する必要があります。 これ以外の ReiserFS や XFS などには 2G の制限がありません。

多くのプラットフォーム上で、MySQL を本番稼働させていないため、可能であれば選択前に候補のプラットフォームのテストを実行することを推奨します。

その他のヒント:

- RAM が十分にある場合は、スワップデバイスすべてを削除できる。オペレーティングシステムによっては、空きメモリがある場合でもスワップデバイスが使用されることがある。
- --skip-external-locking MySQL オプションを使用して、外部ロックを回避する。実行しているのが 1 サーバだけである限り、これによる MySQL の機能に対する影響はない。myisamchk を実行する前にサーバの記録を取る（または対応するテーブルをロックし、フラッシュする）ことを忘れないようにする。一部のシステムは、外部ロックがまったく機能しないため、このオプションが必須になる。

  MySQL 4.0 以降、--skip-external-locking オプションはデフォルトでオンになっている。 それ以前は、MIT-pthread によるコンパイル時にデフォルトでオンになっている。これは flock() がすべてのプラットフォームで MIT-pthread により完全にサポートされているわけではないことによる。Linux ファイルロックは安全ではないため、Linux でもデフォルトでオンになっている。

  --skip-external-locking を使用できない状況は、同一データに対して複数の MySQL サーバ（クライアントではない）を実行している場合と、サーバに対して初めにテーブルのフラッシュとロックを行う指示を出さずに、テーブルに対して myisamchk を実行する場合に限られる。

  --skip-external-locking を使用している場合でもLOCK TABLES/UNLOCK TABLES は使用できる。

5.5.2. サーバパラメータのチューニング

mysqld サーバで使用されるデフォルトのバッファサイズは次のコマンドで確認できます。

```
shell> mysqld --help
```

このコマンドによって、mysqld オプションと設定可能な変数すべての一覧が生成されます。この出力には、デフォルトの変数値も記載され、以下のように表示されます。

```
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
myisam_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
read_buffer_size        current value: 131072
read_rnd_buffer_size    current value: 262144
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
```

現在実行中の mysqld サーバがある場合は、次のステートメントで変数に実際に使用されている値を調べることができます。

```
mysql> SHOW VARIABLES;
```

また、次のステートメントでは、実行中のサーバの統計やステータスインジケータを調べることができます。

```
mysql> SHOW STATUS;
```

すべての変数の詳細説明については、本マニュアルの SHOW VARIABLES セクションを参照してください。 See 項4.6.8.4.「SHOW VARIABLES」。 ステータス変数詳細については、項4.6.8.3.「SHOW STATUS」 を参照してください。

サーバ変数とステータス情報は、mysqladmin でも入手できます。

```
shell> mysqladmin variables
shell> mysqladmin extended-status
```

MySQL は非常にスケーラブルなアルゴリズムを使用しているため、通常は実行時のメモリ消費が非常に小さくなります。しかし、MySQL に対するメモリを多く割り当てると、通常はパフォーマンスが向上します。

MySQL サーバをチューニングする際に使用される最も重要な変数は key_buffer_size と table_cache の 2 つです。他の変数の変更を行う前にこの変数をあらかじめ適切に設定しておくことで自信がつきます。

以下に典型的な変数を実行時に設定している例を示します。この例は mysqld_safe スクリプトを使用し、--name=value 構文で変数 name を値 value に設定しています。この構文は、MySQL 4.0 から利用できます。旧バージョンの MySQL の場合は、以下の相違点を考慮してください。

- mysqld_safe ではなく、safe_mysqld を使用する。
- --set-variable=name=value または -O name=value 構文を使用して変数を設定する。
- _size で終わる変数名は _size なしでの指定が必要な場合がある。たとえば、sort_buffer_size の旧名は sort_buffer である。read_buffer_size の旧名は record_buffer である。サーババージョンで認識される変数を調べるときは mysqld --help を使用する。

最小 256M のメモリで多数のテーブルがあり、中程度のクライアントで最大のパフォーマンスを得るには、次のように使用します。

```
shell> mysqld_safe --key_buffer_size=64M --table_cache=256 \
        --sort_buffer_size=4M --read_buffer_size=1M &
```

メモリが 128M で、テーブルは少数で大量のソートの実行が必要な場合は、次のように使用できます。

```
shell> mysqld_safe --key_buffer_size=16M --sort_buffer_size=1M
```

メモリがほとんどなく大量の接続がある場合は、次のように使用します。

```
shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=100K \
        --read_buffer_size=100K &
```

また、次のようにもできます。

```
shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=16K \
        --table_cache=32 --read_buffer_size=8K -O net_buffer_length=1K &
```

使用可能メモリより大幅に大きいテーブルで GROUP BY または ORDER BY を実行する場合は read_rnd_buffer_size の値を大きくしてソート操作後のレコードの読み取りの速度を上げる必要があります。

MySQL をインストールしたときは、support-files ディレクトリに複数の my.cnf サンプルファイルの、my-huge.cnf、my-large.cnf、my-medium.cnf、および my-small.cnf が格納され、システム最適化のベースとして使用できます。

同時接続が非常に多い場合、接続ごとに mysqld で使用されるメモリを非常に小さくしていないとスワップの問題が発生することがあります。言うまでもなく、すべての接続に使用可能なメモリが十分ある場合は mysqld のパフォーマンスが向上します。

mysqld または mysqld_safe のコマンドラインでオプションを指定した場合、そのサーバの呼び出しでしか有効性が保持されないことに注意してください。 サーバ実行のたびにオプションを使用する場合は、オプション設定ファイルに配置します。

パラメータ変更の有効性を調べるには、次のように実行します。

```
shell> mysqld --key_buffer_size=32m --help
```

必ず --help を最後に指定します。最後にしないと、コマンドラインのそれ以降に記載されたオプションの効果が出力に反映されません。

5.5.3. MySQL の速度に対するコンパイルとリンクの影響

以下のテストのほとんどは、MySQL ベンチマークを使用した Linux で実行されていますが、これ以外のオペレーティングシステムおよびワークロードに対しても一定の指針になります。

-static とリンクした場合に最速のバイナリが得られます。

Linux 上では、pgcc および -O3 でコンパイルした場合に最速のコードが得られます。これらのオプションで sql_yacc.cc をコンパイルする場合は、gcc/pgcc で関数のすべてをインラインにする際に大量のメモリが要求されるため約 200M のメモリが必要です。MySQL のコンフィギャ時に CXX=gcc も設定して、libstdc++ ライブラリ（これは不要です）が含まれないようにします。pgcc の一部のバージョンでは、生成されたコードを x586 タイプのプロセッサ（AMD など）すべてで動作可能にするコンパイラオプションを使用しても、コードが純正 Pentium プロセッサでしか実行できないため注意が必要

です。

適切なコンパイラおよびコンパイラオプションを使用することで、アプリケーションの速度が 10−30% 改善されます。これは各自で SQL サーバをコンパイルする場合に特に重要です。

Cygnus CodeFusion と Fujitsu コンパイラの両方をテストしましたが、いずれもバグフリーではなく、最適化をオンにして MySQL をコンパイルするには不十分でした。

MySQL のコンパイル時は、使用するキャラクタセットのサポートのみを含めます（オプション --with-charset=xxx）。 標準の MySQL バイナリディストリビューションは、すべてのキャラクタセットをサポートするようにコンパイルされています。

以下に実施した測定結果の一部を紹介します。

- pgcc を使用し、すべてを -O6 でコンパイルした場合、mysqld サーバは gcc 2.95.2 と比較して 1% 速度が上がる。
- 動的にリンクした場合（-static なし）は、結果が Linux 上で 13% 遅くなった。クライアントアプリケーションには動的リンクの MySQL ライブラリを使用できることに注意する。これは、サーバのパフォーマンス上重大である。
- strip libexec/mysqld を使用して mysqld バイナリをストリップすると、生成されたバイナリの速度を 4% まで上げられる。
- 同一ホスト上で実行されるクライアントからサーバへの接続で、Unix ソケットファイルではなく、TCP/IP で接続すると、7.5% パフォーマンスが遅くなった（ホスト名 localhost に接続する場合、MySQL ではデフォルトでソケットファイルが使用される）。
- クライアントからサーバへの TCP/IP 接続で別のホストにあるリモートサーバに接続した場合、100M イーサネットによる接続でも、同一ホスト上のローカルサーバに接続した場合と比較して、8-11% 遅くなった。
- 暗号化した接続（内部 SSL サポートによるすべてのデータの暗号化）を使用してベンチマークテストを実行した場合、パフォーマンスが 55% 遅くなった。
- --with-debug=full でコンパイルすると、ほとんどのクエリが 20% 遅くなる。 一部のクエリはかなり長くかかった（たとえば MySQL ベンチマークは 35% の速度低下）。--with-debug を使用すると、この速度低下は 15% で済む。--with-debug=full でコンパイルされた mysqld バージョンは、--skip-safemalloc オプションで起動すると実行時のメモリチェックを無効化できる。この場合の最終的な結果は、--with-debug で構成した場合に非常に近くなる。
- Sun UltraSPARC-IIe, Forte 5.0 は、gcc 3.2 より 4% 速度が上がった。
- Sun UltraSPARC-IIe, Forte 5.0 では、64 ビットモードより 32 ビットモードのほうが 4% 速かった。
- gcc 2.95.2 for UltraSPARC にオプション -mcpu=v8 -Wa,-xarch=v8plusa を付けてコンパイルすると、パフォーマンスが 4% 改善した。
- Solaris 2.5.1, MIT-pthreads は、単一プロセッサ上で Solaris ネイティブスレッドより 8-12% 遅かった。CPU の負荷が増加するとこの差はさらに拡大する。
- --log-bin を使用して実行すると mysqld が 1% 遅くなった。
- フレームポインタ -fomit-frame-pointer または -fomit-frame-pointer -ffixed-ebp なしで gcc を使用して Linux-x86 でコンパイルすると、mysqld が 1-4% 速くなった。

pgcc によるコンパイルに MySQL AB 提供の MySQL-Linux ディストリビューションを使用したが、AMD で実行されないコードを生成するバグが pgcc にあったため、通常の gcc の使用に戻さざるを得ませんでした。このバグが解決されるまで gcc の使用を続行します。 ただし、AMD 以外のコンピュータを使用する場合は、pgcc でコンパイルすると高速なバイナリが得られます。標準の MySQL Linux バイナリは、速度および移植性を高めるため静的にリンクされています。

5.5.4. MySQL でのメモリの使用

以下の一覧は、mysqld サーバでのメモリの使用方法の一部を示しています。可能な場合は、メモリ使用に関連するサーバ変数名も記載されています。

- キーバッファ（変数 key_buffer_size ）はすべてのスレッドで共有される。サーバが使用するこれ以外のバッファは必要に応じて割り当てられる。 See 項5.5.2.「サーバパラメータのチューニング」。
- それぞれの接続は、スタック（デフォルト 64K、変数 thread_stack ）、接続バッファ（変数 net_buffer_length ）、および結果バッファ（ net_buffer_length ）のスレッド固有領域を使用する。接続バッファと結果バッファは必要に応じて max_allowed_packet まで動的に拡張される。クエリの実行中は現在のクエリ文字列のコピーも割り当てられる。
- すべてのスレッドで同じベースメモリが共有される。
- 圧縮 ISAM および MyISAM テーブルのみがメモリにマップされる。これは、4 GB の 32 ビットメモリ領域では大型のほとんどのテーブルに十分なほどは大きくないことによる。64 ビットアドレス領域のあるシステムが一般的になれば、メモリマップの一般サポートの追加が可能になる。
- テーブルの順次スキャンを行う要求はそれぞれ、読み取りバッファ（変数 read_buffer_size ）を割り当てる。
- レコードを ``ランダムな" 順序で読み取る場合（ソート後など）、ランダム読み取りバッファが割り当てられディスクシークが回避される（変数 read_rnd_buffer_size ）。
- 結合はすべて 1 回の受け渡しで実行され、ほとんどの結合はテンポラリテーブルを使用せずに実行される。テンポラリテーブルのほとんどはメモリベース（ HEAP ）テーブルである。レコード長の大きなテンポラリテーブル（すべてのカラム長の合計として算出）や BLOB カラムが含まれるテンポラリテーブルはディスク上に格納される。

  バージョン 3.23.2 より前の MySQL には、メモリ内の HEAP テーブルが tmp_table_size のサイズを超えた場合にエラー The table tbl_name is full が出力される問題があった。3.23.2 以降、この問題は必要に応じてメモリ内 HEAP テーブルをディスクベース MyISAM テーブルに変更されることで自動的に処理される。この問題を回避するには、tmp_table_size オプションを mysqld に設定するか、クライアントプログラムで SQL オプション BIG_TABLES を設定することで、テンポラリテーブルのサイズを拡張する。See 項5.5.6.「SET 構文」。MySQL バージョン 3.20 では、テンポラリテーブルの最大サイズが record_buffer*16 であった。このバージョンを使用している場合は、record_buffer の値を拡大する必要がある。また、--big-tables オプションで mysqld を起動して、常にテンポラリテーブルをディスクに格納することもできる。ただし、これは複雑なクエリのほとんどで処理速度に影響を及ぼす。
- ソートを実行する要求のほとんどで、ソートバッファおよび結果セットサイズに応じた 0 から 2 つのテンポラリファイルが割り当てられる。 See 項A.4.4.「MySQL がテンポラリファイルを格納する場所」。
- 解析および計算のほとんどすべてが、ローカルメモリストアで実行される。小さいアイテムにはメモリオーバヘッドが不要で、通常の低速メモリの割り当ておよび解放は回避される。メモリは、予測外の規模の文字列の場合のみ割り当てられ、これは、malloc() および free() で実行される。
- インデックスファイルはそれぞれ 1 回開かれ、データファイルは、同時実行スレッドごとに 1 回開かれる。同時スレ

ッドのそれぞれに対して、テーブル構造、各カラムのカラム構造、サイズ 3 * n のバッファが割り当てられる (n は、レコードの最大長、ただし BLOB カラムは計算外) 。BLOB カラムは、5 から 8 バイトに BLOB データの長さを加算したバイト数を使用する。ISAM および MyISAM ストレージエンジンは、内部使用のための追加レコードを 1 つ使用する。

- BLOB カラムがあるテーブルのそれぞれで、大きな BLOB 値を読み込むためにバッファが動的に拡張される。テーブルをスキャンする場合は、最大 BLOB 値と同じ大きさのバッファが割り当てられる。
- 使用中テーブルすべてのハンドラ構造がキャッシュに保存され、FIFO 形式で管理される。一般にキャッシュには 64 のエントリがある。テーブルが同時に 2 つの実行スレッドで使用されている場合、キャッシュにはそのテーブルのエントリが 2 つ配置される。 See 項5.4.7.「MySQL でのテーブルのオープンとクローズの方法」。
- mysqladmin flush-tables コマンド (または FLUSH TABLES ステートメント) によって、使用中でないテーブルすべてが閉じられ、現在実行中のスレッドの終了時に使用中のテーブルすべてが閉じられるように指定される。これで効率的に使用中メモリに空きを作ることができる。

ps およびその他のステータスプログラムによって、mysqld が大量のメモリを使用していることを示すレポートが行われることがあります。これは、複数のメモリアドレスでのスレッドスタックによって発生します。たとえば、Solaris バージョンの ps ではスタック間の使用していないメモリが使用メモリにカウントされます。これは、swap -s で使用可能スワップをチェックすることで検証できます。市販のメモリリーク検出装置で mysqld をテストし、メモリリークがないと判明しています。

5.5.5. MySQL の DNS の使用

新たなクライアントが mysqld に接続すると、mysqld によって要求を処理する新規のスレッドが作成されます。このスレッドでは、まずホスト名がホスト名キャッシュにあるかどうかがチェックされます。ない場合は、ホスト名の解決が試行されます。

- オペレーティングシステムがスレッドセーフの gethostbyaddr_r() と gethostbyname_r() の呼び出しをサポートしている場合、スレッドではこれを使用してホスト名の解決が実行される。
- オペレーティングシステムがスレッドセーフの呼び出しをサポートしていない場合、スレッドでは相互排除ロックを行い、代わりに gethostbyaddr() と gethostbyname() が呼び出される。この場合、他のスレッドでは最初のスレッドが相互排除ロックを解除するまでホスト名キャッシュ内のホスト名を解決できなくなることに注意する。

--skip-name-resolve を mysqld オプションを指定して起動すると、DNS ホスト名ルックアップを無効化できます。ただし、この場合は、MySQL 権限テーブルで IP 番号しか使用できなくなります。

非常に低速の DNS と多数のホストがある場合は、--skip-name-resolve で DNS ルックアップを無効化するか、HOST_CACHE_SIZE の定義 (デフォルト値: 128) を拡張し、mysqld を再コンパイルすることで、パフォーマンスを改善できます。

--skip-host-cache オプションを使用してサーバを起動すると、ホスト名キャッシュを無効化できます。ホスト名のキャッシュをクリアするには、FLUSH HOSTS ステートメントを使用するか、mysqladmin flush-hosts コマンドを実行します。

TCP/IP 接続すべてを認めない場合は、--skip-networking オプションを指定して mysqld を開始します。

5.5.6. SET 構文

```
SET [GLOBAL | SESSION] sql_variable=expression,
  [[GLOBAL | SESSION] sql_variable=expression] ...
```

SET は、サーバやクライアントの動作に影響を及ぼすさまざまなオプションを設定します。

以下の例は、変数の設定に使用できる各種の構文を示しています。

旧バージョンの MySQL では、SET OPTION 構文の使用を許可していましたが、今は廃止されています。

MySQL 4.0.3 では、GLOBAL オプション、SESSION オプション、および最も重要なスタートアップ変数へのアクセスを追加しています。

LOCAL は、SESSION のシノニムとして使用できます。

1 つのコマンドラインに複数の変数を設定する場合は、最後の GLOBAL | SESSION モードが使用されます。

```
SET sort_buffer_size=10000;
SET @@local.sort_buffer_size=10000;
SET GLOBAL sort_buffer_size=1000000, SESSION sort_buffer_size=1000000;
SET @@sort_buffer_size=1000000;
SET @@global.sort_buffer_size=1000000, @@local.sort_buffer_size=1000000;
```

@@variable_name 構文は、MySQL 構文とその他のデータベースとの互換性を保持を目的にサポートされています。

このマニュアルのシステム変数のセクションに設定可能な多様なシステム変数に関する説明があります。 See 項6.1.5.「システム変数」。

SESSION (デフォルト) を使用している場合、現在のセッションを終了するまで、あるいはこのオプションに別の値を設定するまで、設定したオプションが有効になります。SUPER 特権を必要とする GLOBAL を使用した場合、サーバの再起動が行われるまでオプションが記憶され、新規接続時も使用されます。オプションを永続的にする場合は、オプション設定ファイルに設定します。 See 項4.1.2.「my.cnf オプション設定ファイル」。

不適切な使用を防ぐため、SET SESSION でしか使用できない変数とともに SET GLOBAL を使用した場合や、グローバル変数に SET GLOBAL を使用しない場合には MySQL からエラーが出力されます。

SESSION 変数を GLOBAL 値に、あるいは GLOBAL 値を MySQL のデフォルト値に設定する場合は、DEFAULT として設定することができます。

```
SET max_join_size=DEFAULT;
```

これは以下と等しいことになります。

```
SET @@session.max_join_size=@@global.max_join_size;
```

SET コマンドで設定可能なサーバ変数に最大値を設定して制限する場合、--maximum-variable-name コマンドラインオプションを使用して最大値を指定できます。 See 項4.1.1.「mysqld コマンドラインオプション」。

SHOW VARIABLES を使用すると、ほとんどの変数の一覧が出力されます。 See 項4.6.8.4.「SHOW VARIABLES」。
@@[global.|local.]variable_name 構文を使用すると特定の変数の値を取得できます。

```
SHOW VARIABLES like "max_join_size";
```

```
SHOW GLOBAL VARIABLES like "max_join_size";
SELECT @@max_join_size, @@global.max_join_size;
```

以下に、非標準 SET 構文を使用する変数およびその他の変数の一部について説明します。これ以外の変数定義は、システム変数セクションのスタートアップオプションの部分または SHOW VARIABLES の説明に記載されています。 See 項6.1.5.「システム変数」。 See 項4.1.1.「mysqld コマンドラインオプション」。 See 項4.6.8.4.「SHOW VARIABLES」。

- AUTOCOMMIT= 0 | 1

  1 に設定すると、テーブルに対する変更すべてがただちに実行される。トランザクションを有効にする場合は、BEGIN ステートメントを使用する必要がある。See 項6.7.1.「START TRANSACTION、COMMIT、ROLLBACK の各構文」。0 に設定した場合は、そのトランザクションを COMMIT で受け入れるか、ROLLBACK で取り消す必要がある。 See 項6.7.1.「START TRANSACTION、COMMIT、ROLLBACK の各構文」。 AUTOCOMMIT モードを 0 から 1 に変更すると、開いているすべてのトランザクションに対して MySQL が COMMIT を自動実行するため注意が必要である。

- BIG_TABLES = 0 | 1

  1 に設定すると、テンポラリテーブルのすべてがメモリではなくディスクに格納される。これによって速度が少し低下するが、大きなテンポラリテーブルを必要とする大規模な SELECT 操作でもエラー The table tbl_name is full が出力されなくなる。新たに接続した場合のデフォルト値は 0（メモリ内テンポラリテーブルを使用）である。 この変数は旧称 SQL_BIG_TABLES であった。MySQL 4.0 では、MySQL によって必要に応じてメモリ内テーブルがディスクベーステーブルに自動変換されるため、通常この変数の設定が必要な状況はない。

- CHARACTER SET character_set_name | DEFAULT

  これは、クライアントとの間でやり取りされるすべての文字列に指定のマッピングを行う。 現在、character_set_name の唯一のオプションは cp1251_koi8 のみであるが、MySQL ソースディストリビューションの sql/convert.cc ファイルを編集して容易に新規のマッピングを追加できる。デフォルトのマッピングは、DEFAULT の character_set_name 値を使用してリストアできる。

  CHARACTER SET オプションを設定する構文は、他のオプションを設定する構文とは異なるため注意が必要である。

- DATE_FORMAT = format_str

  サーバで DATE 値を文字列に変換する方法を設定する。この変数は、グローバルオプション、ローカルオプション、コマンドラインオプションのいずれでも使用できる。format_str の指定には GET_FORMAT() 関数の使用が便利である。 See 項6.3.4.「日付と時刻関数」 を参照。

- DATETIME_FORMAT = format_str

  サーバで DATETIME 値を文字列に変換する方法を設定する。この変数は、グローバルオプション、ローカルオプション、コマンドラインオプションのいずれでも使用できる。format_str の指定には GET_FORMAT() 関数の使用が便利である。 See 項6.3.4.「日付と時刻関数」 を参照。

- INSERT_ID = #

  次の INSERT や ALTER TABLE コマンドで使用される AUTO_INCREMENT 値を設定する。これは主としてバイナリログとともに使用される。

- LAST_INSERT_ID = #

  LAST_INSERT_ID() から返される値を設定する。これは、テーブルを更新するコマンドで LAST_INSERT_ID() 関数を使用した場合にバイナリログに格納される。

- LOW_PRIORITY_UPDATES = 0 | 1

  1 に設定した場合、全ての INSERT、UPDATE、DELETE、および LOCK TABLE WRITE ステートメントが、同じテーブルに対して実行されている SELECT や LOCK TABLE READ がなくなるまで待機する。 この変数は旧称 SQL_LOW_PRIORITY_UPDATES であった。

- MAX_JOIN_SIZE = value | DEFAULT

  value を超えるレコードの組み合わせを調べることが必要な SELECT ステートメント、または value を超えるディスクシークの実行が見込まれる SELECT ステートメントを許可しない。この値を設定すると、キーの使用が不適切で長時間かかると見込まれる SELECT ステートメントを捕捉できる。DEFAULT 以外の値に設定すると、SQL_BIG_SELECTS 値が 0 にリセットされる。SQL_BIG_SELECTS 値を設定しなおすと、SQL_MAX_JOIN_SIZE 変数は無視される。mysqld を --max_join_size=value オプションを指定して起動すると、この変数にデフォルト値を設定できる。この変数は旧称 SQL_MAX_JOIN_SIZE であった。

  クエリ結果がすでにクエリキャッシュにある場合は、結果がすでに計算されており、クライアントへの送信による負荷がサーバにかからないため、結果サイズのチェックは実行されない。

- PASSWORD = PASSWORD ('パスワード')

  現在のユーザのパスワードを設定する。すべての非匿名ユーザは各自のパスワードを変更できる。

- PASSWORD FOR user = PASSWORD ('パスワード')

  現在のサーバホストの特定ユーザのパスワードを設定する。この実行は、mysql データベースへのアクセスがあるユーザに限られる。ユーザは user@hostname の形式で指定する必要がある。user と hostname は、mysql.user テーブルエントリの User カラムと Host カラムの記載どおりにする必要がある。たとえば、User フィールドと Host フィールドのエントリが 'bob' および '%.loc.gov' の場合は次のように入力する。

```
mysql> SET PASSWORD FOR 'bob'@'%.loc.gov' = PASSWORD('newpass');
```

  これは以下と等しいことになる。

```
mysql> UPDATE mysql.user SET Password=PASSWORD('newpass')
    ->             WHERE User='bob' AND Host='%.loc.gov';
mysql> FLUSH PRIVILEGES;
```

- QUERY_CACHE_TYPE = OFF | ON | DEMAND , QUERY_CACHE_TYPE = 0 | 1 | 2

  スレッドにクエリキャッシュ設定を行う。

オプション	説明
0 または OFF	結果のキャッシュや取り出しを行わない。
1 または ON	SELECT SQL_NO_CACHE ... クエリを除くすべての結果をキャッシュする。

2 または DEMAND は、	SELECT SQL_CACHE ... クエリのみをキャッシュする。

- SQL_AUTO_IS_NULL = 0 | 1

  1 (デフォルト) に設定すると、WHERE auto_increment_column IS NULL の構造を使用して AUTO_INCREMENT カラムが含まれるテーブルで最後に挿入されたレコードを検索できる。これは、Access などの ODBC プログラムの一部で使用される。

- SQL_BIG_SELECTS = 0 | 1

  0 に設定すると、長時間要すると見込まれる (オプティマイザによって MAX_JOIN_SIZE の値を超えるレコード数が調べられると見込まれる) SELECT ステートメントが MySQL によって中止される。 これは不適切な WHERE ステートメントが使用された場合に役立つ。新たに接続した場合のデフォルト値は 1で、すべての 1 ステートメントが許可される。

  MAX_JOIN_SIZE を DEFAULT 以外の値に設定すると、SQL_BIG_SELECTS が 0 に設定される。

- SQL_BUFFER_RESULT = 0 | 1

  SQL_BUFFER_RESULT は SELECT ステートメントからの結果を強制的にテンポラリテーブルに入れる。これは、MySQL によるテーブルロック解除の早期化と、結果セットのクライアントへの送信に長時間かかる場合に役立つ。

- SQL_LOG_BIN = 0 | 1

  0 に設定すると、クライアントに SUPER 特権がある場合、そのクライアントに関するバイナリログへのログ記録が行われなくなる。

- SQL_LOG_OFF = 0 | 1

  1 に設定すると、クライアントに SUPER 特権がある場合、そのクライアントに関する標準ログへのログ記録が行われなくなる。

- SQL_LOG_UPDATE = 0 | 1

  0 に設定すると、クライアントに SUPER 特権がある場合、そのクライアントに関する更新ログへのログ記録が行われなくなる。 バージョン 5.0 以降はこの変数が廃止されている。

- SQL_QUOTE_SHOW_CREATE = 0 | 1

  1 に設定すると、SHOW CREATE TABLE でテーブル名とカラム名がクオートされる。これはデフォルトでオンになっており、マルチバイト文字などを使用したカラム名をもつテーブルのレプリケーションを可能にする。 項4.6.8.8.「SHOW CREATE TABLE」。

- SQL_SAFE_UPDATES = 0 | 1

  1 に設定すると、MySQL によって WHERE 節でキーや LIMIT を使用しない UPDATE または DELETE ステートメントが中止される。これで、手入力で SQL ステートメントを作成した場合に誤った更新の捕捉が実現される。

- SQL_SELECT_LIMIT = value | DEFAULT

  SELECT ステートメントから返されるレコードの最大数。SELECT に LIMIT 節がある場合、SQL_SELECT_LIMIT の値

よりも LIMIT のほうが優先される。新たに接続した場合のデフォルト値は ``無制限" である。この制限を変更した場合、DEFAULT の SQL_SELECT_LIMIT 値を使用してデフォルト値をリストアできる。

- TIMESTAMP = timestamp_value | DEFAULT

  クライアントに時間を設定する。これは、レコードのリストアにバイナリログを使用する場合、オリジナル(ログが記録された時点)のタイムスタンプにするために使用される。timestamp_value は MySQL タイムスタンプではなく Unix 基準時点のタイムスタンプにする必要がある。

- TIME_FORMAT = format_str

  サーバで TIME 値を文字列に変換する方法を設定する。この変数は、グローバルオプション、ローカルオプション、コマンドラインオプションのいずれでも使用できる。format_str の指定には GET_FORMAT() 関数の使用が便利である。See 項6.3.4.「日付と時刻関数」 を参照。

5.6. ディスク関連の問題

- 前述のように、ディスクシークはパフォーマンスに対する大きなボトルネックである。この問題は、データが拡大し、効率的なキャッシュが実行不能になるほど大きくなるにつれて明白になる。大規模データベースで、事実上ランダムにデータにアクセスする場合、読み取りでは最低 1 回、書き込みでは最低 2 回のディスクシークが必要になることがわかる。この問題を最小にするには、シーク回数を減らすようにディスクを使用する。

- 複数のディスクに対してファイルをシンボリックリンクするか、ストライピングを行って、利用可能なディスクスピンドル数を増加する（およびそれによるシークのオーバヘッドを軽減する）。

  - シンボリックリンクの使用

    MyISAM テーブルの場合、通常のデータディレクトリ内の位置から別のディスクへのインデックスファイルやデータファイルのシンボリックリンクを行う（ストライピングも可能）。これによって、ディスクが他の用途に使用されていなければ、シークと読み取り時間がいずれも改善される。See 項5.6.1.「シンボリックリンクの使用」。

  - ストライピング

    ストライピングは、ディスクが多数ある場合に、第 1 ブロックを第 1 ディスクに、第 2 ブロックは第 2 ディスクに、第 N ブロックは（N mod number_of_disks）ディスクにといった配置を意味する。これにより、通常のデータサイズがストライプサイズより小さい（または完全に一致している）場合にパフォーマンスが大幅に改善する。ストライピングはオペレーティングシステムとストライプサイズへの依存性が非常に高いため、ストライプサイズをさまざまに変えながらアプリケーションのベンチマークを行う。See 項5.1.5.「独自のベンチマークの使用」。

    ストライピングの速度はパラメータによって大きく異なることに注意する。ストライピングパラメータの設定方法とディスク数によって桁ちがいの差異が発生する。ランダムアクセスか順次アクセスのいずれの最適化を行うかの選択が必要なことに注意する。

- 信頼性を高めるため、RAID 0+1（ストライピング + ミラーリング）の使用が必要な場合、N 個のドライブのデータの保持に 2*N 個のドライブが必要になる。財務上の余裕がある場合はこれが最適な選択肢になる。しかし、処理の効率化にボリューム管理ソフトウェアへの投資が必要なこともある。

- データの種類の重大性に応じて RAID レベルを変える選択も推奨される。たとえば、ホスト情報や口グなどの重要度の

高いデータは、RAID 0+1 または RAID N ディスクに格納し、再生成が可能で重要性が中程度のデータは RAID 0 ディスクに格納することなどができる。RAID N は、パリティビットの更新に時間がかかるため、書き込みが多いと問題になる場合がある。

- Linux の場合、hdparm を使用してディスクのインタフェースを構成することでパフォーマンスを大幅に改善できる（負荷時に 100% 改善できることも珍しくない）。次の例は、MySQL（およびその他の多数のアプリケーション）に非常に適した hdparm オプションである。

```
hdparm -m 16 -d 1
```

上記を使用した場合のパフォーマンスと信頼性は使用ハードウェアに依存するため、hdparm の使用後はシステムを総合的にテストするように強く推奨する。詳細については、hdparm のページを参照。hdparm の使用が適切でない場合は、ファイルシステムの損傷が発生することがあるため、テストの際はあらかじめすべてのバックアップを取っておく必要がある。

- データベースが使用するファイルシステムのパラメータを設定することもできる。
  - ファイルへの最終アクセス時を認識する必要がない（データベースサーバでは重要度が低い）場合、-o noatime オプションを使用してファイルシステムをマウントできる。これで、ファイルシステムの i ノードへの最終アクセス時間の更新がスキップされ、一部のディスクシークを回避できる。
  - 多数のオペレーティングシステムで、-o async オプションを使用してディスクをマウントし、ファイルシステムが非同期で更新されるように設定できる。使用しているコンピュータが適度に安定している場合は、信頼性を損なわずにさらにパフォーマンスを改善できる（Linux ではこのフラグがデフォルトでオンになっている）。

5.6.1. シンボリックリンクの使用

テーブルとデータベースをデータベースディレクトリから他の位置に移動し、新しい位置へのシンボリックリンクに置換することができます。 これは、たとえば、データベースを空き領域の多いファイルシステムに移動し、テーブルを別のディスクに分散することでシステムの速度を上げる場合などに実行できます。

この推奨される実行方法は、データベースだけ別のディスクへのシンボリックリンクを行い、最後の手段としてのみテーブルのシンボリックリンクを行うことです。

5.6.1.1. Unix 上のデータベースに対するシンボリックリンクの使用

Unix の場合、データベースのシンボリックリンクは、まず、空き領域のあるディスクにディレクトリを作成し、次に MySQL データベースディレクトリからそのディレクトリへのシンボリックリンクを作成します。

```
shell> mkdir /dr1/databases/test
shell> ln -s /dr1/databases/test mysqld-datadir
```

MySQL は、1 つのディレクトリに対して複数のデータベースをリンクさせることをサポートしていません。データベースディレクトリをシンボリックリンクに置換すると、複数のデータベースへシンボリックリンクを張らない限り、問題なく機能します。 仮に MySQL データディレクトリにデータベース db1 がある場合に、db1 を指すシンボリックリンク db2 を作成するとします。

```
shell> cd /path/to/datadir
shell> ln -s db1 db2
```

これで、db1 のテーブル tbl_a が、db2 のテーブル tbl_a としても表示されます。あるスレッドで db1.tbl_a が更新され、別のスレッドで db2.tbl_a が更新されると、問題が発生します。

このようにすることが実際に必要な場合は、mysys/mf_format.c で次のコードを変更する必要があります。

```
if (flag & 32 || (!lstat(to,&stat_buff) && S_ISLNK(stat_buff.st_mode)))
```

これを次のようにします。

```
if (1)
```

Windows では、-DUSE_SYMDIR を使用して MySQL をコンパイルして、ディレクトリへの内部シンボリックリンクを使用できます。これによって、複数のデータベースを複数のディスクに配置できるようになります。 See 項5.6.1.3.「Windows 上のデータベースに対するシンボリックリンクの使用」。

5.6.1.2. Unix 上のテーブルに対するシンボリックリンクの使用

MySQL 4.0 より前は、テーブルのシンボリックリンクの実行を非常に慎重に行う必要がありました。シンボリックリンクが行われているテーブルで、ALTER TABLE、REPAIR TABLE、あるいは OPTIMIZE TABLE を実行する際に、シンボリックリンクが削除され、オリジナルファイルに置換されるという問題がありました。これらのステートメントは、データベースディレクトリにテンポラリファイルを作成し、ステートメントの操作が完了するとオリジナルファイルとテンポラリファイルの置換が行われる仕様であるため、この問題が発生しました。

realpath() の呼び出しの機能が完全でないシステムではテーブルのシンボリックリンクを行わないでください（少なくとも、Linux と Solaris では realpath() がサポートされています）。

MySQL 4.0 では MyISAM テーブルでのみシンボリックリンクが完全サポートされています。これ以外のテーブル型で上記のコマンドを使用すると、予想外の問題の発生の恐れがあります。

MySQL 4.0 でのシンボリックリンクの処理は、次のように機能します（ほとんどが MyISAM テーブルのみに適しています）。

- データディレクトリには常にテーブル定義ファイル、データファイルおよびインデックスファイルがある。データファイルとインデックスファイルは、別の場所に移動し、データディレクトリ内でシンボリックリンクによって置換できる。定義ファイルはこれができない。
- データファイルとインデックスファイルは、それぞれ独立して別のディレクトリにシンボリックリンクを作成できる。
- シンボリックリンクは、オペレーティングシステムレベル（mysqld が実行されていない場合）、または SQL で CREATE TABLE に DATA DIRECTORY および INDEX DIRECTORY オプションを指定して実行できる。 See 項6.5.3.「CREATE TABLE 構文」。
- myisamchk は、データファイルやインデックスファイルのシンボリックリンクを置き換えない。 myisamchk はリンクで指し示されているファイルに直接作用する。テンポラリファイルはすべてデータファイルやインデックスファイルが配置されているのと同じディレクトリに作成される。
- シンボリックリンクを使用しているテーブルをドロップすると、シンボリックリンクとシンボリックリンクが指しているファイルの両方がドロップされる。このため、root として mysqld を実行すべきではなく、また、MySQL データベースディレクトリへの書き込みアクセスをユーザに許可するべきでもない。

- ALTER TABLE RENAME を使用してテーブルの名前を変更し、テーブルを他のデータベースに移動しない場合、データベースディレクトリのシンボリックリンクの名前が新しい名前に変更され、データファイルとインデックスファイルもそれに従って名前が変更される。
- ALTER TABLE RENAME を使用してテーブルを別のデータベースに移動すると、テーブルが別のデータベースディレクトリに移動され、それまであったシンボリックリンクとそれが指すファイルが削除される（新規テーブルのシンボリックリンクは作成されない）。
- シンボリックリンクを使用していない場合は、mysqld に --skip-symlink オプションを指定して使用し、確実に誰もデータディレクトリの外でファイルのドロップや名前の変更を行う mysqld を使用できないようにする。

サポートされていない事項

- ALTER TABLE では DATA DIRECTORY と INDEX DIRECTORY テーブルオプションが無視される。
- MySQL 4.0.15 より前は、テーブルにシンボリックリンクがある場合、SHOW CREATE TABLE でレポートが行われない。これは、SHOW CREATE TABLE を使用して CREATE TABLE ステートメントを生成する mysqldump についても同様である。
- BACKUP TABLE と RESTORE TABLE ではシンボリックリンクが考慮されない。
- frm ファイルはシンボリックリンクにすることがまったくできない（前述のように、データファイルとインデックスファイルのみシンボリックリンクにできる）。 これを実行した場合（シノニム作成など）、正しい結果が得られなくなる。 MySQL データディレクトリにデータベース db1 があり、このデータベースにはテーブル tbl1 が、db1 ディレクトリには tbl1 を指すシンボリックリンク tbl2 があるとする。

```
shell> cd /path/to/datadir/db1
shell> ln -s tbl1.frm tbl2.frm
shell> ln -s tbl1.MYD tbl2.MYD
shell> ln -s tbl1.MYI tbl2.MYI
```

あるスレッドで db1.tbl1 が読み取られ、別のスレッドで db1.tbl2 が更新されると、問題が発生する。クエリキャッシュが欺かれ（tbl1 が更新されていないと判断され、最新でない結果が返される）、tbl2 に対する ALTER コマンドもエラーになる。

5.6.1.3. Windows 上のデータベースに対するシンボリックリンクの使用

MySQL バージョン 3.23.16 から、MySQL ディストリビューションの mysqld-max および mysql-max-nt サーバが -DUSE_SYMDIR オプションでコンパイルされるようになりました。これにより、シンボリックリンクを設定して別のディスクにデータベースディレクトリを配置できます。 リンクの設定手順は異なりますが、機能は Unix のシンボリックリンクと同様です。

Windows では、対象ディレクトリへのパスが記載されたファイルを作成して MySQL データベースに対するシンボリックリンクを作成します。ファイル名 db_name.sym を使用してデータディレクトリにファイルを保存します。この db_name はデータベース名です。

たとえば、MySQL データディレクトリが C:\mysql\data で、データベース foo を D:\data\foo に配置する場合、パス名 D:\data\foo\ が記載されたファイル C:\mysql\data\foo.sym を作成する必要があります。このようにすると、データベース

foo に作成されているすべてのテーブルが D:\data\foo に作成されます。 この作業には D:\data\foo ディレクトリが存在している必要があります。また、データベース名のディレクトリが MySQL データディレクトリにあるとシンボリックリンクが使用されなくなるため、注意が必要です。言い換えると、すでに foo という名前のデータベースディレクトリがデータディレクトリにある場合、これを D:\data に移動しないとシンボリックリンクが有効にならないことになります（問題を回避するため、データベースディレクトリの移動時はサーバを実行しないでください）。

どのテーブルを開く場合でも速度が低下するため、これをサポートするように MySQL をコンパイルした場合でも、デフォルトでは有効化されていません。シンボリックリンクを有効化するには、my.cnf または my.ini ファイルを次のエントリに入力する必要があります。

```
[mysqld]
symbolic-links
```

MySQL 4.0 では、シンボリックリンクがデフォルトで有効化されています。不要の場合は、skip-symbolic-links オプションで無効化できます。

第6章 MySQL SQL言語リファレンス

MySQL は非常に複雑でありながら、直感的に使用できる覚えやすい SQL インタフェースです。この章では、MySQL を効率的かつ効果的に使用するために知っておく必要がある、さまざまなコマンド、データ型、および関数について説明します。また、この章から、MySQL に組み込まれているすべての機能について参照することもできます。多岐にわたるインデックスの中からそれぞれの内容を参照することによって、この章を効果的に使用することができます。

6.1. 言語構造

6.1.1. リテラル:文字列と数値の記述方法

このセクションでは、MySQL で文字列と数値を記述するさまざまな方法を説明します。 また、MySQL でこれらの基本データ型を処理する時に遭遇するであろう、さまざまなニュアンスと ``了解事項" についても扱います。

6.1.1.1. 文字列

文字列は、単一引用符 ('') または二重引用符 ("") で囲まれた文字の並び (シーケンス) です (ANSI モードでの実行時は引用符のみ)。次に例を示します。

```
'a string'
"another string"
```

一部のシーケンスは、個々の文字列内で特別な意味を持ちます。これらのシーケンスは、いずれも、エスケープ文字として知られるバックスラッシュ ('\') で始まります。MySQL では、次のエスケープシーケンスが認識されます。

- \0

  ASCII 0 (NUL) 文字。

- \'

  単一引用符 ('')。

- \"

  二重引用符 ("")。

- \b

  バックスペース文字。

- \n

  改行文字(LF)。

- \r

復帰改行文字(CR)。

- \t

  タブ文字。

- \z

  ASCII(26)（Control-Z）。この文字をコード化することによって、ASCII(26) が Windows では END-OF-FILE を表すという問題を回避することができる（ASCII(26) では、mysql database < filename を使用する場合に問題が発生する）。

- \\

  バックスラッシュ（'\'）文字。

- \%

  '%' 文字。これは、'%' をそのまま使用したときにワイルドカード文字として解釈されてしまうコンテキストで '%' 自体を検索する場合に使用する。 See 項6.3.2.1.「文字列比較関数」。

- _

  '_' 文字。これは、'_' をそのまま使用したときにワイルドカード文字として解釈されてしまうコンテキストで '_' 自体を検索する場合に使用する。 See 項6.3.2.1.「文字列比較関数」。

文字列の一部のコンテキストでは、'\%' または '_' を使用したときに、'%' と '_' の代わりに、文字列 '\%' と '_' がそれぞれ返されることに注意してください。

文字列に引用符を含める方法は、いくつかあります。

- ''' で囲んだ文字列内で ''' を使用する場合、文字列内の ''' は '''' と記述することができる。
- '"' で囲んだ文字列内で '"' を使用する場合、文字列内の '"' は '""' と記述することができる。
- 引用符の直前にエスケープ文字（'\'）を使用することができる。
- '"' で囲んだ文字列内で ''' を使用する場合は、''' を 2 つ続けて入力したり、エスケープしたりなどの特別な処置を行う必要はない。同様に、''' で囲んだ文字列内で '"' を使用する場合も、特別扱いする必要はない。

次の SELECT ステートメントは、文字列の引用とエスケープが実際にどのように働くかを示しています。

```
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 "This\nIs\nFour\nlines";
+--------------------+
| This
Is
Four
lines |
+--------------------+
```

文字列のカラム (BLOB など) にバイナリデータを挿入する場合、次の文字はエスケープシーケンスを使って表現する必要があります。

- NUL

  ASCII 0。この文字は '\0' (バックスラッシュ + ASCII '0' 文字) で表現する。

- \

  ASCII 92、バックスラッシュ。'\\' として表現する。

- '

  ASCII 39、単一引用符。'\'' として表現する。

- "

  ASCII 34、二重引用符。'\"' として表現する。

C コードを書く場合は、INSERT ステートメントの文字をエスケープする目的で C API 関数 mysql_real_escape_string() を使用できます。See 項11.1.2. 「C API 関数の概要」。Perl では、DBI パッケージの quote メソッドを使用して、特殊文字を適切なエスケープシーケンスに変換することができます。See 項11.5.2. 「DBI インタフェース」。

上記の特殊文字のいずれかが含まれている可能性がある文字列には、必ずエスケープ関数を使用するようにします。

または、MySQL API の多くのものが一種のプレースホルダ機能を備えているため、この機能を使ってクエリ文字列に特殊なマーカーを挿入し、クエリの発行時にデータ値をそれらのマーカーにバインドすることもできます。この場合、値内の特殊文字のエスケープ処理が API によって自動で行われます。

6.1.1.2. 数値

整数は数字の列として表現されます。浮動小数点では、小数を区切るために '.' が使用されます。どちらの型の数値でも、先頭に '-' を付けることによって、負数を表すことができます。

有効な整数の例:

```
1221
0
-32
```

有効な浮動小数点数の例:

```
294.42
-32032.6809e+10
148.00
```

浮動小数点のコンテキストで整数を使用することもできます。この場合、整数は同等の浮動小数点数として解釈されます。

バージョン 4.1.0 以降、定数 TRUE は 1 として評価され、定数 FALSE は 0 として評価されます。

6.1.1.3. 16 進値

MySQL では、16 進値をサポートしています。数値のコンテキストでは、16 進値は整数（64 ビット精度）のように動作します。文字列のコンテキストでは、16 進値はバイナリ文字列のように動作します。この場合、16 進数の各ペアが 1 文字に変換されます。

```
mysql> SELECT x'4D7953514C';
    -> MySQL
mysql> SELECT 0xa+0;
    -> 10
mysql> SELECT 0x5061756c;
    -> Paul
```

MySQL 4.1（および --new オプションを使用した MySQL 4.0）では、16 進値のデフォルトのデータ型は文字列です。16 進値の文字列が確実に数値として扱われるようにするには、その文字列に対して CAST(... AS UNSIGNED) を使用します。

x'hexstring' 構文（4.0 の新機能）は標準 SQL に基づいており、0x 構文は ODBC に基づいています。16 進文字列は、BLOB カラムの値を提供する目的で、ODBC によって使用されることがよくあります。 文字列または数値を 16 進形式の文字列に変換するには、HEX() 関数を使用できます。

6.1.1.4. NULL 値

NULL 値は ``データなし" を意味し、数値型での 0 や文字列型での空文字列などの値とは異なります。 See 項A.5.3.「NULL 値の問題」。

テキストファイルのインポートまたはエクスポート形式（LOAD DATA INFILE、SELECT ... INTO OUTFILE）の使用時、NULL は \N で表現することができます。 See 項6.4.8.「LOAD DATA INFILE 構文」。

6.1.2. データベース名、テーブル名、インデックス名、カラム名、エイリアス名

MySQL では、データベース名、テーブル名、インデックス名、およびエイリアス名には、すべて同じ規則が適用されます。

注意: 識別子（データベース名、テーブル名、カラム名）を ‘`’ で引用する事ができます。MySQL バージョン 3.23.6 以降では、ANSI モードで実行した時は、‘"’ も識別子の引用処理に使用することができます。 See 項1.8.2.「ANSI モードでの MySQL の実行」。

識別子	最大長（バイト）	使用可能な文字
データベース	64	ディレクトリ名に使用可能なすべての文字（ただし、‘/’、‘\’、および ‘.’ を除く）

テーブル	64	ファイル名に使用可能なすべての文字（ただし、'/' と '.' を除く）
カラム	64	すべての文字
エイリアス	255	すべての文字

上記に補足して、ASCII(0)、ASCII(255)、および引用文字はいずれも識別子内では使用できないことに注意してください。

識別子が予約語である場合や、識別子に特殊文字が含まれている場合は、引用符として使用したバッククォート（「`」）文字でその識別子を必ず囲む必要があります。

```
mysql> SELECT * FROM `select` WHERE `select`.id > 100;
```

See 項6.1.7.「MySQL での予約語の扱い」。

MAXDB または ANSI_QUOTES モードで MySQL を実行する場合は、識別子を囲む引用符として二重引用符も使用できます。

```
mysql> CREATE TABLE "test" (col INT);
ERROR 1064: You have an error in your SQL syntax. (...)
mysql> SET SQL_MODE="ANSI_QUOTES";
mysql> CREATE TABLE "test" (col INT);
Query OK, 0 rows affected (0.00 sec)
```

See 項4.1.1.「mysqld コマンドラインオプション」。

バージョン 3.23.6 より前の MySQL バージョン では、名前に関して次の規則が適用されます。

- 個々の名前は、現在のキャラクタセットに従った英数字、および '_' と '$' で構成することができる。デフォルトのキャラクタセットは ISO-8859-1 Latin1。このキャラクタセットは、 mysqld に --default-character-set オプションを指定することにより変更可能。 See 項4.7.1.「データおよびソート用キャラクタセット」。
- 個々の名前では、名前に使用可能な任意の文字を、先頭文字とすることができる。先頭を数字にすることも可能（この点は、他の多くのデータベースシステムと異なる）。ただし、数字だけで構成される名前は使用できない。
- '.' 文字は名前には使用できない。この文字は、カラムを参照するための拡張形式で使用される（これについては後述する）。

1e のような名前は、使用しないでください。これは、1e+1 のような式があいまいになるためです。1e+1 は、式 1e + 1 として、または数値 1e+1 として解釈されます。

MySQL では、次の形式のいずれかを使用してカラムを参照することができます。

カラム参照	意味
col_name	この名前のカラムが組み込まれたクエリで使用されているテーブル内のカラム col_name。
tbl_name.col_name	カレントデータベースのテーブル col_name 内のカラム tbl_name。
db_name.tbl_name.col_name	データベース db_name のテーブル tbl_name 内のカラム col_name。この形式は

	MySQL Version 3.22 以降で使用可能。
`column_name`	それ自体がキーワードであるか、その中に特殊文字を含んでいるカラム。

対象となる参照があいまいな場合、カラム参照の前に tbl_name や db_name.tbl_name を付ける必要があります。 たとえば、テーブル t1 と t2 のそれぞれに同名のカラム c があり、t1 と t2 の両方を使用する SELECT ステートメントで c を読み取るとします。この場合 c は、ステートメントで使用されている 2 つのテーブル中で一意なカラムを表すものではなく、あいまいであるため、t1.c または t2.c と記述することによって、どちらのテーブルが対象か指定する必要があります。同様に、データベース db1 のテーブル t とデータベース db2 のテーブル t に含まれているカラムを取り出す場合は、それぞれのテーブルのカラムを db1.t.col_name、db2.t.col_name として参照します。

構文 .tbl_name はカレントデータベースのテーブル tbl_name を意味します。この構文は ODBC との互換性を確保する目的で許容されています。これは、一部の ODBC プログラムでテーブル名の先頭に '.' 文字が付けられるためです。

6.1.3. 名前におけるケース依存

MySQL において、データベースとテーブルは、ディレクトリとそれらのディレクトリ内のファイルに対応しています。そのため、ベースとなっているオペレーティングシステムで大文字と小文字が区別される場合 (ケース依存) 、データベース名とテーブル名でも大文字と小文字が区別されます。つまり、Windows ではデータベース名とテーブル名で大文字と小文字は区別されず、ほとんどの種類の Unix では大文字と小文字が区別されることになります。ただし、重要な例外が 1 つあります。Mac OS X でデフォルトの HFS+ ファイルシステムを使用している場合です。 しかし、Mac OS X は UFS ボリュームもサポートしています。UFS ボリュームでは Unix の場合と同じように Mac OS X でも大文字と小文字が区別されます。 See 項1.8.3.「SQL-92 標準に対する MySQL 拡張機能」。

注意: Windows ではデータベース名とテーブル名で大文字と小文字は区別されませんが (ケース非依存) 、同じクエリ内で異なるケースを使用して同じデータベースやテーブルを参照しないようにしてください。次のクエリでは、同じテーブルが my_table および MY_TABLE として参照されています。したがって、このクエリは機能しません。

```
mysql> SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

カラム名とカラムのエイリアスは、あらゆる状況においてケース非依存です。

テーブルのエイリアスはケース依存です。次のクエリでは、同じエイリアスが a および A として参照されています。したがって、このクエリは機能しません。

```
mysql> SELECT col_name FROM tbl_name AS a
    ->        WHERE a.col_name = 1 OR A.col_name = 2;
```

データベース名やテーブル名に大文字と小文字のどちらを使用したか覚えにくい場合は、データベースとテーブルは必ず小文字の名前で作成するなどの一貫した規則を設けるようにします。

この問題に対処する 1 つの方法は、mysqld の先頭に -O lower_case_table_names=1 を付けることです。このオプションのデフォルトは Windows では 1 で、Unix では 0 です。

lower_case_table_names が 1 の場合、MySQL では、保管およびルックアップ時にすべてのテーブル名が小文字に変換されます (バージョン 4.0.2 以降、このオプションはデータベース名にも適用されます。4.1.1 以降、このオプションはテーブルエイリアスにも適用されます) 。

このオプションを変更する場合は、最初に元のテーブル名を小文字に変換してから mysqld を起動する必要があります。

MyISAM ファイルを Windows から Unix のディスクに移動した場合、mysql_fix_extensions ツールを使用して、指定した各データベースディレクトリ内のファイル拡張子のケース (小文字の .frm、大文字の .MYI および .MYD) を修正する必要がある場合があります mysql_fix_extensions は scripts サブディレクトリにあります。

6.1.4. ユーザ変数

MySQL では、@variablename 構文での接続ごとのユーザ変数をサポートしています。変数名は、現在のキャラクタセット内の英数字、および '_'、'$'、'.' で構成することができます。デフォルトのキャラクタセットは ISO-8859-1 Latin1 です。このキャラクタセットは、--default-character-set オプションを指定した mysqld で変更可能です。See 項4.7.1.「データおよびソート用キャラクタセット」。ユーザ変数名は、バージョン 5.0 以降のバージョンではケース非依存で、バージョン 5.0 より前のバージョンではケース依存です。

変数は初期化する必要はありません。変数の値はデフォルトでは NULL であり、整数、実数、または文字列値を格納することができます。スレッドのすべての変数は、そのスレッドが終了すると自動的に解放されます。

変数は SET 構文を使用して設定することができます。

```
SET @variable= { integer expression | real expression | string expression }
[,@variable= ...].
```

SET 以外のステートメントで変数に値を代入することも可能です。 ただし、この場合、代入演算子は = ではなく := です。= は、SET 以外のステートメントにおいて、比較用に予約されています。

```
mysql> SET @t1=0, @t2=0, @t3=0;
mysql> SELECT @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
+----------------------+------+------+------+
| @t1:=(@t2:=1)+@t3:=4 | @t1  | @t2  | @t3  |
+----------------------+------+------+------+
|                    5 |    5 |    1 |    4 |
+----------------------+------+------+------+
```

ユーザ変数は、式を使用できる箇所に使用することができます。ただし、SELECT 文の LIMIT 節や LOAD DATA 文の IGNORE number LINES 節など、数値が明示的に要求されている文脈での使用は含まれません。

注意: SELECT 文においては、それぞれの式は、クライアントに送られた時にはじめて評価されます。したがって、HAVING、GROUP BY、ORDER BY 節において、SELECT 部に設定された変数を含む式を参照することはできません。たとえば、次の文は、期待どおりに機能しません。

```
mysql> SELECT (@aa:=id) AS a, (@aa+3) AS b FROM table_name HAVING b=5;
```

その理由は、@aa の値が現在のレコードの値ではなく、前に受け取ったレコードの id 値であるためです。

原則として、変数への値の代入と使用の両方の処理を、同じステートメントでは行わないでください。

変数の設定とその使用を同じステートメントで行った場合、変数のデフォルトの結果型がそのステートメントの開始時におけるその変数のデータ型に基づいて決まってしまう、という問題もあります (値が代入されていない変数は NULL 値を取り、STRING 型であると想定されます) 。この例を次に示します。

```
mysql> SET @a="test";
mysql> SELECT @a,(@a:=20) FROM table_name;
```

この場合、MySQL では、カラム 1 が文字列としてクライアントに報告されます。そして、2 番目のレコードで @a が数値に設定されるにもかかわらず、@a へのすべてのアクセスが文字列に変換されます。ステートメントの実行後、@a は数値とみなされるようになります。

この場合、何か問題がある場合は、変数の設定とその使用を同じステートメントで行わないようにするか、もしくはその変数を使用する前に値を 0、0.0、または "" に設定するようにします。

6.1.5. システム変数

MySQL 4.0.3 以降では、多くのシステム変数や接続変数にアクセスしやすくなっています。それらの変数のほとんどは、サーバを停止することなく変更できます。

システム変数には、現在の接続だけに該当するスレッド固有（接続固有）変数と、グローバルイベントの設定に使用されるグローバル変数の 2 つの種類があります。 グローバル変数は、新しい接続に対応するスレッド固有変数の初期値を設定するためにも使用されます。

mysqld の起動時には、コマンドライン引数とオプションファイルからすべてのグローバル変数が初期化されます。この値は SET GLOBAL コマンドを使用して変更することができます。新しいスレッドが作成されると、グローバル変数からスレッド固有変数が初期化されます。この値は新しい SET GLOBAL コマンドを発行しても変更されません。

GLOBAL 変数の値を設定するには、次の構文のいずれかを使用します（ここでは、変数の例として sort_buffer_size を使用します）。

```
SET GLOBAL sort_buffer_size=value;
SET @@global.sort_buffer_size=value;
```

SESSION 変数の値を設定するには、次の構文のいずれかを使用することができます。

```
SET SESSION sort_buffer_size=value;
SET @@session.sort_buffer_size=value;
SET sort_buffer_size=value;
```

GLOBAL と SESSION のどちらも指定しないと、SESSION が使用されます。 See 項5.5.6.「SET 構文」。

LOCAL は SESSION のシノニムです。

GLOBAL 変数の値を取り出すには、次のコマンドのいずれかを使用することができます。

```
SELECT @@global.sort_buffer_size;
SHOW GLOBAL VARIABLES like 'sort_buffer_size';
```

SESSION 変数の値を取り出すには、次のコマンドのいずれかを使用することができます。

```
SELECT @@session.sort_buffer_size;
SHOW SESSION VARIABLES like 'sort_buffer_size';
```

@@variable_name 構文で変数を取り出すときに GLOBAL と SESSION のどちらも指定しないと、MySQL では、スレッド固有（SESSION）の値がある場合は、その値が返されます。スレッド固有の値がない場合は、グローバル値が返されます。

GLOBAL の値しか存在しない変数の場合、取り出しには GLOBAL と指定する必要はありませんが、設定には GLOBAL と

指定する必要があります。これは、後から、同名のスレッド固有変数を導入したり、削除したりするときに問題が発生しないようにするためです。この場合、自分の接続だけでなく、サーバの状態そのものを誤って変更してしまう可能性があります。

次の一覧に、変更および取り出し対象となるすべての変数と、それらの変数で GLOBAL と SESSION のどちらを使用できるかを示します。

変数名	値のデータ型	タイプ
autocommit	bool	SESSION
big_tables	bool	SESSION
binlog_cache_size	num	GLOBAL
bulk_insert_buffer_size	num	GLOBAL | SESSION
concurrent_insert	bool	GLOBAL
connect_timeout	num	GLOBAL
convert_character_set	string	SESSION
delay_key_write	OFF | ON | ALL	GLOBAL
delayed_insert_limit	num	GLOBAL
delayed_insert_timeout	num	GLOBAL
delayed_queue_size	num	GLOBAL
error_count	num	SESSION
flush	bool	GLOBAL
flush_time	num	GLOBAL
foreign_key_checks	bool	SESSION
identity	num	SESSION
insert_id	bool	SESSION
interactive_timeout	num	GLOBAL | SESSION
join_buffer_size	num	GLOBAL | SESSION
key_buffer_size	num	GLOBAL
last_insert_id	bool	SESSION
local_infile	bool	GLOBAL
log_warnings	bool	GLOBAL
long_query_time	num	GLOBAL | SESSION
low_priority_updates	bool	GLOBAL | SESSION
max_allowed_packet	num	GLOBAL | SESSION
max_binlog_cache_size	num	GLOBAL
max_binlog_size	num	GLOBAL
max_connect_errors	num	GLOBAL
max_connections	num	GLOBAL

max_error_count	num	GLOBAL | SESSION
max_delayed_threads	num	GLOBAL
max_heap_table_size	num	GLOBAL | SESSION
max_join_size	num	GLOBAL | SESSION
max_relay_log_size	num	GLOBAL
max_sort_length	num	GLOBAL | SESSION
max_tmp_tables	num	GLOBAL
max_user_connections	num	GLOBAL
max_write_lock_count	num	GLOBAL
myisam_max_extra_sort_file_size	num	GLOBAL | SESSION
myisam_repair_threads	num	GLOBAL | SESSION
myisam_max_sort_file_size	num	GLOBAL | SESSION
myisam_sort_buffer_size	num	GLOBAL | SESSION
net_buffer_length	num	GLOBAL | SESSION
net_read_timeout	num	GLOBAL | SESSION
net_retry_count	num	GLOBAL | SESSION
net_write_timeout	num	GLOBAL | SESSION
query_cache_limit	num	GLOBAL
query_cache_size	num	GLOBAL
query_cache_type	enum	GLOBAL
read_buffer_size	num	GLOBAL | SESSION
read_rnd_buffer_size	num	GLOBAL | SESSION
rpl_recovery_rank	num	GLOBAL
safe_show_database	bool	GLOBAL
server_id	num	GLOBAL
slave_compressed_protocol	bool	GLOBAL
slave_net_timeout	num	GLOBAL
slow_launch_time	num	GLOBAL
sort_buffer_size	num	GLOBAL | SESSION
sql_auto_is_null	bool	SESSION
sql_big_selects	bool	SESSION
sql_big_tables	bool	SESSION
sql_buffer_result	bool	SESSION
sql_log_binlog	bool	SESSION
sql_log_off	bool	SESSION

sql_log_update	bool	SESSION
sql_low_priority_updates	bool	GLOBAL | SESSION
sql_max_join_size	num	GLOBAL | SESSION
sql_quote_show_create	bool	SESSION
sql_safe_updates	bool	SESSION
sql_select_limit	bool	SESSION
sql_slave_skip_counter	num	GLOBAL
sql_warnings	bool	SESSION
table_cache	num	GLOBAL
table_type	enum	GLOBAL | SESSION
thread_cache_size	num	GLOBAL
timestamp	bool	SESSION
tmp_table_size	enum	GLOBAL | SESSION
tx_isolation	enum	GLOBAL | SESSION
wait_timeout	num	GLOBAL | SESSION
warning_count	num	SESSION
unique_checks	bool	SESSION

値のデータ型が num となっている変数には、数値を設定することができます。 bool となっている変数には、0、1、ON、または OFF を設定することができます。 enum 型の変数には、通常、その変数に対して利用可能な値の 1 つを設定できますが、該当の enum (列挙) 値に対応する数値を設定することもできます (最初の列挙値は 0 です)。

これらの変数のいくつかについて説明します。

変数	説明
identity	last_insert_id のエイリアス (Sybase との互換性を確保するため)
sql_low_priority_updates	low_priority_updates のエイリアス
sql_max_join_size	max_join_size のエイリアス
version	VERSION() のエイリアス (Sybase (?) との互換性を確保するため)

その他の変数については、スタートアップオプション、SHOW VARIABLES、および SET に関するセクションで説明しています。 See 項4.1.1.「mysqld コマンドラインオプション」。 See 項4.6.8.4.「SHOW VARIABLES」。 See 項5.5.6.「SET 構文」。

6.1.6. コメント構文

MySQL サーバでは、コメントスタイルとして、# (行末まで)、-- (行末まで)、および /* (行中または複数行) */をサポートしています。

```
mysql> SELECT 1+1;     # このコメントは行末まで続く
```

```
mysql> SELECT 1+1;     -- このコメントは行末まで続く
mysql> SELECT 1 /* これは行中コメント */ + 1;
mysql> SELECT 1+
/*
これは
複数行コメント
*/
1;
```

-- (ダッシュ2つ) のコメントスタイルでは、2 つ目のダッシュの後にスペースを 1 つ以上挿入する必要があることに注意してください。

サーバは上記のコメント構文を理解しますが、mysql クライアントでの /* ... */ コメントの解析には一定の制約があります。

- 単一引用符と二重引用符は、コメント内であっても、引用文字列の開始を示すものとして解釈される。そのコメント内に、最初の引用符と一致するもう 1 つの引用符がない場合、パーサはそのコメントの終了を認識しない。mysql を対話的に実行している場合は、プロンプトが mysql> から '> または "> に変わるため、パーサがコメントの終了を認識できずにいることがわかる。
- セミコロンは、現在の SQL ステートメントの終了を表すものとして解釈され、セミコロンの後は、次のステートメントの開始を表すものとして解釈される。

これらの制約は、mysql を対話的に実行する場合と、mysql < some-fileを使用して、コマンドを格納したファイルから入力を読み取るよう mysql に指示する場合の両方に適用されます。

2 つ目のダッシュの後にスペースが 1 つもない場合、SQL-99 のコメントスタイル '--' は MySQL で有効とされません。See 項1.8.4.7.「コメントの開始記号としての '--'」。

6.1.7. MySQL での予約語の扱い

一般的な問題の 1 つとして、TIMESTAMP や GROUP など、MySQL に組み込まれているデータ型や関数の名前を使用するカラム名を含むテーブルの作成に関連する問題があります。このようなカラム名を含むテーブルの作成は可能です (たとえば、ABS はカラム名として使用可能です) 。しかし、デフォルトでは、関数の呼び出し時に、関数名とそれに続く '(' 文字との間に空白を挿入することはできません。これは、関数呼び出しをカラム名の参照と区別するためです。

--ansi または --sql-mode=IGNORE_SPACE オプションを指定してサーバを起動した場合は、関数呼び出しで、関数名とそれに続く '(' 文字との間に空白を挿入することができます。これらのオプションを指定すると、関数名が予約語として扱われるようになります。そのため、関数名と同一のカラム名は、項6.1.2.「データベース名、テーブル名、インデックス名、カラム名、エイリアス名」 で説明しているように引用符で囲む必要があります。

以下の語は MySQL において明示的に予約されています。これらの語のほとんど (たとえば、GROUP) は、SQL-92 では、カラム名またはテーブル名としての使用を禁止されています。 いくつかの語は、MySQL でそれらの語を必要とし、 (現在) MySQL で yacc パーサが使用されていることから予約されています。

ADD	ALL	ALTER
ANALYZE	AND	AS
ASC	BEFORE	BETWEEN
BIGINT	BINARY	BLOB

BOTH	BY	CASCADE
CASE	CHANGE	CHAR
CHARACTER	CHECK	COLLATE
COLUMN	COLUMNS	CONSTRAINT
CONVERT	CREATE	CROSS
CURRENT_DATE	CURRENT_TIME	CURRENT_TIMESTAMP
CURRENT_USER	DATABASE	DATABASES
DAY_HOUR	DAY_MICROSECOND	DAY_MINUTE
DAY_SECOND	DEC	DECIMAL
DEFAULT	DELAYED	DELETE
DESC	DESCRIBE	DISTINCT
DISTINCTROW	DIV	DOUBLE
DROP	DUAL	ELSE
ENCLOSED	ESCAPED	EXISTS
EXPLAIN	FALSE	FIELDS
FLOAT	FLOAT4	FLOAT8
FOR	FORCE	FOREIGN
FROM	FULLTEXT	GRANT
GROUP	HAVING	HIGH_PRIORITY
HOUR_MICROSECOND	HOUR_MINUTE	HOUR_SECOND
IF	IGNORE	IN
INDEX	INFILE	INNER
INSERT	INT	INT1
INT2	INT3	INT4
INT8	INTEGER	INTERVAL
INTO	IS	JOIN
KEY	KEYS	KILL
LEADING	LEFT	LIKE
LIMIT	LINES	LOAD
LOCALTIME	LOCALTIMESTAMP	LOCK
LONG	LONGBLOB	LONGTEXT
LOW_PRIORITY	MATCH	MEDIUMBLOB
MEDIUMINT	MEDIUMTEXT	MIDDLEINT
MINUTE_MICROSECOND	MINUTE_SECOND	MOD
NATURAL	NOT	NO_WRITE_TO_BINLOG

NULL	NUMERIC	ON
OPTIMIZE	OPTION	OPTIONALLY
OR	ORDER	OUTER
OUTFILE	PRECISION	PRIMARY
PRIVILEGES	PROCEDURE	PURGE
READ	REAL	REFERENCES
REGEXP	RENAME	REPLACE
REQUIRE	RESTRICT	REVOKE
RIGHT	RLIKE	SECOND_MICROSECOND
SELECT	SEPARATOR	SET
SHOW	SMALLINT	SONAME
SPATIAL	SQL_BIG_RESULT	SQL_CALC_FOUND_ROWS
SQL_SMALL_RESULT	SSL	STARTING
STRAIGHT_JOIN	TABLE	TABLES
TERMINATED	THEN	TINYBLOB
TINYINT	TINYTEXT	TO
TRAILING	TRUE	UNION
UNIQUE	UNLOCK	UNSIGNED
UPDATE	USAGE	USE
USING	UTC_DATE	UTC_TIME
UTC_TIMESTAMP	VALUES	VARBINARY
VARCHAR	VARCHARACTER	VARYING
WHEN	WHERE	WITH
WRITE	XOR	YEAR_MONTH
ZEROFILL		

以下はMySQL 4.0で登場する新規の予約語です。

CHECK	FORCE	LOCALTIME
LOCALTIMESTAMP	REQUIRE	SQL_CALC_FOUND_ROWS
SSL	XOR	

以下はMySQL 4.1で登場する新規の予約語です。

BEFORE	COLLATE	CONVERT
CURRENT_USER	DAY_MICROSECOND	DIV
DUAL	FALSE	HOUR_MICROSECOND

MINUTE_MICROSECOND	MOD	NO_WRITE_TO_BINLOG
SECOND_MICROSECOND	SEPARATOR	SPATIAL
TRUE	UTC_DATE	UTC_TIME
UTC_TIMESTAMP	VARCHARACTER	

以下のシンボル (上の表に含まれるもの)は SQL-99 では使用を禁止されていますが、MySQL ではカラム名またはテーブル名として使用可能です。これは、それらの名前がごく一般的なものであり、多くの人々にすでに使用されているためです。

- ACTION
- BIT
- DATE
- ENUM
- NO
- TEXT
- TIME
- TIMESTAMP

6.2. カラム型

MySQL では、複数のカラム型をサポートしています。これらのカラム型は、数値型、日付と時刻型、文字列 (文字) 型の 3 つのカテゴリに分類することができます。このセクションでは、まず、使用できるカラム型の概要を示し、各カラム型で必要となる記憶容量について簡単に説明します。その後、各カテゴリのカラム型の特性を詳しく説明します。 概要はあえて簡単にまとめてあります。値の有効な指定形式など、個々のカラム型の追加情報については、それぞれの詳細な説明を参照してください。

以下に、MySQL でサポートしているカラム型を示します。 説明内では、次のコード文字を使用します。

- M

  最大表示サイズを表す。正式な最大表示サイズは 255。

- D

  小数点型に適用され、小数点以下の桁数を表す。最大値は 30 だが、M-2 より大きくしないようにする。

角かっこ ('[' と ']') は、オプションの型指定子の一部であることを表します。

カラムに対して ZEROFILL を指定すると、そのカラムに UNSIGNED 属性が自動で追加されることに注意してください。

警告: 整数値の減算で、どちらか一方の整数値が UNSIGNED 型の場合、結果の値は符号なしになります。 See 項6.3.5.「キャスト関数」。

- TINYINT[(M)] [UNSIGNED] [ZEROFILL]

  非常に小さな整数。符号付きの範囲は -128 ～ 127。符号なしの範囲は 0 ～ 255。

- BIT , BOOL , BOOLEAN

  いずれも TINYINT(1) のシノニム。 シノニム BOOLEAN はバージョン 4.1.0 で追加された。

  ブール型の完全な処理は SQL-99 に基づいて導入される。

- SMALLINT[(M)] [UNSIGNED] [ZEROFILL]

  小さな整数。符号付きの範囲は -32768 ～ 32767。符号なしの範囲は 0 ～ 65535。

- MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]

  中間サイズの整数。符号付きの範囲は -8388608 ～ 8388607。符号なしの範囲は 0 ～ 16777215。

- INT[(M)] [UNSIGNED] [ZEROFILL]

  通常サイズの整数。符号付きの範囲は -2147483648 ～ 2147483647。符号なしの範囲は 0 ～ 4294967295。

- INTEGER[(M)] [UNSIGNED] [ZEROFILL]

  INT のシノニム。

- BIGINT[(M)] [UNSIGNED] [ZEROFILL]

  大きい整数。符号付きの範囲は -9223372036854775808 ～ 9223372036854775807。符号なしの範囲は 0 ～ 18446744073709551615。

  BIGINT 型のカラムに関しては、次の点について注意すること。

  - すべての算術演算は符号付き BIGINT または DOUBLE 型の値を使って行われる。そのため、9223372036854775807 (63 ビット) を超える、符号なしの大きい整数は、ビット関数以外では使用しないようにする。ビット関数以外でこのような大きい整数を使用すると、BIGINT から DOUBLE への変換時に発生する丸め誤差の影響で、結果の最後の桁の一部に誤りが出る場合がある。

    次の場合、SQL 4.0 では BIGINT を処理できる。

    - 整数を使って、符号なしの大きい値を BIGINT カラムに格納する場合
    - MIN(big_int_column) および MAX(big_int_column) において
    - 演算子 (+、-、* など) の使用時に両方のオペランドが整数の場合

  - 文字列として格納すれば、正確な整数値を BIGINT カラムに常に格納することができる。この場合、倍精度表現を介在しない、文字列から数値への変換が実行される。
  - '-'、'+'、および '*' で両方の引数が整数値のときは、BIGINT 演算が使用される。したがって、2 つの大きな整数 (または整数を返す関数の結果) を掛け算する場合、結果が 9223372036854775807 より大きいと、予期しない結果が返される場合がある。

- FLOAT(precision) [UNSIGNED] [ZEROFILL]

  浮動小数点数。precision は、単精度浮動小数点数の場合は <=24 で、倍精度浮動小数点数の場合は 25 〜 53 の間。これらの型はこのすぐ後に説明する FLOAT 型と DOUBLE 型に類似している。 FLOAT(X) は対応する FLOAT 型および DOUBLE 型と範囲は同じだが、表示サイズと小数部桁数は定義されない。

  MySQL バージョン 3.23 では、これは真の浮動小数点値。それ以前の MySQL バージョンでは、FLOAT(precision) の小数部は常に 2 桁になる。

  MySQL の計算はすべて倍精度で行われるため、FLOAT の使用時には予期しない問題が発生する場合があることに注意する。 See 項A.5.6.「不整合レコードの問題解決」。

  この構文は ODBC との互換性を確保するために提供されている。

- FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]

  単精度浮動小数点数。使用可能な値は -3.402823466E+38 〜 -1.175494351E-38、0、および 1.175494351E-38 〜 3.402823466E+38。UNSIGNED を指定した場合、負数は使用できない。M は表示幅で、D は小数部桁数。引数のない FLOAT や、X が 24 以下の FLOAT(X) は、単精度浮動小数点数を表す。

- DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]

  倍精度浮動小数点数。 使用可能な値は -1.7976931348623157E+308 〜 -2.2250738585072014E-308、0、および 2.2250738585072014E-308 〜 1.7976931348623157E+308。UNSIGNED を指定した場合、負数は使用できない。M は表示幅で、D は小数部桁数。引数のない DOUBLE や、X が 25 以上 53 以下である FLOAT(X) は、倍精度浮動小数点数を表す。

- DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL] , REAL[(M,D)] [UNSIGNED] [ZEROFILL]

  いずれも DOUBLE のシノニム。

- DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]

  アンパック浮動小数点数。CHAR カラムのように動作する。``アンパック" とは、その数値が、各桁に 1 文字ずつ使用して文字列として格納されることを意味する。M では、小数点と、負数に使用される '-' 記号はカウントされない (しかし、これらのためのスペースは確保される) 。D が 0 の場合、値は小数点も小数部も持たない。DECIMAL 値の最大範囲は、DOUBLE と同じだが、個々の DECIMAL カラムの実際の範囲は、M と D の値によって制限される。UNSIGNED を指定した場合、負数は使用できない。

  D を省略した場合、デフォルトは 0。M を省略した場合、デフォルトは 10。

MySQL バージョン 3.23 より前のバージョンでは、M 引数に、符号と小数点に必要なスペースを含める必要がある。

- DEC[(M[,D])] [UNSIGNED] [ZEROFILL] , NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL] , FIXED[(M[,D])] [UNSIGNED] [ZEROFILL]

  いずれも DECIMAL のシノニム。

  FIXED エイリアスは、他のサーバとの互換性を確保する目的で、バージョン 4.1.0 で追加された。

- DATE

  日付。サポートしている範囲は、'1000-01-01' ～ '9999-12-31'。MySQL では、DATE 値は 'YYYY-MM-DD' 形式で表示されるが、DATE カラムへの値の割り当てには文字列または数値のいずれかを使用することができる。 See 項6.2.2.2.「DATETIME、DATE、TIMESTAMP 型」。

- DATETIME

  日付と時刻の組み合わせ。サポートしている範囲は、'1000-01-01 00:00:00' ～ '9999-12-31 23:59:59'。MySQL では、DATETIME 値は 'YYYY-MM-DD HH:MM:SS' 形式で表示されるが、DATETIME カラムへの値の割り当てには文字列または数値のいずれかを使用することができる。

  See 項6.2.2.2.「DATETIME、DATE、TIMESTAMP 型」。

- TIMESTAMP[(M)]

  タイムスタンプ。範囲は '1970-01-01 00:00:00' ～ 2037 年の一定の時点。

  MySQL 4.0 以前のバージョンでは、TIMESTAMP 値は、M が 14 (または指定なし) 、12、8、または 6 のどれであるかに応じて、YYYYMMDDHHMMSS、YYMMDDHHMMSS、YYYYMMDD、または YYMMDD 形式で表示されるが、TIMESTAMP カラムへの値の割り当てには文字列または数値のいずれかを使用することができる。

  MySQL 4.1 以降では、TIMESTAMP は 'YYYY-MM-DD HH:MM:SS' 形式の文字列として返される。数値として返されるようにするには、タイムスタンプカラムに +0 を追加する。異なるタイムスタンプ長はサポートしていない。バージョン 4.0.12 以降、--new オプションを使用することで、バージョン 4.1 と同じようにサーバを動作させることができる。

  TIMESTAMP カラムに値を指定しないと、最後に行われた操作の日付と時刻が自動的に設定されるため、このカラムは INSERT 操作や UPDATE 操作の日付と時刻を記録するのに役立つ。また、このカラムに NULL 値を割り当てることによって、現在の日付と時刻をカラムに設定することができる。 See 項6.2.2.「日付と時刻型」。

  M 引数は、TIMESTAMP カラムの表示方法にのみ作用する。 この値は常に 4 バイトで格納される。

  M が 8 または 14 の TIMESTAMP(M) カラムは数値として報告され、その他の TIMESTAMP(M) カラムは文字列として報告されることに注意する。これは単に、これらのデータ型のテーブルのダンプとリストアを確実に実行できるようにすることを目的としている。 See 項6.2.2.2.「DATETIME、DATE、TIMESTAMP 型」。

- TIME

  時刻。範囲は '-838:59:59' ～ '838:59:59'。 MySQL では、TIME 値は 'HH:MM:SS' 形式で表示されるが、TIME カラムへ

の値の割り当てには文字列または数値のいずれかを使用することができる。 See 項6.2.2.3.「TIME 型」。

- YEAR[(2|4)]

  2 桁または 4 桁の形式の年 (デフォルトは 4 桁) 。使用可能な値は、4 桁形式では 1901 〜 2155 と 0000、2 桁形式では 1970 〜 2069 (70 〜 69) 。MySQL では、YEAR 値は YYYY 形式で表示されるが、YEAR カラムへの値の割り当てには文字列または数値のいずれかを使用することができる (YEAR 型は MySQL バージョン 3.22 より前のバージョンでは使用できない) 。 See 項6.2.2.4.「YEAR 型」。

- [NATIONAL] CHAR(M) [BINARY | ASCII | UNICODE]

  固定長の文字列。格納時には、指定の長さになるよう、右側にスペースが埋め込まれる。M の範囲は 0 〜 255 文字 (3.23 より前の MySQL バージョンでは 1 〜 255) 。 値の取り出し時には、後続のスペースが削除される。BINARY キーワードを指定しない場合、CHAR 型の値のソートと比較は、デフォルトのキャラクタセットに基づいてケース非依存方式で行われる。

  バージョン 4.1.0 以降では、255 より大きい M 値を指定すると、カラム型が TEXT 型に変換される。

  これは互換性を考慮した機能。

  NATIONAL CHAR (または、これに対応する短縮形式 NCHAR) は、SQL-99 における、CHAR カラムでデフォルトの CHARACTER セットを使用することを定義する方法。MySQL では、これはデフォルト。

  CHAR は CHARACTER の省略形。

  バージョン 4.1.0 以降では、latin1 キャラクタセットを CHAR カラムに割り当てる ASCII 属性を指定することができる。

  バージョン 4.1.1 以降では、ucs2 キャラクタセットを CHAR カラムに割り当てる UNICODE 属性を指定することができる。

  MySQL では、CHAR(0) 型のカラムを作成することができる。これは、主に、カラム自体は必要とするが、そのカラムの値を実際に使用することはない、というような古いアプリケーションに対応する必要があるときに役立つ。また、2 つの値しか取らないカラムが必要な場合にも非常に役立つ。 CHAR(0) は NOT NULL としては定義されず、1 ビットのみ占め、NULL または "" の 2 つの値しか取らない。 See 項6.2.3.1.「CHAR 型と VARCHAR 型」。

- CHAR

  CHAR(1) のシノニム。

- [NATIONAL] VARCHAR(M) [BINARY]

  可変長文字列。注意: 後続のスペースは値の格納時に削除される (これは SQL-99 の仕様とは異なる) 。M の範囲は 0 〜 255 文字 (MySQL バージョン 4.0.2 では 1 〜 255) 。 BINARY キーワードを指定しないと、VARCHAR 値のソートと比較は、ケース非依存方式で行われる。 See 項6.5.3.1.「カラムの暗黙的な変更」。

  バージョン 4.1.0 以降では、255 より大きい M 値を指定すると、カラム型が TEXT 型に変換される。 これは互換性を考慮した機能。

VARCHAR は CHARACTER VARYING の省略形。

See 項6.2.3.1.「CHAR 型と VARCHAR 型」。

- TINYBLOB , TINYTEXT

  最大長が 255 (2^8 - 1) 文字の BLOB 型または TEXT 型のカラム。 See 項6.5.3.1.「カラムの暗黙的な変更」。 See 項6.2.3.2.「BLOB 型と TEXT 型」。

- BLOB , TEXT

  最大長が 65535 (2^16 - 1) 文字の BLOB 型または TEXT 型のカラム。 See 項6.5.3.1.「カラムの暗黙的な変更」。 See 項6.2.3.2.「BLOB 型と TEXT 型」。

- MEDIUMBLOB , MEDIUMTEXT

  最大長が 16777215 (2^24 - 1) 文字の BLOB 型または TEXT 型のカラム。 See 項6.5.3.1.「カラムの暗黙的な変更」。 See 項6.2.3.2.「BLOB 型と TEXT 型」。

- LONGBLOB , LONGTEXT

  最大長が 4294967295 または 4G (2^32 - 1) バイトの BLOB 型または TEXT 型のカラム。 See 項6.5.3.1.「カラムの暗黙的な変更」。 MySQL バージョン 3.23 まで、サーバ/クライアントプロトコルおよび MyISAM テーブルでは、通信パケットまたはテーブルレコードごとに 16M の制約があった。バージョン 4.x 以降、LONGTEXT 型または LONGBLOB 型のカラムで許容される最大長は、クライアント/サーバプロトコル間の通信バッファおよび使用可能なメモリ量にしたがって調整された最大パケットサイズによって決まる。 See 項6.2.3.2.「BLOB 型と TEXT 型」。

- ENUM('value1','value2',...)

  列挙。値のリスト 'value1'、'value2'、...、NULL または特殊な "" エラー値から選択された、1 つの値のみ持つことができる文字列オブジェクト。ENUM には最大 65535 の重複のない値を組み込むことができる。 See 項6.2.3.3.「ENUM 型」。

- SET('value1','value2',...)

  セット。0 個以上の値を持つことができる文字列オブジェクト。これらの値はいずれも値のリスト 'value1', 'value2', ... から選択する必要がある。1 つの SET には、最大 64 個の要素を組み込むことができる。 See 項6.2.3.4.「SET 型」。

6.2.1. 数値型

MySQL では、SQL-92 のすべての数値データ型をサポートしています。これらのデータ型は、正確な数値データ型 (NUMERIC、DECIMAL、INTEGER、SMALLINT) だけでなく、近似数値データ型 (FLOAT、REAL、DOUBLE PRECISION) を含みます。キーワード INT は INTEGER のシノニムで、キーワード DEC は DECIMAL のシノニムです。

MySQL では、NUMERIC 型と DECIMAL 型は、SQL-92 標準で使用可能なデータ型と同じデータ型として実装されます。これらのデータ型は、金額データに関する値など、正確な精度で保存することが重要となる値に対して使用されます。こ

れらのいずれかの型のカラムを宣言する際には、次のように、精度とスケールを指定することができます (通常、これらが指定されます) 。

```
salary DECIMAL(5,2)
```

この例で、5 (precision) は、値に対して格納される 10 進数の桁数を表わし、2 (scale) は、小数点に続いて格納される桁数を表わします。したがって、この場合、salary カラムに格納できる値の範囲は、-99.99 〜 99.99 になります (MySQL では、正数の符号を格納する必要がないため、このカラムには、実際には、999.99 までの数値を格納することができます) 。

SQL-92 では、構文 DECIMAL(p) は DECIMAL(p,0) と同じです。同様に、構文 DECIMAL は DECIMAL(p,0) と同じです。この場合、p の値を決定する実装を行うことができます。MySQL では、現在のところ、DECIMAL および NUMERIC データ型のこれらの異型をサポートしていません。通常、これらの型の主な利点は精度とスケールを明示的に制御できることによるため、これはそれほど問題にはなりません。

DECIMAL 値と NUMERIC 値は、値の小数部の精度を維持するため、バイナリの浮動小数点数としてではなく、文字列として格納されます。値の各桁、小数点 (scale > 0 の場合) 、そして '-' 符号 (負数の場合) に対して、1 文字が使用されます。scale が 0 の場合、DECIMAL 値と NUMERIC 値には小数点も小数部も含まれません。

DECIMAL 値と NUMERIC 値の最大範囲は DOUBLE 値と同じですが、個々の DECIMAL または NUMERIC カラムの実際の範囲は、個々のカラムの precision または scale によって制限されます。指定されている scale で許容される桁数を超える桁数を小数部に持つ値がカラムに割り当てられた場合、値は指定されている scale に合わせて丸められます。指定されている (またはデフォルトの) precision と scale によって暗黙的に指定された範囲を超える大きさの値が DECIMAL または NUMERIC カラムに割り当てられた場合、その範囲の最大値が格納されます。

SQL-92 標準の拡張として、MySQL では、前出の表に挙げているように、TINYINT、MEDIUMINT、および BIGINT 型もサポートしています。もう 1 つの拡張として、MySQL には、INT(4) のように、型の基本キーワードに続いて整数値の表示幅をかっこ内に指定できるオプションがあります。このオプションの表示幅の指定は、カラムに指定された幅より小さい幅を持つ値で表示の左側を埋める目的で使用されますが、そのカラムに格納できる値の範囲が制限されたり、そのカラムに指定された幅を超える幅を持つ値の桁数が制限されたりすることはありません。オプションの拡張属性 ZEROFILL と組み合せて使用した場合、デフォルトのスペースに代わってゼロが埋め込まれます。 たとえば、INT(5) ZEROFILL として宣言されたカラムの場合、値 4 は 00004 として取り出されます。注意: 整数カラムの表示幅より大きい値を格納すると、MySQL で一部の複雑な結合のテンポラリテーブルを生成するときに問題が発生することがあります。この場合、データはあくまでも本来のカラム幅に合っているものとして扱われます。

すべての整数型には、オプション (非標準) 属性 UNSIGNED を設定することができます。符号なしの値は、正数値だけを入力できるようにしたいカラムで、やや大きい数値範囲を必要とする場合に使用することができます。

MySQL 4.0.2 以降では、浮動小数点型にも UNSIGNED を設定することができます。 この属性が指定されていると、整数型の場合と同じように、カラムに負数の値を格納できなくなりますが、整数型とは異なり、カラム値の上の範囲は変わりません。

FLOAT 型は近似数値データ型を表現する目的で使用されます。 SQL-92 標準では、キーワード FLOAT に続くかっこ内にオプションの精度をビットで指定することができます (指数の範囲は指定できません) 。このオプションの精度指定は MySQL 実装でもサポートしています。精度を指定しないでカラムに対して FLOAT キーワードを使用した場合、MySQL では 4 バイトを使って値が格納されます。FLOAT キーワードに続けてかっこ内に 2 つの数字を指定する可変の構文も使用できます。このオプションでは、最初の数字は値に必要なバイト単位の記憶容量を表わし、2 番目の数字は格納および表示する小数部の桁数を表わします (DECIMAL および NUMERIC と同様) 。MySQL では、カラムに対して指定されている小数部桁数を超える数の桁を格納しようとすると、格納時に値が丸められ、余分な桁が削除されます。

REAL 型と DOUBLE PRECISION 型では精度の指定は行えません。SQL-92 標準の拡張として、MySQL では、DOUBLE 型は DOUBLE PRECISION 型のシノニムとして認識されます。SQL-92 標準では、REAL 型の精度は DOUBLE PRECISION 型で使用されている精度より小さくなければならないのに対し、MySQL では、これらの両方が 8 バイトの倍精度浮動小数点値として実装されます (非 ``ANSI モード'' で実行した場合) 。 最大限の移植性を確保するためには、近似数値データ値の格納を必要とするコードでは、FLOAT または DOUBLE PRECISION の使用時に精度と小数部の桁数をいずれも指定しないようにします。

数値型のカラムに、そのカラム型で許容されている範囲を超える値を格納しようとすると、MySQL では、値は許容範囲の最大値または最低値に丸められて格納されます。

たとえば、INT カラムの範囲は -2147483648 から 2147483647 です。この場合、INT カラムに -9999999999 という値を挿入しようとすると、値は範囲の最低値に丸められ、-2147483648 として格納されます。同様に、9999999999 という値を挿入しようとすると、値は 2147483647 として格納されます。

INT カラムが UNSIGNED として設定されている場合、このカラムの範囲のサイズは変わりませんが、最小値と最大値はそれぞれ 0 と 4294967295 になります。 したがって、-9999999999 および 9999999999 という値を格納しようとすると、このカラムにはそれぞれ 0 および 4294967296 という値が格納されます。

ALTER TABLE、LOAD DATA INFILE、UPDATE、および複数行の INSERT ステートメントでは、切り落としによる値の変換は ``警告'' として報告されます。

型	バイト	最小値	最大値
TINYINT	1	-128	127
SMALLINT	2	-32768	32767
MEDIUMINT	3	-8388608	8388607
INT	4	-2147483648	2147483647
BIGINT	8	-9223372036854775808	9223372036854775807

6.2.2. 日付と時刻型

日付と時刻型には、DATETIME、DATE、TIMESTAMP、TIME、YEAR があります。これらの型のカラムは、いずれも、一定の範囲の正しい値を取りますが、その他に、実際には不正な値を指定するときに使用される ``ゼロ'' の値も取ります。MySQL では、'厳密' には正しくない一部の日付値 (1999-11-31 など) も格納可能であることに注意してください。 これは、日付チェックはアプリケーションで実行されるので SQL サーバ側で行う必要はない、という前提に立っているためです。日付チェックを '迅速' に行うために、MySQL では、指定された月が 0 ー 12 の範囲にあるかどうかと、指定された日付が 0 ー 31 の範囲にあるかどうかのみチェックします。これらの範囲に 0 が含まれている理由は、MySQL では、DATE または DATETIME カラムの日の値または月日の値がゼロである日付の格納が許容されているためです。これは、誕生日を格納する必要があるアプリケーションで正確な日付がわからないときなどに非常に役立ちます。この場合、単に 1999-00-00 や 1999-01-00 などとして、日付を格納します (このような日付を指定した場合、DATE_SUB() や DATE_ADD などの関数によって正しい値が返ると想定することはできません) 。

以下に、日付と時刻型に関して留意すべき一般考慮事項をいくつか示します。

- MySQL では、個々の日付または時刻型の値の取り出しは標準形式で行われるが、入力した値 (たとえば、日付または時刻型に割り当てる値、またはこれらの型と比較する値の指定時など) については、さまざまな形式で解釈が試みられ

る。ただし、サポートしている形式は、以降のセクションで説明している形式に限られる。正しい値を指定することが前提となるため、非サポート形式で値を指定すると、予測できない結果が発生する場合がある。

- MySQL では、値の解釈が複数の形式で試みられるが、日付値の年部分は必ず左端と想定される。日付は、年-月-日の順序 (例: '98-09-04') で指定する必要がある。一般に使用されている月-日-年や日-月-年の順序 (例: '09-04-98'、'04-09-98') は使用しない。
- MySQL では、日付または時刻型の値を数値型のコンテキストで使用した場合、日付または時刻型の値は数値に自動的に変換される。その逆の場合も、同様の変換が行われる。
- MySQL では、日付または時刻型で範囲外の値や不正な値 (このセクションの最初の部分を参照) を入力すると、値はその型の ``ゼロ" 値に変換される ( この例外として、範囲外の TIME 値は TIME 型の範囲の最大値または最小値に切り落とされる ) 。次の表に、それぞれの型の ``ゼロ" 値を示す。

カラム型	``ゼロ" 値
DATETIME	'0000-00-00 00:00:00'
DATE	'0000-00-00'
TIMESTAMP	00000000000000 (長さは表示サイズに依存)
TIME	'00:00:00'
YEAR	0000

- ``ゼロ" 値は特殊な値だが、上の表に示した値を使用することによって、``ゼロ" 値を明示的に格納または参照することができる。また、この他に、より簡単に記述できる値 '0' または 0 も、``ゼロ" 値の格納または参照に使用できる。
- ODBC では、日付または時刻型の値 ``ゼロ" を処理できないため、MyODBC バージョン 2.50.12 以降では、MyODBC を通して日付または時刻型の値 ``ゼロ" を使用すると、自動的に NULL に変換される。

6.2.2.1. 西暦 2000 年問題と日付型

MySQL 自体は西暦 2000 年問題に対応していますが (see 項1.2.5.「西暦 2000 年対応」) 、MySQL への入力値がこの問題に対応していない場合があります。2 桁の年の値を持つ入力値は、世紀がわからないためいずれもあいまいです。MySQL では年の値は内部で 4 桁を使用して格納されるため、このような値は 4 桁形式に変換する必要があります。

DATETIME 型、DATE 型、TIMESTAMP 型、YEAR 型のカラムであいまいな年の値を含む日付が指定された場合、MySQL では、次の規則に従ってそれらの日付が解釈されます。

- 範囲 00-69 の年の値は 2000-2069 に変換
- 範囲 70-99 の年の値は 1970-1999 に変換

これらの規則では、入力したデータが何を表すかに関して、単に、妥当な推測が行われるだけです。MySQL で使用される発見的手法で正しい年の値が生成されない場合は、4 桁の年の値を含む、あいまいでない値を入力してください。

ORDER BY では、2 桁の YEAR/DATE/DATETIME 型が正しくソートされます。

一部の関数 (MIN() や MAX()など) では、TIMESTAMP/DATE 型の値が数値に変換されることに注意してください。した

がって、2 桁の年を持つタイムスタンプはこれらの関数では正常に機能しません。この場合の修正方法としては、TIMESTAMP/DATE を 4 桁の年形式に変換するか、または MIN(DATE_ADD(timestamp,INTERVAL 0 DAYS)) のようなものを使用します。

6.2.2.2. DATETIME、DATE、TIMESTAMP 型

データ型 DATETIME、DATE、TIMESTAMP はそれぞれ関連しています。 このセクションでは、これらのデータ型の特徴を示すとともに、これらのデータ型の類似点と相違点について説明します。

DATETIME 型は、日付と時刻の両方の情報を含む値を必要とするときに使用します。MySQL では、DATETIME 型の値の取り出しと表示は 'YYYY-MM-DD HH:MM:SS' 形式で行われます。サポートしている範囲は '1000-01-01 00:00:00' ー '9999-12-31 23:59:59' です (``サポート" 範囲より前の値でも動作する場合がありますが、確実に動作するという保証はありません) 。

DATE 型は、日付値のみ必要とし、時刻部分は必要でない場合に使用されます。MySQL では、DATE 型の値の取り出しと表示は 'YYYY-MM-DD' 形式で行われます。サポートしている範囲は、'1000-01-01' ー '9999-12-31' です。

TIMESTAMP カラム型の特性と動作は、MySQL のバージョンとサーバでの SQL 実行モードに応じて異なります。

MAXDB モードでの実行時の TIMESTAMP の動作

MySQL を MAXDB モードで実行している場合、TIMESTAMP は DATETIME と同じように動作します。TIMESTAMP カラムの自動更新 (次の段落で説明) は行われません。MySQL の MAXDB モードでの実行は、バージョン 4.1.1 以降で可能です。See 項4.1.1.「mysqld コマンドラインオプション」。

MAXDB モードで実行していないときの TIMESTAMP の動作

TIMESTAMP カラム型では、INSERT または UPDATE 操作に対して現在の日付と時刻を自動的に指定することができます。TIMESTAMP カラムが複数ある場合は、最初のカラムのみが自動で更新されます。

最初の TIMESTAMP カラムの自動更新は、次のいずれかの条件で発生します。

- INSERT または LOAD DATA INFILE ステートメントで、カラムが明示的に指定されていない場合。
- UPDATE ステートメントおよびその他の何らかのカラム変更値でカラムが明示的に指定されていない場合 (注意: カラムにすでに設定されている値を設定しようとする UPDATE では、TIMESTAMP カラムは更新されない。カラムに現在の値を設定しようとしても、MySQL では、効率性を考慮して更新操作が無視される) 。
- TIMESTAMP カラムに値 NULL が明示的に設定された場合。

2 番目以降の TIMESTAMP カラムにも、現在の日付と時刻を設定することができます。カラムに NULL または NOW() を設定します。

TIMESTAMP 型のいずれかのカラムに現在の日付と時刻以外の値を設定するには、そのカラムに対して、必要な値を明示的に設定します。これは、最初の TIMESTAMP カラムについても同様です。この特性は、次に示すように、レコードの作成時に TIMESTAMP カラムに現在の日時を設定し、その後そのレコードの更新時には設定済の値を変更しないようにする場合などに役立ちます。

- レコードの作成時に、MySQL によってカラムを設定する。 それにより、そのカラムが現在の日付と時刻に初期化される。

- そのレコードの他のカラムに対して後続の更新を行ったときには、TIMESTAMP カラムにそのカラムの現在の値を明示的に設定する。

とはいえ、DATETIME カラムを使用してレコードの作成時に値を NOW() に初期化し、後続の更新時にはそのままにしておくというのも、手軽な方法です。

MAXDB モードでの実行時の TIMESTAMP の特性

MySQL を MAXDB モードで実行している場合、TIMESTAMP は DATETIME とまったく変わりません。格納と表示には同じ形式が使用され、また値の範囲も同じです。MySQL の MAXDB モードでの実行は、バージョン 4.1.1 以降で可能です。See 項4.1.1.「mysqld コマンドラインオプション」。

MAXDB モードで実行していないときの、MySQL 4.1 以降での TIMESTAMP の特性

MySQL 4.1.0 では、TIMESTAMP カラムの格納および表示には、 DATETIME カラムと同じ形式が使用されます。したがって、次の段落で説明している方法で狭くしたり、広げたりすることはできません。つまり、TIMESTAMP(2) や TIMESTAMP(4) などは使用できないことになります。それ以外の特性は、以前の MySQL バージョンと同じです。

MySQL 4.1 より前のバージョンにおける TIMESTAMP の特性

TIMESTAMP 値は 1970 年の始まりから 2037 年の一定の時点までを範囲とし、時間分解能は 1 秒です。値は数値として表示されます。

MySQL で TIMESTAMP 型の値の取り出しと表示に使用される形式は、次の表に示すように、表示サイズによって異なります。最長の TIMESTAMP 形式は 14 桁ですが、TIMESTAMP 型のカラムはもっと短い表示サイズで作成することもできます。

カラム型	表示形式
TIMESTAMP(14)	YYYYMMDDHHMMSS
TIMESTAMP(12)	YYMMDDHHMMSS
TIMESTAMP(10)	YYMMDDHHMM
TIMESTAMP(8)	YYYYMMDD
TIMESTAMP(6)	YYMMDD
TIMESTAMP(4)	YYMM
TIMESTAMP(2)	YY

TIMESTAMP 型のカラムでは、表示サイズにかかわらず、格納サイズはすべて同じです。最も一般的な表示サイズは 6、8、12、14 です。テーブルの作成時に任意の表示サイズを指定できますが、値 0 と 14 を超える値は強制的に 14 に設定されます。1 ー 13 の奇数値のサイズは強制的にすぐ上の偶数に設定されます。

注意: バージョン 4.1 以降、TIMESTAMP は 'YYYY-MM-DD HH:MM:SS' 形式の文字列として返されます。その他のタイムスタンプ長のサポートは中止されました。

DATETIME、DATE、TIMESTAMP 型の値は、以下の一連の共通形式のいずれかを使用して指定することができます。

- 'YYYY-MM-DD HH:MM:SS' または 'YY-MM-DD HH:MM:SS' 形式の文字列として指定。``柔軟" な構文が許容される ---

日付部分と時刻部分の区切り記号として、任意の句読文字を使用することができる。 たとえば、'98-12-31 11:30:45'、'98.12.31 11+30+45'、'98/12/31 11*30*45'、'98@12@31 11^30^45' はいずれも同じ。

- 'YYYY-MM-DD' または 'YY-MM-DD' 形式の文字列として指定。 この場合も ``柔軟'' な構文が許容される。たとえば、'98-12-31'、'98.12.31'、'98/12/31'、'98@12@31' はいずれも同じ。
- 'YYYYMMDDHHMMSS' または 'YYMMDDHHMMSS' 形式の、区切り記号のない文字列 (日付として適切なもの) として指定。たとえば、'19970523091528' と '970523091528' は '1997-05-23 09:15:28' として解釈されるが、'971122129015' は正しくないため (分の部分が不適切)、'0000-00-00 00:00:00' になる。
- 'YYYYMMDD' または 'YYMMDD' 形式の、区切り記号のない文字列 (日付として適切なもの) として指定。たとえば、'19970523' と '970523' は '1997-05-23' として解釈される、'971332' は正しくないため (月と日付の部分が不適切)、'0000-00-00' になる。
- YYYYMMDDHHMMSS または YYMMDDHHMMSS 形式の数値 (日付として適切なもの) として指定。たとえば、19830905132800 と 830905132800 は '1983-09-05 13:28:00' として解釈される。
- YYYYMMDD または YYMMDD 形式の数値 (日付として適切なもの) として指定。たとえば、19830905 と 830905 は '1983-09-05' として解釈される。
- DATETIME、DATE、または TIMESTAMP 型のコンテキストで許容される値を返す、NOW() や CURRENT_DATE などの関数の結果として指定。

不適切な DATETIME、DATE、TIMESTAMP 値は、それぞれの型の ``ゼロ'' 値 ('0000-00-00 00:00:00'、'0000-00-00'、00000000000000) に変換されます。

日付部分の区切り記号を含む文字列として値を指定する場合、10 より少ない月または日の値を 2 桁で指定する必要はありません。'1979-6-9' は '1979-06-09' と同じ意味になります。同様に、時刻部分の区切り記号を含む文字列として値を指定する場合、10 より少ない時、分、または秒の値を 2 桁で指定する必要はありません。'1979-10-30 1:2:3' は '1979-10-30 01:02:03' と同じです。

数値として指定する値は 6、8、12、14 のいずれかの桁数にします。数値を 8 桁または 14 桁の長さにすると、YYYYMMDD または YYYYMMDDHHMMSS 形式であり、最初の 4 桁が年であると想定されます。数値を 6 桁または 12 桁の長さにすると、YYMMDD または YYMMDDHHMMSS 形式であり、最初の 2 桁が年であると想定されます。これら以外の長さの数値は、最も近い長さになるよう先頭にゼロが追加された数値として解釈されます。

区切りなしの文字列として指定した値は、その文字列の長さに基づいて解釈されます。文字列が 8 文字または 14 文字の場合、最初の 4 文字が年であると想定されます。それ以外の長さの文字列の場合、最初の 2 文字が年であると解釈されます。文字列は、それに含まれている各部分に対応して、左から右に、年、月、日、時、分、秒の値として解釈されます。したがって、6 文字より少ない文字列は使用できません。たとえば、1999 年 3 月を表わす値として '9903' を指定すると、MySQL では、テーブルに日付値として ``ゼロ'' が格納されます。これは、年と月の値が 99 と 03 として指定されていても日の部分がまったくないことから、正しい日付ではないためです。しかし、MySQL 3.23 以降では、欠落している月または日の部分を表わすゼロの値を明示的に指定することができます。たとえば、'990300' と指定することで、'1999-03-00' という日付値を格納することができます。

TIMESTAMP カラムでは、正しい値は、表示サイズにかかわらず、その値が指定されたときの完全な精度で格納されます。このことは、暗黙的に次のことを意味します。

- カラム型が TIMESTAMP(4) または TIMESTAMP(2) の場合でも、必ず年、月、日を指定する必要がある。このすべて

を指定しないと、値は正しい日付とはならず、0 が格納される。

- ALTER TABLE を使用して表示桁数の小さな TIMESTAMP カラムの桁数を広げると、それまで ``非表示" になっていた情報が表示される。
- 同様に、TIMESTAMP カラムを狭くしても、単に、値を表示したときに表示される情報が少なくなるだけで、情報自体が失われるわけではない。
- TIMESTAMP 型の値は完全な精度で格納されるが、基盤の格納値に対して直接作用する関数は UNIX_TIMESTAMP() だけである。他の関数は、取り出され、形式設定された値に作用する。したがって、HOUR() や SECOND() などの関数は、形式設定された値に TIMESTAMP 値の該当の部分が含まれていない場合、使用できない。たとえば、TIMESTAMP カラムの HH 部分は、表示サイズが 10 以上でないと表示されないため、それより短い TIMESTAMP 値に対して HOUR() 操作を行っても、無意味な値しか得られない。

ある日付型の値を別の日付型のオブジェクトに割り当てることは、ある程度までは可能です。しかし、何らかの値の変化や情報の消失が起こる可能性があります。

- DATE 型の値を DATETIME または TIMESTAMP 型のオブジェクトに割り当てた場合、値に時刻情報が含まれていないため、結果の値の時刻部分は '00:00:00' に設定される。
- DATETIME または TIMESTAMP 型の値を DATE 型のオブジェクトに割り当てた場合、DATE 型には時刻情報が格納されないため、結果の値の時刻部分は削除される。
- DATETIME、DATE、および TIMESTAMP 型の値はいずれも、同じ形式セットで指定することができるが、値の範囲については、すべての型で同じであるわけではない。たとえば、TIMESTAMP 型の値は、1970 より前にしたり、2037 より後にすることはできない。したがって、'1968-01-01' などの日付は、DATETIME 型や DATE 型の値としては正しいが、TIMESTAMP 型の値としては正しくないため、このようなオブジェクトに割り当てると、値は 0 に変換される。

日付値を指定する際には、次の点に注意してください。

- 文字列として指定する値で許容される柔軟な形式は、まぎらわしいことがある。 たとえば、'10:11:12' などの値は、':' 区切り記号のせいで時刻値のように見えるが、日付のコンテキストで使用した場合は、年 '2010-11-12' として解釈される。値 '10:45:15' は、'45' が正しい月ではないため、'0000-00-00' に変換される。
- MySQL サーバでは、日 00-31、月 00-12、年 1000-9999 の有効性に関する基本チェックのみ実行される。この範囲外の日付はすべて 0000-00-00 に戻される。 この場合、2002-04-31 のような誤った日付も格納可能であることに注意する。Web アプリケーションでは、追加のチェックを行わずに、フォームからデータを格納できる。日付が有効なものかどうか確認するには、アプリケーションでチェックを行う必要がある。
- 2 桁で指定された年の値は、何世紀かわからないため、あいまいである。MySQL では、次の規則に基づいて、2 桁の年の値が解釈される。
  - 範囲 00-69 の年の値は 2000-2069 に変換
  - 範囲 70-99 の年の値は 1970-1999 に変換

6.2.2.3. TIME 型

MySQL では、TIME 型の値の取り出しと表示は、'HH:MM:SS' 形式 (時間の部分が大きい値では 'HHH:MM:SS' 形式) で行われます。TIME 値の範囲は '-838:59:59' ～ '838:59:59' です。時間の部分が大きい理由は、TIME 型が 1 日の時刻を表現するためだけでなく、経過時間や 2 つのイベント間の間隔を表現するために使用される場合があるためです。1 日の時刻は 24 時以上になることはありませんが、イベント間の経過時間は 24 時間を大きく上回ったり、マイナスになったりすることがあります。

TIME 型の値は、次に示すさまざまな形式で指定することができます。

- 'D HH:MM:SS.fraction' 形式の文字列として指定 (注意: MySQL では、小数部は今のところ時刻カラムに格納されない) 。``柔軟" な構文として、HH:MM:SS.fraction、HH:MM:SS、HH:MM、D HH:MM:SS、D HH:MM、D HH、SS のいずれかも使用できる。この場合、D は 0 ～ 33 の間の日。
- 区切り記号のない、'HHMMSS' 形式の文字列 (時刻として適切なもの) として指定。たとえば、'101112' は '10:11:12' として認識されるが、'109712' は正しくないため (分部分が不適切) 、'00:00:00' になる。
- HHMMSS 形式の数値 (時刻として適切なもの) として指定。 たとえば、101112 は '10:11:12' として認識される。代替形式として、SS、MMSS、HHMMSS、HHMMSS.fraction も認識される。MySQL では、小数部分は今のところ格納されないことに注意する。
- TIME 型のコンテキストで許容される値を返す、CURRENT_TIME などの関数の結果として指定。

TIME 型の値を時刻部分の区切り記号を含む文字列として指定する場合、10 より少ない時、分、または秒の値を 2 桁で指定する必要はありません。'8:3:2' は '08:03:02' と同じです。

TIME 型のカラムに ``短い" TIME 値を割り当てるときは注意してください。MySQL では、コロンがない場合、右側の桁が秒を表すという前提にたって値を解釈します (この場合、TIME 型の値は 1 日の時刻ではなく、経過時間として解釈されます) 。たとえば、'1112' や 1112 は、'11:12:00' (11 時 12 分) を表すように見えますが、MySQL では、'00:11:12' (11 分 12 秒) として解釈されます。 同様に、'12' や 12 は '00:00:12' として解釈されます。 それに対し、コロンのある TIME 値は、常に 1 日の時刻として扱われます。そのため、'11:12' は、'00:11:12' ではなく '11:12:00' を意味します。

正しい値のうち、TIME 型の範囲外の値は、範囲の最大値または最小値に切り落とされます。たとえば、'-850:00:00' と '850:00:00' は、それぞれ '-838:59:59' と '838:59:59' に変換されます。

不正な TIME 値は '00:00:00' に変換されます。'00:00:00' 自体は正当な TIME 値ですが、元の値が '00:00:00' として指定されたのか、それとも不正な値だったのかを、テーブルに格納されている '00:00:00' 値から見分けることはできないことに注意してください。

6.2.2.4. YEAR 型

YEAR 型は、年を表現するときに使用する 1 バイトの型です。

MySQL では、YEAR 型の値の取り出しと表示は YYYY 形式で行われます。範囲は 1901 ～ 2155 です。

YEAR 型の値は、次に示すさまざまな形式で指定することができます。

- 範囲 '1901' ～ '2155' の、4 桁の文字列として指定。
- 範囲 1901 ～ 2155 の、4 桁の数値として指定。

- 範囲 '00' 〜 '99' の、2 桁の文字列として指定。範囲 '00' 〜 '69' と '70' 〜 '99' の値は、それぞれ範囲 2000 〜 2069 と 1970 〜 1999 の YEAR 値に変換される。
- 範囲 1 〜 99 の、2 桁の数値として指定。範囲 1 〜 69 と 70 〜 99 の値は、それぞれ範囲 2001 〜 2069 と 1970 〜 1999 の YEAR 値に変換される。注意: 2 桁の数値の範囲は、2 桁の文字列の範囲と多少異なる。数値として指定した場合、ゼロを直接入力して 2000 と解釈させることはできない。2000 と解釈させるには、必ず文字列の '0' または '00' として指定する必要がある。数値として指定すると、0000 として解釈される。
- YEAR 型のコンテキストで許容される値を返す、NOW() などの関数の結果として指定。

不正な YEAR 値は 0000 に変換されます。

6.2.3. 文字列型

文字列型には、CHAR、VARCHAR、BLOB、TEXT、ENUM、SET があります。このセクションでは、これらの型の機能を示すとともに、それぞれの記憶容量と、クエリでの使用方法について説明します。

型	最大サイズ	バイト
TINYTEXT または TINYBLOB	2^8-1	255
TEXT または BLOB	2^16-1 (64K-1)	65535
MEDIUMTEXT または MEDIUMBLOB	2^24-1 (16M-1)	16777215
LONGBLOB	2^32-1 (4G-1)	4294967295

6.2.3.1. CHAR 型と VARCHAR 型

CHAR 型と VARCHAR 型は似ていますが、格納および取り出しの方法が異なります。

CHAR 型のカラムの長さは、テーブルの作成時に宣言した長さに固定されます。この長さとして、1 〜 255 の任意の値を指定することができます (MySQL バージョン 3.23 以降では、CHAR 型の長さとして、0 〜 255 が可能)。CHAR 型の値は、格納時に、指定された長さになるよう右側にスペースが埋め込まれます。CHAR 値の取り出し時には、後続のスペースが削除されます。

VARCHAR 型のカラムの値は可変長の文字列です。VARCHAR カラムは、CHAR カラム同様、1 〜 255 の間の任意の長さとして宣言することができます。しかし、CHAR 型の値とは異なり、VARCHAR 型の値は、必要な文字と、長さを記録するための 1 バイトのみで格納されます。値に埋め込み処理が行われることはありません。値の格納時、後続のスペースは削除されます (このスペースの削除は SQL-99 の仕様と異なります)。格納時や取り出し時に、ケースの変換処理は行われません。

CHAR 型または VARCHAR 型のカラムに、そのカラムの最大長を超える値を割り当てると、カラムのサイズに合わせて値が切り捨てられます。

次の表に、さまざまな文字列値を CHAR(4) 型と VARCHAR(4) 型のカラムを格納したときの結果に基づく、これらのカラム型の違いについて示します。

値	CHAR(4)	必要な記憶容量	VARCHAR(4)	必要な記憶容量
''	' '	4 バイト	''	1 バイト

'ab'	'ab '	4 バイト	'ab'	3 バイト
'abcd'	'abcd'	4 バイト	'abcd'	5 バイト
'abcdefgh'	'abcd'	4 バイト	'abcd'	5 バイト

CHAR 型のカラムでは、値の取り出し時に後続のスペースが削除されるため、CHAR(4) 型と VARCHAR(4) 型から取り出した値はそれぞれの場合で変わりません。

CHAR 型と VARCHAR 型のカラム値のソートと比較は、テーブルの作成時に BINARY 属性が指定されている場合を除いて、ケース非依存方式で行われます。BINARY 属性は、カラム値のソートと比較を、MySQL サーバが稼動しているマシンの ASCII 順に従って、ケース依存方式で行うことを表します。BINARY 属性はカラムの格納方法と取り出し方法には影響しません。

バージョン 4.1.0 以降では、カラム型 CHAR BYTE が CHAR BINARY として使用されます。これは互換性を考慮した機能です。

BINARY 属性は強固な属性です。BINARY として設定したカラムを式で使用すると、その式全体が BINARY 値として比較されます。

MySQL では、CHAR 型や VARCHAR 型のカラムの型が、テーブルの作成時に暗黙的に変更される場合があります。 See 項6.5.3.1.「カラムの暗黙的な変更」。

6.2.3.2. BLOB 型と TEXT 型

BLOB 型は、可変長のデータを格納できる、大きなバイナリオブジェクトです。TINYBLOB、BLOB、MEDIUMBLOB、LONGBLOB の 4 つの BLOB 型では、格納可能な値の最大長のみが異なります。

See 項6.2.6.「各カラム型に必要な記憶容量」。

TINYTEXT、TEXT、MEDIUMTEXT、LONGTEXT の 4 つの TEXT 型は、4 つの BLOB 型に対応しており、最大長と記憶容量がそれぞれの BLOB 型と同じです。BLOB 型と TEXT 型の唯一の違いは、ソートと比較が、BLOB 値ではケース依存方式で行われ、 TEXT 値ではケース非依存方式で行われる点です。つまり、TEXT 型は大文字と小文字を区別しない BLOB 型と考えることができます。格納時や取り出し時に、ケースの変換処理は行われません。

BLOB 型または TEXT 型のカラムにそのカラム型の最大長を超える値を割り当てると、カラムのサイズに合わせて値が切り捨てられます。

ほとんどの面において、TEXT 型のカラムは、任意の長さに設定できる VARCHAR 型のカラムとみなすことができます。同様に、BLOB 型のカラムについては、VARCHAR BINARY 型のカラムとみなすことができます。相違点は以下のとおりです。

- MySQL バージョン 3.23.2 以降では、BLOB 型と TEXT 型のカラムには、インデックスを付けることができる。それより前の MySQL バージョンでは、インデックスはサポートしていない。
- VARCHAR 型のカラムでは値の格納時に後続のスペースが削除されるが、BLOB 型と TEXT 型のカラムではこの削除は行われない。
- BLOB 型と TEXT 型のカラムには、DEFAULT 値は設定できない。

バージョン 4.1.0 以降では、LONG 型と LONG VARCHAR 型は MEDIUMTEXT データ型にマップされます。これは互換性を考慮した機能です。

MyODBC では、BLOB 型の値は LONGVARBINARY として定義され、TEXT 型の値は LONGVARCHAR として定義されます。

BLOB 型と TEXT 型の値は極度に長くなることがあるため、これらの値の使用時には、次に示す一定の制約が適用されます。

- GROUP BY または ORDER BY を BLOB 型または TEXT 型のカラムに使用する場合、カラムの値を固定長のオブジェクトに変換する必要がある。標準的な変換方法としては、次のように、SUBSTRING 関数を使用する。

```
mysql> SELECT comment FROM tbl_name,SUBSTRING(comment,20) AS substr
    ->              ORDER BY substr;
```

  この変換を行わないと、ソート時にカラムの最初の max_sort_length バイトだけが使用される。max_sort_length のデフォルト値は 1024。この値は、mysqld サーバの起動時に -O オプションを指定することによって変更可能。次に示すように、カラムの位置を指定するか、エイリアスを使用することによって、BLOB 型または TEXT 型の値を含む式に対してグループ化操作を行うことができる。

```
mysql> SELECT id,SUBSTRING(blob_col,1,100) FROM tbl_name GROUP BY 2;
mysql> SELECT id,SUBSTRING(blob_col,1,100) AS b FROM tbl_name GROUP BY b;
```

- BLOB 型または TEXT 型オブジェクトの最大サイズは、その型によって決まるが、クライアントとサーバ間で実際に送信できる最大値は使用可能なメモリ量と通信バッファのサイズで決まる。メッセージバッファのサイズ (max_allowed_packet) は変更できるが、サーバ側とクライアント側の両方で変更を行う必要がある。 See 項5.5.2.「サーバパラメータのチューニング」。

注意: BLOB 型または TEXT 型のそれぞれの値は、内部で、個別に割り当てられたオブジェクトとして表現されます。これは他のすべてのカラム型と異なります。他のカラム型では、テーブルが開かれたときに、カラムごとに 1 度だけ記憶領域が割り当てられます。

6.2.3.3. ENUM 型

ENUM 型は、テーブルの作成時にカラムの仕様で明示的に列挙された使用可能な値のリストから、通常、値が選択される文字列オブジェクトです。

次に示すように、一定の状況では、空の文字列 ("") または NULL も値として使用できます。

- 不正な値 (使用可能な値のリストに含まれない文字列) を ENUM 型のカラムに挿入すると、特殊なエラー値として、空の文字列が挿入される。この文字列は数値 0 を持つため、'通常' の空の文字列とは区別することができる (これについては後述) 。
- ENUM 型が NULL として宣言されている場合、そのカラムでは NULL も正しい値となり、デフォルト値は NULL になる。ENUM 型が NOT NULL として宣言されている場合は、使用可能な値の最初の要素がデフォルト値として使用される。

次に示すように、各列挙値にはインデックスを付けることができる。

- カラムの仕様の使用可能な要素のリストに含まれる値は 1 から順に番号付けされる。
- 空の文字列エラー値のインデックス値は 0。したがって、次の SELECT ステートメントでは、無効な ENUM が割り当てられているレコードを検索できる。

```
mysql> SELECT * FROM tbl_name WHERE enum_col=0;
```

- NULL 値のインデックスは NULL。

たとえば、ENUM("one", "two", "three") として指定されたカラムの場合、次の値のいずれかを取ります。それぞれの値では、対応するインデックスも示しています。

値	インデックス
NULL	NULL
""	0
"one"	1
"two"	2
"three"	3

列挙には、最大 65535 個の要素を組み込むことができます。

3.23.51 以降、ENUM 値の後続のスペースはテーブルの作成時に自動で削除されます。

ENUM 型のカラムに値を割り当てる際には、大文字と小文字の区別はありません。その後、カラムから取り出される値では、大文字と小文字が区別されます。この区別は、テーブルの作成時に使用可能な値として指定された値に基づいて決まります。

ENUM 型の値を数値のコンテキストで取り出すと、そのカラム値のインデックスが返されます。たとえば、ENUM 型のカラムから次のような数値を取り出すことができます。

```
mysql> SELECT enum_col+0 FROM tbl_name;
```

ENUM 型のカラムに数値を格納すると、その数値はインデックスとして扱われ、そのインデックスを持つ列挙要素が値として格納されます (ただし、LOAD DATA においては異なり、すべての入力が文字列として扱われます)。 まぎらわしいので、ENUM 型の文字列には数値を含めないようにしてください。

ENUM 型の値は、カラムの仕様で列挙要素がリストされたときの順序にもとづいてソートされます (つまり、ENUM はインデックス番号順にソートされます)。たとえば、ENUM("a", "b") の場合、"a" は "b" の前にソートされますが、ENUM("b", "a") の場合は、"b" が "a" の前にソートされます。空の文字列は空以外の文字列の前にソートされ、NULL 値はその他すべての列挙値の前にソートされます。 予期しない結果が起こらないよう、ENUM リストはアルファベット順に指定するようにします。また、GROUP BY CONCAT(col) を使用して、カラムがインデックス番号順ではなく、アルファベット順に確実にソートされるようにすることもできます。

ENUM 型のカラムの使用可能なすべての値が必要な場合は、SHOW COLUMNS FROM table_name LIKE enum_column_name を使用し、2 番目のカラムで ENUM 定義を解析します。

6.2.3.4. SET 型

SET 型は、ゼロ以上の値を持つことができる文字列オブジェクトです。これらの値はいずれもテーブルの作成時に指定された使用可能な値のリストから選択する必要があります。SET 型のカラム値が複数の要素で構成される場合は、各要素間をカンマ (‘,’) で区切ります。このことから、SET 要素の値自体には、カンマを使用できません。

たとえば、SET("one", "two") NOT NULL として指定されたカラムは、次に示す値のいずれかを取ります。

```
""
"one"
"two"
"one,two"
```

SET には最大 64 個の異なる要素を組み込むことができます。

3.23.51 以降、SET 値の後続のスペースはテーブルの作成時に自動で削除されます。

MySQL では、SET 値は数値として格納されます。格納値の最下位のビットが最初のセット要素に対応します。SET 値を数値型のコンテキストで取り出すと、取り出される値は、カラム値を構成するセット要素に対応するビットセットを持ちます。たとえば、次のように、SET 型のカラムから数値を取り出すことができます。

```
mysql> SELECT set_col+0 FROM tbl_name;
```

SET 型のカラムに数値を格納する場合、その数値のバイナリ表現に設定されたビットによって、カラム値のセット要素が決まります。たとえば、カラムが SET("a","b","c","d") として指定されているとします。この場合、セット要素は次のビット値を持ちます。

SET 要素	10 進数	2 進数
a	1	0001
b	2	0010
c	4	0100
d	8	1000

このカラムに値 9 を割り当てた場合、2 進数では 1001 になるため、SET 値の 1 番目と 4 番目の要素である "a" と "d" が選択され、結果の値は "a,d" になります。

SET 要素を複数持つ値では、値の挿入時には、要素を列記する順序は重要ではありません。値の中で特定の要素を列記する回数も重要ではありません。 その後、値を取り出すときには、値内の各要素は 1 回のみ出現し、テーブルの作成時に指定された順序でそれぞれの要素が列記されます。たとえば、カラムが SET("a","b","c","d") として指定されている場合、このカラムの値を取り出したときには、"a,d"、"d,a"、"d,a,a,d,d" はいずれも "a,d" として表現されます。

SET 型のカラムに非サポート値を設定すると、その値は無視されます。

SET 型の値は数値としてソートされます。NULL 値は非 NULL SET 値より前にソートされます。

次に示すように、通常、SET 型のカラムに対する SELECT 操作には、LIKE 演算子または FIND_IN_SET() 関数を使用します。

```
mysql> SELECT * FROM tbl_name WHERE set_col LIKE '%value%';
mysql> SELECT * FROM tbl_name WHERE FIND_IN_SET('value',set_col)>0;
```

次のステートメントも有効です。

```
mysql> SELECT * FROM tbl_name WHERE set_col = 'val1,val2';
mysql> SELECT * FROM tbl_name WHERE set_col & 1;
```

これらのステートメントの最初のものでは、完全に一致するものが検索されます。2 番目のステートメントでは、最初のセット要素を持つ値が検索されます。

SET 型のカラムの使用可能なすべての値が必要な場合は、SHOW COLUMNS FROM table_name LIKE set_column_name を使用し、2 番目のカラムで SET 定義を解析します。

6.2.4. 正しいカラム型の選択

記憶領域を最も効率よく使用するためには、それぞれの場合に最も適した型を選択するようにします。たとえば、1 ～ 99999 の範囲の整数を格納するカラムには、MEDIUMINT UNSIGNED 型が最も適しています。

一般的な問題の 1 つとして、金額値の正確な表現があります。この場合、MySQL では、DECIMAL 型を使用すべきです。この型は文字列として格納されるため、正確さが失われることはありません。正確さがそれほど重要でない場合は、DOUBLE 型でも間に合います。

高精度が必要な場合は、BIGINT として格納される固定小数点型にいつでも変換することができます。それによって、すべての計算を整数で行い、必要に応じて、結果を浮動小数点値に再び変換できます。

6.2.5. 他のデータベースエンジンのカラム型の使用

他のベンダの SQL 実装用に記述されたコードを使いやすくするために、MySQL では、次の表に示すカラム型がマップされます。このマッピングによって、他のデータベースエンジンから MySQL へのテーブル定義の移植が簡単に行えるようになっています。

他のベンダの型	MySQL の型
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

カラム型のマッピングは、テーブルの作成時に行われます。他のベンダで使用されている型でテーブルを作成した後に DESCRIBE tbl_name ステートメントを発行すると、その型に対応する MySQL の型を使用したテーブル構造が報告されます。

6.2.6. 各カラム型に必要な記憶容量

以下に、MySQL でサポートしている各カラム型に必要な記憶容量をカテゴリ別に示します。

数値型に必要な記憶容量

カラム型	必要な記憶容量
TINYINT	1 バイト
SMALLINT	2 バイト
MEDIUMINT	3 バイト
INT	4 バイト
INTEGER	4 バイト
BIGINT	8 バイト
FLOAT(X)	X <= 24 の場合は 4 バイト、25 <= X <= 53 の場合は 8 バイト
FLOAT	4 バイト
DOUBLE	8 バイト
DOUBLE PRECISION	8 バイト
REAL	8 バイト
DECIMAL(M,D)	D > 0 の場合は M+2 バイト、D = 0 の場合は M+1 バイト (M < D の場合は D+2 バイト)
NUMERIC(M,D)	D > 0 の場合は M+2 バイト、D = 0 の場合は M+1 バイト (M < D の場合は D+2 バイト)

日付型と時刻型に必要な記憶容量

カラム型	必要な記憶容量
DATE	3 バイト
DATETIME	8 バイト
TIMESTAMP	4 バイト
TIME	3 バイト
YEAR	1 バイト

文字列型に必要な記憶容量

カラム型	必要な記憶容量
CHAR(M)	M バイト (1 <= M <= 255)

VARCHAR(M)	L+1 バイト (L <= M で、かつ 1 <= M <= 255)
TINYBLOB、TINYTEXT	L+1 バイト (L < 2^8)
BLOB、TEXT	L+2 バイト (L < 2^16)
MEDIUMBLOB、MEDIUMTEXT	L+3 バイト (L < 2^24)
LONGBLOB、LONGTEXT	L+4 バイト (L < 2^32)
ENUM('value1','value2',...)	列挙値の数 (最大 65535 個の値) に応じて、1 または 2 バイト
SET('value1','value2',...)	セット要素の数 (最大 64 個の要素) に応じて、1、2、3、4、8 バイトのいずれか

VARCHAR 型、および BLOB 型と TEXT 型は可変長であり、必要な記憶容量は、その型で使用できる最大サイズではなく、カラム値の実際の長さ (上の表の L) に応じて決まります。たとえば、VARCHAR(10) のカラムには、最大 10 文字の長さの文字列を格納することができます。実際に必要な記憶容量は、文字列の長さ (L) に、その文字列の長さを記録するために必要な 1 バイトを加えたものです。'abcd' という文字列の場合、L は 4 であり、必要な記憶容量は 5 バイトになります。

BLOB 型と TEXT 型では、その型で使用できる最大長に応じて、カラム値の長さを記録するために、1、2、3、または 4 バイトが必要になります See 項6.2.3.2.「BLOB 型と TEXT 型」。

テーブルに可変長のカラム型が含まれている場合、記録形式も可変長になります。 注意: MySQL では、テーブルの作成時、一定の条件下でカラムが可変長型から固定長型 (またはその逆) に変換される場合があります。 See 項6.5.3.1.「カラムの暗黙的な変更」。

ENUM 型オブジェクトのサイズは、異なる列挙値の数で決まります。使用可能な値の数が 255 までの列挙には、1 バイトが使用されます。値の数が 65535 までの列挙には、2 バイトが使用されます。 See 項6.2.3.3.「ENUM 型」。

SET 型オブジェクトのサイズは、異なるセット要素の数で決まります。セットのサイズが N の場合、そのオブジェクトは (N+7)/8 バイト (1、2、3、4、または 8 バイトに切り上げ) を占有します。SET には最大 64 個の要素を組み込むことができます。 See 項6.2.3.4.「SET 型」。

MyISAM テーブルの 1 つのレコードの最大サイズは 65534 バイトです。各 BLOB 型と TEXT 型は、このサイズに対して 5 〜 9 バイトしか占有しません。

6.3. SELECT 節と WHERE 節で使用する関数

SQL ステートメント内の select_expression または where_definition は、以下に説明する関数を使用した任意の式で構成することができます。

NULL を含む式は、その式に含まれている演算子と関数に関する記述で特に断りがなければ、常に NULL 値を生成します。

注意: 関数名とそれに続くかっこの間には、空白は挿入できません。これは、関数呼び出しと、関数とたまたま同じ名前を持つテーブルまたはカラムの参照とを、MySQL のパーサが区別できるようにするためです。しかし、引数の周囲にはスペースを挿入できます。

--ansi を指定して mysqld を起動するか、または mysql_connect() に CLIENT_IGNORE_SPACE を使用すると、MySQL で関数名の後のスペースが許容されるようになりますが、このようにした場合、すべての関数名が予約語になります。 See

項1.8.2.「ANSI モードでの MySQL の実行」。

簡潔にするため、mysql プログラムの出力例では、出力を省略形式で示します。たとえば、次の出力の場合、

```
mysql> SELECT MOD(29,9);
1 rows in set (0.00 sec)

+-----------+
| mod(29,9) |
+-----------+
|         2 |
+-----------+
```

例では、次のように示します。

```
mysql> SELECT MOD(29,9);
       -> 2
```

6.3.1. 各データ型共通の演算子と関数

6.3.1.1. かっこ

```
( ... )
```

式での評価を特定の順序で行うよう強制するには、次のようにかっこを使用します。

```
mysql> SELECT 1+2*3;
       -> 7
mysql> SELECT (1+2)*3;
       -> 9
```

6.3.1.2. 比較演算子

比較演算は、1（TRUE）、0（FALSE）、または NULL の値を返します。このような関数は、数値と文字列の両方に作用します。必要に応じて、文字列は数値に、数値は文字列に、自動的に変換されます（Perl と同様）。

MySQL では、比較は次の規則に基づいて行われます。

- どちらかまたは両方の引数が NULL の場合、<=> 演算子を使用しているときを除いて、比較結果は NULL になる。
- 比較演算の両方の引数が文字列の場合、それらは文字列として比較される。
- 両方の引数が整数の場合、それらは整数として比較される。
- 16 進値は、数値と比較する場合を除いて、バイナリ文字列として扱われる。
- 引数のどちらかが TIMESTAMP または DATETIME 型のカラムで、もう一方が定数の場合、定数は比較の実行前にタイムスタンプに変換される。これは ODBC との互換性を確保するために行われる。
- その他のすべての場合には、引数は浮動小数点数（実数）として比較される。

デフォルトでは、文字列の比較は、現在のキャラクタセット（デフォルトでは ISO-8859-1 Latin1。これは英語でも非常に

よく動作する）を使用して、ケース非依存方式で行われます。

いずれかの標準演算子（= や <> など。LIKE は含まない）を使用してケース非依存文字列を比較する場合、後続の空白（スペース、タブ、改行復帰）は無視されます。

```
mysql> SELECT "a" ="A \n";
    -> 1
```

以下の例は、比較演算での文字列から数値への変換を示したものです。

```
mysql> SELECT 1 > '6x';
    -> 0
mysql> SELECT 7 > '6x';
    -> 1
mysql> SELECT 0 > 'x6';
    -> 0
mysql> SELECT 0 = 'x6';
    -> 1
```

注意: 次のように、文字列型のカラムを数値型と比較する場合、MySQL では、インデックスを使用した値の迅速な検索は実行できません。

```
SELECT * FROM table_name WHERE string_key=1
```

その理由は、値 1 を返す可能性があるさまざまな文字列（"1"、" 1"、"1a" など）があるためです。

- =

  等しい

  ```
  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
  ```

- <> , !=

  等しくない

  ```
  mysql> SELECT '.01' <> '0.01';
      -> 1
  mysql> SELECT .01 <> '0.01';
      -> 0
  mysql> SELECT 'zapp' <> 'zappp';
  ```

```
    -> 1
```

- **<=**

  より小さいか等しい

```
mysql> SELECT 0.1 <= 2;
    -> 1
```

- **<**

  より小さい

```
mysql> SELECT 2 < 2;
    -> 0
```

- **>=**

  より大きいか等しい

```
mysql> SELECT 2 >= 2;
    -> 1
```

- **>**

  より大きい

```
mysql> SELECT 2 > 2;
    -> 0
```

- **<=>**

  等しい（NULL 対応）

```
mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
    -> 1 1 0
```

- IS NULL , IS NOT NULL

  値が NULL であるかないかのテスト

```
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
```

他のプログラムに対して適切に動作するように、MySQL では、IS NULL の使用時において次の追加機能をサポートしている。

- 直前に挿入されたレコードの検索

  ```
  SELECT * FROM tbl_name WHERE auto_col IS NULL
  ```

  これを無効化するには、SQL_AUTO_IS_NULL=0 を設定する。 See 項5.5.6.「SET 構文」。

- NOT NULL DATE 型と DATETIME 型のカラムでの、特殊な日付 0000-00-00 の検索

  ```
  SELECT * FROM tbl_name WHERE date_column IS NULL
  ```

  これは、一部の ODBC アプリケーションで必要になる（ODBC では 0000-00-00 日付をサポートしていないため）。

- expr BETWEEN min AND max

  expr が min 以上で、max 以下の場合は 1 を返す。それ以外の場合は、0 を返す。すべての引数が同じ型の場合、これは式 (min <= expr AND expr <= max) と同じ。引数の型が異なる場合、前述の規則にもとづいて型変換が実行される。この場合、型変換は 3 つすべての引数に適用される。注意: 4.0.5 より前のバージョンでは、引数は expr の型に変換されていた。

  ```
  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

  NOT (expr BETWEEN min AND max) と同じ。

- expr IN (value,...)

  expr が IN リストのいずれかの値のときは 1 を返す。それ以外の場合は 0 を返す。すべての値が定数なら、値はすべて

expr の型に基づいて評価され、ソートされる。その後、バイナリ検索によってアイテムが検索される。したがって、IN の値リストが定数だけで構成されている場合、IN は非常に迅速に行われる。expr がケース依存の文字列式の場合、文字列の比較はケース依存方式で行われる。

```
mysql> SELECT 2 IN (0,3,5,'wefwf');
    -> 0
mysql> SELECT 'wefwf' IN (0,3,5,'wefwf');
    -> 1
```

IN リストの値の数は、max_allowed_packet 値のみによって制限される。

4.1 以降 (SQL-99 標準に準拠するため)、IN では、左側の式が NULL の場合だけでなく、リストに一致するものが見つからず、リスト内の式の 1 つが NULL の場合にも NULL が返る。

MySQL バージョン 4.1 以降では、IN() 節にサブクエリを組み込むこともできる。 See 項6.4.2.3.「ANY、IN、SOME とともに使用したサブクエリ」。

- expr NOT IN (value,...)

  NOT (expr IN (value,...)) と同じ。

- ISNULL(expr)

  expr が NULL の場合は 1 を返す。それ以外の場合は 0 を返す。

```
mysql> SELECT ISNULL(1+1);
    -> 0
mysql> SELECT ISNULL(1/0);
    -> 1
```

注意: = を使用した NULL 値の比較は常に false になる。

- COALESCE(list)

  リスト内の最初の非 NULL 要素を返す。

```
mysql> SELECT COALESCE(NULL,1);
    -> 1
mysql> SELECT COALESCE(NULL,NULL,NULL);
    -> NULL
```

- INTERVAL(N,N1,N2,N3,...)

N < N1 の場合は 0、N < N2 の場合は 1 (以下同様) というように値を返す。N が NULL の場合は -1 を返す。引数はすべて整数として扱われる。この関数で N1 < N2 < N3 < ... < Nn を正しく動作させるには、引数を整数として扱う必要がある。これはバイナリ検索 (非常に迅速) が使用されるからである。

```
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
```

6.3.1.3. 論理演算子

SQL では、すべての論理演算子は、TRUE、FALSE、または NULL (UNKNOWN) を返します。 MySQL では、これは 1 (TRUE)、0 (FALSE)、NULL として実装されています。ほとんどの場合、これらの値は異なる SQL データベース間で共通していますが、場合によっては、TRUE に対してゼロ以外の値が返ることもあります。

- NOT , !

  論理 NOT。 オペランドが 0 の場合は 1 を返し、ゼロでない場合は 0 を返し、NOT NULL の場合は NULL を返す。

```
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
```

  最後の例の場合、式が (!1)+1 と同様に評価されるため、1 が返る。

- AND , &&

  論理積。 すべてのオペランドがゼロでも NULL でもない場合は 1 を返し、1 つ以上のオペランドが 0 の場合は 0 を返す。それ以外の場合は NULL を返す。

```
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
```

注意: 4.0.5 より前のバージョンの MySQL では、NULL が検出されると、使用可能な 0 値をチェックするプロセスが継続されずに、評価が停止される。そのため、これらのバージョンの場合、SELECT (NULL AND 0) では 0 ではなく NULL が返る。 バージョン 4.0.5 では、引き続き可能な限り最適化を図る一方で、常に SQL標準で規定されたとおりの結果になるようコードが再設計されている。

- OR , ||

  論理和。 いずれかのオペランドが非ゼロの場合は 1 を返し、いずれかのオペランドが NULL の場合は NULL を返します。それ以外の場合は 0 を返す。

```
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

  排他論理和。 いずれかのオペランドが NULL の場合は NULL を返す。 NULL 以外のオペランドに対しては、奇数個のオペランドが非ゼロの場合は 1 を返し、それ以外の場合は 0 を返す。

```

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 は数学的に (a AND (NOT b)) OR ((NOT a) and b) と等価。

  XOR はバージョン 4.0.2 で追加された。

6.3.1.4. フロー制御関数

- CASE value WHEN [compare-value] THEN result [WHEN [compare-value] THEN result ...][ELSE result] END , CASE WHEN [condition] THEN result [WHEN [condition] THEN result ...][ELSE result] END

最初の式では、value=compare-value である result を返す。2 番目の式では、最初の条件が true なら、その result 値を返す。一致する結果が検出されない場合は、ELSE 後の result 値を返す。ELSE 部分がない場合は、NULL を返す。

```
mysql> SELECT CASE 1 WHEN 1 THEN "one"
         WHEN 2 THEN "two" ELSE "more" END;
    -> "one"
mysql> SELECT CASE WHEN 1>0 THEN "true" ELSE "false" END;
    -> "true"
mysql> SELECT CASE BINARY "B" WHEN "a" THEN 1 WHEN "b" THEN 2 END;
    -> NULL
```

戻り値の型（INTEGER、DOUBLE、または STRING）は最初の戻り値（最初の THEN 後の式）の型と同じ。

- IF(expr1,expr2,expr3)

  expr1 が TRUE（expr1 <> 0 および expr1 <> NULL）の場合 IF() は expr2 を返し、それ以外の場合は expr3 を返す。IF() は、使用されているコンテキストに応じて、数値または文字列を返す。

```
mysql> SELECT IF(1>2,2,3);
    -> 3
mysql> SELECT IF(1<2,'yes','no');
    -> 'yes'
mysql> SELECT IF(STRCMP('test','test1'),'no','yes');
    -> 'no'
```

  expr2 または expr3 が明示的に NULL の場合、IF() 関数の結果の型は NULL 以外のカラムの型になる（この動作は MySQL 4.0.3 で新たに導入された）。

  expr1 は整数値として評価される。したがって、浮動小数点型または文字列型の値をテストする場合は、比較演算を使用するようにする。

```
mysql> SELECT IF(0.1,1,0);
    -> 0
mysql> SELECT IF(0.1<>0,1,0);
    -> 1
```

  上の最初の式では、IF(0.1) が 0 を返す。これは、0.1 が整数型の値に変換されることによって、IF(0) のテストが行われるためである。これでは期待した結果が得られない場合がある。2 番目の式では、元の浮動小数点値がゼロ以外の値かどうか比較によってテストされる。比較の結果は整数として使用される。

  IF() のデフォルトの戻り値型は、MySQL バージョン 3.23 では次のように計算される（デフォルトの戻り値型はテンポラリテーブルへの格納時に重要になる場合がある）。

式	戻り値
expr2 または expr3 が文字列を返す。	文字列

expr2 または expr3 が浮動小数点値を返す。	浮動小数点
expr2 または expr3 が整数を返す。	整数

expr2 と expr3 が文字列の場合、両方の文字列がケース非依存のときは結果もケース非依存になる (バージョン 3.23.51 以降) 。

- IFNULL(expr1,expr2)

  expr1 が NULL でない場合は expr1 を返し、それ以外の場合は expr2 を返す。IFNULL() は、使用されているコンテキストに応じて、数値または文字列を返す。

```
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'
```

  MySQL 4.0.6 以降では、IFNULL(expr1,expr2) のデフォルトの結果値は、2 つの式のより ``一般的" な方 (STRING、REAL または INTEGER の順) になる。それ以前の MySQL バージョンとの違いは、式に基づくテーブルの作成時や、IFNULL() から得られた値を MySQL で内部的にテンポラリテーブルに格納する必要があるときに最も顕著である。

```
CREATE TABLE foo SELECT IFNULL(1,"test") as test;
```

  MySQL 4.0.6 では、カラム 'テスト' の型は CHAR(4) だが、それ以前のバージョンでは、BIGINT。

- NULLIF(expr1,expr2)

  expr1 = expr2 が TRUE の場合は NULL を返し、それ以外の場合は expr1 を返す。 これは CASE WHEN x = y THEN NULL ELSE x END と同じ。

```
mysql> SELECT NULLIF(1,1);
    -> NULL
mysql> SELECT NULLIF(1,2);
    -> 1
```

  注意: MySQL では、引数が等しくない場合、expr1 は 2 回評価される。

6.3.2. 文字列関数

文字列関数では、結果が max_allowed_packet サーバパラメータの設定より長くなると、NULL が返ります。 See 項5.5.2.「サーバパラメータのチューニング」。

文字列の位置を処理する関数では、最初の位置は 1 として番号付けされます。

- ASCII(str)

  文字列 str の左端の文字の ASCII コード値を返す。str が空の文字列のときは 0 を返す。str が NULL のときは NULL を返す。

```
mysql> SELECT ASCII('2');
    -> 50
mysql> SELECT ASCII(2);
    -> 50
mysql> SELECT ASCII('dx');
    -> 100
```

  ORD() 関数も参照。

- BIN(N)

  longlong（ BIGINT ）型の数値である N のバイナリの文字列式を返す。これは CONV(N,10,2) と同じ。N が NULL のときは NULL を返す。

```
mysql> SELECT BIN(12);
    -> '1100'
```

- BIT_LENGTH(str)

  文字列 str の長さをビットで返す。

```
mysql> SELECT BIT_LENGTH('text');
    -> 32
```

- CHAR(N,...)

  引数を整数として解釈し、それらの整数の ASCII コード値で表現された文字から成る文字列を返す。NULL 値はスキップされる。

```
mysql> SELECT CHAR(77,121,83,81,'76');
    -> 'MySQL'
mysql> SELECT CHAR(77,77.3,'77.3');
    -> 'MMM'
```

- CHAR_LENGTH(str)

  文字列 str の長さ（文字数）を返す。 マルチバイト文字は 1 文字とみなされる。 したがって、5 個のマルチバイト文字で構成される文字列の場合、LENGTH() では 10 が返るが、CHAR_LENGTH() では 5 が返る。

- CHARACTER_LENGTH(str)

  CHARACTER_LENGTH() は CHAR_LENGTH() のシノニム。

- CONCAT(str1,str2,...)

  引数を連結した結果の文字列を返す。いずれかの引数が NULL のときは NULL を返す。3 つ以上の引数の指定が可能。数値型の引数は同等の文字列形式に変換される。

```
mysql> SELECT CONCAT('My', 'S', 'QL');
    -> 'MySQL'
mysql> SELECT CONCAT('My', NULL, 'QL');
    -> NULL
mysql> SELECT CONCAT(14.3);
    -> '14.3'
```

- CONCAT_WS(separator, str1, str2,...)

  CONCAT_WS() は CONCAT() の特殊型で、区切り文字付きの CONCAT（CONCAT With Separator）。最初の引数は残りの引数の区切り文字である。 この区切り文字は連結する各文字列の間に挿入される。 区切り文字は残りの引数同様、文字列として指定することができる。区切り文字が NULL の場合、結果は NULL になる。 区切り文字引数の後の NULL 値はスキップされる。

```
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'
```

  MySQL 4.1.1 より前のバージョンの CONCAT_WS() では、NULL 値だけでなく、空の文字列もスキップされる。

- CONV(N,from_base,to_base)

  数値を、進数の異なる数値に変換する。数値 N を from_base 進数から to_base 進数に変換した数値の文字列表現を返す。いずれかの引数が NULL のときは NULL を返す。 引数 N は整数として解釈されるが、整数または文字列として指定することができる。最小の進数は 2 で、最大の進数は 36。to_base が負数の場合、N は符号付きの数値とみなされる。それ以外の場合、N は符号なしの数値として処理される。CONV は 64 ビットの精度で動作する。

```
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'
```

- ELT(N,str1,str2,str3,...)

  N = 1 のときは str1 を返し、N = 2 のときは str2 を返す（以下同様）。N が 1 より小さいか、引数の数より大きい場合は、NULL を返す。ELT() は FIELD() の逆。

```
mysql> SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');
    -> 'ej'
mysql> SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo');
    -> 'foo'
```

- EXPORT_SET(bits,on,off,[separator,[number_of_bits]])

  'bits' に指定された数値に対し、1 のビットを 'on' で指定された文字列で表現し、0 のビットを 'off' で指定された文字列で表現した文字列を返す。各文字列の間は 'separator' に指定された区切り文字'（デフォルトは ','）で区切られ、'number_of_bits' に指定されたビット数（デフォルトは 64）だけが表示される。

```
mysql> SELECT EXPORT_SET(5,'Y','N',',',4)
    -> Y,N,Y,N
```

- FIELD(str,str1,str2,str3,...)

  str1, str2, str3, ... リスト内の str のインデックスを返す。 str が検出されない場合は 0 を返す。 FIELD() は ELT() の逆。

```
mysql> SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
    -> 2
mysql> SELECT FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');
    -> 0
```

- FIND_IN_SET(str,strlist)

  N 個の部分文字列で構成されるリスト strlist に、文字列 str が含まれている場合は、1 から N までのいずれかの値を返

す。文字列のリストは、それぞれの間を ‘,’ 文字で区切られた各部分文字列で構成される文字列である。最初の引数が定数文字列で、2 番目の引数が SET 型のカラムの場合、FIND_IN_SET() 関数はビット演算を使用するよう最適化される。 str が strlist に含まれていない場合や、strlist が空の文字列の場合は、0 を返す。どちらの引数も NULL の場合は、NULL を返す。 最初の引数にカンマ ‘,’ が含まれていると、この関数は正しく動作しない。

```
mysql> SELECT FIND_IN_SET('b','a,b,c,d');
    -> 2
```

- HEX(N_or_S)

  N_OR_S が数値の場合、longlong (BIGINT) 型の数値である N の 16 進値の文字列表現を返す。 これは CONV(N,10,16) と同じ。

  N_OR_S が文字列の場合は、N_OR_S の各文字を 2 桁の 16 進数に変換した N_OR_S の 16 進文字列を返す。これは 0xff 文字列の逆。

```
mysql> SELECT HEX(255);
    -> 'FF'
mysql> SELECT HEX("abc");
    -> 616263
mysql> SELECT 0x616263;
    -> "abc"
mysql> SELECT HEX('あ');
    -> 'A4A2'   ; # 注意 EUC-JP コードでの結果
```

- INSERT(str,pos,len,newstr)

  str 内の、位置 pos から始まる長さ len の部分文字列を文字列 newstr に置き換えた文字列を返す。

```
mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
    -> 'QuWhattic'
mysql> SELECT INSERT('あいうえお', 3, 2, 'か');
    -> あいかお
```

  この関数はマルチバイト文字に対応している。

- INSTR(str,substr)

  文字列 str に最初に出現する部分文字列 substr の位置を返す。これは LOCATE() の 2 つの引数を使用する形式と同じだが、引数の順序が入れ替わっている。

```
mysql> SELECT INSTR('foobarbar', 'bar');
```

```
    -> 4
mysql> SELECT INSTR('xbar', 'foobar');
    -> 0
mysql> SELECT INSTR('あいうえお', 'う');
    -> 3
```

この関数はマルチバイト文字に対応している。 MySQL 3.23 では、この関数では大文字と小文字が区別されるが（ケース依存）、バージョン 4.0 では、どちらかの引数がバイナリ文字列の場合にのみケース依存になる。

- LCASE(str)

  LCASE() は LOWER() のシノニム。

- LEFT(str,len)

  文字列 str の左端にある長さ len の文字を返す。

```
mysql> SELECT LEFT('foobarbar', 5);
    -> 'fooba'
mysql> SELECT LEFT('あいうえお', 2);
    -> 'あい'
```

- LENGTH(str)

  文字列 str の長さ（バイト）を返す。1個のマルチバイト文字は複数バイトになる。 したがって、5 個の 2 バイト文字で構成される文字列の場合、LENGTH() では 10 が返るが、CHAR_LENGTH() では 5 が返る。

```
mysql> SELECT LENGTH('text');
    -> 4
mysql> SELECT LENGTH('あいうえお');
    -> 10
```

- LOAD_FILE(file_name)

  ファイルを読み取り、ファイルの内容を文字列として返す。ファイルはサーバ上に存在しなければならず、ファイルのフルパス名を指定する必要がある。また、FILE 権限が必要となる。ファイルは全ユーザが読み取り可能でなければならず、max_allowed_packet より小さいサイズでなければならない。

  ファイルが存在しない場合や、上記の理由のいずれかによって読み取り不可能な場合、この関数は NULL を返す。

```
mysql> UPDATE tbl_name
       SET blob_column=LOAD_FILE("/tmp/picture")
       WHERE id=1;
```

MySQL バージョン 3.23 以外のバージョンを使用している場合は、ファイルの読み取りをアプリケーション内で行い、その後 INSERT/UPDATE ステートメントを作成して、アプリケーションからデータベースにファイルの内容を書き出す必要がある。MySQL++ ライブラリを使用している場合は、これを行う方法の 1 つを http://www.mysql.com/documentation/mysql++/mysql++-examples.html で参照できる。

- LOCATE(substr,str) , LOCATE(substr,str,pos)

  最初の構文は、文字列 str に最初に出現する部分文字列 substr の位置を返す。 2 番目の構文は、文字列 str の位置 pos 以降に最初に出現する部分文字列 substr の位置を返す。 str 内に substr が存在しない場合は 0 を返す。

```
mysql> SELECT LOCATE('bar', 'foobarbar');
    -> 4
mysql> SELECT LOCATE('xbar', 'foobar');
    -> 0
mysql> SELECT LOCATE('bar', 'foobarbar',5);
    -> 7
mysql> SELECT LOCATE('え', 'あいうえおうえ');
    -> 4
mysql> SELECT LOCATE('え', 'あいうえおうえ', 5);
    -> 7
```

  この関数はマルチバイト文字に対応している。 MySQL 3.23 では、この関数では大文字と小文字が区別されるが (ケース依存) 、バージョン 4.0 では、どちらかの引数がバイナリ文字列の場合にのみケース依存になる。

- LOWER(str)

  str のすべての文字を現在のキャラクタセット (デフォルトは ISO-8859-1 Latin1) のマッピングに基づいて小文字に変更した文字列を返す。

```
mysql> SELECT LOWER('QUADRATICALLY');
    -> 'quadratically'
mysql> SELECT LOWER('あいうえお');
    -> 'あいうえお'
```

  この関数はマルチバイト文字に対応している。

- LPAD(str,len,padstr)

  文字列 str が len に指定された長さの文字になるよう、str の左側に文字列 padstr を埋め込んだ文字列を返す。str が len より長い場合は、戻り値は len の長さの文字に縮められる。

```
mysql> SELECT LPAD('hi',4,'??');
    -> '??hi'
```

```
#バージョン4.1以上では以下のようになります。
mysql> SELECT LPAD('あいう', 5 , 'X');
    -> 'XXあいう'

#バージョン4.1未満では以下のようになります。
mysql> SELECT LPAD('あいう', 8 , 'X');
    -> 'XXあいう'
mysql> SELECT LPAD('あいう', 8 , 'お');
    -> 'おあいう'
mysql> SELECT LPAD('あいう', 5 , 'X');
    -> 'あい ' ; #注意。最後は1byte
```

- LTRIM(str)

  文字列 str から先頭のスペース文字を削除した文字列を返す。

```
mysql> SELECT LTRIM('  barbar');
    -> 'barbar'
```

- MAKE_SET(bits,str1,str2,...)

  bits に指定されたビットのセットに対応するビットを持つ文字列から成るセット (',' 文字で間を区切られた各部分文字列を含む文字列) を返す。str1 はビット 0 に対応し、str2 はビット 1 に対応する (以下同様に続く) 。str1, str2, ... 内の NULL 文字列は結果には組み込まれない。

```
mysql> SELECT MAKE_SET(1,'a','b','c');
    -> 'a'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice','world');
    -> 'hello,world'
mysql> SELECT MAKE_SET(0,'a','b','c');
    -> ''
```

- MID(str,pos,len)

  MID(str,pos,len) は SUBSTRING(str,pos,len) のシノニム。

- OCT(N)

  longlong 型の数値である N の 8 進値の文字列表現を返す。これは CONV(N,10,8) と同じ。 N が NULL のときは NULL を返す。

```
mysql> SELECT OCT(12);
    -> '14'
```

- OCTET_LENGTH(str)

  OCTET_LENGTH() は LENGTH() のシノニム。

- ORD(str)

  文字列 str の左端の文字がマルチバイト文字の場合に、次の式を使用して計算した、その文字を構成する各文字の ASCII コード値を返す。((最初のバイトの ASCII コード)*256+(2 番目のバイトの ASCII コード))[*256+3 番目のバイトの ASCII コード...] 左端の文字がマルチバイト文字でないときは、ASCII() 関数を使用した場合と同じ値を返す。

```
mysql> SELECT ORD('2');
    -> 50
mysql> SELECT ORD('あ');
    -> 42146  ; #注意 EUC-JP コードでの結果
```

- POSITION(substr IN str)

  POSITION(substr IN str) は LOCATE(substr,str) のシノニム。

- QUOTE(str)

  指定された文字列を、SQL ステートメントで正しいエスケープデータ値として使用できるように引用符で囲んだ文字列を返す。結果の文字列は単一引用符で囲まれ、文字列内に単一引用符 (‘’’) 、バックスラッシュ (‘\’) 、ASCII NUL 、Control-Z が含まれている場合はその直前にバックスラッシュが挿入される。引数が NULL の場合は、``NULL'' という語が単一引用符で囲まれずに返される。 QUOTE() 関数は MySQL バージョン 4.0.3 で追加された。

```
mysql> SELECT QUOTE("Don't");
    -> 'Don\'t!'
mysql> SELECT QUOTE(NULL);
    -> NULL
```

- REPEAT(str,count)

  文字列 str を、count に指定された回数だけ繰り返した文字列を返す。count <= 0 の場合は、空の文字列を返す。str または count が NULL の場合は、NULL を返す。

```
mysql> SELECT REPEAT('MySQL', 3);
    -> 'MySQLMySQLMySQL'
mysql> SELECT REPEAT('げ', 3);
    -> 'げげげ'
```

- REPLACE(str,from_str,to_str)

  文字列 str に含まれる文字列 from_str をすべて文字列 to_str に置換した文字列を返す。

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
    -> 'WwWwWw.mysql.com'
mysql> SELECT REPLACE('あいあいうえお', 'い', 'す');
    -> 'あすあすうえお'
```

  この関数はマルチバイト文字に対応している。

- REVERSE(str)

  文字列 str 内の文字の順序を逆にした文字列を返す。

```
mysql> SELECT REVERSE('abc');
    -> 'cba'
mysql> SELECT REVERSE('あいうえお');
    -> 'おえういあ'
```

  この関数はマルチバイト文字に対応している。

- RIGHT(str,len)

  文字列 str の右端にある長さ len の文字を返す。

```
mysql> SELECT RIGHT('foobarbar', 4);
    -> 'rbar'
mysql> SELECT RIGHT('あいうえお', 4);
    -> 'いうえお'
```

  この関数はマルチバイト文字に対応している。

- RPAD(str,len,padstr)

  文字列 str が len に指定された長さの文字になるよう、str の右側に文字列 padstr を埋め込んだ文字列を返す。str が len より長い場合は、戻り値は len の長さの文字に縮められる。

```
mysql> SELECT RPAD('hi',5,'?');
    -> 'hi???'

#バージョン4.1以上では以下のようになります。
mysql> SELECT RPAD('あい', 5, 'う');
```

```
    -> 'あいううう'

#バージョン4.1未満では以下のようになります。
mysql> SELECT RPAD('あい', 5, 'X');
    -> 'あいX'
mysql> SELECT RPAD('あい', 5, 'う');
    -> 'あい ' ; # 注意: 最後はマルチバイト文字の上位1byte
```

- RTRIM(str)

  文字列 str から後続のスペース文字を削除した文字列を返す。

```
mysql> SELECT RTRIM('barbar   ');
    -> 'barbar'
```

  この関数はマルチバイト文字に対応している。

- SOUNDEX(str)

  文字列 str の soundex 文字列を返す。発音が似通った 2 つの文字列の soundex 文字列は同じものになる。標準の soundex 文字列の長さは 4 文字だが、SOUNDEX() 関数は任意の長さの文字列を返す。返された値に対して SUBSTRING() を使用すると、標準の soundex 文字列が得られる。指定された文字列内の英数字以外の文字は無視される。A 〜 Z の範囲外の各国語のアルファベット文字はすべて母音として扱われる。

```
mysql> SELECT SOUNDEX('Hello');
    -> 'H400'
mysql> SELECT SOUNDEX('Quadratically');
    -> 'Q36324'
```

- expr1 SOUNDS LIKE expr2

  SOUNDEX(expr1)=SOUNDEX(expr2) と同じ (バージョン 4.1 以降でのみ利用可能) 。

- SPACE(N)

  N に指定された数のスペース文字で構成される文字列を返す。

```
mysql> SELECT SPACE(6);
    -> '      '
```

- SUBSTRING(str,pos) , SUBSTRING(str FROM pos) , SUBSTRING(str,pos,len) , SUBSTRING(str FROM pos FOR len)

len 引数のない形式の場合は、文字列 str 内の位置 pos 以降の部分文字列を返す。 len 引数のある形式の場合は、文字列 str の位置 pos 以降の、len に指定された長さの部分文字列を返す。 FROM を使用する形式は SQL-92 の構文。

```
mysql> SELECT SUBSTRING('Quadratically',5);
    -> 'ratically'
mysql> SELECT SUBSTRING('foobarbar' FROM 4);
    -> 'barbar'
mysql> SELECT SUBSTRING('Quadratically',5,6);
    -> 'ratica'
mysql> SELECT SUBSTRING('あいうえおかきくけこ', 5);
    -> 'おかきくけこ'
```

この関数はマルチバイト文字に対応している。

- SUBSTRING_INDEX(str,delim,count)

  文字列 str から、delim に指定された区切り記号の count 個目の出現位置の直前までの部分文字列を返す。 count に指定された数が正数の場合は、（左から数えて）指定された数の最後の区切り記号の左側にあるすべての文字を返す。 count に指定された数が負数の場合は、（右から数えて）指定された数の最後の区切り記号の右側にあるすべての文字を返す。

```
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
    -> 'www.mysql'
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
    -> 'mysql.com'
mysql> SELECT SUBSTRING_INDEX('あいんうえおんかき', 'ん', 2);
    -> 'あいんうえお'
mysql> SELECT SUBSTRING_INDEX('あいんうえおんかき', 'ん', -2);
    -> 'うえおんかき'
```

この関数はマルチバイト文字に対応している。

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

  remstr に指定されたすべてのプリフィックスまたはサフィックス（あるいはその両方）を、文字列 str から削除した文字列を返す。指定子 BOTH、LEADING、TRAILING がいずれも指定されていない場合は、BOTH が指定されたとみなされる。remstr が指定されていない場合は、スペースが削除される。

```
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 'barxxyz');
```

```
    -> 'barx'
mysql> SELECT TRIM(BOTH 'お' FROM 'おあいうえお');
    -> 'あいうえ'
```

この関数はマルチバイト文字に対応している。

- UCASE(str)

  UCASE() は UPPER() のシノニム。

- UPPER(str)

  str のすべての文字を現在のキャラクタセット（デフォルトは ISO-8859-1 Latin1）のマッピングに基づいて大文字に変更した文字列を返す。

```
mysql> SELECT UPPER('Hej');
    -> 'HEJ'
mysql> SELECT UPPER('へい');
    -> 'へい'
```

この関数はマルチバイト文字に対応している。

6.3.2.1. 文字列比較関数

MySQL では、必要に応じて数値から文字列への、また文字列から数値への変換が自動で行われます。

```
mysql> SELECT 1+"1";
    -> 2
mysql> SELECT CONCAT(2,' test');
    -> '2 test'
```

数値を文字列に明示的に変換するには、対象の数値を引数として CONCAT() に渡します。

文字列関数の引数としてバイナリ文字列を指定した場合、返される文字列もバイナリ文字列になります。文字列に変換された数値はバイナリ文字列として扱われます。これは比較にのみ影響します。

通常、文字列の比較でいずれかの式がケース依存の場合、比較はケース依存方式で実行されます。

- expr LIKE pat [ESCAPE 'escape-char']

  SQL の単純な正規表現比較を使用したパターンマッチ。1（TRUE）または 0（FALSE）を返す。LIKE では、次の 2 つのワイルドカード文字をパターン内で使用できる。

文字	説明
%	任意の数の文字（ゼロ文字を含む）と一致する。
_	厳密に 1 つの文字と一致する。

```
mysql> SELECT 'David!' LIKE 'David_';
    -> 1
mysql> SELECT 'David!' LIKE '%D%v%';
    -> 1
```

ワイルドカード文字を通常の文字として扱う場合は、直前にエスケープ文字を付ける。特定の ESCAPE 文字を指定しないと、'\' をエスケープ文字として使用するものとみなされる。

文字列	説明
\%	1 つの % 文字と一致する。
_	1 つの _ 文字と一致する。

```
mysql> SELECT 'David!' LIKE 'David\_';
    -> 0
mysql> SELECT 'David_' LIKE 'David\_';
    -> 1
```

別のエスケープ文字を指定する場合は、次のように、ESCAPE 節を使用する。

```
mysql> SELECT 'David_' LIKE 'David|_' ESCAPE '|';
    -> 1
```

次の 2 つのステートメントは、オペランドのどちらか一方がバイナリ文字列の場合、文字列の比較がケース依存方式で行われることを示している。

```
mysql> SELECT 'abc' LIKE 'ABC';
    -> 1
mysql> SELECT 'abc' LIKE BINARY 'ABC';
    -> 0
```

LIKE は数値式で使用できる (これは SQL-99 の LIKE に対する MySQL の拡張) 。

```
mysql> SELECT 10 LIKE '1%';
    -> 1
mysql> SELECT 'いあ' LIKE '%い%';
    -> 1
mysql> SELECT 'イあ' LIKE '%い%';
    -> 0
# 注意： EUC-JP コードで、'イあ' は 0xA5A4A4A2。'い' は 0xA4A4。
```

注意:MySQL では文字列で C のエスケープ構文 ('\n' など) を使用するため、LIKE の文字列に '\' を挿入するときは、2 倍の数の '\' を使用する必要がある。たとえば、'\n' を検索する場合は、'\\n' と指定する。また、'\' を検索する場合は、'\\\\' と指定する (バックスラッシュはパーサによって一度取り除かれ、パターンマッチの実行時にもう一度取り除かれる。そして、残った 1 つのバックスラッシュがマッチングの対象となる) 。

- MATCH (col1,col2,...)AGAINST (expr [IN BOOLEAN MODE | WITH QUERY EXPANSION])

  MATCH ... AGAINST() は全文検索に使用され、カラム (col1,col2,...) のテキストと、クエリ expr のテキストとの関連性（類似度）を返す。関連性は正の浮動小数点数で表現される。関連性がゼロのときは、類似性がまったくないことを意味する。 MATCH ... AGAINST() は MySQL バージョン 3.23.23 以降で利用できる。IN BOOLEAN MODE 拡張はバージョン 4.0.1 で追加され、WITH QUERY EXPANSION は 4.1.1 で追加された。 詳細および使用例については、項6.8.「MySQL 全文検索」を参照。

- expr NOT LIKE pat [ESCAPE 'escape-char']

  NOT (expr LIKE pat [ESCAPE 'escape-char']) と同じ。

- expr NOT REGEXP pat , expr NOT RLIKE pat

  NOT (expr REGEXP pat) と同じ。

- expr REGEXP pat , expr RLIKE pat

  パターン pat に対する文字列式 expr のパターンマッチを実行する。パターンとして、拡張正規表現を使用できる。See 付録 G. MySQL の正規表現。 expr が pat とマッチする場合は 1 を返し、マッチしない場合は 0 を返す。RLIKE は、mSQL との互換性を確保するための、REGEXP のシノニム。 注意:MySQL では、文字列で C のエスケープ構文（'\n' など）を使用するため、REGEXP の文字列に '\' を挿入するときは、2 倍の数の '\' を使用する必要がある。 MySQL バージョン 3.23.4 以降では、REGEXP は通常（非バイナリ）の文字列に対してはケース非依存。

```
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 と RLIKE では、現在のキャラクタセット（デフォルトでは ISO-8859-1 Latin1）に基づいて文字の型が決定される。

- STRCMP(expr1,expr2)

  STRCMP() 両方の文字列が同じものである場合は 0 を返し、現在のソート順で最初の引数が 2 番目の引数より小さい場合は -1 を返し、それ以外の場合は -1 を返す。

```
mysql> SELECT STRCMP('text', 'text2');
    -> -1
mysql> SELECT STRCMP('text2', 'text');
    -> 1
mysql> SELECT STRCMP('text', 'text');
    -> 0
```

6.3.2.2. ケース依存性

- BINARY

  BINARY 演算子はこの演算子の後に続く文字列をバイナリ文字列にキャストする。 この演算子を使用すれば、BINARY または BLOB として定義されていないカラムに対しても、強制的にケース依存にしたカラムの比較を簡単に行うことができる。

```
mysql> SELECT "a" = "A";
    -> 1
mysql> SELECT BINARY "a" = "A";
    -> 0
```

  BINARY string は CAST(string AS BINARY) の省略形。 See 項6.3.5.「キャスト関数」。 BINARY は MySQL バージョン 3.23.0 で導入された。

  注意: MySQL の一部のコンテキストでは、インデックス付きのカラムを BINARY にキャストすると、インデックスを効率的に使用できなくなる。

BLOB 型の文字列をケース非依存方式で比較する必要がある場合は、比較の実行前に BLOB から大文字に変換することによって、常にこの比較が可能になります。

```
SELECT 'A' LIKE UPPER(blob_col) FROM table_name;
```

まもなく、異なるキャラクタセット間のキャストを導入する予定です。この導入によって、文字列の比較をさらに柔軟に実行できるようになります。

6.3.3. 数値関数

6.3.3.1. 算術演算

通常の算術演算子を利用することができます。注意: '-'、'+'、'*' で、両方の引数が整数の場合、結果は BIGINT (64 ビット) の精度で計算されます。 どちらか一方の引数が符号なしの整数で、もう一方も整数の場合、結果は符合なしの整数になります。 See 項6.3.5.「キャスト関数」。

- +

  加算

```
mysql> SELECT 3+5;
    -> 8
```

- -

  減算

```
mysql> SELECT 3-5;
    -> -2
```

- *

  乗算

```
mysql> SELECT 3*5;
    -> 15
mysql> SELECT 18014398509481984*18014398509481984.0;
    -> 324518553658426726783156020576256.0
mysql> SELECT 18014398509481984*18014398509481984;
    -> 0
```

  最後の式の結果は正しくない。これは、この整数乗算の結果が BIGINT 計算の 64 ビットの範囲を超えるため。

- /

  除算

```
mysql> SELECT 3/5;
    -> 0.60
```

  ゼロで割ると結果は NULL になる。

```
mysql> SELECT 102/(1-1);
    -> NULL
```

  除算が BIGINT 演算で計算されるのは、結果が整数に変換されるコンテキストでその演算が実行された場合に限られる。

6.3.3.2. 数学関数

すべての数学関数は工ラーの場合 NULL を返します。

- -

  単項マイナス。引数の符号を変更する。

```
mysql> SELECT - 2;
    -> -2
```

  注意: BIGINT 型の値に対してこの演算子を使用すると、戻り値は BIGINT になる。したがって、値が -2^63 になる可能性がある整数に対しては - の使用を避ける。

- ABS(X)

  X の絶対値を返す。

```
mysql> SELECT ABS(2);
    -> 2
mysql> SELECT ABS(-32);
    -> 32
```

  この関数は BIGINT 型の値に対して使用しても問題ない。

- ACOS(X)

  X のアークコサイン (つまり、コサインが X である値) を返す。X が -1 から 1 の範囲にないときは NULL を返す。

```
mysql> SELECT ACOS(1);
    -> 0.000000
mysql> SELECT ACOS(1.0001);
    -> NULL
mysql> SELECT ACOS(0);
    -> 1.570796
```

- ASIN(X)

  X のアークサイン (つまり、サインが X である値) を返す。X が -1 ～ 1 の範囲にないときは NULL を返す。

```
mysql> SELECT ASIN(0.2);
    -> 0.201358
mysql> SELECT ASIN('foo');
    -> 0.000000
```

- ATAN(X)

  X のアークタンジェント (つまり、タンジェントが X である値) を返す。

```
mysql> SELECT ATAN(2);
    -> 1.107149
mysql> SELECT ATAN(-2);
    -> -1.107149
```

- ATAN(Y,X) , ATAN2(Y,X)

  2 つの変数 X と Y のアークタンジェントを返す。これは Y / X によるアークタンジェントの計算と似ているが、異なる点として、この場合、両方の引数の符号に基づいて結果の四分円が決定される。

```
mysql> SELECT ATAN(-2,2);
    -> -0.785398
mysql> SELECT ATAN2(PI(),0);
    -> 1.570796
```

- CEILING(X) , CEIL(X)

  X 以上の最も小さい整数値を返す。

```
mysql> SELECT CEILING(1.23);
    -> 2
mysql> SELECT CEIL(-1.23);
    -> -1
```

  CEIL() エイリアスはバージョン 4.0.6 で追加された。

  注意: 戻り値は BIGINT に変換される。

- COS(X)

  X (ラジアン) のコサインを返す。

```
mysql> SELECT COS(PI());
    -> -1.000000
```

- COT(X)

  X のコタンジェントを返す。

```
mysql> SELECT COT(12);
    -> -1.57267341
mysql> SELECT COT(0);
    -> NULL
```

- CRC32(expr)

  巡回冗長検査値を計算し、32 ビットの符号なしの値を返す。 引数が NULL の場合、結果は NULL になる。引数では、文字列の指定が前提となる。文字列以外の値を指定しても、文字列として扱われる。

```
mysql> SELECT CRC32('MySQL');
    -> 3259397556
```

  CRC32() は MySQL 4.1.0 以降で利用できる。

- DEGREES(X)

  引数 X をラジアンから度に変換して返す。

```
mysql> SELECT DEGREES(PI());
    -> 180.000000
```

- DIV

  整数の除算。 FLOOR() と同様。ただし、BIGINT 型の値に対応している。

```
mysql> SELECT 5 DIV 2
    -> 2
```

  DIV は MySQL 4.1.0 の新機能。

- EXP(X)

  e（自然対数の底）を X 乗した値を返す。

```
mysql> SELECT EXP(2);
    -> 7.389056
mysql> SELECT EXP(-2);
    -> 0.135335
```

- FLOOR(X)

  X 以下の最も大きい整数値を返す。

```
mysql> SELECT FLOOR(1.23);
    -> 1
mysql> SELECT FLOOR(-1.23);
```

```
    -> -2
```

注意: 戻り値は BIGINT に変換される。

- GREATEST(X,Y,...)

  最も大きい（つまり、最も大きい値を指定された）引数を返す。 各引数の比較は LEAST の場合と同じ規則に基づいて行われる。

```
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"
```

  バージョン 3.22.5 より前の MySQL では、GREATEST の代わりに MAX() を使用できる。

- LEAST(X,Y,...)

  指定された 2 つ以上の引数の中で、最も小さい（つまり、最も小さい値を指定された）引数を返す。 各引数の比較は、次の規則に基づいて行われる。

  - 戻り値が INTEGER のコンテキストで使用される場合、またはすべての引数に整数値が指定されている場合は、各引数は整数として比較される。
  - 戻り値が REAL のコンテキストで使用される場合、またはすべての引数に実数が指定されている場合は、各引数は実数として比較される。
  - いずれかの引数がケース依存文字列である場合は、各引数はケース依存文字列として比較される。
  - その他の場合、引数はケース非依存文字列として比較される。

```
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"
```

  バージョン 3.22.5 より前の MySQL では、LEAST の代わりに MIN() を使用できる。

- LN(X)

  X の自然対数を返す。

```
mysql> SELECT LN(2);
```

```
    -> 0.693147
mysql> SELECT LN(-2);
    -> NULL
```

この関数は MySQL バージョン 4.0.3 で追加された。 MySQL における LOG(X) のシノニム。

- LOG(X) , LOG(B,X)

  パラメータ 1 つで呼び出された場合、この関数は X の自然対数を返す。

```
mysql> SELECT LOG(2);
    -> 0.693147
mysql> SELECT LOG(-2);
    -> NULL
```

パラメータ 2 つで呼び出された場合、この関数は任意の底 B に対する X の自然対数を返す。

```
mysql> SELECT LOG(2,65536);
    -> 16.000000
mysql> SELECT LOG(1,100);
    -> NULL
```

任意の底を指定できるオプションは MySQL バージョン 4.0.3 で追加された。 LOG(B,X) は LOG(X)/LOG(B) と同じ。

- LOG2(X)

  2 を底とする X の自然対数を返す。

```
mysql> SELECT LOG2(65536);
    -> 16.000000
mysql> SELECT LOG2(-100);
    -> NULL
```

LOG2() は、特定の数値の格納に何ビット必要か調べるのに役立つ。 この関数は MySQL バージョン 4.0.3 で追加された。 それより前のバージョンでは、代わりに LOG(X)/LOG(2) を使用できる。

- LOG10(X)

  10 を底とする X の自然対数を返す。

```
mysql> SELECT LOG10(2);
    -> 0.301030
mysql> SELECT LOG10(100);
    -> 2.000000
mysql> SELECT LOG10(-100);
    -> NULL
```

- MOD(N,M) , %

  モジュロ (C の % 演算子に類する) 。 N を M で割ったときの余りを返す。

```
mysql> SELECT MOD(234, 10);
    -> 4
mysql> SELECT 253 % 7;
    -> 1
mysql> SELECT MOD(29,9);
    -> 2
mysql> SELECT 29 MOD 9;
    -> 2
```

  この関数は BIGINT 型の値に使用しても問題ない。 最後の例は MySQL 4.1 でのみ有効。

- PI()

  π の値を返す。小数部の表示桁数はデフォルトで 5 桁だが、MySQL の内部では、π に完全な倍精度が使用される。

```
mysql> SELECT PI();
    -> 3.141593
mysql> SELECT PI()+0.000000000000000000;
    -> 3.141592653589793116
```

- POW(X,Y) , POWER(X,Y)

  X を Y 乗した値を返す。

```
mysql> SELECT POW(2,2);
    -> 4.000000
mysql> SELECT POW(2,-2);
    -> 0.250000
```

- RADIANS(X)

  引数 X を度からラジアンに変換して返す。

```
mysql> SELECT RADIANS(90);
    -> 1.570796
```

- RAND() , RAND(N)

  0 ～ 1.0 の範囲のランダムな浮動小数点数を返す。 整数の引数 N が指定された場合、その整数がシード値 (繰り返し

可能な数列を生成）として使用される。

```
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
```

ORDER BY 節ではカラムが複数回評価されるため、RAND() の値を含むカラムは、ORDER BY 節では使用できない。しかし、バージョン 3.23 以降では、SELECT * FROM table_name ORDER BY RAND() を使用できる。

これは、SELECT * FROM table1,table2 WHERE a=b AND c<d ORDER BY RAND() LIMIT 1000 のセットからランダムなサンプルを得るのに役立つ。

注意: WHERE 節の RAND() は、WHERE が実行されるたびに再評価される。

RAND() は完全な乱数ジェネレータではなく、むしろ同じ MySQL バージョンのプラットフォーム間で移植可能な乱数を随時、すばやく生成するための方法として利用できる。

- ROUND(X) , ROUND(X,D)

  引数 X を最も近い整数に丸めた値を返す。 引数が 2 つの形式では、小数部の桁数を D に指定された桁数に丸める。

```
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
mysql> SELECT ROUND(23.298, -1);
    -> 20
```

  注意: 引数の値が 2 つの整数の中間にある場合の ROUND() の動作は、C ライブラリの実装に応じて決まる。実装に応じて、最も近い偶数に丸められる場合と、常に切り上げまたは切り下げられる場合、あるいは常にゼロ方向に丸められる場合がある。特定の丸め方法を必要とする場合は、この関数ではなく、TRUNCATE() や FLOOR() などの明確に定義された関数を使用するようにする。

- SIGN(X)

  X が負数か、ゼロか、正数かに応じて、この引数の符号を -1、0、または 1 として返す。

```
mysql> SELECT SIGN(-32);
    -> -1
mysql> SELECT SIGN(0);
    -> 0
mysql> SELECT SIGN(234);
    -> 1
```

- SIN(X)

  X (ラジアン) のサインを返す。

```
mysql> SELECT SIN(PI());
    -> 0.000000
```

- SQRT(X)

  X の負数でない平方根を返す。

```
mysql> SELECT SQRT(4);
    -> 2.000000
mysql> SELECT SQRT(20);
    -> 4.472136
```

- TAN(X)

  X (ラジアン) のタンジェントを返す。

```
mysql> SELECT TAN(PI()+1);
    -> 1.557408
```

- TRUNCATE(X,D)

  数値 X の小数部を D に指定された桁数に切り捨てた値を返す。D が 0 の場合、結果の値は小数点も小数部も持たない。

```
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
```

MySQL 3.23.51 以降、すべての数値はゼロ方向に丸められる。

D が負数の場合、その数値の小数部以外の部分もゼロになる。

```
mysql> SELECT TRUNCATE(122,-2);
    -> 100
```

注意: 小数は通常、正確な数値としてではなく、倍精度の値としてコンピュータに格納されるため、次のようなおかしな結果が出ることがある。

```
mysql> SELECT TRUNCATE(10.28*100,0);
    -> 1027
```

これは、10.28 が実際には 10.2799999999999999 のような値として格納されることによって発生する。

6.3.4. 日付と時刻関数

このセクションでは、時間に関係する値の操作に使用できる関数について説明します。 日付と時刻の各データ型の範囲と、値を指定するときの有効な形式については、項6.2.2.「日付と時刻型」を参照してください。

日付関数の使用例を次に示します。このクエリでは、過去 30 日以内の日付の date_col 値を持つすべてのレコードが選択されます。

```
mysql> SELECT something FROM tbl_name
       WHERE TO_DAYS(NOW()) - TO_DAYS(date_col) <= 30;
```

(このクエリでは、将来の日付を持つレコードも選択されます。)

日付値を前提とする関数は、通常、日付時刻値を受け入れて、時刻部分を無視します。時刻値を前提とする関数は、通常、日付時刻値を受け入れて、日付部分を無視します。

現在の日付または時刻をそれぞれ返す関数は、クエリごとに、そのクエリの開始時に 1 回だけ評価されます。したがって、1 つのクエリ内で NOW() などの関数を何度も参照しても、常に同じ結果になります。この原則は、CURDATE()、CURTIME()、UTC_DATE()、UTC_TIME()、UTC_TIMESTAMP()、およびこれらのあらゆるシノニムに同様に適用されます。

以下の関数の説明で示している戻り値の範囲は、完全な日付に対して適用されます。日付が ``ゼロ" 値であったり、'2001-11-00' などの不完全な日付である場合、日付の一部を取り出す関数では 0 が返ります。たとえば、DAYOFMONTH('2001-11-00') からは 0 が返ります。

- ADDDATE(date,INTERVAL expr type) , ADDDATE(expr,days)

  2 番目の引数の INTERVAL を指定する形式で呼び出した場合、ADDDATE() は DATE_ADD() とシノニムになる。関連する関数 SUBDATE() は DATE_SUB() のシノニム。

```
mysql> SELECT DATE_ADD('1998-01-02', INTERVAL 31 DAY);
```

```
    -> '1998-02-02'
mysql> SELECT ADDDATE('1998-01-02', INTERVAL 31 DAY);
    -> '1998-02-02'
```

2 番目の構文は MySQL 4.1.1 以降で使用できる。expr には日付式か日付時刻式を指定し、days には expr に加える日数を指定する。

```
mysql> SELECT ADDDATE('1998-01-02', 31);
    -> '1998-02-02'
```

- ADDTIME(expr,expr2)

  ADDTIME() では、expr に expr2 を加えた結果が返される。 expr には日付式か日付時刻式を指定し、expr2 には時刻式を指定する。

```
mysql> SELECT ADDTIME("1997-12-31 23:59:59.999999", "1 1:1:1.000002");
    -> '1998-01-02 01:01:01.000001'
mysql> SELECT ADDTIME("01:00:00.999999", "02:00:00.999998");
    -> '03:00:01.999997'
```

  ADDTIME() は MySQL 4.1.1 で追加された。

- CURDATE()

  文字列と数値のどちらのコンテキストで使用されているかに応じて、'YYYY-MM-DD' または YYYYMMDD 形式の値として、現在の日付を返す。

```
mysql> SELECT CURDATE();
    -> '1997-12-15'
mysql> SELECT CURDATE() + 0;
    -> 19971215
```

- CURRENT_DATE , CURRENT_DATE()

  CURRENT_DATE と CURRENT_DATE() は CURDATE() のシノニム。

- CURTIME()

  文字列と数値のどちらのコンテキストで使用されているかに応じて、'HH:MM:SS' または HHMMSS 形式の値として、現在の時刻を返す。

```
mysql> SELECT CURTIME();
```

```
    -> '23:50:26'
mysql> SELECT CURTIME() + 0;
    -> 235026
```

- CURRENT_TIME , CURRENT_TIME()

  CURRENT_TIME と CURRENT_TIME() は CURTIME() のシノニム。

- CURRENT_TIMESTAMP , CURRENT_TIMESTAMP()

  CURRENT_TIMESTAMP と CURRENT_TIMESTAMP() は NOW() のシノニム。

- DATE(expr)

  日付式または日付時刻式 expr の日付部分を取り出す。

```
mysql> SELECT DATE('2003-12-31 01:02:03');
    -> '2003-12-31'
```

  DATE() は MySQL 4.1.1 以降で使用できる。

- DATEDIFF(expr,expr2)

  DATEDIFF() は開始日 expr と終了日 expr2 との間の日数を返す。 expr と expr2 には、日付式か、日付時刻式を指定する。 計算は値の日付部分のみに基づいて行われる。

```
mysql> SELECT DATEDIFF('1997-12-31 23:59:59','1997-12-30');
    -> 1
mysql> SELECT DATEDIFF('1997-11-31 23:59:59','1997-12-31');
    -> -30
```

  DATEDIFF() は MySQL 4.1.1 で追加された。

- DATE_ADD(date,INTERVAL expr type) , DATE_SUB(date,INTERVAL expr type)

  これらの関数では日付演算が実行される。

  MySQL バージョン 3.23 以降では、+ 演算子のどちらかの側にINTERVAL expr type を使用できる。この場合、もう一方の側の式には日付値または日付時刻値を指定する。 - 演算子を使用する場合は、演算子の右側にのみ INTERVAL expr type を使用できる。これは、日付値または日付時刻値を間隔から差し引いても意味がないため（下の例を参照）。

  date には、開始日とする DATETIME または DATE 値を指定する。expr には、開始日に加える、または開始日から差し引く間隔値を表す式を文字列として指定する。先頭に '-' を付けて負の間隔を表すこともできる。type には、式の解

釈方法を示すキーワードを指定する。

次の表に、type 引数と expr 引数がどのように関連するかを示す。

type 値	前提となる expr の形式
SECOND	SECONDS
MINUTE	MINUTES
HOUR	HOURS
DAY	DAYS
MONTH	MONTHS
YEAR	YEARS
MINUTE_SECOND	'MINUTES:SECONDS'
HOUR_MINUTE	'HOURS:MINUTES'
DAY_HOUR	'DAYS HOURS'
YEAR_MONTH	'YEARS-MONTHS'
HOUR_SECOND	'HOURS:MINUTES:SECONDS'
DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'
DAY_MICROSECOND	'DAYS.MICROSECONDS'
HOUR_MICROSECOND	'HOURS.MICROSECONDS'
MINUTE_MICROSECOND	'MINUTES.MICROSECONDS'
SECOND_MICROSECOND	'SECONDS.MICROSECONDS'
MICROSECOND	'MICROSECONDS'

type 値 DAY_MICROSECOND、HOUR_MICROSECOND、MINUTE_MICROSECOND、SECOND_MICROSECOND、MICROSECOND は、MySQL 4.1.1 以降で使用できる。

MySQL では、expr の形式で任意の句読記号を使用できる。 上の表では、これらの句読記号の一部を示している。date 引数に DATE 型の値を指定し、計算に YEAR、MONTH、DAY 部分のみを組み込む (つまり、時刻部分は含めない) 場合、結果は DATE 型の値になる。それ以外の場合、DATETIME 型の値が返される。

```
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'
mysql> SELECT DATE_ADD('1992-12-31 23:59:59.000002',
  ->              INTERVAL '1.999999' SECOND_MICROSECOND);
    -> '1993-01-01 00:00:01.000001'
```

指定した間隔値が短すぎる場合（つまり、type キーワードによって想定される間隔部分の一部を含んでいない場合）、MySQL では、間隔値の左端の部分がないとみなされる。たとえば、type を DAY_SECOND として指定すると、expr の値は、日、時、分、秒の各部で構成されると想定される。この場合、'1:10' のような値を指定すると、値は日部分と時間部分を持たず、分と秒を表すと解釈される。つまり、'1:10' DAY_SECOND と指定した場合、'1:10' MINUTE_SECOND と指定した場合と同じように解釈されることになる。これは、MySQL で TIME 値が 1 日の時刻ではなく経過時間として解釈されるのに類する。

注意：時刻部分を持つ値を日付値に加えたり、日付値から差し引いたりすると、結果は自動的に日付時刻型の値に変換される。

```
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'
```

誤った日付を使用すると、結果は NULL になる。MONTH、YEAR_MONTH、YEAR のいずれかを加えた結果の日付が、新しい月の最大日数より大きい場合、新しい月の最大日数に調整される。

```
mysql> SELECT DATE_ADD('1998-01-30', interval 1 month);
    -> '1998-02-28'
```

注意: 上記の例で、キーワード INTERVAL と type 指定子はケース非依存。

- DATE_FORMAT(date,format)

  format 文字列に合わせて、date 値を形式設定する。format 文字列では、次の指定子を使用できる。

指定子	説明
%M	月の名前（January..December）。
%W	曜日名（Sunday..Saturday）。
%D	英語のサフィックス付きの日付（0th、1st、2nd、3rd など）。
%Y	4 桁の数値で表した年。

%y	2 桁の数値で表した年。
%X	日曜日を週の最初の日とした場合の週に使用する、4 桁の数値で表した年。%V と組み合わせて使用。
%x	月曜日を週の最初の日とした場合の週に使用する、4 桁の数値で表した年。%v と組み合わせて使用。
%a	略式の曜日名 (Sun..Sat)。
%d	数値で表した日付 (00..31)。
%e	数値で表した日付 (0..31)。
%m	数値で表した月 (00..12)。
%c	数値で表した月 (0..12)。
%b	略式の月名 (Jan..Dec)。
%j	年間を通した日にち (001..366)。
%H	時 (00..23)。
%k	時 (0..23)。
%h	時 (01..12)。
%I	時 (01..12)。
%l	時 (1..12)。
%i	数値で表した分 (00..59)。
%r	12 時間形式の時刻 (hh:mm:ss に続けて AM または PM)。
%T	24 時間形式の時刻 (hh:mm:ss)。
%S	秒 (00..59)。
%s	秒 (00..59)。
%f	マイクロ秒 (000000..999999)。
%p	AM または PM
%w	曜日 (0=Sunday..6=Saturday)。
%U	日曜日を週の最初の日とした場合の週 (00..53)。
%u	月曜日を週の最初の日とした場合の週 (00..53)。
%V	日曜日を週の最初の日とした場合の週 (01..53)。%X と組み合わせて使用。
%v	月曜日を週の最初の日とした場合の週 (01..53)。%x と組み合わせて使用。
%%	リテラルの '%'。

その他の文字はいずれも、解釈されずにそのまま結果にコピーされる。

%f 形式指定子は MySQL 4.1.1 以降で使用できる。

MySQL バージョン 3.23 以降では、形式指定子文字の前に '%' 文字を挿入する必要がある。それより前のバージョンの MySQL では、'%' の使用は任意。

月と日の指定子の範囲がゼロから始まっている理由は、MySQL 3.23 以降では、'2004-00-00' のような不完全な日付の

格納が許容されるため。

```
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'
```

- DAY(date)

  DAY() は DAYOFMONTH() のシノニム。 MySQL 4.1.1 以降で使用できる。

- DAYNAME(date)

  date に指定された日付の曜日名を返す。

```
mysql> SELECT DAYNAME('1998-02-05');
    -> 'Thursday'
```

- DAYOFMONTH(date)

  date に指定された日付に対し、1 ～ 31 の範囲の日にちを返す。

```
mysql> SELECT DAYOFMONTH('1998-02-03');
    -> 3
```

- DAYOFWEEK(date)

  date の曜日インデックス (1 = 日曜、2 = 月曜、... 7 = 土曜) を返す。これらのインデックス値は ODBC 標準に対応している。

```
mysql> SELECT DAYOFWEEK('1998-02-03');
    -> 3
```

- DAYOFYEAR(date)

  date に指定された日付に対し、1 ～ 366 の範囲の年間を通した日にちを返す。

```
mysql> SELECT DAYOFYEAR('1998-02-03');
    -> 34
```

- EXTRACT(type FROM date)

  EXTRACT() 関数では、DATE_ADD() や DATE_SUB() と同じ種類の間隔型指定子が使用される。日付演算を行なわず、日付の一部を抽出する。

```
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
mysql> SELECT EXTRACT(MICROSECOND FROM "2003-01-02 10:30:00.00123");
    -> 123
```

- FROM_DAYS(N)

  指定された日数 N に対して DATE 型の値を返す。

```
mysql> SELECT FROM_DAYS(729669);
    -> '1997-10-07'
```

  FROM_DAYS() では、グレゴリオ暦 (1582 年) より前の値の指定は想定していない。これは、カレンダが変更されたときに失われた日数が考慮されないためである。

- FROM_UNIXTIME(unix_timestamp) , FROM_UNIXTIME(unix_timestamp,format)

  文字列と数値のどちらのコンテキストで使用されているかに応じて、unix_timestamp の値を 'YYYY-MM-DD HH:MM:SS' または YYYYMMDDHHMMSS 形式で返す。

```
mysql> SELECT FROM_UNIXTIME(875996580);
    -> '1997-10-04 22:23:00'
mysql> SELECT FROM_UNIXTIME(875996580) + 0;
    -> 19971004222300
```

format を指定した場合、結果は format 文字列に基づいて形式設定される。format には、DATE_FORMAT() 関数の入力値と同じ指定子を組み込める。

```
mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(),
    ->                 '%Y %D %M %h:%i:%s %x');
    -> '2003 6th August 06:22:58 2003'
```

- GET_FORMAT(DATE | TIME | TIMESTAMP, 'EUR' | 'USA' | 'JIS' | 'ISO' | 'INTERNAL')

  フォーマットの文字列を返す。この関数は、DATE_FORMAT() 関数や STR_TO_DATE() 関数と組み合わせて使用したり、サーバ変数 DATE_FORMAT、TIME_FORMAT、DATETIME_FORMAT を設定したりするときに役立つ。最初の引数には 3 つの入力可能値があり、2 番目の引数には 5 つの入力可能値があるため、結果として返されるフォーマット文字列は全部で 15 通りある（使用される指定子については、DATE_FORMAT() 関数の説明に含まれている表を参照）。

関数呼び出し	結果
GET_FORMAT(DATE,'USA')	'%m.%d.%Y'
GET_FORMAT(DATE,'JIS')	'%Y-%m-%d'
GET_FORMAT(DATE,'ISO')	'%Y-%m-%d'
GET_FORMAT(DATE,'EUR')	'%d.%m.%Y'
GET_FORMAT(DATE,'INTERNAL')	'%Y%m%d'
GET_FORMAT(TIMESTAMP,'USA')	'%Y-%m-%d-%H.%i.%s'
GET_FORMAT(TIMESTAMP,'JIS')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(TIMESTAMP,'ISO')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(TIMESTAMP,'EUR')	'%Y-%m-%d-%H.%i.%s'
GET_FORMAT(TIMESTAMP,'INTERNAL')	'%Y%m%d%H%i%s'
GET_FORMAT(TIME,'USA')	'%h:%i:%s %p'
GET_FORMAT(TIME,'JIS')	'%H:%i:%s'
GET_FORMAT(TIME,'ISO')	'%H:%i:%s'
GET_FORMAT(TIME,'EUR')	'%H.%i.%S'
GET_FORMAT(TIME,'INTERNAL')	'%H%i%s'

ISO 形式は ISO 8601 ではなく、ISO 9075。

```
mysql> SELECT DATE_FORMAT('2003-10-03', GET_FORMAT(DATE, 'EUR')
    -> '03.10.2003'
mysql> SELECT STR_TO_DATE('10.31.2003', GET_FORMAT(DATE, 'USA'))
    -> 2003-10-31
```

```
mysql> SET DATE_FORMAT=GET_FORMAT(DATE, 'USA'); SELECT '2003-10-31';
    -> 10-31-2003
```

GET_FORMAT() は MySQL 4.1.1 以降で使用できる。 See 項5.5.6.「SET 構文」 を参照。

- HOUR(time)

  time に指定された時刻の時間部分の値を返す。戻り値の範囲は、1 日の時間を示す値の場合は 0 から 23。

```
mysql> SELECT HOUR('10:05:03');
    -> 10
```

しかし、TIME 値の範囲は、実際にはこれよりずっと広いため、23 より大きい値が HOUR として返ることがある。

```
mysql> SELECT HOUR('272:59:59');
    -> 272
```

- LAST_DAY(date)

  指定された日付または日付時刻の値に対し、その値が含まれる月の最後の日にちを返す。引数が無効な場合は、NULL を返す。

```
mysql> SELECT LAST_DAY('2003-02-05'), LAST_DAY('2004-02-05');
    -> '2003-02-28', '2004-02-29'
mysql> SELECT LAST_DAY('2004-01-01 01:01:01');
    -> '2004-01-31'
mysql> SELECT LAST_DAY('2003-03-32');
    -> NULL
```

LAST_DAY() は MySQL 4.1.1 以降で使用できる。

- LOCALTIME , LOCALTIME()

  LOCALTIME と LOCALTIME() は NOW() のシノニム。

- LOCALTIMESTAMP , LOCALTIMESTAMP()

  LOCALTIMESTAMP と LOCALTIMESTAMP() は NOW() のシノニム。

- MAKEDATE(year,dayofyear)

  指定された年の値と年間を通した日にちの値に対応する日付を返す。 dayofyear には、0 より大きい値を指定する必要がある。0 以下の値を指定すると、結果として NULL が返される。

```
mysql> SELECT MAKEDATE(2001,31), MAKEDATE(2001,32);
    -> '2001-01-31', '2001-02-01'
mysql> SELECT MAKEDATE(2001,365), MAKEDATE(2004,365);
    -> '2001-12-31', '2004-12-30'
mysql> SELECT MAKEDATE(2001,0);
    -> NULL
```

MAKEDATE() は MySQL 4.1.1 以降で使用できる。

- MAKETIME(hour,minute,second)

  引数 hour、minute、second に指定された値から計算した時刻値を返す。

```
mysql> SELECT MAKETIME(12,15,30);
    -> '12:15:30'
```

MAKETIME() は MySQL 4.1.1 以降で使用できる。

- MICROSECOND(expr)

  時刻式または日付時刻式 expr からマイクロ秒を、0 ～ 999999 の範囲内の数値として返す。

```
mysql> SELECT MICROSECOND('12:00:00.123456');
    -> 123456
mysql> SELECT MICROSECOND('1997-12-31 23:59:59.000010');
    -> 10
```

MICROSECOND() は MySQL 4.1.1 以降で使用できる。

- MINUTE(time)

  time に指定された時刻の分を、0 ～ 59 の範囲の値として返す。

```
mysql> SELECT MINUTE('98-02-03 10:05:03');
    -> 5
```

- MONTH(date)

  date に指定された日付の月を、1 ～ 12 の範囲の値として返す。

```
mysql> SELECT MONTH('1998-02-03');
    -> 2
```

- MONTHNAME(date)

  date に指定された月の名前を返す。

```
mysql> SELECT MONTHNAME('1998-02-05');
    -> 'February'
```

- NOW()

  文字列と数値のどちらのコンテキストで使用されているかに応じて、'YYYY-MM-DD HH:MM:SS' または YYYYMMDDHHMMSS 形式の値として、現在の日付と時刻を返す。

```
mysql> SELECT NOW();
    -> '1997-12-15 23:50:26'
mysql> SELECT NOW() + 0;
    -> 19971215235026
```

- PERIOD_ADD(P,N)

  N に指定された月数を期間 P （ YYMM または YYYYMM の形式 ）に加える。戻り値は YYYYMM の形式で返る。

  注意: 期間引数 P には、日付値を指定しないようにする。

```
mysql> SELECT PERIOD_ADD(9801,2);
    -> 199803
```

- PERIOD_DIFF(P1,P2)

  期間 P1 と P2 の間の月数を返す。 P1 と P2 の指定は YYMM または YYYYMM 形式で行う。

  注意: 期間引数 P1 と P2 には、日付値を指定しないようにする。

```
mysql> SELECT PERIOD_DIFF(9802,199703);
    -> 11
```

- QUARTER(date)

date に指定された日付に対応する四半期を、1 ～ 4 の範囲の値として返す。

```
mysql> SELECT QUARTER('98-04-01');
    -> 2
```

- SECOND(time)

  time に指定された時刻の秒を、0 ～ 59 の範囲の値として返す。

```
mysql> SELECT SECOND('10:05:03');
    -> 3
```

- SEC_TO_TIME(seconds)

  seconds 引数に指定された秒数を時、分、秒の値に変換し、文字列と数値のどちらのコンテキストで使用されているかに応じて、'HH:MM:SS' または HHMMSS 形式の値として返す。

```
mysql> SELECT SEC_TO_TIME(2378);
    -> '00:39:38'
mysql> SELECT SEC_TO_TIME(2378) + 0;
    -> 3938
```

- STR_TO_DATE(str,format)

  DATE_FORMAT() 関数と逆の働きをする。文字列 str と形式文字列 format を取り、DATETIME 型の値を返す。

  str に指定した日付、時刻、または日付時刻の値が、format に指定した形式で返される。format に使用できる指定子については、DATE_FORMAT() 関数の説明にある表を参照。他の文字はいずれも、解釈されずに文字どおり受け取られる。 str に誤った日付、時刻、または日付時刻が含まれている場合、STR_TO_DATE() は NULL を返す。

```
mysql> SELECT STR_TO_DATE('03.10.2003 09.20', '%d.%m.%Y %H.%i')
```

```
    -> 2003-10-03 09:20:00
mysql> SELECT STR_TO_DATE('10rap', '%crap')
    -> 0000-10-00 00:00:00
mysql> SELECT STR_TO_DATE('2003-15-10 00:00:00', '%Y-%m-%d %H:%i:%s')
    -> NULL
```

STR_TO_DATE() は MySQL 4.1.1 以降で使用できる。

- SUBDATE(date,INTERVAL expr type) , SUBDATE(expr,days)

  2 番目の引数の INTERVAL を指定する形式で呼び出した場合、SUBDATE() は DATE_SUB() とシノニムになる。

```
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
    -> '1997-12-02'
mysql> SELECT SUBDATE('1998-01-02', INTERVAL 31 DAY);
    -> '1997-12-02'
```

  2 番目の構文は MySQL 4.1.1 以降で使用できる。expr には日付式か日付時刻式を指定し、days には expr から差し引く日数を指定する。

```
mysql> SELECT SUBDATE('1998-01-02 12:00:00', 31);
    -> '1997-12-02 12:00:00'
```

- SUBTIME(expr,expr2)

  SUBTIME() では、expr から expr2 を差し引いた結果が返される。 expr には日付式か日付時刻式を指定し、expr2 には時刻式を指定する。

```
mysql> SELECT SUBTIME("1997-12-31 23:59:59.999999", "1 1:1:1.000002");
    -> '1997-12-30 22:58:58.999997'
mysql> SELECT SUBTIME("01:00:00.999999", "02:00:00.999998");
    -> '-00:59:59.999999'
```

  SUBTIME() は MySQL 4.1.1 で追加された。

- SYSDATE()

  SYSDATE() は NOW() のシノニム。

- TIME(expr)

  時刻式または日付時刻式 expr の時刻部分を取り出す。

```
mysql> SELECT TIME('2003-12-31 01:02:03');
    -> '01:02:03'
```

```
mysql> SELECT TIME('2003-12-31 01:02:03.000123');
    -> '01:02:03.000123'
```

TIME() は MySQL 4.1.1 以降で使用できる。

- TIMEDIFF(expr,expr2)

  TIMEDIFF() では、開始時刻 expr と終了時刻 expr2 の間の時間が返される。 expr と expr2 には、時刻式か日付時刻式を指定できるが、どちらも同じ型にしなければならない。

```
mysql> SELECT TIMEDIFF('2000:01:01 00:00:00', '2000:01:01 00:00:00.000001');
    -> '-00:00:00.000001'
mysql> SELECT TIMEDIFF('1997-12-31 23:59:59.000001','1997-12-30 01:01:01.000002');
    -> '46:58:57.999999'
```

TIMEDIFF() は MySQL 4.1.1 で追加された。

- TIMESTAMP(expr) , TIMESTAMP(expr,expr2)

  引数が 1 つの場合は、日付式または日付時刻式 expr を日付時刻値として返す。 引数が 2 つの場合は、日付式または日付時刻式 expr に時刻式 expr2 を加えた結果を日付時刻値として返す。

```
mysql> SELECT TIMESTAMP('2003-12-31');
    -> '2003-12-31 00:00:00'
mysql> SELECT TIMESTAMP('2003-12-31 12:00:00','12:00:00');
    -> '2004-01-01 00:00:00'
```

TIMESTAMP() は MySQL 4.1.1 以降で使用できる。

- TIME_FORMAT(time,format)

  この関数の使用法は DATE_FORMAT() 関数と同様だが、この関数の場合、format に指定できる文字列は、時、分、秒を処理する形式指定子に限られる。その他の指定子を指定した場合は、NULL 値または 0 が返される。

  time 値の時間部分が 23 より大きい場合、時間形式指定子 %H と %k では、通常の範囲 0..23 より大きい値が生成される。その他の時間形式指定子では、時間値のモジュロ 12 が生成される。

```
mysql> SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');
    -> '100 100 04 04 4'
```

- TIME_TO_SEC(time)

  time 引数に指定された時刻を秒数に変換して返す。

```
mysql> SELECT TIME_TO_SEC('22:23:00');
    -> 80580
mysql> SELECT TIME_TO_SEC('00:39:38');
    -> 2378
```

- TO_DAYS(date)

  date に指定された日付を日数 (年 0 からの通し日数) に変換して返す。

```
mysql> SELECT TO_DAYS(950501);
    -> 728779
mysql> SELECT TO_DAYS('1997-10-07');
    -> 729669
```

  TO_DAYS() では、グレゴリオ暦 (1582 年) より前の値の指定は想定していない。これは、カレンダが変更されたときに失われた日数が考慮されないため。

- UNIX_TIMESTAMP() , UNIX_TIMESTAMP(date)

  引数なしで呼び出された場合、Unix タイムスタンプ ('1970-01-01 00:00:00' GMT からの秒数) を符号なしの整数として返す。date 引数を指定して呼び出された場合、UNIX_TIMESTAMP() は引数に指定された日付を '1970-01-01 00:00:00' GMT からの秒数として返す。引数 date には、DATE 型文字列、DATETIME 型文字列、TIMESTAMP 型の値、YYMMDD または YYYYMMDD の形式の現地時間のいずれかを指定できる。

```
mysql> SELECT UNIX_TIMESTAMP();
    -> 882226357
mysql> SELECT UNIX_TIMESTAMP('1997-10-04 22:23:00');
    -> 875996580
```

  UNIX_TIMESTAMP を TIMESTAMP 型のカラムに使用すると。``文字列から Unix タイムスタンプへの" 暗黙的な変換を行わずに、内部のタイムスタンプ値が直接返される。 UNIX_TIMESTAMP() に範囲外の日付を渡すと、0 が返されるが、この場合、実行されるチェックは基本チェック (年 1970 ー 2037、月 01 ー 12、日 01 ー 31) に限られる。

  UNIX_TIMESTAMP() カラム値を差し引く場合、結果を符号付き整数にキャストする必要がある場合がある。 See 項 6.3.5.「キャスト関数」。

- UTC_DATE , UTC_DATE()

  文字列と数値のどちらのコンテキストで使用されているかに応じて、'YYYY-MM-DD' または YYYYMMDD形式の値として、現在の UTC 日付を返す。

```
mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
    -> '2003-08-14', 20030814
```

UTC_DATE() は MySQL 4.1.1 以降で使用できる。

- UTC_TIME , UTC_TIME()

  文字列と数値のどちらのコンテキストで使用されているかに応じて、'HH:MM:SS' または HHMMSS 形式の値として、現在の UTC 時刻を返す。

```
mysql> SELECT UTC_TIME(), UTC_TIME() + 0;
    -> '18:07:53', 180753
```

UTC_TIME() は MySQL 4.1.1 以降で使用できる。

- UTC_TIMESTAMP , UTC_TIMESTAMP()

  文字列と数値のどちらのコンテキストで使用されているかに応じて、'YYYY-MM-DD HH:MM:SS' または YYYYMMDDHHMMSS 形式の値として、現在の UTC の日付と時刻を返す。

```
mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
    -> '2003-08-14 18:08:04', 20030814180804
```

UTC_TIMESTAMP() は MySQL 4.1.1 以降で使用できる。

- WEEK(date [,mode])

  date に指定された日付に対応する週数を返す。 引数を 2 つ取る形式の WEEK() では、週を日曜と月曜のどちらから開始するかと、戻り値の範囲を 0 ～ 53 と 1 ～ 52 のどちらにするかを指定できる。mode 引数を省略した場合は、default_week_format サーバ変数の値 (または MySQL 4.0 以前のバージョンでは 0)が適用される。 See 項5.5.6.「SET 構文」。

  次の表に、mode 引数の動作を示す。

値	意味
0	日曜から週を開始。戻り値の範囲は 0 ～ 53。週 1 はその年に開始される最初の週。
1	月曜から週を開始。戻り値の範囲は 0 ～ 53。週 1 は 4 日以上の日を持つその年の最初の週。
2	日曜から週を開始。戻り値の範囲は 1 ～ 53。週 1 はその年に開始される最初の週。
3	月曜から週を開始。戻り値の範囲は 1 ～ 53。週 1 は 4 日以上の日を持つその年の最初の週。
4	日曜から週を開始。戻り値の範囲は 0 ～ 53。週 1 は 4 日以上の日を持つその年の最初の週。
5	月曜から週を開始。戻り値の範囲は 0 ～ 53。週 1 はその年に開始される最初の週。

6	日曜から週を開始。戻り値の範囲は 1 〜 53。週 1 は 4 日以上の日を持つその年の最初の週。
7	月曜から週を開始。戻り値の範囲は 1 〜 53。週 1 はその年に開始される最初の週。

mode 値 3 は MySQL 4.0.5 以降で使用できる。 mode 値 4 以上は MySQL 4.0.17 以降で使用できる。

```
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
```

注意:バージョン 4.0 で、WEEK(date,0) は米国のカレンダに合わせて変更された。それ以前のバージョンでは、WEEK() の計算は米国の日付に対して正しく行われなかった (事実上、すべての場合で、WEEK(date) と WEEK(date,0) で結果が誤っていた) 。

注意: 指定された日付が前年の最後の週にあたる場合、オプションの mode 引数として 2、3、6、7 のいずれかが指定されていないと、MySQL から値 0 が返る。

```
mysql> SELECT YEAR('2000-01-01'), WEEK('2000-01-01',0);
    -> 2000, 0
```

この場合、指定された日付は実際には 1999 年の 52 週目に含まれるため、WEEK() 関数の結果として値 52 を返すべきだとする考え方もあるが、MySQL ではそのようにはせず、値 0 を返すようにした。これは、この関数によって、あくまでも ``指定された年の週数'' が返されるようにするためである。そうすることで、日付から日付部分を取り出す他の関数と組み合わせて WEEK() 関数を使用した場合の信頼性が確保される。

結果の評価を、指定した日付の週の最初の日が含まれる年に関連して行うには、オプションの mode 引数として 2、3、6、7 のいずれかを指定するようにする。

```
mysql> SELECT WEEK('2000-01-01',2);
    -> 52
```

または、YEARWEEK() 関数を使用する。

```
mysql> SELECT YEARWEEK('2000-01-01');
    -> 199952
mysql> SELECT MID(YEARWEEK('2000-01-01'),5,2);
    -> '52'
```

- WEEKDAY(date)

  date に指定された日付の曜日インデックス (0 = 月曜、1 = 火曜、... 6 = 日曜) を返す。

```
mysql> SELECT WEEKDAY('1998-02-03 22:23:00');
    -> 1
mysql> SELECT WEEKDAY('1997-11-05');
    -> 2
```

- WEEKOFYEAR(date)

  指定された日付のカレンダ上の週を、1 ～ 53 の数値として返す。

```
mysql> SELECT WEEKOFYEAR('1998-02-20');
    -> 8
```

  WEEKOFYEAR() は MySQL 4.1.1 以降で使用できる。

- YEAR(date)

  date に指定された日付の年を、1000 ～ 9999 の数値として返す。

```
mysql> SELECT YEAR('98-02-03');
    -> 1998
```

- YEARWEEK(date) , YEARWEEK(date,start)

  指定された日付の年と週を返す。start 引数は、WEEK() に対する start 引数とまったく同じように作用する。注意: date 引数に指定された日付の週がその年の最初の週である場合と最後の週である場合、結果の年が date 引数の年と異なることがある。

```
mysql> SELECT YEARWEEK('1987-01-01');
    -> 198653
```

  結果の週数は、オプション引数 0 または 1 を指定した WEEK() 関数で返る値と異なる。WEEK() 関数では、指定した年のコンテキストで週数が返される。

6.3.5. キャスト関数

CAST() 関数と CONVERT() 関数は、あるデータ型の値から別のデータ型の値を生成する目的で使用することができます。構文は以下のとおりです。

```
CAST(expression AS type)
CONVERT(expression,type)
CONVERT(expr USING transcoding_name)
```

type には、次のいずれかのデータ型を指定できます。

- BINARY
- CHAR
- DATE
- DATETIME
- SIGNED {INTEGER}
- TIME
- UNSIGNED {INTEGER}

CAST() と CONVERT() は MySQL 4.0.2 以降で使用することができます。 変換型 CHAR は 4.0.6 以降で使用可能です。 USING を指定する CONVERT() は、4.1.0 以降で使用可能です。

CAST() と CONVERT(... USING ...) は SQL-99 の構文です。 USING を指定しない CONVERT() は ODBC の構文です。

キャスト関数は、CREATE ... SELECT ステートメントで特定の型のカラムを作成する必要があるときに役立ちます。

```
CREATE TABLE new_table SELECT CAST('2000-01-01' AS DATE);
```

また、キャスト関数は ENUM 型のカラムを語彙の順序でソートするときにも役立ちます。通常、ENUM 型のカラムのソートは、内部の数値に基づいて行われます。値を CHAR 型にキャストすると、語彙の順序でソートされるようになります。

```
SELECT enum_col FROM tbl_name ORDER BY CAST(enum_col AS CHAR);
```

CAST(string AS BINARY) は BINARY string と同じです。 CAST(expr AS CHAR) では、指定した式がデフォルトのキャラクタセットの文字列として扱われます。

注意: MysQL 4.0 では、DATE、DATETIME、TIME 型のカラムに CAST() 操作を行った場合、カラムが特定の型としてマークされるだけで、カラム値が変わるわけではありません。

MySQL 4.1.0 では、カラム値は、ユーザに送られるときに適切なカラム型に変換されます (これは、クライアントへの日付情報の送信における、4.1 の新しいプロトコルの機能です) 。

```
mysql> SELECT CAST(NOW() AS DATE);
    -> 2003-05-26
```

後続の MySQL バージョン (おそらく 4.1.2 か 5.0) では、CONCAT("Date: ",CAST(NOW() AS DATE)) のような、より複雑な式の一部として CAST を使用した場合にも、結果のデータ型が適切に変換されるよう修正する予定です。

データを別の形式で取り出すときには、CAST() は使用せず、代わりに、LEFT や EXTRACT() のような文字列関数を使用してください。 See 項6.3.4.「日付と時刻関数」。

文字列を数値にキャストするときには、通常は、特に何もしないで、そのままその文字列を数値として使用します。

```
mysql> SELECT 1+'1';
    -> 2
```

文字列のコンテキストで数値を使用すると、数値は BINARY 文字列に自動で変換されます。

```
mysql> SELECT CONCAT("hello you ",2);
    ->  "hello you 2"
```

MySQL では、符号付きと符号なしのどちらでも、64 ビット値の演算をサポートしています。 数値演算 (+ など) でどちらか一方のオペランドが unsigned integer の場合、結果の値は符号なしになります。 ただし、キャスト演算子 SIGNED と UNSIGNED を使用することによって、それぞれ符号付き、または符号なしの 64 ビット整数にキャストできます。

```
mysql> SELECT CAST(1-2 AS UNSIGNED)
    -> 18446744073709551615
mysql> SELECT CAST(CAST(1-2 AS UNSIGNED) AS SIGNED);
    -> -1
```

注意: どちらか一方のオペランドが浮動小数点数の場合、結果の値は浮動小数点数になり、上記の規則は適用されません (このコンテキストでは、DECIMAL 型の値は浮動小数点数とみなされます) 。

```
mysql> SELECT CAST(1 AS UNSIGNED) - 2.0;
    -> -1.0
```

算術演算に文字列を使用すると、その文字列は浮動小数点数に変換されます。

符号なしの値の扱いについては、BIGINT 型の値に適切に対応するよう、MySQL 4.0 で変更されました。MySQL 4.0 と 3.23 の両方で実行するコードがあるとき (通常、この場合、CAST() 関数は使用できない) 、次の方法によって、2 つの符号なし整数カラムの減算の実行時に符号付きの結果値を得ることができます。

```
SELECT (unsigned_column_1+0.0)-(unsigned_column_2+0.0);
```

この方法では、減算の実行前に両方のカラムを浮動小数点型に変換しています。

以前の MySQL アプリケーションの UNSIGNED 型のカラムを MySQL 4.0 に移植した場合に問題があるときは、mysqld の起動時に --sql-mode=NO_UNSIGNED_SUBTRACTION オプションを指定します。注意: このオプションを指定した場合、BIGINT UNSIGNED カラム型を効率的に使用することはできません。

USING を指定する CONVERT() は、データのキャラクタセットを別のキャラクタセットに変換するときに使用します。MySQL では、トランスコーディング名は対応するキャラクタセット名と同じです。たとえば、次のステートメントでは、サーバのデフォルトのキャラクタセットに基づく文字列 'abc' が、utf8 キャラクタセットの対応する文字列に変換されます。

```
SELECT CONVERT('abc' USING utf8);
```

6.3.6. その他の関数

6.3.6.1. ビット関数

MySQL では、ビット演算に BIGINT (64 ビット) 演算を使用します。したがって、以下のビット演算子の範囲は最大 64 ビットになります。

- |

  ビットごとの OR。

```
mysql> SELECT 29 | 15;
    -> 31
```

  結果は符合なしの 64 ビット整数。

- &

  ビットごとの AND。

```
mysql> SELECT 29 & 15;
    -> 13
```

  結果は符合なしの 64 ビット整数。

- ^

  ビットごとの XOR。

```
mysql> SELECT 1 ^ 1;
    -> 0
mysql> SELECT 1 ^ 0;
    -> 1
mysql> SELECT 11 ^ 3;
    -> 8
```

  結果は符合なしの 64 ビット整数。

  XOR はバージョン 4.0.2 で追加された。

- <<

  longlong (BIGINT) 値のビットを左にシフトする。

```
mysql> SELECT 1 << 2;
    -> 4
```

  結果は符合なしの 64 ビット整数。

- >>

longlong (BIGINT) 値のビットを右にシフトする。

```
mysql> SELECT 4 >> 2;
    -> 1
```

結果は符合なしの 64 ビット整数。

- ~

  すべてのビットを反転させる。

```
mysql> SELECT 5 & ~1;
    -> 4
```

結果は符合なしの 64 ビット整数。

- BIT_COUNT(N)

  引数 N に指定された数値を 2 進表記したときに 1 に設定されるビット数を返す。

```
mysql> SELECT BIT_COUNT(29);
    -> 4
```

6.3.6.2. その他の各種関数

- AES_ENCRYPT(string,key_string) , AES_DECRYPT(string,key_string)

  これらの関数では、公式の AES (Advanced Encryption Standard) アルゴリズム (以前、Rijndael と呼ばれていたもの) を使用してデータの暗号化と解読を実行できる。 エンコードに使用されるキー長は 128 ビットだが、ソースを変更することによって、256 ビットまで拡張できる。MySQL では 128 ビットを選択している。その理由は、そのほうが迅速で、かつ通常、安全性も十分に確保できるため。

  入力引数は任意の長さにすることができる。どちらかの引数として NULL を指定した場合、この関数の結果も NULL になる。

  AES はブロックレベルのアルゴリズムであるため、不揃いの長さの文字列のエンコード時には埋め込みが行われる。したがって、結果の文字列の長さは 16*(trunc(string_length/16)+1) として計算することができる。

  AES_DECRYPT() では、無効なデータや誤った埋め込みが検出されると、NULL が返される。しかし、入力データやキーが無効でも、AES_DECRYPT() から NULL 以外の値 (ガベージなど) が返されることがある。

  クエリを次のように変更することによって、AES 関数を使用して暗号形式でデータを格納できる。

```
INSERT INTO t VALUES (1,AES_ENCRYPT('text','password'));
```

各クエリの接続を通したキーの転送を行わないようにすれば、さらにセキュリティを強化することができる。この場合、キーは接続時にサーバ側の変数に格納する。

```
SELECT @password:='my password';
INSERT INTO t VALUES (1,AES_ENCRYPT('text',@password));
```

AES_ENCRYPT() と AES_DECRYPT() はバージョン 4.0.2 で追加された。この 2 つは、現在のところ MySQL で使用可能な、暗号化方式として最も安全性の高い暗号化関数としてみなすことができる。

- BENCHMARK(count,expr)

  BENCHMARK() 関数は、expr に指定された式を count に指定された回数だけ繰り返し実行する。この関数は、MySQL での式の処理速度を計測するために使用できる。結果の値としては、常に 0 が返る。この関数を mysql クライアントで使用することによって、クエリの実行時間をレポートすることができる。

```
mysql> SELECT BENCHMARK(1000000,ENCODE("hello","goodbye"));
+----------------------------------------------+
| BENCHMARK(1000000,ENCODE("hello","goodbye")) |
+----------------------------------------------+
|                                            0 |
+----------------------------------------------+
1 row in set (4.74 sec)
```

  レポートされる時間は、サーバ側の CPU 時間ではなく、クライアント側の経過時間である。BENCHMARK() を数回実行し、結果を分析することによって、サーバマシンの負荷がどれくらいか調べることができる。

- COMPRESS(string_to_compress)

  文字列を圧縮する。

```
mysql> SELECT LENGTH(COMPRESS(REPEAT("a",1000)));
    -> 21
mysql> SELECT LENGTH(COMPRESS(""));
    -> 0
mysql> SELECT LENGTH(COMPRESS("a"));
    -> 13
mysql> SELECT LENGTH(COMPRESS(REPEAT("a",16)));
    -> 15
```

  COMPRESS() は MySQL バージョン 4.1.1 で追加された。 この関数を使用するためには、zlib などの圧縮ライブラリを使用して MySQL をコンパイルしておく必要がある。コンパイルを行っていないと、戻り値は常に NULL になる。

  圧縮した文字列の内容は次の方法で格納される。

- 空の文字列は空の文字列として格納される。
- 空以外の文字列は、まず 4 バイトの長さの非圧縮文字列として格納され (下位バイトが先) 、その後 gzip によって文字列が圧縮される。文字列がスペースで終わっている場合は、追加の '.' が挿入される。これは、結果を CHAR 型または VARCHAR 型のカラムに格納する場合に末尾のスペースの切り取りで問題が発生しないようにするため。しかし、CHAR 型や VARCHAR 型カラムへの圧縮文字列の格納は推奨されない。代わりに、BLOB 型のカラムを使用するようにする。

- CONNECTION_ID()

  接続の接続 ID (スレッド ID) を返す。 各接続にはその接続固有の一意な ID が割り当てられる。

```
mysql> SELECT CONNECTION_ID();
    -> 23786
```

- CURRENT_USER()

  現在のセッションの認証に使用されたユーザ名とホスト名を返す。この値はアクセス権限の評価に使用されるアカウントに対応している。USER() の値とは異なる場合がある。

```
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'
```

  上の例では、クライアントがユーザ名として davida を指定している (USER() の値により) にもかかわらず、サーバは匿名ユーザアカウントでクライアントの認証を行っている (CURRENT_USER() 値のユーザ名の部分が空) のがわかる。このようなことが起こる原因の 1 つとして、davida の権限テーブルにアカウントが登録されていない場合が考えられる。

- DATABASE()

  カレントデータベース名を返す。

```
mysql> SELECT DATABASE();
    -> 'test'
```

  カレントデータベースがない場合、DATABASE() は、MySQL 4.1.1 以降では NULL を返し、それ以前のバージョンでは空の文字列を返す。

- DECODE(crypt_str,pass_str)

  pass_str に指定された文字列をパスワードとして使用して、暗号化された文字列 crypt_str を解読する。crypt_str は ENCODE() から返された文字列でなければならない。

- ENCODE(str,pass_str)

  pass_str に指定された文字列をパスワードとして使用して、str を暗号化する。 暗号化された結果を解読するには、DECODE() を使用する。

  結果として、string と同じ長さのバイナリ文字列が返される。 この文字列をカラムに保存するには、BLOB 型のカラムを使用する。

- DES_DECRYPT(string_to_decrypt [, key_string])

  DES_ENCRYPT() で暗号化された文字列を解読する。

  注意: この関数は、MySQL で SSL のサポートが組み込まれている場合にのみ機能する。 See 項4.4.10.「安全な接続の使用」。

  key_string 引数が指定されていない場合、DES_DECRYPT() は暗号化文字列の最初のバイトを調べて、元の文字列の暗号化に使用された DES キー番号を判別し、des-key-file からキーを読み取ってメッセージの暗号化を解除する。これを行うためには、ユーザは SUPER 権限を持っていなければならない。

  この関数に key_string 引数を渡した場合、この引数に指定した文字列がメッセージの暗号化を解除するキーとして使用される。

  string_to_decrypt に指定した文字列が暗号化された文字列として MySQL に認識されない場合、指定した文字列がそのまま返される。

  エラー時には、NULL が返される。

- DES_ENCRYPT(string_to_encrypt [, (key_number | key_string)])

  指定されたキーを使用して、Triple-DES アルゴリズムによって文字列の暗号化を解除する。

  注意: この関数は、MySQL で SSL のサポートが組み込まれている場合にのみ機能する。 See 項4.4.10.「安全な接続の使用」。

  使用する暗号化キーは次の方法で選択される。

引数	説明
引数が 1 つだけ	des-key-file ファイル内の最初のキーを使用。
キー番号	des-key-file ファイル内の、指定されたキー (0 〜 9) を使用。
文字列	key_string に指定された文字列を使用して string_to_encrypt の暗号化を解除。

  関数の結果として、バイナリ文字列が返される。この文字列の最初の文字は CHAR(128 | key_number) になる。

128 は暗号化されたキーを見分けやすくする目的で追加される。 文字列キーを使用すると、key_number は 127 になる。

エラー時には、NULL が返される。

結果の文字列長は new_length= org_length + (8-(org_length % 8))+1 になる。

des-key-file の形式は次のとおり。

```
key_number des_key_string
key_number des_key_string
```

各 key_number は 0 〜 9 の範囲の番号でなければならない。ファイル内の行は任意の順序にすることができる。des_key_string は、メッセージの暗号化に使用する文字列。番号とキーの間には、1 つ以上のスペースを挿入する。最初のキーは、DES_ENCRYPT() にキー引数を何も指定しない場合に使用されるデフォルトのキー。

FLUSH DES_KEY_FILE コマンドを使用すると、MySQL にキーファイルから新しいキー値を読み取らせることができる。このコマンドを使用するには、Reload_priv 権限が必要になる。

デフォルトキーのセットを用意しておく利点の 1 つは、暗号化されたカラム値が存在するかどうかアプリケーションでチェックできることである。この場合、それらの値を解読する権利をエンドユーザに与える必要はない。

```
mysql> SELECT customer_address FROM customer_table WHERE
    crypted_credit_card = DES_ENCRYPT("credit_card_number");
```

- ENCRYPT(str[,salt])

  Unix の crypt() システムコールを使用して、str に指定された文字列を暗号化する。salt 引数は、2 つの文字から成る文字列でなければならない（MySQL バージョン 3.22.16 以降では、salt に 2 文字を超える長さの文字列を指定できる）。

```
mysql> SELECT ENCRYPT("hello");
    -> 'VxuFAJXVARROc'
```

  一部のシステムでは、ENCRYPT() は str に指定された文字列の最初の 8 文字以外のすべての文字を無視する。この動作は、基盤となる crypt() システムコールの実装によって決まる。

  使用システムで crypt() を利用できない場合、ENCRYPT() は常に NULL を返す。そのため、ENCRYPT() の代用として、MD5() または SHA1() の使用が推奨される。これらの 2 つの関数はすべてのプラットフォームで利用できる。

- FORMAT(X,D)

  X に指定された数値の小数部を、D に指定された桁数に丸めて、'#,###,###.##' のような形式に設定し、結果を文字列として返す。 D が 0 の場合、結果の値は小数点も小数部も持たない。

```
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'
```

- FOUND_ROWS()

  SELECT ステートメントに LIMIT 節を組み込むことによって、サーバがクライアントに返すレコード数を制限できる。 状況によっては、LIMIT を指定しなかった場合にいくつのレコードが返されるかを、ステートメントを再度実行することなく確認したいことがある。 このレコード数を確認するには SELECT ステートメントに SQL_CALC_FOUND_ROWS オプションを指定し、その後 FOUND_ROWS() を呼び出す。

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name
    WHERE id > 100 LIMIT 10;
mysql> SELECT FOUND_ROWS();
```

  2 番目の SELECT では、最初の SELECT を LIMIT 節なしで実行した場合に返されるレコード数を示す数が返る（先行する SELECT ステートメントに SQL_CALC_FOUND_ROWS オプションが含まれていないと、FOUND_ROWS() は LIMIT が使用されない場合とは異なる、LIMIT が使用されたときの結果を返す）。

  注意: SELECT SQL_CALC_FOUND_ROWS ... を使用した場合、MySQL では完全な結果セットに含まれるレコード数を計算する必要が生じる。しかし、この場合、結果セットをクライアントに送る必要がないため、LIMIT なしで再度クエリを実行するより時間がかからない。

  SQL_CALC_FOUND_ROWS と FOUND_ROWS() は、クエリで返されるレコード数を制限する必要がある場合に、完全な結果セットに含まれるレコード数を（クエリを再実行することなく）確認したいときに役立つ。例として、検索結果の別のセクションを示すページへのリンクを含むページ画面を表示する Web スクリプトを挙げることができる。FOUND_ROWS() を使用すると、結果の残りの部分を表示するのにあと何ページ必要か確認できる。

  SQL_CALC_FOUND_ROWS と FOUND_ROWS() を UNION クエリで使用した場合、単純な SELECT ステートメントで使用した場合よりも複雑になる。これは、LIMIT が UNION 内の複数の場所で起こり得るためである。LIMIT は、UNION 内の個々の SELECT ステートメントに適用することも、UNION の結果全体にグローバルに適用することもできる。

  UNION に対して SQL_CALC_FOUND_ROWS を使用する目的は、このオプションによって、グローバルな LIMIT を指定しなかった場合に返されるレコード数を確認できることにある。 UNION での SQL_CALC_FOUND_ROWS の使用には、以下の条件が適用される。

  - UNION の最初の SELECT に SQL_CALC_FOUND_ROWS キーワードが指定されていなければならない。
  - FOUND_ROWS() の値は UNION ALL の使用時のみ正確になる。 ALL なしで UNION を使用した場合、重複の削除が行われるため、FOUND_ROWS() の値は近似値にすぎない。

  - UNION 内に LIMIT が存在しない場合、SQL_CALC_FOUND_ROWS は無視され、UNION を処理するために作成されたテンポラリテーブル内のレコード数が返される。

  SQL_CALC_FOUND_ROWS と FOUND_ROWS() は MySQL バージョン 4.0.0 以降で使用できる。

- GET_LOCK(str,timeout)

  str に指定された名前のロックを取得しようと試みる。この場合、timeout に指定された秒数をタイムアウトとする。正常にロックが取得されたときは 1 を返し、ロックの取得がタイムアウトになった (指定された名前が別のクライアントによってすでにロックされている場合など) ときは 0 を返し、エラーが発生した (メモリ不足や、mysqladmin kill でスレッドが強制終了された場合など) ときは NULL を返す。ロックは、RELEASE_LOCK() が実行されるか、新しい GET_LOCK() が実行されるか、またはスレッドが終了する (正常終了または異常終了) と解除される。

  この関数はアプリケーションのロックを実行したり、レコードのロックをシミュレーションしたりする目的で使用できる。名前はサーバ全体の範囲で有効である。 1 つのクライアントがある名前のロックを得ている場合、GET_LOCK() は、その名前のロックに対する、別のクライアントからのすべての要求をブロックする。 ある名前のロックをもとにすれば、各クライアント間での同期をとらせるような動作が可能である :

```
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
```

  注意: 2 番目の RELEASE_LOCK() 呼び出しでは NULL が返される。これは、ロック "lock1" が 2 番目の GET_LOCK() 呼び出しによって自動的に解除されるため。

- INET_ATON(expr)

  文字列として指定された、ドット 10 進表記のネットワークアドレスを、そのアドレスの数値を表す整数として返す。アドレスとして、4 バイトまたは 8 バイトのアドレスを指定できる。

```
mysql> SELECT INET_ATON("209.207.224.40");
    ->  3520061480
```

  生成される数値は、常にネットワークバイトオーダーで生成される。たとえば、上の例の数値は、209*256^3 + 207*256^2 + 224*256 +40 として計算される。

- INET_NTOA(expr)

数値として表現されたネットワークアドレス (4 または 8 バイト) を、ドット 10 進表記のアドレス (文字列) として返す。

```
mysql> SELECT INET_NTOA(3520061480);
    -> "209.207.224.40"
```

- IS_FREE_LOCK(str)

  str に指定された名前をもつロックが解放されているかどうか (つまり、ロックされていないかどうか) 確認する。 ロックが解放されている (誰もそのロックを使用していない) 場合は 1 を返し、そのロックが使用されている場合は 0 を返し、エラーが発生した (引数に誤りがあるなど) 場合は NULL を返す。

- LAST_INSERT_ID([expr])

  AUTO_INCREMENT カラムに挿入された値のうち、最後に自動生成された値を返す。

```
mysql> SELECT LAST_INSERT_ID();
    -> 195
```

  生成された最後の ID は、接続ごとにサーバで維持される。したがって、この関数から個々のクライアントに返される値は、そのクライアントによって生成された最新の AUTO_INCREMENT 値である。この値は、他のクライアントがそれぞれの AUTO_INCREMENT 値を生成しても、それによって影響されることはない。この動作によって、他のクライアントの活動にかかわりなく、また、ロックやトランザクションを必要とすることなく、自分の ID を確実に取り出すことができる。

  非マジック値 (つまり、NULL でも 0 でもない値) を持つレコードの AUTO_INCREMENT カラムを更新しても、LAST_INSERT_ID() の値は変更されない。

  INSERT ステートメントで同時に複数のレコードを挿入した場合、LAST_INSERT_ID() は最初に挿入されたレコードの値を返す。これは、同じ INSERT ステートメントを他のいずれかのサーバに対して簡単に再生成できるようにするためである。

  LAST_INSERT_ID() の引数として expr を指定した場合、引数の値が関数から返され、この値が LAST_INSERT_ID() によって返される次の値として設定される。これはシーケンスのシミュレーションに使用できる。

  まず、テーブルを作成する。

```
mysql> CREATE TABLE sequence (id INT NOT NULL);
mysql> INSERT INTO sequence VALUES (0);
```

  その後、このテーブルを使用してシーケンス番号を次のように生成できる。

```
mysql> UPDATE sequence SET id=LAST_INSERT_ID(id+1);
```

シーケンスの生成は LAST_INSERT_ID() を呼び出さなくても可能だが、この関数をこのように使用した場合、ID 値が最後に自動生成された値としてサーバに維持される（マルチユーザ対応）。新しい ID は、MySQL で通常の AUTO_INCREMENT 値を読み取るときと同じように取り出せる。たとえば、LAST_INSERT_ID()（引数はいずれもなし）では、新しい ID が返される。C API 関数 mysql_insert_id() もこの ID 値の取得に使用できる。

注意: mysql_insert_id() が更新されるのは、INSERT ステートメントや UPDATE ステートメントの後になるため、SELECT や SET などの他の SQL ステートメントの実行後に、C API 関数を使用して LAST_INSERT_ID(expr) の値を取り出すことはできない。 See 項11.1.3.32.「mysql_insert_id()」。

- MASTER_POS_WAIT(log_name, log_pos [, timeout])

  スレーブがマスタログの指定された位置に達する（つまり、指定された位置まで更新をすべて読み取って適用する）までブロックする。マスタ情報が初期化されていない場合や、引数に誤りがある場合は、NULL を返す。スレーブが実行されていない場合は、スレーブが開始され、指定された位置まで到達するか、これを通過するまで、ブロックして待機する。スレーブが指定された位置をすでに通過している場合、この関数は直ちに戻る。

  timeout（4.0.10 で導入）が指定されている場合は、timeout に指定された秒数が経過すると待機を中止する。timeout 値は 0 より大きくなければならない。ゼロや負の timeout 値はタイムアウトなしを意味する。戻り値として、指定された位置に到達するまでに待機しなければならかった口グイベントの数が返される。エラー時には、NULL が返され、タイムアウトを超過した場合は -1 が返される。

  このコマンドはマスタとスレーブの同期化に役立つ。

- MD5(string)

  指定された文字列の MD5 128 ビットチェックサムを計算する。値は 32 ビットの 16 進数として返される。この値は、ハッシュキーなどとして使用できる。

```
mysql> SELECT MD5("testing");
    -> 'ae2b1fca515949e5d54fb22b8ed95575'
```

  これは "RSA Data Security, Inc. の MD5 Message-Digest Algorithm"。

- PASSWORD(str) , OLD_PASSWORD(str)

  平文テキストのパスワード str からパスワード文字列を計算する。これは、user 権限テーブルの Password カラムに格納する MySQL パスワードの暗号化に使用される関数。

```
mysql> SELECT PASSWORD('badpwd');
    -> '7f84554057dd964b'
```

  PASSWORD() の暗号化は可逆ではない。

  PASSWORD() は、パスワードの暗号化を Unix パスワードの暗号化と同じ方法では行わない。ENCRYPT() を参照。

注意: PASSWORD() 関数は、MySQL サーバの認証システムで使用される。アプリケーションでは使用しないこと。アプリケーション用には、MD5() か SHA1() を代わりに使用する。 アプリケーションでのパスワードと認証の安全な処理の詳細については、 RFC-2195 も参照。

- RELEASE_LOCK(str)

  GET_LOCK() によって取得された、文字列 str を名前として持つロックを解除する。そのロックが解除された場合は 1 を返し、そのロックがこのスレッドによってロックされているのでない場合は 0 を返し（この場合、ロックは解除されない）、指定されたロックが存在しない場合は NULL を返す（そのロックが GET_LOCK() を呼び出して取得されたのでない場合や、すでに解除されている場合、そのロックは存在しないことになる）。

  DO ステートメントを RELEASE_LOCK() で使用すると便利である。 See 項6.4.10.「DO 構文」。

- SESSION_USER()

  SESSION_USER() は USER() のシノニム。

- SHA1(string) , SHA(string)

  指定された文字列に対して、RFC 3174（Secure Hash Algorithm）に定義された SHA1 160 ビットチェックサムを計算する。値は 40 桁の 16 進数として返される。入力引数が NULL の場合は、NULL が返される。 この関数の用途の 1 つとして、戻り値をハッシュキーとして使用できる。また、この関数は、パスワードを格納するための安全な暗号化関数としても使用できる。

```
mysql> SELECT SHA1("abc");
    -> 'a9993e364706816aba3e25717850c26c9cd0d89d'
```

  SHA1() はバージョン 4.0.2 で追加された。この関数は MD5() よりもさらに安全度が高い暗号化関数とみなすことができる。 SHA() は SHA1() のシノニム。

- SYSTEM_USER()

  SYSTEM_USER() は USER() のシノニム。

- UNCOMPRESS(string_to_uncompress)

  COMPRESS() 関数によって圧縮された文字列の圧縮を解除する。

```
mysql> SELECT UNCOMPRESS(COMPRESS("any string"));
    -> 'any string'
```

  UNCOMPRESS() は MySQL バージョン 4.1.1 で追加された。 この関数を使用するためには、zlib などの圧縮ライブラリを使用して MySQL をコンパイルしておく必要がある。コンパイルを行っていないと、戻り値は常に NULL になる。

- UNCOMPRESSED_LENGTH(compressed_string)

  圧縮文字列の圧縮前の長さを返す。

```
mysql> SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT("a",30)));
    -> 30
```

  UNCOMPRESSED_LENGTH() は MySQL バージョン 4.1.1 で追加された。

- USER()

  現在の MySQL ユーザ名とホスト名を返す。

```
mysql> SELECT USER();
    -> 'davida@localhost'
```

  戻り値は、サーバへの接続時に指定したユーザ名と、接続元のクライアントホストを表す (MySQL バージョン 3.22.11 より前のバージョンでは、この関数の戻り値にはクライアントホスト名は含まれない) 。

  次のようにした場合、値にホスト名の部分が含まれているかどうかにかかわらず、ユーザ名の部分だけを取り出すこともできる。

```
mysql> SELECT SUBSTRING_INDEX(USER(),"@",1);
    -> 'davida'
```

- VERSION()

  MySQL サーバのバージョンを示す文字列を返す。

```
mysql> SELECT VERSION();
    -> '3.23.13-log'
```

  注意: バージョンの後ろに -log が付いている場合、ログが有効になっていることを表す。

6.3.7. GROUP BY 節で使用する関数と修飾子

6.3.7.1. GROUP BY 関数

GROUP BY 節を含まないステートメントでグループ関数を使用すると、すべてのレコードに対してグループ化操作を実行するのと同じになります。

- AVG(expr)

  expr の平均値を返す。

```
mysql> SELECT student_name, AVG(test_score)
    ->       FROM student
    ->       GROUP BY student_name;
```

- BIT_AND(expr)

  expr のすべてのビットに対するビットごとの AND を返す。この計算は 64 ビット (BIGINT) の精度で実行される。

  MySQL 4.0.17 以降では、マッチするレコードがない場合、この関数は 18446744073709551615 を返す (これはすべてのビットを 1 に設定した、符号なしの BIGINT 値) 。 4.0.17 より前のバージョンでは、マッチするレコードがない場合、この関数は -1 を返す。

- BIT_OR(expr)

  expr のすべてのビットに対するビットごとの OR を返す。この計算は 64 ビット (BIGINT) の精度で実行される。

  マッチするレコードがない場合、この関数は 0 を返す。

- BIT_XOR(expr)

  expr のすべてのビットに対するビットごとの XOR を返す。この計算は 64 ビット (BIGINT) の精度で実行される。

  マッチするレコードがない場合、この関数は 0 を返す。

  この関数は MySQL 4.1.1 以降で使用できる。

- COUNT(expr)

  SELECT ステートメントで取り出されたレコードのうち、NULL 以外の値の数のカウントを返す。

```
mysql> SELECT student.student_name,COUNT(*)
    ->       FROM student,course
    ->       WHERE student.student_id=course.student_id
    ->       GROUP BY student_name;
```

  COUNT(*) は、レコードに NULL 値が含まれているかどうかにかかわらず、取り出されたレコード数のカウントを返すという点で多少異なる。

  SELECT で 1 つのテーブルから値を取り出し、他のカラムからは値を取り出さず、かつ WHERE 節がない場合、COUNT(*) は迅速に戻るよう最適化される。次に例を示す。

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

この最適化は、MyISAM テーブルと ISAM テーブルにのみ適用される。これは、この 2 つのテーブル型では正確なレコードカウントが格納され、迅速にアクセスできるためである。トランザクションストレージエンジン (InnodB, BDB) の場合、複数のトランザクションが発生し、それぞれがカウントに影響する場合があるため、正確なレコードカウントの格納はより難しくなる。

- COUNT(DISTINCT expr,[expr...])

  NULL 以外の異なる各値の数のカウントを返す。

```
mysql> SELECT COUNT(DISTINCT results) FROM student;
```

MySQL では、式のリストを指定することによって、NULL を含まない、重複のない式の組み合せの数を得ることができる。 SQL-99 では、COUNT(DISTINCT ...) 内ですべての式を連結する必要がある。

- GROUP_CONCAT(expr)

  完全な構文

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
        [ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC] [,col ...]]
        [SEPARATOR str_val])
```

この関数は MySQL バージョン 4.1 で追加された。 この関数はグループ内の値を連結した結果の文字列を返す。

```
mysql> SELECT student_name,
    ->       GROUP_CONCAT(test_score)
    ->       FROM student
    ->       GROUP BY student_name;
または
mysql> SELECT student_name,
    ->       GROUP_CONCAT(DISTINCT test_score
    ->                ORDER BY test_score DESC SEPARATOR " ")
    ->       FROM student
    ->       GROUP BY student_name;
```

MySQL では、式の組み合わせの連結値を得ることができる。 DISTINCT を使用することで、重複する値は排除できる。 結果の値をソートするには、ORDER BY 節を使用する。 逆の順序でソートするには、ORDER BY 節でソートキーとするカラムの名前に DESC (降順) キーワードを付ける。デフォルトは昇順だが、昇順を明示的に指定するには、ASC キーワードを指定する。 SEPARATOR は結果の値と値の間に挿入する文字列値を表す。デフォルトはカンマ (",") 。区切り記号をまったく使用しない場合は、SEPARATOR "" と指定する。

使用可能な最大長は、変数 group_concat_max_len をオプション設定ファイルに記述することで設定できる。 設定を SQL クエリで行う構文は次のとおり。

```
SET [SESSION | GLOBAL] group_concat_max_len = unsigned_integer;
```

最大長が設定されている場合、結果はこの最大長に合わせて切り捨てられる。

GROUP_CONCAT() 関数は、Sybase SQL Anywhere でサポートされている基本の LIST() 関数を拡張した実装である。 カラムが 1 つだけで、その他のオプションが指定されていない場合、GROUP_CONCAT() は機能がかなり制限されている LIST() と下位互換になる。ただし、LIST() にはデフォルトのソート順序はある。

- MIN(expr) , MAX(expr)

  expr の最小値または最大値を返す。MIN() と MAX() は文字列引数を取ることができる。その場合、最小または最大の文字列値を返す。 See 項5.4.3.「MySQL でのインデックスの使用」。

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
    ->       FROM student
    ->       GROUP BY student_name;
```

MySQL では、現在のところ、MIN() と MAX() やその他の集約関数において ENUM 型と SET 型のカラムの比較を、セット内での文字列の相対的な位置ではなくそれぞれの文字列値に基づいて行っている。 これは改正される予定。

- STD(expr) , STDDEV(expr)

  expr の標準偏差 (VARIANCE() の平方根) を返す。これは SQL-99 に対する拡張。この関数の STDDEV() の形式は Oracle との互換性を確保するために提供されている。

- SUM(expr)

  expr の合計を返す。注意: 戻り値のセットにレコードが含まれていない場合、NULL を返す。

- VARIANCE(expr)

  expr の標準偏差を返す (レコードはサンプルではなく、母集団全体とみなされる。そのため、この関数はレコード数を分母として取る)。これは SQL-99 に対する拡張 (バージョン 4.1 以降で使用可能)。

6.3.7.2. GROUP BY の修飾子

MySQL 4.1.1 以降では、GROUP BY 節に WITH ROLLUP 修飾子を使用することによって、合計出力に追加のレコードを挿入できます。これらの追加レコードは上位レベル (上位集約) の集計演算を表します。そのため、ROLLUP では、複数の分析レベルでの疑問に 1 回のクエリで答えることができます。たとえば、この修飾子は、OLAP (オンライン分析処理) をサポートする目的で使用できます。

例として、sales という名前のテーブルに、売上の利益を記録するための year、country、product、profit という名前のカラムがあるとします。

```
CREATE TABLE sales
(
```

```
  year    INT NOT NULL,
  country VARCHAR(20) NOT NULL,
  product VARCHAR(32) NOT NULL,
  profit  INT
);
```

このテーブルの内容は、次のように、単純な GROUP BY 節の使用により、集計することができます。

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year;
+------+-------------+
| year | SUM(profit) |
+------+-------------+
| 2000 |        4525 |
| 2001 |        3010 |
+------+-------------+
```

この出力からは各年度の総利益がわかりますが、すべての年度の利益を合計した総利益を調べる必要がある場合は、個々の値を自分で足すか、またはもう一度クエリを実行しなければなりません。

しかし、ROLLUP を使用すれば、2 つのレベルの分析を 1 回のクエリで実行することができます。 GROUP BY 節に WITH ROLLUP 修飾子を追加すると、クエリですべての年度値を足した総計を示すレコードが別に生成されます。

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year WITH ROLLUP;
+------+-------------+
| year | SUM(profit) |
+------+-------------+
| 2000 |        4525 |
| 2001 |        3010 |
| NULL |        7535 |
+------+-------------+
```

総計の上位集約行は、year カラムの値 NULL によって示されます。

ROLLUP は、GROUP BY カラムが複数ある場合、より複雑に作用します。この場合、最後のグループ化カラム以外の場所で ``ブレーク" (値の変化) があると、そのつどクエリで追加の上位集約の集計レコードが生成されます。

たとえば、ROLLUP を指定していない場合に、sales テーブルの、year、country、product に基づく集計が次のように出力されるとします。

```
mysql> SELECT year, country, product, SUM(profit)
    -> FROM sales
    -> GROUP BY year, country, product;
+------+---------+------------+-------------+
| year | country | product    | SUM(profit) |
+------+---------+------------+-------------+
| 2000 | Finland | Computer   |        1500 |
| 2000 | Finland | Phone      |         100 |
| 2000 | India   | Calculator |         150 |
| 2000 | India   | Computer   |        1200 |
| 2000 | USA     | Calculator |          75 |
| 2000 | USA     | Computer   |        1500 |
| 2001 | Finland | Phone      |          10 |
| 2001 | USA     | Calculator |          50 |
| 2001 | USA     | Computer   |        2700 |
| 2001 | USA     | TV         |         250 |
```

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

この出力は、年/国/製品レベルでの分析における集計値のみを示しています。ROLLUP を追加すると、クエリでいくつかの追加のレコードが生成されます。

```
mysql> SELECT year, country, product, SUM(profit)
    -> FROM sales
    -> GROUP BY year, country, product WITH ROLLUP;
+------+---------+------------+-------------+
| year | country | product    | SUM(profit) |
+------+---------+------------+-------------+
| 2000 | Finland | Computer   |        1500 |
| 2000 | Finland | Phone      |         100 |
| 2000 | Finland | NULL       |        1600 |
| 2000 | India   | Calculator |         150 |
| 2000 | India   | Computer   |        1200 |
| 2000 | India   | NULL       |        1350 |
| 2000 | USA     | Calculator |          75 |
| 2000 | USA     | Computer   |        1500 |
| 2000 | USA     | NULL       |        1575 |
| 2000 | NULL    | NULL       |        4525 |
| 2001 | Finland | Phone      |          10 |
| 2001 | Finland | NULL       |          10 |
| 2001 | USA     | Calculator |          50 |
| 2001 | USA     | Computer   |        2700 |
| 2001 | USA     | TV         |         250 |
| 2001 | USA     | NULL       |        3000 |
| 2001 | NULL    | NULL       |        3010 |
| NULL | NULL    | NULL       |        7535 |
+------+---------+------------+-------------+
```

このクエリの場合、ROLLUP を追加すると、単に 1 レベルではなく、4 レベルでの分析における集計情報が出力に組み込まれます。ROLLUP の出力は次のように解釈されます。

- 特定の年度と国に対応する各製品レコードセットの後に、製品ごとのすべて値の合計を示す追加の集計レコードが 1 つずつ生成される。これらのレコードでは、product カラムの値が NULL に設定される。
- 特定の年度に対応する各レコードセットの後に、国ごと、製品ごとのすべての値の合計を示す追加の集計レコードが 1 つずつ生成される。これらのレコードでは、country カラムと products カラムの値が NULL に設定される。
- 最後に、他のすべてのレコードの後に、年度ごと、国ごと、製品ごとのすべての値の総計を示す追加の集計レコードが 1 つ生成される。このレコードでは、year、country、products の各カラムの値が NULL に設定される。

ROLLUP 使用時のその他の考慮事項

以下に、MySQL における ROLLUP の実装固有の動作について、いくつか説明します。

ROLLUP の使用時には、ORDER BY 節を使用して結果をソートすることはできません (ROLLUP と ORDER BY は相互排他的です) 。しかし、ソート順序をある程度制御することは可能です。 MySQL で GROUP BY を使用すると結果がソートされます。また、GROUP BY リストに指定したカラムに明示的な ASC または DESC キーワードを付けることによって、個々のカラムのソート順序を指定できます (この場合も、ROLLUP によって追加される上位レベルの集計レコードは、ソート順序とはかかわりなく、それぞれの計算の対象となったレコードの後に表示されます) 。

LIMIT を使用すると、クライアントに返されるレコードの数を制限することができます。LIMIT は ROLLUP の後に適用されるため、制限は ROLLUP によって挿入される追加のレコードにも適用されます。次に例を示します。

```
mysql> SELECT year, country, product, SUM(profit)
    -> FROM sales
    -> GROUP BY year, country, product WITH ROLLUP
    -> LIMIT 5;
+------+---------+------------+-------------+
| year | country | product    | SUM(profit) |
+------+---------+------------+-------------+
| 2000 | Finland | Computer   |        1500 |
| 2000 | Finland | Phone      |         100 |
| 2000 | Finland | NULL       |        1600 |
| 2000 | India   | Calculator |         150 |
| 2000 | India   | Computer   |        1200 |
+------+---------+------------+-------------+
```

注意: LIMIT を ROLLUP とともに使用すると、上位集約レコードを理解するためのコンテキストが少なくなるため、生成される結果の解釈が難しくなることがあります。

各上位集約レコードの NULL インジケータは、各レコードがクライアントに送られるときに生成されます。サーバは、値が変わった左端のカラムの後に、GROUP BY 節に指定された各カラムを確認します。 これらのカラム名と語彙が一致する名前を持つカラムが結果セットにあると、そのカラムの値を NULL に設定します (カラム番号によるカラムのグループ化が指定されている場合、サーバは NULL に設定するカラムを番号で識別します) 。

上位集約レコードの NULL 値はクエリ処理の後の方の段階で結果セットに組み込まれるため、クエリ自体の中で上位集約レコードを NULL 値としてテストすることはできません。たとえば、クエリに HAVING product IS NULL を追加して、上位集約レコード以外のすべての出力を排除することはできません。

それに対し、クライアント側では、NULL 値が表示されるため、MySQL クライアントプログラミングインタフェースを使用して NULL 値としてテストすることができます。

6.3.7.3. 非表示のフィールドに対する GROUP BY

MySQL では GROUP BY の使用を拡張しています。GROUP BY 部分にないカラムや計算を SELECT 式で使用することができます。これは、そのグループの使用可能なあらゆる値を表します。この機能により、不要項目に対するソートやグループ化を行わないことで、パフォーマンスを向上させることができます。たとえば、次のクエリの場合、customer.name をグループ化する必要はありません。

```
mysql> SELECT order.custid,customer.name,MAX(payments)
    ->       FROM order,customer
    ->       WHERE order.custid = customer.custid
    ->       GROUP BY order.custid;
```

標準 SQL では、customer.name を GROUP BY 節に組み込む必要があります。MySQL では、非 ANSI モードでの実行時にはこのカラム名は不要です。

GROUP BY 部分から取り除くカラムがグループ内で一意なものではない場合は、この機能を使用しないでください。予測不可能な結果になります。

場合によっては、MIN() と MAX() を使用することによって、一意なものでないカラムの値を取り出すことができます。次の例では、sort カラムの値が最も小さいレコードの column 値が得られます。

```
SUBSTR(MIN(CONCAT(RPAD(sort,6,' '),column)),7)
```

See 項3.6.4.「特定のフィールドのグループごとの最大値が格納されているレコード」。

注意: MySQL バージョン 3.22 (またはこれより前のバージョン) を使用している場合や、SQL-99 に従う必要がある場合、GROUP BY 節と ORDER BY 節では式は使用できません。この制約は式の工イリアスを使用することによって回避できます。

```
mysql> SELECT id,FLOOR(value/100) AS val FROM tbl_name
    ->       GROUP BY id,val ORDER BY val;
```

MySQL バージョン 3.23 では、次のようにします。

```
mysql> SELECT id,FLOOR(value/100) FROM tbl_name ORDER BY RAND();
```

6.4. データの操作: SELECT、INSERT、UPDATE、DELETE

6.4.1. SELECT 構文

```
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]
    select_expression,...
    [INTO {OUTFILE | DUMPFILE} 'file_name' export_options]
    [FROM table_references
      [WHERE where_definition]
      [GROUP BY {unsigned_integer | col_name | formula} [ASC | DESC], ...
        [WITH ROLLUP]]
      [HAVING where_definition]
      [ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC] ,...]
      [LIMIT [offset,] row_count | row_count OFFSET offset]
      [PROCEDURE procedure_name(argument_list)]
      [FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT は1 つ以上のテーブルからレコードを選択して取り出すときに使用します。 各 select_expression は、取り出すカラムを表します。 SELECT は、どのテーブルも参照することなく、計算によって求められたレコードを取り出すときにも使用できます。

次に例を示します。

```
mysql> SELECT 1 + 1;
       -> 2
```

使用する節はいずれも、構文の説明で示している順序とまったく同じ順序で記述する必要があります。たとえば、HAVING 節はあらゆる GROUP BY 節の後に、またあらゆる ORDER BY 節の前に配置しなければなりません。

- SELECT 式には、AS alias_name を使用して工イリアスを指定することができる。 このエイリアスは式のカラム名として使用されるもので、ORDER BY 節または HAVING 節とともに使用できる。次に例を示す。

  ```
  mysql> SELECT CONCAT(last_name,', ',first_name) AS full_name
  ```

```
  FROM mytable ORDER BY full_name;
```

SELECT 式のエイリアスの指定時、AS キーワードは省略できる。 前の例は、次のように記述することも可能。

```
mysql> SELECT CONCAT(last_name,', ',first_name) full_name
  FROM mytable ORDER BY full_name;
```

AS は省略できるため、2 つの SELECT 式の間にカンマを付け忘れると微妙な問題が生じることがある。この場合、2 番目の式は MySQL によって エイリアス名として解釈される。たとえば、次のステートメントで、columnb はエイリアス名とみなされる。

```
mysql> SELECT columna columnb FROM mytable;
```

- WHERE 節ではカラム名のエイリアスは使用できない。これは、WHERE 節の実行時点でカラム値がまだ判別されないことがあるため。 See 項A.5.4.「alias の問題」。

- FROM table_references 節は、レコードの取り出し元のテーブルを表す。複数のテーブルを指定する場合は、結合操作を実行する。結合構文については、項6.4.1.1.「JOIN 構文」 を参照。 指定する各テーブルについて、必要に応じて、そのテーブルのエイリアスを指定することができる。

  ```
  table_name [[AS] alias] [[USE INDEX (key_list)] | [IGNORE INDEX (key_list)] | FORCE INDEX (key_list)]]
  ```

  MySQL バージョン 3.23.12 以降では、テーブルからのデータの取り出し時に、どのインデックスを使用すべきか MySQL に知らせるためのヒントを指定することができる。これは、EXPLAIN で MySQL が使用可能なインデックスのリストの中から誤ったインデックスを使用していることが明らかになったときに役立つ。USE INDEX (key_list) と指定することによって、使用可能なインデックスの中から特定のインデックスを使ってテーブル内のレコードを検索するよう MySQL に指示できる。もう 1 つの IGNORE INDEX (key_list) 構文は、特定のインデックスを使用しないよう MySQL に指示するときに使用する。

  MySQL 4.0.9 では、FORCE INDEX も使用できる。これは USE INDEX (key_list) と同じように動作するが、異なる点として、この構文の場合、テーブルのスキャンは非常にコストがかかるという前提に立つ。つまり、テーブルのスキャンが実行されるのは、どのインデックスを使用してもテーブル内のレコードを検索できない場合に限られる。

  USE/IGNORE/FORCE KEY は USE/IGNORE/FORCE INDEX のシノニム。

  注意: USE/IGNORE/FORCE INDEX は、MySQL でテーブル内のレコードの検索方法と結合の実行方法を決めるときにどのインデックスを使用するか、という点にのみ影響する。ORDER BY や GROUP BY の解決時にインデックスを使用するかどうかには影響しない。

  MySQL 4.0.14 では、MySQL にテーブルのスキャンよりもキーのスキャンを優先させるための代替方法として、SET MAX_SEEKS_FOR_KEY=value も使用できる。

- テーブルは (カレントデータベース内のものなら) tbl_name として参照できる。データベースを明示的に指定する場合は、dbname.tbl_name として参照する。 カラムは、col_name、tbl_name.col_name、または db_name.tbl_name.col_name として参照できる。SELECT ステートメントでカラムを参照するときには、その参照があいまいにならない限り、カラム名の前に tbl_name や db_name.tbl_name を付ける必要はない。より明示的なカラム参照形式が必要となるあいまいな例については、項6.1.2.「データベース名、テーブル名、インデックス名、カラム名、エイリアス名」を参照。

- バージョン 4.1.0 以降では、どのテーブルも参照されない状況において、ダミーテーブル名として DUAL を指定するこ

とができる。これはもっぱら互換性を考慮した機能である。一部の他のサーバではこの構文が必要となる。

```
mysql> SELECT 1 + 1 FROM DUAL;
    -> 2
```

- テーブル参照には、tbl_name [AS] alias_name を使用して エイリアスを指定することができる。

```
mysql> SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
  ->       WHERE t1.name = t2.name;
mysql> SELECT t1.name, t2.salary FROM employee t1, info t2
  ->       WHERE t1.name = t2.name;
```

- 出力用に選択されるカラムは、ORDER BY 節と GROUP BY 節において、カラム名、カラムエイリアス、またはカラム位置を使用して参照することができる。カラム位置は 1 から始まる。

```
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;
```

逆の順序でソートするには、ORDER BY 節においてソートキーとするカラムの名前に DESC (降順) キーワードを付ける。 デフォルトは昇順だが、昇順を明示的に指定するには、ASC キーワードを指定する。

- WHERE 節では、MySQL でサポートしている任意の機能を使用できるが、例外として集約 (集計) 機能だけは使用できない。 See 項6.3.「SELECT 節と WHERE 節で使用する関数」。

- HAVING 節では、select_expression で指定した任意のカラムまたは エイリアスを参照することができる。この節は、処理の終わり近くになって (項目がクライアントに送られる直前)、最適化されることなく適用される (LIMIT は HAVING の後に適用)。WHERE 節に組み込むべき項目に対しては、HAVING を使用してはならない。たとえば、次のように記述しないようにする。

```
mysql> SELECT col_name FROM tbl_name HAVING col_name > 0;
```

正しくは次のように記述する。

```
mysql> SELECT col_name FROM tbl_name WHERE col_name > 0;
```

MySQL バージョン 3.22.5 以降では、次のようなクエリも使用できる。

```
mysql> SELECT user,MAX(salary) FROM users
  ->       GROUP BY user HAVING MAX(salary)>10;
```

それより前の MySQL バージョンでは、代わりに次のように記述する。

```
mysql> SELECT user,MAX(salary) AS sum FROM users
  ->       group by user HAVING sum>10;
```

- DISTINCT、DISTINCTROW、ALL の各オプションでは、重複レコードを返すかどうかを指定する。デフォルト (ALL) では、マッチするすべてのレコードが返される。DISTINCT と DISTINCTROW はシノニムで、これらを指定した場

合、結果セットから重複レコードが削除される。

- STRAIGHT_JOIN と HIGH_PRIORITY、および SQL_ で始まる各オプションは、SQL-99 に対する MySQL の拡張。
  - STRAIGHT_JOIN では、オプティマイザに、FROM 節に指定された順序で強制的にテーブルを結合させることができる。オプティマイザが最適でない順序でテーブルを結合するときには、このオプションを使用することによって、クエリを高速化できる。 See 項5.2.1.「EXPLAIN 構文 (SELECT に関する情報の取得) 」。
  - HIGH_PRIORITY を指定すると、SELECT ステートメントがテーブルを更新するステートメントよりも優先される。このオプションは、非常に迅速に処理され、かつ直ちに実行する必要があるクエリに対してのみ使用するようにする。SELECT HIGH_PRIORITY クエリは、対象のテーブルが読み取り用にロックされれば、たとえテーブルが解放されるのを待っている更新ステートメントがあっても、実行される。
  - SQL_BIG_RESULT は、GROUP BY または DISTINCT とともに使用することができ、それによって、結果セットに多くのレコードが組み込まれることをオプティマイザに通知できる。この場合、MySQL で、ディスクベースの一時テーブルが必要に応じて直接使用される。 また、この場合、GROUP BY 要素のキーで一時テーブルを処理するよりソートを行うほうが優先される。
  - SQL_BUFFER_RESULT では、結果が一時テーブルに強制的に格納される。結果セットをクライアントに送るのに時間がかかる場合には、このオプションを指定することによって、MySQL によるテーブルのロックを早く解放させることができる。
  - SQL_SMALL_RESULT は MySQL 固有のオプション。このオプションを GROUP BY または DISTINCT とともに使用することで、結果セットが小さいものになることをオプティマイザに通知できる。この場合、ソートを行う代わりに、迅速な一時テーブルを使用して結果のテーブルが格納される。MySQL バージョン 3.23 では、通常、このオプションを使用する必要はない。
  - SQL_CALC_FOUND_ROWS (バージョン 4.0.0 以降) では、LIMIT 節を無視した場合に結果セットに含まれるすべてのレコード数を計算するよう MySQL に指示できる。 その後、SELECT FOUND_ROWS() を使用して、計算されたレコード数を取り出せる。 See 項6.3.6.2.「その他の各種関数」。

    注意: 4.1.0 より前のバージョンでは、LIMIT 0 を指定した場合、このオプションは機能しない。この場合、迅速に戻るよう最適化される (その結果、レコードカウントは 0 になる) 。 See 項5.2.9.「MySQL による LIMIT の最適化」。
  - SQL_CACHE では、QUERY_CACHE_TYPE=2 (DEMAND) を使用した場合に、クエリの結果をクエリキャッシュに格納するよう MySQL に指示できる。 See 項6.9.「MySQL クエリキャッシュ」。 UNION またはサブクエリを使用したクエリでは、このオプションはクエリのあらゆる SELECT ステートメントで適用される。
  - SQL_NO_CACHE では、クエリの結果をクエリキャッシュに格納しないよう、MySQL に指示することができる。See 項6.9.「MySQL クエリキャッシュ」。 UNION またはサブクエリを使用したクエリでは、このオプションはクエリのあらゆる SELECT ステートメントで適用される。
- GROUP BY を使用すると、GROUP BY のすべてのフィールドに対して ORDER BY を使用した場合と同じように、出力レコードが GROUP BY に指定した基準に基づいてソートされる。MySQL では、GROUP BY 節を拡張しており、この節で指定したカラムの後に ASC と DESC のいずれかを指定することができる。

```
SELECT a,COUNT(b) FROM test_table GROUP BY a DESC
```

- MySQL では、GROUP BY の使用を拡張しており、GROUP BY 節に指定していないフィールドも選択できる。 クエリ

で期待した結果が得られない場合は、GROUP BY の説明を参照。 See 項6.3.7.「GROUP BY 節で使用する関数と修飾子」。

- MySQL 4.1.1 以降、GROUP BY には WITH ROLLUP 修飾子を使用できる。 See 項6.3.7.2.「GROUP BY の修飾子」。
- LIMIT 節を使用すると、SELECT ステートメントで返されるレコード数を制限することができる。LIMIT は 1 つまたは 2 つの数値引数を取る。これらの引数は整数定数でなければならない。

  引数が 1 つの場合、その値は、戻り値として返す、結果セットの冒頭からのレコード数を表す。 引数が 2 つの場合、最初の引数は戻り値として返す最初のレコードまでのオフセットを表し、2 つ目の引数は戻り値として返す最大レコード数を表す。最初のレコードのオフセット値は 0 (1 ではない) 。

  PostgreSQL との互換性を確保するため、MySQL では LIMIT row_count OFFSET offset 構文もサポートしている。

```
mysql> SELECT * FROM table LIMIT 5,10;  # Retrieve rows 6-15
```

  特定のオフセット位置から結果セットの終わりまでのすべてのレコードを取り出すには、2 つ目のパラメータに大きな数値を指定できる。

```
mysql> SELECT * FROM table LIMIT 95,18446744073709551615; # Retrieve rows 96-last.
```

  引数が 1 つの場合、その値は戻り値として返す最大レコード数を表す。

```
mysql> SELECT * FROM table LIMIT 5;     # Retrieve first 5 rows
```

  つまり、LIMIT n は LIMIT 0,n と指定するのと同じである。

- SELECT ... INTO OUTFILE 'file_name' 形式の SELECT では、SELECT したレコードがファイルに書き込まれる。ファイルはサーバホスト上に作成される。既存のファイルを指定することはできない (この制限は、特に、/etc/passwd などのデータベーステーブルやファイルが破壊されないようにするため) 。この形式の SELECT を使用するには、サーバホストに対する FILE 権限が必要となる。

  SELECT ... INTO OUTFILE ステートメントの用途は、主に、サーバマシン上のテーブルのダンプをきわめて迅速に実行できるようにすることである。サーバホスト以外のホストに結果のファイルを作成する必要があるときには、SELECT ... INTO OUTFILE は使用できない。この場合、代わりに、mysqldump --tab や mysql -e "SELECT ..." > outfile などのクライアントプログラムを使用してファイルを生成する。

  SELECT ... INTO OUTFILE は LOAD DATA INFILE の逆。このステートメントの export_options 部分の構文は、LOAD DATA INFILE ステートメントで使用される FIELDS および LINES 節と同じ構成になっている。 See 項6.4.8.「LOAD DATA INFILE 構文」。

  結果のテキストファイルでは、以下の文字だけが ESCAPED BY 指定の文字で エスケープされる。

  - ESCAPED BY 指定文字自体
  - FIELDS TERMINATED BY 指定の最初の文字
  - LINES TERMINATED BY 指定の最初の文字

  また、ASCII 0 は、ESCAPED BY の指定文字の後ろに 0 (ASCII 48) を付けたものに変換される。

上記の文字をエスケープする理由は、ファイルを確実に読み返せるようにするには、FIELDS TERMINATED BY、ESCAPED BY、LINES TERMINATED BY 指定されている文字をすべてエスケープしなければならないため。ASCII 0 は一部のページャーでの表示を容易化するためにエスケープされる。

結果のファイルは SQL の構文に従う必要はないため、その他のエスケープ処理は必要ない。

次に、多くの古いプログラムで使用されている形式でファイルを生成する例を示す。

```
SELECT a,b,a+b INTO OUTFILE "/tmp/result.text"
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ""
LINES TERMINATED BY "\n"
FROM test_table;
```

- INTO OUTFILE の代わりに INTO DUMPFILE を使用すると、MySQL によってファイルに 1 つのレコードだけが書き込まれる。この場合、カラムや行の終了は何も含まれず、エスケープ処理もまったく行われない。これは、ファイルに BLOB 値を格納するときに役立つ。
- 注意: INTO OUTFILE と INTO DUMPFILE で作成されるファイルは、サーバホスト上にあるアカウントで書き込み可能であるかもしれない。その理由は、MySQL サーバを実行しているアカウントで、それらファイルを作成するからであり、作成されたファイルはそのアカウントがオーナーになっているからである (mysqld は root として実行すべきでない) 。 ファイルの内容を操作できるようにするには、ファイルを全ユーザ書き込み可能にする必要がある。
- PROCEDURE 節には、結果セット内のデータの処理に使用するプロシージャを指定する。例については、項13.3.1.「プロシージャの分析」を参照。
- ページロックまたは行ロック機能があるストレージエンジンに対して FOR UPDATE を使用すると、現在のトランザクションが終了するまで、検査対象のレコードは書き込みに対してロックされる。

6.4.1.1. JOIN 構文

MySQL では、SELECT ステートメントでの以下の JOIN 構文の使用をサポートしています。

```
table_reference, table_reference
table_reference [INNER | CROSS] JOIN table_reference [join_condition]
table_reference STRAIGHT_JOIN table_reference
table_reference LEFT [OUTER] JOIN table_reference [join_condition]
table_reference NATURAL [LEFT [OUTER]] JOIN table_reference
{ OJ table_reference LEFT OUTER JOIN table_reference ON conditional_expr }
table_reference RIGHT [OUTER] JOIN table_reference [join_condition]
table_reference NATURAL [RIGHT [OUTER]] JOIN table_reference
```

table_reference は次のように定義します。

```
table_name [[AS] alias] [[USE INDEX (key_list)] | [IGNORE INDEX (key_list)] | [FORCE INDEX (key_list)]]
```

join_condition は次のように定義します。

```
ON conditional_expr |
USING (column_list)
```

結果セットに含めるレコードを制限するために使用する ON 部分には、通常、条件は何も指定しません。それらの条件は WHERE 節で指定します。ただし、この規則には例外があります。

注意:INNER JOIN 構文で join_condition を使用できるのは、MySQL 3.23.17 以降に限られます。同様に、JOIN と CROSS JOIN に関しても、MySQL 4.0.11 以降でのみ条件を指定できます。

上の構文リストの最後に挙げた LEFT OUTER JOIN 構文は、単に ODBC との互換性を確保するためのものです。

- テーブル参照では、tbl_name AS alias_name または tbl_name alias_name を使用して工イリアスを指定することができる。

```
mysql> SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
    ->       WHERE t1.name = t2.name;
```

- ON 条件句には、WHERE 節で使用できる形式の任意の条件を指定できる。
- LEFT JOIN の ON または USING 部分に右側のテーブルにマッチするレコードがない場合、すべてのカラムを NULL に設定した 1 つのレコードが右側のテーブルとして使用される。この規則に基づいて、対応するレコードが別のテーブルに存在しないレコードをテーブル内で検索することができる。

```
mysql> SELECT table1.* FROM table1
    ->       LEFT JOIN table2 ON table1.id=table2.id
    ->       WHERE table2.id IS NULL;
```

  この例では、table2 に存在しない id 値を持つすべてのレコード (table2 に対応するレコードがないすべてのレコード) が table1 で検索される。当然、この場合、table2.id が NOT NULL として宣言されていることが前提となる。 See 項 5.2.7.「MySQL による LEFT JOIN と RIGHT JOIN の最適化」。

- USING (column_list) 節に指定するカラムリスト内のカラムは、両方のテーブルに存在しなければならない。次の 2 つの節は同じことを意味する。

```
a LEFT JOIN b USING (c1,c2,c3)
a LEFT JOIN b ON a.c1=b.c1 AND a.c2=b.c2 AND a.c3=b.c3
```

- 2 つのテーブルの NATURAL [LEFT] JOIN の定義は、両方のテーブルに存在するすべてのカラムを指定した USING 節を持つ INNER JOIN または LEFT JOIN と同じ意味になる。
- INNER JOIN と , (カンマ) は、結合条件が指定されていない場合、同じ意味になり、どちらについても、指定されたテーブル間でデカルト積が生成される (つまり、最初のテーブルの各レコードが 2 番目のテーブルのすべてのレコードに結合される) 。
- RIGHT JOIN は LEFT JOIN の類似機能。コードをデータベース間で移植可能にするには、RIGHT JOIN ではなく LEFT JOIN を使用するようにする。
- STRAIGHT_JOIN は JOIN と同様のものだが、異なる点として、STRAIGHT_JOIN では常に右側のテーブルの前に左側のテーブルが読み取られる。この構文は、結合オプティマイザによってテーブルが誤った順序で並べられるといった (まれな) 事態に対処するために使用できる。
- MySQL バージョン 3.23.12 以降では、テーブルからのデータの取り出し時に、どのインデックスを使用すべきか MySQL に知らせるためのヒントを指定することができる。これは、EXPLAIN で MySQL が使用可能なインデックスの

リストの中から誤ったインデックスを使用していることが明らかになったときに役立つ。USE INDEX (key_list) と指定することによって、使用可能なインデックスの中から特定のインデックスを使ってテーブル内のレコードを検索するよう MySQL に指示できる。もう 1 つの IGNORE INDEX (key_list) 構文は、特定のインデックスを使用しないよう MySQL に指示するときに使用する。

MySQL 4.0.9 では、FORCE INDEX も使用できる。これは USE INDEX (key_list) と同じように動作するが、異なる点として、この構文の場合、テーブルのスキャンは非常にコストがかかるといいう前提に立つ。つまり、テーブルのスキャンが実行されるのは、どのインデックスを使用してもテーブル内のレコードを検索できない場合に限られる。

USE/IGNORE KEY は USE/IGNORE INDEX のシノニム。

注意: USE/IGNORE/FORCE INDEX は、MySQL でテーブル内のレコードの検索方法と結合の実行方法を決めるときにどのインデックスを使用するか、という点にのみに影響します。ORDER BY や GROUP BY の解決時にインデックスを使用するかどうかには影響しません。

以下に、例をいくつか示します。

```
mysql> SELECT * FROM table1,table2 WHERE table1.id=table2.id;
mysql> SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id;
mysql> SELECT * FROM table1 LEFT JOIN table2 USING (id);
mysql> SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id
    ->        LEFT JOIN table3 ON table2.id=table3.id;
mysql> SELECT * FROM table1 USE INDEX (key1,key2)
    ->        WHERE key1=1 AND key2=2 AND key3=3;
mysql> SELECT * FROM table1 IGNORE INDEX (key3)
    ->        WHERE key1=1 AND key2=2 AND key3=3;
```

See 項5.2.7.「MySQL による LEFT JOIN と RIGHT JOIN の最適化」。

6.4.1.2. UNION 構文

```
SELECT ...
UNION [ALL]
SELECT ...
 [UNION
  SELECT ...]
```

UNION は MySQL 4.0.0 で導入されました。

UNION は、多くの SELECT ステートメントを 1 つの結果セットに結合するために使用します。

SELECT の select_expression 部分のカラムリストには、同じ型のカラムを指定します。最初の SELECT クエリで指定したカラム名が、返される結果のカラム名として使用されます。

これらの SELECT コマンドは通常の SELECT コマンドですが、次の制限が適用されます。

- 最後の SELECT コマンドにのみ INTO OUTFILE を指定できる。

UNION にキーワード ALL を付けないと、総結果セットに対して DISTINCT を指定した場合と同じように、重複しない一意なレコードだけが返されます。ALL を指定すると、実行されたすべての SELECT ステートメントから、一致するすべて

のレコードが返されます。

UNION の総結果に対して ORDER BY を適用する必要があるときは、かっこを使用します。

```
(SELECT a FROM table_name WHERE a=10 AND B=1 ORDER BY a LIMIT 10)
UNION
(SELECT a FROM table_name WHERE a=11 AND B=2 ORDER BY a LIMIT 10)
ORDER BY a;
```

UNION の結果セット内のカラムの型と長さでは、すべての SELECT ステートメントで取り出された値が考慮されます。MySQL 4.1.1 より前のバージョンの UNION では、最初の SELECT で使用された値のみに基づいて結果の型と長さが決まる、という制約がありました。 この場合、たとえば、最初の SELECT の値よりも長い値が 2 番目の SELECT で取り出されると、切り捨てが行われることがあります。

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+--------------+
| REPEAT('a',1) |
+--------------+
| a            |
| b            |
+--------------+
```

MySQL 4.1.1 以降、この制約はなくなりました。

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+--------------+
| REPEAT('a',1) |
+--------------+
| a            |
| bbbbbbbbbb   |
+--------------+
```

6.4.2. サブクエリ構文

サブクエリは、別のステートメントに含まれた SELECT ステートメントです。 次に例を示します。

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

上の例で、 SELECT * FROM t1 ... は外側のクエリ（または外側のステートメント）であり、(SELECT column1 FROM t2) はサブクエリです。 いわば、サブクエリは外側のクエリ内にネストされていることになります。実際のところ、サブクエリ内にさらにサブクエリをネストし、クエリを何重にも入れ子化することができます。 サブクエリは必ずかっこで囲む必要があります。

MySQL バージョン 4.1 以降では、標準 SQL で規定されているサブクエリのすべての形式と操作に加え、MySQL 固有のいくつかの機能をサポートしています。サブクエリを使用する主な利点は次のとおりです。

- クエリの各部分が互いに分離されるように、クエリを記述することができる。
- 複雑な結合や集合処理を必要とする操作を、それらの処理を行わずに実行できる。
- （多くの人々の意見として）判読しやすい。 事実、初期の SQL が ``構造化問い合わせ言語" と呼ばれるようになったきっかけは、サブクエリにある。

バージョン 4.0 より前の MySQL では、サブクエリの使用を避けたり、その使用に対処する必要がありましたが、これからコードを書き始める人々は、サブクエリがツールキットの非常に便利な要素であることがわかるはずです。

次のステートメント例は、標準 SQL で規定され、MySQL でサポートしているサブクエリ構文の重要な点を示したものです。

```
DELETE FROM t1
WHERE s11 > ANY
 (SELECT COUNT(*) /* no hint */ FROM t2
 WHERE NOT EXISTS
  (SELECT * FROM t3
   WHERE ROW(5*t2.s1,77)=
    (SELECT 50,11*s1 FROM t4 UNION SELECT 50,77 FROM
     (SELECT * FROM t5) AS t5)));
```

4.1 より前のバージョンの MySQL については、結合などの方法によって、ほとんどのサブクエリを記述し直すことができます。 See 項6.4.2.11.「初期の MySQL バージョンに合わせたサブクエリの書き換え」。

6.4.2.1. スカラオペランドとしてのサブクエリ

最も単純な形式 (スカラ副問い合わせ) では、サブクエリは単純なオペランドです (スカラ副問い合わせと対立するものとして行またはテーブル副問い合わせがあります。これらについては後述します)。したがって、サブクエリは、正しいカラム値やリテラルが使用されているところならどこでも使用でき、また、他のすべてのオペランドと同じように、データ型、長さ、値が NULL になる場合があるかどうかの区別などの特性を持ちます。 次に例を示します。

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5) NOT NULL);
SELECT (SELECT s2 FROM t1);
```

上の SELECT 内のサブクエリのデータ型は CHAR で、長さは 5 です。また、CREATE TABLE 時において有効なデフォルトのキャラクタセットと照合順序を持っており、カラム値が NULL の場合があることが示されています。事実、ほぼすべてのサブクエリは NULL になる場合があります。これは、例にあるようにテーブルが空の場合、サブクエリの値が NULL になるためです。 次に示す、わずかな制約があります。

- サブクエリの外側のステートメントとしては、SELECT、INSERT、UPDATE、DELETE、SET、DO のいずれかを使用できる。
- サブクエリには、通常の SELECT に組み込める任意のキーワードまたは節 (DISTINCT、GROUP BY、ORDER BY、LIMIT、結合、ヒント、UNION 構造、コメント、関数など) を組み込める。

そのため、以降のセクションでは、どちらかというと簡素な構造 (SELECT column1 FROM t1) を含む例を示していますが、実際のコードにはもっと多様で複雑な構造が含まれると想像してください。

たとえば、次のように 2 つのテーブルを作るとします。

```
CREATE TABLE t1 (s1 INT);
INSERT INTO t1 VALUES (1);
CREATE TABLE t2 (s1 INT);
INSERT INTO t2 VALUES (2);
```

その後、SELECT を実行します。

```
SELECT (SELECT s1 FROM t2) FROM t1;
```

結果は 2 になります。なぜなら、t2 には、カラム s1 を持つレコードがあり、s1 の値は 2 であるためです。

サブクエリは式の一部にすることができます。関数のオペランドにするときは、かっこを忘れないでください。 次に例を示します。

```
SELECT UPPER((SELECT s1 FROM t1)) FROM t2;
```

6.4.2.2. サブクエリを使用した比較

サブクエリの最も一般的は使用法は、次の形式のものです。

```
<non-subquery operand> <comparison operator> (<subquery>)
```

<comparison operator> には、次のいずれかを指定します。

```
= > < >= <= <>
```

次に例を示します。

```
... 'a' = (SELECT column1 FROM t1)
```

以前は、サブクエリを配置する正しい位置は比較の右側に限られていました。一部の古い DBMS で、今でもこの制約が適用されていることがあります。

次に示すのは、結合では処理できない、一般的な形式のサブクエリを使用した比較の例です。この例では、テーブル t2 の最大値と等しいすべての値がテーブル t1 で検索されます。

```
SELECT column1 FROM t1
    WHERE column1 = (SELECT MAX(column2) FROM t2);
```

もう 1 つ例を示します。テーブルの 1 つに対する集約が含まれているため、この場合も結合では処理できません。この例では、2 回出現する値を持つすべてのレコードがテーブル t1 で検索されます。

```
SELECT * FROM t1
    WHERE 2 = (SELECT COUNT(column1) FROM t1);
```

6.4.2.3. ANY、IN、SOME とともに使用したサブクエリ

構文

```
<operand> <comparison operator> ANY (<subquery>)
<operand> IN (<subquery>)
<operand> <comparison operator> SOME (<subquery>)
```

ANY という語は比較演算子の後に指定するもので、``サブクエリが返すレコードの ANY (いずれか) に対して比較が TRUE の場合 TRUE を返す" ことを表します。 次に例を示します。

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

テーブル t1 に {10} という値を含むレコードがあるとします。 テーブル t2 に含まれている値が {21,14,7} の場合、この式は TRUE になります。なぜなら t2 に、10 よりも小さい 7 という値が存在するからです。 テーブル t2 に含まれている値が {20,10} の場合や、テーブル t2 が空の場合、この式は FALSE になります。 テーブル t2 に含まれている値が {NULL,NULL,NULL} の場合、この式は UNKNOWN になります。

IN という語は = ANY のエイリアスです。したがって、次の 2 つのステートメントは同じです。

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 IN    (SELECT s1 FROM t2);
```

SOME という語は ANY のエイリアスです。したがって、次の 2 つのステートメントは同じです。

```
SELECT s1 FROM t1 WHERE s1 <> ANY  (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);
```

SOME はめったに使用されませんが、上の例では、この語がどのような場合に役立つかを示しています。``a is not equal to any b" という英語のフレーズを、ほとんどの人は ``a と等しい b はまったく存在しない" という意味に受け取ります。しかし、SQL 構文では意味が異なります。ANY の代わりに <> SOME を使用することによって、このクエリの本当の意味を誰もが確実に理解できるようにすることができます。

6.4.2.4. ALL とともに使用したサブクエリ

構文

```
<operand> <comparison operator> ALL (<subquery>)
```

ALL という語は比較演算子の後に指定するもので、``サブクエリが返すレコードの ALL (すべて) に対して比較が TRUE の場合 TRUE を返す" ことを表します。 次に例を示します。

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

テーブル t1 に {10} という値を含むレコードがあるとします。 テーブル t2 に含まれている値が {-5,0,+5} の場合、この式は TRUE になります。なぜなら t2 に含まれている値はすべて 10 より小さいからです。 テーブル t2 に含まれている値が {12,6,NULL,-100} の場合、この式は FALSE になります。なぜなら t2 に、10 よりも大きい 12 という値が 1 つ存在するからです。 テーブル t2 に含まれている値が {0,NULL,1} の場合、この式は UNKNOWN になります。

最後に、テーブル t2 が空の場合、結果は TRUE になります。 この場合、結果は UNKNOWN になると考えたくなりますが、実際には TRUE です。わかりにくいかもしれませんが、

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT s1 FROM t2);
```

上の例の場合、テーブル t2 が空なら TRUE になりますが、

```
SELECT * FROM t1 WHERE 1 > (SELECT s1 FROM t2);
```

上の例の場合、テーブル t2 が空なら結果は UNKNOWN になります。

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT MAX(s1) FROM t2);
```

さらに、上の例の場合も、テーブル t2 が空なら結果は UNKNOWN になります。 一般に、NULL 値を持つテーブルと空の

テーブルは、境目のケースです。サブクエリのコードを書くときには、この 2 つの可能性を常に念頭に置くようにしてください。

6.4.2.5. 相関副問い合わせ

相関副問い合わせとは、その中に含まれるカラムの参照が外側のクエリにも含まれているサブクエリのことを言います。次に例を示します。

```
SELECT * FROM t1 WHERE column1 = ANY
    (SELECT column1 FROM t2 WHERE t2.column2 = t1.column2);
```

注意: 上の例のサブクエリには、t1 の参照が含まれていますが、このサブクエリの FROM 節ではテーブル t1 については言及されていません。そのため、MySQL によってサブクエリの外側が調べられ、外側のクエリで t1 が検出されます。

テーブル t1 に、column1 = 5 で、 column2 = 6 であるレコードが含まれており、テーブル t2 に、column1 = 5 で、column2 = 7 であるレコードが含まれているとします。... WHERE column1 = ANY (SELECT column1 FROM t2) という簡単な式の結果は TRUE になりますが、この例の場合、サブクエリ内の WHERE 節が FALSE になるので (7 <> 5 であるため)、サブクエリ全体は FALSE になります。

スコープの規則:MySQL では、内側から外側に向かって評価を行います。 次に例を示します。

```
SELECT column1 FROM t1 AS x
 WHERE x.column1 = (SELECT column1 FROM t2 AS x
   WHERE x.column1 = (SELECT column1 FROM t3 WHERE x.column2 = t3.column1));
```

上の例では、SELECT column1 FROM t2 AS x ... によって t2 の名前が変更されることから、x.column2 はテーブル t2 のカラムでなければなりません。SELECT column1 FROM t1 ... はずっと離れた外側のクエリなので、x.column2 はテーブル t1 のカラムではありません。

HAVING 節や ORDER BY 節のサブクエリについても、MySQL では、外側の選択リストに指定されたカラム名が調べられます。

MySQL では、相関クエリを使用しないよう非公式に推奨しています。相関クエリは構成が複雑化するだけでなく、実行にもより多くの時間がかかります。

6.4.2.6. EXISTS と NOT EXISTS

サブクエリで値がまったく返らない場合、EXISTS <subquery> は TRUE になり、NOT EXISTS <subquery> は FALSE になります。 次に例を示します。

```
SELECT column1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

従来、EXISTS サブクエリは SELECT * で開始しますが、SELECT 5 や SELECT column1 などのように、どのように開始してもかまいません。このようなサブクエリ内の SELECT リストは MySQL で無視されるため、重要ではありません。

上の例で、t2 にレコードが含まれていれば、たとえそのレコードに含まれている値が NULL であっても、EXISTS 条件は TRUE になります。しかし、この例は現実的ではありません。なぜなら、ほとんどの場合、[NOT] EXISTS サブクエリには相関が含まれるためです。 次に、より現実に即した例をいくつか示します。

例: 1 つ以上の都市に存在するのは、どの種類の店舗ですか。

```
SELECT DISTINCT store_type FROM Stores
```

```
 WHERE EXISTS (SELECT * FROM Cities_Stores
            WHERE Cities_Stores.store_type = Stores.store_type);
```

例: どの都市にも存在しないのは、どの種類の店舗ですか。

```
SELECT DISTINCT store_type FROM Stores
 WHERE NOT EXISTS (SELECT * FROM Cities_Stores
                WHERE Cities_Stores.store_type = Stores.store_type);
```

例: すべての都市にも存在するのは、どの種類の店舗ですか。

```
SELECT DISTINCT store_type FROM Stores S1
 WHERE NOT EXISTS (
   SELECT * FROM Cities WHERE NOT EXISTS (
     SELECT * FROM Cities_Stores
      WHERE Cities_Stores.city = Cities.city
      AND Cities_Stores.store_type = Stores.store_type));
```

最後の例は、二重にネストされた NOT EXISTS クエリです。このクエリでは、NOT EXISTS 内にさらに NOT EXISTS 節が含まれています。形式的には、このクエリは ``Stores に存在しない店舗が含まれる都市は存在するか" という疑問に答えるものですが、むしろ、ネストされている NOT EXISTS によって、``x はすべての y に対して TRUE か" という疑問の答えが得られると言ったほうが簡単です。

6.4.2.7. 行副問い合わせ

ここまでの内容は、スカラ副問い合わせ -- つまり、単一のカラム値を返す -- サブクエリについてのものでした。行副問い合わせとは、単一のレコード値を返す異型サブクエリです。したがって、複数のカラム値を返すことがあります。 次に例を 2 つ示します。

```
SELECT * FROM t1 WHERE (1,2) = (SELECT column1, column2 FROM t2);
SELECT * FROM t1 WHERE ROW(1,2) = (SELECT column1, column2 FROM t2);
```

column1 = 1 で、column2 = 2 であるレコードがテーブル t2 に含まれている場合、これらのクエリはどちらも TRUE になります。

式 (1,2) はコンストラクタと呼ばれることもあり、他のコンテキストでも正式に使用できます。次に、例を示します。

```
SELECT * FROM t1 WHERE (column1,column2) = (1,1);
```

上のステートメントは次のものと同じです。

```
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

しかし、通常、コンストラクタは、複数のカラムを返すサブクエリを使った比較で使用されます。たとえば、次のクエリは、``テーブル t2 に重複するレコードを持つすべてのレコードを t1 で検索する" という要求に対応します。

```
SELECT column1,column2,column3
    FROM t1
    WHERE (column1,column2,column3) IN
        (SELECT column1,column2,column3 FROM t2);
```

6.4.2.8. FROM 節のサブクエリ

サブクエリは SELECT ステートメントの FROM 節で正式に使用できます。 実際の構文は次のとおりです。

```
SELECT ... FROM (<subquery>) AS <name> ...
```

FROM 節のテーブルには名前が必要なため、AS <name> 節は必ず指定する必要があります。<subquery> 選択リスト内のカラムはいずれも一意な名前を持っていなければなりません。この構文については、このマニュアルの別の場所で、``派生テーブル" という用語に関連して説明しています。

たとえば、次のようなテーブルがあると想定してください。

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5), s3 FLOAT);
```

このテーブルに基づいて、FROM 節のサブクエリ機能を使用するには、次のように記述します。

```
INSERT INTO t1 VALUES (1,'1',1.0);
INSERT INTO t1 VALUES (2,'2',2.0);
SELECT sb1,sb2,sb3
    FROM (SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM t1) AS sb
    WHERE sb1 > 1;
```

結果: 2、'2'、4.0

もう 1 つ例を示します。グループ化されたテーブルの合計の平均値を確認する必要があるとします。この場合、次のクエリは機能しません。

```
SELECT AVG(SUM(column1)) FROM t1 GROUP BY column1;
```

しかし、次のクエリを使用すれば、必要な情報が得られます。

```
SELECT AVG(sum_column1)
    FROM (SELECT SUM(column1) AS sum_column1
        FROM t1 GROUP BY column1) AS t1;
```

注意: サブクエリで使用されているカラム名 (sum_column1) は外側のクエリで認識されます。

現在のところ、FROM 節のサブクエリを相関クエリにすることはできません。

6.4.2.9. サブクエリのエラー

サブクエリだけに適用される新しい戻りエラーがいくつかあります。 このセクションでは、それらのエラーを一つにまとめ、重要な点についていくつか説明します。

- ```
  ERROR 1235 (ER_NOT_SUPPORTED_YET)
  SQLSTATE = 42000
  Message = "This version of MySQL doesn't yet support
  'LIMIT & IN/ALL/ANY/SOME subquery'"
  ```

  これは、次のステートメントが機能しないことを意味する。
  ```

```
SELECT * FROM t1 WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1)
```

ただし、これは、一部の初期のバージョン（MySQL 4.1.1 など）にのみ適用される。

- ```
  ERROR 1240 (ER_CARDINALITY_COL)
  SQLSTATE = 21000
  Message = "Operand should contain 1 column(s)"
  ```

  このエラーは次のような場合に発生する。

  ```
  SELECT (SELECT column1, column2 FROM t2) FROM t1;
  ```

  比較を目的とするものならば、複数のカラムを返すサブクエリを使用できる。 See 項6.4.2.7.「行副問い合わせ」。しかし、他のコンテキストでは、サブクエリはスカラオペランドでなければならない。

- ```
  ERROR 1241 (ER_SUBSELECT_NO_1_ROW)
  SQLSTATE = 21000
  Message = "Subquery returns more than 1 row"
  ```

  このエラーは次のような場合に発生する。

  ```
  SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
  ```

  ただし、これは t2 に複数のレコードが存在する場合に限られる。たとえば、このエラーは長い間使用されているコードで発生することがある。つまり、サブクエリで返すことができるレコード数に影響するような変更を誰かが行ったような場合である。1 つに限らず任意の数のレコードをオブジェクトで検出するようにするときの正しいステートメントは、次のようになる。

  ```
  SELECT * FROM t1 WHERE column1 = ANY (SELECT column1 FROM t2);
  ```

- ```
  Error 1093 (ER_UPDATE_TABLE_USED)
  SQLSTATE = HY000
  Message = "You can't specify target table 'x' for update in FROM clause"
  ```

  このエラーは次のような場合に発生する。

  ```
  UPDATE t1 SET column2 = (SELECT MAX(column1) FROM t1);
  ```

サブクエリは SELECT ステートメントと同じく、UPDATE ステートメントや DELETE ステートメントでも正式に使用できるので、UPDATE ステートメント内の割り当てにサブクエリを使用しても問題ありません。 しかし、サブクエリの FROM 節と更新対象の両方に同じテーブル（この場合、テーブル t1）を使用することはできません。

通常、サブクエリがエラーになると、ステートメント全体がエラーになります。

6.4.2.10. サブクエリの最適化

開発は現在進行形で進められているため、長期間有効な最適化のヒントというのはありません。ここでは、いくつかのテ
  ```

クニックについて説明します。

- サブクエリのレコード数やレコードの順序に影響するサブクエリ節を使用する。次はその例。

```
SELECT * FROM t1 WHERE t1.column1 IN
 (SELECT column1 FROM t2 ORDER BY column1);
SELECT * FROM t1 WHERE t1.column1 IN
 (SELECT DISTINCT column1 FROM t2);
SELECT * FROM t1 WHERE EXISTS
 (SELECT * FROM t2 LIMIT 1);
```

- 結合をサブクエリに置き換える。次はその例。

```
SELECT DISTINCT column1 FROM t1 WHERE t1.column1 IN (
 SELECT column1 FROM t2);
```

上のステートメントは次のステートメントの代わりに使用できる。

```
SELECT DISTINCT t1.column1 FROM t1, t2
 WHERE t1.column1 = t2.column1;
```

- サブクエリの外側から内側へ節を移動する。次はその例。

```
SELECT * FROM t1
 WHERE s1 IN (SELECT s1 FROM t1 UNION ALL SELECT s1 FROM t2);
```

上のステートメントは次のステートメントの代わりに使用できる。

```
SELECT * FROM t1
 WHERE s1 IN (SELECT s1 FROM t1) OR s1 IN (SELECT s1 FROM t2);
```

もう 1 つの例。

```
SELECT (SELECT column1 + 5 FROM t1) FROM t2;
```

上のステートメントは次のステートメントの代わりに使用できる。

```
SELECT (SELECT column1 FROM t1) + 5 FROM t2;
```

- 相関副問い合わせの代わりに行副問い合わせを使用する。次はその例。

```
SELECT * FROM t1
 WHERE (column1,column2) IN (SELECT column1,column2 FROM t2);
```

上のステートメントは次のステートメントの代わりに使用できる。

```
SELECT * FROM t1
 WHERE EXISTS (SELECT * FROM t2 WHERE t2.column1=t1.column1
 AND t2.column2=t1.column2);
```

- a <> ALL (...) ではなく、NOT (a = ANY (...)) を使用する。

- x=1 OR x=2 ではなく x = ANY (table containing {1,2}) を使用する。
- EXISTS ではなく = ANY を使用する。

上記のテクニックを使用すると、プログラムの実行が速くなったり、遅くなったりする場合があります。MySQL の BENCHMARK() 関数などの機能を使用すると、それぞれの状況において何が適切か、調べることができます。旧バージョンとの互換性を確保する場合を除いて、結合への変換についてはあまり気にしないでください。

MySQL 自体が行う最適化として次のものがあります。

1. 非相関副問い合わせは 1 度しか実行されない (EXPLAIN を使用して、指定されたサブクエリが実際に非相関副問い合わせかどうか確認できる)
2. IN/ALL/ANY/SOME サブクエリは、サブクエリ内の選択リストカラムがインデックス付きである可能性があるため、書き直される。
3. 次の形式のサブクエリはインデックスルックアップ関数と置き換えられる。

   ```
   ... IN (SELECT indexed_column FROM single_table ...)
   ```

   インデックスルックアップ関数は、EXPLAIN では特殊な結合型として記述される。
4. 次の形式の式は、MIN または MAX を含む式で拡張される (NULL 値や空のセットが含まれている場合を除く)。

   ```
   value {ALL|ANY|SOME} {> | < | >= | <=} (non-correlated subquery)
   ```

   例

   ```
   WHERE 5 > ALL (SELECT x FROM t)
   ```

   上の式は次のように処理される。

   ```
   WHERE 5 > (SELECT MAX(x) FROM t)
   ```

MySQL Internals Manual (MySQL 内部情報マニュアル) に ``How MySQL Transforms Subqueries (MySQL によるサブクエリの変換)" という章があります。このマニュアルを参照するには、MySQL のソースパッケージをダウンロードして、internals.texi というファイルを探してください。

6.4.2.11. 初期の MySQL バージョンに合わせたサブクエリの書き換え

バージョン 4.0 までは、ネストされたクエリのサポートは、INSERT ... SELECT ... 形式と REPLACE ... SELECT ... 形式だけに限定されています。 その他のコンテキストでは、IN() 構造を使用することができます。

サブクエリを含むクエリは、多くの場合、サブクエリなしのクエリに書き換えることができます。

```
SELECT * FROM t1 WHERE id IN (SELECT id FROM t2);
```

上のクエリは次のように書き換えることができます。

```
SELECT t1.* FROM t1,t2 WHERE t1.id=t2.id;
```

次のクエリの場合、

```
SELECT * FROM t1 WHERE id NOT IN (SELECT id FROM t2);
SELECT * FROM t1 WHERE NOT EXISTS (SELECT id FROM t2 WHERE t1.id=t2.id);
```

次のように書き換えることができます。

```
SELECT table1.* FROM table1 LEFT JOIN table2 ON table1.id=table2.id
                              WHERE table2.id IS NULL;
```

LEFT [OUTER] JOIN は同等のサブクエリよりも迅速に実行されることがあります。なぜなら、LEFT [OUTER] JOIN のほうがサーバによる最適化がより適切に行われることがあるためです。これは、MySQL Server だけでなく、サーバー一般について言えます。 SQL-92 より前は外部結合が存在しなかったため、それまで、サブクエリは特定の処理を行う唯一の方法でした。しかし現在、MySQL サーバを始めとする最新のデータベースシステムでは、さまざまな型の外部結合を提供しています。

より複雑なサブクエリでは、多くの場合、サブクエリを保持するためのテンポラリテーブルを作成することができます。しかし、一部のケースでは、この方法は有効ではありません。このようなケースの多くは、DELETE ステートメントで発生します。この場合、標準 SQL では、結合をサポートしていません (サブクエリ内を除く) 。この状況には、次の 3 つの方法のいずれかで対処することができます。

- 第 1 の選択肢: MySQL バージョン 4.1 にアップグレードする。
- 第 2 の選択肢: 手続き型プログラミング言語 (Perl や PHP など) を使用し、SELECT クエリを送信して削除対象のレコードの主キーを取得し、その後、それらの値を使用して DELETE ステートメント (DELETE FROM ... WHERE ... IN (key1, key2, ...)) を構築する。
- 第 3 の選択肢: 対話式 SQL で、MySQL の拡張 CONCAT() を使用して(標準の || 演算子の代わりに) 、DELETE ステートメントのセットを自動で構築する。 次はその例。

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

  このクエリをスクリプトファイルに保管することによって、ファイルからの入力を mysql コマンドラインインタープリタにリダイレクトし、さらにインタープリタの 2 つ目のインスタンスに出力を渡すことができる。

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

MySQL サーバ 4.0 では、複数テーブルの DELETE ステートメントをサポートしています。このようなステートメントを使用すると、1 つのテーブルのデータだけでなく、複数のテーブルのデータに基づくレコードを一度に効率的に削除することができます。 バージョン 4.0 以降では、複数テーブルの UPDATE ステートメントもサポートしています。

6.4.3. INSERT 構文

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    VALUES ((expression | DEFAULT),...),(...),...
```

```
    [ ON DUPLICATE KEY UPDATE col_name=expression, ... ]
または  INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
または  INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] tbl_name
    SET col_name=(expression | DEFAULT), ...
    [ ON DUPLICATE KEY UPDATE col_name=expression, ... ]
```

INSERT ステートメントでは、既存のテーブルに新しいレコードが挿入されます。INSERT ... VALUES 形式の INSERT では、明示的に指定した値に基づくレコードが挿入されます。INSERT ... SELECT 形式の INSERT では、別の 1 つまたは複数のテーブルから選択されたレコードが挿入されます。複数の値のリストを持つ INSERT ... VALUES 形式は、MySQL バージョン 3.22.5 以降でサポートしています。col_name=expression 構文は MySQL バージョン 3.22.10 以降でサポートしています。

tbl_name には、レコードを挿入するテーブルを指定します。カラム名のリストまたは SET 節には、そのステートメントで値を指定する対象のカラムを指定します。

- INSERT ... VALUES または INSERT ... SELECT でカラムリストを指定しない場合、VALUES() リストまたは SELECT で、テーブルのすべてのカラムの値を提供する必要がある。テーブル内のカラムの順序がわからない場合は、DESCRIBE tbl_name を使用して順序を調べる。
- 値が明示的に指定されていないカラムは、そのカラムのデフォルトに設定される。たとえば、カラムリストでテーブルのすべのカラムが指定されていない場合、指定されていないカラムはそのデフォルト値に設定される。デフォルト値の割り当てについては、項6.5.3.「CREATE TABLE 構文」 で説明している。

  キーワード DEFAULT を使用して、カラムにデフォルト値を設定することもできる（MySQL 4.0.3 の新機能）。この場合、不完全な VALUES() リスト（テーブル内のすべてのカラムの各値を含んでいないリスト）を記述しなくてすむため、一部のわずかなカラムを除いたすべてのカラムに値を割り当てる INSERT ステートメントを書きやすくなる。 このキーワードを使用しない場合、VALUES() リストの各値に対応するカラム名のリストを記述しなければならない。

  MySQL では、常にすべてのフィールドにデフォルト値がある。デフォルト値の存在は、MySQL がトランザクションテーブルと非トランザクションテーブルのどちらも処理できるようにするための必要条件となっている。

  MySQL では、フィールド内容のチェックはデータベースサーバではなくアプリケーション側で行う、という見解に立っている。

- expression では、値リストに先に設定された任意のカラムを参照することができる。たとえば、次のように記述できる。

  ```
  mysql> INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
  ```

  しかし、次のようには記述できない。

  ```
  mysql> INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
  ```

- キーワード DELAYED が指定されていると、サーバはレコードをバッファに挿入する。その後、INSERT DELAYED ステートメントを発行したクライアントは処理を続行することができる。テーブルが使用されていると、サーバはレコードを保持する。テーブルが解放されると、サーバはレコードの挿入を開始し、そのテーブルに対する新しい読み取り要

求がないか定期的にチェックする。新しい読み取り要求があると、そのテーブルが再び解放されるまで、遅延されたレコードのキューの処理は中断される。

- キーワード LOW_PRIORITY が指定されている場合、他のクライアントによるそのテーブルからの読み取りがなくなるまで、INSERT の実行は遅らされる。これには、既存のクライアントの読み取り中、および INSERT LOW_PRIORITY ステートメントの待機中に読み取りを開始した他のクライアントも含まれる。したがって、読み取り過多な環境の場合、INSERT LOW_PRIORITY ステートメントを発行したクライアントは長時間（または永続的に）待機させられることがある（それに対して INSERT DELAYED では、クライアントは直ちに処理を続行することができる）。See 項6.4.3.2.「INSERT DELAYED 構文」注意: LOW_PRIORITY では同時挿入が無効になるため、このオプションは通常 MyISAM テーブルに対しては使用しない。 See 項7.1.「MyISAM テーブル」。

- 多くのレコードの INSERT でキーワード IGNORE が指定されていると、テーブルの既存の PRIMARY または UNIQUE キーと重複するレコードはすべて無視され、挿入されない。IGNORE が指定されていない場合に既存のキー値を重複して持つレコードがあると、挿入処理が中断される。C API 関数 mysql_info() では、テーブルに挿入されたレコード数を調べることができる。

- ON DUPLICATE KEY UPDATE 節（MySQL 4.1.0 の新機能）が指定されている場合に、PRIMARY または UNIQUE キーでの重複値の生成を招くレコードが挿入されると、古いレコードの UPDATE が実行される。次に例を示す。

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3)
   -> ON DUPLICATE KEY UPDATE c=c+1;
```

このコマンドでは、a が UNIQUE として宣言されていて、かつすでに 1 度、値 1 を保持している場合、次のコマンドと同じになる。

```
mysql> UPDATE table SET c=c+1 WHERE a=1;
```

注意: カラム b も一意である場合、UPDATE コマンドは次のようになる。

```
mysql> UPDATE table SET c=c+1 WHERE a=1 OR b=2 LIMIT 1;
```

a=1 OR b=2 が複数のレコードに一致する場合、1 つのレコードだけが更新される。通常、複数の UNIQUE キーを持つテーブルに対しては ON DUPLICATE KEY 節を使用しないようにする。

MySQL 4.1.1 以降では、関数 VALUES(col_name) を使用して、INSERT ... UPDATE コマンドの INSERT 部分のカラム値を参照できる。これは、重複キーのコンフリクトがない場合に挿入される値である。この関数は複数行の挿入時に特に役立つ。当然、VALUES() 関数は INSERT ... UPDATE コマンドでのみ意味を持ち、それ以外のコマンドで使用した場合は NULL が返される。

次に例を示す。

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
   -> ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

上のコマンドは次のコマンドと同じである。

```
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;
```

ON DUPLICATE KEY UPDATE が指定されている場合、DELAYED オプションは無視される。

- MySQL のコンフィギャ時に DONT_USE_DEFAULT_FIELDS オプションが指定されている場合、NULL 以外の値を必要とするすべてのカラムに明示的に値を指定しないと、INSERT ステートメントでエラーが生成される。 See 項2.3.3.「一般的な configure オプション」。
- AUTO_INCREMENT カラムに使用されている値は、mysql_insert_id 関数を使用して確認することができる。 See 項11.1.3.32.「mysql_insert_id()」。

INSERT ... SELECT または INSERT ... VALUES ステートメントを複数の値のリスト付きで使用する場合は、C API 関数 mysql_info() を使用してクエリに関する情報を取得することができます。情報文字列の形式は次のとおりです。

```
Records: 100 Duplicates: 0 Warnings: 0
```

Duplicates は、特定の既存のユニークインデックス値と重複することになるため挿入されなかったレコード数を表します。 Warnings は、何らかの問題があるカラム値を挿入しようとした回数を表します。Warnings (警告) は次のいずれかの条件の下に発生します。

- NOT NULL として宣言されているカラムへの NULL の挿入。 この場合、カラムにはそのカラム型に対応するデフォルト値が設定される。 つまり、数値型のカラムでは 0、文字列型のカラムでは空の文字列 ('') 、日付と時刻型のカラムでは ``ゼロ" 値が適用される。
- 数値型のカラムへの、そのカラムの値の範囲を超える値の設定。 この場合、値は範囲の最大値または最小値に切り落とされる。
- 数値型のカラムへの '10.34 a' などの値の設定。この場合、後続のガーベジは除去され、残りの数値の部分だけが挿入される。値が数値として意味をなさない場合、カラムの値は 0 に設定される。
- CHAR 型、VARCHAR 型、TEXT 型、または BLOB 型のカラムへの、そのカラムの最大長を超える文字列の挿入。この場合、値はカラムの最大長に合わせて切り捨てられる。
- 日付または時刻型のカラムへの、そのカラム型に不適切な値の挿入。この場合、カラムの値はその型に対応するゼロ値に設定される。

6.4.3.1. INSERT ... SELECT 構文

```
INSERT [LOW_PRIORITY] [IGNORE] [INTO] tbl_name [(column list)] SELECT ...
```

INSERT ... SELECT ステートメントでは、1 つまたは複数のテーブルの数多くのレコードを別の 1 つのテーブルにすばやく挿入することができます。

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

INSERT ... SELECT ステートメントでは、以下の条件が適用されます。

- MySQL 4.0.1 より前のバージョンでは、INSERT ... SELECT は暗黙的に IGNORE モードで動作する。MySQL 4.0.1 以

降では、重複キー違反を引き起こすレコードを無視するには、IGNORE を明示的に指定しなければならない。

- MySQL 4.0.14 より前のバージョンでは、INSERT ステートメントの対象テーブルをクエリの SELECT 部分の FROM 節に示すことはできない。 この制約は 4.0.14 でなくなった。
- AUTO_INCREMENT カラムは通常どおり機能する。
- C プログラムでは、C API 関数 mysql_info() を使用してクエリに関する情報を取得できる。 See 項6.4.3.「INSERT 構文」。
- バイナリログを使用して元のテーブルを確実に再作成できるようにするため、MySQL では、INSERT ... SELECT 実行中の同時挿入は行えない。

以前のレコードを上書きするには、INSERT の代わりに REPLACE を使用します。以前のレコードの値と重複するユニークキー値を持つ新しいレコードの処理に関して、REPLACE は INSERT IGNORE に対立する働きをします。つまり、新しいレコードが廃棄されるのではなく、新しいレコードによって以前のレコードが置き換えられます。

6.4.3.2. INSERT DELAYED 構文

```
INSERT DELAYED ...
```

INSERT の完了を待てないクライアントがある場合、MySQL 固有のオプションである DELAYED を指定した INSERT ステートメントが非常に役立ちます。このようなクライアントの問題は、MySQL を使用してログを記録する一方で、完了までに時間がかかる SELECT や UPDATE ステートメントを定期的に実行している場合によく起こります。DELAYED は、MySQL バージョン 3.22.15 で導入された、SQL-92 に対する MySQL の拡張です。

INSERT DELAYED は、ISAM および MyISAM テーブルに対してのみ作用します。注意: MyISAM テーブルでは、データファイルの中央に空きブロックがない場合、同時 SELECT と同時 INSERT が可能なため、MyISAM テーブルで INSERT DELAYED を使用する必要はほとんどありません。 See 項7.1.「MyISAM テーブル」。

INSERT DELAYED を使用すると、クライアントは直ちに処理の続行を許可されます。そして、テーブルが別のスレッドによって使用されていなければ、レコードが挿入されます。

INSERT DELAYED のもう 1 つの主な利点は、多くのクライアントによる挿入が 1 つにまとめられて、1 ブロックに書き込まれることです。この場合、数多くの挿入を個別に行うより、処理速度がはるかに速くなります。

注意: 現在のところ、キューに入れられたレコードは、テーブルに挿入されるまでは単にメモリに格納されているに過ぎません。つまり、mysqld を kill -9 によって強制終了した場合や、mysqld が予期しない状況で突然終了した場合、ディスクに書き込まれていないキューの中のレコードはすべて消失します。

以下に、INSERT または REPLACE ステートメントで DELAYED オプションを指定した場合の処理について詳しく説明します。この説明中、``スレッド" は INSERT DELAYED コマンドを受け取ったスレッドを指し、``ハンドラ" は個々のテーブルの INSERT DELAYED ステートをすべて処理するスレッドを指します。

- スレッドがいずれかのテーブルに対する DELAYED ステートメントを実行すると、そのテーブルのすべての DELAYED ステートメントを処理するハンドラスレッドが作成される (このようなハンドラがすでに存在しない場合) 。
- スレッドは、ハンドラが DELAYED ロックをすでに取得しているかチェックし、まだの場合はロックを取得するようハンドラに指示する。DELAYED ロックは、他のスレッドがそのテーブルに対する READ または WRITE ロックを持っ

ている場合でも取得できる。ただし、その場合ハンドラは、必ず最新のテーブル構造が得られるよう、すべての ALTER TABLE ロックまたは FLUSH TABLES が完了するまで待機する。

- スレッドは INSERT ステートメントを実行する。ただし、この場合、スレッドはレコードをテーブルに書き込む代わりに、ハンドラスレッドが管理するキューに最後のレコードのコピーを入れる。構文エラーがある場合はスレッドによって検出され、クライアントプログラムに報告される。
- クライアントは重複の数や結果のレコードの AUTO_INCREMENT 値はいずれも報告できない。INSERT は挿入処理が完了する前に戻るため、クライアントはこれらの情報をサーバから取得できない。C API を使用している場合、同じ理由で、mysql_info() 関数でも、意味のある情報は何も返されない。
- バイナリログは、レコードがテーブルに挿入されると、ハンドラスレッドによって更新される。複数のレコードの挿入時では、バイナリログは最初のレコードが挿入された時点で更新される。
- 各 delayed_insert_limit レコードが書き込まれると、そのつど、ハンドラはまだ保留中になっている SELECT ステートメントがないかチェックする。ある場合、ハンドラは処理を続行する前に保留中のステートメントの実行を許可する。
- ハンドラのキューにレコードが何もなくなると、テーブルのロックが解除される。delayed_insert_timeout に指定された秒数が経過する前に、新しい INSERT DELAYED コマンドが渡されないと、ハンドラは終了する。
- delayed_queue_size に指定された数を超えるレコードが特定のハンドラキューにすでに保留になっていると、INSERT DELAYED を要求しているスレッドはキューに空きスペースができるまで待機する。これは、mysqld サーバで遅延メモリキューのメモリをすべて使ってしまわないようにするためである。
- ハンドラスレッドは MySQL プロセスリストの Command カラムに delayed_insert として表示される。FLUSH TABLES コマンドを実行するか、または KILL thread_id を使用して強制終了すると、ハンドラスレッドは強制終了される。しかし、この場合ハンドラスレッドは、キュー内のすべてのレコードをテーブルに格納してから終了する。この間、ハンドラスレッドは別のスレッドからの新しい INSERT コマンドをまったく受け入れない。この後に INSERT DELAYED コマンドを実行すると、新しいハンドラスレッドが作成される。

  注意:これは、すでに実行されている INSERT DELAYED が存在する場合、通常の INSERT コマンドより INSERT DELAYED コマンドの方が優先されることを意味する。他の更新コマンドは、INSERT DELAYED キューが空になるか、誰かがハンドラスレッドを強制終了するか (KILL thread_id を使用して)、または誰かが FLUSH TABLES を実行するまで待機しなければならない。
- 以下のステータス変数では、INSERT DELAYED コマンドの情報が提供される。

変数	意味
Delayed_insert_threads	ハンドラスレッドの数
Delayed_writes	INSERT DELAYED で書き込まれたレコード数
Not_flushed_delayed_rows	書き込み待ちのレコード数

これらの変数を表示するには、SHOW STATUS ステートメントを発行するか、または mysqladmin extended-status コマンドを実行する。

注意: 対象のテーブルが使用中でない場合、INSERT DELAYED は通常の INSERT より遅くなります。また、INSERT DELAYED を使用する各テーブルに対応するスレッドを個別に処理するためにサーバで追加のオーバーヘッドが発生しま

す。そのため、INSERT DELAYED はどうしても使う必要があるときのみ使用してください。

6.4.4. UPDATE 構文

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
    SET col_name1=expr1 [, col_name2=expr2 ...]
    [WHERE where_definition]
    [ORDER BY ...]
    [LIMIT row_count]

または

UPDATE [LOW_PRIORITY] [IGNORE] tbl_name [, tbl_name ...]
    SET col_name1=expr1 [, col_name2=expr2 ...]
    [WHERE where_definition]
```

UPDATE は既存のテーブルレコードのカラムを新しい値で更新します。 SET 節は値の変更対象のカラムと値を示します。WHERE 節がある場合、この節は更新するレコードを示します。更新対象のレコードが指定されていない場合は、すべてのレコードが更新されます。ORDER BY 節が指定されていると、そこに指定された順序でレコードが更新されます。

キーワード LOW_PRIORITY が指定されていると、他のクライアントによるそのテーブルからの読み取りがなくなるまで、UPDATE の実行は遅らされます。

キーワード IGNORE が指定されていると、更新中に重複キーエラーが発生しても更新ステートメントは中断されません。コンフリクトを発生させるレコードは更新されません。

式内の tbl_name からカラムがアクセスされる場合、UPDATE はそのカラムの現在の値を使用します。たとえば次のステートメントで、age カラムの値はこのカラムの現在の値より 1 だけ多い値に設定されます。

```
mysql> UPDATE persondata SET age=age+1;
```

UPDATE は左から右へ評価されます。たとえば、次のステートメントでは、age カラムの値がまず 2 倍にされ、その後、加算されます。

```
mysql> UPDATE persondata SET age=age*2, age=age+1;
```

カラムの値がそのカラムの現在の値に設定される場合、MySQL はそれが現在の値であることを認識し、更新処理を行いません。

UPDATE は実際に変更されたレコード数を返します。 MySQL バージョン 3.22 以降では、C API 関数 mysql_info() を使用すると、更新されたレコード数と UPDATE の実行中に発生した警告の数が返されます。 NOT NULL として宣言されたカラムに対して NULL 値を設定し、このカラムを更新すると、そのカラムにはそのカラム型に対応するデフォルト値が設定され、警告のカウントが加算されます。デフォルト値は数値型では 0 であり、文字列型では空の文字列 ('') であり、日付と時刻型では ``ゼロ'' 値です。

MySQL バージョン 3.23 以降では、LIMIT row_count を使用して UPDATE のスコープを制限することができます。LIMIT 節は次のように機能します。

- MySQL 4.0.13 より前のバージョンでは、LIMIT による制限は影響を受けたレコードに基づいて適用される。 WHERE 節の指定にマッチするレコードが、row_count に指定された数だけ変更されると、ステートメントは直ちに停止する。

- 4.0.13 以降では、LIMIT による制限はマッチしたレコードに基づいて適用される。WHERE 節の指定にマッチするレコードが、row_count に指定された数だけ検出されると、それらのレコードが実際に変更されたかどうかにかかわらず、ステートメントは直ちに終了する。

ORDER BY が使用されている場合（MySQL 4.0.0 以降で使用可能）、レコードは指定された順序で更新されます。ORDER BY 節は、実際には、LIMIT と組み合わせて使用した場合にのみ役立ちます。

MySQL バージョン 4.0.4 以降では、複数のテーブルに対する UPDATE 操作も実行可能です。

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

上の例では、カンマ演算子を使用した内部結合を示していますが、複数テーブルの UPDATE ステートメントでは、LEFT JOIN など、SELECT ステートメントで使用可能な任意の結合型を使用することができます。

注意: 複数テーブルの UPDATE では、ORDER BY と LIMIT はいずれも使用できません。

6.4.5. DELETE 構文

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM table_name
    [WHERE where_definition]
    [ORDER BY ...]
    [LIMIT row_count]

または

DELETE [LOW_PRIORITY] [QUICK] [IGNORE] table_name[.*] [, table_name[.*] ...]
    FROM table-references
    [WHERE where_definition]

または

DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    FROM table_name[.*] [, table_name[.*] ...]
    USING table-references
    [WHERE where_definition]
```

DELETE は、where_definition で指定されている条件にマッチするレコードを、table_name に指定されているテーブルから削除し、削除したレコードの数を返します。

WHERE 節が指定されていない DELETE では、すべてのレコードが削除されます。AUTOCOMMIT モードで実行された場合、WHERE 節なしの DELETE は TRUNCATE と同じように動作します。See 項6.4.6.「TRUNCATE 構文」。MySQL 3.23 では、WHERE 節のない DELETE は影響を受けたレコード数として値ゼロを返します。

すべてのレコードの削除時に、削除されたレコード数を確認する必要がある場合、処理速度が遅くなってもかまわなければ、次の形式の DELETE ステートメントを使用できます。

```
mysql> DELETE FROM table_name WHERE 1>0;
```

注意: この形式の場合、レコードが一度に 1 つずつ削除されるため、WHERE 節のない DELETE FROM table_name よりはるかに処理が遅くなります。

キーワード LOW_PRIORITY が指定されていると、他のクライアントによるそのテーブルからの読み取りがなくなるまで、DELETE の実行は遅らされます。

MyISAM テーブルの場合、QUICK という語が指定されていると、削除時にストレージエンジンでインデックスのリーフをマージしません。そのため、一部の削除操作では、処理が迅速化されます。

削除操作の速度は、項5.2.12.「DELETE クエリの速度」 で説明している要因にも影響されます。

オプション IGNORE が指定されていると、MySQL はレコードの削除中にすべてのエラーを無視します。解析段階で発生したエラーは通常の方法で処理されます。このオプションの使用によって無視されたエラーは警告として返されます。このオプションはバージョン 4.1.1 で導入されました。

MyISAM テーブルの場合、削除されたレコードはリンクされたリストに維持され、元のレコード位置が後続の INSERT 操作で再利用されます。使用されていないスペースを取り戻してファイルのサイズを削減するには、OPTIMIZE TABLE ステートメントまたは myisamchk ユーティリティを使用してテーブルを再編成します。 この場合、OPTIMIZE TABLE の方が簡単ですが、myisamchk の方が処理が迅速です。項4.6.1.「OPTIMIZE TABLE 構文」 および 項4.5.6.10.「テーブルの最適化」 を参照してください。

複数テーブルの削除の最初の形式は、MySQL 4.0.0 以降でサポートしています。 複数テーブルの削除の 2 番目の形式は、MySQL 4.0.2 以降でサポートしています。

複数テーブルの削除では、FROM 節の前、または USING 節の前にリストされたテーブル内の一致するレコードのみが削除されます。そのため、多くのテーブルのレコードを一度に削除できるとともに、検索に使用するその他のテーブルを指定することができます。

テーブル名の後に .* を付けるのは、単に Access との互換性を確保するためです。

```
DELETE t1,t2 FROM t1,t2,t3 WHERE t1.id=t2.id AND t2.id=t3.id

または

DELETE FROM t1,t2 USING t1,t2,t3 WHERE t1.id=t2.id AND t2.id=t3.id
```

これらの例では、テーブル t1 と t2 内の一致するレコードだけが削除されます。

上の例ではカンマ演算子を使用した内部結合を示していますが、複数テーブルの DELETE ステートメントでは、LEFT JOIN など、SELECT ステートメントで使用可能な任意の結合型を使用することができます。

ORDER BY が使用されている場合（MySQL 4.0.0 以降で使用可能）、レコードは指定された順序で削除されます。ORDER BY 節は、実際には、LIMIT と組み合わせて使用した場合にのみ役立ちます。次に例を示します。

```
DELETE FROM somelog
WHERE user = 'jcole'
ORDER BY timestamp
LIMIT 1
```

この場合、 WHERE 節の指定に一致するレコードで最も古いエントリ（timestamp 値に基づく）が削除されます。

DELETE に MySQL 固有のオプションである LIMIT row_count を指定すると、クライアントに制御を戻す前に削除する最大レコード数をサーバに指示することができます。このオプションは、特定の DELETE コマンドで実行にあまり時間がかからないようにする必要があるときに使用できます。この場合、影響を受けたレコードの数が LIMIT 値よりも小さくなるまで、DELETE コマンドを単純に繰り返すことができます。

MySQL 4.0 以降では、DELETE ステートメントに複数のテーブルを指定することによって、複数テーブルにおける特定の条件に応じて、1 つ以上のテーブルからレコードを削除することができます。しかし、複数テーブルの DELETE では、ORDER BY と LIMIT はいずれも使用できません。

6.4.6. TRUNCATE 構文

```
TRUNCATE TABLE table_name
```

3.23 では、TRUNCATE TABLE は COMMIT; DELETE FROM table_name にマップされる。 See 項6.4.5.「DELETE 構文」。

TRUNCATE TABLE は次の点で DELETE FROM ... と異なります。

- 切り捨て操作では、テーブルが破棄され、再作成される。この操作は、レコードを 1 つずつ削除するよりはるかに迅速に処理される。
- 切り捨て操作はトランザクションセーフではない。アクティブなトランザクションやアクティブなテーブルのロックがあると、エラーになる。
- 削除されたレコード数は返されない。
- テーブル定義ファイル table_name.frm が有効である限り、データやインデックスのファイルが破損しても、この方法でテーブルを再作成できる。

TRUNCATE TABLE は Oracle の SQL 拡張です。 このステートメントは MySQL 3.23.28 で追加されました。ただし、3.23.28 〜 3.23.32 では、キーワード TABLE を省略する必要があります。

6.4.7. REPLACE 構文

```
  REPLACE [LOW_PRIORITY | DELAYED]
      [INTO] tbl_name [(col_name,...)]
      VALUES (expression,...),(...),...
または  REPLACE [LOW_PRIORITY | DELAYED]
      [INTO] tbl_name [(col_name,...)]
      SELECT ...
または  REPLACE [LOW_PRIORITY | DELAYED]
      [INTO] tbl_name
      SET col_name=expression, col_name=expression,...
```

REPLACE は INSERT とほぼ同じように動作しますが、唯一異なる点として、UNIQUE インデックスまたは PRIMARY KEY に関して新しいレコードと同じ値がテーブル内の以前のレコードに含まれていると、以前のレコードが削除されてから新しいレコードが挿入されます。 See 項6.4.3.「INSERT 構文」。

つまり、REPLACE ステートメントでは、以前からあるレコードの値にはアクセスできないことになります。一部の初期バージョンの MySQL では、このアクセスが可能のように見えましたが、これはバグであり、修正されました。

REPLACE を使用するためには、対象のテーブルに対する INSERT 権限と DELETE 権限が必要です。

REPLACE コマンドの使用時、以前のレコード 1 つが新しいレコードで置き換えられると、mysql_affected_rows() によって 2 が返されます。これは重複の削除後にレコードが 1 つ挿入されたためです。

これに基づき、影響されたレコードの値が 1 (追加) と 2 (置換) のどちらかチェックすることによって、REPLACE によってレコードが 1 つ追加されたのか、または置き換えられたのかを簡単に判別できます。

注意: テーブルに UNIQUE インデックスまたは PRIMARY KEY が設定されていない場合、REPLACE コマンドを使用しても意味がありません。この場合、新しいレコードが既存のものと重複しているかどうかの判別に使用するインデックスがないため、REPLACE コマンドは INSERT と同じになります。

以下は、使用されるアルゴリズムを詳細に示したものです (このアルゴリズムは LOAD DATA ... REPLACE でも使用されます)。

```
- Insert the row into the table
  - While duplicate key error for primary or unique key
    - Revert changed keys
    - Read conflicting row from the table through the duplicate key value
    - Delete conflicting row
    - Try again to insert the original primary key and unique keys in the tree
```

6.4.8. LOAD DATA INFILE 構文

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name.txt'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [FIELDS
        [TERMINATED BY '\t']
        [[OPTIONALLY] ENCLOSED BY '']
        [ESCAPED BY '\\' ]
    ]
    [LINES
        [STARTING BY '']
        [TERMINATED BY '\n']
    ]
    [IGNORE number LINES]
    [(col_name,...)]
```

LOAD DATA INFILE ステートメントは、テキストファイルからテーブルにレコードを高速で読み取ります。LOCAL キーワードが指定されている場合、このキーワードは接続のクライアント側に関連して解釈されます。LOCAL が指定されていると、クライアントホスト上のクライアントプログラムによってファイルが読み取られ、サーバに送られます。LOCAL が指定されていない場合、ファイルはサーバホスト上に存在しなければならず、サーバによって直接読み取られなければなりません (LOCAL は MySQL バージョン 3.22.6 以降で使用できます)。

セキュリティ上の理由から、サーバに存在するテキストファイルを読み取るときには、そのファイルがデータベースディレクトリに存在するか、または全ユーザがそのファイルを読み取り可能でなければなりません。また、サーバのファイルに対して LOAD DATA INFILE を使用するには、サーバホストでの FILE 権限が必要になります。

See 項4.3.7.「MySQL が提供する権限」。

MySQL 3.23.49 と MySQL 4.0.2 (Windows では 4.0.13) 以降では、LOCAL はサーバとサーバクライアントの両方でこれが有効として設定されている場合にのみ機能します。たとえば、--local-infile=0 を指定して mysqld を起動した場合、LOCAL は機能しません。 See 項4.3.4.「LOAD DATA LOCAL のセキュリティ関連事項」。

キーワード LOW_PRIORITY を指定すると、LOAD DATA ステートメントの実行は、他のクライアントによるテーブルからの読み取りが終了するまで遅らされます。

MyISAM テーブルに対してキーワード CONCURRENT を指定すると、LOAD DATA の実行中に他のスレッドがこのテーブルからデータを取り出すことができます。当然ながら、このオプションの使用は、同時にテーブルを使用している他のスレッドがなくても、LOAD DATA のパフォーマンスに多少影響します。

LOCAL を指定した場合、ファイルの内容を接続によってクライアントからサーバに送らなければならないため、サーバがファイルに直接アクセスする場合よりも処理がやや遅くなります。その反面、ローカルファイルのロードには FILE 権限は必要ありません。

バージョン 3.23.24 より前のバージョンの MySQL を使用している場合は、LOAD DATA INFILE を使用して FIFO から読み取ることはできません。FIFO から読み取る必要がある場合は（gunzip の出力など）、代わりに LOAD DATA LOCAL INFILE を使用します。

データファイルのロードは、mysqlimport ユーティリティでも実行できます。この場合、LOAD DATA INFILE コマンドがサーバに送られます。--local オプションを指定した mysqlimport では、データファイルがクライアントホストから読み取られます。--compress オプションを指定すると、クライアントとサーバで圧縮されたプロトコルがサポートされている場合に、速度の遅いネットワークのパフォーマンスを良くすることができます。

ファイルがサーバホスト上にある場合、サーバは次の規則に従います。

- 絶対パス名が指定されている場合、サーバはそのパス名をそのまま使用する。
- 相対パス名が指定されている場合、サーバはサーバのデータディレクトリを基準にしてファイルを検索する。
- ディレクトリの指定なしにファイル名が指定されている場合、サーバはカレントデータベースのデータベースディレクトリでファイルを検索する。

注意: これらの規則に基づき、./myfile.txt という名前のファイルはサーバのデータディレクトリから読み取られるのに対し、myfile.txt という名前の同じファイルについてはカレントデータベースのデータベースディレクトリから読み取られます。たとえば、次の LOAD DATA ステートメントでは、db2 データベース内のテーブルへのファイルのロードが明示的に指定されていますが、ファイル data.txt はカレントデータベースである db1 のデータベースディレクトリから読み取られます。

```
mysql> USE db1;
mysql> LOAD DATA INFILE "data.txt" INTO TABLE db2.my_table;
```

REPLACE キーワードと IGNORE キーワードでは、既存のレコードの値と重複するユニークキー値を持つ入力レコードの処理が制御されます。

REPLACE が指定されている場合、入力レコードによって既存のレコードが置き換えられます（つまり、レコードのプライマリまたはユニークインデックスの値が既存のレコードの値と同じである場合、そのレコードによって既存のレコードが置き換えられます）。 See 項6.4.7.「REPLACE 構文」。

IGNORE が指定されている場合、ユニークキー値が既存のレコードの値と重複する入力レコードがスキップされます。 どちらのオプションも指定されていない場合、動作は LOCAL キーワードが指定されているかどうかによって異なります。LOCAL が指定されていない場合、重複したキーの値が検出されるとエラーになり、テキストファイルの残りの部分が無視されます。 LOCAL が指定されている場合、デフォルトの動作は IGNORE が指定されている場合と同じです。なぜなら、サーバは処理の最中にファイルの送信を停止することができないためです。

ロード時に外部キー制約を無視するには、LOAD DATA の実行前に SET FOREIGN_KEY_CHECKS=0 を指定します。

空の MyISAM テーブルに対して LOAD DATA INFILE を使用すると、非ユニークなインデックスのすべてが別のバッチに作成されます (REPAIR の場合と同様) 。インデックスが数多くある場合、通常、これによって LOAD DATA INFILE の処理がはるかに迅速化されます。通常、この処理は非常に迅速ですが、極端なケースでは、ALTER TABLE .. DISABLE KEYS でインデックスを無効にした後、ALTER TABLE .. ENABLE KEYS を使用してインデックスを再作成する方がさらに迅速にインデックスを作成できることもあります。 See 項4.5.6.「myisamchk を使用したテーブルの保守とクラッシュのリカバリ」。

LOAD DATA INFILE は SELECT ... INTO OUTFILE の逆です。 See 項6.4.1.「SELECT 構文」。 テーブルからファイルにデータを書き込むには、SELECT ... INTO OUTFILE を使用します。 ファイルを再びテーブルに読み取るには、LOAD DATA INFILE を使用します。 FIELDS 節と LINES 節の構文は、どちらのコマンドでも同じです。どちらの節もオプションとして指定できますが、これらの節を両方とも指定する場合は、FIELDS を LINES より前に指定する必要があります。

FIELDS 節を指定する場合は、その節の従属節 (TERMINATED BY、[OPTIONALLY] ENCLOSED BY、ESCAPED BY) もオプションとして指定できますが、少なくともこれらのうちの 1 つは必ず指定する必要があります。

FIELDS 節を指定しない場合、デフォルトは次のように記述した場合と同じです。

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
```

LINES 節を指定しない場合、デフォルトは次のように記述した場合と同じです。

```
LINES TERMINATED BY '\n'
```

注意: Winodows では行終端記号として 2 文字を使用しているため、テキストファイルを Windows システムで生成した場合は、通常、上の記述を LINES TERMINATED BY '\r\n' に変更する必要があります。wordpad などの一部のプログラムでは、行終端記号として \r を使用できます。

読み取り対象行のすべてに存在するプリフィックスをスキップする必要があるときは、LINES STARTING BY prefix_string を使用できます。

したがって、デフォルトの LOAD DATA INFILE は入力データの読み取り時に次のように動作します。

- 改行文字では、行の境界を探す。
- LINES STARTING BY prefix が指定されている場合、プリフィックスが検出されるまで読み取り、プリフィックスの後ろにある文字から再び読み取りを開始する。行にプリフィックスが含まれていない場合は、その行をスキップする。
- タブ文字では、行をフィールドに区切る。
- フィールドはどの引用文字にも囲まれていないとみなす。
- '\' に続くタブ、改行、'\' の各文字は、フィールド値の一部を成すリテラル文字として解釈する。

逆に、デフォルトの SELECT ... INTO OUTFILE は出力の書き込み時に次のように動作します。

- フィールド間にはタブ文字を書き込む。
- フィールドは引用文字で囲まない。
- フィールド値内のタブ、改行、'\' の各文字は、それぞれ '\' でエスケープする。

- 行末には改行文字を書き込む。

注意: FIELDS ESCAPED BY '\\' と記述する際には、単一バックスラッシュとして読み取らせる値を表すものとして、バックスラッシュを 2 つ指定する必要があります。

IGNORE number LINES オプションは、ファイルの先頭にある行を無視させる目的で使用します。たとえば、次のように、IGNORE 1 LINES と指定することによって、カラム名が含まれた最初のヘッダ行 (1 行) をスキップさせることができます。

```
mysql> LOAD DATA INFILE "/tmp/file_name" INTO TABLE test IGNORE 1 LINES;
```

SELECT ... INTO OUTFILE と LOAD DATA INFILE を並行して使用して、データベースからファイルにデータを書き込み、その後ファイルからデータベースに再び読み取る場合は、フィールドと行の処理に関する両方のコマンドのオプションが一致していなければなりません。一致していない場合、ファイルの内容が LOAD DATA INFILE によって正しく解釈されません。たとえば、SELECT ... INTO OUTFILE を使用して、カンマで区切られたフィールドを持つファイルを書き込むとします。

```
mysql> SELECT * INTO OUTFILE 'data.txt'
    ->        FIELDS TERMINATED BY ','
    ->        FROM ...;
```

カンマで区切られたファイルを再び読み取る正しいステートメントは、次のようになります。

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE table2
    ->        FIELDS TERMINATED BY ',';
```

上のステートメントではなく、次に示すステートメントでファイルを読み取ろうとすると、処理は正しく行われません。なぜなら、このステートメントでは、フィールド間にタブ文字を探すよう LOAD DATA INFILE に指示しているためです。

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE table2
    ->        FIELDS TERMINATED BY '\t';
```

この場合、入力ファイルの各行が単一のフィールドとして解釈されてしまいます。

LOAD DATA INFILE では、外部ソースから得られたファイルを読み取ることもできます。たとえば、dBASE 形式のファイルには、カンマで区切られ、かつ二重引用符で囲まれたフィールドが含まれています。ファイルの各行の終端が改行文字によって示されている場合、このファイルをロードするには次のコマンドを使用します。このコマンドでは、このファイルのフィールドと行の処理に関するオプションを指定しています。

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE tbl_name
    ->        FIELDS TERMINATED BY ',' ENCLOSED BY '"'
    ->        LINES TERMINATED BY '\n';
```

フィールドや行の各処理オプションには、いずれも空の文字列 ('') を指定することができます。空の文字列でない場合、FIELDS [OPTIONALLY] ENCLOSED BY と FIELDS ESCAPED BY の値は単一の文字でなければなりません。FIELDS TERMINATED BY と LINES TERMINATED BY の値には、複数の文字も使用できます。たとえば、終端に改行/復帰のペアが付いた行を書き込んだり、このような行を含むファイルを読み取る場合は、LINES TERMINATED BY '\r\n' 節を指定します。

たとえば、%% の行で区切られたジョーク集のファイルを SQL テーブルに読み取る場合は、次のように記述します。

```
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 では、フィールドの引用処理が制御されます。出力（SELECT ... INTO OUTFILE）の場合、OPTIONALLY という語を省略すると、すべてのフィールドが ENCLOSED BY 指定文字で囲まれます。このような出力（フィールド区切り記号としてカンマを使用したもの）の例を、次に示します。

```
"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"
```

OPTIONALLY を指定すると、ENCLOSED BY 指定文字は CHAR 型と VARCHAR 型のフィールドを囲むためだけに使用されます。

```
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
```

注意: フィールド値内に ENCLOSED BY 指定文字が含まれる場合は、直前に ESCAPED BY 指定文字を付けることによってエスケープされます。注意: ESCAPED BY に空の文字列を指定すると、生成された出力を LOAD DATA INFILE が正しく読み取れないことがあります。たとえば、エスケープ文字が空の場合、前出の例の出力は次のように表示されます。4 行目の 2 番目のフィールドには、引用符に続いてカンマが含まれています。このカンマはフィールドの（実際には終端記号ではないのに）終端記号のように見えます。

```
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
```

入力の場合、フィールド値の終わりに ENCLOSED BY 指定文字があると、その文字が除去されます（これは、OPTIONALLY の指定の有無にかかわらず適用されます。OPTIONALLY は入力の解釈には作用しません）。 ESCAPED BY 指定文字に続いて ENCLOSED BY 指定文字があると、その文字は、現在のフィールド値の一部を成すものとして解釈されます。

フィールドが ENCLOSED BY 指定文字で始まっている場合、後続の ENCLOSED BY 指定文字は、その後ろにフィールドまたは行の TERMINATED BY 指定文字列がある場合にのみ、フィールド値の終端として解釈されます。 あいまい性を排除するため、ENCLOSED BY 指定文字をフィールド内で通常の文字として使用する場合、2 回続けて使用することで通常の 1 文字として解釈させることができます。たとえば、ENCLOSED BY "" と指定されている場合、引用符は次のように処理されます。

```
"The ""BIG"" boss"  -> The "BIG" boss
The "BIG" boss      -> The "BIG" boss
The ""BIG"" boss    -> The ""BIG"" boss
```

FIELDS ESCAPED BY では、特殊文字の書き込みと読み取り方法が制御されます。FIELDS ESCAPED BY に空以外の文字が指定されている場合、出力でその文字が次の文字の前に付けられます。

- FIELDS ESCAPED BY 指定文字自体
- FIELDS [OPTIONALLY] ENCLOSED BY 指定文字
- FIELDS TERMINATED BY と LINES TERMINATED BY に指定された値の最初の文字
- ASCII 0 (エスケープ文字に続いて実際に記述されるのは、ゼロ値のバイトではなく ASCII '0')

FIELDS ESCAPED BY の指定が空の場合、どの文字もエスケープ処理されません。 データのフィールド値に上記の文字が含まれている場合は特に、エスケープ文字として空文字は指定すべきではありません。

入力では、FIELDS ESCAPED BY に空以外の文字が指定されている場合、その文字は除去され、それに続く文字がフィールド値の一部として通常の文字と同じように解釈されます。ただし、'0' または 'N' がエスケープされている場合 (たとえば、エスケープ文字が '\' の場合の \0 または \N) は例外です。これらの文字列は ASCII 0 (ゼロ値のバイト) および NULL として解釈されます。下記の NULL の処理に関する規則を参照してください。

'\' を使用したエスケープ構文の詳細については、項6.1.1.「リテラル:文字列と数値の記述方法」 を参照してください。

場合によっては、次に示すように、フィールドと行の各処理オプションが相互に作用することがあります。

- LINES TERMINATED BY に空の文字列が指定されていて、FIELDS TERMINATED BY に空以外の文字列が指定されている場合、行の終端も FIELDS TERMINATED BY 指定文字列によって示される。
- FIELDS TERMINATED BY と FIELDS ENCLOSED BY の値がどちらも空 ('') の場合、固定レコード (区切りなし) の形式が使用される。 固定長レコードの形式では、フィールド間に区切り記号は使用されない (行終端記号は使用可能)。代わりに、カラムの ``表示" 幅に基づいてカラム値が読み書きされる。たとえば、カラムが INT(7) として宣言されている場合、そのカラムの値は 7 文字のフィールドを使用して書き込まれる。入力時には、そのカラムの値は、7 文字を読み取ることによって取得される。

  この場合も、各行の分離には、LINES TERMINATED BY が使用される。1 行にすべてのフィールドが含まれていない場合、残りのフィールドにはそれぞれのデフォルト値が設定される。行終端記号がない場合は、終端記号を '' として指定するようにする。この場合、テキストファイルに各レコードのすべてのフィールドが含まれていなければならない。

  固定長レコード形式は、NULL 値の処理にも影響します。下記を参照。 注意: マルチバイト文字のキャラクタセットを使用している場合、固定サイズ形式は機能しない。

NULL 値の処理は、使用されている FIELDS オプションと LINES オプションによって異なります。

- FIELDS と LINES の値がデフォルトの場合、出力では NULL が \N として書き込まれ、入力では \N が NULL として読み取られる (ESCAPED BY 指定文字が '\' の場合)。
- FIELDS ENCLOSED BY に空以外の文字が指定されている場合、リテラルの NULL を値として持つフィールドは NULL 値として読み取られる (これは、FIELDS ENCLOSED BY 指定文字で囲まれた語 NULL が、文字列 'NULL' として読み取られるのと異なる)。
- FIELDS ESCAPED BY の指定が空の場合、NULL は NULL という語として書き込まれる。
- 固定長レコード形式 (FIELDS TERMINATED BY と FIELDS ENCLOSED BY の両方の指定が空の場合) では、NULL は空の文字列として書き込まれる。注意: それによって、ファイルへの書き込み時に、テーブル内の NULL 値と空の文

字列の両方が空の文字列として書き込まれるため、それらを区別できなくなる。ファイルを再び読み取るときにこれらを区別できるようにする必要がある場合は、固定レコード形式は使用しないようにする。

次に示すケースは LOAD DATA INFILE でサポートしていません。

- 固定長サイズのレコード (FIELDS TERMINATED BY と FIELDS ENCLOSED BY の指定がどちらも空) と BLOB または TEXT 型のカラム。
- 1 つの区切り記号が別の区切り文字とまったく同じか、別の区切り記号の先頭の部分と同じである場合、LOAD DATA INFILE は入力を正しく解釈できない。たとえば、次の FIELDS 節では問題が発生する。

```
FIELDS TERMINATED BY "" ENCLOSED BY ""
```

- FIELDS ESCAPED BY の指定が空の場合、FIELDS ENCLOSED BY または LINES TERMINATED BY の指定文字に続いて FIELDS TERMINATED BY 指定値があると、LOAD DATA INFILE はフィールドまたは行の読み取りを誤って早く停止する。 これは、フィールドまたは行の値の終端を LOAD DATA INFILE が正しく判別できないために起こる。

次の例では、persondata テーブルのすべてのカラムがロードされます。

```
mysql> LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

フィールドリストは指定されていないため、LOAD DATA INFILE は入力レコードに各テーブルカラムのフィールドが含まれているとみなします。デフォルトの FIELDS 値と LINES 値が使用されます。

テーブルの一部のカラムのみロードする場合は、フィールドリストを指定します。

```
mysql> LOAD DATA INFILE 'persondata.txt'
    ->         INTO TABLE persondata (col1,col2,...);
```

入力ファイル内のフィールドの順序がテーブルのカラムの順序と異なる場合も、フィールドリストを指定する必要があります。リストを指定しないと、入力フィールドとテーブルの各カラムをどのように一致させたらいいか MySQL で認識できません。

レコードに含まれているフィールドの数が足りない場合、入力フィールドが欠落しているカラムにはデフォルト値が設定されます。デフォルト値の割り当てについては、項6.5.3.「CREATE TABLE 構文」 で説明しています。

空のフィールド値はフィールド値が欠落している場合とは異なる解釈をされます。

- 文字列型のカラムの場合は、値として空の文字列が設定される。
- 数値型のカラムの場合は、値として 0 が設定される。
- 日付と時刻型のカラムの場合は、値として、その型に対応する ``ゼロ" 値が設定される。 See 項6.2.2.「日付と時刻型」。

注意: これらは、INSERT や UPDATE ステートメントで文字列型、数値型、日付または時刻型のカラムに空白の文字列を明示的に割り当てた場合と同じ値です。

TIMESTAMP 型のカラムでは、そのカラムに NULL 値（つまり \N）があるときか、フィールドリストの指定時にその TIMESTAMP 型のカラムが省略されていたときだけ（これは最初の TIMESTAMP カラムに対してのみ適用）、カラムの値として現在の日時が設定されます。

入力レコードに含まれるフィールドが多すぎる場合、余分なフィールドは無視され、警告数が加算されます。注意: MySQL 4.1.1 より前のバージョンでは、警告は発生した何らかの問題を示す数でしかありませんでした。 MySQL 4.1.1 では、SHOW WARNINGS によって、発生した問題の詳細を表示することができます。

LOAD DATA INFILE はすべての入力を文字列としてみなすため、INSERT ステートメントの場合とは異なり、ENUM 型や SET 型のカラムに数値を使用することはできません。ENUM 型と SET 型の値はすべて文字列として指定する必要があります。

C API を使用している場合は、LOAD DATA INFILE クエリの完了時に API 関数 mysql_info() を呼び出すことによって、クエリに関する情報を取得できます。情報文字列の形式は次のとおりです。

```
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
```

Warnings（警告）は、INSERT ステートメント（see 項6.4.3.「INSERT 構文」）で値を挿入するときと同じ状況で発生しますが、その他に、入力レコードに含まれるフィールドが多すぎる場合と少なすぎる場合にも警告が生成されます。警告はどこにも格納されません。単に、警告の数を、処理が正常に行われたかどうかを示すインジケータとして利用できるだけです。

警告が出力された場合にその正確な理由を確認する必要があるときは、1 つの方法として、SELECT ... INTO OUTFILE を使用して別のファイルに出力し、元の入力ファイルと比較することができます。

LOAD DATA にパイプから読み取らせる必要がある場合は、次の方法を使用できます。

```
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
```

3.23.25 より前のバージョンの MySQL を使用している場合は、上記の方法は LOAD DATA LOCAL INFILE でのみ使用可能です。

MySQL 4.1.1 では、SHOW WARNINGS を使用して、max_error_count に指定された数の警告の最初のリストを取得できます。 See 項4.6.8.9.「SHOW WARNINGS | ERRORS」。

INSERT と LOAD DATA INFILE の効率性の比較や、LOAD DATA INFILE の迅速化の詳細については、See 項5.2.10.「INSERT クエリの速度」。

6.4.9. HANDLER 構文

```
HANDLER tbl_name OPEN [ AS alias ]
HANDLER tbl_name READ index_name { = | >= | <= | < } (value1,value2,...)
  [ WHERE ... ] [LIMIT ... ]
HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST }
  [ WHERE ... ] [LIMIT ... ]
HANDLER tbl_name READ { FIRST | NEXT }
  [ WHERE ... ] [LIMIT ... ]
HANDLER tbl_name CLOSE
```

HANDLER ステートメントでは、MyISAM テーブルのストレージエンジンインタフェースに直接アクセスすることができます。

最初の形式の HANDLER ステートメントでは、テーブルが開かれ、後続の HANDLER ... READ ステートメントでのアクセスが可能になります。 このテーブルオブジェクトは他のスレッドと共有されていないため、そのスレッドが HANDLER tbl_name CLOSE を呼び出すか、またはそのスレッドが終了するまで、テーブルオブジェクトが閉じられることはありません。

2 番目の形式の HANDLER ステートメントでは、指定したインデックスが指定した値にマッチし、かつ WHERE 条件を満たしている 1 つ (または、LIMIT 節の指定に基づく複数) のレコードが読み取られます。複合インデックスがある場合は、対象のインデックスカラム値をカンマで区切られたリストとして指定します。この場合、インデックスのすべてのカラムの各値を指定するか、またはインデックスカラムの左端のプリフィックスの値を指定します。たとえば、col_a、col_b、col_c という名前の 3 つのカラムがこの順序でインデックスに含まれているとします。 HANDLER ステートメントでは、インデックスの 3 つすべてのカラムの値を指定するか、または左端のプリフィックスのカラムの値を指定することができます。次に例を示します。

```
HANDLER ... index_name = (col_a_val,col_b_val,col_c_val) ...
HANDLER ... index_name = (col_a_val,col_b_val) ...
HANDLER ... index_name = (col_a_val) ...
```

3 番目の形式の HANDLER ステートメントでは、WHERE 条件を満たしている 1 つ (または、LIMIT節の指定に基づく複数) のレコードがテーブルからインデックスの順序で読み取られます。

4 番目の形式の HANDLER ステートメント (インデックスの指定なし) では、WHERE 条件を満たしている 1 つ (または、LIMIT 節の指定に基づく複数) のレコードがテーブルからそのままのレコード順序 (データファイルに格納されている順序) で読み取られます。テーブルのフルスキャンを行う必要がある場合、この形式は HANDLER tbl_name READ index_name 形式より迅速です。

HANDLER ... CLOSE では、HANDLER ... OPEN で開かれたテーブルが閉じられます。

注意:PRIMARY KEY の HANDLER インタフェースを使用している場合は、HANDLER tbl READ `PRIMARY` > (...) のように、キーワード PRIMARY をバッククォート記号で必ず囲みます。

HANDLER は、やや低レベルのステートメントです。たとえば、整合性は確保されません。つまり、HANDLER ... OPEN はテーブルのスナップショットを取るものではなく、テーブルのロックも行いません。そのため、HANDLER ... OPEN の発行後に、テーブルデータが変更される (そのスレッドまたは他のスレッドによって) 可能性があります。このような変更は HANDLER ... NEXT スキャンや HANDLER ... PREV スキャンでは部分的にしか示されないことがあります。

通常の SQL の代わりにこのインタフェースを使用する理由は、次のとおりです。

- 次の理由で、SELECT より処理が迅速。
  - HANDLER OPEN では、指定したストレージエンジンがスレッドに割り当てられる。
  - 実行される解析が少ない。
  - オプティマイザやクエリチェックのオーバヘッドがない。
  - 2 つのハンドラ要求間で、使用テーブルをロックする必要がない。
  - ハンドラインタフェースでは、整合性のあるデータの提供は重要とされていないため(たとえば、ダーティリードが

可能）、SQL によって通常許可されない最適化をストレージエンジンで実行することができる。

- ISAM のような MySQL インタフェースを使用するアプリケーションの移植をはるかに容易に実行できる。
- SQL では難しい（場合によっては不可能な）方法で、データベースをスキャンできる。データベースの対話式ユーザインタフェースを提供するアプリケーションを処理する場合には、データを参照する方法としてハンドラインタフェースを使用したほうがより自然である。

6.4.10. DO 構文

```
DO expression, [expression, ...]
```

上の構文では、式が実行されますが、結果は何も返されません。これは SELECT expression, expression の省略形ですが、結果が必要ない場合には、処理速度がやや速いという利点があります。

この構文は、主に、RELEASE_LOCK などのように副次的な影響のある関数に使用すると便利です。

6.5. データ定義: CREATE、DROP、ALTER

6.5.1. CREATE DATABASE 構文

```
CREATE DATABASE [IF NOT EXISTS] db_name
```

CREATE DATABASE では、指定した名前のデータベースが作成されます。

使用可能なデータベース名の規則は、項6.1.2.「データベース名、テーブル名、インデックス名、カラム名、エイリアス名」 で説明しています。IF NOT EXISTS を指定していないときに、データベースが既に存在していると、エラーが発生します。

MySQL では、データベースは、対応するファイルをその内部に含むディレクトリとして実装されます。作成当初のデータベースにはテーブルは何もないため、CREATE DATABASE ステートメントの実行では、MySQL データディレクトリの下にディレクトリが作成されるだけです。

データベースの作成は mysqladmin でも可能です。 See 項4.9.「MySQL クライアントサイドのスクリプトとユーティリティ」。

6.5.2. DROP DATABASE 構文

```
DROP DATABASE [IF EXISTS] db_name
```

DROP DATABASE では、データベース内のすべてのテーブルが破棄され、データベースが削除されます。シンボリックリンクのあるデータベースに対して DROP DATABASE を実行すると、リンクと元のデータベースとの両方が削除されます。このコマンドの使用時には、十分に注意してください。

DROP DATABASE では、データベースディレクトリから削除されたファイルの数が返されます。MyISAM テーブルの場合、通常、各テーブルが .MYD、.MYI、.frm の 3 つのファイルに対応しているため、ファイル数はテーブル数の 3 倍になります。

DROP DATABASE コマンドでは、次の拡張子を持つすべてのファイルが、指定したデータベースディレクトリから削除されます。

拡張子	拡張子	拡張子	拡張子
.BAK	.DAT	.HSH	.ISD
.ISM	.ISM	.MRG	.MYD
.MYI	.db	.frm	

2 桁の数値で構成されるサブディレクトリ (RAID ディレクトリ) もすべて削除されます。

MySQL バージョン 3.22 以降では、キーワード IF EXISTS を使用して、データベースが存在しない場合に発生するエラーを回避することができます。

データベースの破棄は mysqladmin でも可能です。 See 項4.9.「MySQL クライアントサイドのスクリプトとユーティリティ」。

6.5.3. CREATE TABLE 構文

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name [(create_definition,...)]
[table_options] [select_statement]

または

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name [(] LIKE old_tbl_name [)];

create_definition:
    col_name type [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT]
          [[PRIMARY] KEY] [COMMENT 'string'] [reference_definition]
  | [CONSTRAINT [symbol]] PRIMARY KEY (index_col_name,...)
  | KEY [index_name] (index_col_name,...)
  | INDEX [index_name] (index_col_name,...)
  | [CONSTRAINT [symbol]] UNIQUE [INDEX] [index_name] (index_col_name,...)
  | FULLTEXT [INDEX] [index_name] (index_col_name,...)
  | [CONSTRAINT [symbol]] FOREIGN KEY [index_name] (index_col_name,...)
          [reference_definition]
  | CHECK (expr)

type:
    TINYINT[(length)] [UNSIGNED] [ZEROFILL]
  | SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
  | MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
  | INT[(length)] [UNSIGNED] [ZEROFILL]
  | INTEGER[(length)] [UNSIGNED] [ZEROFILL]
  | BIGINT[(length)] [UNSIGNED] [ZEROFILL]
  | REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]
  | NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]
  | CHAR(length) [BINARY | ASCII | UNICODE]
  | VARCHAR(length) [BINARY]
  | DATE
  | TIME
```

```
  | TIMESTAMP
  | DATETIME
  | TINYBLOB
  | BLOB
  | MEDIUMBLOB
  | LONGBLOB
  | TINYTEXT
  | TEXT
  | MEDIUMTEXT
  | LONGTEXT
  | ENUM(value1,value2,value3,...)
  | SET(value1,value2,value3,...)

index_col_name:
        col_name [(length)] [ASC | DESC]

reference_definition:
        REFERENCES tbl_name [(index_col_name,...)]
                  [MATCH FULL | MATCH PARTIAL]
                  [ON DELETE reference_option]
                  [ON UPDATE reference_option]

reference_option:
        RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

table_options: table_option [table_option] ...

table_option:
    TYPE = {BDB | HEAP | ISAM | InnoDB | MERGE | MRG_MYISAM | MYISAM }
  | AUTO_INCREMENT = #
  | AVG_ROW_LENGTH = #
  | CHECKSUM = {0 | 1}
  | COMMENT = 'string'
  | MAX_ROWS = #
  | MIN_ROWS = #
  | PACK_KEYS = {0 | 1 | DEFAULT}
  | PASSWORD = 'string'
  | DELAY_KEY_WRITE = {0 | 1}
  | ROW_FORMAT = { DEFAULT | DYNAMIC | FIXED | COMPRESSED }
  | RAID_TYPE = { 1 | STRIPED | RAID0 } RAID_CHUNKS=#  RAID_CHUNKSIZE=#
  | UNION = (table_name,[table_name...])
  | INSERT_METHOD = { NO | FIRST | LAST }
  | DATA DIRECTORY = 'absolute path to directory'
  | INDEX DIRECTORY = 'absolute path to directory'
  | DEFAULT CHARACTER SET character_set_name [COLLATE collation_name]

select_statement:
    [IGNORE | REPLACE] [AS] SELECT ...    (Some legal select statement)
```

CREATE TABLE では、指定した名前のテーブルが作成されます。 使用可能なテーブル名の規則は、項6.1.2.「データベース名、テーブル名、インデックス名、カラム名、エイリアス名」 で説明しています。 デフォルトでは、テーブルはカレントデータベースに作成されます。 テーブルがすでに存在する場合、カレントデータベースがない場合、またはデータベースが存在しない場合には、エラーが発生します。

MySQL バージョン 3.22 以降では、テーブル名を db_name.tbl_name の形式で指定することで、指定したデータベースにテーブルを作成することができます。 これは、カレントデータベースの有無にかかわらず有効です。

MySQL バージョン 3.23 以降では、テーブルの作成時に TEMPORARY キーワードを指定することができます。テンポラリテーブルは現在の接続の間のみ有効で、接続が閉じると自動で削除されます。そのため、異なる 2 つの接続が同じテンポラリテーブル名を使用できます。この場合、それぞれの接続のテンポラリテーブル間でコンフリクトが発生したり、同名の既存のテーブルとの間でコンフクリトが発生したりすることはありません（既存のテーブルはテンポラリテーブルが削除されるまで表示されません）。MySQL 4.0.2 以降では、テンポラリテーブルの作成には、CREATE TEMPORARY TABLES 権限が必要です。

MySQL バージョン 3.23 以降では、キーワード IF NOT EXISTS を使用して、テーブルがすでに存在する場合に発生するエラーを回避することができます。注意: 既存のテーブルの構造が、CREATE TABLE ステートメントで指定されたテーブルの構造と同じかどうかは検証されません。

バージョン 4.1.0 以降では、属性 SERIAL を BIGINT NOT NULL AUTO_INCREMENT UNIQUE のエイリアスとして使用できます。これは互換性を考慮した機能です。

MySQL 3.23 以降では、CREATE TABLE ステートメントの最後に SELECT ステートメントを追加することによって、1 つのテーブルから別のテーブルを作成することができます。

```
CREATE TABLE new_tbl SELECT * FROM orig_tbl;
```

インデックスは新しいテーブルに持ち越されません。また、一部のカラム型の変換が行われる場合があります。たとえば、AUTO_INCREMENT 属性は維持されず、VARCHAR 型のカラムは CHAR 型のカラムになることがあります。

CREATE ... SELECT でテーブルを作成するときには、クエリ内の関数呼び出しや式に対して必ずエイリアスを指定するようにします。エイリアスを指定しないと、CREATE ステートメントが正常に実行されなかったり、不適切なカラム名が生成されたりする場合があります。

```
CREATE TABLE artists_and_works
SELECT artist.name, COUNT(work.artist_id) AS number_of_works
FROM artist LEFT JOIN work ON artist.id = work.artist_id
GROUP BY artist.id;
```

MySQL 4.1 以降では、生成されるカラムの型を明示的に指定することができます。

```
CREATE TABLE foo (a tinyint not null) SELECT b+1 AS 'a' FROM bar;
```

MySQL 4.1 では、LIKE でも、別のテーブルの定義に基づいて新しいテーブル（元のテーブルのカラム属性やインデックスをすべて含む）を作成することができます。

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

CREATE TABLE ... LIKE では、元のテーブルに指定された DATA DIRECTORY や INDEX DIRECTORY のテーブルオプションはいずれもコピーされません。

データベースディレクトリ内の一部のファイルには、各テーブルの tbl_name が反映されます。MyISAM 型のテーブルの場合、次のようになります。

ファイル	用途
tbl_name.frm	テーブル形式（定義）ファイル
tbl_name.MYD	データファイル

tbl_name.MYI	インデックスファイル

さまざまなカラム型の特性の詳細については、項6.2.「カラム型」 を参照してください。

- NULL も NOT NULL も指定されていない場合、カラムは NULL が指定されているときと同じように処理される。
- 整数型カラムは追加属性 AUTO_INCREMENT を持つことができる。 インデックス付きの AUTO_INCREMENT カラムに NULL (推奨) または 0 を挿入すると、そのカラムには連続値の次の値が設定される。 通常、これは value+1 になる (value はテーブルに現在格納されているそのカラムの最大値)。 AUTO_INCREMENT は 1 から開始される。 See 項11.1.3.32.「mysql_insert_id()」。

  MySQL 4.1.1 以降では、--sql-mode サーバオプションまたは sql_mode サーバ変数に対して NO_AUTO_VALUE_ON_ZERO フラグを指定することによって、新しい連続値を生成する代わりに、AUTO_INCREMENT カラムに 0 を 0 として格納することができる。 See 項4.1.1.「mysqld コマンドラインオプション」。

  AUTO_INCREMENT カラムの最大値が入ったレコードを削除した場合、その値は ISAM テーブルや BDB テーブルでは再利用されるが、MyISAM テーブルや InnoDB テーブルでは再利用されない。DELETE FROM table_name (WHERE 節なし) を AUTOCOMMIT モードで実行してテーブル内のすべてのレコードを削除した場合、InnoDB を除くすべてのテーブル型で、連続値が初めから開始される。 See 項7.5.12.5.「InnoDB での AUTO_INCREMENT カラムの仕組み」。

  注意: AUTO_INCREMENT カラムはテーブルごとに 1 つだけ存在できる。このカラムにはインデックスを付ける必要があり、また DEFAULT 値は設定できない。 MySQL バージョン 3.23 では、AUTO_INCREMENT カラムは、正の値だけを持つ場合にのみ正しく機能する。負の数値が挿入されると、その値はきわめて大きな正数としてみなされる。 これは、数値が正の数から負の数に ``折り返す'' ときに発生する精度の問題を回避するためと、0 が入った AUTO_INCREMENT カラムが誤って取得されないようにするためである。

  MyISAM テーブルと BDB テーブルでは、複合インデックスで AUTO_INCREMENT セカンダリカラムを指定できる。See 項3.6.9.「AUTO_INCREMENT の使用」。

  MySQL と一部の ODBC アプリケーションとの互換性を確保するため、最後に挿入されたレコードの AUTO_INCREMENT 値を次のクエリで検出することができる。

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

- TIMESTAMP 型カラムでは、他のカラム型とは異なる方法で NULL 値が処理される。TIMESTAMP 型カラムには、NULL は格納できない。カラムに NULL を挿入すると、カラムの値として現在の日時が設定される。TIMESTAMP 型のカラムはこのように動作するため、NULL 属性と NOT NULL 属性は通常どおりには適用されず、それらを指定しても無視される。

  その一方で、TIMESTAMP 型のカラムを MySQL クライアントにとって使用しやすくするために、サーバは、(実際には、TIMESTAMP 型カラムには NULL 値は格納されないにもかかわらず) TIMESTAMP 型カラムについて、NULL 値の割り当てが可能 (true) と報告する。DESCRIBE tbl_name を使用してテーブルに関する記述を取得すると、これを確認できる。

  注意: TIMESTAMP 型のカラムに、値として 0 を設定するのは、NULL を設定するのとは異なる。0 は TIMESTAMP 型の有効な値である。

- DEFAULT 値は定数でなければならず、関数や式を使用することはできない。

  DEFAULT 値が指定されていない場合、そのカラムには、次の方法で、MySQL によって DEFAULT 値が自動的に割り当てられる。

  値として NULL を取れるカラムの場合、デフォルト値は NULL になる。

  NOT NULL として宣言されているカラムの場合、デフォルト値はそれぞれのカラム型によって決まる。

  - AUTO_INCREMENT 属性を持つと宣言されていない数値型カラムの場合、デフォルトは 0。AUTO_INCREMENT カラムの場合、デフォルト値は連続値の次の値になる。
  - TIMESTAMP 型以外の日付と時刻型の場合、デフォルトはその型に対応するゼロ値。テーブル内の最初の TIMESTAMP 型カラムのデフォルト値は、現在の日時になる。 See 項6.2.2.「日付と時刻型」。
  - ENUM 型以外の文字列型の場合、デフォルト値は空の文字列。ENUM の場合、デフォルト値は最初の列挙値になる。

  デフォルト値は定数でなければならない。したがって、日付カラムのデフォルト値として、NOW() や CURRENT_DATE などの関数を設定することはできない。

- COMMENT オプションでは、カラムに関するコメントを指定することができる。 コメントは SHOW CREATE TABLE ステートメントと SHOW FULL COLUMNS で表示される。 このオプションは MySQL 4.1 から利用可能 (それ以前のバージョンでは、使用は可能だが無視される) 。

- KEY は、通常 INDEX のシノニム。 バージョン 4.1 以降、キー属性 PRIMARY KEY は単に KEY として指定することもできる。この機能は他のデータベースとの互換性を考慮して実装された。

- MySQL では、UNIQUE キーには重複のない値以外は格納できない。 既存のレコードのキーと同じキーを持つ新しいレコードを追加しようとすると、エラーが発生する。

- PRIMARY KEY は、すべてのキーカラムが NOT NULL として定義されていなければならないユニーク KEY である。NOT NULL として明示的に定義されていないと、暗黙的 (かつ自動的) に NOT NULL に設定される。MySQL において、このキーは PRIMARY と呼ばれる。個々のテーブルは PRIMARY KEY を 1 つだけ持つことができる。 PRIMARY KEY がない場合に、何らかのアプリケーションがテーブルの PRIMARY KEY を要求すると、MySQL では、NULL カラムをまったく持たない最初の UNIQUE キーが PRIMARY KEYとして返される。

- 複合インデックスを PRIMARY KEY にすることもできる。しかし、カラムの仕様で PRIMARY KEY キー属性を使用して複合インデックスは作成することはできない。そのようにしても、単一のカラムがプライマリとしてマークされるにすぎない。 この場合、別に PRIMARY KEY(index_col_name, ...) 節を使用する必要がある。

- UNIQUE インデックスでは、インデックスのすべての値に重複がない状態でなければならない。ただし、例外として、そのインデックスのカラムの 1 つで NULL 値が格納可能な場合、複数の NULL 値を格納できる。 この例外は BDB 型のテーブルには適用されない。BDB 型のテーブルには単一の NULL のみ格納可能。

- PRIMARY キーまたは UNIQUE キーが単一のカラムで構成されていて、そのカラム型が整数型の場合、そのキーは _rowid として参照することもできる (バージョン 3.23.11 の新機能) 。

- PRIMARY KEY でないインデックスに名前を割り当てないと、そのインデックスは index_col_name の最初のカラム名と同じ名前が割り当てられ、そのインデックスを一意 (ユニーク) なものとするためのオプションのサフィックス (_2、_3、...) が付けられる。テーブルのインデックス名は SHOW INDEX FROM tbl_name を使用して確認できる。 See

項4.6.8.1.「データベース、テーブル、カラム、およびインデックスに関する情報の取得」。

- NULL 値を持つことができるカラムに対するインデックスの作成は、MyISAM、InnoDB、BDB の各テーブル型でのみサポートしている。その他のテーブルでは、カラムが NOT NULL と宣言されていないとエラーになる。
- インデックスの指定で col_name(length) 構文を使用することによって、CHAR 型または VARCHAR 型カラムの最初から length に指定した長さのバイトのみを使用するインデックスを作成できる。この方法で、インデックスファイルのサイズをかなり小さくすることができる。 See 項5.4.4.「カラムインデックス」。
- BLOB 型と TEXT 型のカラムのインデックスの作成は、MyISAM テーブルと（MySQL 4.0.14 以降の）InnoDB テーブルでのみサポートしている。BLOB 型または TEXT 型のカラムにインデックスを付けるときには、インデックスの長さ（最大 255 バイト）を必ず指定する必要がある。次に例を示す。

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

- index_col_name の指定では、最後に ASC または DESC を付けることができる。 これらのキーワードは、昇順または降順によるインデックス値の格納を指定できるようにする今後の拡張に対応している。現時点では、これらのキーワードは解析されるが無視される。インデックス値は常に昇順で格納される。
- TEXT 型または BLOB 型のカラムで ORDER BY または GROUP BY が使用されていると、サーバは、最初の部分の、max_sort_length サーバ変数が示す数のバイトのみを使用して値をソートする。 See 項6.2.3.2.「BLOB 型と TEXT 型」。
- MySQL バージョン 3.23.23 以降では、特殊な FULLTEXT インデックスも作成できる。これは全文検索に使用される。FULLTEXT インデックスは、MyISAM テーブルでのみサポートしている。このインデックスは、CHAR 型、VARCHAR 型、TEXT 型のカラムからのみ作成できる。 インデックスの作成は常にカラム全体に対して行われる。部分的なインデックスの作成は行われない。処理の詳細については、項6.8.「MySQL 全文検索」 を参照。
- MySQL バージョン 3.23.44 以降では、InnoDB テーブルで外部キー制約のチェックをサポートしている。See 項7.5.「InnoDB テーブル」。 注意: InnoDB での FOREIGN KEY 構文は上に示した構文より制限されている。参照テーブルのカラム名は、必ず明示的に指定する必要がある。 InnoDB では、MySQL 3.23.50以降、外部キーに対する ON DELETE アクションを、また MySQL 4.0.8 以降では ON UPDATE アクションをサポートしている。 正確な構文については、このマニュアルの InnoDB のセクションを参照。 See 項7.5.5.2.「FOREIGN KEY 制約」。 他のテーブル型については、MySQL サーバは CREATE TABLE コマンドの FOREIGN KEY、CHECK、REFERENCES の各構文を解析するが、それ以上の処理は行わない。 See 項1.8.4.5.「外部キー」。
- MyISAM テーブルと ISAM テーブルの場合、各 NULL カラムは 1 ビット余分に使用して、最も近いバイトに切り上げられる。 バイトでの最大レコード長は次のように計算される。

```
レコード長 = 1
       + (カラム長の合計)
       + (NULL カラムの数 + delete_flag + 7)/8
       + (可変長カラムの数)
```

静的レコード形式のテーブルでは、delete_flag は 1。静的テーブルでは、行レコードで、そのレコードが削除されたものであるかを示すフラグ用に 1 ビットが使用される。動的テーブルでは、このフラグは可変長レコードの頭に格納されるため、delete_flag は 0 になる。

これらの計算は InnoDB テーブルには適用されない。InnoDB テーブルでは、NULL カラムの格納サイズは NOT NULL カラムと比較して変わらない。

- table_options オプションと SELECT オプションは MySQL バージョン 3.23 以降でのみ実装されている。

  テーブル型を指定する TYPE オプションは次の値を取る。

テーブル型	説明
BDB または BerkeleyDB	ページのロックを行う、トランザクションセーフテーブル。 See 項7.6.「BDB または BerkeleyDB テーブル」。
HEAP	このテーブルのデータはメモリにしか格納されない。 See 項7.4.「HEAP テーブル」。
ISAM	元のストレージエンジン。 See 項7.3.「ISAM テーブル」。
InnoDB	行ロックを行う、トランザクションセーフテーブル。 See 項7.5.「InnoDB テーブル」。
MERGE	1 つのテーブルとして使用される MyISAM テーブルの集まり。 See 項7.2.「MERGE テーブル」。
MRG_MyISAM	MERGE のエイリアス。
MyISAM	ISAM に代わる、バイナリの移植が可能な新しいストレージエンジン。 See 項7.1.「MyISAM テーブル」。

  See 章 7. MySQL のテーブル型。

  テーブル型が指定されているときに、そのテーブル型が使用できない場合、MySQL では、代わりに MyISAM が使用される。 たとえば、テーブル定義に TYPE=BDB オプションが含まれているときに、BDB 型のテーブルを MySQL サーバでサポートしていない場合、テーブルは MyISAM テーブルとして作成される。そのため、マスタにトランザクションテーブルがありながら、スレーブでは非トランザクションテーブルを作成する (速度を上げるため) といった、レプリケーション設定が可能になる。MySQL 4.1.1 では、指定したテーブル型が受け付けられないと警告が出力される。

  その他のテーブルオプションは、テーブルの動作を最適化するために使用される。ほとんどの場合、これらはいずれも指定しなくてよい。 これらのオプションは、特に断りがない限り、すべてのテーブル型に適用される。

オプション	説明
AUTO_INCREMENT	テーブルに設定する次の AUTO_INCREMENT 値 (MyISAM テーブルのみ。InnoDB テーブルの最初の AUTO_INCREMENT 値を設定するには、1 つ少ない値を持つダミーレコードを挿入し、後からそのダミーレコードを削除する) 。
AVG_ROW_LENGTH	テーブルのレコードの長さの平均の近似値。可変サイズレコードを持つ大きなテーブル以外では、この値を設定する必要はない。
CHECKSUM	すべての口ーのチェクサムを MySQL に維持させるには、この値を 1 に設定する (それにより、テーブルの更新速度はやや遅くなるが、破損したテーブルを見つけやすくなる) (MyISAM テーブルのみ) 。
COMMENT	テーブルに関する、60 文字のコメント。
MAX_ROWS	テーブルに格納する予定の最大レコード数。
MIN_ROWS	テーブルに格納する予定の最小レコード数。
PACK_KEYS	インデックスのサイズを小さくするには、この値を 1 に設定する。通常、それにより、更新速度が遅くなるが、読み取りは速くなる (MyISAM テーブルと ISAM テーブルのみ) 。この値を 0 に設定すると、キーのパックがすべて無効化される。この値を

	DEFAULT に設定すると（MySQL 4.0 の場合）、ストレージエンジンによって、CHAR 型または VARCHAR 型の長いカラムのみがパックされる。
PASSWORD	パスワードを使用して .frm ファイルを暗号化する。標準の MySQL バージョンでは、このオプションを指定しても何も行われない。
DELAY_KEY_WRITE	テーブルが閉じられるまでキーテーブルの更新を遅らせるには、この値を1 に設定する（MyISAM テーブルのみ）。
ROW_FORMAT	レコードの格納方法を定義する。現在、このオプションは、形式として DYNAMIC と FIXED をサポートしている MyISAM テーブルに対してのみ機能する。 See 項7.1.2.「MyISAM テーブル形式」。

MyISAM テーブルの使用時には、MySQL で、MAX_ROWS * AVG_ROW_LENGTH の積によって、結果のテーブルがどれくらいのサイズになるかの計算が行われる。上記のオプションのどれも指定しないと、テーブルの最大サイズは 4G（使用しているオペレーティングシステムで 2G のテーブルしかサポートされていないときは 2G）になる。その理由は、単に大きなファイルが必要でない場合に、ポインタのサイズを抑制することによって、インデックスをより小さく、より迅速化することにある。

PACK_KEYS を指定しないと、デフォルトでは、文字列のパックのみ行われ、数値のパックは行われない。PACK_KEYS=1 と指定すると、数値もパックされる。

バイナリ数値キーのパック時には、MySQL によってプリフィックスの圧縮が行われる。 つまり、これによって大きな利益を得られるのは、同じ数値が数多く存在する場合に限られる。プリフィックスの圧縮では、前のキーの何バイトが次のキーと同じであるかを示す追加の 1 バイトが各キーで必要になる（注意: 圧縮のパフォーマンスを良くするため、キーの直後に、レコードのポインタが上位バイトから下位バイトの順に格納される）。したがって、連続する 2 つのレコードに同じキーが数多くあるとき、通常、``同じ'' キーの後に続くものはいずれも 2 バイト（レコードのポインタも含めて）しか取らない。それに比べて通常の場合は、後に続くキーで storage_size_for_key + pointer_size 分（通常 4 バイト）必要になる。その一方、すべてのキーがまったく異なる場合は、NULL 値を持てない各キーで 1 バイト余分に使用される（この場合、パックされたキーの長さは、キーが NULL であるかどうかを示すバイトと同じバイトに格納される）。

- MySQL 3.23 以降では、CREATE ステートメントの後に SELECT を指定すると、SELECT のすべての要素に対応する新しいフィールドが MySQL によって作成される。次に例を示す。

```
mysql> CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,
    ->       PRIMARY KEY (a), KEY(b))
    ->       TYPE=MyISAM SELECT b,c FROM test2;
```

この場合、a、b、c の 3 つのカラムを持つ MyISAM テーブルが作成される。 注意: SELECT ステートメントからのカラムは、テーブルの上に重ねられるのではなく、テーブルの右側に追加される。次に例を示す。

```
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)
```

テーブル foo の各レコードに対応するレコードが、foo からの値と新しいカラムのデフォルト値とともに bar に挿入される。

CREATE TABLE ... SELECT では、インデックスの作成は自動では行われない。これは、このコマンドをできるだけ柔軟なものにするためである。作成するテーブルにインデックスが必要な場合は、SELECT ステートメントの前にインデックスを指定する。

```
mysql> CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

テーブルへのデータのコピー時にエラーが発生した場合、データは自動で削除される。

SELECT の前に IGNORE または REPLACE を付けることで、ユニークキー値が重複するレコードの処理方法を指定できる。 IGNORE の場合、新しいレコードのユニークキー値が既存のレコードの値と重複していると、その新しいレコードは破棄される。REPLACE の場合、新しいレコードによって、同じユニークキー値を持つ既存のレコードが置換される。IGNORE と REPLACE のどちらも指定していない場合、ユニークキー値の重複が検出されるとエラーになる。

更新ログやバイナリログを使用して元のテーブルを確実に再作成できるようにするため、MySQL では CREATE TABLE ... SELECT 実行中の同時挿入は行えない。

- RAID_TYPE オプションでは、大きなファイルをサポートしていないオペレーティングシステムで MyISAM データファイル (インデックスファイルではなく) に対する 2G または 4G の制限を超すことができる。注意: このオプションは大きなファイルをサポートしているファイルシステムでは推奨されない。

  異なる物理ディスク上に RAID ディレクトリを配置することによって I/O のボトルネックを迅速化することができる。RAID_TYPE はあらゆるオペレーティングシステムで機能するが、MySQL のコンフィギャを --with-raid として行っておく必要がある。現時点で使用可能な RAID_TYPE は STRIPED (1 と RAID0 はこのエイリアス) 。

  MyISAM テーブルに対して RAID_TYPE=STRIPED と指定すると、データベースディレクトリに、00、01、02 という RAID_CHUNKS サブディレクトリが MyISAM によって作成される。これらのディレクトリのそれぞれで、table_name.MYD が MyISAM によって作成される。データファイルへのデータの書き込み時、RAID ハンドラによって、最初の RAID_CHUNKSIZE *1024 バイトが最初のファイルにマップされ、次の RAID_CHUNKSIZE *1024 バイトが次のファイルにマップされる (以下同様) 。

- UNION は、同一テーブルのコレクションを 1 つのものとして使用する場合に指定する。これは、MERGE テーブルに対してのみ機能する。 See 項7.2.「MERGE テーブル」。

  現在のところ、MERGE テーブルにマップするテーブルに対する SELECT、UPDATE、DELETE の各権限が必要になる。 マップ対象のテーブルはいずれも MERGE テーブルと同じデータベースに存在しなければならない。

- MERGE テーブルにデータを挿入するには、レコードの挿入先とするテーブルを INSERT_METHOD で指定する必要がある。 INSERT_METHOD は MERGE テーブルのみに使用できるオプションである。See 項7.2.「MERGE テーブル

」。このオプションは MySQL 4.0.0 で導入された。

- 作成されたテーブルでは、最初に PRIMARY キーが置かれ、続いてすべての UNIQUE キー、通常キーの順に配置される。そのため、MySQL オプティマイザで使用するキーを優先させることができ、また重複する UNIQUE キーをより迅速に検出できる。
- DATA DIRECTORY='directory' または INDEX DIRECTORY='directory' を使用することによって、ストレージエンジンのデータファイルとインデックスファイルの格納場所を指定できる。注意: この directory には、対象のディレクトリのフルパス (相対パスではなく) を指定する。

  これは、MySQL 4.0 において、--skip-symlink オプションは指定しないときの MyISAM テーブルに対してのみ機能する。 See 項5.6.1.2.「Unix 上のテーブルに対するシンボリックリンクの使用」。

6.5.3.1. カラムの暗黙的な変更

場合によっては、CREATE TABLE ステートメントで指定されているカラムの型、属性が、MySQL によって暗黙的に変更されることがあります (このような変更は ALTER TABLE でも発生する場合があります)。

- 長さが 4 文字に満たない VARCHAR 型のカラムは CHAR 型に変更される。
- テーブルのいずれかのカラムが可変長である場合は、結果的にそのレコード全体が可変長になる。したがって、テーブルに可変長のカラム (VARCHAR、TEXT、BLOB) が含まれている場合、長さが 3 文字を超す CHAR 型のカラムはいずれも VARCHAR 型カラムに変更される。これはカラムの使用方法には影響しない。MySQL では、VARCHAR 型は単に文字を格納するもう 1 つの手段として使用されている。MySQL でこの変換が実行される理由は、スペースを節約し、テーブル処理を迅速化するためである。 See 章 7. MySQL のテーブル型。
- バージョン 4.1.0 以降では、255 文字を超える長さを持つ CHAR 型または VARCHAR 型のフィールドはいずれも TEXT 型に変換される。 これは互換性を考慮した機能。
- TIMESTAMP 型の表示サイズは 2 ～ 14 の範囲の偶数でなければならない。 表示サイズとして14 を超す数値や 0 を指定すると、サイズは強制的に 14 に設定される。1 ～ 13 の範囲の奇数値を指定すると、サイズは強制的に 1 つ上の偶数に設定される。
- TIMESTAMP 型のカラムにはリテラルの NULL は格納できない。このカラムに NULL を挿入すると、値として現在の日時が設定される。TIMESTAMP 型のカラムはこのように動作するため、NULL 属性と NOT NULL 属性は通常どおりには適用されず、それらを指定しても無視される。DESCRIBE tbl_name では、TIMESTAMP 型のカラムは常に NULL 値の割り当てが可能と報告される。
- MySQL では、他の SQL データベースベンダが使用している一部のカラム型が MySQL のカラム型にマップされる。See 項6.2.5.「他のデータベースエンジンのカラム型の使用」。

指定した以外のカラム型が MySQL によって使用されたかどうか確認するには、テーブルの作成または変更後に、DESCRIBE tbl_name ステートメントを発行します。

myisampack を使用してテーブルを圧縮すると、特定の他のカラム型の変更が発生する場合があります。 See 項7.1.2.3.「圧縮テーブルの特性」。

6.5.4. ALTER TABLE 構文

```
ALTER [IGNORE] TABLE tbl_name alter_specification [, alter_specification ...]

alter_specification:
    ADD [COLUMN] create_definition [FIRST | AFTER column_name ]
  | ADD [COLUMN] (create_definition, create_definition,...)
  | ADD INDEX [index_name] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]] PRIMARY KEY (index_col_name,...)
  | ADD [CONSTRAINT [symbol]] UNIQUE [index_name] (index_col_name,...)
  | ADD FULLTEXT [index_name] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]] FOREIGN KEY [index_name] (index_col_name,...)
        [reference_definition]
  | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
  | CHANGE [COLUMN] old_col_name create_definition
        [FIRST | AFTER column_name]
  | MODIFY [COLUMN] create_definition [FIRST | AFTER column_name]
  | DROP [COLUMN] col_name
  | DROP PRIMARY KEY
  | DROP INDEX index_name
  | DISABLE KEYS
  | ENABLE KEYS
  | RENAME [TO] new_tbl_name
  | ORDER BY col
  | CHARACTER SET character_set_name [COLLATE collation_name]
  | table_options
```

ALTER TABLE では、既存のテーブルの構造を変更することができます。 たとえば、カラムの追加や削除、インデックスの作成や破壊、既存のカラムの型変更、カラム名やテーブル名自体の変更などの操作を実行できます。また、テーブルおよびテーブル型に関するコメントを変更することもできます。

See 項6.5.3.「CREATE TABLE 構文」。

ALTER TABLE を使用してカラムの仕様を変更したにもかかわらず、カラムが変更されていないと DESCRIBE tbl_name で示された場合は、項6.5.3.1.「カラムの暗黙的な変更」 で挙げている理由のいずれかにより、変更が MySQL によって無視された可能性があります。たとえば、VARCHAR 型のカラムを CHAR 型に変更しようとしたときに、他の可変長カラムがテーブルにまだ含まれていると、このカラムに対しては VARCHAR 型が引き続き使用されます。

ALTER TABLE の処理では、元のテーブルの一時的なコピーが作成されます。 変更はこのコピーに対して実行されます。その後元のテーブルが削除され、新しいテーブルの名前が変更されます。この変更処理は、すべての更新が、エラーになることなく、確実に新しいテーブルに自動でリダイレクトされるように実行されます。ALTER TABLE の実行中、元のテーブルは他のクライアントによって読み取り可能です。このテーブルの更新とテーブルへの書き込みは、新しいテーブルの準備が整うまで停止されます。

注意: RENAME 以外のオプションを ALTER TABLE に指定した場合は、厳密にはデータをコピーする必要がないとき（カラム名の変更時など）でも、必ずテンポラリテーブルが MySQL によって作成されます。これについては今後修正する予定ですが、通常 ALTER TABLE はそれほど頻繁に使用されないため、TODOリストにおけるこの修正の優先順位はそれほど高くありません。 MyISAM テーブルについては、myisam_sort_buffer_size 変数に高い値を設定することによって、インデックスの再作成部分（再作成プロセスでもっとも処理が遅い部分）を迅速化することができます。

- ALTER TABLE を使用するためには、対象のテーブルに対する ALTER、INSERT、CREATE の各権限が必要。
- IGNORE は SQL-92 に対する MySQL の拡張。 IGNORE では、重複するユニークキーが新しいテーブルに存在する場合の ALTER TABLE の動作が制御される。 IGNORE を指定しない場合は、コピー処理が中断され、ロールバックされ

る。 IGNORE を指定すると、重複するユニークキーを持つレコードがある場合、最初のレコードのみが使用され、その他のレコードが削除される。

- 1 つの ALTER TABLE ステートメントで、複数の ADD、ALTER、DROP、CHANGE 節を発行できる。これは SQL-92 に対する MySQL の拡張。SQL-92 では、1 つの ALTER TABLE ステートメントでこれらのいずれか 1 つの節しか使用できない。
- CHANGE col_name, DROP col_name と DROP INDEX は SQL-92 に対する MySQL の拡張。
- MODIFY は ALTER TABLE に対する Oracle の拡張。
- オプションの語 COLUMN は純粋なノイズワードであり、省略可能。
- その他のオプションを指定しないで ALTER TABLE tbl_name RENAME TO new_name を使用すると、単にテーブル tbl_name に対応するファイルの名前が MySQL によって変更される。テンポラリテーブルを作成する必要はない。 See 項6.5.5.「RENAME TABLE 構文」。
- create_definition 節では、ADD および CHANGE 用に CREATE TABLE と同じ構文が使用される。注意: この構文には、カラム型だけでなく、カラム名が含まれる。 See 項6.5.3.「CREATE TABLE 構文」。
- カラム名の変更には、CHANGE old_col_name create_definition 節を使用できる。この変更を行うには、元のカラム名と新しいカラム名を指定し、さらに現在のこのカラムの型を指定する。たとえば、INTEGER 型のカラム a の名前を b に変更するには、次のようにする。

```
mysql> ALTER TABLE t1 CHANGE a b INTEGER;
```

CHANGE 構文では、カラムの名前ではなく型を変更する場合にも、元のカラム名と新しいカラム名の両方を (たとえ同じであっても) 指定する必要がある。 次に例を示す。

```
mysql> ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
```

ただし、MySQL バージョン 3.22.16a 以降では、MODIFY を使用することで、カラムの名前を変更することなく、カラムの型変更を実行できる。

```
mysql> ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
```

- カラムの一部に関するインデックスがある場合 (たとえば、VARCHAR 型カラムの最初の 10 文字のインデックスがある場合など) 、CHANGE または MODIFY を使用してそのカラムを短縮するときには、カラムの長さを、インデックスが作成されている文字の数より短くすることはできない。
- CHANGE または MODIFY を使用してカラムの型を変更する際には、MySQL によって、新しい型へのデータの変換ができる限り実行される。
- MySQL バージョン 3.22 以降では、FIRST または ADD ... AFTER col_name を使用して、テーブルレコード内の特定の位置にカラムを追加することができる。デフォルトでは、レコードの最後にカラムが追加される。 MySQL バージョン 4.0.1 以降では、CHANGE や MODIFY でも、FIRST および AFTER キーワードを使用できる。
- ALTER COLUMN では、カラムの新しいデフォルト値を指定するか、または以前のデフォルト値を削除できる。 以前のデフォルトを削除した場合、そのカラムで NULL の格納が可能なら、新しいデフォルト値は NULL になる。そのカラムで NULL の格納が不可能な場合は、項6.5.3.「CREATE TABLE 構文」 で説明しているデフォルト値が MySQL に

よって割り当てられる。

- DROP INDEX では、インデックスが削除される。これは、SQL-92 に対する MySQL の拡張。See 項6.5.8.「DROP INDEX 構文」。
- カラムがテーブルから破棄された場合、そのカラムは構成要素となっているすべてのインデックスからも削除される。インデックスを構成するすべてのカラムが破棄された場合は、そのインデックス自体も破棄される。
- テーブルにカラムが 1 つしか含まれない場合、そのカラムを破棄することはできない。 テーブルの削除を実行することが目的である場合は、カラムを破棄するのではなく、DROP TABLE を実行する。
- DROP PRIMARY KEY では、プライマリインデックスが破棄される。プライマリインデックスが存在しない場合は、そのテーブルの最初の UNIQUE インデックスが破棄される（明示的に指定された PRIMARY KEY がまったく存在しない場合は、MySQL によって、最初の UNIQUE キーが PRIMARY KEY としてマークされる）。

  テーブルに UNIQUE INDEX または PRIMARY KEY を追加すると、その値は非 UNIQUE なあらゆるインデックスの前に格納される。これは、MySQL で重複キーをできる限り迅速に検出できるようにするためである。
- ORDER BY では、レコードを特定の順序で並べた新しいテーブルを作成できる。注意: 挿入や削除を行った後は、テーブル内の元の順序は維持されない。場合によっては、後でテーブル内の順序付けの基準とするカラムに基づいて、テーブル内の順序を設定しておくと、MySQL でのソートがより容易化されることがある。このオプションは主に、レコードのクエリをたいてい特定の順序で行うことが明らかな場合に役立つ。テーブルを大幅に変更した後にこのオプションを使用することによって、パフォーマンスが良くなる場合がある。
- MyISAM テーブルに対して ALTER TABLE を使用すると、非ユニークなインデックスのすべてが別のバッチに作成される（REPAIR の場合と同様）。 インデックスが数多くある場合は、これによって ALTER TABLE の処理がはるかに迅速化される。
- MySQL 4.0 以降では、上記の機能を明示的に有効化することができる。 そのためには、ALTER TABLE ... DISABLE KEYS によって、MySQL による MyISAM テーブルの非ユニークなインデックスの更新を停止する。 その後、ALTER TABLE ... ENABLE KEYS によって、欠落しているインデックスを再作成する。MySQL において、この処理は 1 つずつキーを挿入する処理よりはるかに早い特殊アルゴリズムを使用して実行されるため、大量の挿入ではキーを無効化することによって処理が大幅に迅速化される。
- C API 関数 mysql_info() を使用すると、コピーされたレコードの数と、ユニークキー値の重複により削除されたレコードの数（IGNORE を指定した場合）を確認できる。
- ... ADD [CONSTRAINT [symbol]] FOREIGN KEY (...) REFERENCES ... (...) と ... DROP FOREIGN KEY ... をサポートしている InnoDB 型のテーブルを対象としている場合を除いて、FOREIGN KEY、CHECK、REFERENCES の各節では実際には何も行われない。 See 項7.5.5.2.「FOREIGN KEY 制約」。 他のテーブル型に関しては、この構文は互換性を確保する目的で提供されている。つまり、他の SQL サーバにコードを移植し、参照を含むテーブルを作成するアプリケーションを実行しやすくするためである。 See 項1.8.4.「MySQL と SQL-92 との違い」。
- ALTER TABLE では、テーブルオプション DATA DIRECTORY と INDEX DIRECTORY は無視される。
- CHAR 型、VARCHAR 型、TEXT 型のすべてのカラムを新しいキャラクタセットに変更するには（たとえば、MySQL 4.0.x から 4.1.1 にアップグレードした後などに）、次のようにする。

```
ALTER TABLE table_name CHARACTER SET character_set_name;
```

注意: 次のコマンドでは、テーブルの default character set しか変更されない。

```
ALTER TABLE table_name DEFAULT CHARACTER SET character_set_name;
```

default character set とは、テーブルに追加する (ALTER TABLE ... ADD column などで) 新しいカラムに対してキャラクタセットを指定しなかった場合に使用されるキャラクタセット。

以下に、ALTER TABLE に使用例をいくつか示します。まず、次のコマンドでテーブル t1 を作成するとします。

```
mysql> CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

このテーブルの名前を t1 から t2 に変更するには、次のようにします。

```
mysql> ALTER TABLE t1 RENAME t2;
```

カラム a を INTEGER から TINYINT NOT NULL に変更し (名前は変えずに)、さらにカラム b を CHAR(10) から CHAR(20) に変更し、かつこのカラムの名前を b から c に変更するには、次のようにします。

```
mysql> ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

d という名前を持つ TIMESTAMP 型の新しいカラムを追加するには、次のようにします。

```
mysql> ALTER TABLE t2 ADD d TIMESTAMP;
```

カラム d にインデックスを追加し、カラム a を主キーにするには、次のようにします。

```
mysql> ALTER TABLE t2 ADD INDEX (d), ADD PRIMARY KEY (a);
```

カラム c を削除するには、次のようにします。

```
mysql> ALTER TABLE t2 DROP COLUMN c;
```

c という名前を持つ、整数型の新しい AUTO_INCREMENT カラムを追加するには、次のようにします。

```
mysql> ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
       ADD INDEX (c);
```

注意： 上の例で c のインデックスを作成しているのは、AUTO_INCREMENT カラムにはインデックスが必要なためです。また、c を NOT NULL として宣言しているのは、インデックス付きカラムは値として NULL を取れないためです。

AUTO_INCREMENT カラムを追加すると、カラム値として連続番号が自動的に挿入されます。最初の連続番号を設定するには、ALTER TABLE の前に SET INSERT_ID=value を実行するか、または AUTO_INCREMENT=value テーブルオプションを指定します。 See 項5.5.6.「SET 構文」。

MyISAM テーブルでは、AUTO_INCREMENT カラムを変更しない限り、連続番号は影響されません。AUTO_INCREMENT カラムを破棄した後に別の AUTO_INCREMENT カラムを追加すると、再び 1 から採番されます。

See 項A.7.1.「ALTER TABLE の問題」。

6.5.5. RENAME TABLE 構文

```
RENAME TABLE tbl_name TO new_tbl_name[, tbl_name2 TO new_tbl_name2,...]
```

名前の変更は原子的に実行されます。つまり、テーブル名が変更されている間、他のスレッドからはこれらのテーブルのいずれにもアクセスできなくなります。それによって、テーブルを空のテーブルと置換することが可能になります。

:

```
CREATE TABLE new_table (...);
RENAME TABLE old_table TO backup_table, new_table TO old_table;
```

名前の変更は左から右へ実行されるため、2 つのテーブルの名前を交換する場合は、次のように記述する必要があります。

```
RENAME TABLE old_table    TO backup_table,
             new_table    TO old_table,
             backup_table TO new_table;
```

データベース名を変更することもできますが、その場合は、変更後のデータベースが変更前のデータベースと同じディスク上に存在していなければなりません。

```
RENAME TABLE current_db.tbl_name TO other_db.tbl_name;
```

ロックされたテーブルやアクティブなトランザクションがあると、RENAME は実行できません。また、元のテーブルに対する ALTER 権限と DROP 権限、新しいテーブルに対する CREATE 権限と INSERT 権限が必要です。

MySQL で複数テーブルの名前の変更時にエラーが発生した場合、名前を変更されたすべてのテーブルに対して逆方向の名前の変更処理が行われ、すべてが元の状態に戻されます。

RENAME TABLE は MySQL 3.23.23 で追加されました。

6.5.6. DROP TABLE 構文

```
DROP [TEMPORARY] TABLE [IF EXISTS] tbl_name [, tbl_name,...] [RESTRICT | CASCADE]
```

DROP TABLE では、1 つ以上のテーブルが削除されます。テーブルデータとテーブル定義のすべてが削除されるため、このコマンドは慎重に使用してください。

MySQL バージョン 3.22 以降では、キーワード IF EXISTS を使用することによって、指定したテーブルが存在しない場合に発生するエラーを回避できます。4.1 では、IF EXISTS を指定した場合、存在しないすべてのテーブルに関する NOTE が出力されます。 See 項4.6.8.9.「SHOW WARNINGS | ERRORS」。

RESTRICT と CASCADE は、移植を容易化するためのものです。 現在のところ、これらを指定しても何も行われません。

注意: DROP TABLE では、現在のアクティブなトランザクションが自動的にコミットされます (4.1 を使用していて、TEMPORARY キーワードを指定した場合を除く) 。

オプション TEMPORARY は、4.0 では無視されます。4.1 では、このオプションは次のように動作します。

- テンポラリテーブルの破棄のみ行う。
- 実行中のトランザクションは終了されない。
- アクセス権のチェックは行われない。

TEMPORARY は、実際のテーブルが誤って廃棄されないようにするための手段として役立ちます。

6.5.7. CREATE INDEX 構文

```
CREATE [UNIQUE|FULLTEXT] INDEX index_name
      ON tbl_name (index_col_name,...)

index_col_name:
       col_name [(length)] [ASC | DESC]
```

バージョン 3.22 より前の MySQL の場合、CREATE INDEX ステートメントでは何も実行されません。バージョン 3.22 以降では、CREATE INDEX は、インデックスを作成する ALTER TABLE ステートメントにマップされています。

See 項6.5.4.「ALTER TABLE 構文」。

通常、テーブルのインデックスはすべて、テーブル自体を CREATE TABLE で作成するときに一緒に作成します。 See 項6.5.3.「CREATE TABLE 構文」。 CREATE INDEX では、既存のテーブルにインデックスを追加することができます。

(col1,col2,...) 形式のカラムリストでは、複合インデックスが作成されます。インデックス値は、リストに指定したカラムの値を連結して作成されます。

CHAR 型と VARCHAR 型については、カラムの一部のみを使用するインデックスを作成できます。この場合、col_name(length) 構文を使用して、各カラム値の最初から length に指定した数のバイトのインデックスを作成します (BLOB 型と TEXT 型では、プリフィックスの長さを必ず指定する必要があります。length には 255 までの数値を指定できます) 。次のステートメントでは、name カラムの最初の 10 文字を使用したインデックスが作成されます。

```
mysql> CREATE INDEX part_of_name ON customer (name(10));
```

ほとんどの名前は最初の 10 文字が異なるため、このインデックスの場合、name カラム全体から作成したインデックスよりはるかに遅くなるということはありません。 また、カラムの一部でインデックスを作成するとインデックスファイルのサイズを大幅に削減できるため、ディスク領域が節約されるとともに、INSERT 操作が迅速化される場合があります。

注意: NULL 値を持てるカラムに対するインデックスの追加は、MySQL バージョン 3.23.2 以降で MyISAM、InnoDB、BDB のいずれかのテーブル型を使用している場合にのみ可能です。 BLOB 型や TEXT 型のカラムに対するインデックスの追加は、MySQL バージョン 3.23.2 以降で MyISAM または BDB のいずれかのテーブル型を使用している場合か、MySQL バージョン 4.0.14 以降で InnoDB テーブル型を使用している場合にのみ可能です。 BLOB 型や TEXT 型カラムのインデックスでは、プリフィックスの長さを必ず指定する必要があります。

index_col_name の指定では、最後に ASC または DESC を付けることができます。 これらのキーワードは、昇順または降順によるインデックス値の格納を指定できるようにする今後の拡張に対応するものです。現時点では、これらのキーワードは解析されても無視され、インデックス値は常に昇順で格納されます。

MySQL でのインデックスの使用方法の詳細については、項5.4.3.「MySQL でのインデックスの使用」 を参照してください。

FULLTEXT インデックスでは、MyISAM テーブルにおける CHAR 型、VARCHAR 型、TEXT 型のカラムに対してのみ、インデックスを作成することができます。FULLTEXT インデックスは MySQL バージョン 3.23.23 以降で使用できます。項6.8.「MySQL 全文検索」。

6.5.8. DROP INDEX 構文

```
DROP INDEX index_name ON tbl_name
```

DROP INDEX では、index_name に指定したインデックスが tbl_name に指定したテーブルから破棄されます。バージョン 3.22 より前の MySQL では、DROP INDEX では何の処理も行われません。バージョン 3.22 以降では、DROP INDEX はインデックスを破棄する ALTER TABLE ステートメントにマップされています。 See 項6.5.4.「ALTER TABLE 構文」。

6.6. MySQL 基本ユーザユーティリティコマンド

6.6.1. USE 構文

```
USE db_name
```

USE db_name ステートメントでは、db_name に指定したデータベースを、後続のクエリのデフォルトのデータベースとして使用するよう MySQL に指示することができます。指定したデータベースは、セッションの終了まで、または別の USE ステートメントを発行するまでカレントデータベースになります。

```
mysql> USE db1;
mysql> SELECT COUNT(*) FROM mytable;     # selects from db1.mytable
mysql> USE db2;
mysql> SELECT COUNT(*) FROM mytable;     # selects from db2.mytable
```

USE ステートメントで特定のデータベースをカレントにしても、それによって、他のデータベースのテーブルにアクセスできなくなるわけではありません。次の例では、db1 データベースの author テーブルと、db2 データベースの editor テーブルにアクセスします。

```
mysql> USE db1;
mysql> SELECT author_name,editor_name FROM author,db2.editor
    ->       WHERE author.editor_id = db2.editor.editor_id;
```

USE ステートメントは Sybase との互換性を確保するためのものです。

6.6.2. DESCRIBE 構文（カラムに関する情報の取得）

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

DESCRIBE は SHOW COLUMNS FROM の簡略形です。

See 項4.6.8.1.「データベース、テーブル、カラム、およびインデックスに関する情報の取得」。

DESCRIBE では、テーブルのカラムに関する情報が出力されます。col_name には、カラム名を指定する他に、SQL のワイルドカード文字 '%' と '_' を含む文字列を指定できます。この場合、この文字列に一致する名前を持つカラム名のみが出

力されます。文字列を引用符で工スケープする必要はありません。

カラム情報で示されたカラムの型は、そのカラムの CREATE TABLE ステートメントで指定したものと異なっている場合があります。これは、カラムの型が MySQL によって自動で変更されることがあるためです。 See 項6.5.3.1.「カラムの暗黙的な変更」。

このステートメントは Oracle との互換性を確保するためのものです。

SHOW ステートメントでも、同様の情報が出力されます。 See 項4.6.8.「SHOW 構文」。

6.7. MySQL トランザクションコマンドとロックコマンド

6.7.1. START TRANSACTION、COMMIT、ROLLBACK の各構文

デフォルトでは、MySQL は自動コミットモードで稼動します。この場合、テーブルを更新（変更）するステートメントを実行すると、MySQL によって直ちにその更新がディスクに格納されます。

トランザクションセーフテーブル（InnoDB や BDB）を使用している場合は、次のコマンドを使用して MySQL を非自動コミットモードに設定することができます。

```
SET AUTOCOMMIT=0
```

AUTOCOMMIT 変数をゼロに設定して自動コミットモードを無効にした後は、COMMIT を使用して変更内容をディスクに格納するか、または、トランザクションの開始以来行った変更を無視する場合は ROLLBACK を使用する必要があります。

ひと続きのステートメントのみに対して自動コミットモードを無効にするときには、START TRANSACTION ステートメントを使用できます。

:

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summmary=@A WHERE type=1;
COMMIT;
```

トランザクションを開始するときには、START TRANSACTION の代わりに BEGIN と BEGIN WORK も使用できます。START TRANSACTION は MySQL 4.0.11 で追加されました。これは SQL-99 の構文であり、トランザクションを随時開始するときにはこの構文が推奨されます。BEGIN は MySQL 3.23.17 以降で、BEGIN WORK は MySQL 3.23.19 以降で使用できます。

注意: 使用しているテーブルがトランザクションセーフテーブルでない場合は、自動コミットモードのステータスにかかわらず、すべての変更が直ちに格納されます。

非トランザクションテーブルを更新した後に ROLLBACK ステートメントを発行すると、エラー（ER_WARNING_NOT_COMPLETE_ROLLBACK）が警告として出力されます。トランザクションセーフテーブルはいずれもリストアされますが、非トランザクションセーフテーブルは変更されません。

START TRANSACTION または SET AUTOCOMMIT=0 を使用している場合は、以前の更新ログの変わりにMySQL バイナリログをバックアップ用に使用してください。トランザクションは COMMIT と同時にひとまとまりでバイナリログに格納されるため、ロールバックされたトランザクションが格納されることはありません。 See 項4.10.4.「バイナリログ」。

トランザクションの分離レベルを変更するには、SET TRANSACTION ISOLATION LEVEL を使用します。 See 項6.7.6.「SET TRANSACTION 構文」。

6.7.2. ロールバックできないステートメント

一部のステートメントはロールバックできません。通常、このようなステートメントとして、データベースの作成や破棄を行うステートメントや、テーブルの作成、破棄、変更を行うステートメントなどのデータ定義言語（DDL）ステートメントがあります。

場合によっては、このようなステートメントを組み込まないよう、トランザクションを設計する必要があります。 ロールバックできないステートメントをトランザクションの始めの方で発行したときに、その後別のステートメントでエラーが発生した場合、トランザクションの結果全体を ROLLBACK ステートメントでロールバックすることはできません。

6.7.3. 暗黙的なコミットを引き起こすステートメント

次のコマンドでは、トランザクションが（コマンドの実行前に COMMIT を発行した場合と同じように）暗黙的に終了します。

コマンド	コマンド	コマンド
ALTER TABLE	BEGIN	CREATE INDEX
DROP DATABASE	DROP INDEX	DROP TABLE
LOAD MASTER DATA	LOCK TABLES	RENAME TABLE
SET AUTOCOMMIT=1	START TRANSACTION	TRUNCATE

いずれかのテーブルが現在ロックされている場合は、UNLOCK TABLES でもトランザクションが終了します。MySQL 4.0.13 より前のバージョンでは、バイナリ更新ログが有効になっていると、CREATE TABLE でもトランザクションが終了します。

トランザクションはネストできません。これは、START TRANSACTION ステートメントやそのいずれかのシノニムの発行時に現在のトランザクションに対して実行される暗黙的な COMMIT の影響によるものです。

6.7.4. SAVEPOINT および ROLLBACK TO SAVEPOINT 構文

MySQL 4.0.14 および 4.1.1 以降では、InnoDB で SQL コマンド SAVEPOINT および ROLLBACK TO SAVEPOINT をサポートしています。

```
SAVEPOINT identifier
```

このステートメントでは、identifier に指定した名前を持つトランザクションセーブポイントが設定されます。現在のトランザクションに同名のセーブポイントがすでに存在する場合は、元のセーブポイントが削除され、新しいセーブポイントが設定されます。

```
ROLLBACK TO SAVEPOINT identifier
```

このステートメントでは、指定したセーブポイントまでトランザクションがロールバックされます。 セーブポイントの設定後にこのトランザクションでレコードに対して行った変更は、ロールバックによって取り消されますが、InnoDB では、

セーブポイントの後にメモリに格納された行ロックは解除されません (注意: 新たに挿入されたレコードについては、ロック情報はレコードに格納されたトランザクション ID によって渡されます。ロック自体が別にメモリに格納されることはありません。この場合、レコードのロックは取消し処理で解除されます)。 指定したセーブポイントより後に設定されたセーブポイントは削除されます。

コマンドで次のエラーが返された場合は、指定した名前のセーブポイントが存在しないことを意味します。

```
ERROR 1181: Got error 153 during ROLLBACK
```

セーブポイントを指定しないで COMMIT または ROLLBACK を実行すると、現在のトランザクションのセーブポイントがすべて削除されます。

6.7.5. LOCK TABLES および UNLOCK TABLES 構文

```
LOCK TABLES tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE}
        [, tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE} ...]
...
UNLOCK TABLES
```

LOCK TABLES では、現在のスレッドのテーブルがロックされます。UNLOCK TABLES では、現在のスレッドが保有しているロックが解除されます。現在のスレッドが別の LOCK TABLES を発行したり、サーバへの接続が閉じられると、現在のスレッドがロックしているテーブルのロックはすべて暗黙的に解除されます。

MySQL 4.0.2 で LOCK TABLES を使用するには、関連するテーブルに対するグローバルな LOCK TABLES 権限と SELECT 権限が必要です。MySQL 3.23 では、関連するテーブルに対する SELECT、insert、DELETE、UPDATE の各権限が必要です。

LOCK TABLES を使用する主な理由は、トランザクションをエミュレートしたり、テーブルの更新時に処理を迅速化したりするためです。これについては、詳しく後述します。

あるスレッドがテーブルに対する READ ロックを取得すると、そのスレッド (および他のすべてのスレッド) はそのテーブルからの読み取りのみ実行できるようになります。あるスレッドがテーブルに対する WRITE ロックを取得すると、ロックを保有しているスレッドだけがそのテーブルからの読み取りやテーブルへの書き込みを実行できるようになります。他のスレッドはブロックされます。

READ LOCAL と READ の違いは、READ LOCAL の場合、ロックの保有中に、コンフクリトを発生させない INSERT ステートメントを実行できることです。しかし、ロックの保有中にデータベースを MySQL の外部で操作しようとする場合は、この機能を使用できません。

LOCK TABLES の使用時には、使用するテーブルをすべてロックし、またクエリで使用するエイリアスと同じ名前を使用する必要があります。1 つのクエリで同じテーブルを何度も指定する (エイリアスを使用して) 場合は、各エイリアスに対してロックを取得しなければなりません。

更新をできるだけ早く処理するために、WRITE ロックは、通常、READ ロックより優先されます。そのため、あるスレッドが READ ロックを取得し、別のスレッドが WRITE ロックを要求している場合、後続の READ ロック要求は、WRITE スレッドがロックを取得し、その後そのロックを解除するまで待機します。LOW_PRIORITY WRITE ロックでは、そのスレッドが WRITE ロックを取得する前に、他のスレッドが READ ロックを取得することができます。その間、LOW_PRIORITY WRITE ロックを発行したスレッドは待機します。LOW_PRIORITY WRITE は、READ ロックを取得しようとするスレッドが皆無になるときがあるとわかっている場合にのみ使用してください。

LOCK TABLES は次のように機能します。

1. ロックするテーブルを内部定義された順序 (ユーザから見た場合は未定義) でソートする。
2. 読み取りロックと書き込みロックでテーブルがロックされる場合は、読み取りロックより書き込みロックを優先する。
3. 一度に 1 つのテーブルをロックし、スレッドがすべてのロックを取得するまでロック処理を繰り返す。

このポリシーによって、テーブルのロックでデッドロックの発生が回避されます。ただし、このスキーマに関しては注意すべき点が他にもあります。

MySQL でテーブルに対して LOW_PRIORITY WRITE ロックを使用した場合、そのスレッドは READ ロックを要求する他のスレッドがなくなるまでの間のみ待機し、READ を要求する他のスレッドがなくなった時点でロックを取得します。そのスレッドで WRITE ロックを取得した後に、ロックテーブルリストの次のテーブルのロックを取得しようと待機しているときには、他のスレッドの方がいずれも WRITE ロックが解除されるまで待つことになります。これによってアプリケーションで深刻な問題が発生する場合は、一部のテーブルをトランザクションセーフテーブルに変換することを検討してください。

テーブルのロックを取得しようと待機しているスレッドを、安全な方法で強制終了するには、KILL を使用します。 See 項 4.6.7.「KILL 構文」。

注意: INSERT DELAYED で使用しているテーブルは、ロックすべきではありません。なぜなら、この場合、INSERT が独立したスレッドで実行されるためです。

個々の UPDATE ステートメントでは、いずれも処理が原子的に行われるため、通常、テーブルをロックする必要はありません。現在実行中の SQL ステートメントが、他のスレッドによって妨害されることはまったくありません。しかし、次に示すように、テーブルをロックする必要が生じる場合もいくつかあります。

- 一連のテーブルに対して多くの操作を実行する場合、使用するテーブルをロックした方が処理がはるかに迅速になる。当然ながら、この場合の短所は、READ ロックされているテーブルについては、どのスレッドも (ロックを保有しているスレッドも含む) このテーブルを更新できなくなり、WRITE ロックされているテーブルについては、ロックを保有しているスレッド以外、どのスレッドもこのテーブルを読み取れなくなることである。

  LOCK TABLES の使用時にいくつかの面で処理が迅速になる理由は、MySQL でキーのキャッシュが UNLOCK TABLES が呼び出されるまでフラッシュされないためである (通常、キーのキャッシュは各 SQL ステートメントの後にフラッシュされる)。それによって、MyISAM テーブルに対する挿入、更新、削除処理が迅速化される。

- トランザクションをサポートしていないストレージエンジンを MySQL で使用している場合、SELECT と UPDATE の間に他のスレッドに割り込まれないようにするには、LOCK TABLES を使用する必要がある。次の例では、安全に処理を実行するために LOCK TABLES を発行する必要がある。

```
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;
```

  LOCK TABLES を発行しないと、SELECT ステートメントと UPDATE ステートメントの実行の間に別のスレッドが

trans テーブルに新しいレコードを挿入してしまう可能性がある。

多くの場合、自身の加算を行う更新 (UPDATE customer SET value=value+new_value) や LAST_INSERT_ID() 関数の使用により、LOCK TABLES の使用を避けることができます。

また、ユーザレベルのロック関数 GET_LOCK() と RELEASE_LOCK() を使用して問題を解決できる場合もあります。これらのロックはサーバのハッシュテーブルに保存され、高速にするため pthread_mutex_lock() と pthread_mutex_unlock() で実装されます。 See 項6.3.6.2.「その他の各種関数」。

ロックポリシーの詳細については、項5.3.1.「MySQL のテーブルロック方法」 を参照してください。

全データベースの全テーブルを読み取りロックでロックするには、FLUSH TABLES WITH READ LOCK コマンドを使用します。See 項4.6.4.「FLUSH 構文」。特定の時点のスナップショットを取ることができる、Veritas などのファイルシステムを使用している場合には、このコマンドがバックアップを作成するときに非常に役立ちます。

注意: LOCK TABLES はトランザクションセーフではありません。アクティブなトランザクションは、テーブルロックの試行前に暗黙的にコミットされます。

6.7.6. SET TRANSACTION 構文

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }
```

この構文では、トランザクションの分離レベルが、グローバルなセッション全体または次のトランザクションのどちらかとして設定されます。

デフォルトの動作では、次の (まだ開始されていない) トランザクションの分離レベルが設定されます。このステートメントに GLOBAL キーワードを使用すると、それ以降に作成されるすべての新しい接続 (既存の接続は対象外) に対してグローバルにデフォルトのトランザクションレベルが設定されます。 これを行うには、SUPER 権限が必要です。SESSION キーワードを使用すると、現在の接続で実行されるすべての新しいトランザクションに対してデフォルトのトランザクションレベルが設定されます。

InnoDB トランザクションの各分離レベルについては、項7.5.9.1.「InnoDB と SET ... TRANSACTION ISOLATION LEVEL ...」 を参照してください。MySQL 4.0.5 以降、InnoDB ではこれらの各レベルをサポートしています。デフォルトのレベルは REPEATABLE READ です。

mysqld のデフォルトのグローバル分離レベルは、--transaction-isolation=... で設定することができます。 See 項4.1.1.「mysqld コマンドラインオプション」。

6.8. MySQL 全文検索

```
MATCH (col1,col2,...) AGAINST (expr [IN BOOLEAN MODE | WITH QUERY EXPANSION] )
```

バージョン 3.23.23 以降、MySQL ではフルテキストインデックスと全文検索をサポートしています。 MySQL のフルテキストインデックスは FULLTEXT 型のインデックスです。 FULLTEXT インデックスは MyISAM テーブルにのみ使用され、CHAR 型、VARCHAR 型、TEXT 型のいずれかのカラムから作成することができます。この作成は CREATE TABLE 時に行えますが、ALTER TABLE か CREATE INDEX を使用して後から追加することも可能です。データセットのサイズが大きい場合は、FULLTEXT インデックスを持たないテーブルにデータをロードしてから、ALTER TABLE (または CREATE

INDEX) を使用してインデックスを作成する方が処理がはるかに迅速です。すでに FULLTEXT インデックスを持っているテーブルにデータをロードすると、処理がかなり遅くなることがあります。

全文検索を実行するには、MATCH() 関数を使用します。

```
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,'Optimizing 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)
```

MATCH() 関数では、テキストのコレクション (FULLTEXT インデックスに含まれる 1 つ以上のカラムのセット) に対して、文字列の自然言語検索が実行されます。検索文字列は AGAINST() の引数として指定します。検索はケース非依存方式で実行されます。 MATCH() からは、テーブルの各レコードについて、関連性を示す値 (つまり、検索文字列と、そのレコードの MATCH() リストに指定したカラムのテキストとの間の類似度) が返されます。

MATCH() を WHERE 節で使用すると (上の例を参照)、返されるレコードは関連性が最も高いレコードから低いレコードの順に自動でソートされます。 関連性を示す値は負の数でない浮動小数点数です。関連性がゼロのときは、類似性がまったくないことを意味します。関連性は、レコードに含まれるワード数、そのレコードに含まれる一意のワード数、コレクションに含まれる合計ワード数、特定のワードを含むドキュメント (レコード) 数に基づいて計算されます。

ブール値モードの検索も実行できます。これについては後述します。

上記の例は、MATCH() 関数の基本的な使用方法を示したものです。レコードは関連性が高いものから低いものの順に返されます。

次の例は、関連値を明示的に取り出す方法を示したものです。 WHERE 節も ORDER BY 節もないので、返されるレコードは順序付けられていません。

```
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)
```

次の例はより複雑です。このクエリでは、関連性が返され、関連性の高いものから低いものの順にレコードがソートされます。この結果を出力するためには、MATCH() を 2 回指定します。それによって追加のオーバーヘッドが発生することはありません。なぜなら、MySQL のオプティマイザは 2 つの MATCH() 呼び出しが同じものであると認識し、全文検索コードを 1 度しか起動しないためです。

```
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)
```

バージョン 4.1.1 以降、全文検索ではクエリの拡張 (特に、その異型の ``ブラインドクエリ拡張'' ) をサポートしています。通常、これは、検索語句が短すぎるときに役立ちます。検索語句が短い場合、その語句を指定したユーザが暗黙的な知識に頼っていることがよくあります。暗黙的な知識といったものは、通常、全文検索エンジンは備えていません。たとえば、ユーザが ``データベース'' という語句を検索している場合、実際にはそのユーザは、単に ``データベース'' だけでなく、``MySQL''、``Oracle''、``DB2''、``RDBMS'' といった語句もすべて ``データベース'' に一致し、返されるはずだと想定していることがあります。暗黙的な知識とは、このようなことを意味します。

ブラインドクエリ拡張 (自動関連性フィードバック) では、検索が 2 回実行されます。2 回目の検索の検索語句には、元の検索語句に、最初の検索で上位に検出された少数のドキュメントが結び付けられたものが使用されます。したがって、たとえば、これらのドキュメントの 1 つに ``データベース'' という語と ``MySQL'' という語が含まれている場合、2 回目の検索では、``MySQL'' という語を含み、``データベース'' という語は含まないドキュメントが検索されます。また、たとえば、Georges Simenonの著書で ``Maigret'' 警視に関する本を検索しようとするときに、``Maigret'' のスペルが正確にわからないとします。この場合、``Megre and the reluctant witnesses'' を検索語句として指定すると、クエリの拡張を使用しなければ ``Maigret and the Reluctant Witnesses'' しか検出されません。しかし、クエリの拡張を使用すると、2 回目の検索で ``Maigret'' という語が含まれるすべての本が検出されます。注意:ブラインドクエリ拡張では、関連しないドキュメントが返されることでノイズが大幅に増加しがちです。そのため、この機能を使用する意味があるのは、検索語句が比較的短い場合に限られます。

MySQL では、非常に単純なパーサを使用してテキストをワード (語) に分割します。``ワード'' とは、文字、数字、'''、'_' で構成される文字列です。ストップワードリストに含まれる ``ワード'' や短すぎるものは無視されます。全文検索で検出されるワードのデフォルトの最小長は 4 文字です。この長さは 項6.8.2.「MySQL 全文検索の調整」 の説明に従って変更可能です。

コレクションおよびクエリに含まれる正しい各ワードには、そのクエリまたはコレクションでのそのワードの重要度に基づいて重みが設定されます。そのため、多くのドキュメントに存在するワードは低く重み付けされます (重みがゼロの場

合もあります）。なぜなら、そのワードはそのコレクションにおいて意味値が低いためです。そのワードがまれにしか存在しない場合は、高く重み付けされます。その後、各ワードの重みが結合されてレコードの関連性が計算されます。

このようなテクニックは、サイズの大きなコレクションの場合に最も効果があります（実際に、それを目的として入念に調整されています）。小さいテーブルでは、ワードの分布はそれぞれの意味値を正しく反映するものとはならず、このモデルを使用した場合、奇妙な結果が出ることがあります。

```
mysql> SELECT * FROM articles WHERE MATCH (title,body) AGAINST ('MySQL');
Empty set (0.00 sec)
```

上の例では、MySQL というワードの検索で何も結果が生成されません。これは、このワードが半分以上のレコードに存在するためです。したがって、このワードは事実上ストップワード（意味値がゼロのワード）として扱われます。これは最も望ましい動作です。自然言語のクエリの場合 、1 GB のテーブルから 1 つおきにレコードが返されるのは適切ではありません。

ワードがテーブルの半分を占めるレコードに一致する場合、関連するドキュメントが検出される見込みはあまりありません。むしろ、無関係のドキュメントが大量に検出される可能性が多分にあります。 これは、検索エンジンを使用してインターネットで検索をするときに誰もが頻繁に経験することです。このようなレコードに該当のデータセットにおいて低い意味値が設定されている理由は、ここにあります。

バージョン 4.0.1 以降、MySQL では、IN BOOLEAN MODE 修飾子を使用してブール値の全文検索も実行できます。

```
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 | Optimizing 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 ...   |
+----+-----------------------------+---------------------------------------+
```

このクエリでは、MySQL というワードを含み（注意: 50% のしきい値は使用されません）、YourSQL というワードは含まないすべてのレコードが取り出されます。注意: ブール値モードの検索では、関連値の高いもの順のレコードの自動ソートは行われません。これは上のクエリの結果を見るとわかります。上のクエリでは、関連性が最も高いレコード（MySQL というワードを 2 つ含むレコード）が最初ではなく最後にリストされています。また、ブール値の全文検索は FULLTEXT インデックスがなくても機能します。ただし、この場合、処理速度は遅くなります。

ブール値の全文検索機能では、次の演算子をサポートしています。

- +

  先行するプラス記号は、返される各レコードにそのワードが存在しなければならないことを表す。

- -

  先行するマイナス記号は、返される各レコードにそのワードが存在してはならないことを表す。

- デフォルト（プラスもマイナスも指定しない場合）では、そのワードは必ずしも存在する必要はないが、そのワードを

含むレコードは高く評価される。これは、IN BOOLEAN MODE 修飾子を指定していない MATCH() ... AGAINST() の動作に類似する。

- < >

  これら 2 つの演算子は、各レコードに割り当てられる関連性の値に対する個々のワードの貢献度を変更するために使用される。< 演算子は貢献度を減少させ、> 演算子は貢献度を増加させる。 下の例を参照。

- ( )

  かっこは、ワードを副次式にグループ化するために使用される。

- ~

  先行するチルダは否定演算子として機能し、レコードの関連性に対するワードの貢献度をマイナスにする。これはノイズワードをマークするのに役立つ。このようなワードを含むレコードは、他のワードより低く評価されるが、- 演算子を使用したときのように完全に除外されるわけではない。

- *

  アスタリスクは切り捨て演算子。他の演算子と異なり、これはワードの前ではなく後ろに付ける。

- "

  二重引用符 " で囲んだ語句は、この語句とまったく同じ語句を含むレコードにのみ一致する。

次に、例をいくつか示します。

- apple banana

  これらのワードの最低 1 つを含むレコードが検索される。

- +apple +juice

  ... 両方のワードを含むレコードを検索。

- +apple macintosh

  ... ワード ``apple'' を含むレコードを検索。``apple'' に加えて ``macintosh'' というワードも含んでいれば、高く評価される。

- +apple -macintosh

  ... ワード ``apple'' を含み、``macintosh'' は含まないレコードを検索。

- +apple +(>turnover <strudel)

  ... ``apple'' と ``turnover''、または ``apple'' と ``strudel'' を含むレコード (2 つのワードの順序は問わない) を検索。ただし、``apple pie'' は ``apple strudel'' より高く評価される。

- apple*

  ... ``apple''、``apples''、``applesauce''、``applet'' はいずれもこれと一致する。

- "some words"

  ... ``some words of wisdom" とは一致するが、``some noise words" とは一致しない。

6.8.1. 全文検索における制約

- 全文検索は MyISAM テーブルでのみ可能。
- 全文検索は UCS-2 では使用できない (しかし、MySQL 4.1.1 以降では、UTF-8 で機能する) 。
- MATCH() 関数のパラメータはすべて、同じ FULLTEXT インデックスの一部を成す同じテーブルのカラムでなければならない (IN BOOLEAN MODE で MATCH() を実行する場合を除く) 。
- FULLTEXT インデックスのカラムはすべて同じキャラクタセットを使用していなければならない。
- MATCH() のカラムリストはテーブルの一部の FULLTEXT インデックス定義のカラムリストと正確に一致していなければならない (IN BOOLEAN MODE で MATCH() を実行する場合を除く) 。
- AGAINST() の引数は定数文字列でなければならない。

6.8.2. MySQL 全文検索の調整

残念ながら、ユーザによる調整が可能な全文検索のパラメータは現在のところ少ししかありません。しかし、調整可能なパラメータの追加は、TODO リストにおいて優先度の高い位置にランクされています。MySQL ソースディストリビューション (see 項2.3.「MySQL ソースディストリビューションのインストール」) がある場合は、全文検索の動作をより詳細に制御できます。

注意: 全文検索は検索の効率を良くするよう入念に調整されています。デフォルトの動作を変更すると、多くの場合、結果的に検索結果が悪くなります。MySQL のソースは、十分な知識がない限り変更しないでください。

以下に説明するフルテキスト変数は、サーバの起動時に設定されていなければなりません。サーバの稼動中にこれらの変数を動的に変更することはできません。

- インデックスを作成するワードの最小長は、MySQL 変数 ft_min_word_len で定義される。 See 項4.6.8.4.「SHOW VARIABLES」。 (この変数は MySQL バージョン 4.0 以降でのみ使用できる) 。 デフォルト値は 4 文字。 この値を必要な値に変更して、FULLTEXT インデックスを再ビルドする。 たとえば、3 文字のワードを検索可能にするには、オプションファイルに以下の行を書き込むことによってこの変数を設定する。

  ```
  [mysqld]
  ft_min_word_len=3
  ```

  その後、サーバを再起動して、FULLTEXT インデックスを再ビルドする。

- ストップワードリストは、ft_stopword_file 変数に指定したファイルからロードすることができる。 See 項4.6.8.4.「SHOW VARIABLES」。 ストップワードリストを変更した後、FULLTEXT インデックスを再ビルドする (この変数は MySQL バージョン 4.0.10 以降でのみ使用できる) 。
- 50% のしきい値は、選択されている特定の重み付け設定で決められている。 これを無効にするには、myisam/ftdefs.h

の次の行を変更する。

```
#define GWS_IN_USE GWS_PROB
```

次のように変更する。

```
#define GWS_IN_USE GWS_FREQ
```

その後、MySQL を再コンパイルする。 この場合、インデックスを再ビルドする必要はない。 注意: この設定によって、MATCH() 関数で適切な関連性を示す値が得られるようにする MySQL の機能が大幅に弱くなる。 実際にこのような一般的な語を検索する必要がある場合は、むしろ、IN BOOLEAN MODE を使用して検索を行う方が良い。この場合、50% のしきい値が考慮されない。

- 検索エンジンの管理者が、ブール値のフルテキスト検索に使用する演算子を変更したくなることもある。これらの演算子は ft_boolean_syntax 変数で定義される。 See 項4.6.8.4.「SHOW VARIABLES」。 しかし、この変数は読み取り専用であり、値は myisam/ft_static.c に設定されている。

FULLTEXT インデックスの再構築が必要になるフルテキスト関連の変更の場合、MyISAM テーブルに対してこれを行う最も簡単な方法は、インデックスファイルを再ビルドする次のステートメントを使用することです。

```
mysql> REPAIR TABLE tbl_name QUICK;
```

6.8.3. 全文検索に関連する TODO 項目

- FULLTEXT インデックスを使用するすべての操作の迅速化。
- 近接演算子。
- "常にインデックスワード" のサポート。これは、"C++"、"AS/400"、"TCP/IP" など、ユーザがワードとして認識させたい任意の文字列である。
- MERGE テーブルでの全文検索のサポート。
- UCS-2 のサポート。
- ストップワードリストをデータの言語に依存させる。
- ステミング (= 語幹の抽出。言うまでもなく、データの言語に依存する抽出) 。
- ユーザによる提供が可能な汎用 UDF プリパーサ。
- モデルのよりいっそうの柔軟化 (CREATE/ALTER TABLE の FULLTEXT に調整可能なパラメータをいくつか追加する) 。

6.9. MySQL クエリキャッシュ

バージョン 4.0.1 以降、MySQL サーバ には Query Cache 機能があります。 クエリキャッシュの使用時、このキャッシュには、SELECT クエリのテキストと、クライアントに送られたその結果が格納されます。 後でまったく同じクエリを受け取ると、サーバはそのクエリの解析と実行をもう一度繰り返す代わりに、クエリキャッシュから結果を取り出します。

注意:クエリキャッシュから古いデータが返されることはありません。データが変更されると、クエリキャッシュの関連するエントリがすべてフラッシュされます。

（一部の）テーブルがそれほど頻繁には変更されず、同じクエリが何度も実行される環境では、クエリキャッシュが非常に役立ちます。 動的コンテンツを大量に持つ多くの Web サーバでは、このような状況が一般的です。

クエリキャッシュのパフォーマンスに関するデータの一部を、以下に示します（これらの結果は、2 GB の RAM、64 MB のクエリキャッシュを搭載する Linux Alpha 2 x 500 MHz での MySQL ベンチマークスィートの実行により生成されたものです）：

- 実行しているクエリがすべて単純なもの（レコードが 1 つしかないテーブルからレコードを 1 つ選択するなど）でありながら互いに異なるために、クエリをキャッシュすることができない場合、クエリキャッシュをアクティブにしておくことによるオーバーヘッドは 13%。 これは最悪の場合のシナリオとみなすことができる。現実には、クエリはこの例よりもはるかに複雑なため、通常オーバーヘッドはかなり低くなる。
- レコードが 1 つだけのテーブルでの 1 つのレコードの検索は 238% 迅速化される。 これは、キャッシュに格納されているクエリで想定される迅速化の最小値に近い数値である。
- クエリキャッシュのコードを無効にするには、query_cache_size=0 として設定する。 クエリキャッシュコードを無効にすると、目立ったオーバーヘッドはなくなる（クエリキャッシュは、コンフィギャオプション --without-query-cache を使用してコードから除外できる）。

6.9.1. クエリキャッシュの動作

クエリは解析前に比較されるため、

```
SELECT * FROM tbl_name
```

と

```
Select * from tbl_name
```

は、クエリキャッシュで別のクエリとみなされます。完全に一致する（各バイトが）クエリ以外、同一とはみなされません。 また、たとえば、あるクライアントで新しい形式の通信プロトコルを使用している場合や、別のクライアントが使用しているものとは異なるキャラクタセットを使用している場合も、同じものであるはずのクエリが異なるものとして認識されることがあります。

異なるデータベースを使用するクエリや、使用プロトコルのバージョンが異なるクエリ、またデフォルトのキャラクタセットが異なるクエリは、いずれも異なるクエリとして認識され、別々にキャッシュされます。

キャッシュは SELECT SQL_CALC_FOUND_ROWS ... および SELECT FOUND_ROWS() ... 型のクエリでも機能します。これは、検出されたレコードの数もキャッシュに格納されるためです。

クエリの結果がクエリキャッシュから返された場合、ステータス変数 Com_select の値は増加しませんが、Qcache_hits の値は増加します。 See 項6.9.4.「クエリキャッシュのステータスと保守」。

テーブルが変更されると（INSERT、UPDATE、DELETE、TRUNCATE、ALTER、DROP TABLE|DATABASE のいずれかによって）、そのテーブルを（場合によっては、MRG_MyISAM テーブルを通して）使用したキャッシュ内のクエリはすべて無効になり、キャッシュから削除されます。

変更された InnoDB トランザクションテーブルは、COMMIT が実行されると無効化されます。

MySQL 4.0 では、クエリキャッシュはトランザクション内では無効です (結果は返されません) 。MySQL 4.1.1 以降では、InnoDB テーブルの使用時、クエリキャッシュはトランザクション内でも機能します (テーブルのバージョン番号に基づいて、データがまだカレントかどうかが検出されます) 。

MySQL 5.0 より前のバージョンでは、先行するコメントで始まるクエリの場合、キャッシュへの格納は可能でも、キャッシュから取得できませんでした。この問題は MySQL 5.0 で修正されています。

次の関数のいずれかを含んでいるクエリは、キャッシュできません。

関数	関数	関数
User-Defined Functions	CONNECTION_ID	FOUND_ROWS
GET_LOCK	RELEASE_LOCK	LOAD_FILE
MASTER_POS_WAIT	NOW	SYSDATE
CURRENT_TIMESTAMP	CURDATE	CURRENT_DATE
CURTIME	CURRENT_TIME	DATABASE
ENCRYPT (パラメータを 1 つ持つもの)	LAST_INSERT_ID	RAND
UNIX_TIMESTAMP (パラメータなし)	USER	BENCHMARK

また、ユーザ変数を含んでいるクエリ、mysql システムデータベースを参照しているクエリ、SELECT ... IN SHARE MODE、SELECT ... INTO OUTFILE ...、SELECT ... INTO DUMPFILE ... のいずれかの形式のクエリ、および SELECT * FROM AUTOINCREMENT_FIELD IS NULL 形式のクエリ (最後に挿入された ID の取り出し - ODBC 回避策) についても、キャッシュできません。

ただし、FOUND_ROWS() は、先行するクエリがキャッシュからフェッチされたものであっても正しい値を返します。

クエリでテーブルを何も使用しない、またはテンポラリテーブルを使用する場合や、関連するテーブルのいずれかに対するカラムのアクセス権限をユーザが持っていない場合、クエリはキャッシュされません。

MySQL では、クエリをクエリキャッシュから読み取る前に、関連するすべてのデータベースとテーブルに対する SELECT 権限をユーザが持っているかチェックします。この権限をユーザが持っていない場合、キャッシュに格納されている結果は使用されません。

6.9.2. クエリキャッシュの設定

mysqld の起動時には、コマンドラインで mysqld の少数の MySQL システム変数がクエリキャッシュにより追加されます。これらのシステム変数はオプション設定ファイルで設定することができます。

- query_cache_limit これより大きい結果はキャッシュしない (デフォルト 1M) 。
- query_cache_min_res_unit

  この変数はバージョン 4.1 以降で利用できる。

クエリの結果（クライアントにも送られるデータ）は結果の取り出し時にクエリキャッシュに格納される。したがって、通常、データは 1 つの大きなまとまりとして処理されるのではない。クエリキャッシュでは、データを格納するブロックが要求に応じて割り当てられる。1 つのブロックがいっぱいになると、新しいブロックが割り当てられる。 メモリ割り当て処理はコストが高い（時間的に）ため、クエリキャッシュでのブロックの割り当ては、query_cache_min_res_unit に指定された最小サイズで行われる。 クエリが実行されると、最後の結果ブロックは実際のデータサイズに切り捨てられ、使用していないメモリが解放される。

- query_cache_min_res_unit のデフォルト値は 4 KB。ほとんどの場合、これで十分である。
- 結果が小さいクエリが数多くある場合は、デフォルトのブロックサイズを使用するとメモリがフラグメント化することがある（これは空きブロック（Qcache_free_blocks）の数が多くなることでわかる。フラングメント化すると、メモリ不足によってキャッシュからクエリが削除される（Qcache_lowmem_prunes）ことがある）。その場合は、query_cache_min_res_unit の値を小さくする。
- 結果が大きいクエリが大半を占める場合は（Qcache_total_blocks および Qcache_queries_in_cache を参照）、query_cache_min_res_unit の値を大きくすることによって、パフォーマンスを良くすることができる。ただし、あまり大きくしすぎないよう注意すること（上記を参照）。

- query_cache_size 以前のクエリの結果を格納するために割り当てるメモリ量（バイト単位で指定）。この値を 0 にすると、クエリキャッシュは無効（デフォルト）になる。
- query_cache_type 次のいずれかの値（数値のみ）を設定できる。

オプション	説明
0	（OFF - キャッシュへの格納、結果の取り出しをいずれも行わない）
1	（ON。SELECT SQL_NO_CACHE ... クエリを除くすべての結果をキャッシュする）
2	（DEMAND。SELECT SQL_CACHE ... クエリのみキャッシュする）

クエリキャッシュの動作は、スレッド（接続）内でデフォルトから変更することができます。構文は次のとおりです。

QUERY_CACHE_TYPE = OFF | ON | DEMAND QUERY_CACHE_TYPE = 0 | 1 | 2

オプション	説明
0 または OFF	キャッシュへの格納、結果の取り出しをいずれも行わない。
1 または ON	SELECT SQL_NO_CACHE ... クエリを除くすべての結果をキャッシュする。
2 または DEMAND	SELECT SQL_CACHE ... クエリのみキャッシュする。

6.9.3. SELECT でのクエリキャッシュオプション

SELECT クエリでは、クエリキャッシュ関連の次の 2 つのパラメータを指定することができます。

オプション	説明
SQL_CACHE	QUERY_CACHE_TYPE が DEMAND の場合、クエリをキャッシュ可能にする。 QUERY_CACHE_TYPE が ON の場合、これがデフォルト。 QUERY_CACHE_TYPE が OFF の

	場合、何もしない。
SQL_NO_CACHE	このクエリをキャッシュ不可にする。このクエリをキャッシュに格納しない。

6.9.4. クエリキャッシュのステータスと保守

FLUSH QUERY CACHE コマンドでは、クエリキャッシュをデフラグメント化して、メモリの使用効率を良くすることができます。このコマンドを発行しても、キャッシュからクエリが削除されることはありません。 FLUSH TABLES でも、クエリキャッシュがフラッシュされます。

RESET QUERY CACHE コマンドでは、すべてのクエリ結果がクエリキャッシュから削除されます。

使用している MySQL バージョンにクエリキャッシュがあるかどうかは、次のコマンドで確認できます。

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
1 row in set (0.00 sec)
```

クエリキャッシュのパフォーマンスは、SHOW STATUS で監視できます。

変数	説明
Qcache_queries_in_cache	キャッシュに登録されているクエリ数
Qcache_inserts	キャッシュに追加されたクエリ数
Qcache_hits	キャッシュヒット数
Qcache_lowmem_prunes	メモリ不足のためにキャッシュから削除されたクエリ数
Qcache_not_cached	キャッシュされていない (キャッシュ不可、または QUERY_CACHE_TYPE により) クエリ数
Qcache_free_memory	クエリキャッシュの空きメモリ量
Qcache_free_blocks	クエリキャッシュ内の空きメモリブロック数
Qcache_total_blocks	クエリキャッシュ内の合計ブロック数

合計クエリ数 = Qcache_inserts + Qcache_hits + Qcache_not_cached

クエリキャッシュでは可変長ブロックが使用されるため、Qcache_total_blocks と Qcache_free_blocks によって、クエリキャッシュメモリのフラグメント化が示されることがあります。 FLUSH QUERY CACHE を実行すると、1 つの (大きな) 空きブロックだけが残ります。

注意:各クエリには、最低でも 2 ブロック必要です (クエリテキスト用に 1 ブロック、クエリ結果用に 1 ブロック以上) 。また、クエリで使用される各テーブル用にも 1 ブロック必要です。ただし、複数のクエリで同じテーブルを使用している場合は、1 ブロックの割り当てですみます。

クエリのキャッシュサイズは、Qcache_lowmem_prunes ステータス変数に基づいて調整できます。この値は、新しいクエ

リを格納できるようメモリを解放するためにキャッシュから削除されたクエリ数を示します。クエリキャッシュでは、使用されたうち最も古い (LRU) 方針に従って、キャッシュから削除するクエリが決定されます。

第7章 MySQL のテーブル型

MySQL バージョン 3.23.6 より、3 種類の基本テーブル形式 (ISAM、HEAP、MyISAM) を選択できるようになりました。これより新しい MySQL のバージョンでは、コンパイルの方法に応じて追加のテーブル型 (InnoDB または BDB) をサポートしています。1 つのデータベースに異なる型のテーブルを収容することができます。

新しいテーブルを作成するときに、そのテーブルの型を MySQL に通知できます。通常、デフォルトのテーブル型は MyISAM です。

MySQL では、テーブル定義とカラム定義を保持する .frm ファイルが必ず作成されます。テーブルのインデックスとデータは、テーブル型に応じて、このファイル以外の 1 つ以上のファイルに格納されます。

コンパイルまたはアクティブ化されていないテーブル型を使用しようとすると、MySQL によってそのテーブル型の代わりに MyISAM 型のテーブルが作成されます。この動作は、さまざまなテーブル型をサポートする MySQL サーバ間でテーブルをコピーする場合に便利です (ほとんどの場合、マスタサーバは安全性を高めるためにトランザクションストレージエンジンをサポートし、スレーブサーバは処理速度を高めるために非トランザクションストレージエンジンのみをサポートしています) 。

MySQL の初心者は、このテーブル型の自動変更に戸惑うかもしれません。この点については、バージョン 4.1 で新しいクライアント/サーバプロトコルに警告を導入し、テーブル型が自動変更される際に警告を生成する方法で対応する予定です。

ALTER TABLE ステートメントを使用すれば、テーブルを別の型に変換できます。 See 項6.5.4.「ALTER TABLE 構文」。

MySQL では 2 種類のテーブルをサポートしていることに注意してください。1 つはトランザクションセーフのテーブル (InnoDB と BDB) 、もう 1 つは非トランザクションセーフのテーブル (HEAP、ISAM、MERGE、MyISAM) です。

トランザクションセーフのテーブル (TST) には次の利点があります。

- 安全性が高い。MySQL のクラッシュやハードウェア障害が発生した場合でも、自動リカバリによって、またはバックアップとトランザクションログからデータを復元できる。
- 多数のステートメントを組み合わせ、それらすべてを COMMIT コマンドで一括して受け付けられる。
- ROLLBACK を実行して変更を無効にできる (オートコミットモードで実行していない場合) 。
- 更新が失敗した場合は、すべての変更がリストアされる (NTST テーブルでは、行われたすべての変更が確定される) 。
- テーブルで読み取りと同時に多数の更新が行われる場合に、優れた並行処理を実現する。

InnoDB テーブルを使用するには、少なくとも innodb_data_file_path 起動オプションを使用する必要があることに注意してください。 See 項7.5.3.「InnoDB 起動オプション」。

非トランザクションセーフのテーブル (NTST) には次の利点があります。

- トランザクションのオーバーヘッドがないため、処理がはるかに迅速。

- トランザクションのオーバヘッドがないため、使用するディスク領域が少なくて済む。
- 少ないメモリで更新を実行できる。

同じステートメントで TST テーブルと NTST テーブルを組み合わせると、両方の利点を活かすことができます。

7.1. MyISAM テーブル

MyISAM は、MySQL バージョン 3.23 でのデフォルトのテーブル型です。この型は ISAM コードに基づいており、多数の便利な拡張機能を備えています。

インデックスは .MYI (MYIndex) 拡張子の付いたファイルに、データは .MYD (MYData) 拡張子の付いたファイルにそれぞれ格納されます。MyISAM テーブルは、myisamchk ユーティリティで検査および修復することができます。See 項 4.5.6.7.「myisamchk を使用したクラッシュのリカバリ」。 また、MyISAM テーブルを myisampack で圧縮することで、使用する領域を大幅に削減できます。 See 項4.8.4.「myisampack (MySQL 圧縮読み取り専用テーブルジェネレータ)」。

MyISAM の新機能は次のとおりです。

- MyISAM ファイルには、テーブルが正しく閉じられたかどうかを示すフラグがある。mysqld を起動する際に --myisam-recover を指定すると、MyISAM テーブルを開く際にそのテーブルが正しく閉じられたかどうかが自動的に検査され、必要であれば修復される。
- データファイル内に空きブロックがないテーブルに対し、他のスレッドがそのテーブルから読み取りを行うのと同時に新しいレコードを INSERT できる (同時挿入)。空きブロックは、大量のデータを含んだ可変長レコードに含まれるデータの長さが短くなるような更新をした場合、またはレコードを削除した場合に発生する。すべての空きブロックを使い切ると、それ以後の挿入は再び同時挿入になる。
- 大きなファイルをサポートするファイルシステム/オペレーティングシステムで、大きなファイル (63 ビット) がサポートされる。
- すべてのデータが下位バイトから順に格納される。これによって、データがマシンと OS に依存しなくなる。バイナリ移植性を実現するための唯一の要件は、2 の補数で表現された符号付き整数 (すべてのマシンで過去 20 年間使われていた形式) および IEEE 浮動小数点形式 (同じく主流のマシンでの主要な形式) をマシンで使用することである。マシンにおいてバイナリ互換性をサポートしていない可能性がある領域は、埋め込みシステムのみ (特殊なプロセッサを使用している場合があるため)。

  データを下位バイトから先に格納しても、速度上大きな問題はない。テーブル行内のバイトは通常整列されておらず、未整列のバイトを順番に読み取る操作は、逆順に読み取る操作に比べてそれほど処理能力を必要としないからである。また、実際のカラムの値を読み取るコードも他のコードに比べて速度が重視されない。
- すべての数値キーを上位バイトから先に格納して、インデックスの圧縮効率を高めている。
- 1 つの AUTO_INCREMENT カラムを内部処理している。MyISAM では、このカラムが INSERT/UPDATE で自動更新される。AUTO_INCREMENT の値は、myisamchk でリセットできる。これによって AUTO_INCREMENT カラムの処理が速くなる (最低でも 10%)。また、以前の ISAM のように古い番号が再使用されない。マルチパートキーの最後の項目に AUTO_INCREMENT が定義されている場合は、以前の動作が引き続き有効となることに注意する。
- キーツリーは、ソートされた順序で挿入されると (AUTO_INCREMENT カラムを使用しているときなど)、分割され

て上位ノードにキーが 1 つだけ含まれるようになる。これによって、キーツリーでの領域利用率が向上する。

- BLOB カラムと TEXT カラムにインデックスを作成できる。
- インデックスが作成されたカラムで NULL 値が許可される。この場合、1 キー当たり 0 ー 1 バイトが使用される。
- 最大キー長のデフォルト値は 500 バイト (再コンパイルで変更可能) 。長さが 250 バイトを超えるキーの場合は、デフォルトの 1,024 バイトより大きなキーブロックサイズが使用される。
- テーブル当たりの最大キー数のデフォルト値は 32。この数値は、myisamchk を再コンパイルしなくても 64 まで拡大できる。
- myisamchk を --update-state 付きで実行すると、テーブルが検査済みとしてマークされる。 myisamchk --fast は、このマークがないテーブルのみを検査する。
- myisamchk -a は、キーの各部分 (ISAM のようにキー全体だけではなく) に関する統計情報を格納する。
- 更新/挿入と削除が混在している場合に、可変長レコードがフラグメント化されることが少なくなった。これは、隣接する削除済みブロックの結合、および次のブロックが削除されている場合のブロックの拡張が自動的に行われるようになったためである。
- myisampack で、BLOB および VARCHAR カラムをパックできる。
- データファイルとインデックスファイルを別々のディレクトリに配置して速度を高めることができる (CREATE TABLE の DATA/INDEX DIRECTORY="path" オプションを使用) 。 See 項6.5.3.「CREATE TABLE 構文」。

MyISAM は、MySQL で近い将来使用可能となる次の機能もサポートしています。

- 真の VARCHAR 型のサポート。VARCHAR カラムは、長さを示す 2 バイトの値で始まる。
- VARCHAR を使用するテーブルには、固定長と可変長のレコードを収容できる。
- VARCHAR および CHAR は 64K まで可能。すべてのキーセグメントには、それぞれ独自の言語定義がある。これによって、MySQL ではカラムごとに言語定義を変えられる。
- 計算されたハッシュインデックスを UNIQUE インデックスとして使用できる。これによって、テーブル内のカラムの任意の組み合わせを UNIQUE インデックスにすることができる (ただし、計算された UNIQUE インデックスでの検索は不可能) 。

通常、インデックスファイルは ISAM よりも MyISAM の方がはるかに小さいので注意してください。つまり、MyISAM は一般に ISAM よりも少ないシステムリソースを使用しますが、圧縮されたインデックスへデータを挿入する際により多くの CPU 時間を必要とします。

次に示す mysqld のオプションを使用して、MyISAM テーブルの動作を変更することができます。 See 項4.6.8.4.「SHOW VARIABLES」。

オプション	説明
--myisam-recover=#	クラッシュしたテーブルの自動リカバリ。
-O myisam_sort_buffer_size=#	テーブルをリカバリする際に使用されるバッファ。

--delay-key-write=ALL	すべての MyISAM テーブルに対して、書き込み間でキーバッファをフラッシュしない。
-O myisam_max_extra_sort_file_size=#	速度は遅くても安全なキーキャッシュインデックス作成方法をどの時点で使用するかを MySQL が判断できるようにする。注意: このパラメータを指定する単位として、4.0.3 より前はメガバイト、このバージョンからはバイトを使用する。
-O myisam_max_sort_file_size=#	テンポラリファイルがこの値を超えた場合に、作成されたインデックスに対して高速なソートインデックス方法を使用しない。注意: このパラメータを指定する単位として、4.0.3 より前はメガバイト、このバージョンからはバイトを使用する。
-O bulk_insert_buffer_size=#	バルク挿入の最適化で使用されるツリーキャッシュのサイズ。注意: これはスレッド当たりの制限値。

--myisam-recover=# を指定して mysqld を起動すると、自動リカバリがアクティブ化されます。See 項4.1.1.「mysqld コマンドラインオプション」。テーブルが開く際に検査されます。検査の内容は、テーブルにクラッシュのマークが付いているかどうか、またはテーブルのオープンカウント変数が 0 ではなく、かつ --skip-external-locking で実行しているかどうかです。上記のどちらかが当てはまる場合は、次の処理が行われます。

- テーブルにエラーがないかどうかが検査される。
- エラーが見つかった場合は、テーブルの高速修復 (ソートは行うがデータファイルは再構築しない) を試みる。
- データファイルにエラーがあるために (重複キーエラーなど) 修復が失敗した場合は、もう一度修復を試みるが、今度はデータファイルを再構築する。
- この修復が失敗した場合は、以前の修復方法 (ソートせずに 1 レコードずつ書き込む方法) で再度修復を試みる。この方法で、どのようなエラーでもわずかなディスク容量で修復できるはずである。

myisam-recover のオプションとして FORCE を指定しなかった場合に、直前に完了したステートメントからすべてのレコードをリカバリできないときは、自動修復が中止され、エラーファイルに次のエラーメッセージが書き込まれます。

```
Error: Couldn't repair table: test.g00pages
```

FORCE オプションを指定していた場合は、上記のメッセージの代わりに次の警告がエラーファイルに書き込まれます。

```
Warning: Found 344 of 354 rows when repairing ./test/g00pages
```

注意: BACKUP オプションを指定して自動リカバリを実行する場合は、tablename-datetime.BAK のような名前のファイルをデータベースディレクトリからバックアップメディアに自動的に移動する cron スクリプトを用意する必要があることに注意してください。

See 項4.1.1.「mysqld コマンドラインオプション」。

7.1.1. キーに必要な領域

MySQL ではさまざまなインデックスをサポートしていますが、一般に使用されるのは ISAM または MyISAM です。これ

らは B ツリーインデックスを使用します。このインデックスファイルのサイズは、すべてのキーについて (キー長+4)/0.67 を計算し、それを合計することで大まかに算出できます (これは、すべてのキーがソートされた順に挿入され、かつキーが一切圧縮されないという、最悪のケースを想定しています)。

文字列インデックスでは空白が圧縮されます。インデックスの最初の部分が文字列の場合は、プリフィックスも圧縮されます。文字列カラムに含まれる後続の空白が長い場合、またはそのカラムが VARCHAR カラムであるためにその長さがフルに使用されることがない場合は、空白の圧縮によってインデックスファイルが上記の数値よりも小さくなります。プリフィックスの圧縮は、文字列で始まるキーで使用されます。同一のプリフィックスを持つ文字列が多数存在する場合は、プリフィックスの圧縮が役立ちます。

MyISAM テーブルでは、テーブル作成時に PACK_KEYS=1 を指定することで、数値のプリフィックスを圧縮することもできます。この機能は、数値が上位バイトから順に格納される場合に、同一のプリフィックスを持つ整数キーが多数あるときに役立ちます。

7.1.2. MyISAM テーブル形式

MyISAM は、3 種類のテーブル型をサポートします。そのうち 2 つは、使用しているカラムの型に応じて自動的に選択されます。3 番目の圧縮テーブルは、myisampack ツールによってのみ作成されます。

BLOB 値を持たないテーブルを CREATE または ALTER する際に、ROW_FORMAT=# テーブルオプションを使用してテーブル形式を強制的に DYNAMIC または FIXED に設定できます。将来的には、ALTER TABLE に ROW_FORMAT=compressed | default を指定することで、テーブルを圧縮/展開できるようになります。 See 項6.5.3.「CREATE TABLE 構文」。

7.1.2.1. 静的 (固定長) テーブルの特性

これはデフォルトの形式です。この形式は、テーブルに VARCHAR、BLOB、または TEXT 型のカラムが含まれていない場合に使用されます。

この形式は最も単純かつ最も安全です。また、ディスク上の形式としては最も高速です。速度が速いのは、ディスク上で簡単にデータを検出できるためです。検索の対象がインデックスと静的形式を使用している場合、操作はきわめて単純です。単にレコードの番号にレコードの長さを掛けるだけです。

また、テーブルをスキャンする際にも、1 回のディスク読み取りで一定数のレコードを簡単に読み取ることができます。

固定サイズの MyISAM ファイルへ書き込んでいるときにコンピュータがクラッシュした場合の保全性も証明されています。この場合、myisamchk によって各レコードの開始位置と終了位置が簡単に割り出されます。したがって、通常は、部分的に書き込まれたレコードを除くすべてのレコードを回復できます。MySQL ではすべてのインデックスをいつでも再構築できることに注意してください。

- CHAR、NUMERIC、DECIMAL の各カラムは、そのカラム長までの残りの部分が空白で埋められる。
- きわめて高速。
- キャッシュが容易。
- レコードが固定位置にあるため、クラッシュ後の再構築が容易。
- myisamchk で再編成する必要がない。ただし、多数のレコードを削除したために空いたディスク領域をオペレーティングシステムに戻す場合を除く。

- 通常は動的テーブルよりも多くのディスク領域を必要とする。

7.1.2.2. 動的テーブルの特性

この形式は、テーブルに VARCHAR、BLOB、または TEXT カラムが含まれている場合、あるいはテーブルが ROW_FORMAT=dynamic で作成された場合に使用されます。

この形式は少し複雑です。各レコードにそれぞれの長さを記録したヘッダが必要となるからです。1 つのレコードが、更新によって長くなったために、複数の場所に存在することになる可能性もあります。

OPTIMIZE table または myisamchk を使用して、テーブルをデフラグメント化することができます。VARCHAR または BLOB カラムと同じテーブル内に、頻繁にアクセス/変更する静的データがある場合は、フラグメント化を回避するために動的なカラムを他のテーブルに移動するとよいでしょう。

- 文字列カラムはすべて動的である (長さが 4 バイト未満のものを除く) 。
- 各レコードの先頭には、どの文字列カラムが空白 ('') で、どの数値カラムがゼロであるかを示すビットマップが付いている (NULL 値を含んだカラムとは異なる) 。文字列カラムで後続の空白を取り除いた後の長さがゼロになった場合、または数値カラムの値がゼロである場合は、それらのカラムがビットマップでマークされ、ディスクに保存されない。空白でない文字列は、長さが記録されたバイトに文字列の内容を付加して保存される。
- 通常は、使用するディスク領域が固定長テーブルに比べてはるかに少ない。
- 各レコードは、要求されただけの領域を使用する。レコードが大きくなると、必要に応じてそのレコードが断片に分割される。その結果、レコードのフラグメント化が生じる。
- レコードの更新によってレコードの長さが拡張されると、そのレコードがフラグメント化される。この場合は、myisamchk -r をときどき実行して、パフォーマンスを高める必要がある。一部の統計情報には、myisamchk -ei tbl_name を使用する。
- レコードが細かくフラグメント化され、リンク (フラグメント) が失われている可能性があるため、クラッシュ後の再構築は容易ではない。
- 動的なサイズのレコードについては、予期されるレコードの長さを次の方法で算出できる。

```
3
+ (フィールド数 + 7) / 8
+ (char カラムの数)
+ 数値カラムをパックしたサイズ
+ 文字列の長さ
+ (NULL カラムの数 + 7) / 8
```

各リンクには 6 バイトのペナルティがある。動的レコードは、更新によってレコードが拡張されるたびにリンクされる。新しいリンクは少なくともそれぞれ 20 バイトあるため、次回の拡張は同じリンクで行われると考えられる。そうでない場合は、新たなリンクが発生する。リンクの数は、myisamchk -ed でチェックできる。すべてのリンクを削除するには、myisamchk -r を使用する。

7.1.2.3. 圧縮テーブルの特性

これは、オプションの myisampack ツール（ISAM テーブルでは pack_isam ）で生成される読み取り専用テーブルです。

- MySQL ディストリビューションは、MySQL が GPL 版になる前のものも含めて、いずれも myisampack で圧縮されたテーブルを読み取ることができる。
- 圧縮テーブルはごくわずかなディスク領域しか使用しない。ディスク使用量が最小限に抑えられるため、遅いディスク（CD-ROM など）を使用する場合に適している。
- 各レコードは別々に圧縮される（アクセスオーバヘッドがきわめて少ない）。レコードのヘッダは、テーブルで最も大きなレコードに応じて固定される（1 〜 3 バイト）。各カラムは異なる方法で圧縮される。圧縮タイプには次のものがある。
  - 通常は、カラムごとに異なるハフマンテーブルが存在する。
  - サフィックス空白の圧縮。
  - プリフィックス空白の圧縮。
  - 値 0 の数値は 1 ビットで格納する。
  - 値の範囲が小さい整数カラムは、可能な限り小さな型を使って格納する。たとえば、BIGINT カラム（8 バイト）のすべての値が 0 〜 255 の範囲内にある場合は、このカラムを TINYINT カラム（1 バイト）として格納する。
  - カラムの可能値が少ない場合は、カラムの型を ENUM に変換する。
  - 上記の圧縮を組み合わせて使用することもできる。
- 固定長または可変長のレコードを処理できる。
- myisamchk で圧縮を解除できる。

7.1.3. MyISAM テーブルの問題

MySQL がデータの格納に使用するファイル形式は広範な検査を受けていますが、データベーステーブルの破損を招きかねない状況は常に存在します。

7.1.3.1. MyISAM テーブルが破損した場合

MyISAM は信頼性の高いテーブル形式ですが（テーブルに対するすべての変更は SQL ステートメントから制御が戻る前に書き込まれます)、それでも以下の状況が発生した場合はテーブルが破損するおそれがあります。

- 書き込みの途中で mysqld プロセスが強制終了された場合。
- コンピュータが予期せずシャットダウンされた場合（コンピュータの電源が切られた場合など）。
- ハードウェアエラー。
- 稼働中のテーブルで外部プログラム（myisamchk など）を使用した場合。
- MySQL または MyISAM コードのソフトウェアバグ。

テーブルが破損すると、一般に次のような現象が見られます。

- テーブルからデータを選択するときに、Incorrect key file for table: '...'. Try to repair it というエラーが表示される。
- クエリがテーブルでレコードを検出できない、または不完全なデータを返す。

テーブルが破損していないかどうかは、CHECK TABLE コマンドで確認できます。 See 項4.5.4.「CHECK TABLE 構文」。

破損したテーブルは、REPAIR TABLE で修復できます。 See 項4.5.5.「REPAIR TABLE 構文」。 また、mysqld が稼働していないときに、myisamchk コマンドを使ってテーブルを修復することもできます。 myisamchk syntax。

テーブルが大きく破損している場合は、原因を突き止める必要があります。 See 項A.4.1.「MySQL が何度もクラッシュする場合に行うこと」。

この場合に最も重要なのは、mysqld が強制終了されたときにテーブルが破損したのかを確認することです (これは、mysqld エラーファイルに restarted mysqld という行が最近記録されたかどうかをチェックすることで簡単に検証できます)。これが該当しない場合は、その破損のテストケースを作成してみる必要があります。 See 項E.1.6.「テーブルが破損した場合にテストケースを作成する」。

7.1.3.2. クライアントがテーブルを使用している、またはテーブルを適切に閉じていない

各 MyISAM .MYI ファイルのヘッダには、テーブルが適切に閉じられているかをチェックするためのカウンタがあります。

CHECK TABLE または myisamchk から次の警告が返されることがあります。

```
# clients is using or hasn't closed the table properly
```

これは、このカウンタがずれていることを意味します。テーブルの破損を意味しているわけではありませんが、少なくともテーブルを検査して問題がないことを確認する必要があります。

カウンタの仕組みは次のとおりです。

- MySQL でテーブルが最初に更新されるときに、インデックスファイルのヘッダ内にあるカウンタが増加する。
- その後の更新では、カウンタは変更されない。
- FLUSH によって、またはテーブルキャッシュに空きがないために、テーブルの最後のインスタンスが閉じられると、それまでにテーブルが一箇所でも更新されていればカウンタが減少する。
- テーブルを修復するか、テーブルを検査して問題がなかった場合は、カウンタがゼロにリセットされる。
- テーブルを検査する他のプロセスとの相互作用に伴う問題を回避するため、カウンタがゼロである場合は、テーブルを閉じる際にもカウンタが減少しない。

つまり、カウンタがずれる可能性があるのは、次の場合に限られます。

- MyISAM テーブルが LOCK および FLUSH TABLES を使わずにコピーされた。

- 更新されてから閉じられるまでの間に MySQL がクラッシュした（ただし、MySQL は各ステートメントで生じたすべての書き込みを次のステートメントまでに発行するため、テーブルが無事である可能性もある）。
- mysqld が使用していたテーブルで、第三者が myisamchk --recover または myisamchk --update-state を実行した。
- ある mysqld サーバが使用しているテーブルに対し、別の mysqld サーバが REPAIR または CHECK を実行した。この場合、CHECK は実行しても問題ない（ただし他のサーバから警告を受ける）が、REPAIR は避ける必要がある。現時点では、REPAIR を実行するとデータファイルが新しいファイルで置換され、それが他のサーバに通知されないからである。

7.2. MERGE テーブル

MERGE テーブルは、MySQL バージョン 3.23.25 で新たに導入されました。コードはまだガンマ版ですが、比較的安定しているはずです。

MERGE テーブル（MRG_MyISAM テーブルとも呼ばれます）は、1 つのテーブルとして使用できる同一の MyISAM テーブルの集合です。テーブルの集合には、SELECT、DELETE、UPDATE のみを実行できます。MERGE テーブルに対して DROP を実行すると、MERGE の仕様のみが破棄されます。

WHERE なしで DELETE FROM merge_table を使用すると、テーブルに対するマッピングのみが消去され、マップされたテーブルの内容は削除されないことに注意してください（これは 4.1 で修正する予定です）。

同一のテーブルとは、すべてのテーブルが同一のカラムおよびキー情報で作成されていることを意味します。カラムのパック方法が異なるテーブル、保持するカラムがまったく同じでないテーブル、あるいはキーの順序が異なるテーブルはマージできません。ただし、一部のテーブルは myisampack で圧縮できます。 See 項4.8.4.「myisampack（MySQL 圧縮読み取り専用テーブルジェネレータ）」。

MERGE テーブルを作成すると、.frm テーブル定義ファイルおよび .MRG テーブルリストファイルが作成されます。.MRG には、1 つのインデックスファイルとして使用される複数のインデックスファイル（.MYI ファイル）のリストのみが含まれています。4.1.1 より前のバージョンでは、使用されるすべてのテーブルを MERGE テーブルと同じデータベースに配置する必要がありました。

今のところ、MERGE テーブルにマップするテーブルに対しては、SELECT、UPDATE、DELETE の各特権が必要です。

MERGE テーブルには、次のような効果があります。

- 一連のログテーブルを簡単に管理できる。たとえば、各月のデータを別々のファイルに収録し、それらの一部を myisampack で圧縮した後に、MERGE を作成してそれらを 1 つのファイルとして使用することができる。
- 処理速度が速くなる。大きな読み取り専用テーブルをいくつかの基準に基づいて分割し、分割した各部分を別々のディスクに配置できる。この場合、MERGE テーブルを作成すれば、大きなテーブルを使用する場合に比べてはるかに処理が速くなる（RAID を使用した場合も同じ効果が得られる）。
- より効率的に検索できる。処理の対象を正確に把握していれば、一部のクエリに対して分割されたテーブルの 1 つで検索を行い、その他のクエリに対して MERGE を使用することができる。多数のさまざまな MERGE テーブルをアクティブにすることもできる。この場合、ファイルが重複していてもかまわない。
- より効率的に修復できる。1 つの大きなファイルを修復するより、MERGE ファイルにマップされたファイルを個別に修復する方が簡単である。

- 多数のファイルを 1 つのファイルとして即座にマップできる。MERGE テーブルは、個々のテーブルのインデックスを使用する。MERGE テーブル独自のインデックスを保持する必要はない。このため、MERGE テーブルの集合の作成または再マップにはほとんど時間がかからない。MERGE テーブルを作成するときは、キー定義を指定する必要があることに注意する。
- 一連のテーブルをオンデマンドまたはバッチで大きなテーブルに結合する場合は、代わりにそれらのテーブルに対してオンデマンドで MERGE テーブルを作成した方がよい。この方が時間がかからず、ディスク領域も大幅に節約できる。
- オペレーティングシステムのファイルサイズ制限を回避できる。
- 1 つのテーブル上で MERGE を使用するだけで、テーブルのエイリアス/シノニムを作成できる。これによってパフォーマンスが特に影響を受けることはない（読み取りのたびに数回の間接的な呼び出しと memcpy() の呼び出しが発生するのみ）。

MERGE テーブルの欠点は次のとおりです。

- MERGE テーブルには同一の MyISAM テーブルしか使用できない。
- REPLACE を使用できない。
- MERGE テーブルはより多くのファイル記述子を使用する。10 個のテーブルをマップする MERGE テーブルを 10 人のユーザが使用している場合は、10 10 + 10 個のファイル記述子を使用することになる（10 人のユーザに 10 個ずつのデータファイルと、10 個の共有インデックスファイル）。
- キーの読み取りが遅い。キーに対して読み取りを行うと、MERGE ストレージエンジンは構成要素であるべてのテーブルに対して読み取りを発行して、指定されたキーに最も一致するテーブルをチェックする。その後に "read-next" を実行すると、MERGE ストレージエンジンは読み取りバッファを検索して次のキーを検出する。1 つのキーバッファがすべて使われるまでストレージエンジンは次のキーブロックを読み取らない。このため、eq_ref 検索では MERGE キーの処理にかなり時間がかかるが、ref 検索ではそれほど時間がかからない。 See 項5.2.1.「EXPLAIN 構文（SELECT に関する情報の取得）」。
- "開いている" MERGE テーブルによってマップされているテーブルには、DROP TABLE、ALTER TABLE、WHERE 節なしの DELETE FROM table_name、REPAIR TABLE、TRUNCATE TABLE、OPTIMIZE TABLE、または ANALYZE TABLE を実行できない。これを実行すると、MERGE テーブルが元のテーブルを参照するおそれがあり、予期しない結果を得ることがある。この問題を最も簡単に回避するには、FLUSH TABLES コマンドを発行して "開いている" MERGE テーブルを残さないようにする。

MERGE テーブルを作成するときに、1 つにまとめたいテーブルを UNION=(list-of-tables) で指定する必要があります。オプションとして、MERGE テーブルへの挿入が UNION リスト内の最初のテーブルと最後のテーブルのどちらで行われるかを、INSERT_METHOD で指定できます。INSERT_METHOD を指定しなかった場合、または NO を指定した場合は、MERGE テーブルに対するすべての INSERT コマンドでエラーが返されます。

次の例は、MERGE テーブルの使い方を示しています。

```
CREATE TABLE t1 (a INT NOT NULL AUTO_INCREMENT PRIMARY KEY, message CHAR(20));
CREATE TABLE t2 (a INT NOT NULL 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 NOT NULL AUTO_INCREMENT, message CHAR(20), KEY(a))
        TYPE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;
SELECT * FROM total;
```

total テーブルではキーが一意にならないため、このテーブルでは UNIQUE または PRIMARY KEY を作成していないことに注意してください。

MySQL サーバの外部から直接 .MRG ファイルを操作することもできます。

```
shell> cd /mysql-data-directory/current-database
shell> ls -1 t1.MYI t2.MYI > total.MRG
shell> mysqladmin flush-tables
```

これで次のような操作を実行できるようになります。

```
mysql> SELECT * FROM total;
+---+---------+
| a | message |
+---+---------+
| 1 | Testing |
| 2 | table   |
| 3 | t1      |
| 1 | Testing |
| 2 | table   |
| 3 | t2      |
+---+---------+
```

注意: a カラムは、PRIMARY KEY として宣言されていますが、実際には一意ではないことに注意してください。MERGE テーブルでは、これを構成する一連の MyISAM テーブル全体にわたる一意性を確保できないからです。

MERGE テーブルをマップし直すには、次のいずれかの操作を実行します。

- テーブルに対して DROP を実行し、テーブルを再作成する。
- ALTER TABLE table_name UNION=(...) を使用する。
- .MRG ファイルを変更し、MERGE テーブルとこれを構成するすべてのテーブルに対して FLUSH TABLE を発行することで、ストレージエンジンが新しい定義ファイルを読み取るようにする。

7.2.1. MERGE テーブルの問題

次に挙げるのは、MERGE テーブルに関する既知の問題です。

- MERGE テーブルでは、テーブル全体で UNIQUE 制約を保持できない。INSERT を実行すると、INSERT_METHOD=xxx に応じてデータが最初または最後のテーブルに挿入される。挿入先の MyISAM テーブルではそのデータが一意であることが保証されるが、このテーブルは他の MyISAM テーブルについてまったく関知しない。
- WHERE なしで DELETE FROM merge_table を使用すると、テーブルに対するマッピングのみが消去され、マップされたテーブルの内容は一切削除されない。

- アクティブな MERGE テーブルで使用されているテーブルに対して RENAME TABLE を実行すると、テーブルが破損するおそれがある。これは MySQL 4.1.x で修正される予定。
- MERGE 型のテーブルを作成するときに、その構成要素となるテーブルの型に互換性があるか、またはそのテーブル自体が存在するかがチェックされない。MERGE テーブルが使用される際に、マップされたテーブル間でレコード長が等しいかどうかが MySQL によって簡単にチェックされるが、これでは完全な検証にならない。

  このような方法で MERGE をテーブルを使用していると、高い確率で未知の問題が発生する。
- ALTER TABLE を使用して MERGE テーブルで使用されているテーブルに UNIQUE インデックスを追加した後に、再び ALTER TABLE を使用して MERGE テーブルに通常のインデックスを追加すると、以前の一意でないキーがテーブルに存在していた場合に、テーブルに対するキーの順序が変わる。これは、重複キーをできるだけ早く検出できるように、ALTER TABLE が UNIQUE キーを通常のキーの前に配置するためである。
- Windows では、MERGE テーブルが使用しているテーブルに対して DROP TABLE を実行できない。これは、MERGE ストレージエンジンが実行するテーブルマッピングを MySQL の上位レイヤが認識しないためである。Windows では開いているファイルを破棄できないため、対象のテーブルを破棄する前に、FLUSH TABLES ですべての MERGE テーブルをフラッシュするか、MERGE テーブルを破棄する必要がある。これについては、ビューを導入するときに修正する。

7.3. ISAM テーブル

MySQL バージョン 5.0 では、ISAM テーブル型が廃止されます。MySQL 4.1 では、このテーブル型がソースに含まれていてもコンパイルできません。このテーブルハンドラの実装としては MyISAM の方が優れているため、できるだけ早く ISAM テーブルを MyISAM テーブルに変換する必要があります。

ISAM は、B ツリーインデックスを使用します。インデックスは .ISM 拡張子の付いたファイルに保存され、データは .ISD 拡張子の付いたファイルに保存されます。ISAM テーブルは、isamchk ユーティリティで検査および修復することができます。 See 項4.5.6.7.「myisamchk を使用したクラッシュのリカバリ」。

ISAM には、次の機能と特性があります。

- 圧縮された固定長のキー。
- 固定および可変のレコード長。
- 16 個のキーと 1 キー当たり 16 個の部品。
- 最大キー長は 256 バイト (デフォルト) 。
- マシンの形式でデータを格納するため、高速だがマシンと OS に依存する。

MyISAM テーブルに当てはまることは、ほとんどの場合 ISAM テーブルにも当てはまります。See 項7.1.「MyISAM テーブル」。MyISAM テーブルとの主な相違点は次のとおりです。

- ISAM テーブルでは、OS/プラットフォームをまたがってバイナリを移植できない。
- 4 GB を超えるテーブルを処理できない。

- 文字列でのプリフィックスの圧縮のみをサポートする。
- キーの制限値が小さい。
- 動的なテーブルがより頻繁にフラグメント化される。
- テーブルの圧縮に、myisampack ではなく pack_isam が使用される。

ISAM テーブルを MyISAM テーブルに変換して mysqlcheck などのユーティリティを使用できるようにするには、ALTER TABLE ステートメントを使用します。

```
mysql> ALTER TABLE tbl_name TYPE = MYISAM;
```

組み込み式の MySQL では、ISAM テーブルがサポートされません。

7.4. HEAP テーブル

HEAP テーブルは、ハッシュインデックスを使用し、メモリに格納されます。これによって処理は速くなりますが、MySQL がクラッシュすると、このテーブルに格納されたすべてのデータが失われます。HEAP は、テンポラリテーブルとして非常に便利です。

MySQL の内部 HEAP テーブルは、オーバフローエリアなしの 100% 動的ハッシュを使用します。フリーリスト用の余分な領域は必要ありません。また、HEAP テーブルでは、ハッシュテーブルで一般に見られる削除 + 挿入に伴う問題も起こりません。

```
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;
```

次に、HEAP テーブルを使用する際の考慮事項を示します。

- 誤ってメモリを使い切ることがないように、CREATE ステートメントには必ず MAX_ROWS を指定する。
- インデックスには = と <=> しか使用できない（その代わりきわめて高速）。
- HEAP テーブルでは、レコードの検索にキー全体を使用するしかない。一方 MyISAM テーブルでは、キーの任意のプリフィックスを使ってレコードを検索できる。
- HEAP テーブルは、固定レコード長形式を使用する。
- HEAP は、BLOB カラムと TEXT カラムをサポートしない。
- HEAP は、AUTO_INCREMENT カラムをサポートしない。
- MySQL 4.0.2 より前のバージョンでは、HEAP が NULL カラムのインデックスをサポートしない。
- HEAP テーブルでは一意でないキーを使用できる（ハッシュテーブルではあまり使用されない）。
- HEAP テーブルは、他のテーブルと同様にすべてのクライアント間で共有される。

- 次のエントリを順番に検索できない（つまり、インデックスを使って ORDER BY を実行できない）。
- HEAP テーブルのデータは、小さなブロックで割り当てられる。テーブルは 100% 動的である（挿入時）。オーバフローエリアも余分なキースペースも必要ない。削除されたレコードはリンクされたリストに保存され、新しいデータをテーブルに挿入する際に再使用される。
- 同時に使用するすべての HEAP テーブルに対して十分な追加メモリが必要。
- メモリを解放するには、DELETE FROM heap_table、TRUNCATE heap_table、または DROP TABLE heap_table を実行する必要がある。
- MySQL では、2 つの値の間に存在するおおよそのレコード数を確認できない（この値は、使用するインデックスを範囲オプティマイザが決定するために使用する）。MyISAM テーブルを HEAP テーブルに変更すると、この点が一部のクエリに影響する可能性がある。
- 誤った操作を行わないように、max_heap_table_size より大きなサイズの HEAP テーブルは作成できないようになっている。

HEAP テーブルで 1 つのレコードに必要なメモリは、次のように計算します。

```
SUM_OVER_ALL_KEYS(キーの最大長 + sizeof(char*) * 2)
+ ALIGN(レコードの長さ+1, sizeof(char*))
```

sizeof(char*) は、32 ビットマシンでは 4、64 ビットマシンでは 8 です。

7.5. InnoDB テーブル

7.5.1. InnoDB テーブルの概要

InnoDB は、MySQL における、コミット、ロールバック、クラッシュリカバリの各機能を備えたトランザクションセーフ（ACID 準拠）のストレージエンジンです。InnoDB は、行レベルでロックを行い、SELECT ステートメントで Oracle 式の非ロックの読み取り一貫性(consistent read)を実現します。これらの機能によって、マルチユーザでの並行性とパフォーマンスが向上します。InnoDB ではロックエスカレーションが不要です。InnoDB での行レベルロックはわずかなスペースしか使用しないからです。InnoDB は、MySQL で FOREIGN KEY 制約を最初にサポートしたストレージエンジンです。

InnoDB は、大容量のデータを処理する際に最大限のパフォーマンスを実現するように設計されています。その CPU 効率は、おそらく他のディスクベースのリレーショナルデータベースエンジンのどれよりも優れています。

InnoDB は、高いパフォーマンスを必要とする多くの大規模データベースサイトで実際に使用されています。有名なインターネットニュースサイトの Slashdot.org は、InnoDB 上で稼働しています。Mytrix, Inc. では、1 TB を超えるデータを InnoDB に格納し、別のサイトでは InnoDB で 1 秒間に平均 800 件の挿入/更新を処理しています。

技術的には、InnoDB は MySQL のデータベースバックエンドです。InnoDB は、データとインデックスをキャッシュするための専用のバッファプールをメインメモリに持っています。InnoDB のテーブルとインデックスはテーブルスペースに格納されます。これは、複数のファイル（またはローデバイス）で構成されている場合があります。この点は、各テーブルを個別のファイルに格納する MyISAM テーブルなどとは異なっています。InnoDB テーブルは、ファイルサイズが 2 GB に制限されているオペレーティングシステム上でも、任意のサイズにすることができます。

InnoDB に関する最新情報は、http://www.innodb.com/ で参照できます。InnoDB マニュアルの最新版もこのサイトに収録

されています。

InnoDB は、MySQL と同じ GNU GPL License Version 2 (1991 年 6 月付け) の下でリリースされています。MySQL/InnoDB を配布する場合に、アプリケーションが GPL ライセンスの制約を満たしていないときは、https://order.mysql.com/?sub=pg&pg_no=1 から MySQL Pro の商用ライセンスを購入する必要があります。

7.5.2. MySQL バージョン 3.23 での InnoDB

MySQL バージョン 4.0 より、InnoDB はデフォルトで MySQL に組み込まれています。次の情報は、3.23 シリーズだけに該当します。

InnoDB テーブルは、3.23.34a より MySQL ソースディストリビューションに含まれるようになりました。 ただし 3.23 シリーズでは、InnoDB は MySQL-Max バイナリディストリビューションのみに組み込まれています。Windows の場合は、-Max バイナリが標準のバイナリディストリビューションに含まれています。

InnoDB をサポートする MySQL のバイナリバージョンをダウンロードした場合は、MySQL のマニュアルに従って MySQL のバイナリバージョンをインストールしてください。すでに MySQL-3.23 をインストールしている場合に MySQL-Max を最も簡単にインストールするには、サーバの実行ファイル mysqld を、-Max ディストリビューションの実行ファイルで置き換えます。MySQL と MySQL-Max は、サーバの実行ファイルの名前のみが異なります。 See 項2.2.9.「MySQL バイナリディストリビューションのインストール」。 See 項4.8.5.「mysqld-max (拡張 mysqld サーバ) 」。

InnoDB をサポートする MySQL をコンパイルするには、MySQL-3.23.34a 以降のバージョンを http://www.mysql.com/ からダウンロードし、--with-innodb オプションで MySQL をコンフィギャします。MySQL ソースディストリビューションのインストールについては、MySQL のマニュアルを参照してください。 See 項2.3.「MySQL ソースディストリビューションのインストール」。

```
cd /path/to/source/of/mysql-3.23.37
./configure --with-innodb
```

MySQL-Max-3.23 で InnoDB テーブルを使用するには、オプション設定ファイル my.cnf (Windows の場合は my.ini でも可) の [mysqld] セクションで設定パラメータを指定しなければなりません。

3.23 では、少なくとも innodb_data_file_path でデータファイルの名前とサイズを指定する必要があります。my.cnf で innodb_data_home_dir を指定しない場合は、デフォルトで MySQL の datadir にこれらのファイルが作成されます。innodb_data_home_dir を空の文字列として指定すれば、innodb_data_file_path でデータファイルへの絶対パスを指定できます。

これを最小限の方法で変更するには、[mysqld] セクションに次の行を追加します。

```
innodb_data_file_path=ibdata:30M
```

ただし、高いパフォーマンスを得るには、推奨された方法でオプションを指定するのが最善です。 See 項7.5.3.「InnoDB 起動オプション」。

7.5.3. InnoDB 起動オプション

MySQL バージョン 3.23 で InnoDB テーブルを有効にする方法については、項7.5.2.「MySQL バージョン 3.23 での InnoDB」 を参照してください。

MySQL-4.0 では、InnoDB テーブルを有効にするために特に何かをする必要はありません。

MySQL-4.0 および MySQL-4.1 のデフォルトの動作として、MySQL の datadir に自動拡張する 10 MB の ibdata1 ファイルが 1 つと、5 MB の ib_logfile ログファイルが 2 つ作成されます（MySQL-4.0.0 および 4.0.1 のデータファイルは 64 MB で、自動拡張しません）。

注意: 高いパフォーマンスを得るには、この後の例に示されている InnoDB の各種パラメータを明示的に設定する必要があります。

InnoDB テーブルを使用しない場合は、MySQL オプション設定ファイルに skip-innodb オプションを指定して InnoDB を動作させないようにできます。

```
[mysqld]
skip-innodb
```

バージョン 3.23.50 および 4.0.2 より、innodb_data_file_path 行の最後のデータファイルを、自動拡張ファイルとして指定できるようになりました。その場合の innodb_data_file_path の構文は次のとおりです。

```
innodb_data_file_path = データファイルのパス:サイズ;データファイルのパス:サイズ;...
... ;データファイルのパス:サイズ[:autoextend[:max:サイズ]]
```

最後のデータファイルに autoextend オプションを指定すると、テーブルスペースに空きがなくなった場合に、InnoDB が最後のデータファイルを拡張します。1 回の増分は 8 MB です。次に例を示します。

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:100M:autoextend
#
# 注意: innodb_data_home_dir に空文字を指定した場合、
# innodb_data_file_path に与えるパスは絶対パスになる
```

この例では、InnoDB に対し、初期サイズが 100 MB のデータファイルを 1 つだけ作成し、スペースが足りなくなった場合に 8 MB 単位で拡張するように指定しています。ディスクがいっぱいになった場合は、たとえば別のディスクに新たなデータファイルを追加することもできます。その場合は、autoextend が指定されている ibdata1 のサイズを確認し、そのサイズが 1,024 1,024 バイト（= 1 MB）の倍数になるように丸めた値を計算し、計算で得られた値を innodb_data_file_path の ibdata1 のサイズとして明示的に指定する必要があります。これで、新たなデータファイルを ibdata1 の後ろに追加できます。

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:988M;/disk2/ibdata2:50M:autoextend
#
# 注意: この例では最初 100M の ibdata1 が、
# 888M (1024 * 1024の倍数) だけ拡張されていたので、988M の指定になっている。
```

ファイルシステムの最大ファイルサイズが 2 GB である場合は注意が必要です。InnoDB は、OS の最大ファイルサイズを考慮しません。このようなファイルシステムでは、データファイルの最大サイズを指定する必要があります。

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:100M:autoextend:max:2000M
```

単純な my.cnf の例。128 MB の RAM と 1 つのハードディスクを搭載したコンピュータを使用しているとします。次に示すのは、InnoDB の my.cnf または my.ini における設定パラメータの例です。この例では、MySQL-Max-3.23.50 以降、または MySQL-4.0.2 以降が稼働していることを想定しています。この例は、Unix と Windows の両方で、ユーザが InnoDB データファイルとログファイルを複数のディスクに分散させない場合に適しています。この設定パラメータによって、自

動拡張するデータファイル ibdata1 および 2 つの InnoDB ログファイル ib_logfile0 と ib_logfile1 が、MySQL の datadir (一般には /mysql/data) に作成されます。 また、アーカイブされた小さな InnoDB ログファイル ib_arch_log_0000000000 も datadir に作成されます。

```
[mysqld]
# You can write your other MySQL server options here
# ...
#                       Datafile(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
innodb_buffer_pool_size=70M
innodb_additional_mem_pool_size=10M
#                       Set the log file size to about
#                       25 % of the buffer pool size
innodb_log_file_size=20M
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
```

datadir でファイルを作成する権限が MySQL サーバにあるかどうかを確認してください。

一部のファイルシステムでは、データファイルを 2 GB 未満にする必要があることに注意してください。 ログファイルをすべて合わせたサイズは、4 GB 未満でなければなりません。また、データファイルをすべて合わせたサイズは、10 MB 以上でなければなりません。

InnoDB データベースを初めて作成するときには、コマンドプロンプトから MySQL サーバを起動するのが最善です。 InnoDB によってデータベースの作成に関する情報が画面に出力されるので、処理の経過を確認できます。 画面出力の例については、次のセクションを参照してください。 たとえば、Windows では次のようにして mysqld-max.exe を起動することにより、MySQL サーバはコンソールを閉じなくなります。

```
your-path-to-mysqld\mysqld-max --console
```

Windows では、my.cnf または my.ini をどこに配置すればいいでしょうか。 Windows での規則は次のとおりです。

- my.cnf または my.ini の一方のみを作成する必要がある。
- my.cnf ファイルは、C:\my.cnf にする必要がある。
- my.ini ファイルは、C:\WINDOWS や C:\WINNT などの %WINDIR% ディレクトリなどに配置する必要がある。MS-DOS のSET コマンドを使用すれば、%WINDIR% の値を出力できる。
- PC で使用するブートローダのブートドライブが C: ドライブではない場合は、my.ini ファイルしか使用できない。

Unix では、どこでオプションを指定すればいいでしょうか。 Unix では、mysqld が次のファイル (存在する場合) から次の順序でオプションを読み取ります。

- /etc/my.cnf グローバルオプション
- COMPILATION_DATADIR/my.cnf サーバ固有のオプション
- defaults-extra-file --defaults-extra-file=... で指定されたファイル
- ~/.my.cnf ユーザ固有のオプション

COMPILATION_DATADIR は MySQL データディレクトリで、mysqld がコンパイルされたときに ./configure オプションとして指定されたものです (一般にはバイナリインストール用の /usr/local/mysql/data、またはソースインストール用の/usr/local/var)。

mysqld がその my.cnf または my.ini をどこから読み取るかがわからない場合は、サーバへの最初のコマンドラインオプションとしてパスを指定できます (mysqld --defaults-file=your_path_to_my_cnf)。

InnoDB は、データファイルへのディレクトリパスを決定する場合に、innodb_data_home_dir に定義されたパスのテキストを innodb_data_file_path 内のデータファイル名またはパスのテキストと結合し、必要に応じて間にスラッシュまたはバックスラッシュを挿入します。my.cnf でキーワード innodb_data_home_dir がまったく指定されていない場合は、MySQL の datadir を意味する 'ドット' ディレクトリ ./ がデフォルトで使用されます。

高度な my.cnf の例。2 GB の RAM と 3 つの 60 GB ハードディスクを搭載した Linux コンピュータを使用しているとします。ハードディスクのディレクトリパスは、それぞれ /、/dr2、/dr3 です。次に示すのは、InnoDB の my.cnf における設定パラメータの例です。

注意: ディレクトリは InnoDB によって作成されないので、各自で作成する必要があります。データおよびロググループのホームディレクトリを作成するには、Unix または MS-DOS の mkdir コマンドを使用します。

```
[mysqld]
# You can write your other MySQL server options here
# ...
innodb_data_home_dir =
#                       Datafiles 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
innodb_buffer_pool_size=1G
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
innodb_log_files_in_group=3
#                       Set the log file size to about
#                       15 % of the buffer pool size
innodb_log_file_size=150M
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
```

```
innodb_lock_wait_timeout=50
#innodb_flush_method=fdatasync
#innodb_thread_concurrency=5
```

2 つのデータファイルを異なるディスクに配置したことに注意してください。 InnoDB は、データファイルによって形成されるテーブルスペースをボトムアップ式に埋めていきます。場合によっては、すべてのデータを同じ物理ディスクに配置しない方がパフォーマンスが良くなります。ログファイルをデータとは別のディスクに配置すると、ほとんどの場合パフォーマンスは良くなります。 ローデバイス をデータファイルとして使用することもできます。一部の Unix では、この方法で I/O の処理速度が向上します。my.cnf でこのようなデータファイルを指定する方法については、マニュアルで 項 7.5.13.1.「ディスク I/O」 に関するセクションを参照してください。

警告: Linux x86 では、メモリ使用率の設定を高くし過ぎないように注意してください。glibc はプロセスヒープがスレッドスタックよりも大きくなることを許可しており、その場合にサーバがクラッシュします。次の計算式を見てください。

```
innodb_buffer_pool_size + key_buffer_size +
max_connections * (sort_buffer_size + read_buffer_size) + max_connections * 2 MB
```

この値が、2 GB に近いか、2 GB を超えていると危険です。各スレッドはスタックを使用し (通常は 2 MB。ただし MySQL AB バイナリでは 256 KB のみ) 、最悪の場合、sort_buffer_size + read_buffer_size の大きさの追加メモリも使用します。

他の mysqld サーバパラメータはどのように調整すればいいでしょうか。: ほとんどのユーザに適した一般的な値は次のとおりです。

```
skip-locking
max_connections=200
read_buffer_size=1M
sort_buffer_size=1M
#                       Set key_buffer_size to 5 - 50%
#                       of your RAM depending on how
#                       much you use MyISAM tables, but
#                       keep key_buffer_size + InnoDB
#                       buffer pool size < 80% of
#                       your RAM
key_buffer_size=...
```

注意: 4.0 より前のバージョンでは、一部のパラメータを set-variable = innodb... = 123 のように指定する必要があります。

各設定パラメータの意味は次のとおりです。

オプション	説明
innodb_file_per_table	4.1.1 より利用可能。このオプションによって、InnoDB は作成された各テーブルを独自の .ibd ファイルに格納するようになる。複数のテーブルスペースに関するセクションを参照。
innodb_data_home_dir	ディレクトリパスの中の、すべての InnoDB データファイルに共通な部分。my.cnf でこのオプションを指定しなかった場合のデフォルトは、MySQL の datadir。このオプションには空の文字列も指定できる。その場合、innodb_data_file_path に絶対ファイルパスを指定できる。
innodb_data_file_path	個々のデータファイルへのパスとそのサイズ。各データファイルへのフルディ

	レクトリパスは、ここで指定したパスに innodb_data_home_dir を連結することで取得できる。ファイルサイズはメガバイトで指定されるため、上記のようにサイズの後に 'M' が付加されている。InnoDB は 'G' という略称も認識する。1 G は 1,024 MB を意味する。3.23.44 より、大きなファイルをサポートするオペレーティングシステムで 4 GB を超えるファイルサイズも設定できるようになった。一部のオペレーティング システムでは、ファイルを 2 GB 未満にする必要がある。4.0 より、innodb_data_file_path を指定しない場合のデフォルトの動作として、10 MB の自動拡張データファイル ibdata1 が作成される。ファイルの合計サイズは、10 MB 以上でなければならない。
innodb_mirrored_log_groups	データベースのために保持しておくロググループのコピーの数。現時点では 1 に設定する必要がある。
innodb_log_group_home_dir	InnoDB ログファイルへのディレクトリパス。my.cnf でこのオプションを指定しなかった場合は、デフォルトで MySQL の datadir が設定される。
innodb_log_files_in_group	ロググループ内のログファイルの数。InnoDB はこれらのログファイルに循環的に書き込みを行う。このパラメータの推奨値とデフォルト値は共に 2。
innodb_log_file_size	ロググループ内の各ログファイルのサイズ (メガバイト) 。実際的な値の範囲は、1M から、下で指定するバッファプールのサイズの 1/n まで (n はグループ内のログファイルの数) 。この値が大きいほど、バッファプールで必要となるチェックポイントフラッシュの回数が減るため、ディスク I/O が削減される。ただし、ログファイルが大きいと、クラッシュした場合のリカバリに時間がかかる。32 ビットコンピュータでは、ログファイルの合計サイズを 4 GB 未満にする必要がある。デフォルトは 5M。
innodb_log_buffer_size	InnoDB がディスク上のログファイルにログを書き出すために使用するバッファのサイズ。実際的な値は 1M 〜 8M。ログバッファを大きくすると、トランザクションコミットまでディスクにログを書き出すことなく大きなトランザクションを実行できる。大きなトランザクションがある場合は、このようにログバッファを大きくすることでディスク I/O を削減できる。
innodb_flush_log_at_trx_commit	通常、このパラメータは 1 に設定する。これによって、トランザクションコミット時にログがディスクにフラッシュされ、トランザクションによる変更が確定されてデータベースクラッシュを免れる。このような安全性を必要とせず、かつ小さなトランザクションを実行している場合は、このオプションを 0 または 2 に設定してログへのディスク I/O を削減できる。値 0 を指定すると、ログファイルへのログの書き込み、およびディスクへのログファイルのフラッシュが1 秒に約1 回しか行われなくなる。値 2 を指定すると、ログファイルへのログの書き込みはコミットのたびに行われるが、ディスクへのログファイルのフラッシュは 1 秒に約 1 回しか行われなくなる。MySQL-4.0.13 よりデフォルト値が 0 から 1 に変更された。
innodb_log_arch_dir	ログのアーカイブを使用する場合に、いっぱいになったログファイルがアーカイブされるディレクトリ。現時点では、このパラメータを innodb_log_group_home_dir と同じ値に設定する必要がある。
innodb_log_archive	現時点では、このパラメータを 0 に設定する必要がある。バックアップからのリカバリは MySQL が独自のログファイルを使って行うため、現時点では InnoDB のログファイルをアーカイブする必要はない。

innodb_buffer_pool_size	InnoDB が、そのテーブルのデータやインデックスをキャッシュするために使用するメモリバッファのサイズ。この値が大きいほど、テーブル内のデータへのアクセスに必要なディスク I/O が少なくなる。データベース専用サーバでは、このパラメータをマシンの物理メモリの 80% にまで設定できる。ただし、物理メモリの競合によってオペレーティングシステムでページングが発生する可能性があるため、あまり大きな値は設定しないようにする。
innodb_buffer_pool_awe_mem_mb	32 ビット版 Windows の AWE メモリに配置されるバッファプールのサイズ (MB)。4.1.0 から利用可能で、32 ビット版 Windows にのみ関係する。使用する 32 ビット版 Windows オペレーティングシステムが、Address Windowing Extensions (AWE) と呼ばれる 4 GB を超えるメモリをサポートしている場合は、このパラメータを使用して InnoDB バッファプールを AWE 物理メモリに割り当てることができる。指定可能な最大値は、64000。このパラメータを指定した場合、innodb_buffer_pool_size は mysqld の 32 ビットアドレス空間におけるウィンドウとなる (このアドレス空間で InnoDB が AWE メモリをマップする)。その場合の innodb_buffer_pool_size に適したサイズは 500M。
innodb_additional_mem_pool_size	InnoDB がデータディクショナリの情報とその他の内部データ構造を格納するために使用するメモリプールのサイズ。このパラメータの実際的な値は 2M であるが、アプリケーションで使用するテーブルの数が多いほど、この値を大きくする必要がある。InnoDB は、このプールのメモリを使い果たすと、オペレーティングシステムからメモリを割り当てるようになり、MySQL エラーログに警告メッセージを書き込む。
innodb_file_io_threads	InnoDB におけるファイル I/O スレッドの数。通常、この値は 4 にする必要があるが、Windows ではこれより大きな数を指定するとディスク I/O の面で有利になる場合がある。
innodb_lock_wait_timeout	ロック待機の状態になった InnoDB トランザクションがロールバックされるまでのタイムアウト時間 (秒)。InnoDB は、ロックされる対象のテーブルにおいてトランザクションのデッドロックを自動的に検出し、そのトランザクションをロールバックすることができる。 しかしながら、LOCK TABLES コマンドを使用したり、または同じトランザクションで InnoDB 以外のトランザクションセーフのストレージエンジンを使用したりすると、InnoDB が検出できないデッドロックが発生することがある。タイムアウトは、このような状況を解決するのに役立つ。
innodb_flush_method	3.23.40 より使用可能。デフォルト値は fdatasync。これ以外に、O_DSYNC を指定できる。
innodb_force_recovery	警告: このオプションは、破損したデータベースからテーブルをダンプする必要がある緊急事態でのみ定義する必要がある。指定可能な値は 1 〜 6。それぞれの値の意味については、この後の「強制的なリカバリ」を参照。InnoDB では、安全対策として、このオプションが 0 より大きいときはユーザがデータを変更できないようになっている。このオプションは、バージョン 3.23.44 より使用可能。

7.5.4. InnoDB テーブルスペースの作成

MySQL がすでにインストールされ、my.cnf の編集も終わって必要な InnoDB 設定パラメータが設定されているとします。 MySQL を起動する前に、InnoDB データファイルとログファイル用に指定したディレクトリが存在すること、およびそれらのディレクトリへのアクセス権があることを確認する必要があります。InnoDB によって作成されるのはファイルのみで、ディレクトリは作成されません。データファイルとログファイルを保存できるだけの十分なディスク領域があることも確認してください。

MySQL を起動すると、InnoDB はデータファイルとログファイルの作成を開始し、次のようなメッセージを出力します。

```
~/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
```

これで、新しい InnoDB データベースが作成されました。mysql などの通常使用する MySQL クライアントプログラムで MySQL サーバに接続できます。 mysqladmin shutdown で MySQL サーバをシャットダウンすると、InnoDB から次のようなメッセージが出力されます

```
010321 18:33:34  mysqld: Normal shutdown
010321 18:33:34  mysqld: Shutdown Complete
InnoDB: Starting shutdown...
InnoDB: Shutdown completed
```

これで、データファイルとログのディレクトリを参照できるようになり、作成したファイルを確認できます。ログディレクトリには、ib_arch_log_0000000000 という名前の小さなファイルも作成されます。これは、データベース作成時に作成されたファイルで、その後に InnoDB がログのアーカイブをオフに切り替えています。 MySQL を再び起動すると、次のようなメッセージが出力されます。

```
~/mysqlm/sql > mysqld
InnoDB: Started
mysqld: ready for connections
```

7.5.4.1. データベース作成時に問題が生じた場合

InnoDB がファイル操作中にオペレーティングシステムエラーを出力した場合、通常は次のいずれかが原因です。

- InnoDB のデータディレクトリまたはログディレクトリを作成しなかった。
- mysqld に、これらのディレクトリでファイルを作成する権利がない。
- mysqld が正しい my.cnf または my.ini ファイルを読み取らなかったために、ユーザが指定したオプションを認識していない。
- ディスクがいっぱいか、またはディスククォータが超過している。
- 作成したサブディレクトリの名前が、指定したデータファイルと同じになっている。
- innodb_data_home_dir または innodb_data_file_path に構文エラーがある。

InnoDB データベースの作成で問題が生じた場合は、InnoDB が作成したすべてのファイルを削除する必要があります。つまり、MySQL データベースディレクトリから、すべてのデータファイル、すべてのログファイル、アーカイブされた小さなログファイルを削除し、InnoDB テーブルを作成した場合は、これらのテーブルに対応する .frm ファイルも削除します。これで、InnoDB データベースを再び作成することができます。

7.5.5. InnoDB テーブルの作成

mysql test で MySQL クライアントを起動したとします。 InnoDB 形式のテーブルを作成するには、テーブルを作成する SQL コマンドで、TYPE = InnoDB を指定する必要があります。

```
CREATE TABLE CUSTOMER (A INT, B CHAR (20), INDEX (A)) TYPE = InnoDB;
```

この SQL コマンドによって、my.cnf に指定したデータファイルから成る InnoDB テーブルスペースに、テーブルおよびカラム A のインデックスが作成されます。さらに MySQL によって、MySQL データベースディレクトリ test に CUSTOMER.frm ファイルが作成されます。 内部では、InnoDB が独自のデータディクショナリに 'test/CUSTOMER' テーブルのエントリを追加します。したがって、MySQL の別のデータベースに同じ CUSTOMER という名前のテーブルを作成しても、InnoDB 内部でテーブル名が衝突することはありません。

TYPE = InnoDB で作成した任意のテーブルに対して MySQL のテーブルステータスコマンドを発行することで、InnoDB テーブルスペースの空き容量を照会できます。テーブルスペースの空き容量は、SHOW の出力のテーブルコメントセクションに表示されます。次に例を示します。

```
SHOW TABLE STATUS FROM test LIKE 'CUSTOMER';
```

SHOW が InnoDB テーブルについて出力する統計情報は概算であることに注意してください。これらの情報は、SQL 文解析の最適化で使用されます。ただし、テーブルとインデックスに予約されているサイズ (バイト単位) は正確です。

7.5.5.1. MyISAM テーブルから InnoDB への変換

InnoDB には、インデックスを別途作成するための特別な最適化機能がありません。 このため、テーブルをエクスポートしてインポートし、その後にインデックスを作成しなおしても意味がありません。 テーブルを最も速く InnoDB に変換するには、InnoDB テーブルに直接データを挿入します。つまり、ALTER TABLE ... TYPE=INNODB を使用するか、あるいは同じ定義で空の InnoDB テーブルを作成し、INSERT INTO ... SELECT * FROM ... でレコードを挿入します。

挿入処理を管理しやすくするために、大きなテーブルは分割して挿入するとよいでしょう。

```
INSERT INTO newtable SELECT * FROM oldtable
```

```
WHERE yourkey > something AND yourkey <= somethingelse;
```

すべてのデータを挿入した後に、テーブル名を変更することができます。

大きなテーブルを変換する際には、InnoDB のバッファプールサイズを大きくして、ディスク I/O を削減する必要があります。ただし、物理メモリの 80% を超えないようにしてください。 InnoDB のログファイルおよびログバッファも大きなサイズに設定します。

テーブルスペースが不足していないことを確認します。InnoDB テーブルは、MyISAM テーブルよりも多くの領域を使用します。ALTER TABLE で領域が足りなくなると、ロールバックが実行されます。ロールバックがディスクバウンドすると、完了までに数時間を要する可能性があります。 挿入の際は、InnoDB が挿入バッファを使用してセカンダリインデックスレコードをまとめてインデックスにマージします。これによって、ディスク I/O が大幅に削減されます。ロールバックではこのようなメカニズムが使用されないため、挿入の 30 倍の時間がかかる場合があります。

ロールバックが暴走した場合は、データベースに貴重なデータがなければ、膨大なディスク I/O の完了を待つよりも、データベースプロセスを強制終了し、すべての InnoDB データファイルとログファイル、および InnoDB テーブルの .frm ファイルを削除したうえで、再度ジョブを実行した方が得策です。

7.5.5.2. FOREIGN KEY 制約

バージョン 3.23.43b より、InnoDB に外部キー制約が装備されるようになりました。 InnoDB は、データの完全性を守るためにユーザが外部キー制約を定義できるようにした最初の MySQL テーブル型です。

InnoDB における外部キー制約定義の構文は次のとおりです。

```
[CONSTRAINT [symbol]] FOREIGN KEY (index_col_name, ...)
            REFERENCES table_name (index_col_name, ...)
            [ON DELETE {CASCADE | SET NULL | NO ACTION
                    | RESTRICT}]
            [ON UPDATE {CASCADE | SET NULL | NO ACTION
                    | RESTRICT}]
```

どちらのテーブルも InnoDB 型でなければなりません。テーブル内には、外部キーカラムが最初のカラムとして同じ順序で列挙されているインデックスが必要です。また、参照テーブル内には、参照カラムが最初のカラムとして同じ順序で列挙されているインデックスが必要です。InnoDB は、外部キーまたは参照キーに対して自動的にインデックスを作成しません。したがって、ユーザが明示的にインデックスを作成する必要があります。外部キーのチェックを高速化し、テーブルスキャンを不要にするには、インデックスが必要です。

外部キーと参照キーの対応するカラムは、型を変換しなくても比較できるように、InnoDB 内部で同じデータ型にする必要があります。 整数型については、サイズと符号の有無が同じでなければなりません。 文字列型の長さは同じでなくてもかまいません。 SET NULL アクションを指定する場合は、子テーブル内のカラムを NOT NULL と宣言していないことを確認してください。

MySQL が CREATE TABLE ステートメントでエラー番号 1005 を返し、エラーメッセージ文字列に errno 150 が示されている場合は、外部キー制約が正しく作成されなかったためにテーブルの作成が失敗しています。 同様に、ALTER TABLE が失敗して errno 150 が示された場合は、変更されたテーブルに対して外部キー定義が誤って作成されています。バージョン 4.0.13 より、SHOW INNODB STATUS を使用して、サーバで最後に発生した InnoDB 外部キーエラーの詳細な説明を参照できるようになりました。

バージョン 3.23.50 より、InnoDB は NULL カラムを含んでいる外部キーまたは参照キー値で外部キー制約をチェックしな

くなりました。

標準 SQL からの逸脱: 親テーブルに同じ参照キー値を持つ複数のレコードがある場合、InnoDB の外部キーチェックでは、同じキー値を持つ親レコードが他に存在しないものとして処理が行われます。たとえば、RESTRICT 型制約を定義し、かつ複数の親レコードを持つ子レコードが存在する場合、InnoDB はこれらの親レコードの削除を禁止します。

バージョン 3.23.50 より、ON DELETE CASCADE 節または ON DELETE SET NULL 節を外部キー制約に付けることもできるようになりました。対応する ON UPDATE オプションは、4.0.8 より利用可能です。ON DELETE CASCADE が指定されている場合に親テーブル内のレコードが削除されると、InnoDB は子テーブル内で親レコード内の参照キー値と等しい外部キー値を持つすべてのレコードを自動的に削除します。ON DELETE SET NULL が指定されている場合は、子レコードが自動的に更新されて、外部キー内のカラムが SQL の NULL 値に設定されます。

標準 SQL からの逸脱: ON UPDATE CASCADE または ON UPDATE SET NULL は、カスケード中にすでに更新したテーブルを繰り返して更新する場合に、RESTRICT のように動作します。これは、カスケードされた更新から生じる無限ループを防ぐためです。一方、自己参照型の ON DELETE SET NULL が 4.0.13 から動作するようになりました。 自己参照型の ON DELETE CASCADE は、以前から動作していました。

次に例を示します。

```
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;
```

次に示すのは複雑な例です。

```
CREATE TABLE product (category INT NOT NULL, id INT NOT NULL,
                      price DECIMAL,
                      PRIMARY KEY(category, id)) TYPE=INNODB;
CREATE TABLE customer (id INT NOT NULL,
                      PRIMARY KEY (id)) TYPE=INNODB;
CREATE TABLE product_order (no INT NOT NULL AUTO_INCREMENT,
                      product_category INT NOT NULL,
                      product_id INT NOT NULL,
                      customer_id INT NOT NULL,
                      PRIMARY KEY(no),
                      INDEX (product_category, product_id),
                      FOREIGN KEY (product_category, product_id)
                        REFERENCES product(category, id)
                        ON UPDATE CASCADE ON DELETE RESTRICT,
                      INDEX (customer_id),
                      FOREIGN KEY (customer_id)
                        REFERENCES customer(id)) TYPE=INNODB;
```

バージョン 3.23.50 より、InnoDB では次のステートメントによって新しい外部キー制約をテーブルに追加できるようになりました。

```
ALTER TABLE yourtablename
ADD [CONSTRAINT [symbol]] FOREIGN KEY (...) REFERENCES anothertablename(...)
[on_delete_and_on_update_actions]
```

ただし、必要なインデックスを先に作成することを忘れないでください。

バージョン 4.0.13 より、InnoDB が次のステートメントをサポートするようになりました。

```
ALTER TABLE yourtablename DROP FOREIGN KEY internally_generated_foreign_key_id
```

外部キーを破棄する場合は、SHOW CREATE TABLE を使って、内部で生成された外部キー ID を確認する必要があります。

InnoDB が 3.23.50 より前のバージョンである場合は、外部キー制約を持つテーブルまたは外部キー制約で参照されるテーブルに関連して ALTER TABLE または CREATE INDEX を使用しないでください。ALTER TABLE を実行すると、テーブルに定義されているすべての外部キー制約が削除されます。ALTER TABLE は、参照テーブルにも使用しないでください。ただし、スキーマを変更する場合は DROP TABLE および CREATE TABLE を使用します。MySQL は、ALTER TABLE を実行するときに内部的に RENAME TABLE を使用する場合があります。この場合、テーブルを参照する外部キー制約で混乱が生じます。 CREATE INDEX ステートメントは、MySQL では ALTER TABLE として処理されるため、このステートメントにもこれらの制約が適用されます。

InnoDB は、外部キーチェックを実行する際に、参照する子レコードまたは親レコードに対して共有行レベルロックを設定します。 InnoDB は、外部キー制約を即座にチェックします。チェックがトランザクションコミットまで延期されることはありません。

外部キー制約を、たとえば LOAD DATA 処理の間だけ無視する場合は、SET FOREIGN_KEY_CHECKS=0 を実行します。

InnoDB では、外部キー制約によって参照されているテーブルでも破棄できます。この場合、その制約が壊れることになります。テーブルを破棄すると、その作成ステートメントで定義された制約も破棄されます。

破棄されたテーブルを再作成する場合は、そのテーブルを参照する外部キー制約に沿った定義をテーブル内に設定する必要があります。すでに説明したように、このテーブルには、正しい名前と型を持つカラム、および参照キー上のインデックスが必要です。 これらの条件が満たされていないと、MySQL から エラー番号 1005 が返され、エラーメッセージ文字列に errno 150 が示されます。

バージョン 3.23.50 より、次のステートメントを呼び出すと、InnoDB からテーブルの外部キー定義が返されるようになりました。

```
SHOW CREATE TABLE yourtablename
```

また、mysqldump によってテーブルの正しい定義と共に外部キーがダンプファイルに出力されます。

また、次のステートメントでテーブル T の外部キー制約も列挙できます。

```
SHOW TABLE STATUS FROM yourdatabasename LIKE 'T'
```

外部キー制約は、出力のテーブルコメントに列挙されます。

7.5.5.3. 複数テーブルスペース - 各テーブルを独自の .ibd ファイルに入れる

重要: InnoDB-4.1.1 にアップグレードした後に、ダウングレードすることはできません。それは、InnoDB の以前のバージョンが複数のテーブルスペースを認識しないためです。

MySQL-4.1.1 より、各 InnoDB テーブルとそのインデックスを、そのテーブル独自のファイルに格納できるようになりました。各テーブルが独自のテーブルスペースに格納されることから、この機能は複数テーブルスペースと呼ばれます。

この機能を有効にするには、my.cnf の [mysqld] セクションに次の行を追加します。

```
innodb_file_per_table
```

これによって InnoDB は、各テーブルが属しているデータベースディレクトリ内にあるテーブル固有の tablename.ibd ファイルに、それぞれのテーブルを格納します。これは MyISAM の動作と似ていますが、MyISAM ではテーブルがデータファイル tablename.MYD とインデックスファイル tablename.MYI に分けられます。InnoDB の場合は、データとインデックスの両方が .ibd ファイルに格納されます。

my.cnf から行 innodb_file_per_table を削除すると、InnoDB によって再び ibdata ファイル内にテーブルが作成されます。4.1.1 以降のバージョンへアップグレードする前に ibdata ファイルに格納されていた古いテーブルはそのまま残され、.ibd ファイルに変換されることはありません。

InnoDB にはシステムテーブルスペースが必要で、.ibd ファイルだけでは不十分です。 システムテーブルスペースは、これまで使われていた ibdata ファイルで構成されます。InnoDB は、内部データディクショナリと UNDO ログをシステムテーブルスペースに配置します。

MyISAM テーブルとは異なり、.ibd ファイルは自由に移動できません。これは、テーブル定義が InnoDB システムテーブルスペースに格納されているためと、InnoDB でトランザクション ID とログシーケンス番号の整合性を維持する必要があるためです。

従来の RENAME コマンドを使って、.ibd ファイルと関連テーブルを、あるデータベースから同じ MySQL/InnoDB インストール内にある別のデータベースに移すことができます。

```
RENAME TABLE olddatabasename.tablename TO newdatabasename.tablename;
```

同じ MySQL/InnoDB インストールから取得した .ibd ファイルのクリーンバックアップがある場合は、次のコマンドでそのバックアップを InnoDB データベースにリストアできます。

```
ALTER TABLE tablename DISCARD TABLESPACE; /* CAUTION: 現在の .ibd ファイルを削除します! */
<バックアップした .ibd ファイルを適切な場所に置きます>
ALTER TABLE tablename IMPORT TABLESPACE;
```

ここでのクリーンとは、次のことを意味しています。

- .ibd ファイル内に、トランザクションが実行してコミットされていない変更がない。
- .ibd ファイルにマージされていない挿入バッファエントリがない。
- 削除マークされたすべてのインデックスレコードが .ibd ファイルから削除されている。
- mysqld によって、.ibd ファイルの変更されたすべてのページがバッファプールからファイルにフラッシュされている。

このような .ibd ファイルのクリーンバックアップは、次の方法で作成できます。

- mysqld サーバのすべての活動を停止し、すべてのトランザクションをコミットする。
- SHOW INNODB STATUS 文でデータベースにアクティブなトランザクションがないことが示され、InnoDB のメインスレッドが Waiting for server activity になるのを待つ。これで、.ibd ファイルのコピーを作成できる。

クリーンな .ibd ファイルを作成するもう 1 つの方法 (有償) があります。

- InnoDB ホットバックアップを使用して InnoDB のデータをバックアップする。
- 新たな mysqld サーバをバックアップしたファイルを使用するようにして起動し、バックアップ内の .ibd ファイルをクリーンアップする。

TODO には、クリーンな .ibd ファイルも別の MySQL/InnoDB に移動できるようにすることが挙げられています。そのためには、.ibd ファイル内でトランザクション ID と ログシーケンス番号をリセットすることが必要になります。

7.5.6. InnoDB データファイルとログファイルの追加と削除

バージョン 3.23.50 および 4.0.2 より、InnoDB の最後のデータファイルに autoextend を指定できるようになりました。あるいは、追加のデータファイルを指定してテーブルスペースを拡大することができます。そのためには、MySQL サーバをシャットダウンし、my.cnf ファイルで innodb_data_file_path の末尾に新しいデータファイルを追加し、MySQL サーバを再起動します。

現時点では、InnoDB からデータファイルを削除することはできません。データベースのサイズを小さくするには、mysqldump ですべてのテーブルをダンプし、新しいデータベースを作成し、そのデータベースにテーブルをインポートする必要があります。

InnoDB ログファイルの数またはサイズを変更する場合は、MySQL をシャットダウンし、エラーなくシャットダウンすることを確認する必要があります。 その後に、シャットダウンで問題が発生した場合に備えて、古いログファイルを安全な場所にコピーします。これらはデータベースをリカバリする際に必要となります。古いログファイルをログファイルディレクトリから削除し、my.cnf を編集してから MySQL を再び起動します。起動時に、InnoDB から新しいログファイルを作成していることが通知されます。

7.5.7. InnoDB データベースのバックアップとリカバリ

安全なデータベース管理の秘訣は、定期的にバックアップを取ることです。

InnoDB ホットバックアップは、稼働中の InnoDB データベースをバックアップするためのオンラインバックアップツールです。InnoDB ホットバックアップでは、データベースをシャットダウンする必要がなく、ロックが設定されたり、通常のデータベース処理が妨害されたりすることもありません。InnoDB ホットバックアップは、追加で導入する有償のツールで、標準の MySQL ディストリビューションには含まれていません。詳細情報とスクリーンショットについては、InnoDB ホットバックアップのホームページ (http://www.innodb.com/manual.php) を参照してください。

MySQL サーバをシャットダウンできる場合は、データベースの 'バイナリ' バックアップを取るために次の作業を行う必要があります。

- MySQL データベースをシャットダウンし、それがエラーなくシャットダウンすることを確認する。
- すべてのデータファイルを安全な場所にコピーする。
- すべての InnoDB ログファイルを安全な場所にコピーする。
- my.cnf オプション設定ファイルを安全な場所にコピーする。
- InnoDB テーブルのすべての .frm ファイルを安全な場所にコピーする。

上記のバイナリバックアップに加えて、mysqldump を使って定期的にテーブルのダンプを取るようにしてください。その理由は、バイナリファイルが気付かないうちに壊れる可能性があるためです。ダンプされたテーブルはテキストファイルに格納されます。これらは人間による解読が可能で、データベースバイナリファイルよりもはるかに単純です。ダンプされたファイルではテーブルの破損を容易に確認できます。また、このファイルの形式は単純であるため、重大なテーブルの破損が起こる確率も低くなります。

データベースのバイナリバックアップと同時にダンプを取ることをお勧めします。すべてのテーブルのスナップショットを矛盾のない状態でダンプするには、すべてのクライアントをデータベースからシャットアウトする必要があります。その後にバイナリバックアップを取れば、データベースの矛盾のないスナップショットを 2 つの形式で保持することになります。

上記のバイナリバックアップから InnoDB データベースをリカバリできるようにするには、MySQL の一般的なログをオンにして MySQL データベースを実行する必要があります。ここでの一般的なログとは、InnoDB ログとは関係のない MySQL サーバのログメカニズムを意味します。

MySQL サーバプロセスのクラッシュからリカバリする場合に必要な作業は、そのプロセスの再起動だけです。InnoDB は自動的にログをチェックし、現時点までのデータベースのロールフォワードを実行します。InnoDB は、クラッシュ時にコミットされていなかったトランザクションを自動的にロールバックします。リカバリの間、InnoDB は次のようなメッセージを出力します。

```
~/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
```

データベースの破損またはディスク障害が発生した場合は、バックアップからリカバリを実行する必要があります。データベースが壊れた場合は、まず壊れていないバックアップを探します。MySQL のマニュアルに従って、MySQL の一般ログファイルからのリカバリを実行します。

7.5.7.1. 強制的なリカバリ

データベースページが壊れた場合は、SELECT INTO OUTFILE を使ってデータベースからテーブルをダンプできます。通常は、ほとんどのデータが損傷を受けず正常な状態です。ところが、破損が原因で SELECT * FROM table や InnoDB のバックグラウンド処理がクラッシュまたはアサートしたり、InnoDB のロールフォワードリカバリさえもクラッシュすること

があります。InnoDB のバージョン 3.23.44 より、InnoDB を強制的に起動することのできるオプションが my.cnf に追加されました。また、テーブルをダンプできるように、バックグラウンド処理が実行されないようにすることも可能となりました。たとえば、my.cnf で

```
innodb_force_recovery = 4
```

を設定することができます。

innodb_force_recovery の選択肢は下に挙げています。 データベースには、これらのオプションを別の用途で使用しないでください。 InnoDB では、安全対策として、このオプションが 0 より大きいときはユーザが INSERT、UPDATE、または DELETE を実行できないようになっています。

バージョン 3.23.53 および 4.0.4 より、強制リカバリが使用される場合でも、テーブルの DROP または CREATE を実行することができます。ロールバックでクラッシュを引き起こしているテーブルが特定できれば、そのテーブルを破棄できます。 また、問題のある大量インポートまたは ALTER TABLE が原因で暴走したロールバックも、この方法で停止できます。 mysqld プロセスを強制終了し、my.cnf のオプション innodb_force_recovery=3 を使用することで、ロールバックせずにデータベースを正常な状態に戻すことができます。 その後に、ロールバックが暴走する原因となったテーブルを DROP で破棄します。

下に列挙されている各オプションでは、その番号が大きいものがより小さい番号の対策をすべて盛り込んでいます。悪くてもオプション 4 でテーブルをダンプできれば、個々のページが破損しても一部のデータが失われるだけなので比較的安全です。オプション 6 にはより劇的な影響力があります。データベースページが古い状態のまま残るため、B ツリーやその他のデータベース構造へさらに破損が及ぶ可能性があるからです。

- 1 (SRV_FORCE_IGNORE_CORRUPT) では、壊れたページが検出されてもサーバが実行される。SELECT * FROM table で壊れたインデックスレコードやページを飛び越すようにすれば、テーブルをダンプする際に役立つ。
- 2 (SRV_FORCE_NO_BACKGROUND) では、メインスレッドが実行されなくなる。パージの際にクラッシュが発生する場合は、このオプションによってそれが回避される。
- 3 (SRV_FORCE_NO_TRX_UNDO) では、リカバリ後にトランザクションロールバックが実行されない。
- 4 (SRV_FORCE_NO_IBUF_MERGE) では、挿入バッファのマージ操作も実行されなくなる。この操作がクラッシュを引き起こす場合は、実行しない方がよい。また、テーブル統計情報も計算しないようにする。
- 5 (SRV_FORCE_NO_UNDO_LOG_SCAN) では、データベース起動時に UNDO ログが参照されない。InnoDB は、未完了のトランザクションもコミット済みとして扱う。
- 6 (SRV_FORCE_NO_LOG_REDO) では、リカバリに関連してログのロールフォワードが実行されない。

7.5.7.2. チェックポイント

InnoDB には、ファジーチェックポイントと呼ばれるチェックポイントメカニズムが実装されています。InnoDB は、変更されたデータベースページを小規模なバッチ単位でバッファプールからフラッシュします。バッファプールを 1 回のバッチでフラッシュする必要はありません。実際にこれを行うと、ユーザの SQL ステートメントの処理が一時的に停止します。

クラッシュリカバリの際に、InnoDB はログファイルに書き込まれたチェックポイントラベルを検索します。InnoDB は、このラベルより前に実行されたデータベースへの変更が、データベースのディスクイメージにすでに反映されていることを認識しています。 次に InnoDB は、ログファイルでこのチェックポイント以降をスキャンし、ログに記録された変更を

データベースに適用します。

InnoDB はログファイルへの書き込みを循環的に行います。 InnoDB によるリカバリが必要となった場合は、バッファプール内のデータベースページとディスク上のイメージの不一致を引き起こしている全てのコミット済みの変更を、ログファイルから取得できなければなりません。つまり、InnoDB は、ログファイルを循環的に再使用する際に、再使用しようとするログファイルに記録された変更が、ディスク上のデータベースページのイメージにすでに反映されていることを確認する必要があります。そのために InnoDB はチェックポイントを作成する必要があり、それには変更されたデータベースページをディスクにフラッシュする処理が伴います。

これらのことから、ログファイルを大きくしておけば、チェックポイントを実行する際のディスク I/O を削減できると言えます。ログファイルの合計サイズを、バッファプールのサイズ以上に設定することは理に適っています。 ログファイルを大きくした場合の難点として、データベースに適用するログの量が増えるために、クラッシュリカバリに時間がかかるおそれがあります。

7.5.8. InnoDB データベースを別のマシンに移動する

Windows では、InnoDB がデータベース名とテーブル名を内部的に小文字で格納します。バイナリ形式のデータベースを Unix から Windows に、またはその逆に移動するには、すべてのテーブル名とデータベース名を小文字にする必要があります。Unix でこれを簡単に行うには、my.cnf の [mysqld] セクションに次の行を追加します。

```
lower_case_table_names=1
```

これは、テーブルの作成を開始する前に行います。Windows では、デフォルトで 1 に設定されます。

InnoDB のデータファイルとログファイルは、すべてのプラットフォームでバイナリ互換です。ただし、それらのマシンで浮動小数点数の形式が同じであることが必要です。 InnoDB データベースは、すべての関連ファイルをコピーするだけで移動できます。関連ファイルについては、データベースのバックアップに関する前のセクションを参照してください。マシン間で浮動小数点数の形式が異なっていても、テーブル内で FLOAT または DOUBLE データ型を使用していなければ手順は同じです。つまり、関連ファイルをコピーするだけで済みます。浮動小数点数の形式が異なっている場合に、テーブルで浮動小数点データが使用されているときは、mysqldump および mysqlimport を使用してそれらのテーブルを移動する必要があります。

パフォーマンス上のヒントとして、データをデータベースにインポートするときは、大量のインポートトランザクションによって生成される大きなロールバックセグメントに対応できるだけの十分なテーブルスペースがあると想定して、オートコミットモードをオフにします。 コミットは、テーブル全体またはテーブルのセグメントをインポートした後に実行します。

7.5.9. InnoDB トランザクションモデルとロック

InnoDB トランザクションモデルの目標は、マルチバージョニングのデータベースの優れた特性を、従来の 2 相ロックと組み合わせることでした。 InnoDB は、行レベルでロックを行い、デフォルトではクエリを Oracle 式の非ロックの一貫性読み取りとして実行します。 InnoDB のロックテーブルはスペース効率の高い方法で格納されるため、ロックエスカレーションは不要です。一般には、複数のユーザがデータベースのあらゆるレコードまたはレコードのランダムなサブセットをロックすることができ、InnoDB でメモリ不足が発生することもありません。

InnoDB では、すべてのユーザ活動がトランザクションの内部で発生します。MySQL でオートコミットモードが使用されている場合は、各 SQL ステートメントが 1 つのトランザクションとなります。 MySQL は常にオートコミットモードをオンにして新たな接続を開始します。

SET AUTOCOMMIT = 0 でオートコミットモードがオフになると、ユーザが常にトランザクションを開いていると見なされます。このユーザが SQL の COMMIT または ROLLBACK ステートメントを実行すると、現在のトランザクションが終了し、新しいトランザクションが開始されます。どちらのステートメントも、現在のトランザクションで設定されたすべての InnoDB ロックを解除します。COMMIT を実行すると、現在のトランザクションで加えられた変更が確定し、他のユーザが認識できる状態になります。一方、ROLLBACK ステートメントを実行すると、現在のトランザクションによって加えられたすべての変更が取り消されます。

接続に AUTOCOMMIT = 1 が設定されている場合でも、ユーザはトランザクションを実行できます。その場合、START TRANSACTION または BEGIN でトランザクションを開始し、COMMIT または ROLLBACK でトランザクションを終了します。

7.5.9.1. InnoDB と SET ... TRANSACTION ISOLATION LEVEL ...

SQL-92 のトランザクション分離レベルに関する InnoDB のデフォルトは REPEATABLE READ です。 バージョン 4.0.5 より、InnoDB は SQL-92 標準で記述されている 4 種類のトランザクション分離レベルをすべて提供するようになりました。 my.cnf の [mysqld] セクションで、すべての接続に対するデフォルトの分離レベルを設定できます。

```
transaction-isolation = {READ-UNCOMMITTED | READ-COMMITTED
                         | REPEATABLE-READ | SERIALIZABLE}
```

ユーザは、単一セッションの分離レベルまたは新たに接続するすべてのコネクションの分離レベルを、SET TRANSACTION ステートメントで変更できます。その構文は次のとおりです。

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL
                       {READ UNCOMMITTED | READ COMMITTED
                        | REPEATABLE READ | SERIALIZABLE}
```

SQL 構文ではレベル名にハイフンを付けないことに注意してください。

デフォルトの動作として、次の (まだ開始されていない) トランザクションの分離レベルが設定されます。このステートメントに GLOBAL キーワードを使用すると、それ以降に作成されるすべての新しい接続 (既存の接続は対象外) に対してグローバルにデフォルトのトランザクションレベルが設定されます。 これを行うには、SUPER 特権が必要です。 SESSION キーワードを使用すると、現在の接続で実行されるすべての新しいトランザクションに対してデフォルトのトランザクションレベルが設定されます。どのクライアントも、自由にセッション分離レベル (トランザクションの途中であっても) または次のトランザクションの分離レベルを変更できます。 3.23.50 より前のバージョンでは、InnoDB テーブルに SET TRANSACTION が作用しませんでした。4.0.5 より前のバージョンでは、REPEATABLE READ と SERIALIZABLE のみが提供されていました。

グローバルおよびセッションのトランザクション分離レベルは、次のステートメントで確認できます。

```
SELECT @@global.tx_isolation;
SELECT @@tx_isolation;
```

行レベルロックでは、InnoDB がネクストキーロックを使用します。 つまり、インデックスレコードに加えてインデックスレコードの前の ``ギャップ'' もロックすることで、他のユーザがそのインデックスレコードの直前に挿入できないようにします。ここでいうネクストキーロックとは、インデックスレコードとその前のギャップをロックするロックのことです。 ギャップロックとは、あるインデックスレコードの前のギャップのみをロックするロックです。

InnoDB における各分離レベルの詳細は次のとおりです。

- READ UNCOMMITTED ``ダーティリード" とも呼ばれる。ロックを取得しない SELECT ステートメントが実行されるため、前のバージョンのレコードが参照されない。 したがって、この分離レベルでは '一貫した' 読み取りにならない。 それ以外の場合、このレベルは READ COMMITTED と同じように機能する。

- READ COMMITTED Oracle にやや近い分離レベル。 全ての SELECT ... FOR UPDATE および SELECT ... LOCK IN SHARE MODE ステートメントは、インデックスレコードのみをロックし、その前のギャップはロックしない。したがって、ロックされたレコードの次に新しいレコードを自由に挿入できる。 一意の検索条件でユニークインデックスを使用する UPDATE および DELETE は、検出したインデックスレコードのみをロックし、その前のギャップはロックしない。 ただし、範囲指定のある UPDATE および DELETE では、InnoDB が引き続きネクストキーロックまたはギャップロックを設定し、その範囲でカバーされるギャップへ他のユーザが挿入できないようにする必要がある。これは、MySQL のレプリケーションおよびリカバリが機能するように、``ファントム (幻像) 行" をブロックしなければならないからである。 一貫性読み取りは、Oracle と同じように動作する。つまり、一貫性読み取りでは、同じトランザクション内であっても、最新のスナップショットが独自に設定され、読み取られる。

- REPEATABLE READ InnoDB のデフォルトの分離レベル。 SELECT ... FOR UPDATE, SELECT ... LOCK IN SHARE MODE, 一意の検索条件でユニークインデックスを使用する UPDATE および DELETE は、検出したインデックスレコードのみをロックし、その前のギャップはロックしない。 それ以外の場合は、これらの操作でネクストキーロック (次のキーでスキャンされるインデックスの範囲をロック) またはギャップロックが使用され、他のユーザによる新たな挿入がブロックされる。 読み取り一貫性には、前の分離レベルとの間に重要な違いがある。このレベルでは、同じトランザクション内のすべての一貫した読み取りが、最初の読み取りで確立されたスナップショットを読み取る。この規則によって、同じトランザクション内で複数の単純な SELECT ステートメントを発行した場合に、それらの SELECT ステートメントも互いに一貫した状態になる。

- SERIALIZABLE 前の分離レベルに似ているが、単純な SELECT ステートメントがすべて暗黙的に SELECT ... LOCK IN SHARE MODE に変換される。

7.5.9.2. ロックを取得しない読み取り一貫性

読み取り一貫性とは、InnoDB がそのマルチバージョニング機能を使用して、ある時点でのデータベースのスナップショットをクエリに提示することを意味します。 クエリには、その時点より前にコミットされたトランザクションによる変更のみが示され、その時点より後のトランザクションまたはコミットされていないトランザクションによる変更は示されません。例外として、クエリを発行したトランザクション自体による変更はクエリに示されます。

デフォルトの REPEATABLE READ 分離レベルで実行している場合は、最初の読み取り内容がスナップショトとして保存され、同じトランザクション内のすべての読み取りで、そのスナップショットから内容が読み取られます。より新しい状態に更新するには、現在のトランザクションをコミットし、その後に新たなクエリを発行します。

読み取り一貫性はデフォルトのモードです。このモードでは、InnoDB が SELECT ステートメントを READ COMMITTED および REPEATABLE READ 分離レベルで処理します。読み取り一貫性では、アクセスするテーブルに一切ロックが設定されません。このため、読み取り一貫性がテーブルで実行されているときにも、他のユーザはこれらのテーブルを自由に変更できます。

7.5.9.3. ロックを取得する読み取り SELECT ... FOR UPDATE および SELECT ... LOCK IN SHARE MODE

読み取り一貫性は、状況によっては不便な場合があります。 テーブル CHILD に新しいレコードを追加するために、テーブル PARENT 内にこの子レコードの親がすでに存在することを確認するとします。

仮に、読み取り一貫性でテーブル PARENT を読み取り、このテーブルで子レコードの親の存在を確認したとします。これでテーブル CHILD に子レコードを確実に追加できるでしょうか。できません。この処理の間に他のユーザがテーブル PARENT から親レコードを削除しても気付かないからです。

これに対処するには、共有ロックモード LOCK IN SHARE MODE で SELECT 文を実行します。

```
SELECT * FROM PARENT WHERE NAME = 'Jones' LOCK IN SHARE MODE;
```

共有モードで読み取りを実行すると、入手可能な最新のデータが読み取られ、読み取ったレコードに共有モードロックが設定されます。 その最新データが、別のユーザのまだコミットされていないトランザクションに属している場合は、そのトランザクションがコミットされるまで待機します。 共有モードロックによって、読み取ったレコードは他のユーザから更新または削除されなくなります。上記のクエリから親である 'Jones' が返されたことを確認したうえで、テーブル CHILD に 'Jones' の子レコードを確実に追加し、トランザクションをコミットできます。 この例は、アプリケーションコードで参照整合性を実装する方法を示しています。

別の例を見てみましょう。テーブル CHILD_CODES に整数のカウンタフィールドがあります。このテーブルは、テーブル CHILD に追加する各子に一意の識別子を割り当てるために使用します。 このカウンタの現在の値を読み取る場合に、読み取り一貫性または共有モード読み取りが適していないのは明らかです。これは、データベースの 2 人のユーザに同じカウンタ値が返されるために、テーブルに同じ識別子を持つ 2 つの子を追加することになり、重複キーエラーが発生するためです。

この場合、カウンタの読み取りとインクリメントを実装する方法が 2 つあります。(1) カウンタを先に 1 増加してから読み取る方法と (2) カウンタを先にロックモード FOR UPDATE で読み取ってからインクリメントする方法です。

```
SELECT COUNTER_FIELD FROM CHILD_CODES FOR UPDATE;
UPDATE CHILD_CODES SET COUNTER_FIELD = COUNTER_FIELD + 1;
```

SELECT ... FOR UPDATE では、入手可能な最新のデータが読み取られ、読み取った各レコードに排他ロックが設定されます。 したがって、検索された SQL UPDATE がレコードに設定するロックと同じロックが設定されます。

7.5.9.4. ネクストキーロック: ファントムの問題の回避

InnoDB は、行レベルのロックでネクストキーロックと呼ばれるアルゴリズムを使用します。 InnoDB が行レベルロックを行うのは、テーブルのインデックスを検索またはスキャンする際に、検出したインデックスレコードに共有ロックまたは排他ロックを設定するためです。このため、行レベルロックは正確にはインデックスレコードロックと呼ばれます。

InnoDB がインデックスレコードに設定するロックは、そのインデックスレコードの前の 'ギャップ' にも影響します。あるユーザがインデックスのレコード R に共有ロックまたは排他ロックを設定すると、他のユーザはインデックス順で R の直前に新しいインデックスレコードを挿入できなくなります。 このようなギャップのロックには、いわゆるファントムの問題を回避する目的があります。たとえば、テーブル CHILD から 100 より大きい ID を持つすべてのレコードを読み取ってロックし、選択したレコードに対してフィールドを更新するとします。

```
SELECT * FROM CHILD WHERE ID > 100 FOR UPDATE;
```

テーブル CHILD のカラム ID にインデックスがあるとします。クエリでは、ID が 100 より大きい最初のレコードから順にそのインデックスがスキャンされます。 ここで、インデックスレコードに設定されたロックがギャップで行われる挿入をロックアウトしないと、テーブルに新しいレコードが挿入される可能性があります。その場合に、トランザクションで次のステートメントをもう一度実行してみます。

```
SELECT * FROM CHILD WHERE ID > 100 FOR UPDATE;
```

すると、クエリから返される結果セットに新しいレコードが含まれることになります。 これは、トランザクションの分離の原則に反します。 つまり、トランザクションの実行中は、読み取ったデータが変化しないことが必要です。一組のレコードを 1 つのデータ項目と見なすと、新たな 'ファントムの' レコードによって、この分離の原則が破られることになります。

InnoDB では、インデックスをスキャンする際に、インデックス内の最後のレコードの後のギャップもロックできます。前の例と同様に、InnoDB が設定したロックによって、テーブル内の ID が 100 より大きい箇所への挿入が防止されます。

ネクストキーロックを使用して、アプリケーションに一意性のチェックを実装できます。共有モードでデータを読み取り、挿入しようとするレコードに重複が見られなければ、レコードを確実に挿入できます。また、読み取り中は対象となるレコードの後続のレコードにネクストキーロックが設定されて、第三者による重複レコードの挿入を防ぎます。このように、ネクストキーロックによって、テーブル内に存在しないものを 'ロック' することができます。

7.5.9.5. InnoDB で各種 SQL ステートメントによって設定されるロック

- SELECT ... FROM ...: 読み取り一貫性。データベースのスナップショットを読み取り、ロックを設定しない。
- SELECT ... FROM ... LOCK IN SHARE MODE: 読み取りで検出されたすべてのインデックスレコードに共有ネクストキーロックを設定する。
- SELECT ... FROM ... FOR UPDATE: 読み取りで検出されたすべてのインデックスレコードに排他ネクストキーロックを設定する。
- INSERT INTO ... VALUES (...): 挿入されたレコードに排他ロックを設定する。このロックはネクストキーロックではないため、挿入されたレコードの前のギャップに他のユーザがレコードを挿入する可能性がある。重複キーエラーが発生した場合は、重複するインデックスレコードに共有ロックを設定する。
- INSERT INTO T SELECT ... FROM S WHERE ...: T に挿入された各レコードに排他 (非ネクストキー) ロックを設定する。読み取り一貫性として S を検索するが、MySQL ログがオンになっている場合は、S に共有ネクストキーロックを設定する。後者の場合に InnoDB がロックを設定する理由は、バックアップからのロールフォワードリカバリで、すべての SQL ステートメントを元の操作とまったく同じように実行する必要があるためである。
- CREATE TABLE ... SELECT ...: 前の項目と同様に、SELECT を一貫性のある読み取りとして、または共有ロックを設定して実行する。
- REPLACE は、ユニークキーで衝突がなければ、挿入と同じように実行される。衝突がある場合は、更新対象のレコードに排他ネクストキーロックが設定される。
- UPDATE ... SET ... WHERE ...: 検索で検出されたすべてのレコードに排他ネクストキーロックを設定する。
- DELETE FROM ... WHERE ...: 検索で検出されたすべてのレコードに排他ネクストキーロックを設定する。
- テーブルに FOREIGN KEY 制約が定義されている場合は、制約条件のチェックを必要とするすべての挿入、更新、削除によって、制約チェックのために参照するレコードに共有行レベルのロックが設定される。また、制約が失敗した場合も、InnoDB によってこれらのロックが設定される。
- LOCK TABLES ... : テーブルロックを設定する。これらのロックは、実装時に MySQL のコードレイヤによって設定される。InnoDB の自動デッドロック検出では、このようなテーブルロックが関係するデッドロックを検出できない。 詳細については次のセクションを参照。 また、MySQL は行レベルのロックを認識しないため、別のユーザが行レベルロ

ックを設定しているテーブル上でテーブルロックを取得することができる。ただし、それによってトランザクションの完全性が損なわれるおそれがある。 See 項7.5.15.「InnoDB テーブルの制限事項」。

7.5.9.6. デッドロックの検出とロールバック

InnoDB は、トランザクションのデッドロックを自動的に検出し、そのトランザクションをロールバックしてデッドロックを回避します。バージョン 4.0.5 より、InnoDB は小さいほうのトランザクションを選択してロールバックするようになります。トランザクションのサイズは、挿入、更新、または削除したレコードの数によって決定されます。 4.0.5 より前のバージョンの InnoDB では、デッドロックを引き起こすロックを要求したトランザクションを、常にロールバックしていました。

InnoDB は、MySQL の LOCK TABLES ステートメントが設定したロックが関係するデッドロック、または InnoDB 以外のストレージエンジンで設定されたロックが関係するデッドロックを検出できません。これらの状況は、my.cnf で設定する innodb_lock_wait_timeout を使って解決する必要があります。

InnoDB がトランザクションの完全なロールバックを実行すると、そのトランザクションのすべてのロックが解除されます。ところが、エラーのために単一の SQL ステートメントのみがロールバックされると、SQL が設定したロックの一部が保持される場合があります。これは、InnoDB が使用する行ロックの格納形式では、ロックを設定した SQL ステートメントを後から特定できないためです。

7.5.9.7. InnoDB での読み取り一貫性の例

デフォルトの REPEATABLE READ 分離レベルで実行しているとします。 一貫した読み取り(このレベルでは、通常の SELECT ステートメントは読み取り一貫性がある) では、クエリがデータベースを参照するときの基準となるタイムポイントを、InnoDB はトランザクションに割り当てます。こうして、タイムポイントが割り当てられた後に(セッションごとのスナップショットを取った後に)、他のトランザクションがレコードを削除してコミットしたとしても、一度読み取った内容は変わりません (トランザクション中の読み取り内容は同じです) 。挿入と更新も同様です。

割り当てられたタイムポイントを先に進めるには、トランザクションをコミットし、新たな SELECT を実行します。

これは、マルチバージョン並行処理制御(multi-versioned concurrency control)と呼ばれます。

```
           ユーザ A            ユーザ B

         SET AUTOCOMMIT=0;     SET AUTOCOMMIT=0;
時間経過
|        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  |
```

```
---------------------
```

このように、ユーザ B が挿入したレコードをユーザ A が参照できるのは、B が挿入をコミットした後で、A がコミットした後になります。また、A が自分のトランザクションをコミットしたことで、時点が B のコミットより先に進みます。

データベースの ``最新の" 状態を参照するには、共有ロックを利用した読み取りを使用する必要があります。

```
SELECT * FROM t LOCK IN SHARE MODE;
```

7.5.9.8. デッドロックの対処法

デッドロックはトランザクションデータベースにおける従来からの問題ですが、一部のトランザクションを実行できなくなるほど頻繁に発生するのでなければ危険ではありません。通常は、アプリケーションを作成する際に、デッドロックのためにロールバックされたトランザクションを再発行できるようにしておく必要があります。

InnoDB は、自動で行レベルロックを使用します。単一のレコードを挿入または削除するだけのトランザクションでもデッドロックが発生します。それは、これらの操作が実際には原子性を持つものではなく、挿入/削除するレコードのインデックスレコード (おそらくは複数) に自動的にロックを設定するためです。

デッドロックに対処し、デッドロックの発生頻度を減らすための方法を次に示します。

- MySQL の 3.23.52 以降または 4.0.3 以降のバージョンでは、SHOW INNODB STATUS を使用して最後に発生したデッドロックの原因を特定する。これは、デッドロックを回避するようにアプリケーションを調整する際に役立つ。
- デッドロックで失敗したトランザクションを再発行できるようにしておく。デッドロックには危険性がないので、単に再試行すればよい。
- トランザクションを頻繁にコミットする。小さなトランザクションは衝突することが少ない。
- ロックを取得する読み取り SELECT ... FOR UPDATE または ... LOCK IN SHARE MODE を使用している場合は、より低い分離レベル READ COMMITTED を使用してみる。
- テーブルとレコードには一定の順序でアクセスする。これによって、トランザクションが整然と待ち行列を形成するようになり、デッドロックが発生しなくなる。
- 適切に選択されたインデックスをテーブルに追加する。これによって、クエリがスキャンするインデックスレコードの数、および設定するロックの数を削減できる。 EXPLAIN SELECT を使用して、MySQL がクエリに対して適切なインデックスを選択することを確認する。
- ロックの使用を減らす。SELECT で古いスナップショットからデータが返されてもかまわない場合は、このステートメントに FOR UPDATE 節や LOCK IN SHARE MODE 節を追加しないようにする。この場合は、READ COMMITTED 分離レベルを使用するとよい。この分離レベルでは、同じトランザクション内の一貫した読み取りそれぞれが独自の新しいスナップショットを読み取るからである。
- どの方法でも解決しない場合は、テーブルレベルのロック LOCK TABLES t1 WRITE, t2 READ, ... ; [do something with tables t1 and t2 here]; UNLOCK TABLES を使ってトランザクションを直列化する。 テーブルレベルのロックによって、トランザクションが整然と待ち行列に並ぶようになり、デッドロックが回避される。LOCK TABLES は、BEGIN コマンドと同様にトランザクションを暗黙的に開始し、UNLOCK TABLES は、COMMIT でトランザクションを暗黙的に終了することに注意する。

- トランザクションを直列化するもう 1 つの方法として、レコードが 1 つしかない補助的な 'セマフォ' テーブルを作成する。各トランザクションは、このレコードを更新してから他のテーブルにアクセスする。こうして、すべてのトランザクションが直列的に実行される。 MySQL のテーブルレベルロックでは、タイムアウトを使ってデッドロックを解決する必要がある。

7.5.10. パフォーマンスチューニングのヒント

1. Unix の top または Windows のタスクマネージャで表示されたプロセスの CPU 使用率が 70% 未満である場合、おそらくそのプロセスはディスク処理である。トランザクションコミットの数が多すぎるか、バッファプールが小さすぎることが考えられる。 バッファプールは大きくした方がよいが、物理メモリの 80% を超えないようにする。

2. 複数の変更処理を 1 つのトランザクションにまとめる。InnoDB は、データベースを変更するトランザクションがコミットされるたびにログをディスクにフラッシュする必要がある。一般にディスクの回転速度は最高でも 1 秒間に 167 回転(10000rpmの規格の物の場合)であるため、ディスクがオペレーティングシステムを欺かない限り、コミットの回数も同じく 1 秒間に 167 回に制限される。

3. 最近コミットされたトランザクションの一部を失ってもかまわない場合は、my.cnf のパラメータ innodb_flush_log_at_trx_commit を 0 に設定できる。InnoDB は、いずれにしても 1 秒間に 1 回ログをフラッシュしようとする。ただし、このフラッシュは保証されない。

4. ログファイルをバッファプールと同じぐらい大きくする。InnoDB は、ログファイルの最後まで書き込むと、チェックポイントでバッファプールの変更された内容をディスクに書き込まなければならない。ログファイルが小さいと、不必要に何度もディスクへ書き込むことになる。大きなログファイルの難点は、リカバリに時間がかかることである。

5. ログバッファも大きくする必要がある（8 MB など）。

6. （3.23.39 以降に関係）Linux および Unix の一部のバージョンでは、Unix の fdatasync やその他の類似する方法を使ってファイルをディスクにフラッシュする際に、かなりの時間がかかる。 InnoDB は、デフォルトで fdatasync 関数を使用する。 データベースへ書き込む際のパフォーマンスが不満であれば、my.cnf の innodb_flush_method を O_DSYNC に設定してもかまわない。ただし、ほとんどのシステムでは O_DSYNC の方が処理が遅くなると思われる。

7. InnoDB にデータをインポートするときは、autocommit がオン(1)になっていないことを確認する。その場合、挿入のたびにディスクへのログのフラッシュが要求される。 SQL の単純なファイルインポート行の前に次の行を追加する。

   ```
   SET AUTOCOMMIT=0;
   ```

   さらにその後に次の行を追加する。

   ```
   COMMIT;
   ```

   mysqldump のオプション --opt を使用すると、上記のように SET AUTOCOMMIT=0; ... COMMIT; ラッパで囲まなくても、ダンプファイルを取得して InnoDB テーブルに高速でインポートできる。

8. 大量に INSERT する時の大規模なロールバックに注意する。InnoDB は、挿入バッファを使って挿入時のディスク I/O を削減するが、対応するロールバックではそのようなメカニズムが使用されない。ディスクバウンドのロールバックには、対応する挿入の 30 倍の時間がかかる。データベースのプロセスを強制終了しても、データベース起動時に再度ロールバックが開始されるので役に立たない。ロールバックの暴走を回避するには、ロールバックが CPU とメモリだ

けで高速に実行されるようにバッファプールを拡大するか、InnoDB データベース全体を削除するしかない。

9. 他の大規模なディスクバウンドの操作にも注意する。 DROP TABLE または TRUNCATE (MySQL-4.0 以降) を使用してテーブルを空にする。DELETE FROM yourtable は使用しない。

10. 多数のレコードを挿入する場合は、複数行の INSERT を使用して、サーバとクライアント間の通信にかかるオーバヘッドを軽減する。

```
INSERT INTO yourtable VALUES (1, 2), (5, 5);
```

この方法は、InnoDB だけでなく他のテーブル型へ挿入する場合にも有効である。

7.5.10.1. SHOW INNODB STATUS と InnoDB モニタ

バージョン 3.23.42 より、InnoDB の内部状態に関する情報を出力する InnoDB モニタが InnoDB に組み込まれました。 バージョン 3.23.52 および 4.0.3 より、SQL コマンド SHOW INNODB STATUS を使用して、標準 InnoDB モニタの出力を SQL クライアントへ取り込めるようになりました。 このデータは、パフォーマンスチューニングに役立ちます。mysql 対話型 SQL クライアントを使用している場合は、ステートメントの末尾にあるセミコロンを \G に置き換えることで、出力が判読しやすくなります。

```
SHOW INNODB STATUS\G
```

InnoDB モニタのもう 1 つの使い方として、サーバ mysqld の標準出力に InnoDB モニタから継続的にデータを書き込むことができます (注意: MySQL クライアントからは何も出力されません)。 この機能をオンにすると、InnoDB モニタは約 15 秒ごとにデータを出力するようになります。mysqld をデーモンとして実行すると、通常はこの出力が MySQL datadir 内の .err ログに出力されます。 このデータは、パフォーマンスチューニングに役立ちます。 Windows でこの出力を MS-DOS コマンドプロンプトにリダイレクトするには、MS-DOS コマンドプロンプトから --console オプションを指定して mysqld-max を実行する必要があります。

別途用意されている innodb_lock_monitor では、innodb_monitor と同じ情報に加えて、各トランザクションが設定したロックに関する情報も出力されます。

出力される情報には、以下に関するデータが含まれます。

- トランザクションのロックの取得待ち
- スレッドのセマフォ待ち
- 保留中のファイル I/O 要求
- バッファプールの統計情報
- InnoDB のメインスレッドのパージおよび挿入バッファマージ活動

InnoDB モニタは、次の SQL コマンドで起動できます。

```
CREATE TABLE innodb_monitor (a INT) TYPE = innodb;
```

また、次のコマンドで停止できます。

```
DROP TABLE innodb_monitor;
```

CREATE TABLE 構文は、MySQL SQL パーサを通して InnoDB エンジンにコマンドを渡す手段に過ぎません。作成されたテーブルは InnoDB モニタとまったく無関係です。モニタの実行中にデータベースをシャットダウンし、その後で再びモニタを起動する場合は、まず作成したテーブルを破棄しないと、新たな CREATE TABLE 発行してモニタを起動することができません。 この構文は、将来のリリースで変更される可能性があります。

次に示すのは、InnoDB モニタの出力サンプルです。

```
=====================================
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
```

この出力についていくつかの注釈があります。

- LOCKS HELD BY TRANSACTIONS セクションにロック待ちが報告されている場合は、アプリケーションでロックが競合しているおそれがある。この出力から、トランザクションのデッドロックの原因を追跡することもできる。
- univ.i に UNIV_SYNC_DEBUG を定義して InnoDB をコンパイルすると、SYNC INFO セクションに予約済みのセマフォが報告される。
- SYNC ARRAY INFO セクションには、セマフォを待っているスレッドと、スレッドがスピンまたは相互排他ロック (mutex) や 読み書きロックでの待機を必要とした回数に関する統計情報が報告される。多数のスレッドがセマフォを待っている場合は、ディスク I/O または InnoDB 内部の競合が原因となっている可能性がある。競合の原因としては、多数のクエリを並行して処理しているか、オペレーティングシステムでのスレッドのスケジューリングに問題があることが考えられる。
- CURRENT PENDING FILE I/O'S セクションには、保留中のファイル I/O 要求が列挙される。この数が多い場合は、作業負荷がディスク I/O バウンドであることを示している。
- BUFFER POOL セクションには、読み書きされたページに関する統計情報が示される。これらの数値から、クエリが現在実行しているデータファイル I/O の回数を計算できる。

7.5.11. マルチバージョニングの実装

InnoDB はマルチバージョニングのデータベースであるため、テーブルスペース内の古いバージョンのレコードに関する情報を保持しなければなりません。この情報は、ロールバックセグメントに格納されます。 ロールバックセグメントは Oracle のロールバックセグメントのデータ構造に似ています。

InnoDB は内部的に、データベースに格納されている各レコードに 2 つのフィールドを付加します。 6 バイトのフィールドには、レコードを最後に挿入または更新したトランザクションの識別子が記録されます(トランザクション ID)。削除も内部的には更新として扱われ、レコード内の特別なビットに削除済みとしてマークされます。各レコードには、ロールポインタと呼ばれる 7 バイトのフィールドも付加されます。ロールポインタは、ロールバックセグメントに書き込まれた取り消しログレコードを指します。レコードが更新された場合は、更新前のレコードの内容を再構築するために必要な情報が取り消しログレコードに記録されます。

InnoDB は、ロールバックセグメント内の情報を使用して、トランザクションのロールバックで必要な取り消し操作を実行します。また、読み取り一貫性のために、以前のバージョンのレコードを構築する場合にも、この情報が使用されます。

ロールバックセグメント内の UNDO ログは、挿入用の UNDO ログと更新用の UNDO ログに分かれています。挿入用の UNDO ログは、トランザクションロールバックにのみ必要で、トランザクションがコミットされると直ちに破棄されます。更新用の UNDO ログは、読み取り一貫性でも使用されるため、InnoDB がスナップショットを割り当てたトランザクションが存在しなくなるまで破棄できません。このようなトランザクションでは、読み取り一貫性で以前のバージョンのレコードを構築するために、更新用の UNDO ログの情報が必要となる場合があるからです。

トランザクションは定期的にコミットする必要があります。読み取り一貫性のある物のみを発行するトランザクションも同様です。 そうしないと、InnoDB が更新用 UNDO ログからデータを破棄できなくなり、ロールバックセグメントが大きくなって、テーブルスペースがいっぱいになるおそれがあります。

ロールバックセグメントにおける UNDO ログレコードの物理サイズは、通常、対応する挿入レコードまたは更新レコードよりも小さくなります。ロールバックセグメントに必要な領域を、これらのレコード長を使って、計算できます。

このマルチバージョニングの基本構想では、SQL ステートメントでレコードを削除しても、すぐにはそのレコードがデータベースから物理的に削除されません。 削除に対して書き込まれた更新取り消しログレコードを InnoDB が破棄できるようになった時点で、対応するレコードとそのインデックスレコードがデータベースから物理的に削除されます。この削除操作はパージと呼ばれます。この操作はきわめて高速で、通常は削除を行った SQL ステートメントと同程度の時間しかかかりません。

7.5.12. テーブルとインデックスの構造

MySQL は、テーブルのデータディクショナリ情報を、データベースディレクトリ内の .frm ファイルに保存します。ところが、InnoDB 型のテーブルは、テーブルスペース内の InnoDB 内部データディクショナリにも独自のエントリを持っています。MySQL は、テーブルやデータベースを破棄するときに、.frm ファイルおよび InnoDB データディクショナリ内の対応するエントリの両方を削除する必要があります。 このために、単に .frm ファイルを移動するだけではデータベース間で InnoDB テーブルを移動できません。また、MySQL 3.23.43 以下のバージョンで InnoDB 型のテーブルがあると DROP DATABASE が機能しなかったのもこれが原因です。

すべての InnoDB テーブルには、クラスタードインデックスと呼ばれる、レコードのデータを格納する特別なインデックスがあります。テーブルで PRIMARY KEY を定義すると、主キーのインデックスがクラスタードインデックスになります。

テーブルに主キーを定義しない場合は、InnoDB によって内部的にクラスタードインデックスが作成され、そこで InnoDB がテーブル内のレコードに割り当てるロー ID の順にレコードが並べられます。ロー ID は 6 バイトのフィールドで、新しいレコードが挿入されると単純に数が増加していきます。したがって、ロー ID の順に並べられたレコードは、物理的に挿入された順で並ぶことになります。

クラスタードインデックスを介したレコードへのアクセスは迅速です。これは、インデックス検索が行われるページにレコードデータが配置されるためです。他の多くのデータベースでは、データがインデックスレコードとは別のページに伝統的に格納されています。一般に、テーブルが大きい場合は、クラスタードインデックスの方が従来の方法よりもディスク I/O が少なくなります。

InnoDB では、非クラスタードインデックス (セカンダリインデックスとも呼ばれる) のレコードにレコードの主キー値が含まれています。InnoDB はこの主キー値を使用して、クラスタードインデックスからレコードを検索します。主キーが長いと、セカンダリインデックスの使用する領域が増えることに注意してください。

7.5.12.1. インデックスの物理構造

InnoDB のすべてのインデックスは B ツリーで、インデックスのレコードはツリーのリーフページに格納されます。インデックスページのデフォルトサイズは 16 KB です。新しいレコードが挿入されると、InnoDB はページの 1/16 を、将来のインデックスレコードの挿入や更新に備えて空けようとします。

インデックスレコードがシーケンシャル (昇順または降順) に挿入されると、インデックスページの約 15/16 までがいっぱいになります。 レコードがランダムに挿入された場合は、ページの 1/2 ー 15/16 までがいっぱいになります。インデックスページの使用容量が 1/2 未満になると、InnoDB はインデックスツリーを縮小してページを解放しようとします。

7.5.12.2. 挿入バッファ

データベースアプリケーションでは、主キーが一意の識別子であり、新しいレコードが主キーの昇順で挿入されることが一般的です。したがって、クラスタードインデックスへの挿入では、ディスクからのランダムな読み取りを必要としません。

一方、セカンダリインデックスは通常一意ではなく、セカンダリインデックスへの挿入は比較的ランダムに行われます。このため、InnoDB で特別なメカニズムが使用されることなく、多数のランダムなディスク I/O が発生します。

一意でないセカンダリインデックスにインデックスレコードが挿入される場合は、セカンダリインデックスページがすでにバッファプール内にあるかどうかが InnoDB によってチェックされます。すでにある場合は、InnoDB によってインデックスページに直接レコードが挿入されます。バッファプール内にインデックスページがなかった場合は、InnoDB によって特別な挿入バッファ構造にレコードが挿入されます。 挿入バッファは、その全体がバッファプール内に収まるように小さくしてあるため、このバッファへの挿入はきわめて高速です。

挿入バッファは、データベース内のセカンダリインデックスツリーに定期的にマージされます。インデックスツリーの同じページ上で複数の挿入をマージすることで、ディスク I/O を削減できます。 挿入バッファによってテーブルへの挿入速度が最大 15 倍に高められることが測定されています。

7.5.12.3. ハッシュインデックス

データベースのほぼ全体がメインメモリ内に収まる場合に、そのデータベースで最も速くクエリを実行するには、ハッシュインデックスを使用します。InnoDB には、テーブルに定義されたインデックスで実行される検索を監視するメカニズムがあり、ハッシュインデックスの構築がクエリにとって有益であると InnoDB が判断した場合は、自動的にそのインデックスが構築されます。

ただし、ハッシュインデックスは常にテーブルに存在する B ツリーインデックスを基に構築されるので注意が必要です。InnoDB は、B ツリーインデックス上で検出した検索パターンに応じて、任意の長さのキー(B ツリーに定義されたキー)の先頭部分に、ハッシュインデックスを構築できます。 ハッシュインデックスは部分的であってもかまいません。つまり、B ツリーインデックス全体をバッファプールにキャッシュする必要はありません。InnoDB は、頻繁にアクセスされるインデックスページへの要求に応じてハッシュインデックスを構築します。

ある意味では、柔軟なハッシュインデックスのメカニズムを使用して、InnoDB が十分に余裕のあるメインメモリに適応することで、メインメモリデータベースのアーキテクチャに近づいています。

7.5.12.4. 物理的なレコード構造

- InnoDB 内の各インデックスレコードには、6 バイトのヘッダが含まれる。このヘッダは、連続するレコードをリンク

するためと、行レベルロックで使用される。

- クラスタードインデックス内のレコードには、すべてのユーザ定義カラムのフィールドが含まれる。これに加えて、トランザクション ID 用の 6 バイトのフィールドと、ロールポインタ用の 7 バイトのフィールドが 1 つずつ含まれている。
- ユーザがテーブルに主キーを定義していない場合は、クラスタードインデックスの各レコードに 6 バイトのロー ID フィールドも追加される。
- セカンダリインデックスの各レコードには、クラスタードインデックスキーに対して定義されたすべてのフィールドも含まれる。
- レコードには、そのレコードの各フィールドへのポインタも含まれる。 レコード内のフィールド長の合計が 128 バイト未満の場合はポインタが 1 バイト、128 バイト以上の場合はポインタが 2 バイトになる。

7.5.12.5. InnoDB での AUTO_INCREMENT カラムの仕組み

データベース起動後に、オートインクリメントカラムが定義されたテーブル T にユーザが最初に挿入を行う際にそのカラムに明示的に値を指定しなかった場合は、InnoDB によって SELECT MAX(auto-inc-column) FROM T が実行され、その結果の値に 1 を加えた値が、このカラムおよびテーブルのオートインクリメントカウンタに割り当てられます。 これを、「テーブル T のオートインクリメントカウンタが初期化された」と表現します。

InnoDB は、新たに作成されたテーブルに対するオートインクリメントカウンタの初期化で、これと同じ手順を実行します。

ユーザが挿入時にオートインクリメントカラムに値 0 を指定すると、InnoDB は値が指定されなかったものとしてレコードを扱うことに注意してください。

オートインクリメントカウンタが初期化された後に、ユーザがオートインクリメントカラムの値を明示的に指定してレコードを挿入した時、その値が現在のカウンタ値より大きい場合は、カウンタは指定された値に設定されます。ユーザが値を明示的に指定しなかった場合は、InnoDB によってカウンタが 1 つ増加され、その新しい値がカラムに割り当てられます。

カウンタから値が割り当てられるときは、オートインクリメントのメカニズムによって、ロックとトランザクションの処理が省略されます。このため、カウンタから数値を取得したトランザクションをロールバックすると、数値の順序にずれが生じることもあります。

ユーザがオートインクリメントカラムに負の値を指定した場合、または、整数型に格納できる最大の整数値より大きな値を指定した場合の、オートインクリメントの動作は定義されていません。

7.5.13. ファイル領域の管理とディスク I/O

7.5.13.1. ディスク I/O

InnoDB は、ディスク I/O で非同期 I/O を使用します。Windows NT では、オペレーティングシステムが提供するネイティブの非同期 I/O が使用されます。 Unix では、InnoDB に組み込まれた疑似的な非同期 I/O が使用されます。InnoDB は、多数の I/O スレッドを作成して、先読みなどの I/O 操作に対応します。将来のバージョンでは、Windows NT の疑似 aio、および Unix のネイティブ aio (装備されている場合) が追加でサポートされるようになります。

Windows NT では、InnoDB はバッファなしの I/O を使用します。つまり、InnoDB が読み書きするディスクページが、オペレーティングシステムのファイルキャッシュにバッファされません。これによって、メモリの使用帯域幅をある程度節約できます。

3.23.41 より、InnoDB は二重書き込みと呼ばれる斬新なファイルフラッシュ技法を使用するようになりました。 この技法によって、オペレーティングシステムのクラッシュや停電後のリカバリがより安全になります。また、fsync 操作の必要性を軽減することで、ほとんどの Unix フレーバでパフォーマンスが向上します。

二重書き込みでは、InnoDB がデータファイルにページを書き込む前に、まず二重書き込みバッファと呼ばれる隣接するテーブルスペースに、それらのページが書き込まれます。書き込みおよび二重書き込みバッファへのフラッシュが完了した後で初めて、InnoDB はデータファイルの適切な場所にページを書き込みます。このページへの書き込みの最中にオペレーティングシステムがクラッシュした場合は、InnoDB が二重書き込みバッファから適切なページのコピーを探し出してリカバリを行います。

3.23.41 よりローデバイスもデータファイルとして使用できるようになりましたが、このテストはまだ完了していません。新しいデータファイルを作成するときに、innodb_data_file_path で指定したデータファイルサイズの直後に newraw キーワードを付加する必要があります。パーティションは、少なくとも指定したサイズと同じ大きさでなければなりません。InnoDB での 1M は 1,024 x 1,024 バイトですが、通常のディスクの仕様では、1 MB は 1,000,000 バイトを意味することに注意してください。

```
innodb_data_file_path=/dev/hdd1:5Gnewraw;/dev/hdd2:2Gnewraw
```

サーバを再起動する前に、キーワードを raw に変更する必要があります。そうしないと InnoDB がパーティションを上書きします。

```
innodb_data_file_path=/dev/hdd1:5Graw;/dev/hdd2:2Graw
```

Unix の一部のバージョンでは、ローデバイスを使用することで、バッファなしの I/O を実行できます。

ローデバイスを使用するときは、MySQL サーバを実行する OS のアカウントに、それらのパーティション(上記例では/dev/hdd1)に対して読み書きできる権限があることを確認してください。

InnoDB には、先読み方法として、シーケンシャルな先読みとランダムな先読みの 2 種類があります。シーケンシャルな先読みでは、テーブルスペース内のセグメントへのアクセスパターンがシーケンシャルであることを InnoDB が検知します。 この場合 InnoDB は、一連のデータベースページの読み取りを事前に 1 つにまとめて I/O システムに送信します。ランダムな先読みでは、テーブルスペース内のある領域がバッファプールへ完全に読み取られている最中であることを InnoDB が検知します。この場合、InnoDB は残りの読み取りを I/O システムに送信します。

7.5.13.2. ファイル領域管理

オプション設定ファイルに定義するデータファイルから、InnoDB のテーブルスペースが構成されます。これらのファイルは、単純に連結されてテーブルスペースになります。ストライピングは使用されません。 現時点では、テーブルスペースのどの位置にテーブルが割り当てられるかを定義できません。ただし、新たに作成されるテーブルスペースでは、InnoDB が末端からスペースを割り当てます。

テーブルスペースは、デフォルトサイズが 16 KB のデータベースページで構成されます。 これらのページは、64個の連続するページから成るエクステントにグループ化されます。InnoDB では、テーブルスペース内部の 'ファイル' をセグメントと呼びます。ロールバックセグメントという名前は、多少誤解を招くおそれがあります。これは、ロールバックセグメントが実際にはテーブルスペース内の多数のセグメントを含んでいるためです。

InnoDB では、各インデックスに 2 つのセグメントが割り当てられます。1 つは B ツリーの非リーフノード用、もう 1 つはリーフノード用です。これには、データを含んでいるリーフノードで連続性を高める意図があります。

テーブルスペース内でセグメントが大きくなると、InnoDB はそのセグメントに最初の 32 ページを個別に割り当てます。その後は、エクステント全体がセグメントに割り当てられます。 InnoDB では、データの連続性を確保するために、大きなセグメントに一度に最大 4 つのエクステントを追加できます。

テーブルスペースには、他のページのビットマップを含んだページがあるため、InnoDB テーブルスペース内のいくつかのエクステントは、全体としてではなく個別のページとしてのみセグメントに割り当てることができます。

クエリ SHOW TABLE STATUS FROM ... LIKE ... を発行してテーブルスペース内の空き領域を照会すると、InnoDB からテーブルスペース内の完全に空いているエクステントが報告されます。 InnoDB は、常にいくつかのエクステントをクリーンアップとその他の内部的な用途のために確保しています。これらのエクステントは空き領域に含まれません。

テーブルからデータを削除すると、InnoDB によって対応する B ツリーインデックスが縮小されます。これによって個々のページまたはエクステントがテーブルスペースに解放されて、他のユーザがその領域を利用できるようになるかどうかは、削除のパターンによって異なります。テーブルを破棄するか、またはテーブルからすべてのレコードを削除すると、他のユーザに確実に領域が解放されます。ただし、削除されたレコードは、トランザクションロールバックまたは一貫した読み取りでそのレコードが必要なくなった後のパージ操作で初めて物理的に削除されることに留意してください。

7.5.13.3. InnoDB テーブルのデフラグメント化

InnoDB テーブルのインデックスでランダムな挿入または削除が行われると、インデックスがフラグメント化されることがあります。 フラグメント化とは、ディスクでのインデックスページの物理的な順序が、ページでのレコードのアルファベット順とかけ離れていること、またはインデックスに割り当てられた 64 ページのブロック内に多数の未使用ページがあることを意味します。

インデックスのスキャンを速くするには、定期的に mysqldump を使ってテーブルをテキストファイルにダンプしてからテーブルを破棄し、ダンプからテーブルを再ロードします。 デフラグメント化のもう 1 つの方法として、テーブル変更操作 ALTER TABLE tablename TYPE=InnoDB を実行します。 これによって、MySQL がテーブルを再構築します。

インデックスへの挿入が常に昇順で行われ、レコードが必ず末尾から削除される場合は、InnoDB のファイル領域管理アルゴリズムによってインデックスのフラグメント化が発生しないことが保証されます。

7.5.14. エラー処理

InnoDB でのエラー処理は、必ずしも SQL 標準に明記されているとおりではありません。SQL-99 では、SQL ステートメントでエラーが発生した場合は、そのステートメントでロールバックを実行するように記述されています。InnoDB では、ステートメントの一部のみ、またはトランザクション全体がロールバックされることがあります。 次のリストは、InnoDB でのエラー処理の仕様です。

- テーブルスペース内でファイル領域を使い果たすと、MySQL の 'Table is full' エラーが発生し、InnoDB が SQL ステートメントをロールバックする。
- トランザクションデッドロックまたはロック待ちのタイムアウトが発生すると、InnoDB がトランザクション全体をロールバックする。
- 重複キーエラーが発生した場合は、INSERT INTO ... SELECT ... のようなステートメント内であっても、その特定のレコードの挿入のみがロールバックされる。 この仕様は、SQL ステートメントに IGNORE オプションを指定しなかった

場合のみステートメントがロールバックされるように変更される予定。

- 'row too long' エラーが発生した場合は、SQL ステートメントがロールバックされる。
- その他のエラーは主に MySQL のコードレイヤによって検出され、検出された場合は対応する SQL ステートメントがロールバックされる。

7.5.15. InnoDB テーブルの制限事項

- InnoDB テーブルはフルテキストインデックスをサポートしない。
- Windows では、InnoDB がデータベース名とテーブル名を内部的に常に小文字で格納する。バイナリ形式のデータベースを Unix と Windows の間で移動するには、すべてのテーブル名とデータベース名を小文字にする必要がある。
- 警告: MySQL システムテーブル(mysql データベースの配下のテーブル)を MyISAM から InnoDB に変換してはならない。この変換はサポートされていない。変換してしまうと、バックアップから以前のシステムテーブルをリストアするか、mysql_install_db スクリプトでシステムテーブルを再構築しない限り、MySQL を再起動できなくなる。
- SHOW TABLE STATUS から返される InnoDB テーブルの統計情報は、テーブルが確保している物理サイズを除いて正確ではない。 レコードのカウントは、SQL の最適化で使用される大まかな推定値に過ぎない。
- カラムの先頭の一部にユニークインデックスを張ろうとすると、エラーになる。

```
CREATE TABLE T (A CHAR(20), B INT, UNIQUE (A(5))) TYPE = InnoDB;
```

カラムの先頭の一部に非ユニークなインデックスを張ると、InnoDB はそのカラム全体にインデックスを作成する。

- INSERT DELAYED は、InnoDB テーブルではサポートされない。
- MySQL の LOCK TABLES 操作では、すでに完了した SQL ステートメントで設定された InnoDB の行レベルロックが考慮されない。つまり、あるテーブルで他のユーザのトランザクションが行レベルロックを設定していても、そのテーブルでテーブルロックを取得できる。したがって、テーブルで実行した操作が他のユーザのロックと衝突した場合は、操作が待機状態になる可能性がある。また、デッドロックが発生する可能性もある。ただし、InnoDB によって設定される行レベルロックでは常に完全性が配慮されているため、デッドロックによってトランザクションの完全性が損なわれることはない。 また、テーブルロックのために、他のトランザクションはテーブル上で (矛盾するロックモードで) 行レベルロックを追加で取得できなくなる。
- 1 つのテーブルに作成できるカラムは 1,000個 までである。
- DELETE FROM TABLE ではテーブルが再生成されないが、代わりにすべてのレコードが 1 つずつ削除される。この処理はそれほど速くない。MySQL の将来のバージョンでは、処理の速い TRUNCATE を使用できるようになる。
- InnoDB のデフォルトのデータベースページサイズは 16 KB。コードを再コンパイルすることで、このサイズは 8 KB 〜 64 KB に設定できる。 InnoDB の 3.23.40 以下のバージョンでは、レコードの最大長がデータベースページの半分よりもやや小さい。ソースリリース 3.23.41 より、BLOB および TEXT カラムを 4 GB 未満にすることが可能となった。レコードの合計の長さも 4 GB 未満でなければならない。InnoDB は、サイズが 128 バイト以下のフィールドを個別のページに格納しない。InnoDB が長いフィールドを別のページに格納することでレコードを変更した場合、レコードの残りの長さはデータベースページの半分より小さくなければならない。キーの最大長は 7,000 バイトである。

- 一部のオペレーティングシステムでは、データファイルが 2 GB 未満でなければならない。ログファイルを結合した大きさは、4 GB 未満でなければならない。
- テーブルスペースの最大サイズは、40 億 データベースページ。これはテーブルの最大サイズでもある。テーブルスペースの最小サイズは 10 MB。
- MySQL サーバを再起動したときに、InnoDB が AUTO_INCREMENT カラムの古い値を再使用する場合がある。
- InnoDB では、AUTO_INCREMENT カラムの最初の値を CREATE TABLE ... AUTO_INCREMENT=... (または ALTER TABLE ...) で設定できない。最初の値を設定するには、1 を差し引いた値を持つダミーのレコードを挿入し、その後でダミーのレコードを削除する。

7.5.16. InnoDB の変更履歴

7.5.16.1. MySQL/InnoDB-4.1.1 (2003 年 12 月 4 日)

- InnoDB で複数テーブルスペースを利用できるようになった。InnoDB 型の各テーブルとそのインデックスを個別の .ibd ファイルに格納し、.frm ファイルの格納先と同じ MySQL データベースディレクトリの下に配置できる。
- AUTOCOMMIT=0 の場合、またはステートメントが BEGIN ... COMMIT で囲まれている場合は、InnoDB テーブルにも MySQL のクエリキャッシュを使用できるようになった。
- バッファプールのサイズを 8 MB 未満に設定することで、InnoDB のメモリ消費量を数メガバイト削減できる。
- Windows でも ロー デバイスを使用できる。

7.5.16.2. MySQL/InnoDB-4.0.16 (2003 年 10 月 22 日)

- バグ修正: マニュアルでの記述に反し、一意の複合インデックスで一意の完全一致検索条件が使用された場合に、ロックを取得する読み取りで InnoDB が 2 つの行ロックを設定していた。単一カラムの一意のキーでは正しく動作していた。
- バグ修正: テンポラリテーブルをリカバリする目的で名称変更 #sql... -> rsql... を利用した場合に、InnoDB が row_mysql_lock_data_dictionary() でアサートしていた。
- MySQL のバグデータベースに、非クリティカルな未解決のバグがいくつか報告されている。これらのバグの修正は、次にリリースされる 4.1.1 にリソースが配分されているために延期された。

7.5.16.3. MySQL/InnoDB-3.23.58 (2003 年 9 月 15 日)

- バグ修正: mysqld 起動後の最初の B ツリーページ分割で、InnoDB がインデックスページディレクトリを破壊する可能性があった。現象としては、page0page.c の関数 page_dir_find_slot() でアサートエラーが発生していた。
- バグ修正: ごくまれではあるが、ロールバック、パージ、および SELECT が同時に発生した場合に、InnoDB が無関係なレコードを返すことがあった。

- LOCK TABLES の内部で SELECT が使用された場合に、btr0sea.c ラッチでハングが発生しないように修正された。
- バグ修正: 単一の DELETE ステートメントが、いくつかのローを削除した後に FOREIGN KEY エラーまたは Table is full エラーで失敗した場合に、MySQL が SQL ステートメント全体をロールバックしていなかった。

7.5.16.4. MySQL/InnoDB-4.0.15 (2003 年 9 月 10 日)

- バグ修正: レコードを更新したことで最大長の 8,000 バイト (BLOB および TEXT なしで) を超過した場合に、InnoDB は単にクラスタードインデックスからレコードを削除していた。同様の挿入で、InnoDB が予約されたファイル領域のエクステントをリークし、それが次の mysqld 起動時まで解放されなかった。
- バグ修正: ログファイルが比較的小さい場合に大きな BLOB 値を使用すると、大きな BLOB 操作の際に、InnoDB が最新のチェックポイントの後に作成されたログに一時的に上書きする可能性があった。その時点で InnoDB がクラッシュすると、最新のチェックポイントまでログをスキャンできないために、クラッシュリカバリが失敗していた。このバージョンより、InnoDB は最新のチェックポイントが十分に新しいことを保証するようになった。これを保証できない場合、InnoDB は MySQL の .err ログに警告を出力し、ログファイルを拡大するように勧告する。
- バグ修正: innodb_fast_shutdown=0 を設定しても作用しなかった。
- 4.0.13 で検出されたバグを修正: CREATE TABLE がコメントで終わっていると、メモリオーバランが発生する可能性があった。
- バグ修正: Windowsで InnoDB が .err ログに Operating system error number .. in a file operation を出力するときのエラー番号の説明が間違っていた。回避策: Windows のエラー番号については、http://www.innodb.com/ibman.php のセクション 13.2 を参照。
- バグ修正: 固定長の CHAR カラムで、t(a CHAR(200), PRIMARY KEY (a(10))) のようにカラムプリフィックス PRIMARY KEY を作成すると、単純な SELECT でも InnoDB がクラッシュしていた。作成されたキーが PRIMARY でない場合も、CHECK TABLE からテーブルの破損が報告されていた。

7.5.16.5. MySQL/InnoDB-4.0.14 (2003 年 7 月 22 日)

- InnoDB が SQL ステートメント SAVEPOINT および ROLLBACK TO SAVEPOINT をサポートするようになった。構文については、http://www.innodb.com/ibman.php#Savepoints を参照。
- CREATE TABLE t (a BLOB, INDEX (a(10))) のように、カラムプリフィックスキーを作成できるようになった。
- Linux および FreeBSD の最新バージョンでは、O_DIRECT を innodb_flush_method として使用することもできる。ただし、これらのオペレーティングシステムで予想されるバグに注意する。
- データページのチェックサム計算が修正された。以前は、OS のファイルシステムの破損がほとんど気付かれなかった。4.0.14 以降のバージョンから 4.0.14 未満のバージョンにダウングレードすると、最初の起動時に InnoDB から次の警告が出力されることに注意する。

```
InnoDB: Warning: an inconsistent page in the doublewrite buffer
InnoDB: space id 2552202359 page number 8245, 127'th page in dblwr buf.
```

ただし、これは危険ではないので無視してかまわない。

- バッファプール置換アルゴリズムが変更されて、LRU リストの最後の 10% に置換可能なページがない場合に、変更されたページがフラッシュされるようになった。これによって、プロセスが読み取りと書き込みを行う場合に、ディスク I/O を削減できる。
- バッファプールのチェックポイントフラッシュアルゴリズムで、フラッシュリストの最後にあるページに隣接するページもフラッシュされるようになった。これによって、データベースのシャットダウンを高速化できると同時に、InnoDB のログファイルがバッファプールに比べてかなり小さい場合にディスクの書き込みを高速化できる。
- 4.0.13 では、SHOW INNODB STATUS で最新の UNIQUE KEY エラーに関する詳細情報が出力されていたが、この情報を格納すると REPLACE の速度が大きく低下する可能性があった。この情報が格納または出力されなくなった。
- バグ修正: MySQL レプリケーションで、SET FOREIGN_KEY_CHECKS=0 が適切にレプリケートされていなかった。下位バージョンの 3.23 にはこの修正が移植されない。
- バグ修正: パラメータ innodb_max_dirty_pages_pct で、バッファプール内の空きページが考慮されていなかった。このため、バッファプール内に多数の空きページがある場合でも、必要以上にフラッシュされることがあった。回避策: SET GLOBAL innodb_max_dirty_pages_pct = 100。
- バグ修正: 大規模なインデックススキャンが行われると、セマフォ待ちが長くなるために、ファイル読み取り要求でリソースが不足し、InnoDB がアサートする可能性があった。
- バグ修正: AUTOCOMMIT=1 の場合にバイナリロギングがオンになっていないと、MySQL が LOCK TABLES の内部で更新を行う SQL ステートメントの後にコミットを実行できなかった。また、SELECT ステートメントではバイナリロギングの状態に関係なくコミットが実行されなかった。
- バグ修正: mysqld 起動後の最初の B ツリーページ分割で、InnoDB がインデックスページディレクトリを破壊する可能性があった。現象としては、page0page.c の page_dir_find_slot() 関数でアサートが発生していた。
- バグ修正: UPDATE CASCADE 節を伴う FOREIGN KEY で、親カラムの内部ストレージ長が子カラムと異なっている場合にカスケードされた更新を実行すると、子テーブルでカラム長が不適切となり、子テーブルが破損していた。MySQL の '暗黙的なカラム仕様の変更' のために固定長の CHAR カラムが内部的に VARCHAR に変更され、このエラーを引き起こす可能性がある。
- バグ修正: latin1 以外のキャラクタセットが使用された場合、および FOREIGN KEY で親カラムの内部ストレージ長が子カラムと違っている場合に、子テーブルへの挿入がすべて外部キーエラーで失敗していた。
- バグ修正: InnoDB が、クラスタードインデックスレコードが見つからないというメッセージを返したり、ごくまれではあるが、ロールバック、パージ、および SELECT が同時に発生した場合に無関係な口ーを返したりすることがあった。
- LOCK TABLES 内で SELECT が使用された場合に、btr0sea.c ラッチ上でハングが発生しないように修正された。
- バグ修正: 4.0.13 のリリースノートの記載に反し、MySQL のバイナリロギングがオンであるとグループコミットが動作しなかった。
- バグ修正: Unix で os_event_wait() が適切に機能していなかったために、各種ログ操作でリソースの枯渇が発生していた。
- バグ修正: 単一の DELETE ステートメントがいくつかのローを削除した後に FOREIGN KEY エラーまたは 'Table is full error' で失敗した場合に、MySQL は SQL ステートメント全体をロールバックせず、失敗したステートメントをバイナリログに書き込み、その際にゼロ以外の error_code を報告していた。

- バグ修正: テーブルには最大 1,000 個のカラムを収容できるが、InnoDB が CREATE TABLE でその制限をチェックしていなかったために、テーブルに対する後続の INSERT または SELECT でアサートが発生する可能性があった。

7.5.16.6. MySQL/InnoDB-3.23.57 (2003 年 6 月 20 日)

- innodb_flush_log_at_trx_commit のデフォルト値が 0 から 1 に変更された。この新しいリリースでは、my.cnf でこの値を明示的に指定しないと値 1 が設定されるため、トランザクションコミットのたびにディスクへのログのフラッシュが発生してアプリケーションの実行速度が大幅に低下する。
- バグ修正: InnoDB で、テーブルが破棄される際に pthread_mutex_destroy() が呼び出されていなかった。そのために、FreeBSD およびその他の Linux 以外の Unix でメモリリークが発生する可能性があった。
- バグ修正: 空ではないインデックス範囲に対して InnoDB がそのサイズを 0 レコードと推定すると、MySQL から誤って '空集合' が返される可能性があった。また、インデックス範囲が空であると、MySQL はネクストキーロックを実行できなかった。
- バグ修正: GROUP BY および DISTINCT が NULL 値を不等として扱うことがあった。

7.5.16.7. MySQL/InnoDB-4.0.13 (2003 年 5 月 20 日)

- InnoDB が ALTER TABLE DROP FOREIGN KEY をサポートするようになった。外部キーを破棄する場合は、SHOW CREATE TABLE を使って、内部で生成された外部キー ID を検出する必要がある。
- SHOW INNODB STATUS で、最後に検出された FOREIGN KEY エラーおよび UNIQUE KEY エラーの詳細情報が出力されるようになった。InnoDB が CREATE TABLE からエラー 150 を返した原因がわからない場合は、このステートメントを使って原因を調査できる。
- ANALYZE TABLE が InnoDB 型のテーブルでも動作するようになった。このステートメントは、各インデックスツリーをランダムに 10 箇所調べ、それに応じてインデックスのカーディナリティの推定値を更新する。これは推定値に過ぎないため、ANALYZE TABLE を実行するたびに異なる数値が生成される可能性があることに注意する。MySQL は、インデックスカーディナリティの推定値を結合の最適化でのみ使用する。適切に最適化されていない結合がある場合は、ANALYZE TABLE を試すことができる。
- InnoDB のグループコミット機能が、MySQL のバイナリログがオンになっている場合も動作するようになった。グループコミットをアクティブにするには、クライアントスレッドの数が 3 つ以上でなければならない。
- innodb_flush_log_at_trx_commit のデフォルト値が 0 から 1 に変更された。この新しいリリースでは、my.cnf でこの値を明示的に指定しないと値 1 が設定されるため、トランザクションコミットのたびにディスクへのログのフラッシュが発生してアプリケーションの実行速度が大幅に低下する。
- 設定可能な MySQL グローバルシステム変数 innodb_max_dirty_pages_pct が追加された。この変数は 0 ー 100 の整数である。 デフォルト値は 90。InnoDB のメインスレッドは、多くてもこのパーセンテージが常にフラッシュされずに残るように、バッファプールからページをフラッシュしようとする。
- innodb_force_recovery=6 の場合に、InnoDB が破損ページを二重書き込みバッファに基づいて修復しないようになった。
- バッファプール内のメモリをゼロに設定しないようになったため、InnoDB の起動が速くなった。

- バグ修正: MySQL のコメント内に 'foreign key' というキーワードがあると、FOREIGN KEY 定義用の InnoDB パーサで混乱が生じていた。
- バグ修正: FOREIGN KEY で参照されていたテーブルを破棄し、その後に一致しないカラム型で同じテーブルを作成すると、InnoDB が dict0load.c の dict_load_table() 関数でアサートする可能性があった。
- バグ修正: GROUP BY および DISTINCT が NULL 値を不等として扱うことがあった。 また、インデックスの範囲が空である場合に、MySQL がネクストキーロックを実行できなかった。
- バグ修正: MyISAM テーブルが更新されたときに現在のトランザクションはコミットされない。このため、バイナリロギングが有効であっても、CREATE TABLE では InnoDB トランザクションがコミットされない。
- バグ修正: 削除が行われたテーブルは ON DELETE SET NULL で変更できなかったが、これを可能にして、カスケードされた操作で無限ループが生じないようにした。
- バグ修正: 主キーでの一意の検索でカーソルを位置付けた後でも HANDLER PREV および NEXT を使用できるようになった。
- バグ修正: MIN() または MAX() によってデッドロックまたはロック待ちのタイムアウトが発生した場合に、MySQL がエラーを返さずに、関数の値として NULL を返していた。
- バグ修正: テーブルが破棄される際に、InnoDB が pthread_mutex_destroy() を呼び出していなかった。そのために、FreeBSD およびその他の Linux 以外の Unix システムでメモリリークが発生する可能性があった。

7.5.16.8. MySQL/InnoDB-4.1.0 (2003 年 4 月 3 日)

- InnoDB が、Windows を実行する 32 ビットの Intel コンピュータでバッファプールメモリを 64 GB までサポートするようになった。これが実現したのは、InnoDB が Windows の AWE 拡張を使用して 32 ビットプロセスの 4 GB の制限を超えるメモリに対応できるようになったため。新しい起動変数 innodb_buffer_pool_awe_mem_mb によって、AWE が有効になり、バッファプールのサイズがメガバイト単位で設定される。
- バッファヘッダとロックテーブルのサイズが削減された。InnoDB の使用するメモリが 2% 少なくなった。

7.5.16.9. MySQL/InnoDB-3.23.56 (2003 年 3 月 17 日)

- InnoDB のクエリ最適化における主要なバグを修正: SELECT ... WHERE indexcolumn < x および SELECT ... WHERE indexcolumn > x というタイプのクエリで、適切に選択できる場合でもテーブルがスキャンされる可能性があった。
- MySQL がクエリの途中で TL_IGNORE を指定して store_lock を呼び出す場合に発生していたバグが修正された。

7.5.16.10. MySQL/InnoDB-4.0.12 (2003 年 3 月 18 日)

- クラッシュリカバリの際に、InnoDB がトランザクションロールバックの進行状況をパーセンテージで出力するようになった。
- バグ/機能修正: mysql_use_result() を使用するアプリケーションプログラムで、2 つ以上の接続を使って SQL クエリを送信すると、btr0sea.c の適応的なハッシュ S-ラッチでデッドロックが発生する可能性があった。これを改善し、

mysqld が SELECT からのデータをクライアントに渡すたびに S-ラッチを解放するようにした。

- バグ修正: 空ではないインデックス範囲に対して InnoDB がそのサイズを 0 レコードと推定すると、MySQL から誤って '空集合' が返される可能性があった。また、インデックス範囲が空であると、MySQL はネクストキーロックを実行できなかった。

7.5.16.11. MySQL/InnoDB-4.0.11 (2003 年 2 月 25 日)

- 4.0.10 で検出されたバグを修正: SELECT ... FROM ... ORDER BY ... DESC が、無限ループでハングする可能性があった。
- 未解決のバグ: MySQL レプリケーションで、SET FOREIGN_KEY_CHECKS=0 が適切にレプリケートされない。

7.5.16.12. MySQL/InnoDB-4.0.10 (2003 年 2 月 4 日)

- MySQL は、INSERT INTO t1 SELECT ... FROM t2 WHERE ... で、t2 にテーブルレベルのリードロックが設定されていた。このロックが設定されなくなった。
- SHOW INNODB STATUS の最大出力長が 200 KB に拡大された。
- InnoDB のクエリ最適化における主要なバグを修正: SELECT ... WHERE indexcolumn < x および SELECT ... WHERE indexcolumn > x というタイプのクエリで、適切に選択できる場合でもテーブルがスキャンされる可能性があった。
- バグ修正: 主キーのインデックスツリーの高さが 1 である BLOB テーブルで、パージによってハングが発生する可能性があった。現象: btr_free_externally_stored_field() に設定された X-ラッチによって発生するセマフォ待ち。
- バグ修正: 新しいハンドルで InnoDB の HANDLER コマンドを使用すると、ha_innobase::change_active_index() で mysqld がクラッシュしていた。
- バグ修正: MySQL が、SELECT ステートメントの途中でクエリを推定すると、btr0sea.c の適応的なハッシュインデックスラッチで InnoDB がハングする可能性があった。
- バグ修正: 適応的なハッシュインデックスの検索がパージまたは挿入と同時に実行された場合に、InnoDB がテーブルの破損を報告し、page_dir_find_owner_slot() でアサートする可能性があった。
- バグ修正: Windows 2000 のあるファイルシステムスナップショットツールによって、InnoDB のファイル書き込みがエラー 33 ERROR_LOCK_VIOLATION で失敗する可能性があった。 同期書き込みで、InnoDB が書き込みを 1 秒間隔で 100 回再試行するようになった。
- バグ修正: REPLACE INTO t1 SELECT ... が、t1 にオートインクリメントカラムがある場合に機能しなかった。
- 未解決のバグ: MySQL レプリケーションで、SET FOREIGN_KEY_CHECKS=0 が適切にレプリケートされない。

7.5.16.13. MySQL/InnoDB-3.23.55 (2003 年 1 月 24 日)

- MySQL は、INSERT INTO t1 SELECT ... FROM t2 WHERE ... で、t2 にテーブルレベルの読み取りロックが設定されていた。このロックが設定されなくなった。

- バグ修正: 32 ビットコンピュータで、InnoDB のログファイルを結合した大きさが 2 GB 以上であると、InnoDB が誤った位置にログを書き込んでいた。このため、クラッシュリカバリと InnoDB ホットバックアップがログスキャンで失敗する可能性があった。
- バグ修正: インデックスカーソルのリストアが、理論的に失敗する可能性があった。
- バグ修正: btr0sea.c の関数 btr_search_info_update_slow でのアサートが、3 つのスレッドの競合で理論的に失敗する可能性があった。
- バグ修正: 主キーのインデックスツリーの高さが 1 である BLOB テーブルで、パージによってハングが発生する可能性があった。現象: btr_free_externally_stored_field() に設定された X-ラッチによって発生するセマフォ待ち。
- バグ修正: MySQL が、SELECT ステートメントの途中でクエリを推定すると、btr0sea.c の適応的なハッシュインデックスラッチで InnoDB がハングする可能性があった。
- バグ修正: 適応的なハッシュインデックスの検索がパージまたは挿入と同時に実行された場合に、InnoDB がテーブルの破損を報告し、page_dir_find_owner_slot() でアサートする可能性があった。
- バグ修正: Windows 2000 のあるファイルシステムスナップショットツールによって、InnoDB のファイル書き込みがエラー 33 ERROR_LOCK_VIOLATION で失敗する可能性があった。 同期書き込みで、InnoDB が書き込みを 1 秒間隔で 100 回再試行するようになった。
- 未解決のバグ: MySQL レプリケーションで、SET FOREIGN_KEY_CHECKS=0 が適切にレプリケートされない。この修正は 4.0.11 で行われ、3.23 に移植される可能性は低い。
- InnoDB page0cur.c の関数 page_cur_search_with_match で、InnoDB が同じページから永久的に動かなくなるバグが修正された。 このバグは、テーブルに複数のページがある場合のみ発生する。

7.5.16.14. MySQL/InnoDB-4.0.9 (2003 年 1 月 14 日)

- 警告メッセージ 'InnoDB: Out of memory in additional memory pool' が削除された。
- バグ修正: 32 ビットコンピュータで、InnoDB のログファイルを結合した大きさが 2 GB 以上であると、InnoDB が誤った位置にログを書き込んでいた。このため、クラッシュリカバリと InnoDB ホットバックアップが失敗する可能性があった。
- バグ修正: インデックスカーソルのリストアが、理論的に失敗する可能性があった。

7.5.16.15. MySQL/InnoDB-4.0.8 (2003 年 1 月 7 日)

- InnoDB が FOREIGN KEY (...) REFERENCES ...(...) [ON UPDATE CASCADE | ON UPDATE SET NULL | ON UPDATE RESTRICT | ON UPDATE NO ACTION] もサポートするようになった。
- テーブルとインデックスがテーブルスペース内で確保する領域が 4% 削減された。 また、既存のテーブルが確保する領域も削減された。4.0.8 にアップグレードすると、SHOW TABLE STATUS で "InnoDB free" に表示される空き領域が増える。
- バグ修正: ローの主キーを更新すると、更新されるローの二次キーを参照するすべての外部キーで外部キーエラーが発

生していた。また、参照を行う外部キー制約が、インデックスの最初のカラムのみを参照する場合に、そのインデックスに複数のカラムがあると、その他のカラムを更新する際に外部キーエラーが発生していた。

- バグ修正: 同じカラムを 2 つ含んでいるインデックスでそのカラムが更新されると、テーブルが破損する。InnoDB でこのようなインデックスが作成されないようになった。
- バグ修正: ロックを取得する SELECT でデッドロックまたはロック待ちタイムアウトが発生した場合に、.err ログから余分なエラー 149 および 150 が出力されないようになった。
- バグ修正: btr0sea.c の関数 btr_search_info_update_slow でのアサートが、3 つのスレッドの競合で理論的に失敗する可能性があった。
- バグ修正: セッションのトランザクション分離レベルを REPEATABLE READ から別のレベルに設定すると、REPEATABLE READ に戻すことができなかった。

7.5.16.16. MySQL/InnoDB-4.0.7 (2002 年 12 月 26 日)

- 4.0.7 の InnoDB は、基本的に 4.0.6 と同じ。

7.5.16.17. MySQL/InnoDB-4.0.6 (2002 年 12 月 19 日)

- innodb_log_arch_dir は、MySQL に関係しないため、my.cnf でこのパラメータを指定する必要がなくなった。
- AUTOCOMMIT=1 モードの LOAD DATA INFILE が、1 MB 分のバイナリログが書き込まれるたびに暗黙的にコミットを実行しなくなった。
- 4.0.4 で検出されたバグを修正: LOCK TABLES ... READ LOCAL では、読み取られた口ーに行口ックを設定できなかった。このため、mysqldump でデッドロックやロック待ちタイムアウトが発生していた。
- 4.0.4 で検出された 2 つのバグを修正: AUTO_INCREMENT で、REPLACE がカウンタを 1 のままにする可能性があった。 デッドロックまたはロック待ちタイムアウトでも同じ問題が発生する可能性があった。
- バグ修正: テンポラリテーブルで TRUNCATE を実行すると、InnoDB がクラッシュしていた。
- 4.0.5 で検出されたバグを修正: バイナリロギングをオフにしたまま INSERT INTO ... SELECT ... または CREATE TABLE ... SELECT ... を実行すると、btr0sea.c の 128 行目で作成されたセマフォで InnoDB がハングする可能性があった。 回避策: バイナリログをオンにする。
- バグ修正: レプリケーションで、マルチステートメントトランザクションの途中に SLAVE STOP を発行すると、SLAVE START がトランザクションの一部しか実行しなくなる可能性があった。スレーブがクラッシュ後に再起動された場合も、同様のエラーが発生する可能性があった。

7.5.16.18. MySQL/InnoDB-3.23.54 (2002 年 12 月 12 日)

- バグ修正: インデックスツリーでの範囲のエンドポイントへのパスがルート内ですでに分岐している場合に、InnoDB で短いインデックス範囲のサイズが大幅に誇張して推定されていた。 このため、SQL クエリで不必要なテーブルスキャンが実行される可能性があった。

- バグ修正: テーブルに主キーを作成せずに複数のインデックスを定義し、そのうちの少なくとも 1 つをユニークインデックスにして、そのすべてのカラムを NOT NULL として宣言すると、ORDER BY が失敗する可能性があった。
- バグ修正: ON DELETE CASCADE に関連するロック待ちタイムアウトによって、インデックスが破損する可能性があった。
- バグ修正: プライマリインデックスから一意のキーを使って SELECT を実行した場合に、検索で一致したレコードに削除マークが付いていると、InnoDB が誤って次のレコードを返す可能性があった。
- 3.23.53 で検出されたバグを修正: LOCK TABLES ... READ LOCAL では、読み取られたローに行ロックを設定できなかった。このため、mysqldump でデッドロックやロック待ちタイムアウトが発生していた。
- バグ修正: 同じカラムを 2 つ含んでいるインデックスでそのカラムが更新されると、テーブルが破損する。InnoDB でこのようなインデックスが作成されないようになった。

7.5.16.19. MySQL/InnoDB-4.0.5 (2002 年 11 月 18 日)

- InnoDB がトランザクション分離レベル READ COMMITTED および READ UNCOMMITTED をサポートするようになった。READ COMMITTED は Oracle を厳密にエミュレートしているため、Oracle から MySQL へのアプリケーションの移植が容易になる。
- デッドロックが選択的に解消されるようになった。つまり、ローの変更または挿入が少ないトランザクションを選んで犠牲にするようにした。
- 外部キーの定義で、my.cnf の lower_case_table_names の設定が考慮されるようになった。
- 外部キー定義で参照されるテーブルが作成されるテーブルと同じデータベースにある場合は、SHOW CREATE TABLE で外部キー定義にデータベース名が出力されない。
- InnoDB は、インデックスページをデータファイルに書き込む前に、そのほとんどのページに対して整合性チェックを実行する。
- innodb_force_recovery に 0 より大きい値を設定すると、InnoDB はテーブルに対して SELECT * FROM を実行する際に、破損したインデックスレコードとページをとばす。これはダンプの際に役立つ。
- InnoDB が、Windows 2000 と XP で再びバッファなしの非同期 I/O を使用するようになった。NT、95/98/ME ではバッファなしのシミュレートされた非同期 I/O のみが使用される。
- バグ修正: インデックスツリーでの範囲のエンドポイントへのパスがルート内ですでに分岐している場合に、InnoDB で短いインデックス範囲のサイズが大幅に誇張して推定されていた。このため、SQL クエリで不必要なテーブルスキャンが実行される可能性があった。 この修正は、下位の 3.23.54 にも移植される。
- 3.23.52、4.0.3、4.0.4 で検出されたバグを修正: 一部の Windows 95/98/ME コンピュータで、InnoDB の起動に時間がかかり、クラッシュすることもあった。
- バグ修正: ロック待ちの後で付与された AUTO-INC ロックが、トランザクションが終了するまで維持されていた。これによって、不必要なデッドロックが発生する可能性があった。
- バグ修正: SHOW INNODB STATUS、innodb_monitor、または innodb_lock_monitor で、1 回のレポートに数百のトランザクションを出力する必要がある場合に出力が切り捨てられると、srv0srv.c の 1621 行目で作成された mutex に対

する多数の待ちをエラーログに出力する際に InnoDB がハングしていた。

- バグ修正: Unix で SHOW INNODB STATUS を実行すると、平均ファイル読み取りサイズが常に 0 バイトとして報告されていた。
- 4.0.4 の潜在的なバグを修正: InnoDB が、MyISAM のように ORDER BY ... DESC を実行するようになった。
- バグ修正: テーブル上でロールバックが実行されているときに DROP TABLE を実行すると、クラッシュまたはハングが発生する可能性があった。これが実際にユーザにとって問題であると見なされた場合のみ、この修正が下位の 3.23 にも移植される。
- バグ修正: テーブルに主キーを作成せずに複数のインデックスを定義し、そのうちの少なくとも 1 つをユニークインデックスにして、そのすべてのカラムを NOT NULL として宣言すると、ORDER BY が失敗する可能性があった。
- バグ修正: ON DELETE CASCADE に関連するロック待ちタイムアウトによって、インデックスが破損する可能性があった。
- バグ修正: プライマリインデックスから一意のキーを使って SELECT を実行した場合に、検索で一致したレコードに削除マークが付いていると、InnoDB が次のレコードを返す可能性があった。
- 未解決のバグ: 4.0.4 で、AUTO_INCREMENT に関する 2 つのバグが検出された。1 つは REPLACE がカウンタを 1 のままにする可能性があること、もう 1 つはデッドロックまたはロック待ちタイムアウトで同じ問題が発生する可能性があることである。これらは 4.0.6 で修正される。

7.5.16.20. MySQL/InnoDB-3.23.53 (2002 年 10 月 9 日)

- Windows で、データファイルに対してバッファなしのディスク I/O を再び使用するようになった。 通常の I/O では、Windows XP および Windows 2000 での読み取りのパフォーマンスに支障が出ると思われる。
- 範囲推定機能を調整して、インデックス範囲のスキャンがフルインデックススキャンより優先されるようにした。
- innodb_force_recovery が設定されている場合でも、テーブルを破棄または作成できるようになった。これを利用して、ロールバックまたはパージでクラッシュを引き起こすテーブルを破棄したり、テーブルインポートが失敗した後のリカバリでロールバックが暴走した場合にテーブルを破棄することができる。
- 3.23.52、4.0.3、4.0.4 で検出されたバグを修正:一部の Windows 95/98/ME コンピュータで、InnoDB の起動に時間がかかり、クラッシュすることもあった。
- バグ修正: 高速シャットダウン (デフォルト) が、パージまたは挿入バッファのマージによって遅くなることがあった。
- バグ修正: 読み取り一貫性で、可視のローが存在しないテーブルに対して大規模な SELECT を実行すると、btr0cur.c の 310 行目で長時間 (600 秒より長い) のセマフォ待ちが発生する可能性があった。
- バグ修正: ロック待ちの後で付与された AUTO-INC ロックが、トランザクションが終了するまで維持されていた。 これによって、不必要なデッドロックが発生する可能性があった。
- バグ修正: LOCK TABLES 内でテンポラリテーブルを作成し、そのテーブルを使用すると、ha_innobase.cc でアサートエラーが発生していた。

- バグ修正: SHOW INNODB STATUS、innodb_monitor、または innodb_lock_monitor で、1 回のレポートに数百のトランザクションを出力する必要がある場合に出力が切り捨てられると、srv0srv.c の 1621 行目で作成されたミューテックスに対する多数の待ちをエラーログに出力する際に InnoDB がハングしていた。
- バグ修正: Unix で SHOW INNODB STATUS を実行すると、平均ファイル読み取りサイズが常に 0 バイトとして報告されていた。

7.5.16.21. MySQL/InnoDB-4.0.4 (2002 年 10 月 2 日)

- Windows で、バッファなしのディスク I/O が再び使用されるようになった。通常の I/O では、Windows XP および Windows 2000 での読み取りのパフォーマンスに支障が出ると思われる。
- InnoDB テーブルの最大キー長が 500 バイトから 1,024 バイトに拡大された。
- SHOW TABLE STATUS のテーブルコメントフィールドが拡大されて、外部キー定義を 16,000 文字まで出力できるようになった。
- ローの挿入が実行直後にエラーで失敗した場合に、オートインクリメントカウンタがインクリメントされないようになった。
- innodb_force_recovery が設定されている場合でも、テーブルを破棄または作成できるようになった。これを利用して、ロールバックまたはパージでクラッシュを引き起こすテーブルを破棄したり、テーブルインポートが失敗した後のリカバリでロールバックが暴走した場合にテーブルを破棄することができる。
- バグ修正: 4.0.3 で ORDER BY primarykey DESC を使用すると、btr0pcur.c の 203 行目でアサートエラーが発生していた。
- バグ修正: 高速シャットダウン (デフォルト) が、パージまたは挿入バッファのマージによって遅くなることがあった。
- バグ修正: 読み取り一貫性で、可視のローが存在しないテーブルに対して大規模な SELECT を実行すると、btr0cur.c の 310 行目で長時間 (600 秒より長い) のセマフォ待ちが発生する可能性があった。
- バグ修正: MySQL クエリキャッシュが使用された場合に、ON DELETE CASCADE または ...SET NULL で変更が実行されてもクエリキャッシュが無効にならなかった。
- バグ修正: LOCK TABLES 内でテンポラリテーブルを作成し、そのテーブルを使用すると、ha_innodb.cc でアサートエラーが発生していた。
- バグ修正: innodb_flush_log_at_trx_commit を 1 に設定すると、SHOW VARIABLES でその値が1,600 万として表示されていた。

7.5.16.22. MySQL/InnoDB-4.0.3 (2002 年 8 月 28 日)

- 挿入の際に、ロックを取得する読み取り、更新、または削除を待ってそのネクストキーロックを解放する必要がある場合に、不必要なデッドロックが発生しないようになった。
- MySQL の HANDLER SQL コマンドが、InnoDB 型のテーブルでも動作するようになった。InnoDB は、HANDLER の

読み取りを常に一貫した読み取りとして実行する。HANDLER は、ダイレクトアクセスパスとしてテーブルのインデックスを個別に読み取る。場合によっては、HANDLER をサーバ側カーソルの代わりに使用できる。

- 4.0.2 でのバグを修正: 単純な挿入であっても、AIX バージョンがクラッシュする可能性があった。
- バグ修正: DROP TABLE で、テーブル名に文字コードが 127 より大きい文字を使用すると、pars0sym.c の 155 行目で InnoDB がアサートする可能性があった。
- ソースからのコンパイルで、HP-UX-11 および HP-UX-10.20 の両方に作業バージョンが提供されるようになった。以前は、4.0.2 のソースが 11 でのみ機能し、3.23.52 のソースが 10.20 でのみ機能していた。
- バグ修正: 64 ビットの Solaris でコンパイルすると、InnoDB の起動時にバスエラーが発生していた。

7.5.16.23. MySQL/InnoDB-3.23.52 (2002 年 8 月 16 日)

- 3.23 の機能セットは、このバージョンより凍結される。新しい機能は 4.0 ブランチで導入され、3.23 ブランチではバグの修正のみが行われる。
- CPU バウンドの結合クエリの多くで、実行が速くなった。Windows では、その他の CPU バウンドのクエリも実行が速くなった。
- 新たな SQL コマンド SHOW INNODB STATUS によって、InnoDB モニタの出力がクライアントに返されるようになった。InnoDB モニタで、最後に検出されたデッドロックに関する詳細情報が出力されるようになった。
- InnoDB では、SQL クエリオプティマイザがインデックスのみの範囲スキャンを多用する代わりにフルテーブルスキャンを使用していた。これが修正された。
- BEGIN および COMMIT がトランザクション周辺のバイナリログに追加されるようになった。 MySQL レプリケーションでトランザクションの境界が考慮されるようになった。これで、レプリケーションスレーブでユーザに半分のトランザクションが表示されなくなる。
- クラッシュリカバリで、レプリケーションスレーブがマスタバイナリログのどの位置までリカバリできたかが出力されるようになった。
- 新たな設定 innodb_flush_log_at_trx_commit=2 によって、InnoDB がコミットのたびにオペレーティングシステムのファイルキャッシュにログを書き込むようになった。その速度は、設定 innodb_flush_log_at_trx_commit=0 とほぼ同じである。また設定 2 には、クラッシュが発生してもオペレーティングシステムがクラッシュしなければコミットされたトランザクションが失われないという優れた特徴がある。 オペレーティングシステムのクラッシュまたは停電が発生した場合は、設定 2 の安全性が設定 0 より低くなる。
- ログブロックにチェックサムフィールドが追加された。
- 外部キーの規則が考慮されない任意の順序でのテーブルインポートでは、SET FOREIGN_KEY_CHECKS=0 が役立つ。
- セカンダリインデックスに UNIQUE 制約を設定している場合に SET UNIQUE_CHECKS=0 を指定すると、InnoDB へのテーブルのインポートが速くなる。 このフラグは、入力レコードが UNIQUE 制約に違反していないことが確実である場合のみ使用できる。
- SHOW TABLE STATUS で、想定される ON DELETE CASCADE や ON DELETE SET NULL もテーブルのコメントフ

ィールドに列挙されるようになった。

- InnoDB 型のテーブルで CHECK TABLE を実行すると、すべてのテーブルのハッシュインデックスもチェックされるようになった。
- ON DELETE CASCADE または SET NULL を定義し、親レコードで参照キーを更新すると、InnoDB によって子レコードが削除または更新されていた。この動作が変更され、SQL-92 に準拠するようになった。つまり、エラー 'Cannot delete parent row' が表示されるようになった。
- オートインクリメントのアルゴリズムが改善され、最初の挿入または SHOW TABLE STATUS でテーブルのオートインクリメントカウンタが初期化されるようになった。 これによって、SHOW TABLE STATUS で突然デッドロックが発生することがほぼなくなった。
- データファイルへの読み取りと書き込みに使用されていた一部のバッファが調整された。 これによって、Linux でバッファなしのローデバイスをデータファイルとして使用できるようになった。
- バグ修正: 大文字と小文字を変更するだけの目的でテーブルの主キーを更新した場合に、主に page0page.ic の 515 行目でアサートエラーが発生する可能性があった。
- バグ修正: 外部キー制約で参照されているレコードを削除または更新する場合に、外部キーチェックでロック待ちが発生すると、そのチェックから誤った結果が報告されることがある。これは、ON DELETE... 操作にも影響する。
- バグ修正: InnoDB でデッドロックまたはロック待ちタイムアウトエラーが発生すると、InnoDB はトランザクション全体をロールバックするが、MySQL が以前の SQL ステートメントを (InnoDB がそれらをロールバックした後も) バイナリログに書き出す可能性があった。このため、たとえばレプリケートされたデータベース間で同期が取れなくなることがあった。
- バグ修正: コミットの途中でデータベースがクラッシュした場合に、リカバリでテーブルスペースのページがリークされることがあった。
- バグ修正: latin1 以外のキャラクタセットを my.cnf で指定した場合に、マニュアルの記述に反して、外部キー制約の文字列型カラムの長さの指定を参照元テーブルと参照先テーブルで同じにしなければならなかった。
- バグ修正: CREATE TABLE の実行中に DROP TABLE または DROP DATABASE を実行すると、失敗する可能性があった。
- バグ修正: 32 ビットコンピュータで 2 GB を超えるバッファプールを設定した場合に、InnoDB が buf0buf.ic の 214 行目でアサートしていた。
- バグ修正: 64 ビットコンピュータで、あるカラムに SQL NULL を含んでいるレコードを更新した場合に、UNDO ログと通常のログが破損する可能性があった。
- バグ修正: innodb_log_monitor でページに対してロック出力を抑制するとハングが発生していた。
- バグ修正: HP-UX-10.20 バージョンでは、ミューテックスによって InnoDB コードのあらゆる部分でリーク、競合、クラッシュが発生していた。
- バグ修正: AUTOCOMMIT モードで SELECT を実行した直後に RENAME TABLE を実行すると、RENAME が失敗し、MySQL からエラー 1192 が返されていた。
- バグ修正: 64 ビットの Solaris でコンパイルすると、InnoDB の起動時にバスエラーが発生していた。

7.5.16.24. MySQL/InnoDB-4.0.2 (2002 年 7 月 10 日)

- InnoDB は基本的に InnoDB-3.23.51 と同じ。
- innodb_data_file_path が指定されないときは、InnoDB がデータベース作成時に 10 MB の自動拡張データファイル ibdata1 を MySQL のデータディレクトリに作成するようになった。4.0.1 では、ファイルが 64 MB に固定され、自動的に拡張されなかった。

7.5.16.25. MySQL/InnoDB-3.23.51 (2002 年 6 月 12 日)

- バグ修正: テーブル内の BLOB または TEXT カラムに SQL NULL 値が含まれていると、結合でこれらのカラムをコピーする際にセグフォルトが発生する可能性があった。
- バグ修正: ON DELETE CASCADE を指定して自己参照型の外部キー制約をテーブルに追加した場合にレコードを削除すると、カスケード削除のために InnoDB が同じレコードを 2 回削除しようとしてアサートエラーが発生していた。
- バグ修正: MySQL の 'ユーザレベルロック' を使用して接続を閉じると、InnoDB が ha_innobase.cc の 302 行目でアサートする場合がある。

7.5.16.26. MySQL/InnoDB-3.23.50 (2002 年 4 月 23 日)

- InnoDB が自動拡張する最後のデータファイルをサポートするようになった。データベース起動時に、データファイル全体を先に割り当てる必要がなくなった。
- InnoDB ホットバックアップツールを使いやすくするためにいくつかの変更が行われた。この独立した non-free ツールでは、サーバをシャットダウンしたりロックを設定したりすることなく、データベースをオンラインでバックアップできる。
- 自動拡張するデータファイルで InnoDB ホットバックアップツールを実行する場合は、このツールをバージョン ibbackup-0.35 にアップグレードする必要がある。
- クラッシュリカバリでのログスキャンが速くなった。
- このサーババージョンより、ホットバックアップツールがバックアップ InnoDB データファイルの末尾の未使用部分を切り捨てるようになった。
- ホットバックアップツールを機能させるために、Windows でバッファなしの I/O またはネイティブの非同期 I/O ではなく、Unix のようにシミュレートされた非同期 I/O が使用されるようになった。
- 外部キーで ON DELETE CASCADE または ON DELETE SET NULL 節を定義できるようになった。
- 外部キー制約が、ALTER TABLE および CREATE INDEX で生き残るようになった。
- チェックされる外部キーまたは参照キーに含まれるカラム値のいずれかが SQL NULL であると、外部キーチェックが抑制される。この動作は、Oracle などと互換である。
- SHOW CREATE TABLE で外部キー制約も列挙されるようになった。mysqldump でも、テーブル定義内の外部キーが出力されるようになった。

- 新しい外部キー制約を、ALTER TABLE ... ADD CONSTRAINT FOREIGN KEY (...) REFERENCES ... (...) で追加できるようになった。
- 外部キーの定義で、テーブル名とカラム名をバッククォートで囲めるようになった。
- MySQL のコマンド SET TRANSACTION ISOLATION LEVEL ... が、InnoDB テーブルに次のように作用するようになった。トランザクションが SERIALIZABLE として定義されている場合、InnoDB はすべての読み取り一貫性に概念上 LOCK IN SHARE MODE を追加する。トランザクションがそれ以外の分離レベルで定義されている場合、InnoDB はそのデフォルトのロック方法である REPEATABLE READ に従う。
- オートインクリメントカウンタがすでに初期化されている場合に、SHOW TABLE STATUS でオートインクリメントインデックスの末尾に x-ロックが設定されなくなった。これによって、SHOW TABLE STATUS で突然デッドロックが発生することがほぼなくなる。
- バグ修正: CREATE TABLE ステートメントで文字列 'foreign' の後に空白以外の文字が続いていると、FOREIGN KEY パーサで混乱が生じ、テーブル作成がエラー 150 で失敗していた。

7.5.16.27. MySQL/InnoDB-3.23.49 (2002 年 2 月 17 日)

- バグ修正: クエリが実行されているデータベースに対して DROP DATABASE を呼び出すと、MySQL サーバがクラッシュまたはハングする可能性があった。クラッシュについては修正されたが、完全に修正されるには、MySQL コードレイヤでの変更を待つ必要がある。
- バグ修正: Windows で DROP DATABASE を実行するには、データベース名を小文字で指定する必要があった。3.23.49 で修正: Windows では大文字と小文字が区別されなくなった。Unix では引き続きデータベース名で大文字と小文字が区別される。
- バグ修正: latin1 以外のキャラクタセットをデフォルトのキャラクタセットとして定義すると、外部キー制約の定義が dict0crea.c でのアサートエラーで失敗し、内部エラー 17 が報告される可能性があった。

7.5.16.28. MySQL/InnoDB-3.23.48 (2002 年 2 月 9 日)

- SQL オプティマイザで、テーブルスキャンよりも頻繁なインデックス検索が優先されるようになった。
- マルチプロセッサ搭載の Linux コンピュータで、複数の大規模な SELECT クエリが同時に実行された場合のパフォーマンス上の問題が解消された。 CPU バウンドの大規模な SELECT クエリの実行も、すべてのプラットフォームで一様に速くなる。
- MySQL のバイナリログが使用される場合に、InnoDB がクラッシュリカバリ後に最新の MySQL バイナリログファイルの名前と、そのファイルのどの位置までリカバリできたか (= バイトオフセット) を出力するようになった。これは、レプリケーションでマスタデータベースとスレーブデータベースを再同期させる場合などに役立つ。
- インストールの問題で役立つように、よりわかりやすいエラーメッセージが追加された。
- InnoDB テーブルスペース内で親テーブルを失った MySQL テンポラリテーブルもリカバリできるようになった。
- 外部キーの宣言で、参照元整数カラムと参照先整数カラムで符号の有無が異なっている場合に、InnoDB がその宣言を阻止するようになった。

- バグ修正: SHOW CREATE TABLE または SHOW TABLE STATUS を呼び出すと、メモリが破損し、mysqld がクラッシュする可能性があった。 特に、SHOW CREATE TABLE を頻繁に呼び出す mysqldump では危険性が高かった。
- バグ修正: Unix で InnoDB テーブルに対して ALTER TABLE とクエリを同時に実行した場合に、mysqld が row0row.c の 474 行目でアサートエラーを起こしてクラッシュする可能性があった。
- バグ修正: オートインクリメントカラムを含んでいる複数のテーブルへの挿入が 1 つの LOCK TABLES 内にラップされていると、InnoDB が lock0lock.c でアサートしていた。
- 3.23.47 では、一意のセカンダリインデックスに複数の NULL 値があってもかまわなかった。 ところが、CHECK TABLE からはテーブルが破損していると報告されていた。 CHECK TABLE がこの状況でエラーを報告しなくなった。
- バグ修正: ビッグエンディアンを採用する Sparc などのプロセッサで SHOW VARIABLES を実行すると、innodb_flush_log_at_trx_commit などのブール値の起動パラメータが、オンであっても常に OFF と表示されていた。
- バグ修正: Windows NT/2000 で mysqld-max-nt をサービスとして実行した場合に、そのサービスのシャットダウンが、InnoDB シャットダウンの完了を待たずに実行されていた。

7.5.16.29. MySQL/InnoDB-3.23.47 (2001 年 12 月 28 日)

- リカバリが、特に負荷の軽いシステムで速くなった。これは、バックグラウンドでより頻繁にチェックポイントが実行されるようになったためである。
- 一意のセカンダリインデックスで類似する複数のキー値に SQL NULL が含まれていれば、InnoDB がそれらの値を許可するようになった。これで、MyISAM テーブルと同じ規則になった。
- BLOB を含んでいるテーブルについて InnoDB がレコード数を推定する精度が向上した。
- 外部キー制約で、InnoDB がカラム名 (Windows ではテーブル名も) の大文字と小文字を区別するようになった。
- InnoDB では、CHAR 型の外部キーカラムによる VARCHAR 型のカラムの参照、およびその逆の参照が可能である。MySQL によって、一部のカラムの型が CHAR と VARCHAR の間で暗黙的に変更される。この変更が外部キーの宣言を妨げることがなくなった。
- ログファイルの破損に対するリカバリが強化された。
- テンポラリテーブルを生成するクエリから、不要な統計情報の計算が取り除かれた。一部の ORDER BY クエリと DISTINCT クエリの実行が速くなった。
- InnoDB テーブルのテーブルスキャンが主キーを通して実行されていることを、MySQL が認識するようになった。これによって、一部の ORDER BY クエリでソートが不要になる。
- InnoDB テーブルの最大キー長が再び 500 バイトに制限された。 MySQL インタープリタはこれより長いキーを処理できない。
- innodb_lock_wait_timeout のデフォルト値が無限から 50 秒へと変更され、innodb_file_io_threads のデフォルト値が 9 から 4 に変更された。

7.5.16.30. MySQL/InnoDB-4.0.1 (2001 年 12 月 23 日)

- InnoDB は 3.23.47 と同じ。
- 4.0.0 では、MySQL インタープリタが構文 LOCK IN SHARE MODE を認識しなかった。これが修正された。
- 4.0.0 では、トランザクションテーブルに対して複数テーブルの削除を実行できなかった。 これが修正された。

7.5.16.31. MySQL/InnoDB-3.23.46 (2001 年 11 月 30 日)

- 3.23.45 と同じ。

7.5.16.32. MySQL/InnoDB-3.23.45 (2001 年 11 月 23 日)

- これはバグ修正用リリースである。
- バージョン 3.23.42 〜 .44 では、Windows でテーブルを作成する際に、データベース名に小文字を使用しないとテーブルにアクセスできなかった。これが 3.23.45 で修正された。
- InnoDB が stdout と stderr を 10 秒おきにフラッシュするようになった。これらがファイルにリダイレクトされれば、エディタでファイルの内容を参照しやすくなる。
- .44 では、.frm ファイルは作成されているが InnoDB に存在しないテーブルを破棄すると、trx0trx.c の 178 行目でアサートエラーが発生していたが、これが修正された。
- 挿入バッファでのバグが修正された。 挿入バッファツリーが矛盾した状態となり、クラッシュおよびリカバリのクラッシュも引き起こす可能性があった。このバグは、特にテーブルの大規模なインポートや変更で発生していた。
- リカバリでのバグが修正された。InnoDB が、バッファプールに空きブロックがないことを通知する警告メッセージを表示して絶えず無限ループに入る可能性があった。
- バグ修正: InnoDB 型のテンポラリテーブルを作成し、そのテーブルに対して ALTER TABLE を実行すると、MySQL サーバがクラッシュする可能性があった。
- MySQL のシステムテーブル 'mysql.user'、'mysql.host'、または 'mysql.db' を InnoDB 型で作成できないようになった。
- srv0srv.c の 1728 行目でアサートエラーを引き起こすおそれのある 3.23.44 でのバグが修正された。

7.5.16.33. MySQL/InnoDB-3.23.44 (2001 年 11 月 2 日)

- InnoDB テーブルで外部キー制約を定義することができる。例: FOREIGN KEY (col1) REFERENCES table2(col2)
- ファイルシステムで許可されていれば、4 GB を超えるデータファイルを作成することができる。
- InnoDB モニタが改善された。これには、InnoDB 内部データディクショナリの内容を出力するための新たなパラメータ innodb_table_monitor も含まれる。
- DROP DATABASE が InnoDB テーブルでも動作するようになった。
- デフォルトのキャラクタセットである latin1 のアクセント文字が、MySQL の順序に従って並べられるようになった。

注意: latin1 を使用している場合に、インデックスの付いた CHAR カラムに 127 より大きなコードの文字を挿入したときは、3.23.43 にアップグレードするときに、テーブルに対して CHECK TABLE を実行する必要がある。CHECK TABLE からエラーが返された場合は、テーブルを破棄してもう一度インポートする必要がある。

- InnoDB によるテーブルカーディナリティの推定値の計算精度が向上した。
- デッドロックの解決方法が変更された。 .43 ではデッドロックが発生すると SQL ステートメントのみがロールバックされるが、.44 ではトランザクション全体がロールバックされる。
- デッドロック、ロック待ちタイムアウト、および外部キー制約違反 (親レコードがない、子レコードが存在する) が、それぞれネイティブの MySQL エラーコード 1213、1205、1216、1217 を返すようになった。
- my.cnf の新しいパラメータ innodb_thread_concurrency は、並行処理が頻繁に行われる環境でのパフォーマンスチューニングに役立つ。
- my.cnf の新しいオプション innodb_force_recovery は、破損したデータベースからテーブルをダンプする際に役立つ。
- my.cnf の新しいオプション innodb_fast_shutdown は、シャットダウンを高速化する。通常、InnoDB はシャットダウン時に完全なパージと挿入バッファのマージを実行する。
- 最大キー長が以前の制限値である 500 バイトから 7,000 バイトに引き上げられた。
- 複数行挿入を伴うオートインクリメントカラムのレプリケーションでのバグが修正された。
- インデックスの付いたセカンダリカラムの更新で、大文字と小文字が変更されるときのバグが修正された。
- データファイルの数が 24 個を超えた場合に発生するハングが修正された。
- 空のテーブルから MAX(col) が選択される際に、col が複合インデックスの最初のカラムでない場合に発生していたクラッシュが修正された。
- クラッシュの原因となりうるパージでのバグが修正された。

7.5.16.34. MySQL/InnoDB-3.23.43 (2001 年 10 月 4 日)

- 基本的に InnoDB-3.23.42 と同じ。

7.5.16.35. MySQL/InnoDB-3.23.42 (2001 年 9 月 9 日)

- 8,000 バイトを超えるレコードの主キーが更新された場合にテーブルが破損するバグが修正された。
- InnoDB モニタの種類が、innodb_monitor、innodb_lock_monitor、innodb_tablespace_monitor の 3 つになった。innodb_monitor で、バッファプールのヒット率、および挿入、更新、削除、読み取りが行われたレコードの総数が新たに出力されるようになった。
- RENAME TABLE でのバグが修正された。
- オートインクリメントカラムを伴うレプリケーションでのバグが修正された。

7.5.16.36. MySQL/InnoDB-3.23.41 (2001 年 8 月 13 日)

- 4 GB 未満のレコードをサポートする。以前の制限は 8,000 バイト。
- 二重書き込みファイルフラッシュ方式を採用した。
- ローデバイスをデータファイルとしてサポートする。
- InnoDB モニタ。
- ハングの原因となる複数のバグ、および ORDER BY のバグ ('Sort aborted') が修正された。

7.5.16.37. MySQL/InnoDB-3.23.40 (2001 年 7 月 16 日)

- 発生頻度の低い数個のバグのみが修正された。

7.5.16.38. MySQL/InnoDB-3.23.39 (2001 年 6 月 13 日)

- CHECK TABLE が InnoDB テーブルでも動作するようになった。
- my.cnf の新しいパラメータ innodb_unix_file_flush_method が追加された。このパラメータは、ディスク書き込みパフォーマンスのチューニングに使用できる。
- オートインクリメントカラムが、トランザクションメカニズムを通さずに新しい値を取得するようになった。これによって、新しい値を割り当てる際の CPU 時間が削減され、トランザクションデッドロックが排除される。
- 3.23.38 での複数のバグ、特にロールバックでのバグが修正された。

7.5.16.39. MySQL/InnoDB-3.23.38 (2001 年 5 月 12 日)

- 新たな構文 SELECT ... LOCK IN SHARE MODE が導入された。
- ディスクへの書き込みが行われるたびに、InnoDB が fsync() を呼び出し、書き込みまたは読み取りを行うすべてのデータベースページのチェックサムを計算してディスクの不具合を明らかにするようになった。
- 複数のバグが修正された。

7.5.17. InnoDB についての問い合わせ先

InnoDB エンジンの製造元である Innobase Oy の問い合わせ先は次のとおりです。 Web サイト: http://www.innodb.com/。
電子メール: <sales@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 または BerkeleyDB テーブル

7.6.1. BDB テーブルの概要

BerkeleyDB (http://www.sleepycat.com/ で入手可能) は、MySQL にトランザクションストレージエンジンをもたらしました。このストレージエンジンのサポートは、バージョン 3.23.34 から MySQL ソースディストリビューションに組み込まれるようになり、MySQL-Max バイナリでアクティブ化されます。通常、このストレージエンジンは略して BDB と呼ばれます。

BDB テーブルはクラッシュに対する耐久性が高く、トランザクションでの COMMIT および ROLLBACK 操作にも対応します。 MySQL ソースディストリビューションに付属する BDB ディストリビューションには、MySQL でより円滑に動作するための小規模なパッチがいくつか適用されています。 パッチが適用されていない BDB バージョンは、MySQL で使用できません。

MySQL AB は、Sleepycat 社と密接に協力しながら、MySQL/BDB インタフェースの品質維持に努めています。

BDB テーブルのサポートに関しては、ユーザによる問題の特定を支援すると共に、BDB テーブルが関係するあらゆる問題に対する再現可能なテストケースの作成を支援しています。作成されたテストケースは Sleepycat 社に送られ、同社の支援を受けながら問題を特定して修正します。このように 2 段階の作業になるため、BDB テーブルの問題は、他のストレージエンジンよりも修正に若干時間がかかる場合があります。 ただし、BerkeleyDB コード自体はこれまでに MySQL 以外の多くのアプリケーションで使用されているため、大きな問題が発生することは考えられません。 See 項1.4.1.「MySQL AB によって提供されるサポート」。

7.6.2. BDB のインストール

BerkeleyDB をサポートする MySQL のバイナリバージョンをダウンロードした場合は、MySQL のバイナリバージョンをインストールするための指示に従ってください。 See 項2.2.9.「MySQL バイナリディストリビューションのインストール」。 See 項4.8.5.「mysqld-max (拡張 mysqld サーバ) 」。

Berkeley DB をサポートする MySQL をコンパイルするには、MySQL バージョン 3.23.34 以降をダウンロードし、--with-berkeley-db オプションを使って MySQL をコンフィギャします。 See 項2.3.「MySQL ソースディストリビューションのインストール」。

```
cd /path/to/source/of/mysql-3.23.34
./configure --with-berkeley-db
```

詳しい最新情報については、BDB ディストリビューション付属のマニュアルを参照してください。

Berkeley DB 自体は十分にテストされていて信頼できますが、MySQL とのインタフェースはまだガンマ品質と見なされています。 当社は、このインタフェースの 1 日も早い安定化を目指して、積極的に改善と最適化を行っています。

7.6.3. BDB 起動オプション

AUTOCOMMIT=0 で実行している場合、BDB テーブルでの変更は COMMIT を実行するまで反映されません。コミットの代わりに ROLLBACK を実行すると、変更が無効になります。 See 項6.7.1.「START TRANSACTION、COMMIT、

ROLLBACK の各構文」。

AUTOCOMMIT=1（デフォルト）で実行している場合は、変更が直ちにコミットされます。SQL コマンド BEGIN WORK で拡張トランザクションを開始できます。その場合、COMMIT を実行するまで（または変更の ROLLBACK を実行するまで）変更がコミットされません。

次に示す mysqld のオプションを使用すると、BDB テーブルの動作を変更できます。

オプション	説明
--bdb-home=directory	BDB テーブルのベースディレクトリ。--datadir に使用するディレクトリと同じでなければならない。
--bdb-lock-detect=#	Berkely のロック検出。DEFAULT、OLDEST、RANDOM、または YOUNGEST のいずれか。
--bdb-logdir=directory	Berkeley DB のログファイルディレクトリ。
--bdb-no-sync	ログを同期的にフラッシュしない。
--bdb-no-recover	Berkeley DB をリカバリモードで起動しない。
--bdb-shared-data	Berkeley DB をマルチプロセスモードで起動する（Berkeley DB を初期化する際に DB_PRIVATE を使用しない）。
--bdb-tmpdir=directory	Berkeley DB のテンポラリファイルディレクトリ。
--skip-bdb	BDB テーブルの使用を無効にする。
-O bdb_max_lock=1000	ロックの数の上限を設定する。 See 項4.6.8.4.「SHOW VARIABLES」。

--skip-bdb を使用すると、MySQL は Berkeley DB ライブラリを初期化しなくなるため、大量のメモリを節約できます。このオプションを使用しているときは BDB テーブルを使用できません。BDB テーブルを作成しようとすると、代わりに MyISAM テーブルが作成されます。

BDB テーブルを使用する予定であれば、通常は --bdb-no-recover を指定せずに mysqld を起動する必要があります。ただし、BDB ログファイルが破損している場合に mysqld を起動しようとすると、問題が発生する可能性があります。 See 項2.4.2.「MySQL サーバの起動に関する問題」。

bdb_max_lock では、BDB テーブルでアクティブにできるロックの最大数（デフォルトは 10,000）を指定できます。長いトランザクションを実行しているときや、mysqld がクエリの計算で大量のレコードを調べているときに、bdb: Lock table is out of available locks または Got error 12 from ... というエラーが発生する場合は、この数値を大きくしてください。

また、大規模なトランザクションを使用する場合は、binlog_cache_size および max_binlog_cache_size を変更することもできます。 See 項6.7.1.「START TRANSACTION、COMMIT、ROLLBACK の各構文」。

7.6.4. BDB テーブルの特性

- BDB ストレージエンジンは、トランザクションをロールバックできるようにログファイルを保持している。パフォーマンスを最大限に高めるには、--bdb-logdir オプションを使ってこれらのログファイルをデータベースとは別のディスクに配置する必要がある。
- MySQL は、新しい BDB ログファイルが開始されるたびにチェックポイントを実行し、現在のトランザクションに不要なログファイルを削除する。FLUSH LOGS を任意の時点で実行して、Berkeley DB テーブルをチェックポイントす

ることもできる。

障害リカバリには、テーブルバックアップおよび MySQL のバイナリログを使用する必要がある。 See 項4.5.1.「データベースのバックアップ」。

警告: 使用中の古いログファイルを削除すると、問題が発生した場合に BDB がリカバリを実行できなくなり、データが失われるおそれがある。

- MySQL は、前に読み取られたレコードを参照できるように、各 BDB テーブルに主キーを必要とする。ユーザがこのキーを作成しない場合は、MySQL によって隠し主キーが作成され、管理される。この隠しキーは長さが 5 バイトで、挿入が試行されるたびにインクリメントされる。
- BDB テーブルでアクセスするすべてのカラムが、同じインデックスの一部または主キーの一部であれば、MySQL は実際のレコードにアクセスすることなくクエリを実行できる。MyISAM テーブルでは、カラムが同じインデックスの一部である場合のみこれが当てはまる。
- 主キーは、ローデータと共に格納されるため、他のどのキーよりも速く処理される。他のキーはキーデータ + 主キーとして格納されるため、主キーはできるだけ短くしてディスクを節約し、処理速度を高めることが重要である。
- LOCK TABLES は、他のテーブルと同様に BDB テーブルでも動作する。LOCK TABLE を使用しない場合は、MySQL によってテーブルに対する内部的な複数書き込みのロックが発行され、別のスレッドがテーブルロックを発行した場合にテーブルが適切にロックされるようになる。
- BDB テーブルでの内部ロックは、ページレベルで行われる。
- BDB テーブルではテーブル内のレコード数のカウントが管理されないため、SELECT COUNT(*) FROM table_name に時間がかかる。
- BDB テーブルのデータは、独立したデータファイルではなく B ツリーに格納されるため、順次スキャンには MyISAM テーブルよりも時間がかかる。
- アプリケーションでは、BDB テーブルの変更で発生する自動ロールバックおよびデッドロックエラーを伴う読み取りの失敗にいつでも対処できるようにしておく必要がある。
- MyISAM テーブルのキーとは異なり、キーのプリフィックスまたはサフィックスは圧縮されない。つまり、BDB テーブルでは MyISAM テーブルに比べてキー情報の領域が若干大きい。
- BDB テーブルには、多くの場合、キーツリーの途中で新たなレコードを挿入できるようにすき間が空いている。このため、BDB テーブルは MyISAM テーブルよりも若干大きくなる。
- オプティマイザには、テーブル内のおおよそのレコード数を通知する必要がある。MySQL は、挿入の回数をカウントし、その数値を BDB テーブル内の独立したセグメントで管理することで、これに対応している。DELETE または ROLLBACK ステートメントを大量に発行しない限り、この数値は MySQL オプティマイザにとって十分に正確であるが、MySQL はクローズ時までこの数値を格納しないため、MySQL が突然中断されると数値が不正確になる場合がある。この数値が 100% 正確でなくても、致命的な問題にはならない。レコードの数は、ANALYZE TABLE または OPTIMIZE TABLE を実行することで更新できる。 See 項4.6.2.「ANALYZE TABLE 構文」。 See 項4.6.1.「OPTIMIZE TABLE 構文」。
- BDB テーブルでディスクがいっぱいになると、エラー (おそらくエラー 28) が発生し、トランザクションがロールバックされる。これとは対照的に、MyISAM テーブルと ISAM テーブルでは、十分な空きディスクが確保されるのを待って、mysqld が処理を続行する。

7.6.5. 近い将来に修正する必要がある BDB の問題

- 多数の BDB テーブルを同時に開く際にかなり時間がかかる。BDB テーブルを使用する場合は、テーブルキャッシュをあまり大きくしないようにする（たとえば 256 以下にする）必要がある。また、mysql クライアントで --no-auto-rehash を使用する必要がある。これについては、4.0 で部分的に修正する予定である。
- SHOW TABLE STATUS で BDB テーブルに関して提供される情報がまだ十分でない。
- パフォーマンスを最適化する。
- テーブルをスキャンするときにページロックを一切使用しないように変更する。

7.6.6. BDB でサポートされているオペレーティングシステム

今のところ、BDB ストレージエンジンは次のオペレーティングシステムで動作することが確認されています。

- Linux 2.x Intel
- Sun Solaris（sparc および x86）
- FreeBSD 4.x/5.x（x86、sparc64）
- IBM AIX 4.3.x
- SCO OpenServer
- SCO UnixWare 7.1.x

次のオペレーティングシステムでは動作しません。

- Linux 2.x Alpha
- Linux 2.x AMD64
- Linux 2.x IA64
- Linux 2.x s390
- Max OS X

注意: 上記の一覧は完全ではありません。情報が入り次第更新する予定です。

BDB テーブルのサポート付きで MySQL をビルドする場合に、mysqld を起動するとログファイルに次のエラーが記録されることがあります。

```
bdb: architecture lacks fast mutexes: applications cannot be threaded
Can't init databases
```

これは、使用するアーキテクチャで BDB テーブルがサポートされていないことを意味します。 この場合は、BDB テーブ

ルのサポートなしで MySQL をビルドし直す必要があります。

7.6.7. BDB テーブルの制限事項

次に挙げるのは、BDB テーブルを使用する場合の制限事項です。

- BDB テーブルは、.db ファイルに、作成されたときのこのファイルへのパスを格納する（これは、シンボリックリンクをサポートするマルチユーザ環境でロックを検出できるようにするために行われる）。

  このため、BDB テーブルはディレクトリ間で移動できない。

- BDB テーブルのバックアップを取るときは、mysqldump を使用するか、すべての table_name.db ファイルと BDB ログファイルのバックアップを取る必要がある。BDB のログファイルは、ベースデータディレクトリにある log.XXXXXXXXXX（10 桁）という名前のファイルである。 BDB ストレージエンジンは未完了のトランザクションをログファイルに格納する。これらのログは、mysqld が起動されるときに存在していなければならない。

7.6.8. BDB テーブルを使用するときに起こりうるエラー

- mysqld を起動する際に hostname.err ログに次のエラーが記録されることがある。

```
bdb:  Ignoring log file: .../log.XXXXXXXXXX: unsupported log version #
```

  これは、新しいバージョンの BDB が古いログファイル形式をサポートしていないことを意味する。この場合は、データベースディレクトリからすべての BDB ログ（log.XXXXXXXXXX という形式の名前を持つファイル）を削除し、mysqld を再起動する必要がある。また、古い BDB テーブルの mysqldump --opt を実行したうえで、そのテーブルを削除し、ダンプをリストアした方がよい。

- オートコミットモードで実行していない場合に、別のトランザクションで参照されているテーブルを削除すると、MySQL エラーログに次のエラーメッセージが記録されることがある。

```
001119 23:43:56  bdb:  Missing log fileid entry
001119 23:43:56  bdb:  txn_abort: Log undo failed for LSN:
                 1 3644744: Invalid
```

  これは致命的なエラーではないが、この問題が修正されるまでは（小さな修正ではない）、オートコミットモードでないときにテーブルを削除しない方がよい。

第8章 MaxDB の概要

MaxDB は、エンタープライズレベルのデータベースです。以前に SAP DB と呼ばれていたデータベース管理システムが、MaxDB という新しい名前になりました。

8.1. MaxDB の歴史

SAP DB の歴史は 1980 年代初めにさかのぼります。その当時、SAP DB は Adabas (商用製品の 1 つ) として開発されました。以来、このデータベースは数回にわたって名称が変更されています。ドイツのウォルドルフに本社を構える SAP AGがデータベースシステムの開発を引き継いだ時点では、SAP DB という名称でした。

SAP では、堅牢な SAP アプリケーションのすべてに対応する単一のストレージシステムとして機能するように、データベースシステムを進化させました。それが R/3 です。SAP DB は、サードパーティ製データベースシステム (Oracle、Microsoft SQL Server、IBM の DB2 など) に代わる選択肢として提供されました。2000 年 10 月、SAP AG は GNU GPL ライセンス (see 付録 H. GNU General Public License) に基づき SAP DB を公開し、オープンソースソフトウェアとしました。2003 年 10 月、SAP AG の顧客のうち 2,000 社以上は、自社のメインデータベースシステムとして SAP DB を使用していました。また、他の 2,000 社以上は、自社のメインデータベースとは別の独立したデータベースシステムとして SAP DBを使用し、APO/LiveCache ソリューションの一部としていました。

2003 年 5 月、MySQL AB と SAP AG の間で技術パートナーシップが形成されました。このパートナーシップにより、MySQL AB は SAP DB の開発をさらに推進し、名称を変更することができます。また、GNU GPL (see 付録 H. GNU General Public License) に基づきこのデータベースを使用する際の制限に拘束されることを望まない顧客に対し、名称が変更された SAP DB の商用ライセンスを販売する権利も付与されます。2003 年 8 月、MySQL AB によって SAP DB の名称は MaxDB に変更されました。

8.2. ライセンスとサポート

MySQL AB から配布された他の製品に関して提供されたライセンス ((項1.4.3.「MySQL ライセンス」) に基づき、MaxDB も使用することが可能です。したがって、GNU 一般公衆利用許諾契約書 (付録 H. GNU General Public License) と商用ライセンス (項1.4.「MySQL のサポートとライセンス」) に基づき MaxDB を使用できるようになります。

MySQL では、SAP ユーザ以外の顧客に対して MaxDB サポートを提供する予定です。

そのような再ブランド化バージョンの第 1 弾として、MaxDB 7.5.00 が 2003 年後期にリリースされました。

8.3. MaxDB の基本概念

MaxDB は、クライアント/サーバ製品として動作します。インストレーション環境でオンライントランザクションを大量に処理する必要性を満たすために開発されました。オンラインバックアップとデータベース拡張がサポートされています。Microsoft クラスタサーバは複数サーバ導入に直接対応していますが、他のフェールオーバソリューションはマニュアルでスクリプト化する必要があります。データベース管理ツールは、Windows およびブラウザベースの導入の両方で提供されます。

8.4. MaxDB と MySQL の相違点

MaxDB と MySQL の主な違いを以下に示します (概要であり、完全なものではありません)。

- MaxDB は、クライアント/サーバシステムとして動作します。MySQL は、クライアント/サーバシステムまたは組込システムとして実行することができます。
- MySQL によってサポートされているプラットフォームの一部では、MaxDB が動作しない場合もあります。たとえば、MaxDB は IBM の OS/2 では動作しません。
- MaxDB では、クライアント/サーバ通信に専用ネットワークプロトコルが使用されます。それに対し MySQLでは、TCP/IP (SSL 暗号化あり/なし)、ソケット (UNIX ライクなシステムの場合)、名前付きパイプ (Windows NT ファミリシステムの場合) のいずれかが使用されます。
- MaxDB は、ストアドプロシージャをサポートしています。MySQL の場合、ストアドプロシージャの導入はバージョン 5.0 からとなります。MaxDB では SQL 拡張を介したトリガのプログラミングもサポートしていますが、この機能は MySQL 5.1 で予定されています。MaxDB は、ストアドプロシージャ言語のデバッガを内蔵しています。また、ネストされたトリガのカスケードが可能で、アクションおよび行単位の複数トリガをサポートします。
- MaxDB のディストリビューションには、テキストベース、グラフィカルベース、Web ベースのいずれかのユーザインタフェースが含まれています。MySQL のディストリビューションには、テキストベースのユーザインタフェースしか含まれていません。グラフィカルユーザインタフェース (MySQL Control Center) は、主要ディストリビューションとは別に提供されます。MySQL の Web ベースユーザインタフェースは、サードパーティから提供されています。
- MaxDB がサポートする多数のプログラミングインタフェースは、MySQL でもサポートされています。ただし、RDO、ADO、.NET は MaxDB ではサポートされておらず、いずれも MySQL ではサポートされています。MaxDB では、C/C++ でのみ埋め込み SQL がサポートされます。
- 以下の管理機能は MaxDB に含まれていますが、MySQL には含まれていません。 時刻、イベント、警告によるジョブスケジューリングと、警告しきい値に基づくデータベース管理者へのメッセージ送信

8.5. MaxDB と MySQL の相互運用性

最初の 7.5.00 バージョンの直後に公開される MaxDB バージョンには、以下の機能が含まれる予定です。これらの機能により、MaxDB と MySQL の相互運用性が確保されます。

- MySQL プロキシでは、MySQL プロトコルを使用して MaxDB に接続できるようになります。これによって、mysql コマンドラインユーザインタフェース、mysqldump ダンプユーティリティ、または mysqlimport インポートプログラムと同様、MySQL クライアントプログラムを MaxDB に対して使用することができます。mysqldump を使用すると、データをデータベースシステムからダンプし、他のデータベースシステムにエクスポート (パイプ) するのが容易になります。
- MySQL と MaxDB の間のレプリケーションは双方向でサポートされます。つまり、MySQL と MaxDB のいずれかをマスタレプリケーションサーバとして使用することができます。レプリケーション構文を集めて拡張し、両方のデータベースシステムで同じ構文が認識されるようにすることが長期的な計画となっています。 See 項4.11.1.「はじめに」。

8.6. MaxDB 関連リンク

MaxDB 関連情報のメインページは http://www.mysql.com/maxdb です。http://www.sapdb.org の掲載情報はすべて、メインページに最終的に移動される予定です。

8.7. MaxDB の予約語

MySQL と同じく MaxDB にも、特殊な意味を持つ多数の予約語があります。 予約語は通常、データベース名やテーブル名など、識別子の名称として使用することはできません。MaxDB の予約語、予約語の使用コンテキスト、対応する予約語が MySQL に存在するかどうかの区別が、以下の表に示されています。対応する予約語が存在する場合、MySQL での意味は一致することもあれば、多少異なることもあります。MaxDB がどの点で MySQL と異なるかを示すことが主な目的なので、この一覧は完全な内容ではありません。

MySQL の予約語一覧については、See 項6.1.7.「MySQL での予約語の扱い」 を参照してください。

MaxDB で予約済	MaxDB での使用コンテキスト	対応する MySQL の予約語
@	識別子の前に付けることができる（``@table" など）。	使用できない。
ADDDATE()	SQL 関数	ADDDATE()。MySQL バージョン 4.1.1 で追加。
ADDTIME()	SQL 関数	ADDTIME()。MySQL バージョン 4.1.1 で追加。
ALPHA	SQL 関数	類似するものが存在しない。
ARRAY	データ型	実装されていない。
ASCII()	SQL 関数	ASCII()。ただし、異なった意味で実装されている。
AUTOCOMMIT	トランザクション。デフォルトでは ON。	トランザクション。デフォルトでは OFF。
BOOLEAN	カラム型。BOOLEAN は TRUE、FALSE、NULL のみを値として受け取る。	BOOLEAN を MySQL バージョン 4.1.0 で追加。BOOL のシノニムであり、TINYINT(1) にマッピングされている。NULL のほか、TINYINT と同じ範囲で整数値を受け取る。TRUE と FALSE は、1 と 0 のエイリアスとして使用することができる。
CHECK	CHECK TABLE	CHECK TABLE。使用方法は類似しているが、同一ではない。
COLUMN	カラム型	COLUMN。ノイズワード。
CHAR()	SQL 関数	CHAR()。同一の構文。使用方法は類似しているが、同一ではない。
COMMIT	データ定義クエリの発行中にトランザクションの暗黙的なコミットが発生する。	データ定義クエリの発行中（ただし、他のクエリも多数存在）にトランザクションの暗黙的なコミットが発生する。
COSH()	SQL 関数	類似するものが存在しない。
COT()	SQL 関数	COT()。構文と実装が一致する。
CREATE	SQL、データ定義言語	CREATE
DATABASE	SQL 関数	DATABASE()。DATABASE は異なったコンテキストで使用される。たとえば CREATE DATABASE
DATE()	SQL 関数	CURRENT_DATE
DATEDIFF()	SQL 関数	DATEDIFF()。MySQL バージョン 4.1.1 で追加
DAY()	SQL 関数	類似するものが存在しない。

DAYOFWEEK()	SQL 関数	DAYOFWEEK()。初日 (1) のデフォルトは MaxDB では月曜、MySQL では日曜となっている。
DISTINCT	SQL 関数 AVG, MAX, MIN, SUM	DISTINCT。ただし、SELECT DISTINCT という異なったコンテキストで使用される。
DROP	特に DROP INDEX で	DROP INDEX。使用方法は類似しているが、同一ではない。
EBCDIC()	SQL 関数	類似するものが存在しない。
EXPAND()	SQL 関数	類似するものが存在しない。
EXPLAIN	最適化	EXPLAIN。使用方法は類似しているが、同一ではない。
FIXED()	SQL 関数	類似するものが存在しない。
FLOAT()	SQL 関数	類似するものが存在しない。
HEX()	SQL 関数	HEX()。使用方法は類似しているが、同一ではない。
INDEX()	SQL 関数	INSTR() または LOCATE()。構文と意味は類似しているが、同一ではない。
INDEX	USE INDEX, IGNORE INDEX および類似するヒントが SELECT ... USE INDEX と同様、SELECT の直後で使用されている。	USE INDEX, IGNORE INDEX および類似するヒントが SELECT ... FROM ... USE INDEX の場合と同様、SELECT の FROM 節で使用されている。
INITCAP()	SQL 関数	類似するものが存在しない。
LENGTH()	SQL 関数	LENGTH()。構文は同一だが、実装が少し異なる。
LFILL()	SQL 関数	類似するものが存在しない。
LIKE	比較	LIKE。ただし、MaxDB に用意されている拡張 LIKE は MySQL の REGEX に類似している。
LIKE ワイルドカード	MaxDB では、``%"、``_"、``ctrl + アンダースコア"、``ctrl + ↑"、``*"、``?" が LIKE ワイルドカードとして比較でサポートされる。	MySQL では、``%" と ``_" が LIKE ワイルドカードとして比較でサポートされる。
LPAD()	SQL 関数	LPAD()。実装が少し異なる。
LTRIM()	SQL 関数	LTRIM()。実装が少し異なる。
MAKEDATE()	SQL 関数	MAKEDATE()。MySQL バージョン 4.1.1 で追加
MAKETIME()	SQL 関数	MAKETIME()。MySQL バージョン 4.1.1 で追加
MAPCHAR()	SQL 関数	類似するものが存在しない。
MICROSECOND()	SQL 関数	MICROSECOND()。MySQL バージョン 4.1.1 で追加
NOROUND()	SQL 関数	類似するものが存在しない。
NULL	カラム型。比較	NULL。オーバーフローまたはゼロ除算の原因となる算術演算から返される特別な NULL 値が、MaxDB でサポートされる。そのような特別な値は、MySQL ではサポートされない。
PI	SQL 関数	PI()。構文と実装は同一だが、かっこが必要とされる。

REF	データ型	類似するものが存在しない。
RFILL()	SQL 関数	類似するものが存在しない。
ROWNO	WHERE 節における述語	LIMIT 節に類似している。
RPAD()	SQL 関数	RPAD()。実装が少し異なる。
RTRIM()	SQL 関数	RTRIM()。実装が少し異なる。
SEQUENCE	CREATE SEQUENCE, DROP SEQUENCE	AUTO_INCREMENT。概念は類似しているが、実装が異なる。
SINH()	SQL 関数	類似するものが存在しない。
SOUNDS()	SQL 関数	SOUNDEX()。構文が少し異なる。
STATISTICS	UPDATE STATISTICS	ANALYZE。概念は類似しているが、実装が異なる。
SUBSTR()	SQL 関数	SUBSTRING()。実装が少し異なる。
SUBTIME()	SQL 関数	SUBTIME()。MySQL バージョン 4.1.1 で追加
SYNONYM	データ定義言語: CREATE [PUBLIC] SYNONYM、RENAME SYNONYM、DROP SYNONYM	類似するものが存在しない。
TANH()	SQL 関数	類似するものが存在しない。
TIME()	SQL 関数	CURRENT_TIME
TIMEDIFF()	SQL 関数	TIMEDIFF()。MySQL バージョン 4.1.1 で追加
TIMESTAMP()	SQL 関数	TIMESTAMP()。MySQL バージョン 4.1.1 で追加
DAYOFMONTH() と DAYOFYEAR() の引数としての TIMESTAMP()	SQL 関数	類似するものが存在しない。
TIMEZONE()	SQL 関数	類似するものが存在しない。
TRANSACTION()	現行トランザクションの ID を返す。	類似するものが存在しない。
TRANSLATE()	SQL 関数	REPLACE()。構文と実装が一致する。
TRIM()	SQL 関数	TRIM()。実装が少し異なる。
TRUNC()	SQL 関数	TRUNCATE()。構文と実装が少し異なる。
USE	mysql コマンドラインユーザインタフェースコマンド	USE
USER	SQL 関数	USER()。構文は同一だが、実装が異なる。かっこが必要とされる。
UTC_DIFF()	SQL 関数	UTC_DATE()。UTC_DIFF() の結果を計算する手段を提供する。
VALUE()	SQL 関数、COALESCE() のエイリアス	COALESCE()。構文と実装が一致する。
VARIANCE()	SQL 関数	類似するものが存在しない。

WEEKOFYEAR()	SQL 関数	WEEKOFYEAR()。MySQL バージョン 4.1.1 で追加

第9章 各国キャラクタセットと Unicode

MySQL バージョン 4.1 の追加機能の 1 つとして、キャラクタセットの処理が改善されたことがあげられます。この章では、以下について説明します。

- キャラクタセットと照合順序とは
- マルチレベルデフォルトシステム
- MySQL 4.1 での新しい構文
- 影響を受ける関数と演算
- 個別のキャラクタセットと照合順序の意味

ここでは、MySQL 4.1.1 での実装に基づき機能を説明します（MySQL 4.1.0 には、これらの機能の全部ではなく一部のみが存在します。また、実装が異なっている機能もあります）。

9.1. 一般のキャラクタセットおよび照合順序

キャラクタセットとは、シンボルとエンコードのセットです。照合順序とは、キャラクタセット内の文字を比較するためのルールを集めたものです。架空のキャラクタセットを例にして、キャラクタセットと照合順序の違いを見てみましょう。

次の 4 文字で構成されるアルファベットがあるとします : ‘A’、‘B’、‘a’、‘b’次のように、各文字に対して番号を指定します : ‘A’ = 0、‘B’ = 1、‘a’ = 2、‘c’ = 3文字 ‘A’ はシンボルであり、数字 0 は ‘A’ をエンコードしたものです。4 文字のすべてとそれぞれのエンコードを組み合わせたものをキャラクタセットと呼びます。

次に、文字列値 ‘A’ と ‘B’ を比較してみましょう。 最も簡単に比較するには、エンコードを確認します。‘A’ は 0、‘B’ は 1 です。0 は 1 よりも小さいので、‘A’ は ‘B’ よりも小さいと表現することができます。今ここで行ったのは、キャラクタセットに対する照合順序の適用です。照合順序はルールの集まりであり、上記の場合にルールは ``エンコードの比較" のみとなります。これは可能な照合順序のうちで最も単純なものであり、バイナリ照合順序と呼ばれています。

しかし、小文字と大文字を等しいと表現したい場合にはどうなるのでしょうか。その場合、少なくとも次の 2 つのルールが必要です。(1) 小文字の ‘a’ および ‘b’ が大文字の ‘A’ および ‘B’ と同じであると見なす。(2) その後にエンコードを比較する。これは大文字と小文字を区別しない照合順序と呼ばれ、バイナリ照合順序よりも少し複雑です。

実際は、大半のキャラクタセットに多数の文字が含まれています。‘A’ と ‘B’ だけではなく、アルファベットの全体から構成されています。ときには複数のアルファベットや、数千文字からなる東洋の書記体系に、多くの特殊記号や終止符が付属することもあります。また、実際には大半の照合順序に多くのルールがあります。大文字と小文字が区別されないだけではありません。アクセントが区別されない（``アクセント" は、ドイツ語での ‘Ö’ のように文字に追加されるマーク）、あるいは複数文字マッピング（ドイツ語照合順序のどちらかにおける ‘Ö’ = 'OE' のルールなど）などのルールがあります。

MySQL 4.1 では以下が可能です。

- 各種のキャラクタセットを使用して文字列を保存する。

- 各種の照合順序を使用して文字列を比較する。
- 同じサーバ、同じデータベース、あるいは同じテーブル内の異なったキャラクタセットまたは照合順序と文字列を結合する。
- 任意のレベルでキャラクタセットと照合順序を指定できるようにする。

MySQL 4.1 はこれらの点において、MySQL 4.0 よりもはるかに柔軟性があるばかりでなく、他の DBMS に大きく差をつけています。ただし、新機能を効率的に使用するには、利用可能なキャラクタセット、各デフォルトの変更方法、各種の列演算子による処理内容を知っておかなければなりません。

9.2. MySQL におけるキャラクタセットおよび照合順序

キャラクタセットには、少なくとも 1 つの照合順序が含まれており、複数の照合順序が含まれていることもあります。

たとえば、キャラクタセット latin1 (``ISO-8859-1 西ヨーロッパ言語") には以下の照合順序が含まれています。

照合順序	意味
latin1_bin	latin1 エンコードに基づくバイナリ
latin1_danish_ci	デンマーク語/ノルウェー語
latin1_german1_ci	ドイツ語 DIN-1
latin1_german2_ci	ドイツ語 DIN-2
latin1_swedish_ci	スウェーデン語/フィンランド語
latin1_general_ci	マルチリンガル

注意:

- 2 つの異なったキャラクタセットが同じ照合順序を共有することはできません。
- 各キャラクタセットには、デフォルト照合順序が 1 つ存在します。たとえば、latin1 のデフォルト照合順序は latin1_swedish_ci です。

照合順序名には次の規則が適用されます。関連するキャラクタセットの名前で始まる。通常は言語名が含まれており、_ci (大文字と小文字が区別されない) 、_cs (大文字と小文字が区別される) 、_bin (バイナリ) のいずれかで終わる。

9.3. デフォルトのキャラクタセットおよび照合順序の決定

サーバ、データベース、テーブル、接続の 4 段階で、キャラクタセットと照合順序のデフォルト設定が用意されています。以下の説明は複雑に見えるかもしれませんが、マルチレベルのデフォルト設定では自然かつ明確な結果を得られることが実際に判明しています。

9.3.1. サーバのキャラクタセットおよび照合順序

MySQL Server にはサーバキャラクタセットとサーバ照合順序があり、いずれもヌルにすることはできません。

MySQL では、サーバキャラクタセットとサーバ照合順序が次のように決定されます。

- サーバ起動時に有効なオプション設定に従います。

このレベルでは、決定は簡単です。サーバのキャラクタセットと照合順序は、mysqld の起動時に使用するオプションに依存します。--default-character-set=character_set_name をキャラクタセットに対して使用でき、--default-collation=collation_name を照合順序に対して追加することもできます。文字コードを指定しないのは、--default-character-set=latin1 を指定した場合と同じです。キャラクタセット (たとえば latin1) のみを指定して照合順序を指定しないのは、--default-charset=latin1 --collation=latin1_swedish_ci を指定した場合と同じです。これは latin1 のデフォルト照合順序が latin1_swedish_ci であるためです。 したがって、以下の 3 つのコマンドを実行すると、いずれも同じ結果になります。

```
shell> mysqld
shell> mysqld --default-character-set=latin1
shell> mysqld --default-character-set=latin1
       --default-collation=latin1_swedish_ci
```

設定を変更する手段の 1 つは再コンパイルです。ソースからのビルド時にデフォルトのサーバキャラクタセットと照合順序を変更するには、--with-character-set と --with-collation を configure の引数として使用してください。例:

```
shell> ./configure --with-character-set=latin1
```

または

```
shell> ./configure --with-character-set=latin1
       --with-collation=latin1_german1_ci
```

mysqld と configure では、キャラクタセットと照合順序の有効性がチェックされます。組み合わせが有効でない場合、各プログラムによってエラーメッセージが表示され、強制終了されます。

9.3.2. データベースのキャラクタセットおよび照合順序

各データベースにはデータベースキャラクタセットとデータベース照合順序があり、いずれもヌルにすることはできません。CREATE DATABASE および ALTER DATABASE コマンドには現在、データベースのキャラクタセットと照合順序を指定するためのオプション節があります。

```
CREATE DATABASE db_name
  [DEFAULT CHARACTER SET character_set_name [COLLATE collation_name]]

ALTER DATABASE db_name
  [DEFAULT CHARACTER SET character_set_name [COLLATE collation_name]]
```

例:

```
CREATE DATABASE db_name
  DEFAULT CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

MySQL では、データベースキャラクタセットとデータベース照合順序が次のように選択されます。

- CHARACTER SET X と COLLATE Y の両方を指定した場合は、キャラクタセット X と照合順序 Y。
- CHARACTER SET X を指定し、COLLATE を指定しなかった場合は、キャラクタセット X とそのデフォルト照合順序。
- その他の場合は、サーバキャラクタセットとサーバ照合順序。

MySQL の CREATE DATABASE ... DEFAULT CHARACTER SET ... 構文は標準 SQL CREATE SCHEMA ... CHARACTER SET ... 構文に類似しています。このため、キャラクタセットと照合順序が異なる複数のデータベースを同一の MySQL サーバ上に作成することができます。

テーブルのキャラクタセットと照合順序が CREATE TABLE ステートメントに指定されていない場合、データベースのキャラクタセットと照合順序はデフォルト値として使用されます。これらに他の用途はありません。

9.3.3. テーブルのキャラクタセットおよび照合順序

各テーブルにはテーブルキャラクタセットとテーブル照合順序があり、いずれもヌルにすることはできません。CREATE TABLE および ALTER TABLE ステートメントには現在、テーブルのキャラクタセットと照合順序を指定するためのオプション節があります。

```
CREATE TABLE table_name ( column_list )
  [DEFAULT CHARACTER SET character_set_name [COLLATE collation_name]]

ALTER TABLE table_name
  [DEFAULT CHARACTER SET character_set_name] [COLLATE collation_name]
```

例:

```
CREATE TABLE t1 ( ... ) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

MySQL では、テーブルキャラクタセットとテーブル照合順序が次のように選択されます。

- CHARACTER SET X と COLLATE Y の両方を指定した場合は、キャラクタセット X と照合順序 Y。
- CHARACTER SET X を指定し、COLLATE を指定しなかった場合は、キャラクタセット X とそのデフォルト照合順序。
- その他の場合は、データベースキャラクタセットとデータベース照合順序。

カラムのキャラクタセットと照合順序が個別のカラム定義に指定されていない場合、テーブルのキャラクタセットと照合順序はデフォルト値として使用されます。テーブルのキャラクタセットと照合順序は MySQL 拡張であり、同等の機能は標準 SQL に存在しません。

9.3.4. カラムのキャラクタセットおよび照合順序

各 ``文字" カラム (CHAR、VARCHAR または TEXT型) にはカラムキャラクタセットとカラム照合順序があり、いずれもヌルにすることはできません。カラム定義構文には現在、カラムキャラクタセットとカラム照合順序を指定するためのオプション節があります。

```
column_name {CHAR | VARCHAR | TEXT} (column_length)
  [CHARACTER SET character_set_name [COLLATE collation_name]]
```

例:

```
CREATE TABLE Table1
(
  column1 VARCHAR(5) CHARACTER SET latin1 COLLATE latin1_german1_ci
);
```

MySQL では、カラムキャラクタセットとカラム照合順序が次のように選択されます。

- CHARACTER SET X と COLLATE Y の両方を指定した場合は、キャラクタセット X と照合順序 Y。
- CHARACTER SET X を指定し、COLLATE を指定しなかった場合は、キャラクタセット X とそのデフォルト照合順序。
- その他の場合は、テーブルキャラクタセットとテーブル照合順序。

CHARACTER SET および COLLATE 節は標準 SQL です。

9.3.5. キャラクタセットと照合順序の割当の例

MySQL でデフォルトのキャラクタセットと照合順序の値がどのように決定されるかを、以下の例で示します。

例 1:テーブル + カラム定義

```
CREATE TABLE t1
(
 c1 CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci
) DEFAULT CHARACTER SET latin2 COLLATE latin2_bin;
```

ここでは、latin1 キャラクタセットと latin1_german1_ci 照合順序がカラムに指定されています。定義は明示的なので、直接的と言えます。なお、latin1 カラムの保存先が latin2 テーブルになっていることに問題はありません。

例 2:テーブル + カラム定義

```
CREATE TABLE t1
(
  c1 CHAR(10) CHARACTER SET latin1
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

ここでは、latin1 キャラクタセットとデフォルト照合順序がカラムに指定されています。自然な指定に見えますが、デフォルト照合順序はテーブルレベルから取り込まれません。latin1 のデフォルト照合順序は常に latin1_swedish_ci です。したがって、カラム c1 には latin1_danish_ci の照合順序ではなく latin1_swedish_ci の照合順序が設定されます。

例 3:テーブル + カラム定義

```
CREATE TABLE t1
(
  c1 CHAR(10)
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

ここでは、デフォルトキャラクタセットとデフォルト照合順序がカラムに指定されています。この場合に MySQL では、テーブルレベルまで検索してカラムのキャラクタセットと照合順序が決定されます。したがって、カラム c1 のキャラクタセットは latin1、照合順序は latin1_danish_ci となります。

例 4:データベース + テーブル + カラム定義

```
CREATE DATABASE d1 DEFAULT CHARACTER SET latin2 COLLATE latin2_czech_ci;
USE d1;
CREATE TABLE t1
(
  c1 CHAR(10)
);
```

キャラクタセットと照合順序を指定せずにカラムを作成します。テーブルレベルのキャラクタセットと照合順序も指定しません。この場合に MySQL では、データベースレベルまでさかのぼって処理が決定されます。(データベースの設定はテーブルの設定になり、そしてカラムの設定になります。) したがって、カラム c1 のキャラクタセットは latin2、照合順序は latin2_czech_ci となります。

9.3.6. 接続のキャラクタセットおよび照合順序

各接続には接続キャラクタセットと接続照合順序があり、いずれもヌルにすることはできません。実際には 2 つの接続キャラクタセットが存在するため、両者の区別が必要な場合は ``接続/リテラル" および ``接続/結果" の呼称を使い分けています。

``接続" とは、サーバへの接続時に作成されるものです。クライアントは接続を介し、SQL ステートメント (クエリなど) をサーバに送信します。サーバでは接続を介し、応答 (結果セットなど) をクライアントに返します。これによって、次のような疑問が生じます。(a) クライアントから送信される際にどのキャラクタセットでクエリが送られるのか。(b) サーバではクエリを受信した後にどのキャラクタセットに変換するのか。(c) サーバでは結果セットまたはエラーメッセージをクライアントに返送する前にどのキャラクタセットに変換するのか。これらは細かく調整することができますが、デフォルトを適用することもできます。デフォルトを適用する場合、このセクションをとばしてかまいません。

接続キャラクタセットに影響するステートメントが 2 つ存在します。

```
SET NAMES character_set_name
SET CHARACTER SET character_set_name
```

SET NAMES は、クライアントから送信される SQL ステートメントのキャラクタセットを示します。たとえば、SET NAMES cp1251 は 「このクライアントからの入力メッセージは今後、キャラクタセット cp1251 になります」 とサーバに通知します。サーバでは適宜、独自のキャラクタセットへと自由に変換することができます。

SET CHARACTER SET は、クライアントから送信される SQL ステートメントのキャラクタセットと、サーバからクライアントに返される結果セットのキャラクタセットを示します。そのため SET CHARACTER SET は、SET NAMES を含んでいるほか、たとえば SELECT ステートメントを使用する際にどのキャラクタセットでカラムに値が保持されるかを示します。

例:column1 が CHAR(5) CHARACTER SET latin2 として定義されているとします。SET CHARACTER SET が指定されていない場合、SELECT column1 FROM t に対しサーバは、キャラクタセット latin2 を使用して column1 の値をすべて返します。一方、SET CHARACTER SET latin1 が指定されている場合、サーバは送信前に latin2 の値を latin1 に変換します。そのような変換は低速であり、損失につながることもあります。

SET NAMES または SET CHARACTER SET の実行時には、``接続照合順序" も変更していることになります。ただし、接続照合順序は整合性の維持のみを目的として存在しています。通常、その値は重要ではありません。

mysql クライアントでは、起動するたびに SET NAMES を実行する必要はありません。--default-character-set-name オプション設定を mysql のコマンドラインか、オプションファイルに追加することができます。 たとえば、以下のオプション設定ファイルの設定では、mysql を実行するたびに接続キャラクタセットが指定されます。

```
[mysql]
default-character-set-name=character_set_name
```

9.3.7. 文字列リテラルのキャラクタセットおよび照合順序

各文字列リテラルにはキャラクタセットと照合順序があり、いずれもヌルにすることはできません。

文字列リテラルでは、オプションとしてキャラクタセットイントロデューサと COLLATE 節を指定することができます。

```
[_character_set_name]'string' [COLLATE collation_name]
```

例:

```
SELECT 'string';
SELECT _latin1'string';
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

単純なステートメント SELECT 'string' では、接続/リテラルキャラクタセットが使用されます。

_character_set_name は以前、イントロデューサと呼ばれていました。指定すると、``キャラクタセット X の文字列が後続する" ことがパーサに通知されます。 上記はユーザの混乱を招いていたため、ここで強調しておきますが、イントロデューサは変換の原因にはならず、文字列の値が変更されないことを示すにすぎません。標準的な 16 進リテラルおよび数値 16 進リテラルの表記 (x'literal' および 0xnnnn) の前と ? (用意されたステートメントをプログラミング言語インタフェースで使用する際のパラメータ代替) の前でも、イントロデューサは有効です。

例:

```
SELECT _latin1 x'AABBCC';
SELECT _latin1 0xAABBCC;
SELECT _latin1 ?;
```

MySQL では、リテラルのキャラクタセットおよび照合順序が次のように決定されます。

- _X と COLLATE Y の両方が指定された場合、リテラルキャラクタセットは X、リテラル照合順序は Y。
- _X は指定されており、COLLATE が指定されていない場合、リテラルキャラクタセットは X、リテラル照合順序は Xのデフォルト照合順序。
- その他の場合は、接続/リテラルのキャラクタセットおよび照合順序。

例:

- 文字列に latin1 キャラクタセットと latin1_german1_ci 照合順序が指定されている場合

```
SELECT _latin1'Müller' COLLATE latin1_german1_ci;
```

- 文字列に latin1 キャラクタセットとそのデフォルト照合順序 (latin1_swedish_ci) が指定されている場合

```
SELECT _latin1'Müller';
```

- 文字列に接続/リテラルのキャラクタセットおよび照合順序が指定されている場合

```
SELECT 'Müller';
```

キャラクタセットイントロデューサと COLLATE 節は、標準 SQL の指定に基づき実装されます。

9.3.8. SQL クエリの各部分における COLLATE 節

COLLATE 節では、比較に対するデフォルト照合順序が何であれ、無効にすることができます。SQL クエリのさまざまな個所で COLLATE を使用することができます。以下に例を示します。

- ORDER BY を指定した場合

```
SELECT k
FROM t1
ORDER BY k COLLATE latin1_german2_ci;
```

- AS を指定した場合

```
SELECT k COLLATE latin1_german2_ci AS k1
FROM t1
ORDER BY k1;
```

- GROUP BY を指定した場合

```
SELECT k
FROM t1
GROUP BY k COLLATE latin1_german2_ci;
```

- 集計関数を指定した場合

```
SELECT MAX(k COLLATE latin1_german2_ci)
FROM t1;
```

- DISTINCT を指定した場合

```
SELECT DISTINCT k COLLATE latin1_german2_ci
FROM t1;
```

- WHERE を指定した場合

```
SELECT *
```

```
FROM t1
WHERE _latin1 'Müller' COLLATE latin1_german2_ci = k;
```

- HAVING を指定した場合

```
SELECT k
FROM t1
GROUP BY k
HAVING k = _latin1 'Müller' COLLATE latin1_german2_ci;
```

9.3.9. COLLATE 節の優先順位

COLLATE 節は優先順位が高い (|| よりも上) ので、式

```
x || y COLLATE z
```

は以下と同じ意味になります。

```
x || (y COLLATE z)
```

9.3.10. BINARY 演算子

BINARY 演算子は、COLLATE 節の省略形です。たとえば、BINARY 'x' は 'x' COLLATE y と同じであり、この場合に y は該当するバイナリ照合順序の名称を表します。たとえば、カラム a のキャラクタセットが latin1 であると仮定すると、以下のクエリはどちらも同じ結果をもたらします。

```
SELECT * FROM t1 ORDER BY BINARY a;
SELECT * FROM t1 ORDER BY a COLLATE latin1_bin;
```

注意:どのキャラクタセットにもバイナリ照合順序があります。

9.3.11. 照合順序を決定するのが難しい特殊なケース

大多数のクエリでは、MySQL がどの照合順序により比較演算を行うかは明確になっています。たとえば以下の場合、照合順序が ``カラム x のカラム照合順序'' になることは明らかです。

```
SELECT x FROM T ORDER BY x;
SELECT x FROM T WHERE x = x;
SELECT DISTINCT x FROM T;
```

ただし、複数のオペランドが使用されていると、あいまいになることがあります。たとえば、以下のような場合です。

```
SELECT x FROM T WHERE x = 'Y';
```

x の照合順序と文字列リテラル 'Y' の照合順序のどちらがこのクエリで使用されるのでしょうか。

標準 SQL では、``強制可能'' ルールと呼ばれていた方法で上記の問題を解決しました。これについては、以下の発想が基本となっています: x と 'Y' の両方に照合順序があるので、どちらの照合順序を優先するかを判断しなければならない。複

雑な問題だが、これらのルールでほとんどの状況に対応できる。

- 明示的な COLLATE 節は優先順位が 4。
- 照合順序の異なる文字列 2 つの連結は優先順位が 3。
- カラムの照合順序は優先順位が 2。
- リテラルの照合順序は優先順位が 1。

これらのルールによって、あいまいさは次のように解消されます。

- 優先順位が最も高い照合順序を使用する。
- 双方の優先順位が一致する場合、照合順序が一致しないとエラーとなる。

例:

column1 = 'A'	column1 の照合順序を使用する。
column1 = 'A' COLLATE x	'A' の照合順序を使用する。
column1 COLLATE x = 'A' COLLATE y	エラー

9.3.12. 照合順序は適切なキャラクタセットに対応していること

各キャラクタセットには 1 つ以上の照合順序があり、各照合順序は 1 つのキャラクタセットに関連付けられていることを思い出してください。したがって、次のステートメントはエラーになります。latin2_bin 照合順序は latin1 キャラクタセットに対して有効でないからです。

```
mysql> SELECT _latin1 'x' COLLATE latin2_bin;
ERROR 1251: COLLATION 'latin2_bin' is not valid
for CHARACTER SET 'latin1'
```

9.3.13. 照合順序がもたらす結果の例

テーブル T のカラム X に以下の latin1 カラムの値が設定されているとします。

```
Muffler
Müller
MX Systems
MySQL
```

さらに、以下のステートメントを使用してカラムの値を取得したとします。

```
SELECT X FROM T ORDER BY X COLLATE collation_name;
```

異なった照合順序に対する値の順序は以下のようになります。

latin1_swedish_ci	latin1_german1_ci	latin1_german2_ci

Muffler	Muffler	Müller
MX Systems	Müller	Muffler
Müller	MX Systems	MX Systems
MySQL	MySQL	MySQL

この表は、複数の異なった照合順序を ORDER BY 節で使用した結果の一例です。この例では、2 個の点が上に付いている U が問題の原因です。この文字をドイツではウムラウトと呼んでいますが、私たちは分音記号付きの U と呼んでいます。

最初のカラムには、スウェーデン語/フィンランド語の照合ルールを使用したSELECT の結果が示されています。この照合ルールによると、分音記号付きの U は Y と一致します。

2 番目のカラムには、ドイツ語の DIN-1 ルールを使用した SELECT の結果が示されています。このルールによると、分音記号付きの U は U と一致します。

3 番目のカラムには、ドイツ語の DIN-2 ルールを使用したSELECT の結果が示されています。このルールによると、分音記号付きの U は UE と一致します。

3 つの異なった照合順序が、3 つの異なった結果をもたらしています。これこそ正に MySQL で対処すべき状況です。適切な照合順序を使用すると、任意のソート順序を選択できます。

9.4. キャラクタセットのサポートによる影響を受ける演算

このセクションでは、キャラクタセット情報を計算に入れる演算について説明します。

9.4.1. 結果文字列

MySQL には、文字列を返す多数の演算子と関数があります。 このセクションでは、そのような文字列のキャラクタセットと照合順序について解説しています。

文字列の入力を取得して文字列の結果を出力として返す単純な関数では、出力のキャラクタセットおよび照合順序は主要な入力のキャラクタセットおよび照合順序と同じです。たとえば UPPER(X) は、キャラクタセットおよび照合が X と一致する文字列を返します。 同じことは以下についても当てはまります。 INSTR(), LCASE(), LOWER(), LTRIM(), MID(), REPEAT(), REPLACE(), REVERSE(), RIGHT(), RPAD(), RTRIM(), SOUNDEX(), SUBSTRING(), TRIM(), UCASE(), UPPER()。 (注意: REPLACE() 関数は他のすべての関数とは異なり、文字列入力の照合順序を無視し、大文字と小文字が区別されない比較を毎回実行します。)

複数の文字列入力を組み合わせて単一の文字列出力を返す操作には、SQL-99 の ``集約ルール" が適用されます (以下を参照) 。

- 明示的な COLLATE X が存在する場合は X を使用する。
- 明示的な COLLATE X と COLLATE Y が存在する場合はエラーになる。
- 上記以外の場合ですべての照合順序が X であるときは X を使用する。
- その他の場合、結果に照合順序は含まれない。

たとえば、CASE ... WHEN a THEN b WHEN b THEN c COLLATE X END と指定されている場合、照合順序は X になります。同じことは以下についても当てはまります。 CONCAT(), GREATEST(), IF(), LEAST(), CASE, UNION, ||, ELT()。

文字データに変換する操作のため、結果文字列のキャラクタセットと照合順序は 接続/リテラルキャラクタセットに含まれており、接続/リテラル照合順序を持っています。 このことは以下についても当てはまります。 CHAR(), CAST(), CONV(), FORMAT()。 HEX(), SPACE()。

9.4.2. CONVERT()

CONVERT() を使用すると、異なるキャラクタセット間でデータを変換することができます。構文は以下のとおりです。

```
CONVERT(expr USING transcoding_name)
```

MySQL では、トランスコーディング名は対応するキャラクタセット名と同じです。

例:

```
SELECT CONVERT(_latin1'Müller' USING utf8);
INSERT INTO utf8table (utf8column)
  SELECT CONVERT(latin1field USING utf8) FROM latin1table;
```

CONVERT(... USING ...) は、SQL-99 の仕様に基づき実装されています。

9.4.3. CAST()

CAST() を使用し、文字列を別のキャラクタセットに変換することもできます。新しい書式は以下のとおりです。

```
CAST ( character_string AS character_data_type
   CHARACTER SET character_set_name )
```

例:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8);
```

COLLATE 節を CAST() の内部で使用することはできませんが、外部では使用することができます。したがって、CAST(... COLLATE ...) は無効ですが、CAST(...) COLLATE ... は有効です。

例:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8) COLLATE utf8_bin;
```

CAST() を CHARACTER SET の指定なしで使用した場合、キャラクタセットと照合順序は接続/リテラルキャラクタセットとそのデフォルト照合順序になります。CAST() を CHARACTER SET X の指定ありで使用した場合、キャラクタセットは X、照合順序は X のデフォルト照合順序になります。

9.4.4. SHOW CHARACTER SET

SHOW CHARACTER SET コマンドを実行すると、使用可能なキャラクタセットがすべて表示されます。 オプション節 LIKE を指定すると、条件にマッチするキャラクタセットを表示することができます。

例:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+---------+-----------------------------+-------------------+--------+
| Charset | Description                 | Default collation | Maxlen |
+---------+-----------------------------+-------------------+--------+
| latin1  | ISO 8859-1 West European    | latin1_swedish_ci |      1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci |      1 |
| latin5  | ISO 8859-9 Turkish          | latin5_turkish_ci |      1 |
| latin7  | ISO 8859-13 Baltic          | latin7_general_ci |      1 |
+---------+-----------------------------+-------------------+--------+
4 rows in set (0.00 sec)
```

上の一覧に関する注意:

- Maxlen カラムは、1 文字を格納するための最大バイト数を示します。

9.4.5. SHOW COLLATION

SHOW COLLATION コマンドを実行すると、使用可能なキャラクタセットがすべて表示されます。 オプション節 LIKE を指定すると、条件にマッチする照合順序名が表示されます。

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-------------------+---------+----+---------+----------+---------+
| Collation         | Charset | Id | Default | Compiled | Sortlen |
+-------------------+---------+----+---------+----------+---------+
| latin1_german1_ci | latin1  |  5 |         |          |       0 |
| latin1_swedish_ci | latin1  |  8 | Yes     | Yes      |       0 |
| latin1_danish_ci  | latin1  | 15 |         |          |       0 |
| latin1_german2_ci | latin1  | 31 |         | Yes      |       2 |
| latin1_bin        | latin1  | 47 |         | Yes      |       0 |
| latin1_general_ci | latin1  | 48 |         |          |       0 |
| latin1_general_cs | latin1  | 49 |         |          |       0 |
+-------------------+---------+----+---------+----------+---------+
7 rows in set (0.00 sec)
```

Default カラムは、照合順序がそのキャラクタセットのデフォルトであるかどうかを示します。 Compiled は、キャラクタセットがサーバにコンパイルされているかどうかを示します。 Sortlen は、キャラクタセットでソート文字列を表現するために必要なメモリの容量に関連します。

9.4.6. SHOW CREATE DATABASE

以下のクエリは、指定されたデータベースを作成する CREATE DATABASE ステートメントを示します。実行結果には、データベースオプションがすべて含まれています。DEFAULT CHARACTER SET と COLLATE がサポートされています。データベースオプションがすべて保存されている単一のテキストファイルがデータベースディレクトリにあります。

```
mysql> SHOW CREATE DATABASE a;
+----------+-----------------------------------------------------------------+
| Database | Create Database
|
+----------+-----------------------------------------------------------------+
| a        | CREATE DATABASE `a` /*!40100 DEFAULT CHARACTER SET macce
```

```
COLLATE macce_ci_ai */ |
+----------+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

9.4.7. SHOW FULL COLUMNS

SHOW COLUMNS ステートメントを SHOW FULL COLUMNS として呼び出したときは、テーブルのカラムの照合順序を示します。 データ型が CHAR、VARCHAR、TEXT のいずれかであるカラムは、照合順序が NULL になっていません。数値型と文字型以外のカラムは、照合順序が NULL になっています。例:

```
mysql> SHOW FULL COLUMNS FROM a;
+-------+---------+-------------------+------+-----+---------+-------+
| Field | Type    | Collation         | Null | Key | Default | Extra |
+-------+---------+-------------------+------+-----+---------+-------+
| a     | char(1) | latin1_swedish_ci | YES  |     | NULL    |       |
| b     | int(11) | NULL              | YES  |     | NULL    |       |
+-------+---------+-------------------+------+-----+---------+-------+
2 rows in set (0.02 sec)
```

キャラクタセットは表示対象に含まれていません。

9.5. Unicode のサポート

MySQL バージョン 4.1 以降、Unicode データを保存するために次の新しいキャラクタセットが用意されています: ucs2 (UCS-2 Unicode キャラクタセット) および utf8 (Unicode キャラクタセットの UTF-8 エンコード)。

- UCS-2 (Unicode のバイナリ表現) では、各文字は 2 バイトの Unicode と最上位のバイトで最初に表現されます。例 : "ラテン文字の大文字 A" はコード 0x0041 ですが、2 バイトシーケンス 0x00 0x41 として保存されています。"キリル文字の小文字 YERU" (Unicode 0x044B) は2 バイトシーケンス 0x04 0x4B として保存されています。Unicode 文字および対応するコードについては、Unicode Home Page を参照してください。

  一時的な制限:UCS-2 はクライアントキャラクタセットとしては (まだ) 使用できません。つまり、SET NAMES ucs2 は有効ではないということです。

- UTF8 キャラクタセット (Unicode 表現の変換) は、Unicode データを保存する別の方法であり、RFC2279 に基づき実装されています。さまざまな Unicode 文字を長さの異なるバイトシーケンスに適合させることが、UTF8 キャラクタセットの概念です。

  - 基本的なラテン文字、数字、句読点は 1 バイトを使用します。
  - ヨーロッパおよび中東のスクリプト文字の多くは、2 バイトシーケンスに適合します。拡張ラテン文字 (チルダ、長音、鋭アクセント、抑音アクセントその他のアクセント付き)、キリル文字、ギリシア文字、アルメニア文字、ヘブライ文字、アラビア文字、シリア文字その他。
  - 韓国語、中国語、日本語の表意文字は、3 バイトシーケンスを使用します。
  - 現時点では、MySQL UTF8 サポートに 4 バイトシーケンスは含まれていません。

  ヒント:スペースを UTF8 で保存するには、CHAR ではなく VARCHAR を使用してください。 そのようにしないと、MySQL では CHAR(10) CHARACTER SET utf8 カラムに対して 30 バイトを確保しなければなりません。これは、使用

可能な最大長が 30 バイトであるためです。

9.6. メタデータ用の UTF8

メタデータとは、データについてのデータです。データベースの内容となっているデータではなく、データベースについて説明するデータがメタデータです。したがって、カラム名、データベース名、ユーザ名、バージョン名のほか、SHOW を実行して表示される文字列の多くがメタデータに該当します。

すべてのメタデータはキャラクタセットが一致している必要があります (そうなっていない場合、SHOW は正しく実行されるとは限りません。同じカラムに含まれる個々のレコードのキャラクタセットが一致しない可能性があるからです)。一方、すべての言語によるすべての文字がメタデータに含まれている必要があります (そうなっていない場合、ユーザがカラムやテーブルの名称を母国語で設定できない可能性があります)。上記 2 つの目的を考慮するため、MySQL ではメタデータが Unicode キャラクタセット (UTF8) で保存されます。これによって不具合が発生しないのは、アクセント付き文字を使用しない場合です。使用する場合、メタデータのキャラクタセットが UTF8 であることを認識する必要があります。

つまり、USER() (およびそのシノニム SESSION_USER() と SYSTEM_USER())、CURRENT_USER()、VERSION() の各関数では、UTF8 キャラクタセットがデフォルトで使用されます。

ただし、カラムのヘッダと DESCRIBE 関数の実行結果がデフォルトで UTF8 キャラクタセットになるということではありません (SELECT column1 FROM t と指定すると、名称 column1 自体がクライアントのキャラクタセットによりサーバからクライアントに返されます。このキャラクタセットは、SET NAMES ステートメントで決定されたものです)。

サーバからメタデータの結果が UTF8 以外のキャラクタセットで返されるようにするには、(see 項9.3.6.「接続のキャラクタセットおよび照合順序」) の変換を SET CHARACTER SET によってサーバに実行させるか、変換がクライアントで実行されるように設定します。クライアントに変換を実行させる方が常に効率的ですが、このオプションは MySQL 4.x の製品サイクル後期まで使用することができません。

たとえば、USER() 関数を比較または割当のために単一のステートメントで使用しているとします。 MySQL には自動変換機能が用意されています。

```
SELECT * FROM Table1 WHERE USER() = latin1_column;
```

この機能が有効なのは、latin1_column の内容が UTF8 へと自動的に変換されてから比較が行われるからです。

```
INSERT INTO Table1 (latin1_column) SELECT USER();
```

この機能が有効なのは、USER() の内容が latin1 へと自動的に変換されてから割当が行われるからです。 自動変換機能は完全には実装されていませんが、将来のバージョンでは適切に動作する予定です。

自動変換機能は SQL 標準に含まれていません。ただし、どのキャラクタセットも (サポートされている文字に関して) Unicode の ``サブセット" であることが SQL 標準の文書に記載されています。「スーパーセットに適用されるものはサブセットにも適用される」という有名な原則があるので、Unicode の照合順序は Unicode 以外の文字列との比較にも適用できると考えられます。

バージョン 4.1.1 注意:この時点から、errmsg.txt ファイルはすべて UTF8 になります。クライアントのキャラクタセットへの変換は、メタデータに関しては自動的に行われます。結果セットを返す場合のデフォルトの動作を変更することができます。

9.7. 他の DBMS との互換性

SAP DB の互換性に関し、以下の 2 つのステートメントは同じです。

```
CREATE TABLE t1 (f1 CHAR(n) UNICODE);
CREATE TABLE t1 (f1 CHAR(n) CHARACTER SET ucs2);
```

9.8. 新規キャラクタセット設定ファイルの形式

MySQL 4.1 の場合、キャラクタセットの設定は XML ファイルに保存されます。キャラクタセットごとに 1 ファイルが使用されます (以前のバージョンでは、キャラクタセット設定情報は .conf ファイルに保存されました) 。

9.9. 各国キャラクタセット

MySQL-4.x とそれ以前のバージョンでは、NCHAR と CHAR は同義語でした。ANSI では、事前定義キャラクタセットが CHAR カラムで使用されるように指定する方法の 1 つとして NCHAR または NATIONAL CHAR を定義しています。MySQL では、utf8 が事前定義キャラクタセットとして使用されます。たとえば、以下のカラム型宣言

```
CHAR(10) CHARACTER SET utf8
NATIONAL CHARACTER(10)
NCHAR(10)
```

は、以下のカラム型宣言と等価です。

```
VARCHAR(10) CHARACTER SET utf8
NATIONAL VARCHAR(10)
NCHAR VARCHAR(10)
NATIONAL CHARACTER VARYING(10)
NATIONAL CHAR VARYING(10)
```

N'literal' を使用して、各国キャラクタセットの文字列を作成することができます。

以下の 2 つのステートメントは等価です。

```
SELECT N'some text';
SELECT _utf8'some text';
```

9.10. MySQL 4.0 からのアップグレード

ここでは、旧バージョンの MySQL からのアップグレードについて説明します。MySQL 4.1 では、MySQL 4.0 との上位互換性がほぼ完全に確保されており、その理由は単に機能の大半が新しい点にあります。したがって、コンフリクトを引き起こす要素は旧バージョンに存在しません。ただし、相違点と注意点がいくつかあります。

最重要:``MySQL 4.0 キャラクタセット" には、``MySQL 4.1 キャラクタセット" と ``MySQL 4.1 照合順序" の両方の特性が含まれています。しかし、これを覚えておく必要はありません。今後は、同じ複合オブジェクト内でキャラクタセットまたは照合順序特性を並存させない予定です。

MySQL には各国キャラクタセットに関する特別な処理が用意されています。 4.1. NCHAR は CHAR と同じではなく、N'...' リテラルは '...' リテラルと同じではありません。

最後に、キャラクタセットと照合順序に関する情報を格納するための特別なファイル形式があります。新規設定ファイルを含む /share/mysql/charsets/ ディレクトリの再インストールが済んでいることを確認してください。

MySQL 4.0 で作成したデータを使用し、mysqld を 4.1.x ディストリビューションから起動するには、同じキャラクタセットと照合順序でサーバを起動する必要があります。この場合、データのインデックスを再作成する必要はありません。

上記を実行する方法は以下の 2 とおりです。

```
shell> ./configure --with-character-set=... --with-collation=...
shell> ./mysqld --default-character-set=... --default-collation=...
```

たとえば、mysql を MySQL 4.0 danish キャラクタセットと共に使用する場合、latin1 キャラクタセットと latin1_danish_ci 照合順序を使用する必要があります。

```
shell> ./configure --with-character-set=latin1
       --with-collation=latin1_danish_ci
shell> ./mysqld --default-character-set=latin1
       --default-collation=latin1_danish_ci
```

次のセクションに示された表を使用し、古い 4.0 キャラクタセット名とそれに対応する 4.1 キャラクタセット/照合順序のペアを検索します。

9.10.1. 4.0 キャラクタセットおよび対応する 4.1 キャラクタセット/照合順序のペア

ID	4.0 キャラクタセット	4.1 キャラクタセット	4.1 照合順序
1	big5	big5	big5_chinese_ci
2	czech	latin2	latin2_czech_ci
3	dec8	dec8	dec8_swedish_ci
4	dos	cp850	cp850_general_ci
5	german1	latin1	latin1_german1_ci
6	hp8	hp8	hp8_english_ci
7	koi8_ru	koi8r	koi8r_general_ci
8	latin1	latin1	latin1_swedish_ci
9	latin2	latin2	latin2_general_ci
10	swe7	swe7	swe7_swedish_ci
11	usa7	ascii	ascii_general_ci
12	ujis	ujis	ujis_japanese_ci
13	sjis	sjis	sjis_japanese_ci
14	cp1251	cp1251	cp1251_bulgarian_ci
15	danish	latin1	latin1_danish_ci
16	hebrew	hebrew	hebrew_general_ci
17	win1251	(removed)	(removed)
18	tis620	tis620	tis620_thai_ci

19	euc_kr	euckr	euckr_korean_ci
20	estonia	latin7	latin7_estonian_ci
21	hungarian	latin2	latin2_hungarian_ci
22	koi8_ukr	koi8u	koi8u_ukrainian_ci
23	win1251ukr	cp1251	cp1251_ukrainian_ci
24	gb2312	gb2312	gb2312_chinese_ci
25	greek	greek	greek_general_ci
26	win1250	cp1250	cp1250_general_ci
27	croat	latin2	latin2_croatian_ci
28	gbk	gbk	gbk_chinese_ci
29	cp1257	cp1257	cp1257_lithuanian_ci
30	latin5	latin5	latin5_turkish_ci
31	latin1_de	latin1	latin1_german2_ci

9.11. MySQL でサポートされるキャラクタセットと照合順序

MySQL でサポートされるキャラクタセットと照合順序の注釈付き一覧を以下に示します。オプションとインストール時の設定が異なるため、一覧に含まれている項目の一部が表示されないサーバもあります。また、新しいキャラクタセットまたは照合順序の定義は簡単なので、一覧に含まれていない項目が表示されるサーバもあります。

MySQL では、30 を超えるキャラクタセットに対して 70 を超える照合順序がサポートされています。

```
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| cp850    | DOS West European           | cp850_general_ci    |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
| latin1   | ISO 8859-1 West European    | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |
| swe7     | 7bit Swedish                | swe7_swedish_ci     |      1 |
| ascii    | US ASCII                    | ascii_general_ci    |      1 |
| ujis     | EUC-JP Japanese             | ujis_japanese_ci    |      3 |
| sjis     | Shift-JIS Japanese          | sjis_japanese_ci    |      2 |
| cp1251   | Windows Cyrillic            | cp1251_bulgarian_ci |      1 |
| hebrew   | ISO 8859-8 Hebrew           | hebrew_general_ci   |      1 |
| tis620   | TIS620 Thai                 | tis620_thai_ci      |      1 |
| euckr    | EUC-KR Korean               | euckr_korean_ci     |      2 |
| koi8u    | KOI8-U Ukrainian            | koi8u_general_ci    |      1 |
| gb2312   | GB2312 Simplified Chinese   | gb2312_chinese_ci   |      2 |
| greek    | ISO 8859-7 Greek            | greek_general_ci    |      1 |
| cp1250   | Windows Central European    | cp1250_general_ci   |      1 |
| gbk      | GBK Simplified Chinese      | gbk_chinese_ci      |      2 |
| latin5   | ISO 8859-9 Turkish          | latin5_turkish_ci   |      1 |
```

```
| armscii8 | ARMSCII-8 Armenian          | armscii8_general_ci |      1 |
| utf8     | UTF-8 Unicode               | utf8_general_ci     |      3 |
| ucs2     | UCS-2 Unicode               | ucs2_general_ci     |      2 |
| cp866    | DOS Russian                 | cp866_general_ci    |      1 |
| keybcs2  | DOS Kamenicky Czech-Slovak  | keybcs2_general_ci  |      1 |
| macce    | Mac Central European        | macce_general_ci    |      1 |
| macroman | Mac West European           | macroman_general_ci |      1 |
| cp852    | DOS Central European        | cp852_general_ci    |      1 |
| latin7   | ISO 8859-13 Baltic          | latin7_general_ci   |      1 |
| cp1256   | Windows Arabic              | cp1256_general_ci   |      1 |
| cp1257   | Windows Baltic              | cp1257_general_ci   |      1 |
| binary   | Binary pseudo charset       | binary              |      1 |
+----------+-----------------------------+---------------------+--------+
33 rows in set (0.01 sec)
```

注意:すべてのキャラクタセットにバイナリ照合順序が存在します。ただし、以降の説明ではバイナリ照合順序について取り上げていないこともあります。

9.11.1. Unicode キャラクタセット

Unicode キャラクタセットが 2 つ存在することは言うまでもありません。これらのキャラクタセットを使用し、約 650 の言語でテキストを保存することができます。2 つの新しいキャラクタセットに対応する多くの照合順序は追加されていませんが、間もなく追加される予定です。現在、大文字と小文字およびアクセントが区別されないデフォルトの照合順序のほか、バイナリ照合順序があります。

```
+---------+-----------------+--------------------+--------+
| Charset | Description     | Default collation | Maxlen |
+---------+-----------------+--------------------+--------+
| utf8    | UTF-8 Unicode   | utf8_general_ci   |      3 |
| ucs2    | UCS-2 Unicode   | ucs2_general_ci   |      2 |
+---------+-----------------+--------------------+--------+
```

9.11.2. プラットフォーム固有のキャラクタセット

```
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
+----------+-----------------------------+---------------------+--------+
```

9.11.3. 南ヨーロッパおよび中東のキャラクタセット

```
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| armscii8 | ARMSCII-8 Armenian          | armscii8_general_ci |      1 |
| cp1256   | Windows Arabic              | cp1256_general_ci   |      1 |
| hebrew   | ISO 8859-8 Hebrew           | hebrew_general_ci   |      1 |
| greek    | ISO 8859-7 Greek            | greek_general_ci    |      1 |
| latin5   | ISO 8859-9 Turkish          | latin5_turkish_ci   |      1 |
| geostd8  | Georgian                    | geostd8_general_ci  |      1 |
+----------+-----------------------------+---------------------+--------+
```

9.11.4. アジアのキャラクタセット

サポートされているアジアのキャラクタセットには、中国語、日本語、韓国語、タイ語が含まれています。これらは複雑な場合があります。たとえば、中国語のキャラクタセットは数千種類の文字に対応していなければなりません。

```
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| gb2312   | GB2312 Simplified Chinese   | gb2312_chinese_ci   |      2 |
| gbk      | GBK Simplified Chinese      | gbk_chinese_ci      |      2 |
| euckr    | EUC-KR Korean               | euckr_korean_ci     |      2 |
| ujis     | EUC-JP Japanese             | ujis_japanese_ci    |      3 |
| sjis     | Shift-JIS Japanese          | sjis_japanese_ci    |      2 |
| tis620   | TIS620 Thai                 | tis620_thai_ci      |      1 |
+----------+-----------------------------+---------------------+--------+
```

9.11.4.1. cp932キャラクターセット

なぜcp932 キャラクターセットが必要か

MySQLにおけるsjisキャラクターセットはIANAによって 定義されたShift_JISキャラクターセットに相当し、 JIS X0201、及びJIS X0208文字がサポートされます(http://www.iana.org/assignments/character-setsを参照のこと)。

しかしながら、記述用語として一般的に使われている“シフトJIS”の意味は 非常にあいまいで、しばしば各ベンダーが独自にShift_JISを 拡張したものまで含まれることがあります。

例えば、日本語Windows環境で使用される「シフトJIS」は MicrosoftによるShift_JISの拡張で、正式な名称は Microsoft Windows Codepage : 932もしくは cp932といいます。 cp932では Shift_JISでサポートされる文字に加え、NEC特殊文字、 NEC選定IBM拡張文字、IBM拡張文字といった各種拡張文字がサポートされます。

MySQL 4.1以降、多くの日本語ユーザーがこれら拡張文字等の使用にあたって 問題に直面してきましたが、これらの問題は以下の要因によって生じていました:

- MySQLが自動的にキャラクターセットの変換を行う。
- キャラクターセットの変換はUnicode(ucs2)を介して行われる。
- sjisキャラクターセットはこれら拡張文字の変換をサポートしていない。
- いわゆる「シフトJIS」と呼ばれるキャラクターセットからUnicodeへの変換には 複数の変換ルールが存在し、いくつかの文字は変換ルールによって異なるUnicode文字に 変換される。MySQLではこれらの変換ルールのうち、一つだけしかサポートされていない (詳細は後述する)。

MySQLのcp932キャラクタセットはこれらの問題を解決するよう デザインされており、MySQL 4.1.12、及び5.0.3以降で使用することができます。

MySQL 4.1より前では、sjis キャラクターセットの使用にあたってどのような「シフトJIS」文字を 使用しても問題はありませんでしたが、4.1以降では、MySQLがキャラクターセットの 変換をサポートするようになった為、異なる変換ルール

を持つIANAの Shift_JISとcp932を二種類の キャラクターセットとして区別することが重要となります。

cp932はsjisとどう異なるか

cp932キャラクターセットは以下の点で sjisと異なります:

- cp932ではNEC特殊文字、NEC選定IBM拡張文字、IBM拡張文字がサポートされる。
- いくつかのcp932文字については、二つの異なる コードポイントから同一のUnicodeコードポイントに変換される。 よってこれらの文字をUnicodeからcp932に戻す際には 何れか一つのコードポイントが選択されなくてはならない。 この「ラウンドトリップ変換」については、Microsoftによって 推奨されるルールが使用されている (http://support.microsoft.com/kb/170559/EN-US/を参照のこと)。

  この変換ルールは以下の通り:

  - 当該文字がJIS X 0208文字とNEC特殊文字の両方に存在する場合には、 JIS X 0208のコードポイントを使用する。
  - 当該文字がNEC特殊文字とIBM拡張文字の両方に存在する場合には、 NEC特殊文字のコードポイントを使用する。
  - 当該文字がNEC選定IBM拡張文字とIBM拡張文字の両方に存在する場合には、 IBM拡張文字のコードポイントを使用する。

  cp932文字のUnicodeコードポイントに関する情報は、 http://www.microsoft.com/globaldev/reference/dbcs/932.htm にある表を参照のこと。cp932の表に記載されている文字のうち、 下に四桁の数字が表示されているものについては、その数字は対応するUnicode (ucs2)コードポイントを表す。下線付きの二桁の値については これら二桁の値で始まる一連のcp932文字があることを表し、 これらの値をクリックすることで、その二桁の値で始まる cp932文字、及びその Unicodeコードポイントが表示される。

  更に興味のある方は以下のリンクを参照のこと。それぞれ、各文字の対応する Unicodeコードポイントを表示する。

  - NEC特殊文字:

    ```
    http://www.microsoft.com/globaldev/reference/dbcs/932/932_87.htm
    ```

  - NEC選定IBM拡張文字:

    ```
    http://www.microsoft.com/globaldev/reference/dbcs/932/932_ED.htm
    http://www.microsoft.com/globaldev/reference/dbcs/932/932_EE.htm
    ```

  - IBM拡張文字:

    ```
    http://www.microsoft.com/globaldev/reference/dbcs/932/932_FA.htm
    http://www.microsoft.com/globaldev/reference/dbcs/932/932_FB.htm
    http://www.microsoft.com/globaldev/reference/dbcs/932/932_FC.htm
    ```

- 5.0.3以降では、eucjpmsキャラクターセットと組み合わせて 使用することで、 cp932でユーザー定義文字の変換がサポートされ、 sjis/ujis間での変換問題にも対応している。 詳細はhttp://www.opengroup.or.jp/jvc/cde/sjis-euc-e.htmlを参照のこと。
- いくつかの文字については、sjisとcp932で ucs2との変換ルールが異なる。以下の表ではその違いを説明している。

  ucs2への変換

sjis/cp932 のコードポイント	sjis → ucs2変換	cp932 → ucs2変換
5C	005C	005C
7E	007E	007E
815C	2015	2015
815F	005C	FF3C
8160	301C	FF5E
8161	2016	2225
817C	2212	FF0D
8191	00A2	FFE0
8192	00A3	FFE1
81CA	00AC	FFE2

ucs2からの変換:

ucs2 のコードポイント	ucs2 → sjis変換	ucs2 → cp932変換
005C	815F	5C
007E	7E	7E
00A2	8191	3F
00A3	8192	3F
00AC	81CA	3F
2015	815C	815C
2016	8161	3F
2212	817C	3F
2225	3F	8161
301C	8160	3F
FF0D	3F	817C
FF3C	3F	815F
FF5E	3F	8160
FFE0	3F	8191
FFE1	3F	8192
FFE2	3F	81CA

9.11.5. バルト語キャラクタセット

バルト語キャラクタセットには、エストニア語、ラトビア語、リトアニア語が含まれています。現在サポートされている

バルト語キャラクタセットは以下の 2 つです:

- latin7（ISO 8859-13 バルト語):

```
+--------------------+----------+----+---------+----------+---------+
| Collation          | Charset  | Id | Default | Compiled | Sortlen |
+--------------------+----------+----+---------+----------+---------+
| latin7_estonian_cs | latin7   | 20 |         |          |       0 |
| latin7_general_ci  | latin7   | 41 | Yes     |          |       0 |
| latin7_general_cs  | latin7   | 42 |         |          |       0 |
| latin7_bin         | latin7   | 79 |         |          |       0 |
+--------------------+----------+----+---------+----------+---------+
```

- cp1257（Windows バルト語）:

```
+----------------------+----------+----+---------+----------+---------+
| Collation            | Charset  | Id | Default | Compiled | Sortlen |
+----------------------+----------+----+---------+----------+---------+
| cp1257_lithuanian_ci | cp1257   | 29 |         |          |       0 |
| cp1257_bin           | cp1257   | 58 |         |          |       0 |
| cp1257_general_ci    | cp1257   | 59 | Yes     |          |       0 |
+----------------------+----------+----+---------+----------+---------+
```

9.11.6. キリル語キャラクタセット

ベラルーシ語、ブルガリア語、ロシア語、ウクライナ語と共に使用するキリル語のキャラクタセットおよび照合順序を以下に示します。

- cp1251（Windows キリル語）

```
+--------------------+----------+----+---------+----------+---------+
| Collation          | Charset  | Id | Default | Compiled | Sortlen |
+--------------------+----------+----+---------+----------+---------+
| cp1251_bulgarian_ci | cp1251   | 14 |         |          |       0 |
| cp1251_ukrainian_ci | cp1251   | 23 |         |          |       0 |
| cp1251_bin         | cp1251   | 50 |         |          |       0 |
| cp1251_general_ci  | cp1251   | 51 | Yes     |          |       0 |
| cp1251_general_cs  | cp1251   | 52 |         |          |       0 |
+--------------------+----------+----+---------+----------+---------+
```

- cp866（DOS ロシア語）:

```
+--------------------+----------+----+---------+----------+---------+
| Collation          | Charset  | Id | Default | Compiled | Sortlen |
+--------------------+----------+----+---------+----------+---------+
| cp866_general_ci   | cp866    | 36 | Yes     |          |       0 |
| cp866_bin          | cp866    | 68 |         |          |       0 |
+--------------------+----------+----+---------+----------+---------+
```

- koi8r（KOI8-R Relcom Russian、Unix 上で主として ロシア語に使用）:

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

```
| Collation          | Charset  | Id | Default | Compiled | Sortlen |
+--------------------+----------+----+---------+----------+---------+
| koi8r_general_ci   | koi8r    |  7 | Yes     |          |       0 |
| koi8r_bin          | koi8r    | 74 |         |          |       0 |
+--------------------+----------+----+---------+----------+---------+
```

- koi8u（KOI8-R Ukrainian、Unix 上で主としてウクライナ語に使用）:

```
+--------------------+----------+----+---------+----------+---------+
| Collation          | Charset  | Id | Default | Compiled | Sortlen |
+--------------------+----------+----+---------+----------+---------+
| koi8u_general_ci   | koi8u    | 22 | Yes     |          |       0 |
| koi8u_bin          | koi8u    | 75 |         |          |       0 |
+--------------------+----------+----+---------+----------+---------+
```

9.11.7. 中央ヨーロッパのキャラクタセット

チェコ、スロバキア、ハンガリー、ルーマニア、スロベニア、クロアチア、ポーランドで使用されるキャラクタセットも一部サポートされています。

- cp1250（Windows 中央ヨーロッパ）:

```
+--------------------+----------+----+---------+----------+---------+
| Collation          | Charset  | Id | Default | Compiled | Sortlen |
+--------------------+----------+----+---------+----------+---------+
| cp1250_general_ci  | cp1250   | 26 | Yes     |          |       0 |
| cp1250_czech_ci    | cp1250   | 34 |         | Yes      |       2 |
| cp1250_bin         | cp1250   | 66 |         |          |       0 |
+--------------------+----------+----+---------+----------+---------+
```

- cp852（DOS 中央ヨーロッパ）:

```
+--------------------+----------+----+---------+----------+---------+
| Collation          | Charset  | Id | Default | Compiled | Sortlen |
+--------------------+----------+----+---------+----------+---------+
| cp852_general_ci   | cp852    | 40 | Yes     |          |       0 |
| cp852_bin          | cp852    | 81 |         |          |       0 |
+--------------------+----------+----+---------+----------+---------+
```

- macce（Mac 中央ヨーロッパ）:

```
+--------------------+----------+----+---------+----------+---------+
| Collation          | Charset  | Id | Default | Compiled | Sortlen |
+--------------------+----------+----+---------+----------+---------+
| macce_general_ci   | macce    | 38 | Yes     |          |       0 |
| macce_bin          | macce    | 43 |         |          |       0 |
+--------------------+----------+----+---------+----------+---------+
```

- latin2（ISO 8859-2 中央ヨーロッパ）:

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

```
| Collation           | Charset  | Id | Default | Compiled | Sortlen |
+---------------------+----------+----+---------+----------+---------+
| latin2_czech_ci     | latin2   |  2 |         | Yes      |       4 |
| latin2_general_ci   | latin2   |  9 | Yes     |          |       0 |
| latin2_hungarian_ci | latin2   | 21 |         |          |       0 |
| latin2_croatian_ci  | latin2   | 27 |         |          |       0 |
| latin2_bin          | latin2   | 77 |         |          |       0 |
+---------------------+----------+----+---------+----------+---------+
```

- keybcs2（DOS Kamenicky Czech-Slovak）:

```
+---------------------+----------+----+---------+----------+---------+
| Collation           | Charset  | Id | Default | Compiled | Sortlen |
+---------------------+----------+----+---------+----------+---------+
| keybcs2_general_ci  | keybcs2  | 37 | Yes     |          |       0 |
| keybcs2_bin         | keybcs2  | 73 |         |          |       0 |
+---------------------+----------+----+---------+----------+---------+
```

9.11.8. 西ヨーロッパのキャラクタセット

西ヨーロッパのキャラクタセットには、大部分の西ヨーロッパ言語（フランス語、スペイン語、カタロニア語、バスク語、ポルトガル語、イタリア語、アルバニア語、オランダ語、ドイツ語、デンマーク語、スウェーデン語、ノルウェー語、フィンランド語、フェロー語、アイスランド語、アイルランド語、スコットランド語、英語など）が含まれています。

- latin1（ISO 8859-1 西ヨーロッパ）:

```
+---------------------+----------+----+---------+----------+---------+
| Collation           | Charset  | Id | Default | Compiled | Sortlen |
+---------------------+----------+----+---------+----------+---------+
| latin1_german1_ci   | latin1   |  5 |         |          |       0 |
| latin1_swedish_ci   | latin1   |  8 | Yes     | Yes      |       0 |
| latin1_danish_ci    | latin1   | 15 |         |          |       0 |
| latin1_german2_ci   | latin1   | 31 |         | Yes      |       2 |
| latin1_bin          | latin1   | 47 |         | Yes      |       0 |
| latin1_general_ci   | latin1   | 48 |         |          |       0 |
| latin1_general_cs   | latin1   | 49 |         |          |       0 |
+---------------------+----------+----+---------+----------+---------+
```

latin1_swedish_ci 照合順序は、MySQL ユーザの大多数がデフォルトとして使用していると考えられます。一貫して述べてきたように、これはスウェーデン語/フィンランド語の照合順序ルールに基づいていますが、スウェーデンとフィンランドには意見の異なるユーザも存在します。

latin1_german1_ci および latin1_german2_ci 照合順序は DIN-1 および DIN-2 標準に基づきます。ここで DIN とは Deutsches Institut für Normung（ドイツ工業規格。ANSI のドイツ版）を指します。 DIN-1 は辞書照合、DIN-2 は電話帳照合と呼ばれています。

- latin1_german1_ci（辞書）ルール:

  ```
  'Ä' = 'A', 'Ö' = 'O', 'Ü' = 'U', 'ß' = 's'
  ```

  - latin1_german2_ci（電話帳）ルール:

    ```
    'Ä' = 'AE', 'Ö' = 'OE', 'Ü' = 'UE', 'ß' = 'ss'
    ```

- macroman（Mac 西ヨーロッパ）:

```
+---------------------+----------+----+---------+----------+---------+
| Collation           | Charset  | Id | Default | Compiled | Sortlen |
+---------------------+----------+----+---------+----------+---------+
| macroman_general_ci | macroman | 39 | Yes     |          |       0 |
| macroman_bin        | macroman | 53 |         |          |       0 |
+---------------------+----------+----+---------+----------+---------+
```

- cp850（DOS 西ヨーロッパ）:

```
+---------------------+----------+----+---------+----------+---------+
| Collation           | Charset  | Id | Default | Compiled | Sortlen |
+---------------------+----------+----+---------+----------+---------+
| cp850_general_ci    | cp850    |  4 | Yes     |          |       0 |
| cp850_bin           | cp850    | 80 |         |          |       0 |
+---------------------+----------+----+---------+----------+---------+
```

第10章 MySQL における空間情報の機能

MySQL 4.1 では空間情報を扱う機能が導入され、地理的特性の生成、格納、分析が可能になっています。 地理情報は現在、MyISAM テーブルに対してのみ使用することができます。

この章で取り上げるトピックは以下のとおりです。

- OpenGIS ジオメトリモデルにおける空間情報を扱う機能の基本
- 空間データを表現するためのデータ形式
- MySQL で空間データを使用する方法
- 空間データに対する索引作成の使用
- MySQL が OpenGIS 仕様と異なる点

10.1. はじめに

MySQL は、Open GIS Consortium (OGC) の仕様に準拠した空間情報の機能を実装します。OGC は、合計 250 を超える団体からなる国際コンソーシアムです。加盟している企業、政府機関、大学では、公開可能な概念ソリューションを開発しており、それらのソリューションは、空間データを管理するあらゆるアプリケーションで使用できるものです。 OGC の Web サイトは http://www.opengis.org/ です。

1997 年、Open GIS Consortium は OpenGIS (R) Simple Features Specifications For SQL を公開しました。この文書では、SQL RDBMS を拡張して空間データをサポートするための概念的な手法がいくつか提案されています。この仕様は、Open GIS Web サイト (http://www.opengis.org/techno/implementation.htm) で入手可能であり、この章の補足情報が記載されています。

MySQL は、OGC により提案されているジオメトリタイプを有する SQL 環境のサブセットを実装します。 これは、ジオメトリタイプのセットにより拡張された SQL 環境のことです。ジオメトリ値を持つ SQL カラムは、ジオメトリタイプのカラムとして実装されます。仕様には、SQL ジオメトリタイプのセットと、ジオメトリ値の作成と分析に使用される、これらのタイプの各種関数について記述されています。

地理的特性とは、実世界において位置を持つ存在を指します。 地理的特性の例を以下に示します。

- 実体。たとえば山、池、都市。
- 空間。たとえば郵便番号エリア、熱帯地方。
- 定義可能な場所。たとえば交差道路 (2 つの通りが交差する地点) 。

また、geospatial(地理空間) という用語が使用されている文書を探し、地理空間について詳しい情報を得ることもできます。

ジオメトリは、地理情報の機能を表す別の用語です。ジオメトリは本来、数学の一分野である幾何学を意味します。 もう 1 つの意味は地図作成法に由来し、地図の作成時に使用される地理的な特性を指します。

この章では、地理的特性、地理空間特性、地物、ジオメトリをすべて同義語として使用しています。 使用頻度が最も高い用語はジオメトリです。

ここでは、ジオメトリを位置を持ち存在を表すポイントまたはポイントの集まりと定義します。

10.2. OpenGIS ジオメトリモデル

OGC のジオメトリタイプを有する SQL 環境で提案されているジオメトリタイプのセットは、OpenGIS ジオメトリモデルに基づきます。このモデルでは、各ジオメトリオブジェクトに以下の汎用特性があります。

- 空間参照系（オブジェクトが定義される座標空間を示す）に関連付けられている。
- ジオメトリクラスに属している。

10.2.1. ジオメトリクラス階層

ジオメトリクラスの階層は以下のとおりです。

- Geometry（インスタンス化できない）
  - Point（インスタンス化できる）
  - Curve（インスタンスできない）
    - LineString（インスタンス化できる）
      - Line
      - LinearRing
  - Surface（インスタンス化できない）
    - Polygon（インスタンス化できる）
  - GeometryCollection（インスタンス化できる）
    - MultiPoint（インスタンス化できる）
    - MultiCurve（インスタンス化できない）
      - MultiPoint（インスタンス化できる）
    - MultiSurface（インスタンス化できない）
      - MultiPolygon（インスタンス化できる）

これらのクラスの一部は抽象クラス（インスタンス化できないクラス）です。つまり、これらのクラスのオブジェクトを作成することはできません。他のクラスはインスタンス化が可能であり、オブジェクトを作成することができます。 各クラスにはプロパティがあり、インスタンス化可能なクラスにはアサート（有効なクラスインスタンスを定義するルール）が含まれている場合があります。

Geometry は基本クラスです。これは抽象クラスです。 Geometry のインスタンス化可能サブクラスは、2 次元の座標空間に存在する 0 次元、1 次元、2 次元のジオメトリオブジェクトに制限されています。インスタンス化可能なジオメトリクラスはすべて、有効なインスタンスが位相的に閉じるように定義されています (すべての定義済みジオメトリには境界が含まれています)。

基本 Geometry クラスには、Point、Curve、Surface、GeometryCollection のサブクラスがあります。

- Point は 0 次元オブジェクトを表す。
- Curve は 1 次元オブジェクトを表す。このクラスには、サブクラス LineString のほか、サブサブクラス Line および LinearRing が属している。
- Surface は 2 次元オブジェクト用に設計されており、その下にサブクラス Polygon がある。
- GeometryCollection には、0 次元、1 次元、2 次元のコレクションクラス、MultiPoint、MultiLineString、MultiPolygon が含まれている。これらのクラスは、それぞれPoint、LineString、Polygon のコレクションになっているジオメトリをモデル化するためのものである。 MultiCurve と MultiSurface は、複数の Curve と Surface を処理するコレクションインタフェースを汎用化する抽象スーパークラスとして導入されている。

Geometry、Curve、Surface、MultiCurve、MultiSurface は、インスタンス化できないクラスとして定義されます。 これらのクラスには、それぞれのサブクラス用メソッドの共通セットが定義されています。これらは拡張目的で含まれています。

Point、LineString、Polygon、GeometryCollection、 MultiPoint、MultiLineString、MultiPolygon はインスタンス化が可能なクラスです。

10.2.2. Geometry クラス

Geometry は階層のルートクラスです。これはインスタンス化できないクラスですが、Geometry サブクラスから作成されたすべてのジオメトリの値に対して共通な多数のプロパティを含んでいます。 これらのプロパティを以下に示します。特定のサブクラスには固有のプロパティがありますが、それについては後述します。

ジオメトリのプロパティ

ジオメトリ値には以下のプロパティがあります。

- タイプ。 各ジオメトリは、階層上のインスタンス化可能なクラスのいずれかに属す。
- SRID (Spatial Reference Identifier = 空間参照識別子)。ジオメトリに関連付けられる空間参照系 (ジオメトリオブジェクトが定義される座標空間を示す) が、この値で識別される。
- 空間参照系における座標。倍精度 (8 バイト) の数値で表現される。空白以外のジオメトリには、少なくとも 1 組 (X,Y) の座標が含まれている。空白のジオメトリには座標が含まれない。

  座標は SRID に関連付けられている。 たとえば、異なった座標系では、両方のオブジェクトの座標が一致していてもオブジェクト間の距離が一致しないことがある。平面座標系での距離と地心座標系 (地球表面上の座標) での距離は同じものではないからである。

- 内部、境界、外部。

すべてのジオメトリは、空間のどこかの位置を占める。ジオメトリの外部とは、ジオメトリによって占有されていない空間のすべてを指す。ジオメトリの内部とは、ジオメトリによって占有されている空間を指す。境界は、ジオメトリの内部と外部の接触面である。

- MBR（Minimum Bounding Rectangle = 最小外接矩形）または Envelope。 これは境界ジオメトリであり、最小と最大（X,Y）の座標で形成される。

  ```
  ((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
  ```

- 単純であるか非単純であるかの性質。 いくつかのタイプ（LineString、MultiPoint、MultiLineString）のジオメトリ値は単純または非単純になる。単純か非単純かに関する判定は各タイプで決定。
- 閉じているか閉じていないかの性質。 いくつかのタイプ（LineString、MultiString）のジオメトリ値は、閉じている/閉じていないのいずれかになる。閉じているかどうかに関する独自の判定は各タイプで決定。
- 空か空でないかの性質。 ジオメトリは、Point を持たない場合に空になる。 空白ジオメトリの外部、内部、境界は定義されない（つまり、NULL 値で表現される）。 空白ジオメトリは、常に単純で 0 の領域を持つように定義される。
- 次元。ジオメトリは、−1、0、1、2 のいずれかの次元になる。
  - −1 は、ジオメトリが空であることを示す。
  - 0 は、ジオメトリに長さと領域がないことを示す。
  - 1 は、ジオメトリの長さが 0 以外で領域が 0 であることを示す。
  - 2 は、ジオメトリの領域が 0 以外であることを示す。

  Point オブジェクトは 0 次元。LineString オブジェクトは 1 次元。Polygon オブジェクトは 2 次元。MultiPoint、MultiLineString、および MultiPolygon オブジェクトの次元は、それぞれの構成要素の次元と同じ。

10.2.3. クラス Point

Point は、座標空間における単一の場所を表すジオメトリです。

Point の例

- 縮尺の大きい世界地図に多数の都市が含まれている場合、Point は各都市を表す。
- 市内地図の場合、Point はバス停を表す。

Point のプロパティ

- X 座標値。
- Y 座標値。
- Point は、0 次元のジオメトリとして定義される。

- Point の境界は空のセット。

10.2.4. Curve クラス

Curve は 1 次元のジオメトリであり、通常は一連の Point によって表現されます。Curve の特定のサブクラスでは、Point 間の内挿が定義されます。Curve はインスタンス化できないクラスです。

Curve のプロパティ

- 各 Point の座標。
- Curve は、1 次元のジオメトリとして定義される。
- Curve は、同じ Point を 2 回通過しない場合には単純になる。
- Curve は、開始 Point が終了 Point と一致する場合に閉じている。
- 閉じた Curve は空である。
- 閉じていない Curve の境界は、2 つの終了 Point で構成される。
- 単純で閉じている Curve は LinearRing である。

10.2.5. LineString クラス

LineString は、Point 間が直線補間されている Curve です。

LineString の例

- 世界地図の場合、LineString オブジェクトは河川を表す。
- 市内地図の場合、LineString オブジェクトは通りを表す。

LineString のプロパティ

- 連続したポイントのペアによって定義された LineString セグメントの座標。
- LineString は、2 つの Point のみで構成される場合には Line。
- LineString は、閉じていて単純な場合には LinearRing。

10.2.6. Surface クラス

Surface は 2 次元のジオメトリです。インスタンス化できないクラスであり、その下に属するインスタンス化可能なサブクラスは Polygon だけです。

Surface のプロパティ

- Surface は 2 次元のジオメトリとして定義される。
- OpenGIS 仕様では、単純な Surface は、単一の外部境界と 0 以上の内部境界に関連付けられている単一の ``区間" のみで構成されるジオメトリとして定義されている。
- 単純な Surface の境界は、外部境界および内部境界に対応する閉じた Curve のセット。

10.2.7. Polygon クラス

Polygon は、複数の辺を持つジオメトリを表す平面 Surface です。単一の外部境界および 0 以上の内部境界によって定義されます。内部境界はそれぞれ Polygon 内の穴を定義します。

Polygon の例

- 地域地図の場合、Polygon オブジェクトは森林、地区その他を表す。

Polygon のアサート

- Polygon の境界は、外部境界と内部境界を形成する LinearRing オブジェクト (単純で閉じている LineString オブジェクト) のセットで構成される。
- 2 つの Ring が境界で交わることはない。Polygon の境界にある Ring が Point で交わることはある。ただし、その場合は接線として交わるのみ。
- Polygon には、切断線、突起、陥没のいずれもあってはならない。
- 各 Polygon は、接続された Point のセットである。
- 1 つ以上の穴がある Polygon の外部は接続されていない。 それぞれの穴は、外部の接続されたコンポーネントを示す。

上記のアサートでは、Polygon は単純なジオメトリです。これらのアサートにより、Polygon は単純なジオメトリになります。

10.2.8. GeometryCollection クラス

GeometryCollection は単一のジオメトリですが、任意クラスのジオメトリ (1 つ以上) のコレクションとなっています。

GeometryCollection に含まれる要素はすべて、同じ座標系に存在していなければなりません。 GeometryCollection では、他の制限を要素に課していません。ただし、以降で説明する GeometryCollection のサブクラスによってメンバーシップが制限されることがあります。制限は以下に基づきます。

- 要素タイプ (たとえば MultiPoint には Point 要素のみ含まれる)
- 次元
- 要素間の空間的重なりの度合いに対する制限

10.2.9. MultiPoint クラス

MultiPoint は、Point 要素で構成されるジオメトリコレクションです。各ポイントは接続されておらず、順序付けられてもいません。

MultiPoint の例

- 世界地図の場合、Multipoint は小さな群島を表す。
- 市内地図の場合、Multipoint はチケット販売チェーン店を表す。

MultiPoint のプロパティ

- MultiPoint は、0 次元のジオメトリとして定義される。
- MultiPoint は、同じ（座標値が一致する）Point 値のペアがない場合には単純である。
- MultiPoint の境界は空のセット。

10.2.10. MultiCurve クラス

MultiCurve は、Curve 要素で構成されるジオメトリコレクションです。MultiCurve はインスタンス化できないクラスです。

MultiCurve のプロパティ

- MultiCurve は、1 次元のジオメトリとして定義される。
- MultiCurve が単純であるのは、その要素がすべて単純で、あらゆる 2 要素の交点のみが 2 要素の境界上のポイントに存在する場合だけである。
- MultiCurve を取得するには、次の ``mod 2 union rule'' （別名は odd-even rule）を適用する。 すなわち、MultiCurve 要素で奇数の境界にある Point は、MultiCurve の境界にある。
- MultiCurve は、すべての要素が閉じている場合に閉じている。
- 閉じた MultiCurve は常に空である。

10.2.11. MultiLineString クラス

MultiLineString は、LineString 要素で構成される MultiCurve ジオメトリコレクションです。

MultiLineString の例

- 地域地図の場合、MultiLineString は水系または幹線道路網を表す。

10.2.12. MultiSurface クラス

MultiSurface は、地表面要素で構成されるジオメトリコレクションです。 MultiSurface はインスタンス化できないクラスであり、その下に属するインスタンス化可能なサブクラスは MultiPolygon だけです。

MultiSurface のアサート

- MultiSurface では、2 つの Surface の内部が交差することはない。
- MultiSurface では、いずれか 2 つの要素の境界が有限の点数で交差することがある。

10.2.13. MultiPolygon クラス

MultiPolygon は、Polygon 要素で構成される MultiSurface オブジェクトです。

MultiPolygon の例

- 地域地図の場合、MultiPolygon は湖沼系を表す。

MultiPolygon のアサート

- MultiPolygon の要素になっている 2 つの Polygon 値の内部が交差することはない。
- MultiPolygon の要素になっているいずれか 2 つの Polygon 値における境界も交差することはなく、有限ポイント数で接触することがあるだけである。（前述したアサートにおいても、交差は禁止されている。）
- MultiPolygon には、切断線、突起、陥没のいずれもあってはならない。MultiPolygon は、通常の閉じた Point セットである。
- 1 つ以上の Polygon で構成される MultiPolygon の内部は接続されていない。MultiPolygon 内部の接続されたコンポーネントの数は、MultiPolygon における Polygon 値の数と一致する。

MultiPolygon のプロパティ

- MultiPolygon は 2 次元のジオメトリとして定義される。
- MultiPolygon の境界は、Polygon 要素の境界に対応する閉じた Curve（LineString 値）のセットである。
- MultiPolygon の境界内の各 Curve は、単一の要素 Polygon の境界に当たる。
- 要素 Polygon の境界の各 Curve は、MultiPolygon の境界に属す。

10.3. サポートされている空間データ形式

このセクションでは、クエリでジオメトリオブジェクトを表現するために使用される標準的な空間データ形式について説明します。 具体的には、以下のデータ形式が対象となります。

- Well-Known Text（WKT）形式

- Well-Known Binary (WKB) 形式

MySQL でジオメトリ値が内部的に保存される形式は、WKT 形式でも WKB 形式でもありません。

10.3.1. Well-Known Text (WKT) 形式

ジオメトリの Well-Known Text (WKT) 表現は、ジオメトリデータを ASCII 形式で交換するためのものです。

ジオメトリオブジェクトの WKT の例

- Point:

```
POINT(15 20)
```

  注意: Point 座標はカンマなしで指定される。

- Point が 4 つある LineString。

```
LINESTRING(0 0, 10 10, 20 25, 50 60)
```

- 外部 Ring と内部 Ring が 1 つずつある Polygon。

```
POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))
```

- Point 値が 3 つある MultiPoint。

```
MULTIPOINT(0 0, 20 20, 60 60)
```

- LineString 値が 2 つある MultiLineString。

```
MULTILINESTRING((10 10, 20 20), (15 15, 30 15))
```

- Polygon 値が 2 つある MultiPolygon。

```
MULTIPOLYGON(((0 0,10 0,10 10,0 10,0 0)),((5 5,7 5,7 7,5 7, 5 5)))
```

- 2 つの Point 値と 1 つの LineString で構成される GeometryCollection。

```
GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30), LINESTRING(15 15, 20 20))
```

WKT 値を記述するための正式なプロダクションルールを指定する Backus-Naur 文法については、この章の冒頭で言及されている OGC 仕様書を参照してください。

10.3.2. Well-Known Binary (WKB) 形式

ジオメトリック値の Well-Known Binary (WKB) 表現は、OpenGIS 仕様によって定義されています。 また、ISO ``SQL/MM Part 3: Spatial'' 標準でも定義されています。

ジオメトリック WKB 情報を含む BLOB 値で表現されるバイナリストリームとしてジオメトリ値を交換するために、WKB

を使用します。

1 バイトの符号なし整数、4 バイトの符号なし整数、8 バイトの倍精度数値 (IEEE 754 形式) が WKB によって使用されます。1 バイトは 8 ビットです。

たとえば WKB 値が POINT(1 1) に対応する場合、この値は次の 21 バイトの連続 (ここでは 2 桁の 16 進数で 1 バイト) で構成されます。

```
010100000000000000000F03F000000000000F03F
```

上記の数値は以下の構成要素に分割できます。

```
Byte order : 01
WKB type   : 01000000
X          : 000000000000F03F
Y          : 000000000000F03F
```

構成要素の表現は以下のとおりです。

- バイト順位は 0 または 1 であり、それぞれリトルエンディアンとビッグエンディアンによる格納を示す。リトルエンディアンとビッグエンディアンによるバイト順位は、それぞれネットワークデータ表現 (NDR) と外部データ表現 (XDR) としても知られている。
- WKB タイプは、ジオメトリタイプを示すコードである。1 ～ 7 の値はそれぞれ、Point、LineString、Polygon、MultiPoint、MultiLineString、MultiPolygon、GeometryCollection を示す。
- Point の値は X と Y の座標を含んでおり、それぞれ倍精度の値として表現される。

より複雑なジオメトリ値に対応する WKB 値は、より複雑なデータ構造で表現されます (詳細については OpenGIS 仕様を参照してください) 。

10.4. 空間的に有効な MySQL データベースの作成

このセクションでは、MySQL で空間データを表現するために使用できるデータ型と、空間値の作成および検索に使用できる関数について説明します。

10.4.1. MySQL 空間データ型

MySQL には、OpenGIS ジオメトリモデルのクラス階層内のクラスに対応するデータ型のセットが用意されています。これらのデータ型には、単一ジオメトリ値を保持するものもあります。

- GEOMETRY
- POINT
- LINESTRING
- POLYGON

GEOMETRY は、単一値データ型のうちで最も汎用的であり、どのタイプのジオメトリ値でも格納できます。 他の単一値データ型では、特定かつ単一のジオメトリタイプに値が制限されます。

上記以外のデータ型では、値のコレクションが保持されます。

- MULTIPOINT
- MULTILINESTRING
- MULTIPOLYGON
- GEOMETRYCOLLECTION

GEOMETRYCOLLECTION では、あらゆるタイプのオブジェクトのコレクションを格納できます。他のコレクションタイプでは、コレクションメンバは特定のジオメトリタイプを持つものに制限されます。

10.4.2. 空間情報の値の作成

このセクションでは、Well-Known Text および Well-Known Binary 関数（OpenGIS 標準で定義）と MySQL 固有の関数を使用して空間情報の値を作成する方法について説明します。

10.4.2.1. WKT 関数によるジオメトリ値の作成

MySQL には、Well-Known Text 表現（およびオプションとして空間参照系 ID（SRID））を入力パラメータとして受け取り、対応するジオメトリを返す関数が多数用意されています。

GeomFromText() は、任意のジオメトリタイプの WKT を最初の引数として受け取ります。タイプ固有の作成関数も用意されており、これらは各ジオメトリタイプのジオメトリ値を生成するために使用されます。

- GeomFromText(wkt[,srid]) , GeometryFromText(wkt[,srid])

  WKT 表現と SRID を使用して、任意のタイプのジオメトリ値を生成する。

- PointFromText(wkt[,srid])

  WKT 表現と SRID を使用して、POINT 値を生成する。

- LineFromText(wkt[,srid]) , LineStringFromText(wkt[,srid])

  WKT 表現と SRID を使用して、LINESTRING 値を生成する。

- PolyFromText(wkt[,srid]) , PolygonFromText(wkt[,srid])

  WKT 表現と SRID を使用して、POLYGON 値を生成する。

- MPointFromText(wkt[,srid]) , MultiPointFromText(wkt[,srid])

  WKT 表現と SRID を使用して、MULTIPOINT 値を生成する。

- MLineFromText(wkt[,srid]) , MultiLineStringFromText(wkt[,srid])

  WKT 表現と SRID を使用して、MULTILINESTRING 値を生成する。

- MPolyFromText(wkt[,srid]) , MultiPolygonFromText(wkt[,srid])

  WKT 表現と SRID を使用して、MULTIPOLYGON 値を生成する。

- GeomCollFromText(wkt[,srid]) , GeometryCollectionFromText(wkt[,srid])

  WKT 表現と SRID を使用して、GEOMETRYCOLLECTION 値を生成する。

Ring または閉じた LineString 値のコレクションの WKT 表現に基づいて Polygon 値または MultiPolygon 値を生成するためのオプション関数についても、OpenGIS 仕様に記述されています。これらの値は交差できます。これらの関数は、MySQL にまだ実装されていません。

- BdPolyFromText(wkt,srid)

  閉じた LineString 値から成る任意のコレクションが含まれている MultiLineString 値 (WKT 形式) から Polygon 値を生成する。

- BdMPolyFromText(wkt,srid)

  閉じた LineString 値から成る任意のコレクションが含まれている MultiLineString 値 (WKT 形式) から MultiPolygon 値を生成する。

10.4.2.2. WKB 関数によるジオメトリ値の作成

Well-Known Binary 表現が含まれている BLOB (およびオプションとして空間参照系 ID (SRID)) を入力パラメータとして受け取り、対応するジオメトリを返す関数が MySQL に多数用意されています。

GeomFromWKT() は、任意のジオメトリタイプの WKB を最初の引数として受け取ります。タイプ固有の作成関数も用意されており、これらは各ジオメトリタイプのジオメトリ値を生成するために使用されます。

- GeomFromWKB(wkb[,srid]) , GeometryFromWKB(wkt[,srid])

  WKB 表現と SRID を使用して、任意のタイプのジオメトリ値を生成する。

- PointFromWKB(wkb[,srid])

  WKB 表現と SRID を使用して、POINT 値を生成する。

- LineFromWKB(wkb[,srid]) , LineStringFromWKB(wkb[,srid])

  WKB 表現と SRID を使用して、LINESTRING 値を生成する。

- PolyFromWKB(wkb[,srid]) , PolygonFromWKB(wkb[,srid])

  WKB 表現と SRID を使用して、POLYGON 値を生成する。

- MPointFromWKB(wkb[,srid]) , MultiPointFromWKB(wkb[,srid])

  WKB 表現と SRID を使用して、MULTIPOINT 値を生成する。

- MLineFromWKB(wkb[,srid]) , MultiLineStringFromWKB(wkb[,srid])

  WKB 表現と SRID を使用して、MULTILINESTRING 値を生成する。

- MPolyFromWKB(wkb[,srid]) , MultiPolygonFromWKB(wkb[,srid])

  WKB 表現と SRID を使用して、MULTIPOLYGON 値を生成する。

- GeomCollFromWKB(wkb[,srid]) , GeometryCollectionFromWKB(wkt[,srid])

  WKB 表現と SRID を使用して、GEOMETRYCOLLECTION 値を生成する。

Ring または閉じた LineString 値のコレクションの WKB 表現に基づいて Polygon または MultiPolygon を生成するためのオプション関数についても、OpenGIS 仕様に記述されています。これらの値は交差できます。これらの関数は、MySQL にまだ実装されていません。

- BdPolyFromWKB(wkb,srid)

  閉じた LineString 値から成る任意のコレクションが含まれている MultiLineString 値 (WKB 形式) から Polygon 値を生成する。

- BdMPolyFromWKB(wkb,srid)

  閉じた LineString 値から成る任意のコレクションが含まれている MultiLineString 値 (WKB 形式) から MultiPolygon 値を生成する。

10.4.2.3. MySQL 固有の関数によるジオメトリ値の作成

注意: このセクションで取り上げる関数は、MySQL にまだ実装されていません。

ジオメトリ WKB 表現を作成する際に役立つ関数が MySQL に用意されています。このセクションで説明する関数は、OpenGIS 仕様に対する MySQL 拡張です。これらの関数の結果、SRID のないジオメトリ値の WKB 表現を含む BLOB 値が返されます。 これらの関数の結果は、GeomFromWKB() 関数ファミリに含まれるどの関数についても、最初の引数として代用できます。

- Point(x,y)

  座標を使用して WKB Point を生成する。

- MultiPoint(pt1,pt2,...)

  WKB Point 引数を使用して WKB MultiPoint 値を生成する。 いずれかの引数が WKBPoint でない場合、返り値は NULL になる。

- LineString(pt1,pt2,...)

  WKB Point 引数から WKB LineString 値を生成する。いずれかの引数が WKB Point でない場合、返り値は NULL になる。Point 引数が 2 個に満たない場合、返り値は NULL になる。

- MultiLineString(ls1,ls2,...)

  WKB LineString 引数を使用して WKB MultiLineString 値を生成する。いずれかの引数が LineString でない場合、返り値は NULL になる。

- Polygon(ls1,ls2,...)

  WKB LineString 引数から WKB Polygon 値を生成する。いずれかの引数が LinearRing のWKB を表さない場合 (閉じている単純な LineString でない場合) 、返り値は NULL になる。

- MultiPolygon(poly1,poly2,...)

  WKB Polygon 引数から WKB MultiPolygon 値を生成する。 いずれかの引数が WKB Polygon でない場合、返り値は NULL になる。

- GeometryCollection(g1,g2,...)

  WKB GeometryCollection を生成する。いずれかの引数がジオメトリの WKB 表現として適切ではない場合、返り値は NULL になる。

10.4.3. 空間情報カラムの作成

ジオメトリタイプに対して空間情報カラムを作成する標準的な方法が MySQL に用意されています (たとえば CREATE TABLE または ALTER TABLE を使用) 。 空間情報カラムは現在、MyISAM テーブルに対してのみサポートされています。

- CREATE TABLE ステートメントを使用し、空間情報カラムのあるテーブルを作成する。

```
mysql> CREATE TABLE geom (g GEOMETRY);
Query OK, 0 rows affected (0.02 sec)
```

- ALTER TABLE ステートメントを使用し、既存テーブルに空間情報カラムを追加したり、既存テーブルの空間情報カラムを破棄したりする。

```
mysql> ALTER TABLE geom ADD pt POINT;
Query OK, 0 rows affected (0.00 sec)
```

```
Records: 0  Duplicates: 0  Warnings: 0
mysql> ALTER TABLE geom DROP pt;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

10.4.4. 空間情報カラムへのデータ入力

作成した空間情報カラムには、空間データを入力することができます。

値は内部ジオメトリ形式で格納する必要がありますが、Well-Known Text（WKT）または Well-Known Binary（WKB）形式から内部ジオメトリ形式に変換することも可能です。WKT 値を内部ジオメトリ形式に変換してジオメトリ値をテーブルに挿入する方法を、以下の例に示します。

この変換は、INSERT ステートメントで直接実行することができます。

```
INSERT INTO geom VALUES (GeomFromText('POINT(1 1)'));

SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (GeomFromText(@g));
```

INSERT の前に変換が実行される場合もあります。

```
SET @g = GeomFromText('POINT(1 1)');
INSERT INTO geom VALUES (@g);
```

以下の例では、より複雑なジオメトリをテーブルに挿入しています。

```
SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (GeomFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (GeomFromText(@g));

SET @g = 'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (GeomFromText(@g));
```

前述したいずれの例でも、GeomFromText() を使用してジオメトリ値を作成しています。タイプ固有の関数を使用することもできます。

```
SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (PointFromText(@g));

SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (LineStringFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (PolygonFromText(@g));

SET @g = 'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (GeomCollFromText(@g));
```

注意: クライアントアプリケーションプログラムは、ジオメトリ値の WKB 表現を使用する場合、正しい形式の WKB をク

エリでサーバに送信しなければなりません。この要件を満たすにはいくつかの方法があります。たとえば、以下のようにします。

- POINT(1 1) 値を 16 進リテラル構文で挿入する。

```
mysql> INSERT INTO geom VALUES
    -> (GeomFromWKB(0x0101000000000000000000F03F000000000000F03F));
```

- ODBC アプリケーションは WKB 表現を送信し、それをプレースホルダにバインドすることができる（ BLOB 型の引数を使用 ）。

```
INSERT INTO geom VALUES (GeomFromWKB(?))
```

  他のプログラミングインタフェースでも、類似したプレースホルダ機構がサポートされている場合がある。

- C プログラムでは、mysql_real_escape_string() を使用してバイナリ値をエスケープし、サーバに送信されるクエリ文字列に結果を取り込むことができる。 See 項11.1.3.44.「mysql_real_escape_string()」。

10.4.5. 空間データの取り込み

テーブルに格納されたジオメトリ値は、内部形式に変換して取り込めます。この値は WKT または WKB 形式に変換することもできます。

10.4.5.1. 空間データの取り込み（ 内部形式を使用 ）

内部形式によるジオメトリ値の取り込みは、テーブル間の転送に有効です。

```
CREATE TABLE geom2 (g GEOMETRY) SELECT g FROM geom;
```

10.4.5.2. 空間データの取り込み（ WKT 形式を使用 ）

AsText() 関数では、ジオメトリ値をテキストとして使用することができます。この関数は、ジオメトリを内部形式から WKT 文字列に変換します。

```
mysql> SELECT AsText(g) FROM geom;
+------------------------+
| AsText(p1)             |
+------------------------+
| POINT(1 1)             |
| LINESTRING(0 0,1 1,2 2) |
+------------------------+
```

10.4.5.3. 空間データの取り込み（ WKB 形式を使用 ）

AsBinary() 関数では、ジオメトリ値をバイナリとして使用することができます。 この関数は、ジオメトリを内部形式から BLOB（ WKB 値を含む ）に変換します。

```
SELECT AsBinary(g) FROM geom;
```

10.5. 空間情報の分析

空間情報カラムに値を入力した後は、クエリと分析を行うことができます。空間データに対して各種の操作を実行するための関数が MySQL に用意されています。これらの関数は、実行する操作に基づき 4 つの主要なカテゴリに分類されます。

- 各種形式間でジオメトリを変換する関数
- ジオメトリの質的または量的プロパティへのアクセスを提供する関数
- 2 つのジオメトリの関係を記述する関数
- 既存ジオメトリから新規ジオメトリを作成する関数

以下のような多くの状況で、空間情報の分析関数を使用することができます。

- インタラクティブな SQL プログラム (mysql、MySQLCC など) 。
- MySQL クライアント API をサポートする言語で記述されたアプリケーションプログラム。

10.5.1. ジオメトリの形式を変換する関数

MySQL でサポートされている以下の関数では、WKT または WKB 形式と内部形式間でジオメトリ値を変換することができます。

- GeomFromText(wkt[,srid])

  文字列値を WKT 表現から内部ジオメトリ形式へと変換し、結果を返す。 タイプ固有の関数も数多くサポートされている (PointFromText()、LineFromText() など) 。項10.4.2.1.「WKT 関数によるジオメトリ値の作成」 参照。.

- GeomFromWKB(wkb[,srid])

  バイナリ値を WKB 表現から内部ジオメトリ形式へと変換し、結果を返す。 タイプ固有の関数も数多くサポートされている (PointFromWKB()、LineFromWKB() など) 。項10.4.2.2.「WKB 関数によるジオメトリ値の作成」 参照。

- AsText(g)

  内部ジオメトリ形式の値を WKT 表現に変換し、結果文字列を返す。

```
mysql> SET @g = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(GeomFromText(@g));
+--------------------------+
| AsText(GeomFromText(@G)) |
+--------------------------+
| LINESTRING(1 1,2 2,3 3)  |
+--------------------------+
```

- AsBinary(g)

  内部ジオメトリ形式の値を WKB 表現に変換し、結果のバイナリ値を返す。

10.5.2. Geometry プロパティ分析関数

このグループに属す関数は、ジオメトリ値を引数として使用し、ジオメトリの質的または量的プロパティを返します。一部の関数では、引数のタイプが制限されています。そのような関数は、引数のジオメトリタイプが不正な場合に NULL を返します。たとえば、オブジェクトタイプが Polygon と MultiPolygon のいずれでもない場合、Area() は NULL を返します。

10.5.2.1. 汎用 Geometry プロパティ分析関数

このセクションに列挙されている関数には引数に対する制限はなく、どのタイプのジオメトリ値も受け入れられます。

- GeometryType(g)

  ジオメトリインスタンス g がメンバになっているジオメトリタイプの名称を文字列として返す。 この名称は、インスタンス化可能な Geometry サブクラスの 1 つに対応する。

```
mysql> SELECT GeometryType(GeomFromText('POINT(1 1)'));
+---------------------------------------+
| GeometryType(GeomFromText('POINT(1 1)')) |
+---------------------------------------+
| POINT                                 |
+---------------------------------------+
```

- Dimension(g)

  ジオメトリ値 g 固有の次元を返す。結果は −1、0、1、2 のいずれか (これらの値の意味については 項10.2.2.「Geometry クラス」 を参照)。

```
mysql> SELECT Dimension(GeomFromText('LineString(1 1,2 2)'));
+------------------------------------------------+
| Dimension(GeomFromText('LineString(1 1,2 2)')) |
+------------------------------------------------+
|                                              1 |
+------------------------------------------------+
```

- SRID(g)

  ジオメトリ値 g の空間参照系 ID を示す整数を返す。

```
mysql> SELECT SRID(GeomFromText('LineString(1 1,2 2)',101));
+---------------------------------------------+
| SRID(GeomFromText('LineString(1 1,2 2)',101)) |
+---------------------------------------------+
|                                         101 |
+---------------------------------------------+
```

- Envelope(g)

  ジオメトリ値 g の最小外接矩形（MBR）を返す。 結果は Polygon 値として返す。

```
mysql> SELECT AsText(Envelope(GeomFromText('LineString(1 1,2 2)')));
+-----------------------------------------------------+
| AsText(Envelope(GeomFromText('LineString(1 1,2 2)'))) |
+-----------------------------------------------------+
| POLYGON((1 1,2 1,2 2,1 2,1 1))                      |
+-----------------------------------------------------+
```

  境界ボックスのコーナーポイントによって、Polygon が定義される。

```
POLYGON((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

OpenGIS 仕様では以下の関数も定義していますが、これらは MySQL でまだ実装されていません。

- Boundary(g)

  ジオメトリ値 g の組み合わせ境界の終わりとなるジオメトリを返す。

- IsEmpty(g)

  ジオメトリ値 g が空白ジオメトリであれば 1、空白でなければ 0、引数が NULL であれば −1 を返す。 ジオメトリが空の場合、空の Point セットを表す。

- IsSimple(g)

  現在、この関数はプレースホルダとなっており、使用すべきではない。 実装された場合の機能については、以下のとおり。

  ジオメトリ値 g に不正な（自己交差や自己接触が起きている）ジオメトリ Point がない場合、1 を返す。IsSimple() は、引数が単純であれば 0、引数が NULL であれば −1 を返す。

  インスタンス化可能な各ジオメトリクラスについては該当するセクションで説明したが、そこで、そのクラスのインスタンスが非単純と分類される場合の特定の条件について説明している。

10.5.2.2. Point プロパティ分析関数

Point はX 座標と Y 座標から構成されますが、これらは以下の関数を使用して取得できます。

- X(p)

  Point p の X 座標値を倍精度の数値として返す。

```
mysql> SELECT X(GeomFromText('Point(56.7 53.34)'));
+---------------------------------------+
| X(GeomFromText('Point(56.7 53.34)')) |
+---------------------------------------+
|                                  56.7 |
+---------------------------------------+
```

- Y(p)

  Point p の Y 座標値を倍精度の数値として返す。

```
mysql> SELECT Y(GeomFromText('Point(56.7 53.34)'));
+---------------------------------------+
| Y(GeomFromText('Point(56.7 53.34)')) |
+---------------------------------------+
|                                 53.34 |
+---------------------------------------+
```

10.5.2.3. LineString プロパティ分析関数

LineString は Point 値で構成されます。LineString に関し、その特定 Point を抽出するか、含まれている Point 数をカウントするか、その長さを取得することができます。

- EndPoint(ls)

  LineString 値 ls の終点となっている Point を返す。

```
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)                                                  |
+-------------------------------------------------------------+
```

- GLength(ls)

  関連する空間参照における LineString 値 ls の長さを倍精度の数値として返す。

```
mysql> SELECT GLength(GeomFromText('LineString(1 1,2 2,3 3)'));
+---------------------------------------------------+
| GLength(GeomFromText('LineString(1 1,2 2,3 3)')) |
+---------------------------------------------------+
|                          2.8284271247462 |
+---------------------------------------------------+
```

- IsClosed(ls)

  LineString 値 ls が閉じている (StartPoint() 値と EndPoint() 値が一致する) 場合、1 を返す。 ls が閉じてなければ 0、NULL であれば −1 を返す。

```
mysql> SELECT IsClosed(GeomFromText('LineString(1 1,2 2,3 3)'));
+---------------------------------------------------+
| IsClosed(GeomFromText('LineString(1 1,2 2,3 3)')) |
+---------------------------------------------------+
|                                                 0 |
+---------------------------------------------------+
```

- NumPoints(ls)

  LineString 値 ls における Point 数を返す。

```
mysql> SELECT NumPoints(GeomFromText('LineString(1 1,2 2,3 3)'));
+---------------------------------------------------+
| NumPoints(GeomFromText('LineString(1 1,2 2,3 3)')) |
+---------------------------------------------------+
|                                                 3 |
+---------------------------------------------------+
```

- PointN(ls,n)

  LineString 値 ls における n 番目の Point を返す。 Point 番号は 1 から始まる。

```
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)                                                  |
+-------------------------------------------------------------+
```

- StartPoint(ls)

  LineString 値 ls の始点となっている Point を返す。

```
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)                                                  |
+-------------------------------------------------------------+
```

OpenGIS 仕様では以下の関数も定義していますが、これは MySQL でまだ実装されていません。

- IsRing(ls)

  LineString 値 ls が閉じており (StartPoint() 値と EndPoint() 値が一致)、かつ、単純である (同じ Point を 2 回以上通過しない) 場合、1 を返す。 ls が Ring でなければ 0、NULL であれば −1 を返す。

10.5.2.4. MultiLineString プロパティ分析関数

- GLength(mls)

  MultiLineString 値 mls の長さを倍精度の数値として返す。mls の長さは、要素の長さを合計した値と一致する。

```
mysql> SELECT GLength(GeomFromText('MultiLineString((1 1,2 2,3 3),(4 4,5 5))'));
+-----------------------------------------------------------------+
| GLength(GeomFromText('MultiLineString((1 1,2 2,3 3),(4 4,5 5))')) |
+-----------------------------------------------------------------+
|                                        4.2426406871193 |
+-----------------------------------------------------------------+
```

- IsClosed(mls)

  MultiLineString 値 mls が閉じている (mls で各 LineString に対する StartPoint() 値と EndPoint() 値が一致する) 場合、1 を返す。 mls が閉じていなければ 0、NULL であれば −1 を返す。

```
mysql> SELECT IsClosed(GeomFromText('MultiLineString((1 1,2 2,3 3),(4 4,5 5))'));
+-----------------------------------------------------------------+
| IsClosed(GeomFromText('MultiLineString((1 1,2 2,3 3),(4 4,5 5))')) |
+-----------------------------------------------------------------+
|                                                               0 |
+-----------------------------------------------------------------+
```

10.5.2.5. Polygon プロパティ分析関数

- Area(poly)

  空間参照系での測定に基づいて、Polygon 値 poly の領域を倍精度の数値として返す。

```
mysql> SELECT Area(GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))'));
+-----------------------------------------------------------------------+
| Area(GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))')) |
+-----------------------------------------------------------------------+
|                                                                     8 |
+-----------------------------------------------------------------------+
```

- NumInteriorRings(poly)

  Polygon 値 poly における内部 Ring 数を返す。

```
mysql> SELECT NumInteriorRings(GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))'));
+-----------------------------------------------------------------------------+
| NumInteriorRings(GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))')) |
+-----------------------------------------------------------------------------+
|                                                                           1 |
+-----------------------------------------------------------------------------+
```

- ExteriorRing(poly)

  Polygon 値 poly の外部 Ring を LineString として返す。

```
mysql> SELECT AsText(ExteriorRing(GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))')));
+-------------------------------------------------------------------------------------+
| AsText(ExteriorRing(GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))'))) |
+-------------------------------------------------------------------------------------+
| LINESTRING(0 0,0 3,3 3,3 0,0 0)                                                     |
+-------------------------------------------------------------------------------------+
```

- InteriorRingN(poly,n)

  Polygon 値 poly の n 番目の内部 Ring を LineString として返す。 Ring 番号は 1 から始まる。

```
mysql> SELECT AsText(InteriorRingN(GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))'),1));
+-----------------------------------------------------------------------------------------+
| AsText(InteriorRingN(GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))'),1)) |
+-----------------------------------------------------------------------------------------+
| LINESTRING(1 1,1 2,2 2,2 1,1 1)                                                         |
+-----------------------------------------------------------------------------------------+
```

OpenGIS 仕様では以下の関数も定義していますが、これらは MySQL でまだ実装されていません。

- Centroid(poly)

Polygon 値 poly の数学的な重心を Point として返す。結果が Polygon 上に存在することは保証されない。

- PointOnSurface(poly)

  Polygon 値 poly に存在することが保証される Point 値を返す。

10.5.2.6. MultiPolygon プロパティ分析関数

- Area(mpoly)

  空間参照系での測定に基づいて、MultiPolygon 値 mpoly の領域を倍精度の数値として返す。

```
mysql> SELECT Area(GeomFromText('MultiPolygon(((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1)))'));
+---------------------------------------------------------------------------+
| Area(GeomFromText('MultiPolygon(((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1)))')) |
+---------------------------------------------------------------------------+
|                                                                         8 |
+---------------------------------------------------------------------------+
```

OpenGIS 仕様では以下の関数も定義していますが、これらは MySQL でまだ実装されていません。

- Centroid(mpoly)

  MultiPolygon 値 mpoly の数学的な重心を Point として返す。結果が MultiPolygon 上に存在することは保証されない。

- PointOnSurface(mpoly)

  MultiPolygon 値 mpoly 上に存在することが保証される Point 値を返す。

10.5.2.7. GeometryCollection プロパティ分析関数

- NumGeometries(gc)

  GeometryCollection 値 gc におけるジオメトリ数を返す。

```
mysql> SELECT NumGeometries(GeomFromText('GeometryCollection(Point(1 1),LineString(2 2, 3 3))'));
+---------------------------------------------------------------------------+
| NumGeometries(GeomFromText('GeometryCollection(Point(1 1),LineString(2 2, 3 3))')) |
+---------------------------------------------------------------------------+
|                                                                         2 |
+---------------------------------------------------------------------------+
```

- GeometryN(gc,n)

  GeometryCollection 値 gc における n 番目のジオメトリを返す。ジオメトリ番号は 1 から始まる。

```
mysql> SELECT AsText(GeometryN(GeomFromText('GeometryCollection(Point(1 1),LineString(2 2, 3 3))'),1));
+-------------------------------------------------------------------------------------+
| AsText(GeometryN(GeomFromText('GeometryCollection(Point(1 1),LineString(2 2, 3 3))'),1)) |
+-------------------------------------------------------------------------------------+
| POINT(1 1)                                                                          |
+-------------------------------------------------------------------------------------+
```

10.5.3. 既存ジオメトリから新規ジオメトリを作成する関数

10.5.3.1. 新規ジオメトリを作成する関数

セクション 項10.5.2.「Geometry プロパティ分析関数」 では、既存ジオメトリから新規ジオメトリを構築する関数をいくつか取り上げました。

- Envelope(g)
- StartPoint(ls)
- EndPoint(ls)
- PointN(ls,n)
- ExteriorRing(poly)
- InteriorRingN(poly,n)
- GeometryN(gc,n)

10.5.3.2. 空間情報の演算子

OpenGIS では、ジオメトリを作成できるその他の関数が数多く提案されています。これらは、空間演算子を実装するように設計されています。

そのような関数は MySQL では実装されておらず、将来のリリースで実装される予定です。

- Intersection(g1,g2)

  ジオメトリ値 g1 および g2 の Point セット交差を表すジオメトリを返す。

- Union(g1,g2)

  ジオメトリ値 g1 および g2 の Point セット接合を表すジオメトリを返す。

- Difference(g1,g2)

  ジオメトリ値 g1 および g2 の Point セット差異を表すジオメトリを返す。

- SymDifference(g1,g2)

ジオメトリ値 g1 および g2 の Point セット対称差異を表すジオメトリを返す。

- Buffer(g,d)

  ジオメトリ値 g からの距離が d の距離以下になっているすべての Point を表すジオメトリを返す。

- ConvexHull(g)

  ジオメトリ値 g の凸包を表すジオメトリを返す。

10.5.4. ジオメトリオブジェクト間の空間的関係をテストするための関数

これらのセクションで説明されている関数は、2 つのジオメトリを入力パラメータとして受け取り、ジオメトリ間の質的または量的な関係を返します。

10.5.5. 最小外接矩形（MBR）における関係

ジオメトリ g1 および g2 の最小外接矩形間の関係をテストできる関数が MySQL に用意されています。 たとえば、以下の関数があります。

- MBRContains(g1,g2)

  1 または 0 を返すことにより、g1 の最小外接矩形に g2 の最小外接矩形が含まれているかどうかを示す。

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Point(1 1)');
mysql> SELECT MBRContains(@g1,@g2), MBRContains(@g2,@g1);
---------------------+---------------------+
| MBRContains(@g1,@g2) | MBRContains(@g2,@g1) |
+----------------------+----------------------+
|                    1 |                    0 |
+----------------------+----------------------+
```

- MBRWithin(g1,g2)

  1 または 0 を返すことにより、g1 の最小外接矩形が g2 の最小外接矩形に含まれているかどうかを示す。

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Polygon((0 0,0 5,5 5,5 0,0 0))');
mysql> SELECT MBRWithin(@g1,@g2), MBRWithin(@g2,@g1);
+--------------------+--------------------+
| MBRWithin(@g1,@g2) | MBRWithin(@g2,@g1) |
+--------------------+--------------------+
|                  1 |                  0 |
+--------------------+--------------------+
```

- MBRDisjoint(g1,g2)

  1 または 0 を返すことにより、ジオメトリ g1 と g2 の 2 つの最小外接矩形が分離している（交差していない）かどうかを示す。

- MBREquals(g1,g2)

  1 または 0 を返すことにより、ジオメトリ g1 と g2 の 2 つの最小外接矩形が同じかどうかを示す。

- MBRIntersects(g1,g2)

  1 または 0 を返すことにより、ジオメトリ g1 と g2 の 2 つの最小外接矩形が交差しているどうかを示す。

- MBROverlaps(g1,g2)

  1 または 0 を返すことにより、ジオメトリ g1 と g2 の 2 つの最小外接矩形が重なっているかどうかを示す。

- MBRTouches(g1,g2)

  1 または 0 を返すことにより、ジオメトリ g1 と g2 の 2 つの最小外接矩形が接触しているどうかを示す。

10.5.6. ジオメトリ間の空間関係をテストする関数

OpenGIS 仕様では以下の関数を定義していますが、これらは MySQL でまだ実装されていません。将来のリリースで実装される予定です。 実装されると、MBR ベースのサポートのみならず、空間分析の完全なサポートが提供されます。

これらの関数は、2 つのジオメトリ値 g1 および g2 で動作します。

- Contains(g1,g2)

  1 または 0 を返すことにより、g1 に g2 が完全に含まれているかどうかを示す。

- Crosses(g1,g2)

  g1 が g2 と空間的にクロスする場合、1 を返す。 g1 が Polygon または MultiPolygon であるか、あるいは g2 が Point または MultiPoint である場合、NULL を返す。 その他の場合は 0 を返す。

  空間的にクロスするという表現は、以下のプロパティを持つ 2 つの指定されたジオメトリの関係を示す。

  - 2 つのジオメトリは交差する。
  - 交差した部分が単一のジオメトリとなり、そのジオメトリの次元は指定された 2 つのジオメトリの最大次元よりも1つ小さくなる。
  - 交差した部分は、指定された 2 つのジオメトリと等しくない。

- Disjoint(g1,g2)

1 または 0 を返すことにより、g1 と g2 が空間的に分離している（交差しない）かどうかを示す。

- Equals(g1,g2)

  1 または 0 を返すことにより、g1 と g2 が空間的に等しいかどうかを示す。

- Intersects(g1,g2)

  1 または 0 を返すことにより、g1 と g2 が空間的に交差するどうかを示す。

- Overlaps(g1,g2)

  1 または 0 を返すことにより、g1 と g2 が空間的に重なるかどうかを示す。 空間的に重なるという用語が使用されるのは、2 つのジオメトリの交差部分が単一のジオメトリになり、そのジオメトリは指定された 2 つのジオメトリと次元が一致するが、どちらのジオメトリとも等しくない場合である。

- Touches(g1,g2)

  1 または 0 を返すことにより、g1 と g2 が空間的に接触するどうかを示す。2 つのジオメトリが空間的に接触するのは、ジオメトリ内部は交差しないが、一方のジオメトリの境界が他方のジオメトリの境界または内部と交差する場合である。

- Within(g1,g2)

  1 または 0 を返すことにより、g1 が g2 に空間的に含まれているかどうかを示す。

- Distance(g1,g2)

  2 つのジオメトリにおける任意の 2 Point の最短距離を倍精度の数値として返す。

- Related(g1,g2,pattern_matrix)

  1 または 0 を返すことにより、pattern_matrix によって指定された空間関係が g1 と g2 の間に存在するかどうかを示す。 引数が NULL である場合は –1 を返す。 パターン行列は文字列である。パターン行列については、この関数が実装された時点でここに記載する。

10.6. 空間分析の最適化

すでに知られていることですが、非空間データベースでの検索操作はインデックスを使用して最適化できます。このことは、空間データベースについても当てはまります。 さまざまな多次元インデックス作成方法が考案されており、それらを活用すれば空間検索を最適化することができます。代表的な例を以下に示します。

- 指定されたポイントが含まれているオブジェクトをすべて検索するポイントクエリ。
- 指定された領域が含まれているオブジェクトをすべて検索する領域クエリ。

MySQL では、二次分割アルゴリズムを利用した R-Tree を使用して空間カラムにインデックスを付けます。空間インデックスは、ジオメトリの MBR を使用して作成されます。 多くのジオメトリに対しては、MBR はジオメトリを囲む最小矩形です。水平または垂直な LineString に対しては、MBR は LineString に縮退した矩形です。 Point に対しては、MBR は Point に縮退した矩形です。

10.6.1. 空間インデックスの作成

MySQL では空間インデックスを作成できます。作成時に使用する構文は、通常のインデックスを作成するための構文に似ていますが、SPATIAL キーワードで拡張されています。 現在インデックス付けされている空間カラムは、NOT NULL と宣言されなければなりません。空間インデックスを作成する方法を以下の例に示します。

- CREATE TABLE を使用する場合。

```
mysql> CREATE TABLE geom (g GEOMETRY NOT NULL, SPATIAL INDEX(g));
```

- ALTER TABLE を使用する場合。

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
```

- CREATE INDEX を使用する場合。

```
mysql> CREATE SPATIAL INDEX sp_index ON geom (g);
```

空間インデックスを破棄するには、ALTER TABLE または DROP INDEX を使用します。

- ALTER TABLE を使用する場合。

```
mysql> ALTER TABLE geom DROP INDEX g;
```

- DROP INDEX を使用する場合。

```
mysql> DROP INDEX sp_index ON geom;
```

例: テーブル geom に 32,000 個を超えるジオメトリが格納されており、これらが GEOMETRY 型の g で保存されているとします。 このテーブルには AUTO_INCREMENT カラム fid もあり、このカラムにオブジェクト ID 値が保存されます。

```
mysql> SHOW FIELDS FROM geom;
+-------+----------+------+-----+---------+----------------+
| Field | Type     | Null | Key | Default | Extra          |
+-------+----------+------+-----+---------+----------------+
| fid   | int(11)  |      | PRI | NULL    | auto_increment |
| g     | geometry |      |     |         |                |
+-------+----------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM geom;
+----------+
| count(*) |
+----------+
```

```
|   32376 |
+----------+
1 row in set (0.00 sec)
```

カラム g で空間インデックスを追加するには、以下のステートメントを使用します。

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
Query OK, 32376 rows affected (4.05 sec)
Records: 32376  Duplicates: 0  Warnings: 0
```

10.6.2. 空間インデックスの使用

MBRContains() や MBRWithin() などの関数を WHERE 句で使用するクエリの検索で空間インデックスを使用できるかどうかが、オプティマイザによって確認されます。 たとえば、指定された矩形に含まれているオブジェクトをすべて検索する必要があるとします。

```
mysql> SELECT fid,AsText(g) FROM geom WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000 16000,30000 16000,30000 15000))'),g);
+-----+-------------------------------------------------------------------------+
| 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)
```

このクエリがどのように実行されるかを、EXPLAIN を使用して確認します。

```
mysql> EXPLAIN SELECT fid,AsText(g) FROM geom WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000 16000,30000 16000,30000 15000))'),g);
+----+-------------+-------+-------+---------------+------+---------+------+------+-------------+
| id | select_type | table | type  | possible_keys | key  | key_len | ref  | rows | Extra       |
+----+-------------+-------+-------+---------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | geom  | range | g             | g    |      32 | NULL |   50 | Using where |
+----+-------------+-------+-------+---------------+------+---------+------+------+-------------+
1 row in set (0.00 sec)
```

空間インデックスがないとどうなるかを確認します。

```
mysql> EXPLAIN SELECT fid,AsText(g) FROM g IGNORE INDEX (g) WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000 16000,30000 16000,30000 15000))'),g);
+----+-------------+-------+------+---------------+------+---------+------+-------+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows  | Extra       |
+----+-------------+-------+------+---------------+------+---------+------+-------+-------------+
|  1 | SIMPLE      | geom  | ALL  | NULL          | NULL |    NULL | NULL | 32376 | Using where |
+----+-------------+-------+------+---------------+------+---------+------+-------+-------------+
1 row in set (0.00 sec)
```

上記のクエリを実行し、使用可能な空間キーを無視します。

```
mysql> SELECT fid,AsText(g) FROM geom IGNORE INDEX (g) WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000 16000,30000 16000,30000 15000))'),g);
+-----+-------------------------------------------------------------------------+
| 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)
```

空間インデックスを使用しないと、このクエリの実行時間は 0.00 秒から 0.46 秒に増大します。

将来のリリースでは、空間インデックスを使用して他の関数も最適化されるようになります。 See 項10.5.4.「ジオメトリオブジェクト間の空間的関係をテストするための関数」。

10.7. MySQL の適合性と互換性

10.7.1. まだ実装されていない GIS 機能

- 追加のメタデータビュー

  OpenGIS 仕様には、追加のメタデータビューがいくつか提案されている。 たとえば、システムビュー

GEOMETRY_COLUMNS にはジオメトリカラムの説明があり、1 行がデータベース内の各ジオメトリカラムに対応している。

- 空間情報カラムを追加/破棄するための関数

  OpenGIS では、特別な関数 AddGeometryColumn() と DropGeometryColumn() を使用してカラムを追加または破棄できることになっている。MySQL では、カラムを追加または破棄するために ALTER TABLE、CREATE INDEX、DROP INDEX ステートメントを代用する。

- 空間参照系とその ID（SRID）関連

  - Length()、Area() などの関数は平面座標系に対応している。
  - すべてのオブジェクトは現在、同じ平面座標系にあるものと考えられている。

- LineString および MultiLineString 上の OpenGIS 関数 Length() は現在、MySQL では GLength() として呼び出されなければならない。

  この関数と既存の SQL 関数 Length()（文字列値を計算）が競合するのが問題であり、この関数が文字コンテキストと空間コンテキストのどちらで呼び出されたのか区別できないことがある。何らかの方法でこの問題を解決するか、別の関数名を決定する必要がある。

第11章 MySQL API

この章では、MySQL で使用できる API の入手場所および使用方法について説明します。最も豊富な種類が用意されているのが C API です。これは MySQL チームが開発しています。他言語向けの API は、この C API をベースにして開発されています。

11.1. MySQL C API

C API コードは、MySQL とともに配布されています。C API コードは mysqlclient ライブラリに格納され、C プログラムはこのコードを使用してデータベースにアクセスできます。

MySQL のソースディストリビューションに含まれるクライアントの多くは C で記述されています。C API の使用方法がわかるようなコード例を探している場合は、これらのクライアントを参考にしてください。クライアントは、MySQL ソースディストリビューションの clients ディレクトリに格納されています。

他のクライアント API (Connector/J を除くすべての API) の多くは、mysqlclient ライブラリを使用して MySQL サーバと通信します。このライブラリを使用すると、たとえば環境変数を参照できるので、他のクライアントプログラムとほとんどの環境変数を共用できることになります。共用できる環境変数の一覧については、項4.9.「MySQL クライアントサイドのスクリプトとユーティリティ」 を参照してください。

クライアントが使用できる通信バッファサイズには上限があります。内部的に割り当てられているバッファサイズ (16 キロバイト) は、最大 16 メガバイトまで自動的に増加します。バッファサイズが増加するのはバッファの要求量が増加した場合だけであり、デフォルトの最大値を増やしたからといって使用リソースが増加するわけではありません。このサイズチェックでは、誤ったクエリや通信パケットを主にチェックします。

通信バッファサイズは、SQL ステートメント 1 つ (クライアントからサーバへのトラフィック) および結果データ 1 行 (サーバからクライアントへのトラフィック) を格納できるだけの大きさが必要です。各スレッドの通信バッファは、クエリやレコードを処理できるように、その最大値まで動的に拡大されます。たとえば、最大 16 メガバイトのデータを格納する BLOB 値を処理する場合、通信バッファサイズの最大値は (サーバとクライアントの両方で) 少なくとも 16 メガバイトである必要があります。クライアントのデフォルトの最大値は 16 メガバイトですが、サーバのデフォルトの最大値は 1 メガバイトです。サーバのデフォルトの最大値を増やすには、サーバの起動時に max_allowed_packet の値を変更します。 See 項5.5.2.「サーバパラメータのチューニング」。

MySQL サーバは、クエリが終わるたびに通信バッファサイズを net_buffer_length に減らします。クライアント側では、接続が切断されてメモリが解放されるまで、その接続に割り当てられたバッファサイズが減ることはありません。

スレッドを使用するプログラミングについては、項11.1.14.「スレッドクライアントの作成方法」 を参照してください。同一プログラム内に "サーバー" と "クライアント" が存在する (および外部の MySQL サーバと通信しない) スタンドアロンアプリケーションの作成については、項11.1.15.「組み込み MySQL サーバライブラリ libmysqld」 を参照してください。

11.1.1. C API データ型

- MYSQL

  この構造体は、1 つのデータベース接続へのハンドルを表す。ほとんどの MySQL 関数で使用される。

- MYSQL_RES

  この構造体は、レコードを返すクエリ（SELECT、SHOW、EXPLAIN、DESCRIBE）の結果を表す。このセクションでは、クエリから返された情報を結果セットと呼ぶ。

- MYSQL_ROW

  1 行のデータのタイプセーフな表現。現在は、バイト文字列の配列として実装されている（フィールドにはバイナリデータが格納される場合があり、そのようなデータでは内部的にヌルバイトが使用される可能性があるので、バイト文字列をヌル終端文字列として扱うことはできない）。レコードを取得するには、mysql_fetch_row() を呼び出す。

- MYSQL_FIELD

  この構造体には、フィールドの名前、型、サイズなど、フィールドに関する情報が格納される。メンバの詳細については、以下で説明する。 各フィールドに対して mysql_fetch_field() を繰り返し呼び出すことによって、対応する MYSQL_FIELD 構造体を取得できる。フィールド値はこの構造体の一部ではなく、MYSQL_ROW 構造体に含まれる。

- MYSQL_FIELD_OFFSET

  MySQL フィールド一覧に対するオフセットの安全な型表現（mysql_field_seek() が使用）。オフセットはレコード内部でのフィールド番号であり、0 から始まる。

- my_ulonglong

  レコードの数を表すための型であり、mysql_affected_rows()、mysql_num_rows()、および mysql_insert_id() で使用される。0 から 1.84e19 までの範囲の値を表す。

  一部のシステムでは、my_ulonglong 型の値を出力しようとしても、正常に出力されない。この型の値を出力するには、unsigned long に変換して、%lu フォーマットを使用する。以下に例を示す。

```
printf ("Number of rows: %lu\n", (unsigned long) mysql_num_rows(result));
```

MYSQL_FIELD 構造体のメンバを以下に示します。

- char * name

  フィールドの名前。ヌル終端文字列。

- char * table

  このフィールドが計算結果データのフィールドでない場合、このフィールドが属するテーブルの名前。計算結果データのフィールドの場合、table の値は空文字列になる。

- char * def

  このフィールドのデフォルト値。ヌル終端文字列。mysql_list_fields() を使用するときだけ設定される。

- enum enum_field_types type

  フィールドの型。 type の値は次のいずれかになる。

型の値	型の説明
FIELD_TYPE_TINY	TINYINT フィールド
FIELD_TYPE_SHORT	SMALLINT フィールド
FIELD_TYPE_LONG	INTEGER フィールド
FIELD_TYPE_INT24	MEDIUMINT フィールド
FIELD_TYPE_LONGLONG	BIGINT フィールド
FIELD_TYPE_DECIMAL	DECIMAL または NUMERIC フィールド
FIELD_TYPE_FLOAT	FLOAT フィールド
FIELD_TYPE_DOUBLE	DOUBLE または REAL フィールド
FIELD_TYPE_TIMESTAMP	TIMESTAMP フィールド
FIELD_TYPE_DATE	DATE フィールド
FIELD_TYPE_TIME	TIME フィールド
FIELD_TYPE_DATETIME	DATETIME フィールド
FIELD_TYPE_YEAR	YEAR フィールド
FIELD_TYPE_STRING	CHAR フィールド
FIELD_TYPE_VAR_STRING	VARCHAR フィールド
FIELD_TYPE_BLOB	BLOB または TEXT フィールド (max_length を使用して最大長を決定)
FIELD_TYPE_SET	SET フィールド
FIELD_TYPE_ENUM	ENUM フィールド
FIELD_TYPE_NULL	NULL 型フィールド
FIELD_TYPE_CHAR	廃止 (代わりに FIELD_TYPE_TINY を使用すること)

IS_NUM() マクロを使用すると、フィールドが数値型かどうかを調べることができる。type の値を IS_NUM() に渡すと、フィールドが数値型の場合は TRUE と評価される。

```
if (IS_NUM(field->type))
  printf("Field is numeric\n");
```

- unsigned int length

  フィールドの幅。テーブル定義で指定された値に従う。

- unsigned int max_length

  結果セットのフィールドの最大幅 (結果セットに実際に存在するレコードのフィールド値の最大長)。mysql_store_result() または mysql_list_fields() を使用した場合、この値はフィールドの最大長を表す。mysql_use_result() を使用した場合、この値は 0 になる。

- unsigned int flags

フィールドのさまざまな状態を表すビットフラグ。flags 値は、0 または以下のビットが 1 つ以上設定された値になる。

フラグの値	フラグの説明
NOT_NULL_FLAG	NULL を設定できないフィールド
PRI_KEY_FLAG	主キーの一部を構成するフィールド
UNIQUE_KEY_FLAG	一意なキーの一部を構成するフィールド
MULTIPLE_KEY_FLAG	一意でないキーの一部を構成するフィールド
UNSIGNED_FLAG	UNSIGNED 属性を持つフィールド
ZEROFILL_FLAG	ZEROFILL 属性を持つフィールド
BINARY_FLAG	BINARY 属性を持つフィールド
AUTO_INCREMENT_FLAG	AUTO_INCREMENT 属性を持つフィールド
ENUM_FLAG	ENUM 型のフィールド（廃止）
SET_FLAG	SET 型のフィールド（廃止）
BLOB_FLAG	BLOB 型または TEXT 型のフィールド（廃止）
TIMESTAMP_FLAG	TIMESTAMP 型のフィールド（廃止）

BLOB_FLAG、ENUM_FLAG、SET_FLAG、および TIMESTAMP_FLAG の各フラグは、フィールドの属性ではなく型を示しているので、廃止されている。代わりに、field->type を FIELD_TYPE_BLOB、FIELD_TYPE_ENUM、FIELD_TYPE_SET、または FIELD_TYPE_TIMESTAMP と比較する方が望ましい。

次に、flags 値の典型的な使用例を示す。

```
if (field->flags & NOT_NULL_FLAG)
  printf("Field can't be null\n");
```

以下のマクロを使用すると、flags 値を調べてブール値で結果を得ることができる。

フラグの状態	説明
IS_NOT_NULL(flags)	このフィールドが NOT NULL として定義されている場合は True。
IS_PRI_KEY(flags)	このフィールドが主キーの場合は True。
IS_BLOB(flags)	このフィールドが BLOB または TEXT の場合は True （廃止。代わりに field->type による比較を推奨）。

- unsigned int decimals

  数値フィールドの小数部桁数。

11.1.2. C API 関数の概要

ここでは、C API で提供されている関数について簡単に説明します。詳細については、以降のセクションで説明します。
See 項11.1.3.「C API 関数の説明」。

関数	説明
mysql_affected_rows()	最後に実行された UPDATE、DELETE、または INSERT のいずれかのクエリによって変更、削除、または挿入されたレコードの数を返す。
mysql_change_user()	オープンされた接続のユーザおよびデータベースを変更する。
mysql_character_set_name()	接続のデフォルトのキャラクタセットの名前を返す。
mysql_close()	サーバ接続を切断する。
mysql_connect()	MySQL サーバに接続する。この関数は廃止されているので、代わりに mysql_real_connect() を使用すること。
mysql_create_db()	データベースを作成する。この関数は廃止されているので、代わりに SQL コマンド CREATE DATABASE を使用すること。
mysql_data_seek()	クエリ結果セットの任意のレコード番号にシークする。
mysql_debug()	指定された文字列で DBUG_PUSH を実行する。
mysql_drop_db()	データベースを破棄する。この関数は廃止されているので、代わりに SQL コマンド DROP DATABASE を使用すること。
mysql_dump_debug_info()	デバッグ情報を口グに書き込むようにサーバに指示する。
mysql_eof()	結果セットの最後のレコードが読み込まれたかどうかを判定する。 この関数は廃止されており、代わりに mysql_errno() または mysql_error() を使用できる。
mysql_errno()	最後に呼び出された MySQL 関数のエラー番号を返す。
mysql_error()	最後に呼び出された MySQL 関数のエラーを返す。
mysql_escape_string()	文字列に含まれる特殊文字をエスケープして、SQL ステートメントで使用できるようにする。
mysql_fetch_field()	テーブルの次のフィールドの型を返す。
mysql_fetch_field_direct()	フィールド番号で指定されたテーブルフィールドの型を返す。
mysql_fetch_fields()	すべてのフィールド構造体の配列を返す。
mysql_fetch_lengths()	現在のレコードのすべてのカラムについてその長さを返す。
mysql_fetch_row()	結果セットの次のレコードを取得する。
mysql_field_seek()	指定されたカラムにカラムカーソルを移動する。
mysql_field_count()	最後に実行されたクエリの結果セットのカラム数を返す。
mysql_field_tell()	最後に実行された mysql_fetch_field() で使用されたフィールドカーソルの位置を返す。
mysql_free_result()	結果セットで使用したメモリを解放する。
mysql_get_client_info()	クライアントのバージョン情報を文字列として返す。
mysql_get_client_version()	クライアントのバージョン情報を整数として返す。
mysql_get_host_info()	接続を記述する文字列を返す。
mysql_get_server_version()	サーバのバージョン番号を整数として返す (4.1 の新機能) 。

mysql_get_proto_info()	接続に使用しているプロトコルのバージョンを返す。
mysql_get_server_info()	サーバのバージョン番号を返す。
mysql_info()	最後に実行されたクエリに関する情報を返す。
mysql_init()	MYSQL 構造体を取得または初期化する。
mysql_insert_id()	前回実行されたクエリで生成した AUTO_INCREMENT カラムの ID を返す。
mysql_kill()	指定されたスレッドを強制終了する。
mysql_list_dbs()	単純な正規表現に一致するデータベース名を返す。
mysql_list_fields()	単純な正規表現に一致するフィールド名を返す。
mysql_list_processes()	現在のサーバスレッドの一覧を返す。
mysql_list_tables()	単純な正規表現に一致するテーブル名を返す。
mysql_num_fields()	結果セットのカラム数を返す。
mysql_num_rows()	結果セットのレコード数を返す。
mysql_options()	mysql_connect() の接続オプションを設定する。
mysql_ping()	サーバへの接続が正常かどうかを確認し、必要なら再接続する。
mysql_query()	ヌル終端文字列として指定されている SQL クエリを実行する。
mysql_real_connect()	MySQL サーバに接続する。
mysql_real_escape_string()	接続の現在のキャラクタセットを考慮して、文字列に含まれる特殊文字をエスケープし、SQL ステートメントで使用できるようにする。
mysql_real_query()	バイト文字列として指定されている SQL クエリを実行する。
mysql_reload()	権限テーブルの再読み込みをサーバに指示する。
mysql_row_seek()	mysql_row_tell() から返された値をオフセットとして、結果セット内のレコードにシークする。
mysql_row_tell()	レコードカーソルの位置を返す。
mysql_select_db()	データベースを選択する。
mysql_set_server_option()	接続のオプション (multi-statements など) を設定する。
mysql_sqlstate()	最後に発生したエラーの SQLSTATE エラーコードを返す。
mysql_shutdown()	データベースサーバをシャットダウンする。
mysql_stat()	サーバステータスを文字列として返す。
mysql_store_result()	結果セット全体を取得してクライアントに転送する。
mysql_thread_id()	現在のスレッド ID を返す。
mysql_thread_safe()	クライアントがスレッドセーフとしてコンパイルされている場合は 1 を返す。
mysql_use_result()	レコード単位の結果セットの取得を開始する。
mysql_warning_count()	前回実行した SQL ステートメントの警告数を返す。
mysql_commit()	トランザクションをコミットする (4.1 の新機能)。
mysql_rollback()	トランザクションをロールバックする (4.1 の新機能)。
mysql_autocommit()	自動コミットモードのオン/オフを切り替える (4.1 の新機能)。

mysql_more_results()	まだ取得していない結果セットが存在するかどうかを調べる（4.1 の新機能）。
mysql_next_result()	マルチクエリを実行している場合、次の結果セットを取得し、返す（4.1 の新機能）。

サーバに接続するには、mysql_init() を呼び出して接続ハンドラを初期化し、次にそのハンドラ（およびホスト名、ユーザ名、パスワードなどのその他の情報）をパラメータとして mysql_real_connect() を呼び出します。接続が確立したとき、mysql_real_connect() によって reconnect フラグ（MYSQL 構造体の一部）の値が 1 に設定されます。このフラグは、接続が切断されてクエリを実行できない場合に、そこでクエリを終了せずに、サーバへの再接続を試みることを意味します。接続が必要なくなったら、mysql_close() を呼び出して接続を切断します。

接続がアクティブな場合、クライアントは mysql_query() または mysql_real_query() を使用してサーバに SQL クエリを送信できます。この 2 つの違いは、mysql_query() はクエリをヌル終端文字列として、mysql_real_query() はクエリをバイト文字列として、それぞれ受け取ります。文字列にバイナリデータが含まれる（ヌルバイトが含まれる可能性がある）場合は、mysql_real_query() を使用する必要があります。

非 SELECT クエリ（たとえば、INSERT、UPDATE、DELETE）を実行した場合、変更された（影響を受けた）レコードの数を知るには、mysql_affected_rows() を呼び出します。

SELECT クエリを実行した場合、選択されたレコードを結果セットとして取得します（注意: SHOW、DESCRIBE、および EXPLAIN のように、レコードを返すという点で SELECT に似ているステートメントがあり、これらは SELECT ステートメントと同様に扱う必要があります）。

クライアントが結果セットを処理する方法は 2 つあります。1 つは、mysql_store_result() を呼び出して結果セット全体を一括して取得する方法です。この関数は、クエリから返されたすべてのレコードをサーバから取得し、クライアントに保存します。もう 1 つは、mysql_use_result() を呼び出して、クライアントがレコード単位で結果セットの取得を開始する方法です。この関数は結果セットを取得する処理の初期化は行いますが、実際にはサーバからレコードを受け取っていません。

どちらの場合も、レコードにアクセスするには mysql_fetch_row() を呼び出します。mysql_store_result() を使用した場合、mysql_fetch_row() は、すでにサーバから取得されているレコードにアクセスします。mysql_use_result() を使用した場合、mysql_fetch_row() は実際にサーバからレコードを取得します。各レコードのデータサイズは、mysql_fetch_lengths() を呼び出すことによって知ることができます。

結果セットが必要なくなったら、mysql_free_result() を呼び出して、使用していたメモリを解放します。

2 つの取得メカニズムは、状況に応じて使い分けます。クライアントプログラムは、必要に応じて最も適切な方法を選択する必要があります。実際には、mysql_store_result() を多用する傾向があります。

mysql_store_result() の長所は、すべてのレコードがクライアントに取得済みなので、レコードに順次アクセスできるだけでなく、mysql_data_seek() または mysql_row_seek() を使用して結果セット内でのカレントレコードの位置を変更することによって、結果セット内を前後に移動できることです。結果セットに含まれるレコード数を知るには、mysql_num_rows() を呼び出します。一方で、結果セットが大きい場合の mysql_store_result() のメモリ要件は非常に厳しい場合があり、メモリ不足が発生する可能性は高くなります。

mysql_use_result() の長所は、一度に 1 行のレコードしか保持しないので、クライアントが結果セットで使用するメモリが少なくてすむことです（また、メモリ割り当てのオーバヘッドが少ないので mysql_use_result() の実行速度も向上します）。短所は、サーバを拘束しないように各レコードに対する処理を短時間で終わらせる必要があること、結果セット内のレコードに対するランダムアクセス機能がないこと（順次アクセスのみ）、すべてのレコードを取得するまで結果セッ

トに含まれるレコード数がわからないことです。さらに、探していた情報が順次アクセスの途中で見つかった場合でも、最後まですべてのレコードを取得する必要があります。

クライアントは API を使用することで、クエリが SELECT かどうかがわからなくても、クエリに対して適切に応答する（必要に応じてレコードを取得する）ことができます。そのためには、mysql_query()（または mysql_real_query()）を呼び出すたびに mysql_store_result() を呼び出します。この呼び出しが正常終了した場合、クエリは SELECT だったことになり、レコードを読み込むことができます。呼び出しが失敗した場合、mysql_field_count() を呼び出して、この結果がクエリの結果として予想されたものだったかどうかを判断します。mysql_field_count() が 0 を返した場合、クエリが返したデータはなく（INSERT、UPDATE、DELETE などのクエリだったことを意味する）、レコードを返すクエリではなかったと判断されます。mysql_field_count() が 0 以外の値を返した場合、レコードを返すはずのクエリが実行されたけれども、何も返さなかったと判断されます。これは、SELECT クエリが失敗したことを意味します。ここに示した方法のコーディング例については、mysql_field_count() の説明を参照してください。

mysql_store_result() および mysql_use_result() はどちらも、結果セットを構成するフィールドに関する情報（フィールドの数、名前、型など）を取得します。レコードのフィールド情報に順次アクセスするには、mysql_fetch_field() を繰り返し呼び出すか、またはレコードのフィールド番号を指定して mysql_fetch_field_direct() を呼び出します。現在のフィールドカーソルの位置を変更するには、mysql_field_seek() を呼び出します。フィールドカーソルの位置を変更すると、その後に呼び出された mysql_fetch_field() の動作に影響を与えます。mysql_fetch_fields() を呼び出すと、すべてのフィールドの情報を一括して取得できます。

エラーを検出し、報告する場合、mysql_errno() および mysql_error() を使用してエラー情報にアクセスします。この 2 つの関数は、最後に呼び出された、成功または失敗の可能性のある関数のエラーコードまたはエラーメッセージを返します。この結果によって、エラーが発生したタイミングとその内容を判断します。

11.1.3. C API 関数の説明

ここの説明の中で、パラメータまたは戻り値に NULL が使用されている場合、これは C 言語における NULL を表しており、MySQL における NULL 値ではありません。

値を返す関数は、一般にポインタまたは整数を返します。特に指定されていないかぎり、ポインタを返す関数が非 NULL 値を返した場合は正常終了を、NULL 値を返した場合はエラーを示します。同様に、整数を返す関数が 0 を返した場合は正常終了を、0 以外の値を返した場合はエラーを示します。注意: ``0 以外" ということに、それ以上の意味はありません。関数説明に特に記述がないかぎり、0 以外の値に対して比較テストは必要ありません。

```
if (result)               /* correct */
   ... error ...

if (result < 0)            /* incorrect */
   ... error ...

if (result == -1)          /* incorrect */
   ... error ...
```

関数が返すエラーについては、関数説明の中の「エラー」サブセクションを参照してください。発生したエラーを調べるには、mysql_errno() を呼び出します。mysql_error() を呼び出すと、エラー内容の文字列表現を取得できます。

11.1.3.1. mysql_affected_rows()

my_ulonglong mysql_affected_rows(MYSQL *mysql)

説明

最後に呼び出された UPDATE によって変更されたレコード数、最後に呼び出された DELETE によって削除されたレコード数、または最後に呼び出された INSERT によって挿入されたレコード数を返します。UPDATE、DELETE、または INSERT のいずれかのステートメントについて mysql_query() を呼び出した後に、この関数を呼び出します。SELECT ステートメントの場合、mysql_affected_rows() を呼び出すと、mysql_num_rows() と同様に動作します。

戻り値

正の整数は、影響を与えた、または取得した、レコード数を示します。0 は、UPDATE によって更新されたレコードがなかったこと、クエリの WHERE 節に一致するレコードがなかったこと、またはクエリが実行されていないことを示します。-1 は、クエリがエラーを返したこと、または SELECT クエリの場合に mysql_store_result() を呼び出す前に mysql_affected_rows() が呼び出されたことを示します。

エラー

ありません。

例

```
mysql_query(&mysql,"UPDATE products SET cost=cost*1.25 WHERE group=10");
printf("%ld products updated",(long) mysql_affected_rows(&mysql));
```

mysqld に接続する際にフラグ CLIENT_FOUND_ROWS が指定されている場合、mysql_affected_rows() は UPDATE ステートメントの WHERE 節に一致するレコード数を返します。

注意: REPLACE コマンドで既存のレコードが新しいレコードで置き換えられた場合は mysql_affected_rows() は 2 を返します。これは、レコードが 1 行挿入された後で重複したレコードが削除されたためです。

11.1.3.2. mysql_change_user()

my_bool mysql_change_user(MYSQL *mysql, const char *user, const char *password, const char *db)

説明

ユーザを変更し、db で指定されたデータベースを mysql で指定された接続のデフォルト（カレント）データベースにします。このデータベースは、その後実行されるクエリで明示的なデータベース指定子のないテーブル参照が行われる際のデフォルトになります。

この関数は、MySQL 3.23.3 で導入されました。

mysql_change_user() は、指定したユーザが認証されない場合、またはそのユーザがデータベースを使用する権限を持たない場合に、失敗します。この場合、ユーザおよびデータベースは変更されません。

デフォルトデータベースを使用しない場合、db パラメータを NULL に設定します。

MySQL 4.0.6 以降、このコマンドは、新しい接続が確立した場合は常に、任意のアクティブなトランザクションの ROLLBACK の実行、すべてのテンポラリテーブルのクローズ、ロックされたすべてのテーブルのアンロック、および状態のリセットを行います。この処理は、ユーザが変更されなかった場合も必ず行われます。

戻り値

正常終了した場合は 0。エラーが発生した場合は 0 以外。

エラー

mysql_real_connect() が返すエラーと同じ。

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

- ER_UNKNOWN_COM_ERROR

  MySQL サーバでこのコマンドが実装されていない（バージョンが古いと考えられる）。

- ER_ACCESS_DENIED_ERROR

  ユーザまたはパスワードが正しくない。

- ER_BAD_DB_ERROR

  データベースが存在しない。

- ER_DBACCESS_DENIED_ERROR

  ユーザがデータベースへのアクセス権を持っていない。

- ER_WRONG_DB_NAME

  データベース名が長すぎる。

例

```
if (mysql_change_user(&mysql, "user", "password", "new_database"))
{
  fprintf(stderr, "Failed to change user.  Error: %s\n",
          mysql_error(&mysql));
}
```

11.1.3.3. mysql_character_set_name()

const char *mysql_character_set_name(MYSQL *mysql)

説明

現在の接続のデフォルトのキャラクタセットを返します。

戻り値

デフォルトのキャラクタセット。

エラー

ありません。

11.1.3.4. mysql_close()

void mysql_close(MYSQL *mysql)

説明

すでにオープンされている接続をクローズします。mysql で示される接続ハンドルが mysql_init() または mysql_connect() によって自動的に割り当てられたハンドルだった場合は、mysql_close() で割り当てが解除されます。

戻り値

ありません。

エラー

ありません。

11.1.3.5. mysql_connect()

MYSQL *mysql_connect(MYSQL *mysql, const char *host, const char *user, const char *passwd)

説明

この関数は廃止されています。代わりに mysql_real_connect() を使用してください。

mysql_connect() は、host 上で実行している MySQL データベースエンジンへの接続を確立しようとします。mysql_connect() が正常終了しなければ、他の API 関数 (mysql_get_client_info() を除く) を実行できません。

パラメータの意味は、mysql_real_connect() での対応するパラメータと同じですが、接続パラメータに NULL を渡すことができるという違いがあります。この場合、C API は接続構造体に自動的にメモリを割り当てます。このメモリは mysql_close() を呼び出すことによって解放されます。この方法には、接続に失敗したときにエラーメッセージを取得できないという短所があります (mysql_errno() または mysql_error() から エラー情報を取得するには有効な MYSQL ポインタを渡す必要があります) 。

戻り値

mysql_real_connect() と同じです。

エラー

mysql_real_connect() と同じです。

11.1.3.6. mysql_create_db()

int mysql_create_db(MYSQL *mysql, const char *db)

説明

db パラメータで指定された名前でデータベースを作成します。

この関数は廃止されています。代わりに、mysql_query() を使用して CREATE DATABASE ステートメントを発行してください。

戻り値

データベースが正常に作成された場合は 0。エラーが発生した場合は 0 以外。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

例

```
if(mysql_create_db(&mysql, "my_database"))
{
  fprintf(stderr, "Failed to create new database.  Error: %s\n",
          mysql_error(&mysql));
}
```

11.1.3.7. mysql_data_seek()

void mysql_data_seek(MYSQL_RES *result, my_ulonglong offset)

説明

クエリ結果セットの任意のレコードにシークします。offset 値はレコード番号を表し、0 から mysql_num_rows(stmt)-1 の範囲で指定します。

mysql_data_seek() を使用するには、結果セット構造体にクエリの結果全体が格納されている必要があるので、mysql_use_result() ではなく、mysql_store_result() とともに使用する必要があります。

戻り値

ありません。

エラー

ありません。

11.1.3.8. mysql_debug()

void mysql_debug(const char *debug)

説明

指定された文字列で DBUG_PUSH を実行します。mysql_debug() は、Fred Fish デバッグライブラリを使用します。この関数を使用するには、デバッグをサポートするためのクライアントライブラリをコンパイルする必要があります。 See 項E.1.「MySQL サーバのデバッグ」。 See 項E.2.「MySQL クライアントのデバッグ」。

戻り値

ありません。

エラー

ありません。

例

次のコードを実行すると、クライアントライブラリによってクライアントマシンの /tmp/client.trace にトレースファイルが生成されます。

```
mysql_debug("d:t:O,/tmp/client.trace");
```

11.1.3.9. mysql_drop_db()

int mysql_drop_db(MYSQL *mysql, const char *db)

説明

db パラメータで指定された名前のデータベースを破棄します。

この関数は廃止されています。代わりに、mysql_query() を使用して DROP DATABASE ステートメントを発行してください。

戻り値

データベースが正常に破棄された場合は 0。エラーが発生した場合は 0 以外。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_SERVER_GONE_ERROR

MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

例

```
if(mysql_drop_db(&mysql, "my_database"))
  fprintf(stderr, "Failed to drop the database: Error: %s\n",
          mysql_error(&mysql));
```

11.1.3.10. mysql_dump_debug_info()

int mysql_dump_debug_info(MYSQL *mysql)

説明

デバッグ情報をログに書き込むようにサーバに指示します。接続ユーザが SUPER 特権を持たない場合は正常に動作しません。

戻り値

正常に動作した場合は 0。エラーが発生した場合は 0 以外。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

11.1.3.11. mysql_eof()

my_bool mysql_eof(MYSQL_RES *result)

説明

この関数は廃止されています。代わりに mysql_errno() または mysql_error() を使用できます。

mysql_eof() は、結果セットの最後のレコードが読み込まれたかどうかを判定します。

mysql_store_result() を呼び出して結果セットを取得する場合、クライアントは 1 回の動作で結果セット全体を受け取ります。この場合、mysql_fetch_row() を呼び出して戻り値が NULL であれば、すでに結果セットの最後に到達していたことがわかるので、mysql_eof() を呼び出す必要はありません。mysql_store_result() と一緒に使用する場合、mysql_eof() は常に true を返します。

一方、mysql_use_result() を使用して結果セットの取得を開始した場合は、mysql_fetch_row() を繰り返し呼び出すことによってそのレコードを 1 行ずつサーバから取得します。この手順を実行している途中で接続に異常が発生する場合があるので、mysql_fetch_row() を呼び出して戻り値が NULL であっても、それが必ずしも結果セットの最後に正常に到達したことを意味するわけではありません。この場合は、mysql_eof() を使用することによって、どういう状況なのかを判断できます。mysql_eof() は、結果セットの最後に到達していた場合は 0 以外、エラーが発生していた場合は 0 を返します。

mysql_eof() は、歴史的には標準の MySQL エラー関数である mysql_errno() および mysql_error() の前身です。これらのエラー関数は同じ情報を返すので、すでに廃止されている mysql_eof() よりも優先して使用してください (実際、mysql_eof() が返すのはブール値だけで、エラー関数はエラー発生理由を示すより多くの情報を返します)。

戻り値

エラーが発生していない場合は 0。結果セットの最後に到達していた場合は 0 以外。

エラー

ありません。

例

mysql_eof() には次のような使い方が考えられます。

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
   // do something with data
}
if(!mysql_eof(result))  // mysql_fetch_row() failed due to an error
{
   fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

ただし、標準の MySQL エラー関数を次のように使用すれば、同じ動作を実現できます。

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
   // do something with data
}
if(mysql_errno(&mysql))  // mysql_fetch_row() failed due to an error
{
```

```
  fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

11.1.3.12. mysql_errno()

unsigned int mysql_errno(MYSQL *mysql)

説明

mysql_errno() は、mysql で指定される接続について、最後に呼び出された、成功または失敗する可能性のある API 関数のエラーコードを返します。戻り値が 0 の場合、エラーは発生していません。クライアントのエラーメッセージ番号の一覧は、MySQL errmsg.h ヘッダファイルに記述されています。サーバのエラーメッセージ番号の一覧は、mysqld_error.h に記述されています。MySQL ソースディストリビューションに含まれるファイル Docs/mysqld_error.txt には、すべてのエラーメッセージおよびエラー番号の一覧が記述されています。サーバのエラーコードの一覧は、項12.1.「返されるエラー」 にも記載されています。

注意: mysql_fetch_row() などの一部の関数は、成功した場合に mysql_errno() を設定しません。

大体の目安として、サーバに情報を要求する関数は、成功した場合に mysql_errno() をリセットすると考えてください。

戻り値

最後に呼び出された mysql_xxx 関数が失敗していた場合はそのエラーコード。0 はエラーが発生しなかったことを示します。

エラー

ありません。

11.1.3.13. mysql_error()

const char *mysql_error(MYSQL *mysql)

説明

mysql_error() は、mysql で指定された接続について、最後に呼び出された API 関数が失敗した際のエラーメッセージをヌル終端文字列として返します。関数が失敗しなかった場合、mysql_error() の戻り値はその前のエラーメッセージか、エラーがまったく発生していない場合は空文字列になります。

大体の目安として、サーバに情報を要求する関数は、成功した場合に mysql_error() をリセットすると考えてください。

mysql_errno をリセットする関数では、次の 2 つの比較は同等です。

```
if(mysql_errno(&mysql))
{
  // an error occurred
}

if(mysql_error(&mysql)[0] != '\0')
{
  // an error occurred
}
```

クライアントのエラーメッセージの言語は、MySQL クライアントライブラリを再コンパイルすることによって変更できます。現在は、複数の異なる言語でエラーメッセージを返すことができます。 See 項4.7.2.「英語以外のエラーメッセージ」。

戻り値

エラーの内容を説明するヌル終端文字列。エラーが発生していない場合は空文字列。

エラー

ありません。

11.1.3.14. mysql_escape_string()

この関数の代わりに mysql_real_escape_string() を使用してください。

mysql_real_escape_string() は先頭の引数として接続ハンドラを受け取り、現在のキャラクタセットに従って文字列をエスケープしますが、その 2 点を除けば、この関数と mysql_real_escape_string() は同じです。mysql_escape_string() は接続ハンドラを引数として受け取りません。また、現在のキャラクタセットは参照しません。

11.1.3.15. mysql_fetch_field()

MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result)

説明

結果セットに含まれる 1 つのカラムの定義を MYSQL_FIELD 構造体として返します。結果セットに含まれるすべてのカラムの定義を取得するには、この関数を繰り返し呼び出します。定義を取得するカラムが残っていない場合、mysql_fetch_field() は NULL を返します。

新しく SELECT クエリを実行するたびに、mysql_fetch_field() は先頭のカラムの定義を返すようにリセットされます。また、mysql_field_seek() を呼び出した場合も、mysql_fetch_field() が返すフィールドが変わります。

mysql_query() を呼び出してテーブルに対して SELECT を実行した後 mysql_store_result() を呼び出さなかった場合、mysql_fetch_field() を呼び出して BLOB フィールドの長さを要求すると、デフォルトの BLOB 長 (8 キロバイト) が返されます (MySQL は BLOB の最大長がわからないので 8 キロバイトを返します。この値は将来設定可能な値にする必要があります)。結果セットを取得した後は、そのクエリにおけるこのカラムの最大値の長さが field->max_length に渡されます。

戻り値

現在のカラムの MYSQL_FIELD 構造体。カラムが残っていない場合は NULL。

エラー

ありません。

例

```
MYSQL_FIELD *field;

while((field = mysql_fetch_field(result)))
{
```

```
  printf("field name %s\n", field->name);
}
```

11.1.3.16. mysql_fetch_fields()

MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)

説明

結果セットのすべての MYSQL_FIELD 構造体の配列を返します。各構造体には、結果セットに含まれる 1 つのカラムのフィールド定義が格納されています。

戻り値

結果セットに含まれるすべてのカラムの MYSQL_FIELD 構造体の配列。

エラー

ありません。

例

```
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);
}
```

11.1.3.17. mysql_fetch_field_direct()

MYSQL_FIELD *mysql_fetch_field_direct(MYSQL_RES *result, unsigned int fieldnr)

説明

フィールド番号 fieldnr で指定される結果セットのカラムのフィールド定義を MYSQL_FIELD 構造体として返します。この関数を使用すれば、任意のカラムの定義を取得できます。fieldnr の値は 0 から mysql_num_fields(result)-1 の範囲で指定します。

戻り値

指定したカラムの MYSQL_FIELD 構造体。

エラー

ありません。

例

```
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);
}
```

11.1.3.18. mysql_fetch_lengths()

unsigned long *mysql_fetch_lengths(MYSQL_RES *result)

説明

結果セット内のカレントレコードのカラムの長さを返します。フィールド値をコピーしようとする場合、この関数を使用すれば strlen() を呼び出さなくても長さを取得できるので便利です。また、結果セットにバイナリデータが含まれている場合、strlen() はヌル文字が含まれるフィールドの長さを正しく返さないので、この関数を使用してデータサイズを調べる必要があります。

空のカラムおよび NULL 値を含むカラムの長さは 0 です。この 2 つを区別する方法については、mysql_fetch_row() の説明を参照してください。

戻り値

各カラムのサイズ (ヌル終端文字は数えない) を表す unsigned long 整数の配列。エラーが発生した場合は NULL。

エラー

mysql_fetch_lengths() は結果セットのカレントレコードでのみ有効です。mysql_fetch_row() を呼び出す前または結果セットのすべてのレコードを取得した後で呼び出された場合は NULL を返します。

例

```
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]);
   }
}
```

11.1.3.19. mysql_fetch_row()

MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)

説明

結果セットの次のレコードを取得します。mysql_store_result() を呼び出した後に mysql_fetch_row() を使用する場合、取得するレコードが残っていなければ NULL を返します。mysql_use_result() を呼び出した後に mysql_fetch_row() を使用する場合、取得するレコードが残っていないか、またはエラーが発生したときに NULL を返します。

レコードに含まれる値の数は mysql_num_fields(result) によって取得します。mysql_fetch_row() を呼び出してその戻り値が row に格納されている場合、値へのポインタは row[0] から row[mysql_num_fields(result)-1] のように表されます。NULL ポインタは、レコードの値が NULL 値であることを示します。

mysql_fetch_lengths() を呼び出すと、レコードのフィールド値の長さを取得できます。空のフィールドおよび NULL を含むフィールドの長さはどちらも 0 ですが、そのフィールドのポインタを調べることによってその 2 つを区別できます。ポインタが NULL の場合、フィールドの値は NULL です。それ以外の場合、フィールドは空です。

戻り値

次のレコードの MYSQL_ROW 構造体。取得するレコードが残っていないか、エラーが発生した場合は NULL。

エラー

注意: mysql_fetch_row() を 1 回呼び出してから、次に呼び出すまでの間にエラーがリセットされることはありません。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

例

```
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");
}
```

11.1.3.20. mysql_field_count()

unsigned int mysql_field_count(MYSQL *mysql)

MySQL 3.22.24 より古いバージョンを使用している場合、この関数の代わりに unsigned int mysql_num_fields(MYSQL *mysql) を使用してください。

説明

現在の接続で最後に実行されたクエリの結果セットのカラム数を返します。

通常この関数は、mysql_store_result() が NULL を返したとき（結果セットポインタが返されなかったとき）に使用します。この場合、mysql_field_count() を呼び出すことによって、mysql_store_result() が正常な処理として空の結果セットを返したのかどうかを判断できます。これによって、クライアントプログラムは、クエリが SELECT（または SELECT ライクな）ステートメントかどうかを知らなくても適切な処理を実行できます。以下の例に、この処理の実現方法を示します。

See 項11.1.12.1.「mysql_query() が正常に終了した後で mysql_store_result() が NULL を返す場合があるのはなぜか」。

戻り値

結果セットに含まれるフィールド数を表す unsigned 整数。

エラー

ありません。

例

```
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));
    }
  }
}
```

mysql_field_count(&mysql) の呼び出し部分を mysql_errno(&mysql) に置き換える方法もあります。この場合、ステートメントが SELECT だったかどうかは、mysql_field_count() が返す値から推測するのではなく、mysql_store_result() が返すエラーを直接調べて判断します。

11.1.3.21. mysql_field_seek()

MYSQL_FIELD_OFFSET mysql_field_seek(MYSQL_RES *result, MYSQL_FIELD_OFFSET offset)

説明

フィールドカーソルの位置を指定されたオフセットに設定します。次に mysql_fetch_field() を呼び出すと、そのオフセットに関連付けられたカラムのフィールド定義を返します。

レコードの先頭にシークするには、offset 値に 0 を渡します。

戻り値

シークする前のフィールドカーソルの値。

エラー

ありません。

11.1.3.22. mysql_field_tell()

MYSQL_FIELD_OFFSET mysql_field_tell(MYSQL_RES *result)

説明

最後に実行された mysql_fetch_field() で使用されたフィールドカーソルの位置を返します。この戻り値を、mysql_field_seek() に引数として渡すことができます。

戻り値

フィールドカーソルの現在のオフセット。

エラー

ありません。

11.1.3.23. mysql_free_result()

void mysql_free_result(MYSQL_RES *result)

説明

mysql_store_result()、mysql_use_result()、mysql_list_dbs() などが結果セットに割り当てたメモリを解放します。結果セットが必要なくなったら、mysql_free_result() を呼び出して、使用していたメモリを解放します。

戻り値

ありません。

エラー

ありません。

11.1.3.24. mysql_get_client_info()

char *mysql_get_client_info(void)

説明

クライアントのライブラリバージョンを表す文字列を返します。

戻り値

MySQL クライアントのライブラリバージョンを表す文字列。

エラー

ありません。

11.1.3.25. mysql_get_client_version()

unsigned long mysql_get_client_version(void)

説明

クライアントのライブラリバージョンを表す整数を返します。この値は XYYZZ という形式で表され、X はメジャーバージョン、YY はリリースレベル、ZZ はリリースレベル内のバージョン番号です。たとえば、40102 は、クライアントのライブラリバージョンが 4.1.2 であることを表します。

戻り値

MySQL クライアントのライブラリバージョンを表す整数。

エラー

ありません。

11.1.3.26. mysql_get_host_info()

char *mysql_get_host_info(MYSQL *mysql)

説明

使用している接続のタイプを表す文字列を返します。この文字列にはサーバのホスト名も含まれます。

戻り値

サーバのホスト名および接続タイプを表す文字列。

エラー

ありません。

11.1.3.27. mysql_get_proto_info()

unsigned int mysql_get_proto_info(MYSQL *mysql)

説明

現在の接続に使用しているプロトコルのバージョンを返します。

戻り値

現在の接続に使用しているプロトコルのバージョンを表す unsigned 整数。

エラー

ありません。

11.1.3.28. mysql_get_server_info()

char *mysql_get_server_info(MYSQL *mysql)

説明

サーバのバージョン番号を表す文字列を返します。

戻り値

サーバのバージョン番号を表す文字列。

エラー

ありません。

11.1.3.29. mysql_get_server_version()

unsigned long mysql_get_server_version(MYSQL *mysql)

説明

サーバのバージョン番号を整数として返します (4.1 の新機能) 。

戻り値

MySQL サーバのバージョンを次の形式で表す番号。

main_version*10000 + minor_version *100 + sub_version

たとえば、バージョンが 4.1.0 の場合、戻り値は 40100 です。

この値を使用すると、クライアントプログラムはサーバのバージョンから特定の機能が存在するかどうかを簡単に判断できます。

エラー

ありません。

11.1.3.30. mysql_info()

char *mysql_info(MYSQL *mysql)

説明

最後に実行したクエリに関する情報を表す文字列を取得します。ただし、対象となるのは、以下の一覧に含まれるステートメントだけです。最後に実行したのが他のステートメントのクエリだった場合は、mysql_info() は NULL を返します。文字列の形式は、以下に説明するように、クエリのタイプによって異なります。数字は例として示したものです。実際の文字列には実行したクエリに即した値が使用されます。

- INSERT INTO ...SELECT ...

  文字列の形式: Records: 100 Duplicates: 0 Warnings: 0

- INSERT INTO ...VALUES (...),(...),(...)...

  文字列の形式: Records: 3 Duplicates: 0 Warnings: 0

- LOAD DATA INFILE ...

  文字列の形式: Records: 1 Deleted: 0 Skipped: 0 Warnings: 0

- ALTER TABLE

  文字列の形式: Records: 3 Duplicates: 0 Warnings: 0

- UPDATE

  文字列の形式: Rows matched: 40 Changed: 40 Warnings: 0

注意: mysql_info() は、INSERT ... VALUES ステートメントが複数レコードを挿入する形式の場合に限り（複数の値リストが指定された場合に限り）、NULL 以外の値を返します。

戻り値

最後に実行したクエリに関する情報を表す文字列。クエリの情報がない場合は NULL。

エラー

ありません。

11.1.3.31. mysql_init()

MYSQL *mysql_init(MYSQL *mysql)

説明

mysql_real_connect() に適応する MYSQL オブジェクトを割り当て、または初期化します。mysql に NULL ポインタを渡した場合、関数は新しいオブジェクトにメモリを割り当て、初期化し、それを返します。それ以外の値を渡した場合は、オブジェクトを初期化し、そのアドレスを返します。mysql_init() が新しいオブジェクトにメモリを割り当てた場合、そのメモリは mysql_close() を呼び出して接続をクローズする際に解放されます。

戻り値

初期化された MYSQL* ハンドル。新しいオブジェクトに割り当てる十分なメモリがなかった場合は NULL。

エラー

メモリが不足していた場合は NULL を返します。

11.1.3.32. mysql_insert_id()

my_ulonglong mysql_insert_id(MYSQL *mysql)

説明

前回実行されたクエリで生成した AUTO_INCREMENT カラムの ID を返します。この関数は、AUTO_INCREMENT フィールドを含むテーブルに対して INSERT クエリを実行した後で使用します。

注意: mysql_insert_id() は、前回実行されたクエリで AUTO_INCREMENT 値が生成されなかった場合は、0 を返します。将来のためにその値を保存する必要がある場合は、それを生成するクエリの直後に mysql_insert_id() を呼び出す必要があります。

前回実行されたクエリがエラーを返した場合については、mysql_insert_id() の値は未定義です。

mysql_insert_id() は、AUTO_INCREMENT 値を生成するか、またはカラム値を LAST_INSERT_ID(expr) に設定する INSERT ステートメントおよび UPDATE ステートメントの実行後に更新されます。 See 項6.3.6.2.「その他の各種関数」。

注意: SQL LAST_INSERT_ID() 関数の値には、必ず最後に生成された AUTO_INCREMENT 値が含まれます。この値はサーバに保持されるので、あるクエリを実行してから次のクエリを実行するまでの間にリセットされることはありません。

戻り値

前回実行されたクエリによって更新された AUTO_INCREMENT フィールドの値。現在の接続でまだ 1 回もクエリが実行されていなかった場合、または前回実行されたクエリで AUTO_INCREMENT 値が更新されなかった場合、0 を返します。

エラー

ありません。

11.1.3.33. mysql_kill()

int mysql_kill(MYSQL *mysql, unsigned long pid)

説明

pid で指定するスレッドを強制終了するように、サーバに要求します。

戻り値

正常終了した場合は 0。エラーが発生した場合は 0 以外。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

11.1.3.34. mysql_list_dbs()

MYSQL_RES *mysql_list_dbs(MYSQL *mysql, const char *wild)

説明

サーバ上のデータベースから wild パラメータで指定される単純な正規表現に一致するデータベース名を検索し、結果セットとして返します。wild にはワイルドカード文字として '%' または '_' を使用できます。NULL ポインタを指定した場合はすべてのデータベース名が一致します。mysql_list_dbs() を呼び出すと、クエリ SHOW databases [LIKE wild] を実行した場合と同じ結果が得られます。

結果セットに割り当てられたメモリを解放するには、mysql_free_result() を呼び出す必要があります。

戻り値

正常終了した場合は MYSQL_RES 結果セット。エラーが発生した場合は NULL。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_OUT_OF_MEMORY

  メモリが不足していた。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

不明なエラーが発生した。

11.1.3.35. mysql_list_fields()

MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const char *wild)

説明

指定されたテーブルのフィールドから wild パラメータで指定される単純な正規表現に一致するフィールド名を検索し、結果セットとして返します。wild にはワイルドカード文字として '%' または '_' を使用できます。NULL ポインタを指定した場合はすべてのフィールド名が一致します。mysql_list_fields() を呼び出すと、クエリ SHOW COLUMNS FROM tbl_name [LIKE wild] を実行した場合と同じ結果が得られます。

注意: mysql_list_fields() の代わりに SHOW COLUMNS FROM tbl_name を使用することを推奨します。

結果セットに割り当てられたメモリを解放するには、mysql_free_result() を呼び出す必要があります。

戻り値

正常終了した場合は MYSQL_RES 結果セット。エラーが発生した場合は NULL。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

11.1.3.36. mysql_list_processes()

MYSQL_RES *mysql_list_processes(MYSQL *mysql)

説明

現在のサーバスレッドの一覧を表す結果セットを返します。これは、mysqladmin processlist または SHOW PROCESSLIST クエリが返す情報と同じタイプの情報です。

結果セットに割り当てられたメモリを解放するには、mysql_free_result() を呼び出す必要があります。

戻り値

正常終了した場合は MYSQL_RES 結果セット。エラーが発生した場合は NULL。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

11.1.3.37. mysql_list_tables()

MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)

説明

現在のデータベースから wild パラメータで指定される単純な正規表現に一致するテーブル名を検索し、結果セットとして返します。wild にはワイルドカード文字として '%' または '_' を使用できます。NULL ポインタを指定した場合はすべてのテーブル名が一致します。mysql_list_tables() を呼び出すと、クエリ SHOW tables [LIKE wild] を実行した場合と同じ結果が得られます。

結果セットに割り当てられたメモリを解放するには、mysql_free_result() を呼び出す必要があります。

戻り値

正常終了した場合は MYSQL_RES 結果セット。エラーが発生した場合は NULL。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

不明なエラーが発生した。

11.1.3.38. mysql_num_fields()

unsigned int mysql_num_fields(MYSQL_RES *result)

または

unsigned int mysql_num_fields(MYSQL *mysql)

2 番目の形式は、MySQL 3.22.24 以降では動作しません。MYSQL* 引数を渡すには、この関数の代わりに unsigned int mysql_field_count(MYSQL *mysql) を使用する必要があります。

説明

結果セットのカラム数を返します。

注意: 結果セットへのポインタまたは接続ハンドルへのポインタのどちらを使用しても、カラム数を取得できます。mysql_store_result() または mysql_use_result() が NULL を返した場合（結果セットポインタが返されなかった場合）は、接続ハンドルを使用します。この場合、mysql_field_count() を呼び出すことによって、mysql_store_result() が正常な処理として空の結果セットを返したのかどうかを判断できます。これによって、クライアントプログラムは、クエリが SELECT（または SELECT ライクな）ステートメントかどうかを知らなくても適切な処理を実行できます。以下の例に、この処理の実現方法を示します。

See 項11.1.12.1.「mysql_query() が正常に終了した後で mysql_store_result() が NULL を返す場合があるのはなぜか」。

戻り値

結果セットに含まれるフィールド数を表す unsigned 整数。

エラー

ありません。

例

```
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_errno(&mysql))
    {
      fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
    }
    else if (mysql_field_count(&mysql) == 0)
    {
      // query does not return data
      // (it was not a SELECT)
      num_rows = mysql_affected_rows(&mysql);
    }
  }
}
```

（クエリが結果セットを返すことがわかっている場合）mysql_errno(&mysql) を呼び出す代わりに mysql_field_count(&mysql) が 0 を返すかどうかを調べる方法もあります。これは、0 を返すのは、何らかの異常が発生した場合だけだからです。

11.1.3.39. mysql_num_rows()

my_ulonglong mysql_num_rows(MYSQL_RES *result)

説明

結果セットのレコード数を返します。

mysql_num_rows() の使い方は、mysql_store_result() と mysql_use_result() のどちらを使用して結果セットを取得するかによって決まります。mysql_store_result() を使用する場合、すぐに mysql_num_rows() を呼び出すことができます。mysql_use_result() を使用する場合、結果セットに含まれるすべてのレコードを取得するまでは、mysql_num_rows() は正しい値を返しません。

戻り値

結果セットのレコード数。

エラー

ありません。

11.1.3.40. mysql_options()

int mysql_options(MYSQL *mysql, enum mysql_option option, const char *arg)

説明

この関数を使用して、接続オプションを追加設定し、接続の動作を変えることができます。この関数を複数回呼び出すことで、複数のオプションを設定できます。

mysql_options() は、mysql_init() を実行してから mysql_connect() または mysql_real_connect() を呼び出すまでの間に呼び出す必要があります。

option 引数は、設定するオプションです。arg 引数はそのオプションに設定する値です。オプションが整数の場合、arg には整数値へのポインタを渡す必要があります。

設定可能なオプションの値を次に示します。

オプション	引数の型	機能
MYSQL_OPT_CONNECT_TIMEOUT	unsigned int *	接続のタイムアウト時間（秒単位）。
MYSQL_OPT_READ_TIMEOUT	unsigned int *	サーバからの読み込みタイムアウト時間（現在は Windows の TCP/IP 接続でのみ有効）。
MYSQL_OPT_WRITE_TIMEOUT	unsigned int *	サーバへの書き込みタイムアウト時間（現在は Windows の TCP/IP 接続でのみ有効）。
MYSQL_OPT_COMPRESS	未使用	圧縮されたクライアント/サーバプロトコルを使用。
MYSQL_OPT_LOCAL_INFILE	unsigned int への省略可能なポインタ	ポインタが指定されないか、または 0 ではない unsigned int へのポインタが指定された場合、コマンド LOAD LOCAL INFILE は有効。
MYSQL_OPT_NAMED_PIPE	未使用	名前付きパイプを使用して NT 上で動作する MySQL サーバに接続。
MYSQL_INIT_COMMAND	char *	MySQL サーバに接続するときに実行するコマンド。再接続時には自動的に再実行される。
MYSQL_READ_DEFAULT_FILE	char *	my.cnf ではなく、指定されたオプション設定ファイルからオプションを読み込む。
MYSQL_READ_DEFAULT_GROUP	char *	my.cnf の指定されたグループまたは MYSQL_READ_DEFAULT_FILE で指定されたファイルからオプションを読み込む。
MYSQL_OPT_PROTOCOL	unsigned int *	使用するプロトコルのタイプ。mysql.h で定義されている mysql_protocol_type の列挙値のいずれか 1 つであること。
MYSQL_SHARED_MEMORY_BASE_NAME	char*	サーバとの通信に使用する名前付き共有メモリオブジェクト。接続相手の mysqld サーバが使用するオプション --shared-memory-base-name と同じ値を指定する必要がある。

注意: MYSQL_READ_DEFAULT_FILE または MYSQL_READ_DEFAULT_GROUP を使用する場合は、グループ client が常に読み込まれます。

オプション設定ファイルの指定されたグループには以下のオプションを指定可能です。

オプション	説明
connect-timeout	接続のタイムアウト時間（秒単位）。Linux では、サーバから最初の応答を待つ場合にもこのタイムアウト時間を使用する。
compress	圧縮されたクライアント/サーバプロトコルを使用。

database	接続コマンドでデータベースが指定されなかった場合に接続するデータベース。
debug	デバッグオプション。
disable-local-infile	LOAD DATA LOCAL の使用を無効にする。
host	デフォルトのホスト名。
init-command	MySQL サーバに接続するときに実行するコマンド。再接続時には自動的に再実行される。
interactive-timeout	mysql_real_connect() で CLIENT_INTERACTIVE を指定するのと同じ。 See 項 11.1.3.43. 「mysql_real_connect()」。
local-infile[=(0|1)]	引数が指定されないか、または 0 以外の引数が指定された場合に LOAD DATA LOCAL の使用を有効にする。
max_allowed_packet	クライアントがサーバから読み込み可能なパケットの最大サイズ。
password	デフォルトのパスワード。
pipe	名前付きパイプを使用して NT 上で動作する MySQL サーバに接続。
protocol=(TCP | SOCKET | PIPE | MEMORY)	サーバに接続するときに使用するプロトコルを選択（4.1 の新機能）。
port	デフォルトのポート番号。
return-found-rows	UPDATE を使用したときに更新されたレコードではなく、検索結果のレコードを返すように、mysql_info() に指示する。
shared-memory-base-name=name	サーバとの接続に使用する共有メモリの名前（デフォルトは "MySQL"）。MySQL 4.1 の新機能。
socket	デフォルトのソケット番号。
user	デフォルトのユーザ。

注意: timeout の代わりに connect-timeout が提供されていますが、timeout もしばらくは使用できます。

オプション設定ファイルの詳細については、項4.1.2. 「my.cnf オプション設定ファイル」 を参照してください。

戻り値

正常終了した場合は 0。不明なオプションを使用した場合は 0 以外。

例

```
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));
}
```

この例では、圧縮されたクライアント/サーバプロトコルを使用すること、および my.cnf ファイルの odbc セクションのオプションを追加で読み込むことを、クライアントに要求しています。

11.1.3.41. mysql_ping()

int mysql_ping(MYSQL *mysql)

説明

サーバへの接続が正常かどうかを確認します。切断されていた場合は、自動的に再接続を試みます。

長時間にわたってアイドル状態だったクライアントはこの関数を使用することによって、サーバによって接続が切断されていないかどうかを調べて、必要なら再接続することができます。

戻り値

サーバとの接続が正常な場合は 0。エラーが発生した場合は 0 以外。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

11.1.3.42. mysql_query()

int mysql_query(MYSQL *mysql, const char *query)

説明

ヌル終端文字列 query で指定された SQL クエリを実行します。このクエリは単一 SQL ステートメントで構成されている必要があります。このステートメントには、その終わりを示すセミコロン (';') または \g を追加する必要はありません。

mysql_query() はバイナリデータを含むクエリを扱うことができません。そのようなクエリでは代わりに mysql_real_query() を使用する必要があります (バイナリデータには '\0' が含まれる可能性があり、mysql_query() はそれをクエリ文字列の終わりとして解釈してしまうため) 。

クエリが結果セットを返すタイプかどうかを調べるには、mysql_field_count() を使用します。 See 項11.1.3.20.「mysql_field_count()」。

戻り値

クエリが正常に動作した場合は 0。エラーが発生した場合は 0 以外。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

11.1.3.43. 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 long client_flag)

説明

mysql_real_connect() は、host で動作している MySQL データベースエンジンへの接続を確立しようとします。mysql_real_connect() が正常終了しなければ、他の API 関数 (mysql_get_client_info() を除く) を実行できません。

パラメータの指定方法を次に示します。

- 1 つ目のパラメータには、既存の MYSQL 構造体のアドレスを指定する。mysql_real_connect() を呼び出す前に、mysql_init() を呼び出して MYSQL 構造体を初期化する必要がある。mysql_options() を使用すると、さまざまな接続オプションを変更できる。 See 項11.1.3.40.「mysql_options()」。

- host パラメータには、ホスト名または IP アドレスのどちらかを指定する。host に NULL または文字列 "localhost" を指定した場合、ローカルホストに接続するものとみなされる。OS がソケット (Unix) または名前付きパイプ (Windows) をサポートしている場合、TCP/IP の代わりにそれらを使用してサーバに接続する。

- user パラメータにはユーザの MySQL ログイン ID を指定する。user に NULL または空文字列 "" を指定した場合、現在のユーザを使用するものとみなされる。Unix では現在のログイン名が使用される。Windows で ODBC を使用する場合は、現在のユーザ名を明示的に指定する必要がある。 See 項11.2.2.「ODBC アドミニストレータのフィールドの設定方法」。

- passwd パラメータには user のパスワードを指定する。passwd に NULL を指定した場合、user テーブルのエントリのパスワードフィールドに何も指定されていない (空である) ユーザだけがチェック対象となる。データベース管理者は、この機能を使用して、パスワードを指定しているかどうかによってユーザに異なる特権を与えるという MySQL 特権システムをセットアップできる。

  注意:パスワードの暗号化はクライアント API によって自動的に処理されるので、mysql_real_connect() を呼び出す前にパスワードの暗号化は不要である。

- db パラメータにはデータベース名を指定する。db が NULL ではない場合、この値が接続のデフォルトデータベースとして設定される。
- port パラメータに 0 以外の値を指定した場合、この値が TCP/IP 接続のポート番号として使用される。注意: 接続のタイプは host パラメータによって決定される。
- unix_socket パラメータに NULL 以外の値を指定した場合、この文字列がソケットまたは名前付きパイプに使用される。注意: 接続のタイプは host パラメータによって決定される。
- client_flag パラメータには通常 0 を指定するが、極めて特殊な状況では以下に示すフラグを組み合わせて指定できる。

フラグ名	フラグの説明
CLIENT_COMPRESS	圧縮プロトコルを使用。
CLIENT_FOUND_ROWS	影響を受けたレコードの数ではなく、検索結果の（一致した）レコードの数を返す。
CLIENT_IGNORE_SPACE	関数名の末尾の空白を無視する。すべての関数名を予約語とする。
CLIENT_INTERACTIVE	アイドル状態が（wait_timeout 秒ではなく）interactive_timeout 秒継続したら接続をクローズする。
CLIENT_LOCAL_FILES	LOAD DATA LOCAL の処理を有効にする。
CLIENT_MULTI_STATEMENTS	クライアントが複数行クエリ（';' をステートメントの区切りとする）を送信する可能性があることをサーバに通知する。このフラグが設定されていない場合、複数行クエリは無効。MySQL 4.1 の新機能。
CLIENT_MULTI_RESULTS	クライアントが複数クエリまたはストアドプロシージャによって取得した複数の結果セットを処理できることをサーバに通知する。 CLIENT_MULTI_STATEMENTS が設定されている場合、このフラグは自動的に設定される。MySQL 4.1 の新機能。
CLIENT_NO_SCHEMA	db_name.tbl_name.col_name という構文の使用を禁止する。これは ODCB 向けの構文である。この構文を使用した場合、パーサでエラーが発生する。この構文は一部の ODBC プログラムでバグをトラップするために使用する。
CLIENT_ODBC	クライアントが ODBC 経由であることを通知する。この場合、mysqld は ODBC クライアントに適した処理を行う。
CLIENT_SSL	SSL（暗号化プロトコル）を使用する。アプリケーションプログラムではこのフラグを設定しないこと。これはクライアントライブラリが内部で設定するフラグである。

戻り値

接続が正常に確立した場合は MYSQL* 接続ハンドル。接続が確立しなかった場合は NULL。接続が確立した場合、戻り値は 1 つ目のパラメータの値と同じです。

エラー

- CR_CONN_HOST_ERROR

MySQL サーバへの接続が確立しなかった。

- CR_CONNECTION_ERROR

  ローカルの MySQL サーバへの接続が確立しなかった。

- CR_IPSOCK_ERROR

  IP ソケットの作成に失敗した。

- CR_OUT_OF_MEMORY

  メモリが不足していた。

- CR_SOCKET_CREATE_ERROR

  Unix ソケットの作成に失敗した。

- CR_UNKNOWN_HOST

  ホスト名の IP アドレスが見つからなかった。

- CR_VERSION_ERROR

  サーバに接続する際に、異なるプロトコルバージョンのクライアントライブラリを使用したために、プロトコルの不一致が発生した。このエラーは、--old-protocol オプションを指定しないで起動した新しいバージョンのサーバに、非常に古いバージョンのクライアントライブラリを使用して接続を試みた場合に発生する可能性がある。

- CR_NAMEDPIPEOPEN_ERROR

  Windows で名前付きパイプの作成に失敗した。

- CR_NAMEDPIPEWAIT_ERROR

  Windows で名前付きパイプの待機に失敗した。

- CR_NAMEDPIPESETSTATE_ERROR

  Windows でパイプハンドラの取得に失敗した。

- CR_SERVER_LOST

  connect_timeout が正の値であり、connect_timeout 秒経過してもサーバに接続できなかった。または init-command の実行中にサーバがダウンした。

例

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"your_prog_name");
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));
}
```

この例では mysql_options() を呼び出して MySQL ライブラリが my.cnf ファイルの [client] および [your_prog_name] セクションを読み込むように設定しています。このように設定することで、非標準的な方法でセットアップされた MySQL に対してもこのプログラムが正常に動作することを保証します。

注意: 接続が確立したとき、mysql_real_connect() によって reconnect フラグ (MYSQL 構造体の一部) の値が 1 に設定されます。このフラグは、接続が切断されてクエリを実行できない場合に、そこでクエリを終了せずに、サーバへの再接続を試みることを意味します。

11.1.3.44. mysql_real_escape_string()

unsigned long mysql_real_escape_string(MYSQL *mysql, char *to, const char *from, unsigned long length)

説明

この関数は、SQL ステートメントで使用できる正当な SQL 文字列を作成するために使用します。 See 項6.1.1.1.「文字列」。

from に指定した文字列が、エスケープされた SQL 文字列にエンコードされます。この際、接続の現在のキャラクタセットが考慮されます。エンコードした結果は to に設定され、末尾には文字列の終わりを示すヌルバイトが付加されます。エンコードされる文字は、NUL (ASCII 0)、'\n'、'\r'、'\'、'''、'"'、および Ctrl-Z キー (see 項6.1.1.「リテラル:文字列と数値の記述方法」) です (厳密に言えば、MySQL でエスケープする必要があるのはクエリ内で文字列を引用するためのバックスラッシュおよび引用符だけですが、この関数ではログファイルを読みやすくするために他の文字もエスケープします)。

from で指定する文字列の長さは length バイトである必要があります。to バッファには、少なくとも length*2+1 バイトのメモリを割り当てる必要があります (最悪の場合、1 文字エンコードするのに 2 バイトが必要な可能性があり、さらに文字列の終わりを示すヌルバイトを格納する必要があります)。mysql_real_escape_string() が復帰したとき、to にはヌル終端文字列が格納されています。戻り値はエンコード後の文字列の長さです。ただし、文字列の終わりを示すヌルバイトは含みません。

例

```
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));
}
```

この例で使用されている strmov() 関数は、mysqlclient ライブラリに格納されています。strcpy() と同様の処理を行いますが、1 つ目のパラメータで指定された文字列の終わりを示すヌルバイトへのポインタを返します。

戻り値

to に設定された値の長さ。ただし、文字列の終わりを示すヌルバイトは含みません。

エラー

ありません。

11.1.3.45. mysql_real_query()

int mysql_real_query(MYSQL *mysql, const char *query, unsigned long length)

説明

query で指定した文字列で表される SQL クエリを実行します。この文字列の長さは length バイトである必要があります。このクエリは単一 SQL ステートメントで構成されている必要があります。このステートメントには、その終わりを示すセミコロン (';') または \g を追加する必要はありません。

バイナリデータには '\0' 文字が含まれる可能性があるので、そのようなデータを含むクエリの場合は、mysql_query() ではなく、mysql_real_query() を使用する必要があります。また、mysql_real_query() はクエリ文字列に対して strlen() を呼び出さないので、mysql_query() よりも高速に動作します。

クエリが結果セットを返すタイプかどうかを調べるには、mysql_field_count() を使用します。 See 項11.1.3.20.「mysql_field_count()」。

戻り値

クエリが正常に動作した場合は 0。エラーが発生した場合は 0 以外。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

11.1.3.46. mysql_reload()

int mysql_reload(MYSQL *mysql)

説明

権限テーブルの再読み込みを MySQL サーバに要求します。接続ユーザには RELOAD 特権が必要です。

この関数は廃止されています。代わりに、mysql_query() を使用して FLUSH PRIVILEGES ステートメントを発行してください。

戻り値

正常終了した場合は 0。エラーが発生した場合は 0 以外。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

11.1.3.47. mysql_row_seek()

MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET offset)

説明

クエリ結果セットの任意のレコードにローカーソルを設定します。offset の値はレコードオフセットであり、mysql_row_tell() または mysql_row_seek() の戻り値になります。この値はレコード番号ではありません。結果セット内のレコードに番号を指定してシークする場合は、この関数ではなく、mysql_data_seek() を使用します。

mysql_row_seek() を使用するには、結果セット構造体にクエリの結果全体が格納されている必要があるので、mysql_use_result() ではなく、mysql_store_result() とともに使用する必要があります。

戻り値

シークする前のレコードカーソルの値。mysql_row_seek() を次に呼び出すときにこの値を渡すことができます。

エラー

ありません。

11.1.3.48. mysql_row_tell()

MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result)

説明

最後に実行された mysql_fetch_row() で使用されたレコードカーソルの現在の位置を返します。この戻り値を、mysql_row_seek() に引数として渡すことができます。

mysql_row_tell() は、mysql_use_result() の後ではなく、mysql_store_result() の後でのみ使用できます。

戻り値

レコードカーソルの現在のオフセット。

エラー

ありません。

11.1.3.49. mysql_select_db()

int mysql_select_db(MYSQL *mysql, const char *db)

説明

db で指定されたデータベースを mysql で指定された接続のデフォルト（カレント）データベースにします。このデータベースは、その後実行されるクエリで明示的なデータベース指定子のないテーブル参照が行われる際のデフォルトになります。

接続ユーザがデータベースを使用する権限を持つユーザとして認証されない場合、mysql_select_db() は失敗します。

戻り値

正常終了した場合は 0。エラーが発生した場合は 0 以外。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

11.1.3.50. mysql_set_server_option()

int mysql_set_server_option(MYSQL *mysql, enum enum_mysql_set_option option)

説明

接続のオプションを有効または無効にします。option には以下のどちらかの値を指定できます。

MYSQL_OPTION_MULTI_STATEMENTS_ON	複数ステートメントのサポートを有効にする。
MYSQL_OPTION_MULTI_STATEMENTS_OFF	複数ステートメントのサポートを無効にする。

戻り値

正常終了した場合は 0。エラーが発生した場合は 0 以外。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- ER_UNKNOWN_COM_ERROR

  サーバが mysql_set_server_option() をサポートしていなかった (サーバが 4.1.1 より古いバージョンの場合) か、またはサーバが設定しようとしたオプションをサポートしていなかった。

11.1.3.51. mysql_shutdown()

int mysql_shutdown(MYSQL *mysql)

説明

データベースサーバにシャットダウンするように要求します。接続ユーザには SHUTDOWN 特権が必要です。

戻り値

正常終了した場合は 0。エラーが発生した場合は 0 以外。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

11.1.3.52. mysql_sqlstate()

const char *mysql_sqlstate(MYSQL *mysql)

説明

最後に発生したエラーの SQLSTATE エラーコードを含むヌル終端文字列を返します。エラーコードは 5 文字で示されます。'00000' は、``エラーがない'' ことを意味します。値は ANSI SQL および ODBC によって規定されています。考えられる値の一覧については、項12.1.「返されるエラー」 を参照してください。

注意: すべての MySQL エラーが SQLSTATE にマッピングされているわけではありません。マッピングされていないエラーの場合は 'HY000' (一般エラー) が使用されます。

この関数は MySQL 4.1.1 で追加されました。

戻り値

SQLSTATE エラーコードを含むヌル終端文字列。

関連項目

See 項11.1.3.12.「mysql_errno()」。 See 項11.1.3.13.「mysql_error()」。 See 項11.1.7.18.「mysql_stmt_sqlstate()」。

11.1.3.53. mysql_ssl_set()

int mysql_ssl_set(MYSQL *mysql, const char *key, const char *cert, const char *ca, const char *capath, const char *cipher)

説明

mysql_ssl_set() は、SSL を使用して、セキュリティで保護された接続を確立するために使用します。mysql_real_connect() の前に呼び出す必要があります。

mysql_ssl_set() は、クライアントライブラリで OpenSSL サポートが有効になっていない場合は何もしません。

mysql は、mysql_init() から返された接続ハンドラです。他のパラメータの指定方法を次に示します。

- key パラメータには、秘密キーファイルへのパス名を指定する。
- cert パラメータには、証明書ファイルへのパス名を指定する。
- ca パラメータには、認証局発行の証明書ファイルへのパス名を指定する。
- capath パラメータには、PEM 形式の信頼された CA 証明書が格納されたディレクトリへのパス名を指定する。
- cipher パラメータには、SSL 暗号化に使用できる暗号の一覧を指定する。

使用しない SSL パラメータには NULL を指定できます。

戻り値

この関数は常に 0 を返します。SSL が正しくセットアップされていない場合、接続を試みると mysql_real_connect() がエラーを返します。

11.1.3.54. mysql_stat()

char *mysql_stat(MYSQL *mysql)

説明

mysqladmin status コマンドの出力と同じ情報を含む文字列を返します。この情報には、使用可能時間（秒単位）、実行中のスレッド数、質問の数、再ロード回数、およびオープンされているテーブルの数が含まれます。

戻り値

サーバステータスを表す文字列。エラーが発生した場合は NULL。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。
- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。
- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。
- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

11.1.3.55. mysql_store_result()

MYSQL_RES *mysql_store_result(MYSQL *mysql)

説明

正常にデータを取得したクエリ (SELECT、SHOW、DESCRIBE、EXPLAIN) については、mysql_store_result() または mysql_use_result() を呼び出す必要があります。

他のクエリでは mysql_store_result() または mysql_use_result() を呼び出す必要はありませんが、すべてのケースで mysql_store_result() を呼び出しても、異常が発生したり、目立った動作が行われるようなことはありません。mysql_store_result() が 0 を返すかどうかを調べることで、クエリで結果セットが生成されたかどうかを検出できます。これについては後述します。

クエリが結果セットを返すタイプかどうかを調べるには、mysql_field_count() を使用します。 See 項11.1.3.20.「mysql_field_count()」。

mysql_store_result() は、クエリの結果セット全体をクライアントに読み込み、MYSQL_RES 構造体にメモリを割り当て、この構造体に結果セットを格納します。

クエリが結果セットを返さない場合 (たとえば、クエリが INSERT ステートメントだった場合) 、mysql_store_result() はヌルポインタを返します。

また、結果セットの読み込みに失敗した場合も、mysql_store_result() はヌルポインタを返します。エラーが発生したかどうかを調べるには、mysql_error() がヌルポインタを返さないかどうか、mysql_errno() が 0 以外の値を返すかどうか、または mysql_field_count() が 0 以外の値を返すかどうかを調べます。

返すレコードが存在しない場合は空の結果セットが返されます (空の結果セットは、戻り値としてのヌルポインタとは異なります) 。

mysql_store_result() を呼び出して、戻り値がヌルポインタ以外だった場合、mysql_num_rows() を呼び出すと結果セットに含まれるレコード数を取得できます。

mysql_fetch_row() を呼び出すと、結果セットからレコードを取得できます。mysql_row_seek() および mysql_row_tell() を呼び出すと、結果セット内のカレントレコードの位置を設定および取得できます。

結果セットが必要なくなったら、mysql_free_result() を呼び出す必要があります。

See 項11.1.12.1.「mysql_query() が正常に終了した後で mysql_store_result() が NULL を返す場合があるのはなぜか」。

戻り値

結果セットが格納された MYSQL_RES 構造体。エラーが発生した場合は NULL。

エラー

mysql_store_result() は、正常終了した場合に mysql_error および mysql_errno をリセットします。

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_OUT_OF_MEMORY

  メモリが不足していた。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

11.1.3.56. mysql_thread_id()

unsigned long mysql_thread_id(MYSQL *mysql)

説明

現在の接続のスレッド ID を返します。この戻り値を、mysql_kill() に引数として渡してスレッドを強制終了することができます。

現在の接続が切断され、mysql_ping() を使用して再接続した場合、スレッド ID は別の値になります。これは、取得したスレッド ID を保管して後で使用することはできないことを意味します。スレッド ID が必要になったときに、取得する必要があります。

戻り値

現在の接続のスレッド ID。

エラー

ありません。

11.1.3.57. mysql_use_result()

MYSQL_RES *mysql_use_result(MYSQL *mysql)

説明

正常にデータを取得したクエリ (SELECT、SHOW、DESCRIBE、EXPLAIN) については、mysql_store_result() または mysql_use_result() を呼び出す必要があります。

mysql_use_result() は、結果セットの取得を開始しますが、mysql_store_result() のように実際に結果セットをクライアントに読み込むわけではありません。代わりに、mysql_fetch_row() を呼び出して、1 行ずつ個別に取得する必要があります。その場合、クエリの結果は直接サーバから読み込まれ、テンポラリテーブルやローカルバッファには格納されません。これは mysql_store_result() を使用する場合よりも多少高速であり、使用するメモリも少なくてすみます。クライアントは、カレントレコードおよび max_allowed_packet バイトまで増える可能性のある通信バッファにメモリを割り当てるだけです。

一方、それぞれのレコードに対するクライアント側での処理量が多い場合、またはユーザが ^S を入力する (スクロールを停止する) 可能性がある画面にデータを出力する場合は、mysql_use_result() の使用は避ける必要があります。そのような処理の場合、サーバの動作が停止して、読み取られているデータが格納されているテーブルを他のスレッドが更新でき

なくなります。

mysql_use_result() を使用する場合、NULL が返されるまで mysql_fetch_row() を繰り返し呼び出す必要があります。そうしないと、取得されなかったレコードが、次に実行するクエリの結果セットの一部として返されます。取得されなかったレコードが残っている場合、C API はエラー Commands out of sync; you can't run this command now を返します。

mysql_use_result() が返した結果セットに対して mysql_data_seek()、mysql_row_seek()、mysql_row_tell()、mysql_num_rows()、または mysql_affected_rows() を使用することはできません。また、mysql_use_result() が完了するまでは他のクエリを発行できません (ただし、すべてのレコードを取得した後であれば mysql_num_rows() は取得したレコードの数を正確に返します)。

結果セットが必要なくなったら、mysql_free_result() を呼び出す必要があります。

戻り値

MYSQL_RES 構造体。エラーが発生した場合は NULL。

エラー

mysql_use_result() は、正常終了した場合に mysql_error および mysql_errno をリセットします。

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_OUT_OF_MEMORY

  メモリが不足していた。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

11.1.3.58. mysql_warning_count()

unsigned int mysql_warning_count(MYSQL *mysql)

説明

前回実行された SQL ステートメントの実行中に発生した警告数を返します。この関数は、MySQL 4.1 で追加されました。

戻り値

警告数。

エラー

ありません。

11.1.3.59. mysql_commit()

my_bool mysql_commit(MYSQL *mysql)

説明

現在のトランザクションをコミットします。この関数は、MySQL 4.1 で追加されました。

戻り値

正常に動作した場合は 0。エラーが発生した場合は 0 以外。

エラー

ありません。

11.1.3.60. mysql_rollback()

my_bool mysql_rollback(MYSQL *mysql)

説明

現在のトランザクションをロールバックします。この関数は、MySQL 4.1 で追加されました。

戻り値

正常に動作した場合は 0。エラーが発生した場合は 0 以外。

エラー

ありません。

11.1.3.61. mysql_autocommit()

my_bool mysql_autocommit(MYSQL *mysql, my_bool mode)

説明

mode が 1 の場合は自動コミットモードを有効に、mode が 0 の場合は無効にします。この関数は、MySQL 4.1 で追加されました。

戻り値

正常に動作した場合は 0。エラーが発生した場合は 0 以外。

エラー

ありません。

11.1.3.62. mysql_more_results()

my_bool mysql_more_results(MYSQL *mysql)

説明

現在実行しているクエリの他にまだ取得していない結果セットが存在し、アプリケーションが mysql_next_result() を呼び出して取得する必要がある場合は、true を返します。この関数は、MySQL 4.1 で追加されました。

戻り値

取得されていない結果セットが存在する場合は TRUE（1）。取得されていない結果セットが存在しない場合は FALSE（0）。

注意: ほとんどの場合、この関数の代わりに mysql_next_result() を呼び出して取得されていない結果セットがあるかどうかを調べて、存在する場合は次の結果セットを開始します。

See 項11.1.8.「C API における複数クエリの実行の取り扱い」。 See 項11.1.3.63.「mysql_next_result()」。

エラー

ありません。

11.1.3.63. mysql_next_result()

int mysql_next_result(MYSQL *mysql)

説明

取得されていない結果セットが存在する場合、mysql_next_result() は次のクエリの結果セットを読み込み、ステータスをアプリケーションに返します。この関数は、MySQL 4.1 で追加されました。

注意: 前のクエリが結果セットを返していた場合、mysql_free_result() を呼び出す必要があります。

mysql_next_result() を呼び出した後、接続の状態は次のクエリについて mysql_real_query() を呼び出した後と同様になります。これは、この関数を呼び出した後、この接続で mysql_store_result()、mysql_warning_count()、mysql_affected_rows() などを呼び出すことができることを意味します。

mysql_next_result() がエラーを返した場合、他のステートメントは実行されません。また、取得されていない結果セットも存在しません。

See 項11.1.8.「C API における複数クエリの実行の取り扱い」。

戻り値

呼び出しが正常に動作し、取得する結果セットが存在していた場合は 0、取得する結果セットが存在しなかった場合は -1、エラーが発生した場合は正の値。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。たとえば、前の結果セットで mysql_use_result() が呼び出されなかった場合にこのエラーが発生する。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

11.1.4. C API のプリペアドステートメント

MySQL 4.1 では、プリペアドステートメントを使用するためのクライアント/サーバプロトコルが提供されています。この機能は、MYSQL_STMT ステートメントハンドラデータ構造体を使用します。プリペアドステートメントの実行は、ステートメントを繰り返し実行する場合に効率的な方法です。ステートメントは、実行する準備を整えるために、まず解析されます。その後、prepare 関数から返されるステートメントハンドルを使用して、1 回以上実行されます。

プリペアドステートメントの実行は、解析が 1 回しか行われないため、ステートメントを直接複数回実行するよりも高速です。ステートメントを直接実行した場合、そのたびにクエリの解析が行われます。プリペアドステートメントを実行する場合、ネットワークトラフィックも、パラメータとしてデータを送信するときに発生するだけなので、削減できます。

その他にも、プリペアドステートメントの実行ではバイナリプロトコルを使用するので、クライアントおよびサーバ間のデータ転送の効率が向上するという利点があります。プリペアドステートメントは、複数クエリを実行する際の入力および出力のバインディングもサポートします。

11.1.5. C API のプリペアドステートメントのデータ型

注意: プリペアドステートメントの API は、今後改訂される可能性があります。ここで説明する情報は、すぐにこの API を使用する開発者向けに書かれていますが、今後 API が変更になる可能性があることをご承知おきください。

プリペアドステートメントは、主に MYSQL_STMT および MYSQL_BIND の各データ構造体を使用します。その他に、時間的なデータを転送するために MYSQL_TIME を使用します。

- MYSQL_STMT

  この構造体は、プリペアドステートメントを表す。プリペアドステートメントは、mysql_prepare() を呼び出して作成する。この関数は、MYSQL_STMT へのポインタであるステートメントハンドルを返す。このハンドルは、その後に呼び出されるすべてのステートメント関連の関数で使用される。

  MYSQL_STMT 構造体のメンバをアプリケーションが使用することはない。

  1 つの接続に複数のステートメントハンドルを関連付けることができる。関連付けるハンドル数は、使用できるシステムリソースによって制限される。

- MYSQL_BIND

  この構造体は、クエリの入力 (データ値がサーバに送信される) および出力 (結果値がサーバから返される) の両方で使用される。入力の場合、mysql_bind_param() に渡してパラメータデータ値をバッファにバインドし、

mysql_execute() が使用できるようにする。出力の場合、mysql_bind_result() に渡して結果セットをバッファにバインドし、mysql_fetch() でレコードを取得する際に使用できるようにする。

MYSQL_BIND 構造体では、以下に示すメンバをアプリケーションプログラムが使用できる。どのメンバも入力および出力のどちらにも使用するが、データ転送の方向によって異なる目的で使用する場合がある。

- enum enum_field_types buffer_type

  バッファの型。使用できる buffer_type の値のリストを、このセクションの後半に示す。入力の場合、buffer_type はクエリパラメータにバインドする値の型を表す。出力の場合、結果セットのバッファで受け取る値の型を表す。

- void *buffer

  入力の場合、クエリパラメータのデータ値が保存されるバッファへのポインタ。出力の場合、結果セットのカラム値を返すバッファへのポインタ。カラム型が数値の場合、buffer は適切な C 言語の型の変数へのポインタとする必要がある (UNSIGNED 属性を持つカラムに変数を関連付けている場合、変数は C 言語の unsigned 型とする必要がある)。カラム型が日付および時刻の場合、buffer は MYSQL_TIME 構造体へのポインタとする必要がある。カラム型が文字列およびバイナリ文字列の場合、buffer は文字バッファへのポインタとする必要がある。

- unsigned long buffer_length

  バイト単位で表した *buffer の実際のサイズ。この値は、バッファに格納できるデータの最大サイズを示す。C の文字データおよびバイナリデータの場合、buffer_length の値は、mysql_bind_param() で使用する場合は *buffer が示すデータの長さ、または mysql_bind_result() で使用する場合はバッファに取得できる最大データバイト数を指定する。

- unsigned long *length

  unsigned long 変数へのポインタ。*buffer に格納するデータの実際のバイト数を示す。length は、格納するデータが C の文字データまたはバイナリデータの場合に使用する。入力パラメータデータをバインドする場合、length は、*buffer に格納するパラメータ値の長さを示す unsigned long 変数を指す。これは mysql_execute() で使用される。length がヌルポインタの場合、プロトコルはすべての文字データおよびバイナリデータがヌルで終端されていると仮定する。出力値をバインドする場合、mysql_fetch() は、返すデータのカラム値の長さを length がポイントする変数に設定する。

  数値データ型または時間的なデータ型の場合、データ値の長さは buffer_type の値によって決まるので、length は無視される。

- my_bool *is_null

  my_bool 変数へのポインタ。値が NULL の場合は true、NULL 以外の場合は false。入力の場合、*is_null を true に設定することによって、クエリパラメータとして NULL 値を渡していることを示す。出力の場合、クエリが返す値が NULL のときにレコードを取得すると、この値が true に設定される。

- MYSQL_TIME

  この構造体は、サーバと直接 DATE、TIME、DATETIME、および TIMESTAMP データを送受信するために使用する。それには、MYSQL_BIND 構造体の buffer_type メンバをいずれかの時間を表す型に設定し、buffer メンバを MYSQL_TIME 構造体をポイントするように設定する。

  MYSQL_TIME 構造体のメンバを以下に示す。

- unsigned int year

  年。

- unsigned int month

  月。

- unsigned int day

  日。

- unsigned int hour

  時。

- unsigned int minute

  分。

- unsigned int second

  秒。

- my_bool neg

  時間が負かどうかを示すブール値フラグ。

- unsigned long second_part

  秒の小数部。現在は未使用。

MYSQL_TIME 構造体のメンバは、使用する時間的な型に対応する部分だけが使用される。year、month、および day の各メンバは、DATE、DATETIME、および TIMESTAMP の値に使用する。hour、minute、および second の各メンバは、TIME、DATETIME、および TIMESTAMP の値に使用する。 See 項11.1.9.「C API における日付値および時刻値の取り扱い」。

MYSQL_BIND 構造体の buffer_type メンバに指定できる値を以下の表に示します。この表には、それぞれの buffer_type の値に最も近い SQL のデータ型を示し、さらに数値型と時間的な値の型については対応する C のデータ型も示します。

buffer_type 値	SQL のデータ型	C のデータ型
MYSQL_TYPE_TINY	TINYINT	char
MYSQL_TYPE_SHORT	SMALLINT	short int
MYSQL_TYPE_LONG	INT	long int
MYSQL_TYPE_LONGLONG	BIGINT	long long int
MYSQL_TYPE_FLOAT	FLOAT	float

MYSQL_TYPE_DOUBLE	DOUBLE	double
MYSQL_TYPE_TIME	TIME	MYSQL_TIME
MYSQL_TYPE_DATE	DATE	MYSQL_TIME
MYSQL_TYPE_DATETIME	DATETIME	MYSQL_TIME
MYSQL_TYPE_TIMESTAMP	TIMESTAMP	MYSQL_TIME
MYSQL_TYPE_STRING	CHAR	
MYSQL_TYPE_VAR_STRING	VARCHAR	
MYSQL_TYPE_TINY_BLOB	TINYBLOB/TINYTEXT	
MYSQL_TYPE_BLOB	BLOB/TEXT	
MYSQL_TYPE_MEDIUM_BLOB	MEDIUMBLOB/MEDIUMTEXT	
MYSQL_TYPE_LONG_BLOB	LONGBLOB/LONGTEXT	

暗黙のデータ型変換はどちらの方向にも行われる可能性があります。

11.1.6. C API のプリペアドステートメント関数の概要

注意: プリペアドステートメントの API は、今後改訂される可能性があります。ここで説明する情報は、すぐにこの API を使用する開発者向けに書かれていますが、今後 API が変更になる可能性があることをご承知おきください。

ここでは、プリペアドステートメントの処理に使用できる関数について簡単に説明します。詳細については、以降のセクションで説明します。 See 項11.1.7.「C API のプリペアドステートメント関数の説明」。

関数	説明
mysql_prepare()	SQL 文字列を実行するためのプリコンパイルを行う。
mysql_param_count()	プリペアドステートメントのパラメータ数を返す。
mysql_get_metadata()	プリペアドステートメントメタデータを結果セット形式で返す。
mysql_bind_param()	アプリケーションデータバッファを、プリペアドステートメント内のパラメータマーカーに関連付ける。
mysql_execute()	プリペアドステートメントを実行する。
mysql_stmt_affected_rows()	最後に実行された UPDATE、DELETE、または INSERT のいずれかのクエリによって変更、削除、または挿入されたレコードの数を返す。
mysql_bind_result()	アプリケーションデータバッファを、結果セットのカラムに関連付ける。
mysql_stmt_store_result()	結果セット全体を取得してクライアントに転送する。
mysql_stmt_data_seek()	ステートメントの結果セットの任意のレコード番号にシークする。
mysql_stmt_row_seek()	mysql_stmt_row_tell() から返された値をオフセットとして、ステートメントの結果セット内のレコードにシークする。
mysql_stmt_row_tell()	ステートメントのレコードカーソルの位置を返す。
mysql_stmt_num_rows()	バッファに格納されたステートメントの結果セット全体のレコード数を返す。
mysql_fetch()	結果セットの次のレコードのデータを取得し、バインドされたすべてのカラムの

	データを返す。
mysql_stmt_close()	プリペアドステートメントが使用していたメモリを解放する。
mysql_stmt_errno()	最後に実行されたステートメントのエラー番号を返す。
mysql_stmt_error()	最後に実行されたステートメントのエラーメッセージを返す。
mysql_stmt_sqlstate()	最後に実行したステートメントの SQLSTATE エラーコードを返す。
mysql_send_long_data()	long 型データを切り分けてサーバに送信する。

mysql_prepare() を呼び出してステートメントハンドルをプリコンパイルし、初期化します。次に mysql_bind_param() を呼び出してパラメータデータを渡し、mysql_execute() を呼び出してクエリを実行します。mysql_bind_param() 経由で渡すバッファに格納したパラメータ値を変更して、mysql_execute() を繰り返し実行することもできます。

クエリが SELECT ステートメントの場合、または結果セットを生成するその他のクエリの場合、mysql_prepare() は mysql_get_metadata() を使用して MYSQL_RES 結果セット形式で結果セットのメタデータ情報も返します。

結果を格納するバッファを mysql_bind_result() を使用して渡すことによって、mysql_fetch() は自動的にそのバッファにデータを返します。ここでは 1 行ずつの取得が行われます。

オプションとして is_long_data=1 または length=MYSQL_LONG_DATA または -2 を指定した MYSQL_BIND 構造体を mysql_bind_param() で渡すと、mysql_send_long_data() を使用してテキストデータまたはバイナリデータを切り分けてサーバに送信できます。

ステートメントの実行が完了したら、mysql_stmt_close() を呼び出してステートメントハンドルをクローズし、ハンドルに関連付けられているすべてのリソースを解放する必要があります。

mysql_get_metadata() を呼び出して SELECT ステートメントの結果セットメタデータを取得していた場合も、mysql_free_result() を呼び出して解放する必要があります。

実行ステップ

ステートメントをプリコンパイルして実行するには、アプリケーションで以下のステップに従って処理します。

1. SQL ステートメントを含む文字列をパラメータとして mysql_prepare() を呼び出す。プリコンパイルが正常に終了した場合、mysql_prepare() は有効なステートメントハンドラをアプリケーションに返す。
2. 結果セットを生成するクエリを実行する場合、mysql_get_metadata() を呼び出して結果セットメタデータを取得する。このメタデータはそれ自身が結果セット形式である。ただし、クエリが返すレコードを含む結果セットとは別に渡される。結果セットメタデータには、結果セットに含まれるカラム数および各カラムに関する情報が含まれる。
3. mysql_bind_param() を使用して、パラメータの値を設定する。すべてのパラメータの値を設定する必要がある。値が設定されていないパラメータが残っていると、エラーが返されたり、予想外の結果が発生する。
4. mysql_execute() を呼び出してステートメントを実行する。
5. 結果セットを生成するクエリを実行した場合、mysql_bind_result() を呼び出して、レコードの値を取得するためのデータバッファをバインドする。
6. mysql_fetch() を呼び出して、1 行分のデータをバッファに取得する。すべてのレコードのデータを取得できるまでこれを繰り返す。

7. 必要に応じてステップ 3 から 6 までを繰り返して、パラメータの値を変更しながらステートメントを再実行する。

mysql_prepare() が呼び出された場合に実行される MySQL クライアント/サーバのプロトコルを以下に示します。

- サーバはクエリを解析し、ステートメント ID を割り当てて、OK ステータスをクライアントに返信する。結果セットを生成するクエリの場合は、さらに、パラメータの合計数、カラム数、およびそのメタ情報を送信する。この呼び出しで、サーバはクエリのすべての構文および意味を確認する。
- クライアントはこのステートメント ID を使用してその後の処理を実行する。この ID を使用して、サーバはステートメントプールの中から使用するステートメントを識別する。また、クライアントはステートメントハンドルをこの ID に割り当て、ハンドルをアプリケーションに返す。

mysql_execute() が呼び出された場合に実行される MySQL クライアント/サーバプロトコルを以下に示します。

- クライアントはステートメントハンドルを使用して、サーバにパラメータデータを送信する。
- サーバはクライアントから渡された ID を使用してステートメントを識別し、パラメータマーカーを新しく渡されたデータに置き換え、クエリを実行する。結果セットを生成するクエリを実行した場合、サーバはデータをクライアントに返信する。結果セットを生成しないクエリを実行した場合、サーバは OK ステータスおよび変更、削除、または挿入されたレコードの合計数を返す。

mysql_fetch() が呼び出された場合に実行される MySQL クライアント/サーバプロトコルを以下に示します。

- クライアントは、パケットから 1 行分ずつデータを読み込み、必要に応じてデータ変換しながらアプリケーションデータバッファに格納する。アプリケーションバッファの型がサーバから返されたフィールドの型と同じ場合は、直接的に変換が行われる。

mysql_stmt_errno()、mysql_stmt_error()、および mysql_stmt_sqlstate() を使用すると、それぞれステートメントのエラーコード、エラーメッセージ、および SQLSTATE 値を取得できます。

11.1.7. C API のプリペアドステートメント関数の説明

クエリをプリコンパイルして実行するには、以下の関数を使用します。

11.1.7.1. mysql_prepare()

MYSQL_STMT * mysql_prepare(MYSQL *mysql, const char *query, unsigned long length)

説明

ヌル終端文字列 query で指定された SQL クエリをプリコンパイルし、その後の処理に使用するステートメントハンドルを返します。このクエリは単一 SQL ステートメントで構成されている必要があります。このステートメントには、その終わりを示すセミコロン (';') または \g を追加する必要はありません。

アプリケーションは、SQL 文字列の適当な位置に疑問符 ('?') を埋め込むことで、SQL ステートメントで 1 つ以上のパラメータマーカーを使用できます。

マーカーは、SQL ステートメントの特定の位置に埋め込まれた場合にのみ適正に処理されます。たとえば、マーカーは INSERT ステートメントの VALUES() のリストでレコードに設定するカラムの値を指定する部分、または WHERE 節でカラムの値と比較する値を指定する部分に使用できます。しかし、SELECT ステートメントが返すカラムを指定する選択リストで識別子（テーブルまたはカラムの名前など）として使用したり、等号（=）などのバイナリ演算子の両方のオペランドを指定することはできません。パラメータの型を決定することができないので、後者の制約が必要になります。一般に、パラメータは、データ操作言語（DML）ステートメントでのみ適正に処理され、データ定義言語（DDL）ステートメントでは処理されません。

パラメータマーカーは、ステートメントを実行する前に、mysql_bind_param() を使用してアプリケーション変数にバインドする必要があります。

戻り値

コンパイルが正常に終了した場合は MYSQL_STMT 構造体へのポインタ。エラーが発生した場合は NULL。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_OUT_OF_MEMORY

  メモリが不足していた。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

コンパイルが失敗した場合（mysql_prepare() が NULL を返した場合）、エラーメッセージは mysql_error() を呼び出すことによって取得できます。

例

mysql_prepare() の使用方法については、項11.1.7.5.「mysql_execute()」 の「例」を参照してください。

11.1.7.2. mysql_param_count()

unsigned long mysql_param_count(MYSQL_STMT *stmt)

説明

プリコンパイルされたステートメントに含まれるパラメータマーカーの数を返します。

戻り値

ステートメントに含まれるパラメータ数を表す unsigned long 整数。

エラー

ありません。

例

mysql_param_count() の使用方法については、項11.1.7.5.「mysql_execute()」 の「例」を参照してください。

11.1.7.3. mysql_get_metadata()

MYSQL_RES *mysql_get_metadata(MYSQL_STMT *stmt)

説明

mysql_prepare() に渡されたステートメントが結果セットを生成するものである場合、mysql_get_metadata() は MYSQL_RES 構造体へのポインタという形で結果セットメタデータを返します。この構造体を使用して、フィールドの合計数および個々のフィールド情報などのメタ情報を処理できます。この結果セットへのポインタは、結果セットメタデータを処理する以下のフィールドベースの API 関数に引数として渡すことができます。

- mysql_num_fields()
- mysql_fetch_field()
- mysql_fetch_field_direct()
- mysql_fetch_fields()
- mysql_field_count()
- mysql_field_seek()
- mysql_field_tell()
- mysql_free_result()

結果セット構造体は、使用する必要がなくなったら mysql_free_result() に渡して解放する必要があります。これは、mysql_store_result() を呼び出して取得した結果セットを解放する方法と似ています。

mysql_get_metadata() から返される結果セットには、メタデータだけが含まれます。レコードデータは含まれません。レコードを取得するには、ステートメントハンドルをパラメータとして mysql_fetch() を呼び出します。

戻り値

MYSQL_RES 構造体。プリコンパイルしたクエリのメタ情報が存在しない場合は NULL。

エラー

- CR_OUT_OF_MEMORY

メモリが不足していた。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

例

mysql_get_metadata() の使用方法については、項11.1.7.13.「mysql_fetch()」 の「例」を参照してください。

11.1.7.4. mysql_bind_param()

my_bool mysql_bind_param(MYSQL_STMT *stmt, MYSQL_BIND *bind)

説明

mysql_bind_param() は、mysql_prepare() に渡された SQL ステートメントに含まれるパラメータマーカーのデータをバインドするために使用します。データは MYSQL_BIND 構造体を使用して渡します。bind は、MYSQL_BIND 構造体の配列のアドレスです。クライアントライブラリは、この配列に、クエリに含まれる各 '?' パラメータに対応する要素が含まれていると仮定します。

たとえば、以下のステートメントをプリコンパイルするとします。

```
INSERT INTO mytbl VALUES(?,?,?)
```

パラメータをバインドする場合、MYSQL_BIND 構造体には 3 つの要素が必要であり、以下のように宣言できます。

```
MYSQL_BIND bind[3];
```

設定する必要がある MYSQL_BIND の各要素のメンバについては、項11.1.5.「C API のプリペアドステートメントのデータ型」 を参照してください。

戻り値

正常にバインドできた場合は 0。エラーが発生した場合は 0 以外。

エラー

- CR_NO_PREPARE_STMT

  プリコンパイルされたステートメントが存在しない。

- CR_NO_PARAMETERS_EXISTS

  バインドするパラメータが存在しない。

- CR_INVALID_BUFFER_USE

  これがlong データを切り分けて渡すためのバインドか、およびバッファの型が文字列以外またはバイナリ以外かを示す。

- CR_UNSUPPORTED_PARAM_TYPE

  変換がサポートされていない。buffer_type の値が正しくないか、またはサポートされていない型である可能性がある。

- CR_OUT_OF_MEMORY

  メモリが不足していた。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

例

mysql_bind_param() の使用方法については、項11.1.7.5.「mysql_execute()」 の「例」を参照してください。

11.1.7.5. mysql_execute()

int mysql_execute(MYSQL_STMT *stmt)

説明

mysql_execute() は、ステートメントハンドルに関連付けられているプリコンパイルされたクエリを実行します。この呼び出しの際に、現在バインドされているパラメータマーカーの値がサーバに送信され、サーバはマーカーをここで渡されたデータで置き換えます。

ステートメントが UPDATE、DELETE、または INSERT の場合、mysql_stmt_affected_rows() を呼び出すことによって、変更、削除、または挿入されたレコードの合計数を取得できます。これが SELECT などの結果セットを生成するクエリの場合、最初に mysql_fetch() を呼び出してデータを取得してから、他のクエリ処理を必要とする関数を呼び出す必要があります。結果を取得する方法の詳細については、項11.1.7.13.「mysql_fetch()」 を参照してください。

戻り値

正常に実行した場合は 0。エラーが発生した場合は 0 以外。エラーコードおよびエラーメッセージは、mysql_stmt_errno() および mysql_stmt_error() を呼び出すことによって取得できます。

エラー

- CR_NO_PREPARE_QUERY

  実行する前にプリコンパイルされたクエリが存在しない。

- CR_ALL_PARAMS_NOT_BOUND

  渡されていないパラメータデータがある。

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_OUT_OF_MEMORY

メモリが不足していた。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

例

以下の例では、mysql_prepare()、mysql_param_count()、mysql_bind_param()、mysql_execute()、および mysql_stmt_affected_rows() を使用して、テーブルを作成し、データを挿入する方法を示します。mysql 変数は有効な接続ハンドルとします。

```
#define STRING_SIZE 50

#define DROP_SAMPLE_TABLE "DROP TABLE IF EXISTS test_table"
#define CREATE_SAMPLE_TABLE "CREATE TABLE test_table(col1 INT,\
                                     col2 VARCHAR(40),\
                                     col3 SMALLINT,\
                                     col4 TIMESTAMP)"
#define INSERT_SAMPLE "INSERT INTO test_table(col1,col2,col3) VALUES(?,?,?)"

MYSQL_STMT    *stmt;
MYSQL_BIND    bind[3];
my_ulonglong  affected_rows;
int           param_count;
short         small_data;
int           int_data;
char          str_data[STRING_SIZE];
unsigned long str_length;
my_bool       is_null;

if (mysql_query(mysql, DROP_SAMPLE_TABLE))
{
  fprintf(stderr, " DROP TABLE failed\n");
  fprintf(stderr, " %s\n", mysql_error(mysql));
  exit(0);
}

if (mysql_query(mysql, CREATE_SAMPLE_TABLE))
{
  fprintf(stderr, " CREATE TABLE failed\n");
  fprintf(stderr, " %s\n", mysql_error(mysql));
  exit(0);
}

/* Prepare an INSERT query with 3 parameters */
/* (the TIMESTAMP column is not named; it will */
```

```
/* be set to the current date and time) */
stmt = mysql_prepare(mysql, INSERT_SAMPLE, strlen(INSERT_SAMPLE));
if (!stmt)
{
  fprintf(stderr, " mysql_prepare(), INSERT failed\n");
  fprintf(stderr, " %s\n", mysql_error(mysql));
  exit(0);
}
fprintf(stdout, " prepare, INSERT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_param_count(stmt);
fprintf(stdout, " total parameters in INSERT: %d\n", param_count);

if (param_count != 3) /* validate parameter count */
{
  fprintf(stderr, " invalid parameter count returned by MySQL\n");
  exit(0);
}

/* Bind the data for all 3 parameters */

/* INTEGER PARAM */
/* This is a number type, so there is no need to specify buffer_length */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= 0;
bind[0].length= 0;

/* STRING PARAM */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= 0;
bind[1].length= &str_length;

/* SMALLINT PARAM */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null;
bind[2].length= 0;

/* Bind the buffers */
if (mysql_bind_param(stmt, bind))
{
  fprintf(stderr, " mysql_bind_param() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Specify the data values for the first row */
int_data= 10;             /* integer */
strncpy(str_data, "MySQL", STRING_SIZE); /* string  */
str_length= strlen(str_data);

/* INSERT SMALLINT data as NULL */
is_null= 1;
```

```
/* Execute the INSERT statement - 1*/
if (mysql_execute(stmt))
{
  fprintf(stderr, " mysql_execute(), 1 failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get the total number of affected rows */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 1): %ld\n", affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
  fprintf(stderr, " invalid affected rows by MySQL\n");
  exit(0);
}

/* Specify data values for second row, then re-execute the statement */
int_data= 1000;
strncpy(str_data, "The most popular open source database", STRING_SIZE);
str_length= strlen(str_data);
small_data= 1000;         /* smallint */
is_null= 0;               /* reset */

/* Execute the INSERT statement - 2*/
if (mysql_execute(stmt))
{
  fprintf(stderr, " mysql_execute, 2 failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get the total rows affected */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 2): %ld\n", affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
  fprintf(stderr, " invalid affected rows by MySQL\n");
  exit(0);
}

/* Close the statement */
if (mysql_stmt_close(stmt))
{
  fprintf(stderr, " failed while closing the statement\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
```

注意: プリペアドステートメント関数を使用した完全な例については、ファイル tests/mysql_client_test.c を参照してください。このファイルは、MySQL ソースディストリビューションまたは BitKeeper ソースリポジトリから取得できます。

11.1.7.6. mysql_stmt_affected_rows()

my_ulonglong mysql_stmt_affected_rows(MYSQL_STMT *stmt)

説明

最後に実行されたステートメントによって変更、削除、または挿入されたレコードの合計数を返します。UPDATE、DELETE、または INSERT のいずれかのステートメントに対して mysql_execute() を呼び出した直後に呼び出される可能性があります。SELECT ステートメントの場合、mysql_stmt_affected_rows() を呼び出すと、mysql_num_rows() と同様に動作します。

戻り値

正の整数は、影響を与えた、または取得した、レコード数を示します。0 は、UPDATE によって更新されたレコードがなかったこと、クエリの WHERE 節に一致するレコードがなかったこと、またはクエリが実行されていないことを示します。−1 は、クエリがエラーを返したこと、または SELECT クエリの場合に mysql_fetch() を呼び出す前に mysql_stmt_affected_rows() が呼び出されたことを示します。

エラー

ありません。

例

mysql_stmt_affected_rows() の使用方法については、項11.1.7.5.「mysql_execute()」 の「例」を参照してください。

11.1.7.7. mysql_bind_result()

my_bool mysql_bind_result(MYSQL_STMT *stmt, MYSQL_BIND *bind)

説明

mysql_bind_result() は、結果セットのカラムをデータバッファおよび長さバッファに関連付ける（バインドする）ために使用します。mysql_fetch() を呼び出してデータを取得する場合、MySQL クライアント/サーバプロトコルによって、バインドされたカラムのデータが指定されたバッファに書き込まれます。

注意: mysql_fetch() を呼び出す前に、すべてのカラムをバッファにバインドする必要があります。bind は、MYSQL_BIND 構造体の配列のアドレスです。クライアントライブラリは、この配列に、結果セットの各カラムに対応した要素が含まれていると仮定します。要素が含まれていない場合、mysql_fetch() は単にデータを取得しません。また、クライアント/サーバプロトコルはデータの値を切り分けて返すことはしないので、バッファのサイズはデータ値を格納できるだけの十分な大きさである必要があります。

カラムは、任意の時点で、結果セットの一部だけが取得された後でも、バインドまたは再バインドすることができます。新しくバインドした場合、それが有効になるのは、次に mysql_fetch() を呼び出したときです。たとえば、アプリケーションが結果セットのカラムをバインドして mysql_fetch() を呼び出すとします。クライアント/サーバプロトコルは、バインドされたバッファにデータを返します。次にアプリケーションがカラムを別のバッファにバインドするとします。プロトコルは、次に mysql_fetch() が呼び出されるまで、新しくバインドされたバッファにデータを書き込みません。

カラムをバインドするには、アプリケーションは mysql_bind_result() を呼び出して、型、アドレス、および長さバッファのアドレスを渡します。設定する必要がある MYSQL_BIND の各要素のメンバについては、項11.1.5.「C API のプリペアドステートメントのデータ型」 を参照してください。

戻り値

正常にバインドできた場合は 0。エラーが発生した場合は 0 以外。

エラー

- CR_NO_PREPARE_STMT

  プリコンパイルされたステートメントが存在しない。

- CR_UNSUPPORTED_PARAM_TYPE

  変換がサポートされていない。buffer_type の値が正しくないか、またはサポートされていない型である可能性がある。

- CR_OUT_OF_MEMORY

  メモリが不足していた。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

例

mysql_bind_result() の使用方法については、項11.1.7.13.「mysql_fetch()」 の「例」を参照してください。

11.1.7.8. mysql_stmt_store_result()

int mysql_stmt_store_result(MYSQL_STMT *stmt)

説明

クエリが結果セットを正常に生成し (SELECT、SHOW、DESCRIBE、EXPLAIN)、さらにクライアントのバッファに結果セット全体を格納して、その後の mysql_fetch() の呼び出しでバッファされたデータが返されるようにする場合に限り、mysql_stmt_store_result() を呼び出す必要があります。

そのようなクエリ以外では mysql_stmt_store_result() を呼び出す必要はありませんが、呼び出したとしても異常が発生したり、目立った動作が行われるようなことは一切ありません。mysql_get_metadata() が NULL を返すかどうかを調べることで、クエリが結果セットを生成したかどうかを検出できます。詳細については、項11.1.7.3.「mysql_get_metadata()」を参照してください。

戻り値

結果セットが正常にバッファに格納された場合は 0。エラーが発生した場合は 0 以外。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_OUT_OF_MEMORY

メモリが不足していた。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

11.1.7.9. mysql_stmt_data_seek()

void mysql_stmt_data_seek(MYSQL_STMT *stmt, my_ulonglong offset)

説明

ステートメントの結果セットの任意のレコードにシークします。offset 値はレコード番号を表し、0 から mysql_stmt_num_rows(stmt)-1 の範囲で指定します。

mysql_stmt_data_seek() を使用するには、ステートメントの結果セット構造体に最後に実行されたクエリの結果全体が格納されている必要があるので、mysql_stmt_store_result() とともに使用する必要があります。

戻り値

ありません。

エラー

ありません。

11.1.7.10. mysql_stmt_row_seek()

MYSQL_ROW_OFFSET mysql_stmt_row_seek(MYSQL_STMT *stmt, MYSQL_ROW_OFFSET offset)

説明

ステートメントの結果セットの任意のレコードにレコードカーソルを設定します。offset の値はレコードオフセットであり、mysql_stmt_row_tell() または mysql_stmt_row_seek() の戻り値になります。この値はレコード番号ではありません。結果セット内のレコードに番号を指定してシークする場合は、この関数ではなく、mysql_stmt_data_seek() を使用します。

mysql_stmt_row_seek() を使用するには、結果セット構造体にクエリの結果全体が格納されている必要があるので、mysql_stmt_store_result() とともに使用する必要があります。

戻り値

シークする前のレコードカーソルの値。mysql_stmt_row_seek() を次に呼び出すときにこの値を渡すことができます。

エラー

ありません。

11.1.7.11. mysql_stmt_row_tell()

MYSQL_ROW_OFFSET mysql_stmt_row_tell(MYSQL_STMT *stmt)

説明

最後に実行された mysql_fetch() で使用されたレコードカーソルの現在の位置を返します。この戻り値を、mysql_stmt_row_seek() に引数として渡すことができます。

mysql_stmt_row_tell() は、mysql_stmt_store_result() の後でのみ使用できます。

戻り値

レコードカーソルの現在のオフセット。

エラー

ありません。

11.1.7.12. mysql_stmt_num_rows()

my_ulonglong mysql_stmt_num_rows(MYSQL_STMT *stmt)

説明

結果セットのレコード数を返します。

mysql_stmt_num_rows() を使用するかどうかは、mysql_stmt_store_result() を使用して結果セット全体をステートメントハンドルに格納したかどうかによって決まります。

mysql_stmt_store_result() を使用する場合、すぐに mysql_stmt_num_rows() を呼び出すことができます。

戻り値

結果セットのレコード数。

エラー

ありません。

11.1.7.13. mysql_fetch()

int mysql_fetch(MYSQL_STMT *stmt)

説明

mysql_fetch() は、結果セットの次のレコードを取得します。この関数は結果セットが存在する間のみ、すなわち、mysql_execute() を呼び出して結果セットを作成した後、または mysql_execute() を呼び出して結果セット全体をバッファに格納してから mysql_stmt_store_result() を呼び出した後にのみ、呼び出すことができます。

mysql_fetch() は、mysql_bind_result() を呼び出してバインドしたバッファを使用してレコードデータを返します。バッファには、現在のレコードセットのすべてのカラムのデータが格納されます。データの長さは length ポインタに返されます。

注意: アプリケーションは、mysql_fetch() を呼び出す前に、すべてのカラムをバッファにバインドする必要があります。

取得したデータが NULL 値だった場合、対応する MYSQL_BIND 構造体の *is_null の値には TRUE (1) が格納されます。それ以外の値だった場合、アプリケーションが指定したバッファの型に基づいて、データおよびその長さがそれぞれ *buffer 要素および *length 要素に返されます。数値型および時間的な値の型は、以下の表に示すように、それぞれ固定長です。文字列型の長さは、実際のデータの長さによって決まり、data_length によって表されます。

型	長さ
MYSQL_TYPE_TINY	1
MYSQL_TYPE_SHORT	2
MYSQL_TYPE_LONG	4
MYSQL_TYPE_LONGLONG	8
MYSQL_TYPE_FLOAT	4
MYSQL_TYPE_DOUBLE	8
MYSQL_TYPE_TIME	sizeof(MYSQL_TIME)
MYSQL_TYPE_DATE	sizeof(MYSQL_TIME)
MYSQL_TYPE_DATETIME	sizeof(MYSQL_TIME)
MYSQL_TYPE_STRING	data length
MYSQL_TYPE_BLOB	data_length

戻り値

戻り値	説明
0	正常。データはアプリケーションデータバッファに取得されている。
1	エラーが発生した。エラーコードおよびエラーメッセージは、mysql_stmt_errno() および mysql_stmt_error() を呼び出すことによって取得できる。
MYSQL_NO_DATA	取得されていないレコード/データは残っていない。

エラー

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_OUT_OF_MEMORY

  メモリが不足していた。

- CR_SERVER_GONE_ERROR

MySQL サーバがいなくなった。

- CR_SERVER_LOST

  クエリの実行中にサーバへの接続が切断された。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

- CR_UNSUPPORTED_PARAM_TYPE

  バッファの型は MYSQL_TYPE_DATE、MYSQL_TYPE_TIME、MYSQL_TYPE_DATETIME、または MYSQL_TYPE_TIMESTAMP なのに、データの型は DATE、TIME、DATETIME、または TIMESTAMP のいずれでもない。

- その他のサポートされていない変換に関するエラーはすべて mysql_bind_result() から返される。

例

以下の例では、mysql_get_metadata()、mysql_bind_result()、および mysql_fetch() を使用して、テーブルからデータを取得する方法を示します（この例では 項11.1.7.5.「mysql_execute()」 の例で挿入された 2 行のレコードを取得するものと想定しています）。mysql 変数は有効な接続ハンドルとします。

```
#define STRING_SIZE 50

#define SELECT_SAMPLE "SELECT col1, col2, col3, col4 FROM test_table"

MYSQL_STMT    *stmt;
MYSQL_BIND    bind[4];
MYSQL_RES     *prepare_meta_result;
MYSQL_TIME    ts;
unsigned long length[4];
int           param_count, column_count, row_count;
short         small_data;
int           int_data;
char          str_data[STRING_SIZE];
my_bool       is_null[4];

/* Prepare a SELECT query to fetch data from test_table */
stmt = mysql_prepare(mysql, SELECT_SAMPLE, strlen(SELECT_SAMPLE));
if (!stmt)
{
  fprintf(stderr, " mysql_prepare(), SELECT failed\n");
  fprintf(stderr, " %s\n", mysql_error(mysql));
  exit(0);
}
fprintf(stdout, " prepare, SELECT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_param_count(stmt);
fprintf(stdout, " total parameters in SELECT: %d\n", param_count);

if (param_count != 0) /* validate parameter count */
```

```
{
  fprintf(stderr, " invalid parameter count returned by MySQL\n");
  exit(0);
}

/* Fetch result set meta information */
prepare_meta_result = mysql_get_metadata(stmt);
if (!prepare_meta_result)
{
  fprintf(stderr, " mysql_get_metadata(), returned no meta information\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get total columns in the query */
column_count= mysql_num_fields(prepare_meta_result);
fprintf(stdout, " total columns in SELECT statement: %d\n", column_count);

if (column_count != 4) /* validate column count */
{
  fprintf(stderr, " invalid column count returned by MySQL\n");
  exit(0);
}

/* Execute the SELECT query */
if (mysql_execute(stmt))
{
  fprintf(stderr, " mysql_execute(), failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Bind the result buffers for all 4 columns before fetching them */

/* INTEGER COLUMN */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= &is_null[0];
bind[0].length= &length[0];

/* STRING COLUMN */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= &is_null[1];
bind[1].length= &length[1];

/* SMALLINT COLUMN */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null[2];
bind[2].length= &length[2];

/* TIMESTAMP COLUMN */
bind[3].buffer_type= MYSQL_TYPE_TIMESTAMP;
bind[3].buffer= (char *)&ts;
bind[3].is_null= &is_null[3];
bind[3].length= &length[3];
```

```
/* Bind the result buffers */
if (mysql_bind_result(stmt, bind))
{
  fprintf(stderr, " mysql_bind_result() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Now buffer all results to client */
if (mysql_stmt_store_result(stmt))
{
  fprintf(stderr, " mysql_stmt_store_result() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Fetch all rows */
row_count= 0;
fprintf(stdout, "Fetching results ...\n");
while (!mysql_fetch(stmt))
{
  row_count++;
  fprintf(stdout, "  row %d\n", row_count);

  /* column 1 */
  fprintf(stdout, "   column1 (integer)  : ");
  if (is_null[0])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %d(%ld)\n", int_data, length[0]);

  /* column 2 */
  fprintf(stdout, "   column2 (string)   : ");
  if (is_null[1])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %s(%ld)\n", str_data, length[1]);

  /* column 3 */
  fprintf(stdout, "   column3 (smallint) : ");
  if (is_null[2])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %d(%ld)\n", small_data, length[2]);

  /* column 4 */
  fprintf(stdout, "   column4 (timestamp): ");
  if (is_null[3])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %04d-%02d-%02d %02d:%02d:%02d (%ld)\n",
                    ts.year, ts.month, ts.day,
                    ts.hour, ts.minute, ts.second,
                    length[3]);
  fprintf(stdout, "\n");
}
```

```
/* Validate rows fetched */
fprintf(stdout, " total rows fetched: %d\n", row_count);
if (row_count != 2)
{
  fprintf(stderr, " MySQL failed to return all rows\n");
  exit(0);
}

/* Free the prepared result metadata */
mysql_free_result(prepare_meta_result);


/* Close the statement */
if (mysql_stmt_close(stmt))
{
  fprintf(stderr, " failed while closing the statement\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
```

11.1.7.14. mysql_send_long_data()

my_bool mysql_send_long_data(MYSQL_STMT *stmt, unsigned int parameter_number, const char *data, unsigned long length)

説明

アプリケーションはこの関数を使用して、パラメータデータを個別に (または ``切り分けて") サーバに送信できます。この関数を複数回呼び出すことによって、TEXT または BLOB のどちらかのデータ型のカラムの文字データ値またはバイナリデータ値を複数回に分けて送信することができます。

parameter_number は、データを関連付けるパラメータを示す番号です。パラメータの番号は 0 から始まります。data は送信データを含むバッファへのポインタを、length はバッファ内のデータのバイト数を示します。

戻り値

データをサーバに正常に送信できた場合は 0。エラーが発生した場合は 0 以外。

エラー

- CR_INVALID_PARAMETER_NO

  無効なパラメータ番号が指定された。

- CR_COMMANDS_OUT_OF_SYNC

  コマンドが正しい順序で実行されなかった。

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_OUT_OF_MEMORY

メモリが不足していた。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

例

以下の例では、TEXT 型カラムのデータを切り分けて送信する方法を示します。ここでは、データ値 'MySQL - The most popular open source database' を text_column カラムに挿入します。mysql 変数は有効な接続ハンドルとします。

```
#define INSERT_QUERY "INSERT INTO test_long_data(text_column) VALUES(?)"

MYSQL_BIND bind[1];
long       length;

if (!mysql_prepare(mysql, INSERT_QUERY, strlen(INSERT_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;

/* Bind the buffers */
if (mysql_bind_param(stmt, bind))
{
  fprintf(stderr, "\n param bind failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

 /* Supply data in chunks to server */
 if (!mysql_send_long_data(stmt,0,"MySQL",5))
{
  fprintf(stderr, "\n send_long_data failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

 /* Supply the next piece of data */
 if (mysql_send_long_data(stmt,0," - The most popular open source database",40))
{
  fprintf(stderr, "\n send_long_data failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

 /* Now, execute the query */
 if (mysql_execute(stmt))
{
  fprintf(stderr, "\n mysql_execute failed");
```

```
 fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
 exit(0);
}
```

11.1.7.15. mysql_stmt_close()

my_bool mysql_stmt_close(MYSQL_STMT *)

説明

プリペアドステートメントをクローズします。mysql_stmt_close() では、stmt が示すステートメントハンドルに割り当てられたメモリも解放されます。

現在のステートメントの結果セットの処理が途中か、または取得されていないレコードが残っている場合、この関数を呼び出すと、その状態をキャンセルして、次のクエリを実行できるようにします。

戻り値

ステートメントが正常に解放された場合は 0。エラーが発生した場合は 0 以外。

エラー

- CR_SERVER_GONE_ERROR

  MySQL サーバがいなくなった。

- CR_UNKNOWN_ERROR

  不明なエラーが発生した。

例

mysql_stmt_close() の使用方法については、項11.1.7.5.「mysql_execute()」 の「例」を参照してください。

11.1.7.16. mysql_stmt_errno()

unsigned int mysql_stmt_errno(MYSQL_STMT *stmt)

説明

mysql_stmt_errno() は、stmt で指定されるステートメントについて、最後に呼び出された、成功または失敗する可能性のあるステートメント API 関数のエラーコードを返します。戻り値が 0 の場合、エラーは発生していません。クライアントのエラーメッセージ番号の一覧は、MySQL errmsg.h ヘッダファイルに記述されています。サーバのエラーメッセージ番号の一覧は、mysqld_error.h に記述されています。MySQL ソースディストリビューションに含まれるファイル Docs/mysqld_error.txt には、すべてのエラーメッセージおよびエラー番号の一覧が記述されています。サーバのエラーコードの一覧は、項12.1.「返されるエラー」 にも記載されています。

戻り値

エラーコードの値。エラーが発生していない場合は 0。

エラー

ありません。

11.1.7.17. mysql_stmt_error()

const char *mysql_stmt_error(MYSQL_STMT *stmt)

説明

mysql_stmt_error() は、stmt で指定されるステートメントについて、最後に呼び出された、成功または失敗する可能性のあるステートメント API 関数のエラーメッセージを含むヌル終端文字列を返します。エラーが発生していない場合は空文字列 ("") を返します。これは、次の 2 つの比較は等価であることを意味します。

```
if (mysql_stmt_errno(stmt))
{
  // an error occurred
}

if (mysql_stmt_error(stmt)[0])
{
  // an error occurred
}
```

クライアントのエラーメッセージの言語は、MySQL クライアントライブラリを再コンパイルすることによって変更できます。現在は、複数の異なる言語でエラーメッセージを返すことができます。

戻り値

エラーの内容を説明する文字列。エラーが発生していない場合は空文字列。

エラー

ありません。

11.1.7.18. mysql_stmt_sqlstate()

const char *mysql_stmt_sqlstate(MYSQL_STMT *stmt)

説明

mysql_stmt_sqlstate() は、stmt で指定されるステートメントについて、最後に呼び出された、成功または失敗する可能性のあるプリペアドステートメント API 関数の SQLSTATE エラーコードを含むヌル終端文字列を返します。エラーコードは 5 文字で示されます。"00000" は、``エラーがない" ことを意味します。値は ANSI SQL および ODBC によって規定されています。考えられる値の一覧については、項12.1.「返されるエラー」 を参照してください。

注意: すべての MySQL エラーが SQLSTATE にマッピングされているわけではありません。マッピングされていないエラーの場合は "HY000" (一般エラー) が使用されます。

この関数は MySQL 4.1.1 で追加されました。

戻り値

SQLSTATE エラーコードを含むヌル終端文字列。

11.1.8. C API における複数クエリの実行の取り扱い

MySQL は、バージョン 4.1 から 1 つのクエリ文字列で指定された複数ステートメントの実行をサポートします。接続でこの機能を使用するには、接続をオープンするときに mysql_real_connect() のフラグパラメータで CLIENT_MULTI_STATEMENTS オプションを指定する必要があります。このオプションは、mysql_set_server_option(MYSQL_OPTION_MULTI_STATEMENTS_ON) を呼び出すことで、接続に対して設定することもできます。

デフォルトでは、mysql_query() および mysql_real_query() は最初のクエリのステータスだけを返します。その後のクエリのステータスは mysql_more_results() および mysql_next_result() を使用して処理できます。

```
/* Connect to server with option CLIENT_MULTI_STATEMENTS */
mysql_real_connect(..., CLIENT_MULTI_STATEMENTS);

/* Now execute multiple queries */
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";
do
{
  /* Process all results */
  ...
  printf("total affected rows: %lld", mysql_affected_rows(mysql));
  ...
  if (!(result= mysql_store_result(mysql)))
  {
    printf(stderr, "Got fatal error processing query\n");
    exit(1);
  }
  process_result_set(result); /* client function */
  mysql_free_result(result);
} while (!mysql_next_result(mysql));
```

11.1.9. C API における日付値および時刻値の取り扱い

MySQL 4.1 で導入された新しいバイナリプロトコルを使用すると、MYSQL_TIME 構造体を使用して日付および時刻に関する値 (DATE、TIME、DATETIME、および TIMESTAMP) を送受信できます。この構造体のメンバについては、項 11.1.5.「C API のプリペアドステートメントのデータ型」 を参照してください。

時間的なデータを送信するには、mysql_prepare() を使用してプリペアドステートメントを作成します。次に mysql_execute() を呼び出してステートメントを実行する前に、以下の手順を実行して時間的なパラメータを設定します。

1. データ値に関連付けられた MYSQL_BIND 構造体のメンバ buffer_type に、送信する時間的な値の型を示す値を設定する。DATE、TIME、DATETIME、または TIMESTAMP の値の場合、buffer_type に MYSQL_TYPE_DATE、MYSQL_TYPE_TIME、MYSQL_TYPE_DATETIME、または MYSQL_TYPE_TIMESTAMP をそれぞれ設定する。

2. MYSQL_BIND 構造体のメンバ buffer に、時間的な値が格納されている MYSQL_TIME 構造体のアドレスを設定する。

3. 時間的な値の型に合わせて MYSQL_TIME 構造体の必要なメンバの値を設定する。

mysql_bind_param() を使用してパラメータデータをステートメントにバインドします。これで mysql_execute() を呼び出す準備が完了します。

時間的な値を取得する手順はほとんど同じですが、buffer_type メンバには取得しようとする値の型を設定する点、および buffer メンバには返される値を格納する MYSQL_TIME 構造体のアドレスを設定する点が異なります。mysql_execute() を呼び出した後、結果を取得するまでの間に、mysql_bind_results() を呼び出して、バッファをステートメントにバインドします。

以下の例では、DATE、TIME、および TIMESTAMP の型のデータを挿入する方法を示します。mysql 変数は有効な接続ハンドルとします。

```
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)));

 /* setup input buffers for all 3 parameters */
 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);

 /* supply the data to be sent is the ts structure */
 ts.year= 2002;
 ts.month= 02;
 ts.day= 03;

 ts.hour= 10;
 ts.minute= 45;
 ts.second= 20;

 mysql_execute(stmt);
 ..
```

11.1.10. C API スレッド関数の説明

スレッドクライアントを作成するには、以下の関数を使用する必要があります。 See 項11.1.14.「スレッドクライアント

の作成方法」。

11.1.10.1. my_init()

void my_init(void)

説明

この関数は、他の MySQL 関数を呼び出す前に、プログラム内で 1 回だけ呼び出す必要があります。MySQL が必要とするいくつかのグローバル変数を初期化します。スレッドセーフなクライアントライブラリを使用している場合、この関数は現在のスレッドに対して mysql_thread_init() の呼び出しも行います。

mysql_init()、mysql_server_init()、および mysql_connect() は、自動的にこの関数を呼び出します。

戻り値

ありません。

11.1.10.2. mysql_thread_init()

my_bool mysql_thread_init(void)

説明

スレッドを作成したら、スレッド固有の変数を初期化するためにこの関数を呼び出す必要があります。

my_init() および mysql_connect() は、自動的にこの関数を呼び出します。

戻り値

正常に動作した場合は 0。エラーが発生した場合は 0 以外。

11.1.10.3. mysql_thread_end()

void mysql_thread_end(void)

説明

mysql_thread_init() を呼び出す前にこの関数を呼び出して、pthread_exit() によって割り当てられたメモリを解放する必要があります。

注意: この関数は、クライアントライブラリからは自動的に呼び出されません。メモリリークを避けるために、明示的に呼び出す必要があります。

戻り値

ありません。

11.1.10.4. mysql_thread_safe()

unsigned int mysql_thread_safe(void)

説明

この関数は、クライアントがスレッドセーフとしてコンパイルされたかどうかを示します。

戻り値

クライアントがスレッドセーフの場合は 1、それ以外の場合は 0。

11.1.11. C API 組み込みサーバ関数の説明

アプリケーションを組み込み MySQL サーバライブラリとリンクする場合、以下で説明する関数を使用する必要があります。 See 項11.1.15.「組み込み MySQL サーバライブラリ libmysqld」。

プログラムを、-lmysqld ではなく、-lmysqlclient を指定してリンクした場合、これらの関数は何も実行しません。そのため、コードを変更しなくても、組み込み MySQL サーバおよびスタンドアロンサーバのどちらを使用するかを選択できます。

11.1.11.1. mysql_server_init()

int mysql_server_init(int argc, char **argv, char **groups)

説明

この関数は、組み込みサーバを使用するプログラムで、他の MySQL 関数を呼び出す前に、1 回だけ呼び出す必要があります。まず組み込みサーバを起動し、このサーバが使用するサブシステム (mysys、InnoDB など) をすべて初期化します。この関数を呼び出さなかった場合、プログラムはクラッシュします。MySQL に付属の DBUG パッケージを使用している場合は、MY_INIT() を呼び出した後にこの関数を呼び出す必要があります。

引数の argc および argv は、main() の引数と似ています。argv の最初の要素は、通常はプログラム名であり、無視されます。使いやすくするために、サーバに対してコマンドライン引数がない場合、argc は 0 (ゼロ) になります。mysql_server_init() は引数のコピーを作成するので、それを呼び出した後は argv または groups を壊しても問題ありません。

groups の NULL で終端された文字列のリストは、オプション設定ファイルのどのグループをアクティブにするかを選択します。See 項4.1.2.「my.cnf オプション設定ファイル」。 使いやすくするために、groups が NULL の場合は、[server] および [embedded] グループが読み込まれます。

例

```
#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;
}
```

戻り値

正常に終了した場合は 0。エラーが発生した場合は 1。

11.1.11.2. mysql_server_end()

void mysql_server_end(void)

説明

この関数は、他の MySQL 関数をすべて呼び出した後に 1 回呼び出す必要があります。組み込みサーバをシャットダウンします。

戻り値

ありません。

11.1.12. C API の使用に関する一般的な質問および問題

11.1.12.1. mysql_query() が正常に終了した後で mysql_store_result() が NULL を返す場合があるのはなぜか

mysql_query() の呼び出しが正常に終了した後で mysql_store_result() が NULL を返す可能性はあります。このような状況が発生した場合、以下のいずれかの条件が成立したことを意味します。

- malloc() が異常終了した (結果セットが大きすぎた場合など)
- データを読み込めなかった (接続にエラーが発生した)
- データを返さないクエリだった (INSERT、UPDATE、または DELETE だった場合など)

mysql_field_count() を呼び出すことによって、ステートメントが空でない結果セットを生成したかどうかをいつでも調べることができます。mysql_field_count() が 0 を返す場合、結果セットは空であり、最後に実行したクエリは結果を返していません (INSERT または DELETE など)。mysql_field_count() が 0 以外の値を返す場合、ステートメントは空ではない結果セットを返しています。例については、mysql_field_count() 関数の説明を参照してください。

mysql_error() または mysql_errno() を呼び出すことによって、エラーが発生したかどうかを判定できます。

11.1.12.2. クエリから取得できる結果にはどのようなものがあるか

クエリは、結果セットを返す以外に、以下の情報を提供します。

- mysql_affected_rows() は、INSERT、UPDATE、または DELETE を実行する場合、最後に実行したクエリで挿入、更新、または削除されたレコードの数を返す。例外は、WHERE 節を使わずに DELETE を実行した場合で、このときは時間を大幅に短縮するためにテーブルを空の状態で再作成する。この場合、mysql_affected_rows() は削除されたレコードの数として 0 を返す。
- mysql_num_rows() は結果セットのレコード数を返す。mysql_store_result() を使用する場合、mysql_store_result() が復帰した直後に mysql_num_rows() を呼び出すことができる。mysql_use_result() を使用する場合、mysql_fetch_row() を使用してすべてのレコードを取得した後でのみ mysql_num_rows() を呼び出すことができる。
- mysql_insert_id() は、AUTO_INCREMENT インデックスを使用してテーブルにレコードを挿入した最後のクエリが生成した ID を返す。 See 項11.1.3.32.「mysql_insert_id()」。
- 一部のクエリ (LOAD DATA INFILE ...、INSERT INTO ... SELECT ...、UPDATE) は、上記以外にも補足情報を提供する。この補足情報は mysql_info() を呼び出すことによって取得する。この関数が返す文字列のフォーマットについては、mysql_info() の説明を参照すること。mysql_info() は、取得する補足情報がない場合は、NULL ポインタを返す。

11.1.12.3. 最後に挿入したレコードの一意な ID はどのように取得するのか

AUTO_INCREMENT 属性を持つカラムが定義されているテーブルにレコードを挿入する場合、mysql_insert_id() を呼び出すことによって、最後に生成された ID を取得できます。

また、mysql_query() に渡すクエリ文字列で LAST_INSERT_ID() 関数を使用することによって、最後に生成された ID を取得することもできます。

以下のコードを実行すると、AUTO_INCREMENT インデックスが使用されたかどうかを調べることができます。このコードは同時に、クエリが AUTO_INCREMENT インデックスを使用する INSERT だったかどうかも調べます。

```
if (mysql_error(&mysql)[0] == 0 &&
    mysql_num_fields(result) == 0 &&
    mysql_insert_id(&mysql) != 0)
{
    used_id = mysql_insert_id(&mysql);
}
```

最後に生成された ID は、接続ごとにサーバに保持されます。他の接続を使用するクライアントによって変更されることはありません。AUTO_INCREMENT 属性を持つ他のカラムを非マジック値 (NULL でも 0 でもない値) で更新した場合も、この ID は変更されません。

あるテーブル用に生成された ID を別のテーブルに挿入するには、以下の SQL ステートメントを記述します。

```
INSERT INTO foo (auto,text)
    VALUES(NULL,'text');            # generate ID by inserting NULL
INSERT INTO foo2 (id,text)
    VALUES(LAST_INSERT_ID(),'text');  # use ID in second table
```

11.1.12.4. C API とリンクする場合の問題

C API とリンクする場合、一部のシステムでは以下のエラーが発生する可能性があります。

```
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
```

システムでこのエラーが発生した場合、コンパイル/リンク行の最後に -lm を追加して、数学ライブラリをリンクする必要があります。

11.1.13. クライアントプログラムのビルド

自分で作成した MySQL クライアントまたはサードパーティから入手した MySQL クライアントをコンパイルする場合、リンクコマンドで -lmysqlclient -lz オプションを使用してリンクする必要があります。また、場合によっては、ライブラリの所在をリンカに通知するために -L オプションを指定する必要があります。たとえば、ライブラリが /usr/local/mysql/lib にインストールされている場合、リンクコマンドで -L/usr/local/mysql/lib -lmysqlclient -lz を記述します。

MySQL ヘッダファイルを使用するクライアントをコンパイルするとき、場合によっては -I オプションを指定して（たとえば -I/usr/local/mysql/include）、ヘッダファイルの所在をコンパイラに通知する必要があります。

上記の処理を Unix で簡単に実行するために mysql_config スクリプトが用意されています。 See 項4.9.11.「mysql_config（クライアントをコンパイルするためのコンパイルオプションの取得）」。

このスクリプトを使用して以下のように指定することで、MySQL クライアントをコンパイルできます。

```
CFG=/usr/local/mysql/bin/mysql_config
sh -c "gcc -o progname `$CFG --cflags` progname.c `$CFG --libs`"
```

sh -c は、mysql_config からの出力をシェルが 1 語として処理しないようにするために必要です。

11.1.14. スレッドクライアントの作成方法

クライアントライブラリはほぼスレッドセーフです。最大の問題は、net.c で記述されているソケットからデータを読み込むためのサブルーチンが、割り込みが発生したときにスレッドセーフではないことです。これは、サーバに対する読み込みに長時間かかる場合、それを中断できるアラームをユーザが独自に作成する可能性があるという考えに基づく仕様です。割り込み SIGPIPE に対する割り込みハンドラをインストールする場合は、ソケットの処理をスレッドセーフにする必要があります。

バージョン 4.0.16 の新機能: 接続が切断されたときにプログラムが強制終了しないようにするために、MySQL は最初に mysql_server_init()、mysql_init()、または mysql_connect() を呼び出すときに SIGPIPE をブロックします。独自に SIGPIPE に対する割り込みハンドラを作成する場合、まず mysql_server_init() を呼び出してから、そのハンドラをインストールする必要があります。MySQL の古いバージョンでは、スレッドセーフなクライアントライブラリでのみ、mysql_init() を呼び出すたびに、SIGPIPE がブロックされていました。

当社の Web サイト（http://www.mysql.com/）で配布した古いバイナリファイルの場合、クライアントライブラリのコンパイルではスレッドセーフなオプションを使用しないのが普通でした（Windows のバイナリはデフォルトでスレッドセーフでコンパイルされます）。新しく配布するバイナリでは、スレッドセーフなクライアントライブラリとそうでないクライアントライブラリの両方が必要です。

別のスレッドから割り込んだり、MySQL サーバとの通信にタイムアウトを設定できるようなスレッドクライアントを作成

するには、-lmysys、-lmystrings、および -ldbug の各ライブラリ、およびサーバが使用する net_serv.o コードを使用する必要があります。

割り込みやタイムアウトを使用しない場合は、スレッドセーフなクライアントライブラリ (mysqlclient_r) をコンパイルして使用するだけで済みます。See 項11.1.「MySQL C API」。 この場合、net_serv.o オブジェクトファイルや他の MySQL ライブラリを使用する必要はありません。

スレッドクライアントを使用しながら、タイムアウトや割り込みも使用する場合、thr_alarm.c ファイルで定義されているルーチンを活用できます。mysys ライブラリのルーチンを使用している場合、まず my_init() を呼び出す必要があります。See 項11.1.10.「C API スレッド関数の説明」。

mysql_real_connect() 以外のすべての関数は、デフォルトでスレッドセーフです。以下に、スレッドセーフなクライアントライブラリをコンパイルし、スレッドセーフに使用する方法を示します (mysql_real_connect() を使用した以下の記述は mysql_connect() にも同様に適用できますが、mysql_connect() は廃止されたので、いずれにしろ mysql_real_connect() を使用する必要があります)。

mysql_real_connect() をスレッドセーフにするには、以下のコマンドを使用してクライアントライブラリを再コンパイルする必要があります。

```
shell> ./configure --enable-thread-safe-client
```

このコマンドによって、スレッドセーフなクライアントライブラリ libmysqlclient_r が作成されます (使用する OS にスレッドセーフな gethostbyname_r() 関数が提供されていると仮定)。このライブラリは接続ごとにスレッドセーフになります。以下の項目に注意すれば、2 つのスレッドで同じ接続を共有できます。

- 2 つのスレッドが同じ接続で同時に MySQL サーバにクエリを送信することはできない。特に、mysql_query() を呼び出してから mysql_store_result() を呼び出すまでの間は、他のスレッドが同じ接続を決して使用しないようにする必要がある。
- 複数のスレッドが mysql_store_result() で取得した複数の結果セットにアクセスすることができる。
- mysql_use_result を使用する場合、結果セットをクローズするまで、同じ接続を他のスレッドが使用しないようにする必要がある。ただし、最も望ましいのは、同じ接続を共有するスレッドクライアントが mysql_store_result() を使用することである。
- 同じ接続で複数のスレッドを使用する場合、mysql_query() を呼び出す前から mysql_store_result() を呼び出した後までの部分を mutex ロックで囲む必要がある。mysql_store_result() が復帰した後はロックを解放することができ、同じ接続で他のスレッドがクエリを実行できるようになる。
- POSIX スレッドを使用してプログラムする場合、pthread_mutex_lock() および pthread_mutex_unlock() を使用して、mutex ロックを設定および解放できる。

MySQL データベースに接続しない MySQL 関数を呼び出すスレッドを使用する場合、以下の事項を認識しておく必要があります。

mysql_init() または mysql_connect() を呼び出すと、MySQL はスレッドごとにスレッド固有の変数を作成します。この変数は特にデバッグライブラリで使用されます。

スレッドが mysql_init() または mysql_connect() を呼び出す前に MySQL 関数を呼び出すと、スレッドは必要なスレッド固

有の変数を設定せず、そのうちコアダンプが発生します。

処理を円滑に進めるには、以下に示す項目を実行する必要があります。

1. mysql_real_connect() を呼び出す前に他の MySQL 関数を呼び出す場合、プログラムの先頭で my_init() を呼び出す。
2. MySQL 関数を呼び出す前にスレッドハンドラで mysql_thread_init() を呼び出す。
3. スレッドでは、pthread_exit() を呼び出す前に mysql_thread_end() を呼び出す。これにより MySQL が作成したスレッド固有の変数が使用していたメモリが解放される。

クライアントを libmysqlclient_r とリンクするときに、未定義シンボルがあるというエラーが発生する可能性があります。ほとんどの場合、これはリンク/コンパイル行にスレッドライブラリを追加しなかったことが原因です。

11.1.15. 組み込み MySQL サーバライブラリ libmysqld

11.1.15.1. 組み込み MySQL サーバライブラリの概要

組み込み MySQL サーバライブラリを使用することで、クライアントアプリケーション内部で完全な機能を備えた MySQL サーバを実行できます。最大の長所は、処理速度が高速になることと、組み込みアプリケーションの管理性が向上することです。

組み込み MySQL サーバライブラリは、C/C++ で記述された MySQL のクライアント/サーババージョンをベースとしています。そのため、組み込みサーバも C/C++ で記述されています。他の言語での組み込みサーバは提供されていません。

組み込みサーババージョンとクライアント/サーババージョンはどちらも同じ API が用意されています。既存のスレッドアプリケーションも、通常は以下の関数への呼び出しを追加するだけで、組み込みサーバライブラリを使用するように変更できます。

関数	呼び出すタイミング
mysql_server_init()	他の MySQL 関数が呼び出される前に呼び出す必要がある。main() 関数のできるだけ先頭に近い場所が望ましい。
mysql_server_end()	プログラムが終了する前に呼び出す必要がある。
mysql_thread_init()	MySQL にアクセスするスレッドごとに呼び出す必要がある。
mysql_thread_end()	pthread_exit() を呼び出す前に呼び出す必要がある。

次に、libmysqlclient.a ではなく、libmysqld.a を作成したコードにリンクする必要があります。

上記の mysql_server_xxx 関数は libmysqlclient.a にも含まれており、作成したアプリケーションを適切なライブラリとリンクするだけで、組み込みサーババージョンおよびクライアント/サーババージョンのどちらを使用するかを切り替えることができるようになっています。 See 項11.1.11.1.「mysql_server_init()」。

11.1.15.2. libmysqld を使用したプログラムのコンパイル

libmysqld ライブラリを取得するには、--with-embedded-server オプションを使用して MySQL を設定する必要があります。

プログラムを libmysqld とリンクするときは、システム固有の pthread ライブラリおよび MySQL サーバが使用するライブラリもリンクする必要があります。mysql_config --libmysqld-libs を実行すると、ライブラリの詳細なリストを取得できます。

スレッドプログラムをコンパイルおよびリンクするときは、適切なフラグを指定する必要があります。これは、自作コードから直接スレッド関数を呼び出していない場合でも必要です。

11.1.15.3. 組み込み MySQL サーバを使用する際の制約

組み込み MySQL サーバには以下の制約があります。

- ISAM テーブルはサポートしない（ライブラリのサイズを小さくするための方針）。
- ユーザ定義関数（UDF）は使用できない。
- コアダンプでスタックトレースはできない。
- 内部 RAID をサポートしない（現在ほとんどの OS でサイズの大きいファイルをサポートしているので通常は必要ない）。
- サーバまたはマスタとして設定できない（レプリケーションできない）。
- 外部プロセスからソケットまたは TCP/IP を使用して組み込み MySQL サーバに接続することはできない。

上記の制約の中には、mysql_embed.h インクルードファイルを編集して MySQL を再コンパイルすることで変更できるものもあります。

11.1.15.4. 組み込み MySQL サーバでのオプション設定ファイルの使用

オプション設定ファイルを使用して、クライアント/サーバアプリケーションと組み込み MySQL サーバを使用したアプリケーションを簡単に切り替える推奨方法を以下に示します。 See 項4.1.2.「my.cnf オプション設定ファイル」。

- 共通するオプションを [server] セクションに記述する。このオプションは両方のバージョンの MySQL に読み込まれる。
- クライアント/サーバ固有のオプションを [mysqld] セクションに記述する。
- 組み込み MySQL サーバ固有のオプションを [embedded] セクションに記述する。
- アプリケーション固有のオプションを [ApplicationName_SERVER] セクションに記述する。

11.1.15.5. 組み込み MySQL サーバの今後の開発課題

- ライブラリを小さくするために MySQL の機能の一部を組み込まないオプションを提供する。
- 実行速度を向上させる。
- エラーの出力先を stderr にする。その場合にファイル名を指定するオプションを追加する。

- InnoDB を組み込み MySQL サーババージョンで使用する際の冗長度を削減する。

11.1.15.6. 組み込み MySQL サーバを使用した簡単な例

以下に示すプログラムおよび makefile は、Linux や FreeBSD システムでも何も変更することなく動作するはずです。他のオペレーティングシステムで使用する場合は、一部変更が必要です。この例は、実際のアプリケーションを作成する場合には避けられない混乱を回避して、問題点を理解できるだけの詳細な実例を示すことを目的としています。

この例を実際に動作させるには、mysql-4.0 ソースディレクトリと同じレベルに test_libmysqld ディレクトリを作成します。そのディレクトリに test_libmysqld.c ソースファイルと GNUmakefile を保存し、test_libmysqld ディレクトリ内から GNU make を実行します。

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() must be called before any other mysql
   * functions.
   *
   * You can use mysql_server_init(0, NULL, NULL), and it will
   * initialize the server using groups = {
   *   "server", "embedded", NULL
   * }.
   *
   * In your $HOME/.my.cnf file, you probably want to put:

[test_libmysqld_SERVER]
language = /path/to/source/of/mysql/sql/share/english

   * You could, of course, modify argc and argv before passing
   * them to this function.  Or you could create new ones in any
   * way you like.  But all of the arguments in argv (except for
   * argv[0], which is the program name) should be valid options
   * for the MySQL server.
   *
```

```
  * If you link this client against the normal mysqlclient
  * library, this function is just a stub that does nothing.
  */
 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);

 /* This must be called after all other mysql functions */
 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");
 /*
  * Notice that the client and server use separate group names.
  * This is critical, because the server will not accept the
  * client's options, and 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);
    mysql_free_result(res);
  }
  else
    (void)printf("Affected rows: %lld\n", mysql_affected_rows(db));

  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$(inc) -D_THREAD_SAFE -D_REENTRANT
CFLAGS   := -g -W -Wall
LDFLAGS  := -static
# You can change -lmysqld to -lmysqlclient to use the
# client/server library
LDLIBS    = -L$(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
```

11.1.15.7. 組み込み MySQL サーバのライセンス

MySQL のソースコードは GNU GPL ライセンス (see 付録 H. GNU General Public License) の対象です。その結果、libmysqld をリンクすることによって MySQL ソースコードをインクルードするプログラムは、(GPL と互換性のあるライセンスの下で) フリーソフトウェアとしてリリースする必要があります。

フリーソフトウェアを作成したら、GPL またはそれと互換性のあるライセンスの下でコードをリリースすることによってフリーソフトウェアを宣伝するようにしてください。それができない場合は、MySQL AB から MySQL コードの商業的ライセンスを購入するという選択肢があります。詳細については、項1.4.3.「MySQL ライセンス」 を参照してください。

11.2. MySQL の ODBC サポート

MySQL は、MyODBC プログラムという形で ODBC をサポートします。この章では、MyODBC のインストール方法およびその使用方法について説明します。また、MyODBC と連携して動作することが知られている共通プログラムのリストも提供します。

11.2.1. MyODBC のインストール方法

MyODBC 2.50 は、ODBC 対応アプリケーションが MySQL に接続するための 32 ビット ODBC 2.50 仕様レベル 0 (レベル 1 およびレベル 2 の機能を含む) のドライバです。MyODBC は、Windows 9x/Me/NT/2000/XP およびほとんどの Unix プラットフォームで動作します。MyODBC 3.51 は、ODBC 3.5x 仕様レベル 1 (完全なコア API およびレベル 2 の機能を含む) の拡張バージョンです。

MyODBC は Open Source であり、http://www.mysql.com/downloads/api-myodbc.html で最新バージョンが公開されています。注意: 2.50.x バージョンには LGPL ライセンスが適用され、3.51.x バージョンには GPL ライセンスが適用されます。

MyODBC を使用して問題が発生する場合、それが OLEDB と連携して動作するプログラムであれば、OLEDB ドライバを試す必要があります。

通常は、Windows マシンに MyODBC をインストールする必要があるだけです。Unix で MyODBC が必要になるのは、ColdFusion のように、Unix マシン上で動作しながら ODBC を使用してデータベースに接続するプログラムを実行する場合だけです。

Unix マシンに MyODBC をインストールする場合は、ODBC マネージャも必要です。MyODBC は、ほとんどの Unix ODBC マネージャで動作することが知られています。

Windows マシンに MyODBC をインストールするには、適切な MyODBC .zip ファイルをダウンロードし、WinZip などのプログラムで解凍して、SETUP.EXE ファイルを実行します。

Windows/NT/XP マシンでは、MyODBC をインストールする際に以下のエラーが発生する可能性があります。

```
An error occurred while copying C:\WINDOWS\SYSTEM\MFC30.DLL. Restart
Windows and try installing again (before running any applications which
use ODBC)
```

この場合の問題は、他の何らかのプログラムが ODBC を使用中であり、Windows の設計方針として、Microsoft の ODBC セットアッププログラムで新しい ODBC ドライバをインストールできない可能性があることです。ほとんどの場合、Ignore をクリックするだけで残りの MyODBC ファイルのコピーを続行することができ、インストールされたプログラムも正常に動作します。正常に動作しなかった場合の解決策としては、コンピュータを ``セーフモード" でリブートし (リブート中に Windows が立ち上がる前に F8 キーを押す) 、MyODBC をインストールしてから、通常モードでリブートします。

- Unix マシンから Windows マシンに ODBC アプリケーション (MySQL を本来サポートしないアプリケーション) を使用して接続するには、まず Windows マシンに MyODBC をインストールする必要がある。
- ユーザおよび Windows マシンには、Unix マシン上の MySQL サーバへのアクセス権が必要である。このアクセス権を与えるには、GRANT コマンドを使用する。 See 項4.4.1.「GRANT および REVOKE の構文」。
- 以下の手順に従って、ODBC DSN エントリを作成する必要がある。
  - Windows マシンで [コントロールパネル] を開く。
  - [ODBC Data Sources 32-bit] アイコンをダブルクリックする。
  - [User DSN] タブをクリックする。
  - [Add] をクリックする。
  - [Create New Data Source] ダイアログボックスで [MySQL] を選択し、[Finish] をクリックする。
  - [MySQL Driver default configuration] ダイアログボックスが表示される。 See 項11.2.2.「ODBC アドミニストレータのフィールドの設定方法」。
- アプリケーションを起動して、ODBC アドミニストレータで指定した DSN と ODBC ドライバを選択する。

注意: MySQL の画面には上記以外にも設定オプション (トレース、接続時にプロンプトを表示しないなど) があり、問題が発生した場合に使用することができます。

11.2.2. ODBC アドミニストレータのフィールドの設定方法

Windows 95 ではサーバ名を指定する方法が 3 つ考えられます。

- サーバの IP アドレスを使用する。
- 以下の情報を \windows\lmhosts ファイルに追加する。

  ```
  ip hostname
  ```

  たとえば、以下のように指定する。

```
194.216.84.21 my_hostname
```

- DNS を使用するように PC を設定する。

ODBC setup の入力例を示します。

```
Windows DSN name:   test
Description:        This is my test database
MySQL Database:     test
Server:             194.216.84.21
User:               monty
Password:           my_password
Port:
```

Windows DSN name フィールドの値には、Windows の ODBC 設定で一意な名前を指定します。

ODBC setup ダイアログボックスの Server、User、Password、または Port の各フィールドは、入力必須フィールドではありません。ただし、これらのフィールドに値を入力した場合、その値は後で接続する際にデフォルトとして使用されます。その時点で、そのデフォルト値を変更することができます。

ポート番号が指定されなかった場合、デフォルトポート（3306）が使用されます。

Read options from C:\my.cnf を有効にした場合、C:\my.cnf ファイルから client グループおよび odbc グループのデータが読み込まれます。mysql_options() が使用可能なすべてのオプションを使用できます。 See 項11.1.3.40.「mysql_options()」。

11.2.3. MyODBC の接続パラメータ

ODBC.INI ファイルの [Servername] セクションで以下に示す MyODBC のパラメータを指定できます。これらのパラメータは、SQLDriverConnect() を呼び出すときに渡す引数 InConnectionString でも指定できます。

パラメータ	デフォルト値	コメント
user	ODBC（Windows 上）	MySQL に接続する際に使用するユーザ名
server	localhost	MySQL サーバのホスト名
database		デフォルトのデータベース
option	0	MyODBC の動作を指定する整数（下記参照）
port	3306	server が localhost 以外の値の場合に使用する TCP/IP ポート
stmt		MySQL に接続する際に実行するステートメント
password		server と user の組み合わせに対するパスワード
socket		接続するソケットまたは Windows パイプ

option 引数は、クライアントが 100% ODBC 準拠ではないことを MyODBC に通知するために使用します。Windows では、通常、接続画面で複数の選択肢を切り替えることによってオプションフラグを設定しますが、option 引数で設定することもできます。以下に、MyODBC 接続画面に表示されるのと同じ順序でオプションのリストを示します。

ビット	説明
1	クライアントは MyODBC がカラムの実際の幅を返すことに対応していない。
2	クライアントは MySQL が実際に影響を受けたレコードの数を返すことに対応していない。このフラグが設定されている場合、MySQL は影響を受けたレコードではなく、見つかったレコードを返す。MySQL が実際に影響を受けたレコードの数を返すことに対応するには、MySQL 3.21.14 以降を使用する必要がある。
4	デバッグログを c:\myodbc.log に書き込む。これは、AUTOEXEC.BAT に MYSQL_DEBUG=d:t:O,c:\myodbc.log を追加することと同じである。
8	結果およびパラメータにパケット制限を設定しない。
16	ドライバが入力を要求する画面の表示を必要としても、それを表示しない。
32	状況に応じて ODBC 1.0 ドライバをシミュレートする。
64	'database.table.column' でデータベース名の使用を無視する。
128	ODBC マネージャカーソルの使用を強制する（実験的）。
256	拡張取得の使用を無効にする（実験的）。
512	CHAR 型フィールドではカラム幅全体に文字をパッドする
1024	SQLDescribeCol() は完全修飾カラム名を返す。
2048	圧縮されたサーバ/クライアントプロトコルを使用する。
4096	サーバに、関数名の後および '(' の前の空白を無視するように通知する（PowerBuilder で必要）。これによりすべての関数名をキーワードにする。
8192	NT 上で動作する mysqld サーバに名前付きパイプを使用して接続する。
16384	LONGLONG 型のカラムを INT 型のカラムに変更する（一部のアプリケーションでは LONGLONG 型に対応していない）。
32768	SQLTables から Table_qualifier および Table_owner として 'user' を返す（実験的）。
65536	my.cnf の client グループおよび odbc グループからパラメータを読み込む。
131072	安全性チェックを追加する（その必要はないかもしれないが念のため）。

複数のオプションを選択する場合は上記のフラグを加算します。たとえば、オプションを 12（4+8）に設定すると、パッケージの制約を受けずにデバッグできます。

最適なパフォーマンスを得るために、デフォルトで MYODBC.DLL がコンパイルされます。MyODBC をデバッグする場合（たとえばトレース機能を有効にする）、代わりに MYODBCD.DLL を使用する必要があります。このファイルをインストールするには、インストールされている MYODBC.DLL ファイルに MYODBCD.DLL を上書きコピーします。

11.2.4. MyODBC に関する問題を報告する方法

MyODBC は、Access、Admndemo.exe、C++-Builder、Borland Builder 4、Centura Team Developer（旧 Gupta SQL/Windows）、ColdFusion（Solaris および NT 上でサービスパック 5 を使用）、Crystal Reports、DataJunction、Delphi、ERwin、Excel、iHTML、FileMaker Pro、FoxPro、Notes 4.5/4.6、SBSS、Perl DBD-ODBC、Paradox、Powerbuilder、Powerdesigner 32 ビット、VC++、および Visual Basic との動作を試験しています。

MyODBC と連携して動作するアプリケーションを他にご存知の方は、myodbc メーリングリストにその旨を投稿してください。 See 項1.7.1.1.「MySQL メーリングリスト」。

一部のプログラムでは Another user has modifies the record that you have modified のようなエラーが発生する可能性があります。. ほとんどの場合、これは以下のいずれかを実行することで解決できます。

- 主キーが存在しない場合、それをテーブルに追加する。
- タイムスタンプ型のカラムが存在しない場合、それを追加する。
- 倍精度浮動小数点型のフィールドだけを使用する。一部のプログラムでは単精度浮動小数点の比較でエラーが発生する可能性がある。

上記のどれを実行しても問題が解決しなかった場合、MyODBC トレースファイルを有効にして、問題が発生する原因を調べる必要があります。

11.2.5. MyODBC と連携して動作することが知られているプログラム

ほとんどのプログラムは MyODBC と連携して動作しますが、ここに示したプログラムについては、当社が社内で試験したか、またはユーザから動作確認したという連絡を受けています。

- プログラム

  コメント

- Access

  Access を動作させるには、以下の処理を実行する。

  - Access 2000 を使用している場合、http://www.microsoft.com/data/ から最新 (バージョン 2.6 以降) の Microsoft MDAC (Microsoft Data Access Components) をダウンロードし、インストールする必要がある。これによって、MySQL にデータをエクスポートする際にテーブル名およびカラム名が指定されないという Access のバグが修正される。このバグは、MyODBC バージョン 2.50.33 および MySQL バージョン 3.23.x にアップグレードすることによって回避することもできる。

    その他に、http://support.microsoft.com/support/kb/articles/Q 239/1/14.ASP で公開されている Microsoft Jet 4.0 サービスパック 5 をダウンロードし、適用する必要もある。これは、Access でカラムが #deleted# とマークされる場合があるバグを修正する。

    注意: MySQL バージョン 3.22 を使用している場合、この問題を回避するには、MDAC パッチを適用し、MyODBC 2.50.32 または 2.50.34 以降を使用する必要がある。

  - Access のすべてのバージョンで、MyODBC のオプションフラグ Return matching rows を有効にする必要がある。Access 2.0 では、さらに Simulate ODBC 1.0 を有効にする必要がある。
  - 更新可能にするすべてのテーブルでタイムスタンプ型のカラムを定義する必要がある。移植性を最大にするために、TIMESTAMP(14) または単純な TIMESTAMP を、TIMESTAMP(X) の代わりに使用することが推奨される。
  - テーブルに主キーを定義する必要がある。これを定義しない場合、新しく追加したレコードまたは更新したレコードが #DELETED# として表示される可能性がある。
  - DOUBLE 浮動小数点型のフィールドだけを使用する。Access では、単精度浮動小数点型の値を比較できない。そ

の結果、表面上は、新しく追加したレコードまたは更新されたレコードが #DELETED# として表示されたり、レコードを検索または更新できないという現象が発生する。

- BIGINT 型のカラムを持つテーブルを MyODBC 経由でリンクしている場合、結果が #DELETED として表示される。以下に回避策を示す。
  - データ型が TIMESTAMP、できれば TIMESTAMP(14) のダミーカラムを 1 つ追加する。
  - ODBC DSN アドミニストレータの接続オプションを指定するダイアログボックスで、'Change BIGINT columns to INT' をオンにする。
  - Access からテーブルリンクを削除し、再作成する。

  上記の回避策を実行した後も、既存のレコードは #DELETED# として表示されたままだが、新しく追加または更新されたレコードは正しく表示されるようになる。
- TIMESTAMP 型カラムを追加した後でもエラー Another user has changed your data が発生する場合、次の対策で解決する可能性がある。

  テーブルデータシートビューを使用しない。代わりに必要なフィールドを定義したフォームを作成し、フォームデータシートビューを使用する。TIMESTAMP 型カラムの DefaultValue プロパティを NOW() に設定する必要がある。このテーブルを使用するユーザが混乱しないように、TIMESTAMP 型カラムを非表示にすることも 1 つの方法である。
- 場合によっては、Access が、MySQL が解釈できない不正な SQL クエリを生成する可能性がある。これは、Access のメニューから "Query|SQLSpecific|Pass-Through" を選択することによって修正できる。
- NT 上で動作する Access が BLOB 型カラムを OLE OBJECTS 型として報告する。代わりに MEMO 型カラムを使用する場合は、ALTER TABLE を使用してカラムを TEXT 型に変更する必要がある。
- Access が DATE 型カラムを正しく処理できない場合がある。この問題が発生する場合、カラムを DATETIME 型に変更する。
- Access で BYTE 型として定義されているカラムを使用している場合、Access はこのカラムを、TINYINT UNSIGNED 型ではなく、TINYINT 型としてエクスポートしようとする。この場合、カラムに 127 より大きい値が格納されていると問題が発生する。

- ADO

  ADO API および MyODBC を使用してコーディングする場合、MySQL サーバでサポートされない一部のデフォルトプロパティに注意する必要がある。たとえば、CursorLocation プロパティに adUseServer が設定されている場合、RecordCount プロパティは -1 を返す。正しい値が返されるようにするには、以下の VB コードで示すように、このプロパティを adUseClient に設定する必要がある。

```
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
```

他の回避策としては、同じようなクエリで SELECT COUNT(*) ステートメントを使用して適切なレコード数を取得する方法がある。

- Active server pages（ASP）

  オプションフラグ Return matching rows を使用する必要がある。

- BDE アプリケーション

  オプションフラグ Don't optimize column widths および Return matching rows を設定する必要がある。

- Borland Builder 4

  クエリを開始するとき、Active プロパティまたは Open メソッドを使用できる。注意: Active は、自動的に SELECT * FROM ... クエリを発行してクエリを開始するが、テーブルが大きい場合は問題になる場合がある。

- ColdFusion（Unix 上）

  以下の情報は ColdFusion のマニュアルからの抜粋である。

  以下の情報を使用して、MySQL データソースに対して MyODBC とともに unixODBC ドライバを使用するように ColdFusion Server for Linux を設定する。Allaire は、MyODBC バージョン 2.50.26 が MySQL バージョン 3.22.27 および ColdFusion for Linux と連携して動作することを確認済みである（それ以降のバージョンも当然動作する）。MyODBC は、http://www.mysql.com/downloads/api-myodbc.html からダウンロードできる。

  ColdFusion バージョン 4.5.1 を使用すると、ColdFusion Administrator を使用して MySQL データソースを追加できる。ただし、ColdFusion バージョン 4.5.1 にはドライバは付属していない。MySQL ドライバが ODBC データソースのドロップダウンリストに表示されるようにするには、MyODBC ドライバをビルドして、/opt/coldfusion/lib/libmyodbc.so にコピーする必要がある。

  Contrib ディレクトリには mydsn-xxx.zip があり、これを使用して ColuFusion アプリケーションで MyODBC ドライバの DSN レジストリファイルをビルドおよび削除できる。

- DataJunction

  DataJunction では、エクスポートで ENUM が出力されて、MySQL でトラブルの原因となるので、代わりに VARCHAR を出力するように設定を変更する必要がある。

- Excel

  正常に動作する。以下にヒントを示す。

  - 日付データで問題が発生する場合、CONCAT() 関数を使用して日付データを文字列として選択する。たとえば、以

下のように指定する。

```
SELECT CONCAT(rise_time), CONCAT(set_time)
  FROM sunrise_sunset;
```

この方法で文字列として取得した値は、Excel97 で正しく時刻として認識される。

この例で CONCAT() を使用したのは、ODBC にカラムが ``文字列型" であると思わせるためである。CONCAT() を使用しないと、ODBC はカラムが時刻型であることを認識するが、Excel はそれを認識しない。

注意: Excel は自動的に文字列を時刻データに変換するので、上記の件は Excel のバグである。ソースがテキストファイルであれば、この自動変換は非常に便利だが、ソースがカラムごとのデータ型を正しく報告する ODBC 接続である場合はまったく意味をなさない。

- Word

MySQL から Word または Excel ドキュメントにデータを取得するには、MyODBC ドライバおよび Microsoft Query アドインを使用する必要がある。

たとえば、データベースに、2 つのテキスト型カラムを持つテーブルがあるとする。

- mysql クライアントコマンドラインツールを使用してレコードを挿入する。
- ODBC マネージャを使用して、上記のデータベースに対して、たとえば my という DSN ファイルを作成する。
- Word を開く。
- 新規ドキュメントを作成する。
- [Database] ツールバーの [Insert Database] をクリックする。
- [Get Data] をクリックする。
- [Get Data] ダイアログボックスの右側にある [MS Query] をクリックする。
- [MS Query] で DSN ファイル my を使用して新しいデータソースを作成する。
- 新しいクエリを選択する。
- 取得するカラムを選択する。
- 必要ならフィルタを作成する。
- 必要なら並べ替えを指定する。
- [Return Data to Microsoft Word] を選択する。
- [Finish] をクリックする。
- [Insert data] をクリックしてレコードを選択する。
- [OK] をクリックする。Word ドキュメントに選択したレコードが表示される。

- odbcadmin

  ODBC のテストプログラム。

- Delphi

  BDE バージョン 3.2 以降を使用する必要がある。MySQL に接続する場合、オプションフィールド Don't optimize column width を有効にする。

  また、MyODBC の ODBC エントリおよび BDE エントリ（BDE エントリは Delphi Super Page で無料公開されている BDE Alias Editor を必要とする）をセットアップする便利な Delphi コードを以下に示す（これについては Bryan Brunton <bryan@flesherfab.com> に感謝する）。

```
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

  BDE バージョン 3.0 を使用して試験済み。唯一の既知の問題は、テーブルスキーマが変更されたときにクエリのフィールドが更新されないことである。ただし、BDE はインデックスの PRIMARY 以外の主キーを認識しないと思われるが、これはこれまで問題にはなっていない。

- Vision

  オプションフラグ Return matching rows を使用する必要がある。

- Visual Basic

  テーブルを更新できるようにするには、テーブルに主キーを定義する必要がある。

  Visual Basic を ADO とともに使用した場合、大きなサイズの整数を処理できない。これは、SHOW PROCESSLIST のようなクエリは正しく動作しないことを意味する。これを解決するには、ODBC 接続文字列にオプション OPTION=16384 を設定するか、または MyODBC 接続画面で Change BIGINT columns to INT を選択する。場合によっては、Return matching rows も選択する必要がある。

- VisualInterDev

  [Microsoft][ODBC Driver Manager] Driver does not support this parameter というエラーが発生する場合、結果セットに BIGINT が含まれていることが原因である可能性がある。MyODBC 接続画面で Change BIGINT columns to INT を選択すると解決する可能性がある。

- Visual Objects

  オプションフラグ Don't optimize column widths を使用する必要がある。

11.2.6. ODBC で AUTO_INCREMENT 属性を持つカラムの値を取得する方法

よくある問題として、INSERT を実行したときに自動生成される ID の値をどのように取得するかという問題があります。ODBC を使用する場合、以下のようなステートメントによって、自動生成される ID を取得できます（auto は AUTO_INCREMENT 属性を持つフィールドとする）。

```
INSERT INTO foo (auto,text) VALUES(NULL,'text');
SELECT LAST_INSERT_ID();
```

自動生成された ID を他のテーブルに挿入するだけであれば、以下のようなステートメントを実行します。

```
INSERT INTO foo (auto,text) VALUES(NULL,'text');
INSERT INTO foo2 (id,text) VALUES(LAST_INSERT_ID(),'text');
```

See 項11.1.12.3.「最後に挿入したレコードの一意な ID はどのように取得するのか」。

一部の ODBC アプリケーション（少なくとも Delphi および Access）では、以下のクエリを使用して、新しく挿入されたレコードを検索できます。

```
SELECT * FROM tbl_name WHERE auto IS NULL;
```

11.2.7. MyODBC に関する問題の報告

MyODBC を使用して問題が発生した場合、まず MyODBC ログを作成すること、および ODBC マネージャからログファイル（ODBCADMIN で要求すると作成されるログ）を作成することから始めます。

MyODBC ログを取得するには、以下の手順に従います。

1. myodbc.dll ではなく、myodbcd.dll を使用していることを確認する。確認する最も簡単な方法は、MyODBC ディストリビューションから myodbcd.dll を取得して、C:\windows\system32 または C:\winnt\system32 にあると思われる myodbc.dll に上書きでコピーすることである。

   注意: 元の myodbc.dll は myodbcd.dll よりもはるかに高速なので、試験が終わったら元の myodbc.dll に戻すことが望ましい。

2. MyODBC 接続画面または設定画面でオプションフラグ `Trace MyODBC' を有効にする。ログが C:\myodbc.log ファイルに書き込まれる。

   上記の画面を再表示するとトレースオプションが無効になってしまう場合は、myodbcd.dll ドライバが使用されていないことを意味する (上記の項目参照) 。

3. アプリケーションを起動して、異常終了させる。

MyODBC trace file を調べて、おかしいと思われる部分を探します。myodbc.log ファイルで文字列 >mysql_real_query を検索すると、クエリを発行した部分が見つかります。

また、mysql モニタまたは admndemo でクエリを重複して実行し、エラーが MyODBC で発生しているのか、MySQL で発生しているのかを調べる必要があります。

おかしいと思われる部分が見つかった場合、関連する行だけ (最大 40 行) を myodbc メーリングリストに投稿してください。See 項1.7.1.1.「MySQL メーリングリスト」。 決して MyODBC ログファイルや ODBC ログファイルを丸ごと投稿しないでください。

おかしいと思われる部分が見つからなかった場合、最後の手段として、MyODBC トレースファイル、ODBC ログファイル、および問題点を説明する README ファイルを含むアーカイブ (tar または zip) を作成します。このアーカイブを ftp://support.mysql.com/pub/mysql/secret/ にアップロードします。このファイルには MySQL AB の担当者だけがアクセスできます。また、担当者とこれらのデータは完全に分離されています。

この問題が発生するプログラムを他にも作成できる場合は、それもアップロードしてください。

同じプログラムを他の SQL サーバに対して実行すると正常に動作する場合は、そのときの ODBC ログファイルを作成してください。

提供していただける情報が多いほど、その問題を解決できる可能性も高まりますので、できるだけ多くの情報を提供してください。

11.3. MySQL の Java 接続 (JDBC)

MySQL では 2 つの JDBC ドライバをサポートしています。

- MySQL AB 製の MySQL Connector/J。完全にネイティブな Java で実装されている。この製品は、以前は mm.mysql ドライバとして知られていた。MySQL Connector/J は、http://www.mysql.com/products/connector-j/ からダウンロードできる。
- Resin JDBC ドライバ。http://www.caucho.com/projects/jdbc-mysql/index.xtp からダウンロードできる。

マニュアルについては、JDBC のマニュアルおよび MySQL 固有の機能に関する各ドライバのマニュアルを参照してくだ

さい。

11.4. MySQL PHP API

PHP は、HTML 埋め込み型のサーバサイドスクリプト言語で、動的な Web ページの作成に使用できます。PHP は、MySQL をはじめとする複数のデータベースへのアクセス機能をサポートしています。独立したプログラムとして動作させることもできるし、Apache Web サーバで使用するモジュールとしてコンパイルすることもできます。

PHP Web サイト (http://www.php.net/) でディストリビューションおよびマニュアルが公開されています。

11.4.1. MySQL および PHP のよくある問題

- エラー: "Maximum Execution Time Exceeded" これは PHP の制約です。php3.ini ファイルを開いて、必要に応じて最大実行時間を 30 秒からもっと長い時間に設定してください。1 スクリプトあたりで使用できる RAM を 8MB から 16MB に倍増させる方法も考えられます。
- エラー: "Fatal error: Call to unsupported or undefined function mysql_connect() in .." これは、使用している PHP をコンパイルしたときに MySQL のサポートが組み込まれなかったことを意味します。MySQL のダイナミックモジュールをコンパイルして PHP にロードするか、または組み込み MySQL サポートとともに PHP を再コンパイルすることで解決できます。この詳細については、PHP マニュアルを参照してください。
- エラー: "undefined reference to `uncompress'" これは、クライアントライブラリをコンパイルしたときに、クライアント/サーバプロトコルの zlib サポートが組み込まれていたことを意味します。-lmysqlclient を指定してリンクするときに、最後に -lz を追加することで解決できます。

11.5. MySQL Perl API

ここでは、Perl DBI インタフェースについて説明します。以前のインタフェースは、mysqlperl と呼ばれていました。DBI/現在は DBD が Perl インタフェースとして推奨されています。mysqlperl は廃止され、ここでは説明しません。

11.5.1. DBI と DBD::mysql

DBI は、さまざまなデータベース向けの汎用インタフェースです。これは、何も変更しなくても、複数の異なるデータベースで動作するスクリプトを作成できることを意味します。データベースの種類ごとにデータベースドライバ (DBD) を定義する必要があります。MySQL の場合、このドライバは DBD::mysql と呼ばれます。

Perl5 DBI の詳細については、DBI の Web ページにあるマニュアルを参照してください。

```
http://dbi.perl.org/
```

注意: Perl でトランザクションを使用する場合、DBD-mysql バージョン 1.2216 以降が必要です。バージョン 2.1022 以降が推奨されます。

MySQL Perl サポートのインストール手順については、項2.7.「Perl インストールについてのコメント」 を参照してください。

MySQL モジュールをインストールしている場合、以下のどちらかのコマンドを使用して、特定の MySQL 機能に関する情

報を入手できます。

```
shell> perldoc DBD/mysql
shell> perldoc mysql
```

11.5.2. DBI インタフェース

移植可能な DBI メソッドおよび属性

メソッド/属性	説明
connect	データベースサーバへの接続を確立する。
disconnect	データベースサーバへの接続を切断する。
prepare	SQL ステートメントを実行するためのプリコンパイルを行う。
execute	プリペアドステートメントを実行する。
do	SQL ステートメントをプリコンパイルし、実行する。
quote	挿入する文字列または BLOB 値を引用符で囲む。
fetchrow_array	フィールドの配列として次のレコードを取得する。
fetchrow_arrayref	フィールドの参照配列として次のレコードを取得する。
fetchrow_hashref	ハッシュテーブルへの参照として次のレコードを取得する。
fetchall_arrayref	配列の配列としてすべてのデータを取得する。
finish	ステートメントの使用を完了して、リソースを解放する。
rows	影響を受けたレコードの数を返す。
data_sources	localhost で使用できるデータベースの配列を返す。
ChopBlanks	fetchrow_* メソッドで余白を切り取るかどうかを制御する。
NUM_OF_PARAMS	プリペアドステートメントのプレースホルダの数。
NULLABLE	どのカラムに NULL を格納できるかを示す。
trace	デバッグのためにトレースを実行する。

MySQL 固有のメソッドおよび属性

メソッド/属性	説明
mysql_insertid	最新の AUTO_INCREMENT 値を返す。
is_blob	どのカラムが BLOB 値かを示す。
is_key	どのカラムがキーかを示す。
is_num	どのカラムが数値かを示す。
is_pri_key	どのカラムが主キーかを示す。
is_not_null	どのカラムに NULL を格納できないかを示す。NULLABLE を参照。
length	拡大可能な最大カラムサイズを示す。
max_length	結果セットに実際に存在する最大のカラムサイズを示す。

NAME	カラム名を示す。
NUM_OF_FIELDS	結果セットのフィールドの数。
table	結果セットのテーブル名を示す。
type	すべてのカラムの型を示す。

Perl のメソッドについては、以下のセクションで詳細に説明します。メソッドの戻り値に使用される変数には以下の意味があります。

- $dbh

  データベースハンドル

- $sth

  ステートメントハンドル

- $rc

  リターンコード (通常はステータス)

- $rv

  戻り値 (通常はレコード数)

移植可能な DBI メソッドおよび属性

- connect($data_source, $username, $password)

  connect メソッドを使用して、データソースへのデータベース接続を確立する。$data_source 値は DBI:driver_name: で始まる必要がある。DBD::mysql ドライバを使用する connect の使用例を以下に示す。

```
$dbh = DBI->connect("DBI:mysql:$database", $user, $password);
$dbh = DBI->connect("DBI:mysql:$database:$hostname",
                    $user, $password);
$dbh = DBI->connect("DBI:mysql:$database:$hostname:$port",
                    $user, $password);
```

  ユーザ名またはパスワード、あるいはその両方が未定義の場合、DBI はそれぞれ DBI_USER および DBI_PASS の各環境変数の値を使用する。ホスト名を指定しない場合、'localhost' が使用される。ポート番号を指定しない場合、デフォルトの MySQL ポート (3306) が使用される。

  Msql-Mysql-modules バージョン 1.2009 の時点では、$data_source 値に特定の修飾子を使用できる。

  - mysql_read_default_file=file_name

    file_name をオプション設定ファイルとして読み込む。オプション設定ファイルについては、項4.1.2.「my.cnf オプション設定ファイル」 を参照すること。

  - mysql_read_default_group=group_name

オプション設定ファイルを読み込むときのデフォルトグループは、通常は [client] グループである。mysql_read_default_group オプションを指定することによって、デフォルトグループは [group_name] グループになる。

- mysql_compression=1

  クライアントとサーバ間で圧縮された通信を使用する（MySQL バージョン 3.22.3 以降）。

- mysql_socket=/path/to/socket

  サーバへの接続に使用する Unix ソケットのパス名を指定する（MySQL バージョン 3.21.15 以降）。

複数の修飾子をセミコロンで区切って指定できる。

たとえば、ユーザ名およびパスワードを DBI スクリプトにハードコードすることを避けるには、以下のように connect の呼び出しを記述して、ユーザの ~/.my.cnf オプション設定ファイルからユーザ名およびパスワードを読み込む。

```
$dbh = DBI->connect("DBI:mysql:$database"
            . ";mysql_read_default_file=$ENV{HOME}/.my.cnf",
            $user, $password);
```

この呼び出しによって、オプション設定ファイルの [client] グループに定義されたオプションが読み込まれる。上記と同じ処理を、[perl] グループに指定したオプションを使用して実行するには、以下のようなコードを記述する。

```
$dbh = DBI->connect("DBI:mysql:$database"
            . ";mysql_read_default_file=$ENV{HOME}/.my.cnf"
            . ";mysql_read_default_group=perl",
            $user, $password);
```

- disconnect

  disconnect メソッドは、データベースからデータベースハンドルを切断する。通常、このメソッドはプログラムを終了する直前に呼び出される。以下に例を示す。

```
$rc = $dbh->disconnect;
```

- prepare($statement)

  SQL ステートメントを、データベースで実行できるようにプリコンパイルし、ステートメントハンドル ($sth) を返す。ステートメントハンドルを使用して execute メソッドを呼び出すことができる。

  通常、prepare および execute を使用して SELECT ステートメント（および SHOW、DESCRIBE、EXPLAIN などの SELECT ライクなステートメント）を処理する。以下に例を示す。

```
$sth = $dbh->prepare($statement)
  or die "Can't prepare $statement: $dbh->errstr\n";
```

サイズの大きい結果セットをクライアントに読み込む場合、Perl に mysql_use_result() を使用するように指示できる。

```
my $sth = $dbh->prepare($statement { "mysql_use_result" => 1});
```

- execute

  execute メソッドは、プリペアドステートメントを実行する。非 SELECT ステートメントの場合、execute は影響を受けたレコードの数を返す。影響を受けたレコードがない場合、execute は "0E0" を返す。Perl はこの戻り値を 0 として処理するが、true とみなす。エラーが発生した場合、execute は undef を返す。SELECT ステートメントの場合、execute はデータベースで SQL クエリを起動するだけであり、データを取得するには、後述する fetch_* メソッドの中の 1 つを使用する必要がある。以下に例を示す。

```
$rv = $sth->execute
       or die "can't execute the query: " . $sth->errstr;
```

- do($statement)

  do メソッドは、SQL ステートメントをプリコンパイルおよび実行し、影響を受けたレコード数を返す。影響を受けたレコードがない場合、do は "0E0" を返す。Perl はこの戻り値を 0 として処理するが、true とみなす。このメソッドは通常、あらかじめプリコンパイルできない（ドライバの制約により）、または複数回実行する必要がない（挿入、削除など）非 SELECT ステートメントで使用する。以下に例を示す。

```
$rv = $dbh->do($statement)
      or die "Can't execute $statement: $dbh- >errstr\n";
```

  一般に、'do' ステートメントは、パラメータを使用しないステートメントをプリコンパイルして実行するよりも、非常に高速である（使用することが推奨される）。

- quote($string)

  quote メソッドは、文字列に含まれる特殊文字を "エスケープ" して、引用符で囲むために使用する。以下に例を示す。

```
$sql = $dbh->quote($string)
```

- fetchrow_array

  このメソッドは、次のレコードのデータを取得し、フィールド値の配列として返す。以下に例を示す。

```
while(@row = $sth->fetchrow_array) {
      print qw($row[0]\t$row[1]\t$row[2]\n);
}
```

- fetchrow_arrayref

  このメソッドは、次のレコードのデータを取得し、フィールド値の配列として返す。以下に例を示す。

```
while($row_ref = $sth->fetchrow_arrayref) {
```

```
        print qw($row_ref->[0]\t$row_ref->[1]\t$row_ref->[2]\n);
}
```

- fetchrow_hashref

  このメソッドは、レコードデータを取得し、フィールド名と値の組み合わせを含むハッシュテーブルへの参照を返す。これは前述の配列参照を使用する場合ほど効率的ではない。以下に例を示す。

```
while($hash_ref = $sth->fetchrow_hashref) {
        print qw($hash_ref->{firstname}\t$hash_ref->{lastname}\t\
                $hash_ref->{title}\n);
}
```

- fetchall_arrayref

  このメソッドは、SQL ステートメントが返すすべてのデータ（レコード）を取得するために使用する。このメソッドは、"各レコードに対応する配列への参照" の配列への参照を返す。データにアクセスしたり、出力するには、ネストされたループを使用する。以下に例を示す。

```
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

  このステートメントハンドルを使用してこれ以上データを取得しないことを示す。このメソッドを呼び出すことによって、ステートメントハンドルおよびそれに関連付けられたシステムリソースがすべて解放される。以下に例を示す。

```
$rc = $sth->finish;
```

- rows

  最後に実行されたコマンドによって影響を受けた（更新された、削除された、など）レコードの数を返す。通常、非 SELECT execute ステートメントの後に使用される。以下に例を示す。

```
$rv = $sth->rows;
```

- NULLABLE

  カラムに NULL 値を格納できるかどうかを示す値の配列への参照を返す。各配列要素に格納できる値は、カラムに NULL 値を格納できない場合は 0 または空文字列、格納できる場合は 1、カラムの NULL ステータスが不明な場合は 2 である。以下に例を示す。

```
$null_possible = $sth->{NULLABLE};
```

- NUM_OF_FIELDS

  この属性は、SELECT ステートメントまたは SHOW FIELDS ステートメントが返したフィールドの数を示す。これを使用して、結果セットを返すステートメントが実行されたかどうかを調べることができる。値が 0 の場合は、INSERT、DELETE、または UPDATE などの非 SELECT ステートメントが実行されたことを示す。以下に例を示す。

```
$nr_of_fields = $sth->{NUM_OF_FIELDS};
```

- data_sources($driver_name)

  このメソッドは、ホスト 'localhost' 上で動作する MySQL サーバに対して使用できるデータベース名を含む配列を返す。以下に例を示す。

```
@dbs = DBI->data_sources("mysql");
```

- ChopBlanks

  この属性は、fetchrow_* メソッドが、返された値の先頭および末尾の空白を切り取るかどうかを決定する。以下に例を示す。

```
$sth->{'ChopBlanks'} =1;
```

- trace($trace_level) , trace($trace_level, $trace_filename)

  trace メソッドはトレースを有効または無効にする。DBI クラスメソッドとして呼び出された場合は、すべてのハンドルのトレースを有効または無効にする。データベースまたはステートメントハンドルのメソッドとして呼び出された場合は、指定されたハンドル (およびそのハンドルから今後派生する子) のトレースを有効または無効にする。$trace_level を 2 に設定すると、詳細なトレース情報が出力される。$trace_level を 0 に設定すると、トレースが無効になる。トレース情報は、デフォルトでは標準エラー出力に出力される。$trace_filename が指定された場合、そのファイルが追加モードで開かれ、トレースが有効なすべてのハンドルのトレース情報がそのファイルに出力される。以下に例を示す。

```
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
```

  また、DBI_TRACE 環境変数を設定することによって DBI トレースを有効にできる。この変数に数値を設定することは、DBI->(value) を呼び出すことと同等である。この変数にパス名を設定することは、DBI->(2,value) を呼び出すことと同等である。

MySQL 固有のメソッドおよび属性

ここに示すメソッドは MySQL 固有であり、DBI の標準には含まれません。is_blob、is_key、is_num、is_pri_key、is_not_null、length、max_length、table など、その多くは廃止されています。DBI 標準に代替できるメソッドや属性が存在する場合は、ここに明記します。

- mysql_insertid

  MySQL の AUTO_INCREMENT 機能を使用する場合、最新の値がここに格納される。以下に例を示す。

  ```
  $new_id = $sth->{mysql_insertid};
  ```

  古いバージョンの DBI インタフェースでは、$sth->{'insertid'} を使用できる。

- is_blob

  ブール値の配列への参照を返す。配列の要素が TRUE の場合、対応するカラムが BLOB 型であることを示す。以下に例を示す。

  ```
  $keys = $sth->{is_blob};
  ```

- is_key

  ブール値の配列への参照を返す。配列の要素が TRUE の場合、対応するカラムがキーであることを示す。以下に例を示す。

  ```
  $keys = $sth->{is_key};
  ```

- is_num

  ブール値の配列への参照を返す。配列の要素が TRUE の場合、対応するカラムに数値が格納されることを示す。以下に例を示す。

  ```
  $nums = $sth->{is_num};
  ```

- is_pri_key

  ブール値の配列への参照を返す。配列の要素が TRUE の場合、対応するカラムが主キーであることを示す。以下に例を示す。

  ```
  $pri_keys = $sth->{is_pri_key};
  ```

- is_not_null

  ブール値の配列への参照を返す。配列の要素が FALSE の場合、対応するカラムに NULL 値を格納できることを示す。以下に例を示す。

  ```
  $not_nulls = $sth->{is_not_null};
  ```

is_not_null は廃止されている。DBI 標準の NULLABLE 属性（前述）を使用することが推奨される。

- length , max_length

  各メソッドは、カラムサイズの配列への参照を返す。length の配列は、各カラムの（テーブル定義での宣言に従った）拡大可能な最大サイズを示す。max_length の配列は、結果テーブルに実際に存在するデータの最大サイズを示す。以下に例を示す。

```
$lengths = $sth->{length};
$max_lengths = $sth->{max_length};
```

- NAME

  カラム名の配列への参照を返す。以下に例を示す。

```
$names = $sth->{NAME};
```

- table

  テーブル名の配列への参照を返す。以下に例を示す。

```
$tables = $sth->{table};
```

- type

  カラムのデータ型の配列への参照を返す。以下に例を示す。

```
$types = $sth->{type};
```

11.5.3. DBI/DBD に関するその他の情報

perldoc コマンドを使用すると DBI の詳細情報を取得できます。

```
perldoc DBI
perldoc DBI::FAQ
perldoc DBD::mysql
```

また、pod2man、pod2html などのツールを使用して、他のフォーマットのデータに変換できます。

DBI に関する最新情報については、DBI の Web ページ http://dbi.perl.org/ を参照してください。.

11.6. MySQL C++ API

MySQL Connector/C++（または MySQL++）は、C++ のオフィシャル MySQL API です。詳細については、http://www.mysql.com/products/mysql++/ を参照してください。

11.6.1. Borland C++

Borland C++ 5.02 を使用して MySQL Windows ソースをコンパイルできます (Windows ソースには、Microsoft VC++ および Borland C++ 用のプロジェクトファイルしか用意されていないので、プロジェクトファイルを自作する必要があります)。

Borland C++ を使用する場合の唯一の既知の問題は、VC++ とは異なる構造体のアラインメントを使用していることです。これは、デフォルトの libmysql.dll ライブラリ (VC++ でコンパイルされている) を Borland C++ で使用すると、問題が発生することを意味します。以下のいずれかを実行すると、この問題を回避できます。

- http://www.mysql.com/downloads/os-win32.html からダウンロードできる Borland C++ のスタティック MySQL ライブラリを使用する。
- 事前に割り当てられた MYSQL 構造体ではなく、NULL を引数として mysql_init() を呼び出す。

11.7. MySQL Python API

MySQLdb は、Python 向けの MySQL サポートを提供しており、Python DB API バージョン 2.0 に準拠しています。詳細については、http://sourceforge.net/projects/mysql-python/ を参照してください。

11.8. MySQL Tcl API

MySQLtcl は、Tcl プログラム言語から MySQL データベースサーバにアクセスするための簡単な API です。詳細については、http://www.xdobry.de/mysqltcl/ を参照してください。

11.9. MySQL Eiffel Wrapper

Eiffel MysQL は、Mychael Ravits が作成した Eiffel プログラム言語を使用した MySQL データベースサーバへのインタフェースです。詳細については、http://efsa.sourceforge.net/archive/ravits/mysql.htm を参照してください。

第12章 MySQL のエラー処理

この章では、MySQL のエラー処理について説明します。

12.1. 返されるエラー

任意のホスト言語で MySQL を呼び出したときに発生する可能性があるエラーコードのリストを以下に示します。

名前およびエラーコードの列は、MySQL ソースコードファイル include/mysqld_error.h の定義に対応します。

SQLSTATE の列は、MySQL ソースコードファイル include/sql_state.h の定義に対応します。

SQLSTATE エラーコードは、MySQL バージョン 4.1 を使用する場合にのみ表示されます。SQLSTATE は、X/Open/ANSI/ODBC の動作と互換性を持たせるために追加されました。

各エラーコードに対する説明テキストは、エラーメッセージファイル share/english/errmsg.sys に記述されています。

更新は頻繁に行われるので、上記のソースにエラーコードが追加される可能性があります。

- Error: 1000 SQLSTATE: HY000 (ER_HASHCHK)

  Message: hashchk
- Error: 1001 SQLSTATE: HY000 (ER_NISAMCHK)

  Message: isamchk
- Error: 1002 SQLSTATE: HY000 (ER_NO)

  Message: NO
- Error: 1003 SQLSTATE: HY000 (ER_YES)

  Message: YES
- Error: 1004 SQLSTATE: HY000 (ER_CANT_CREATE_FILE)

  Message: '%s' ファイルが作れません (errno: %d)
- Error: 1005 SQLSTATE: HY000 (ER_CANT_CREATE_TABLE)

  Message: '%s' テーブルが作れません.(errno: %d)
- Error: 1006 SQLSTATE: HY000 (ER_CANT_CREATE_DB)

  Message: '%s' データベースが作れません (errno: %d)
- Error: 1007 SQLSTATE: HY000 (ER_DB_CREATE_EXISTS)

  Message: '%s' データベースが作れません.既にそのデータベースが存在します
- Error: 1008 SQLSTATE: HY000 (ER_DB_DROP_EXISTS)

Message: '%s' データベースを破棄できません. そのデータベースがないのです.

- Error: 1009 SQLSTATE: HY000 (ER_DB_DROP_DELETE)

  Message: データベース破棄エラー ('%s' を削除できません, errno: %d)

- Error: 1010 SQLSTATE: HY000 (ER_DB_DROP_RMDIR)

  Message: データベース破棄エラー ('%s' を rmdir できません, errno: %d)

- Error: 1011 SQLSTATE: HY000 (ER_CANT_DELETE_FILE)

  Message: '%s' の削除がエラー (errno: %d)

- Error: 1012 SQLSTATE: HY000 (ER_CANT_FIND_SYSTEM_REC)

  Message: system table のレコードを読む事ができませんでした

- Error: 1013 SQLSTATE: HY000 (ER_CANT_GET_STAT)

  Message: '%s' のステイタスが得られません. (errno: %d)

- Error: 1014 SQLSTATE: HY000 (ER_CANT_GET_WD)

  Message: working directory を得る事ができませんでした (errno: %d)

- Error: 1015 SQLSTATE: HY000 (ER_CANT_LOCK)

  Message: ファイルをロックできません (errno: %d)

- Error: 1016 SQLSTATE: HY000 (ER_CANT_OPEN_FILE)

  Message: '%s' ファイルを開く事ができません (errno: %d)

- Error: 1017 SQLSTATE: HY000 (ER_FILE_NOT_FOUND)

  Message: '%s' ファイルを見付ける事ができません.(errno: %d)

- Error: 1018 SQLSTATE: HY000 (ER_CANT_READ_DIR)

  Message: '%s' ディレクトリが読めません.(errno: %d)

- Error: 1019 SQLSTATE: HY000 (ER_CANT_SET_WD)

  Message: '%s' ディレクトリに chdir できません.(errno: %d)

- Error: 1020 SQLSTATE: HY000 (ER_CHECKREAD)

  Message: Record has changed since last read in table '%s'

- Error: 1021 SQLSTATE: HY000 (ER_DISK_FULL)

  Message: Disk full (%s). 誰かが何かを減らすまでまってください...

- Error: 1022 SQLSTATE: 23000 (ER_DUP_KEY)

Message: table '%s' に key が重複していて書きこめません

- Error: 1023 SQLSTATE: HY000 (ER_ERROR_ON_CLOSE)

  Message: Error on close of '%s' (errno: %d)

- Error: 1024 SQLSTATE: HY000 (ER_ERROR_ON_READ)

  Message: '%s' ファイルの読み込みエラー (errno: %d)

- Error: 1025 SQLSTATE: HY000 (ER_ERROR_ON_RENAME)

  Message: '%s' を '%s' に rename できません (errno: %d)

- Error: 1026 SQLSTATE: HY000 (ER_ERROR_ON_WRITE)

  Message: '%s' ファイルを書く事ができません (errno: %d)

- Error: 1027 SQLSTATE: HY000 (ER_FILE_USED)

  Message: '%s' はロックされています

- Error: 1028 SQLSTATE: HY000 (ER_FILSORT_ABORT)

  Message: Sort 中断

- Error: 1029 SQLSTATE: HY000 (ER_FORM_NOT_FOUND)

  Message: View '%s' が '%s' に定義されていません

- Error: 1030 SQLSTATE: HY000 (ER_GET_ERRNO)

  Message: Got error %d from table handler

- Error: 1031 SQLSTATE: HY000 (ER_ILLEGAL_HA)

  Message: Table handler for '%s' doesn't have this option

- Error: 1032 SQLSTATE: HY000 (ER_KEY_NOT_FOUND)

  Message: '%s'のなかにレコードが見付かりません

- Error: 1033 SQLSTATE: HY000 (ER_NOT_FORM_FILE)

  Message: ファイル '%s' の info が間違っているようです

- Error: 1034 SQLSTATE: HY000 (ER_NOT_KEYFILE)

  Message: '%s' テーブルの key file が間違っているようです. 修復をしてください

- Error: 1035 SQLSTATE: HY000 (ER_OLD_KEYFILE)

  Message: '%s' テーブルは古い形式の key file のようです; 修復をしてください

- Error: 1036 SQLSTATE: HY000 (ER_OPEN_AS_READONLY)

Message: '%s' は読み込み専用です

- Error: 1037 SQLSTATE: HY001 (ER_OUTOFMEMORY)

  Message: Out of memory. デーモンをリスタートしてみてください (%d bytes 必要)

- Error: 1038 SQLSTATE: HY001 (ER_OUT_OF_SORTMEMORY)

  Message: Out of sort memory. sort buffer size が足りないようです.

- Error: 1039 SQLSTATE: HY000 (ER_UNEXPECTED_EOF)

  Message: '%s' ファイルを読み込み中に EOF が予期せぬ所で現れました. (errno: %d)

- Error: 1040 SQLSTATE: 08004 (ER_CON_COUNT_ERROR)

  Message: 接続が多すぎます

- Error: 1041 SQLSTATE: HY000 (ER_OUT_OF_RESOURCES)

  Message: Out of memory; mysqld かその他のプロセスがメモリーを全て使っているか確認してください. メモリーを使い切っていない場合、'ulimit' を設定して mysqld のメモリー使用限界量を多くするか、swap space を増やしてみてください

- Error: 1042 SQLSTATE: 08S01 (ER_BAD_HOST_ERROR)

  Message: その address の hostname が引けません.

- Error: 1043 SQLSTATE: 08S01 (ER_HANDSHAKE_ERROR)

  Message: Bad handshake

- Error: 1044 SQLSTATE: 42000 (ER_DBACCESS_DENIED_ERROR)

  Message: ユーザー '%s'@'%s' の '%s' データベースへのアクセスを拒否します

- Error: 1045 SQLSTATE: 28000 (ER_ACCESS_DENIED_ERROR)

  Message: ユーザー '%s'@'%s' を拒否します.uUsing password: %s)

- Error: 1046 SQLSTATE: 3D000 (ER_NO_DB_ERROR)

  Message: データベースが選択されていません.

- Error: 1047 SQLSTATE: 08S01 (ER_UNKNOWN_COM_ERROR)

  Message: そのコマンドは何？

- Error: 1048 SQLSTATE: 23000 (ER_BAD_NULL_ERROR)

  Message: Column '%s' は null にはできないのです

- Error: 1049 SQLSTATE: 42000 (ER_BAD_DB_ERROR)

Message: '%s' なんてデータベースは知りません.

- Error: 1050 SQLSTATE: 42S01 (ER_TABLE_EXISTS_ERROR)

  Message: Table '%s' は既にあります

- Error: 1051 SQLSTATE: 42S02 (ER_BAD_TABLE_ERROR)

  Message: table '%s' はありません.

- Error: 1052 SQLSTATE: 23000 (ER_NON_UNIQ_ERROR)

  Message: Column: '%s' in %s is ambiguous

- Error: 1053 SQLSTATE: 08S01 (ER_SERVER_SHUTDOWN)

  Message: Server を shutdown 中...

- Error: 1054 SQLSTATE: 42S22 (ER_BAD_FIELD_ERROR)

  Message: '%s' column は '%s' にはありません.

- Error: 1055 SQLSTATE: 42000 (ER_WRONG_FIELD_WITH_GROUP)

  Message: '%s' isn't in GROUP BY

- Error: 1056 SQLSTATE: 42000 (ER_WRONG_GROUP_FIELD)

  Message: Can't group on '%s'

- Error: 1057 SQLSTATE: 42000 (ER_WRONG_SUM_SELECT)

  Message: Statement has sum functions and columns in same statement

- Error: 1058 SQLSTATE: 21S01 (ER_WRONG_VALUE_COUNT)

  Message: Column count doesn't match value count

- Error: 1059 SQLSTATE: 42000 (ER_TOO_LONG_IDENT)

  Message: Identifier name '%s' は長すぎます

- Error: 1060 SQLSTATE: 42S21 (ER_DUP_FIELDNAME)

  Message: '%s' という column 名は重複してます

- Error: 1061 SQLSTATE: 42000 (ER_DUP_KEYNAME)

  Message: '%s' という key の名前は重複しています

- Error: 1062 SQLSTATE: 23000 (ER_DUP_ENTRY)

  Message: '%s' は key %d において重複しています

- Error: 1063 SQLSTATE: 42000 (ER_WRONG_FIELD_SPEC)

Message: Incorrect column specifier for column '%s'

- Error: 1064 SQLSTATE: 42000 (ER_PARSE_ERROR)

  Message: %s : '%s' 付近 : %d 行目

- Error: 1065 SQLSTATE: HY000 (ER_EMPTY_QUERY)

  Message: Query が空です.

- Error: 1066 SQLSTATE: 42000 (ER_NONUNIQ_TABLE)

  Message: '%s' は一意の table/alias 名ではありません

- Error: 1067 SQLSTATE: 42000 (ER_INVALID_DEFAULT)

  Message: Invalid default value for '%s'

- Error: 1068 SQLSTATE: 42000 (ER_MULTIPLE_PRI_KEY)

  Message: 複数の primary key が定義されました

- Error: 1069 SQLSTATE: 42000 (ER_TOO_MANY_KEYS)

  Message: key の指定が多すぎます. key は最大 %d までです

- Error: 1070 SQLSTATE: 42000 (ER_TOO_MANY_KEY_PARTS)

  Message: Too many key parts specified; max %d parts allowed

- Error: 1071 SQLSTATE: 42000 (ER_TOO_LONG_KEY)

  Message: key が長すぎます. key の長さは最大 %d です

- Error: 1072 SQLSTATE: 42000 (ER_KEY_COLUMN_DOES_NOT_EXITS)

  Message: Key column '%s' がテーブルにありません.

- Error: 1073 SQLSTATE: 42000 (ER_BLOB_USED_AS_KEY)

  Message: BLOB column '%s' can't be used in key specification with the used table type

- Error: 1074 SQLSTATE: 42000 (ER_TOO_BIG_FIELDLENGTH)

  Message: column '%s' は,確保する column の大きさが多すぎます. (最大 %d まで). BLOB をかわりに使用してください.

- Error: 1075 SQLSTATE: 42000 (ER_WRONG_AUTO_KEY)

  Message: テーブルの定義が違います; there can be only one auto column and it must be defined as a key

- Error: 1076 SQLSTATE: HY000 (ER_READY)

  Message: %s: 準備完了

- Error: 1077 SQLSTATE: HY000 (ER_NORMAL_SHUTDOWN)

  Message: %s: Normal shutdown

- Error: 1078 SQLSTATE: HY000 (ER_GOT_SIGNAL)

  Message: %s: Got signal %d. 中断!

- Error: 1079 SQLSTATE: HY000 (ER_SHUTDOWN_COMPLETE)

  Message: %s: Shutdown 完了

- Error: 1080 SQLSTATE: 08S01 (ER_FORCING_CLOSE)

  Message: %s: スレッド %ld 強制終了 user: '%s'

- Error: 1081 SQLSTATE: 08S01 (ER_IPSOCK_ERROR)

  Message: IP socket が作れません

- Error: 1082 SQLSTATE: 42S12 (ER_NO_SUCH_INDEX)

  Message: Table '%s' はそのような index を持っていません(CREATE INDEX 実行時に指定されていません). テーブルを作り直してください

- Error: 1083 SQLSTATE: 42000 (ER_WRONG_FIELD_TERMINATORS)

  Message: Field separator argument is not what is expected; check the manual

- Error: 1084 SQLSTATE: 42000 (ER_BLOBS_AND_NO_TERMINATED)

  Message: You can't use fixed rowlength with BLOBs; please use 'fields terminated by'.

- Error: 1085 SQLSTATE: HY000 (ER_TEXTFILE_NOT_READABLE)

  Message: ファイル '%s' は databse の directory にあるか全てのユーザーが読めるように許可されていなければなりません.

- Error: 1086 SQLSTATE: HY000 (ER_FILE_EXISTS_ERROR)

  Message: File '%s' は既に存在します

- Error: 1087 SQLSTATE: HY000 (ER_LOAD_INFO)

  Message: レコード数: %ld 削除: %ld Skipped: %ld Warnings: %ld

- Error: 1088 SQLSTATE: HY000 (ER_ALTER_INFO)

  Message: レコード数: %ld 重複: %ld

- Error: 1089 SQLSTATE: HY000 (ER_WRONG_SUB_KEY)

  Message: Incorrect sub part key; the used key part isn't a string or the used length is longer than the key part

- Error: 1090 SQLSTATE: 42000 (ER_CANT_REMOVE_ALL_FIELDS)

Message: ALTER TABLE で全ての column は削除できません. DROP TABLE を使用してください

- Error: 1091 SQLSTATE: 42000 (ER_CANT_DROP_FIELD_OR_KEY)

  Message: '%s' を破棄できませんでした; check that column/key exists

- Error: 1092 SQLSTATE: HY000 (ER_INSERT_INFO)

  Message: レコード数: %ld 重複数: %ld Warnings: %ld

- Error: 1093 SQLSTATE: HY000 (ER_UPDATE_TABLE_USED)

  Message: You can't specify target table '%s' for update in FROM clause

- Error: 1094 SQLSTATE: HY000 (ER_NO_SUCH_THREAD)

  Message: thread id: %lu はありません

- Error: 1095 SQLSTATE: HY000 (ER_KILL_DENIED_ERROR)

  Message: thread %lu のオーナーではありません

- Error: 1096 SQLSTATE: HY000 (ER_NO_TABLES_USED)

  Message: No tables used

- Error: 1097 SQLSTATE: HY000 (ER_TOO_BIG_SET)

  Message: Too many strings for column %s and SET

- Error: 1098 SQLSTATE: HY000 (ER_NO_UNIQUE_LOGFILE)

  Message: Can't generate a unique log-filename %s.(1-999)

- Error: 1099 SQLSTATE: HY000 (ER_TABLE_NOT_LOCKED_FOR_WRITE)

  Message: Table '%s' は READ lock になっていて、更新はできません

- Error: 1100 SQLSTATE: HY000 (ER_TABLE_NOT_LOCKED)

  Message: Table '%s' は LOCK TABLES によって口ックされていません

- Error: 1101 SQLSTATE: 42000 (ER_BLOB_CANT_HAVE_DEFAULT)

  Message: BLOB column '%s' can't have a default value

- Error: 1102 SQLSTATE: 42000 (ER_WRONG_DB_NAME)

  Message: 指定した database 名 '%s' が間違っています

- Error: 1103 SQLSTATE: 42000 (ER_WRONG_TABLE_NAME)

  Message: 指定した table 名 '%s' はまちがっています

- Error: 1104 SQLSTATE: 42000 (ER_TOO_BIG_SELECT)

Message: The SELECT would examine more than MAX_JOIN_SIZE rows; check your WHERE and use SET SQL_BIG_SELECTS=1 or SET SQL_MAX_JOIN_SIZE=# if the SELECT is okay

- Error: 1105 SQLSTATE: HY000 (ER_UNKNOWN_ERROR)

  Message: Unknown error

- Error: 1106 SQLSTATE: 42000 (ER_UNKNOWN_PROCEDURE)

  Message: Unknown procedure '%s'

- Error: 1107 SQLSTATE: 42000 (ER_WRONG_PARAMCOUNT_TO_PROCEDURE)

  Message: Incorrect parameter count to procedure '%s'

- Error: 1108 SQLSTATE: HY000 (ER_WRONG_PARAMETERS_TO_PROCEDURE)

  Message: Incorrect parameters to procedure '%s'

- Error: 1109 SQLSTATE: 42S02 (ER_UNKNOWN_TABLE)

  Message: Unknown table '%s' in %s

- Error: 1110 SQLSTATE: 42000 (ER_FIELD_SPECIFIED_TWICE)

  Message: Column '%s' specified twice

- Error: 1111 SQLSTATE: HY000 (ER_INVALID_GROUP_FUNC_USE)

  Message: Invalid use of group function

- Error: 1112 SQLSTATE: 42000 (ER_UNSUPPORTED_EXTENSION)

  Message: Table '%s' uses an extension that doesn't exist in this MySQL version

- Error: 1113 SQLSTATE: 42000 (ER_TABLE_MUST_HAVE_COLUMNS)

  Message: テーブルは最低 1 個の column が必要です

- Error: 1114 SQLSTATE: HY000 (ER_RECORD_FILE_FULL)

  Message: table '%s' はいっぱいです

- Error: 1115 SQLSTATE: 42000 (ER_UNKNOWN_CHARACTER_SET)

  Message: character set '%s' はサポートしていません

- Error: 1116 SQLSTATE: HY000 (ER_TOO_MANY_TABLES)

  Message: テーブルが多すぎます; MySQL can only use %d tables in a join

- Error: 1117 SQLSTATE: HY000 (ER_TOO_MANY_FIELDS)

  Message: column が多すぎます

- Error: 1118 SQLSTATE: 42000 (ER_TOO_BIG_ROWSIZE)

  Message: row size が大きすぎます. BLOB を含まない場合の row size の最大は %d です. いくつかの field を BLOB に変えてください.

- Error: 1119 SQLSTATE: HY000 (ER_STACK_OVERRUN)

  Message: Thread stack overrun: Used: %ld of a %ld stack. スタック領域を多くとりたい場合、'mysqld -O thread_stack=#' と指定してください

- Error: 1120 SQLSTATE: 42000 (ER_WRONG_OUTER_JOIN)

  Message: Cross dependency found in OUTER JOIN; examine your ON conditions

- Error: 1121 SQLSTATE: 42000 (ER_NULL_COLUMN_IN_INDEX)

  Message: Column '%s' が UNIQUE か INDEX で使用されました. このカラムは NOT NULL と定義されていません.

- Error: 1122 SQLSTATE: HY000 (ER_CANT_FIND_UDF)

  Message: function '%s' を ロードできません

- Error: 1123 SQLSTATE: HY000 (ER_CANT_INITIALIZE_UDF)

  Message: function '%s' を初期化できません; %s

- Error: 1124 SQLSTATE: HY000 (ER_UDF_NO_PATHS)

  Message: shared library へのパスが通っていません

- Error: 1125 SQLSTATE: HY000 (ER_UDF_EXISTS)

  Message: Function '%s' は既に定義されています

- Error: 1126 SQLSTATE: HY000 (ER_CANT_OPEN_LIBRARY)

  Message: shared library '%s' を開く事ができません (errno: %d %s)

- Error: 1127 SQLSTATE: HY000 (ER_CANT_FIND_DL_ENTRY)

  Message: function '%s' をライブラリー中に見付ける事ができません

- Error: 1128 SQLSTATE: HY000 (ER_FUNCTION_NOT_DEFINED)

  Message: Function '%s' は定義されていません

- Error: 1129 SQLSTATE: HY000 (ER_HOST_IS_BLOCKED)

  Message: Host '%s' は many connection error のため、拒否されました. 'mysqladmin flush-hosts' で解除してください

- Error: 1130 SQLSTATE: HY000 (ER_HOST_NOT_PRIVILEGED)

  Message: Host '%s' は MySQL server に接続を許可されていません

- Error: 1131 SQLSTATE: 42000 (ER_PASSWORD_ANONYMOUS_USER)

Message: MySQL を anonymous users で使用している状態では、パスワードの変更はできません

- Error: 1132 SQLSTATE: 42000 (ER_PASSWORD_NOT_ALLOWED)

  Message: 他のユーザーのパスワードを変更するためには, mysql データベースに対して update の許可がなければなりません.

- Error: 1133 SQLSTATE: 42000 (ER_PASSWORD_NO_MATCH)

  Message: Can't find any matching row in the user table

- Error: 1134 SQLSTATE: HY000 (ER_UPDATE_INFO)

  Message: 一致数(Rows matched): %ld 変更: %ld Warnings: %ld

- Error: 1135 SQLSTATE: HY000 (ER_CANT_CREATE_THREAD)

  Message: 新規にスレッドが作れませんでした (errno %d). もし最大使用許可メモリー数を越えていないのにエラーが発生しているなら, マニュアルの中から 'possible OS-dependent bug' という文字を探してくみてください.

- Error: 1136 SQLSTATE: 21S01 (ER_WRONG_VALUE_COUNT_ON_ROW)

  Message: Column count doesn't match value count at row %ld

- Error: 1137 SQLSTATE: HY000 (ER_CANT_REOPEN_TABLE)

  Message: Can't reopen table: '%s'

- Error: 1138 SQLSTATE: 42000 (ER_INVALID_USE_OF_NULL)

  Message: NULL 値の使用方法が不適切です

- Error: 1139 SQLSTATE: 42000 (ER_REGEXP_ERROR)

  Message: Got error '%s' from regexp

- Error: 1140 SQLSTATE: 42000 (ER_MIX_OF_GROUP_FUNC_AND_FIELDS)

  Message: Mixing of GROUP columns (MIN(),MAX(),COUNT(),...) with no GROUP columns is illegal if there is no GROUP BY clause

- Error: 1141 SQLSTATE: 42000 (ER_NONEXISTING_GRANT)

  Message: ユーザー '%s' (ホスト '%s' のユーザー) は許可されていません

- Error: 1142 SQLSTATE: 42000 (ER_TABLEACCESS_DENIED_ERROR)

  Message: コマンド %s は ユーザー '%s'@'%s' ,テーブル '%s' に対して許可されていません

- Error: 1143 SQLSTATE: 42000 (ER_COLUMNACCESS_DENIED_ERROR)

  Message: コマンド %s は ユーザー '%s'@'%s' カラム '%s' テーブル '%s' に対して許可されていません

- Error: 1144 SQLSTATE: 42000 (ER_ILLEGAL_GRANT_FOR_TABLE)

Message: Illegal GRANT/REVOKE command; please consult the manual to see which privleges can be used.

- Error: 1145 SQLSTATE: 42000 (ER_GRANT_WRONG_HOST_OR_USER)

  Message: The host or user argument to GRANT is too long

- Error: 1146 SQLSTATE: 42S02 (ER_NO_SUCH_TABLE)

  Message: Table '%s.%s' doesn't exist

- Error: 1147 SQLSTATE: 42000 (ER_NONEXISTING_TABLE_GRANT)

  Message: There is no such grant defined for user '%s' on host '%s' on table '%s'

- Error: 1148 SQLSTATE: 42000 (ER_NOT_ALLOWED_COMMAND)

  Message: The used command is not allowed with this MySQL version

- Error: 1149 SQLSTATE: 42000 (ER_SYNTAX_ERROR)

  Message: Something is wrong in your syntax

- Error: 1150 SQLSTATE: HY000 (ER_DELAYED_CANT_CHANGE_LOCK)

  Message: Delayed insert thread couldn't get requested lock for table %s

- Error: 1151 SQLSTATE: HY000 (ER_TOO_MANY_DELAYED_THREADS)

  Message: Too many delayed threads in use

- Error: 1152 SQLSTATE: 08S01 (ER_ABORTING_CONNECTION)

  Message: Aborted connection %ld to db: '%s' user: '%s' (%s)

- Error: 1153 SQLSTATE: 08S01 (ER_NET_PACKET_TOO_LARGE)

  Message: Got a packet bigger than 'max_allowed_packet' bytes

- Error: 1154 SQLSTATE: 08S01 (ER_NET_READ_ERROR_FROM_PIPE)

  Message: Got a read error from the connection pipe

- Error: 1155 SQLSTATE: 08S01 (ER_NET_FCNTL_ERROR)

  Message: Got an error from fcntl()

- Error: 1156 SQLSTATE: 08S01 (ER_NET_PACKETS_OUT_OF_ORDER)

  Message: Got packets out of order

- Error: 1157 SQLSTATE: 08S01 (ER_NET_UNCOMPRESS_ERROR)

  Message: Couldn't uncompress communication packet

- Error: 1158 SQLSTATE: 08S01 (ER_NET_READ_ERROR)

Message: Got an error reading communication packets

- Error: 1159 SQLSTATE: 08S01 (ER_NET_READ_INTERRUPTED)

  Message: Got timeout reading communication packets

- Error: 1160 SQLSTATE: 08S01 (ER_NET_ERROR_ON_WRITE)

  Message: Got an error writing communication packets

- Error: 1161 SQLSTATE: 08S01 (ER_NET_WRITE_INTERRUPTED)

  Message: Got timeout writing communication packets

- Error: 1162 SQLSTATE: 42000 (ER_TOO_LONG_STRING)

  Message: Result string is longer than 'max_allowed_packet' bytes

- Error: 1163 SQLSTATE: 42000 (ER_TABLE_CANT_HANDLE_BLOB)

  Message: The used table type doesn't support BLOB/TEXT columns

- Error: 1164 SQLSTATE: 42000 (ER_TABLE_CANT_HANDLE_AUTO_INCREMENT)

  Message: The used table type doesn't support AUTO_INCREMENT columns

- Error: 1165 SQLSTATE: HY000 (ER_DELAYED_INSERT_TABLE_LOCKED)

  Message: INSERT DELAYED can't be used with table '%s', because it is locked with LOCK TABLES

- Error: 1166 SQLSTATE: 42000 (ER_WRONG_COLUMN_NAME)

  Message: Incorrect column name '%s'

- Error: 1167 SQLSTATE: 42000 (ER_WRONG_KEY_COLUMN)

  Message: The used table handler can't index column '%s'

- Error: 1168 SQLSTATE: HY000 (ER_WRONG_MRG_TABLE)

  Message: All tables in the MERGE table are not defined identically

- Error: 1169 SQLSTATE: 23000 (ER_DUP_UNIQUE)

  Message: Can't write, because of unique constraint, to table '%s'

- Error: 1170 SQLSTATE: 42000 (ER_BLOB_KEY_WITHOUT_LENGTH)

  Message: BLOB column '%s' used in key specification without a key length

- Error: 1171 SQLSTATE: 42000 (ER_PRIMARY_CANT_HAVE_NULL)

  Message: All parts of a PRIMARY KEY must be NOT NULL; if you need NULL in a key, use UNIQUE instead

- Error: 1172 SQLSTATE: 42000 (ER_TOO_MANY_ROWS)

Message: Result consisted of more than one row

- Error: 1173 SQLSTATE: 42000 (ER_REQUIRES_PRIMARY_KEY)

  Message: This table type requires a primary key

- Error: 1174 SQLSTATE: HY000 (ER_NO_RAID_COMPILED)

  Message: This version of MySQL is not compiled with RAID support

- Error: 1175 SQLSTATE: HY000 (ER_UPDATE_WITHOUT_KEY_IN_SAFE_MODE)

  Message: You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column

- Error: 1176 SQLSTATE: HY000 (ER_KEY_DOES_NOT_EXITS)

  Message: Key '%s' doesn't exist in table '%s'

- Error: 1177 SQLSTATE: 42000 (ER_CHECK_NO_SUCH_TABLE)

  Message: Can't open table

- Error: 1178 SQLSTATE: 42000 (ER_CHECK_NOT_IMPLEMENTED)

  Message: The handler for the table doesn't support %s

- Error: 1179 SQLSTATE: 25000 (ER_CANT_DO_THIS_DURING_AN_TRANSACTION)

  Message: You are not allowed to execute this command in a transaction

- Error: 1180 SQLSTATE: HY000 (ER_ERROR_DURING_COMMIT)

  Message: Got error %d during COMMIT

- Error: 1181 SQLSTATE: HY000 (ER_ERROR_DURING_ROLLBACK)

  Message: Got error %d during ROLLBACK

- Error: 1182 SQLSTATE: HY000 (ER_ERROR_DURING_FLUSH_LOGS)

  Message: Got error %d during FLUSH_LOGS

- Error: 1183 SQLSTATE: HY000 (ER_ERROR_DURING_CHECKPOINT)

  Message: Got error %d during CHECKPOINT

- Error: 1184 SQLSTATE: 08S01 (ER_NEW_ABORTING_CONNECTION)

  Message: Aborted connection %ld to db: '%s' user: '%s' host: `%s' (%s)

- Error: 1185 SQLSTATE: HY000 (ER_DUMP_NOT_IMPLEMENTED)

  Message: The handler for the table does not support binary table dump

- Error: 1186 SQLSTATE: HY000 (ER_FLUSH_MASTER_BINLOG_CLOSED)

Message: Binlog closed while trying to FLUSH MASTER

- Error: 1187 SQLSTATE: HY000 (ER_INDEX_REBUILD)

  Message: Failed rebuilding the index of dumped table '%s'

- Error: 1188 SQLSTATE: HY000 (ER_MASTER)

  Message: Error from master: '%s'

- Error: 1189 SQLSTATE: 08S01 (ER_MASTER_NET_READ)

  Message: Net error reading from master

- Error: 1190 SQLSTATE: 08S01 (ER_MASTER_NET_WRITE)

  Message: Net error writing to master

- Error: 1191 SQLSTATE: HY000 (ER_FT_MATCHING_KEY_NOT_FOUND)

  Message: Can't find FULLTEXT index matching the column list

- Error: 1192 SQLSTATE: HY000 (ER_LOCK_OR_ACTIVE_TRANSACTION)

  Message: Can't execute the given command because you have active locked tables or an active transaction

- Error: 1193 SQLSTATE: HY000 (ER_UNKNOWN_SYSTEM_VARIABLE)

  Message: Unknown system variable '%s'

- Error: 1194 SQLSTATE: HY000 (ER_CRASHED_ON_USAGE)

  Message: Table '%s' is marked as crashed and should be repaired

- Error: 1195 SQLSTATE: HY000 (ER_CRASHED_ON_REPAIR)

  Message: Table '%s' is marked as crashed and last (automatic?) repair failed

- Error: 1196 SQLSTATE: HY000 (ER_WARNING_NOT_COMPLETE_ROLLBACK)

  Message: Some non-transactional changed tables couldn't be rolled back

- Error: 1197 SQLSTATE: HY000 (ER_TRANS_CACHE_FULL)

  Message: Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage; increase this mysqld variable and try again

- Error: 1198 SQLSTATE: HY000 (ER_SLAVE_MUST_STOP)

  Message: This operation cannot be performed with a running slave; run STOP SLAVE first

- Error: 1199 SQLSTATE: HY000 (ER_SLAVE_NOT_RUNNING)

  Message: This operation requires a running slave; configure slave and do START SLAVE

- Error: 1200 SQLSTATE: HY000 (ER_BAD_SLAVE)

  Message: The server is not configured as slave; fix in config file or with CHANGE MASTER TO

- Error: 1201 SQLSTATE: HY000 (ER_MASTER_INFO)

  Message: Could not initialize master info structure; more error messages can be found in the MySQL error log

- Error: 1202 SQLSTATE: HY000 (ER_SLAVE_THREAD)

  Message: Could not create slave thread; check system resources

- Error: 1203 SQLSTATE: 42000 (ER_TOO_MANY_USER_CONNECTIONS)

  Message: User %s has already more than 'max_user_connections' active connections

- Error: 1204 SQLSTATE: HY000 (ER_SET_CONSTANTS_ONLY)

  Message: You may only use constant expressions with SET

- Error: 1205 SQLSTATE: HY000 (ER_LOCK_WAIT_TIMEOUT)

  Message: Lock wait timeout exceeded; try restarting transaction

- Error: 1206 SQLSTATE: HY000 (ER_LOCK_TABLE_FULL)

  Message: The total number of locks exceeds the lock table size

- Error: 1207 SQLSTATE: 25000 (ER_READ_ONLY_TRANSACTION)

  Message: Update locks cannot be acquired during a READ UNCOMMITTED transaction

- Error: 1208 SQLSTATE: HY000 (ER_DROP_DB_WITH_READ_LOCK)

  Message: DROP DATABASE not allowed while thread is holding global read lock

- Error: 1209 SQLSTATE: HY000 (ER_CREATE_DB_WITH_READ_LOCK)

  Message: CREATE DATABASE not allowed while thread is holding global read lock

- Error: 1210 SQLSTATE: HY000 (ER_WRONG_ARGUMENTS)

  Message: Incorrect arguments to %s

- Error: 1211 SQLSTATE: 42000 (ER_NO_PERMISSION_TO_CREATE_USER)

  Message: '%s'@'%s' is not allowed to create new users

- Error: 1212 SQLSTATE: HY000 (ER_UNION_TABLES_IN_DIFFERENT_DIR)

  Message: Incorrect table definition; all MERGE tables must be in the same database

- Error: 1213 SQLSTATE: 40001 (ER_LOCK_DEADLOCK)

  Message: Deadlock found when trying to get lock; try restarting transaction

- Error: 1214 SQLSTATE: HY000 (ER_TABLE_CANT_HANDLE_FT)

  Message: The used table type doesn't support FULLTEXT indexes

- Error: 1215 SQLSTATE: HY000 (ER_CANNOT_ADD_FOREIGN)

  Message: Cannot add foreign key constraint

- Error: 1216 SQLSTATE: 23000 (ER_NO_REFERENCED_ROW)

  Message: Cannot add a child row: a foreign key constraint fails

- Error: 1217 SQLSTATE: 23000 (ER_ROW_IS_REFERENCED)

  Message: Cannot delete a parent row: a foreign key constraint fails

- Error: 1218 SQLSTATE: 08S01 (ER_CONNECT_TO_MASTER)

  Message: Error connecting to master: %s

- Error: 1219 SQLSTATE: HY000 (ER_QUERY_ON_MASTER)

  Message: Error running query on master: %s

- Error: 1220 SQLSTATE: HY000 (ER_ERROR_WHEN_EXECUTING_COMMAND)

  Message: Error when executing command %s: %s

- Error: 1221 SQLSTATE: HY000 (ER_WRONG_USAGE)

  Message: Incorrect usage of %s and %s

- Error: 1222 SQLSTATE: 21000 (ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT)

  Message: The used SELECT statements have a different number of columns

- Error: 1223 SQLSTATE: HY000 (ER_CANT_UPDATE_WITH_READLOCK)

  Message: Can't execute the query because you have a conflicting read lock

- Error: 1224 SQLSTATE: HY000 (ER_MIXING_NOT_ALLOWED)

  Message: Mixing of transactional and non-transactional tables is disabled

- Error: 1225 SQLSTATE: HY000 (ER_DUP_ARGUMENT)

  Message: Option '%s' used twice in statement

- Error: 1226 SQLSTATE: 42000 (ER_USER_LIMIT_REACHED)

  Message: User '%s' has exceeded the '%s' resource (current value: %ld)

- Error: 1227 SQLSTATE: HY000 (ER_SPECIFIC_ACCESS_DENIED_ERROR)

  Message: Access denied; you need the %s privilege for this operation

- Error: 1228 SQLSTATE: HY000 (ER_LOCAL_VARIABLE)

  Message: Variable '%s' is a SESSION variable and can't be used with SET GLOBAL

- Error: 1229 SQLSTATE: HY000 (ER_GLOBAL_VARIABLE)

  Message: Variable '%s' is a GLOBAL variable and should be set with SET GLOBAL

- Error: 1230 SQLSTATE: 42000 (ER_NO_DEFAULT)

  Message: Variable '%s' doesn't have a default value

- Error: 1231 SQLSTATE: 42000 (ER_WRONG_VALUE_FOR_VAR)

  Message: Variable '%s' can't be set to the value of '%s'

- Error: 1232 SQLSTATE: 42000 (ER_WRONG_TYPE_FOR_VAR)

  Message: Incorrect argument type to variable '%s'

- Error: 1233 SQLSTATE: HY000 (ER_VAR_CANT_BE_READ)

  Message: Variable '%s' can only be set, not read

- Error: 1234 SQLSTATE: 42000 (ER_CANT_USE_OPTION_HERE)

  Message: Incorrect usage/placement of '%s'

- Error: 1235 SQLSTATE: 42000 (ER_NOT_SUPPORTED_YET)

  Message: This version of MySQL doesn't yet support '%s'

- Error: 1236 SQLSTATE: HY000 (ER_MASTER_FATAL_ERROR_READING_BINLOG)

  Message: Got fatal error %d: '%s' from master when reading data from binary log

- Error: 1237 SQLSTATE: HY000 (ER_SLAVE_IGNORED_TABLE)

  Message: Slave SQL thread ignored the query because of replicate-*-table rules

- Error: 1238 SQLSTATE: HY000 (ER_INCORRECT_GLOBAL_LOCAL_VAR)

  Message: Variable '%s' is a %s variable

- Error: 1239 SQLSTATE: 42000 (ER_WRONG_FK_DEF)

  Message: Incorrect foreign key definition for '%s': %s

- Error: 1240 SQLSTATE: HY000 (ER_KEY_REF_DO_NOT_MATCH_TABLE_REF)

  Message: Key reference and table reference don't match

- Error: 1241 SQLSTATE: 21000 (ER_OPERAND_COLUMNS)

  Message: Operand should contain %d column(s)

- Error: 1242 SQLSTATE: 21000 (ER_SUBQUERY_NO_1_ROW)

  Message: Subquery returns more than 1 row

- Error: 1243 SQLSTATE: HY000 (ER_UNKNOWN_STMT_HANDLER)

  Message: Unknown prepared statement handler (%.*s) given to %s

- Error: 1244 SQLSTATE: HY000 (ER_CORRUPT_HELP_DB)

  Message: Help database is corrupt or does not exist

- Error: 1245 SQLSTATE: HY000 (ER_CYCLIC_REFERENCE)

  Message: Cyclic reference on subqueries

- Error: 1246 SQLSTATE: HY000 (ER_AUTO_CONVERT)

  Message: Converting column '%s' from %s to %s

- Error: 1247 SQLSTATE: 42S22 (ER_ILLEGAL_REFERENCE)

  Message: Reference '%s' not supported (%s)

- Error: 1248 SQLSTATE: 42000 (ER_DERIVED_MUST_HAVE_ALIAS)

  Message: Every derived table must have its own alias

- Error: 1249 SQLSTATE: 01000 (ER_SELECT_REDUCED)

  Message: Select %u was reduced during optimization

- Error: 1250 SQLSTATE: 42000 (ER_TABLENAME_NOT_ALLOWED_HERE)

  Message: Table '%s' from one of the SELECTs cannot be used in %s

- Error: 1251 SQLSTATE: 08004 (ER_NOT_SUPPORTED_AUTH_MODE)

  Message: Client does not support authentication protocol requested by server; consider upgrading MySQL client

- Error: 1252 SQLSTATE: 42000 (ER_SPATIAL_CANT_HAVE_NULL)

  Message: All parts of a SPATIAL index must be NOT NULL

- Error: 1253 SQLSTATE: 42000 (ER_COLLATION_CHARSET_MISMATCH)

  Message: COLLATION '%s' is not valid for CHARACTER SET '%s'

- Error: 1254 SQLSTATE: HY000 (ER_SLAVE_WAS_RUNNING)

  Message: Slave is already running

- Error: 1255 SQLSTATE: HY000 (ER_SLAVE_WAS_NOT_RUNNING)

  Message: Slave has already been stopped

- Error: 1256 SQLSTATE: HY000 (ER_TOO_BIG_FOR_UNCOMPRESS)

  Message: Uncompressed data size too large; the maximum size is %d (probably, length of uncompressed data was corrupted)

- Error: 1257 SQLSTATE: HY000 (ER_ZLIB_Z_MEM_ERROR)

  Message: ZLIB: Not enough memory

- Error: 1258 SQLSTATE: HY000 (ER_ZLIB_Z_BUF_ERROR)

  Message: ZLIB: Not enough room in the output buffer (probably, length of uncompressed data was corrupted)

- Error: 1259 SQLSTATE: HY000 (ER_ZLIB_Z_DATA_ERROR)

  Message: ZLIB: Input data corrupted

- Error: 1260 SQLSTATE: HY000 (ER_CUT_VALUE_GROUP_CONCAT)

  Message: %d line(s) were cut by GROUP_CONCAT()

- Error: 1261 SQLSTATE: 01000 (ER_WARN_TOO_FEW_RECORDS)

  Message: Row %ld doesn't contain data for all columns

- Error: 1262 SQLSTATE: 01000 (ER_WARN_TOO_MANY_RECORDS)

  Message: Row %ld was truncated; it contained more data than there were input columns

- Error: 1263 SQLSTATE: 01000 (ER_WARN_NULL_TO_NOTNULL)

  Message: Data truncated; NULL supplied to NOT NULL column '%s' at row %ld

- Error: 1264 SQLSTATE: 01000 (ER_WARN_DATA_OUT_OF_RANGE)

  Message: Data truncated; out of range for column '%s' at row %ld

- Error: 1265 SQLSTATE: 01000 (ER_WARN_DATA_TRUNCATED)

  Message: Data truncated for column '%s' at row %ld

- Error: 1266 SQLSTATE: HY000 (ER_WARN_USING_OTHER_HANDLER)

  Message: Using storage engine %s for table '%s'

- Error: 1267 SQLSTATE: HY000 (ER_CANT_AGGREGATE_2COLLATIONS)

  Message: Illegal mix of collations (%s,%s) and (%s,%s) for operation '%s'

- Error: 1268 SQLSTATE: HY000 (ER_DROP_USER)

  Message: Can't drop one or more of the requested users

- Error: 1269 SQLSTATE: HY000 (ER_REVOKE_GRANTS)

Message: Can't revoke all privileges, grant for one or more of the requested users

- Error: 1270 SQLSTATE: HY000 (ER_CANT_AGGREGATE_3COLLATIONS)

  Message: Illegal mix of collations (%s,%s), (%s,%s), (%s,%s) for operation '%s'

- Error: 1271 SQLSTATE: HY000 (ER_CANT_AGGREGATE_NCOLLATIONS)

  Message: Illegal mix of collations for operation '%s'

- Error: 1272 SQLSTATE: HY000 (ER_VARIABLE_IS_NOT_STRUCT)

  Message: Variable '%s' is not a variable component (can't be used as XXXX.variable_name)

- Error: 1273 SQLSTATE: HY000 (ER_UNKNOWN_COLLATION)

  Message: Unknown collation: '%s'

- Error: 1274 SQLSTATE: HY000 (ER_SLAVE_IGNORED_SSL_PARAMS)

  Message: SSL parameters in CHANGE MASTER are ignored because this MySQL slave was compiled without SSL support; they can be used later if MySQL slave with SSL is started

- Error: 1275 SQLSTATE: HY000 (ER_SERVER_IS_IN_SECURE_AUTH_MODE)

  Message: Server is running in --secure-auth mode, but '%s'@'%s' has a password in the old format; please change the password to the new format

- Error: 1276 SQLSTATE: HY000 (ER_WARN_FIELD_RESOLVED)

  Message: Field or reference '%s%s%s%s%s' of SELECT #%d was resolved in SELECT #%d

- Error: 1277 SQLSTATE: HY000 (ER_BAD_SLAVE_UNTIL_COND)

  Message: Incorrect parameter or combination of parameters for START SLAVE UNTIL

- Error: 1278 SQLSTATE: HY000 (ER_MISSING_SKIP_SLAVE)

  Message: It is recommended to run with --skip-slave-start when doing step-by-step replication with START SLAVE UNTIL; otherwise, you are not safe in case of unexpected slave's mysqld restart

- Error: 1279 SQLSTATE: HY000 (ER_UNTIL_COND_IGNORED)

  Message: SQL thread is not to be started so UNTIL options are ignored

- Error: 1280 SQLSTATE: 42000 (ER_WRONG_NAME_FOR_INDEX)

  Message: Incorrect index name '%s'

- Error: 1281 SQLSTATE: 42000 (ER_WRONG_NAME_FOR_CATALOG)

  Message: Incorrect catalog name '%s'

- Error: 1282 SQLSTATE: HY000 (ER_WARN_QC_RESIZE)

Message: Query cache failed to set size %lu, new query cache size is %lu

- Error: 1283 SQLSTATE: HY000 (ER_BAD_FT_COLUMN)

  Message: Column '%s' cannot be part of FULLTEXT index

- Error: 1284 SQLSTATE: HY000 (ER_UNKNOWN_KEY_CACHE)

  Message: Unknown key cache '%s'

- Error: 1285 SQLSTATE: HY000 (ER_WARN_HOSTNAME_WONT_WORK)

  Message: MySQL is started in --skip-name-resolve mode. You need to restart it without this switch for this grant to work

- Error: 1286 SQLSTATE: 42000 (ER_UNKNOWN_STORAGE_ENGINE)

  Message: Unknown table engine '%s'

- Error: 1287 SQLSTATE: HY000 (ER_WARN_DEPRECATED_SYNTAX)

  Message: '%s' is deprecated, use '%s' instead

- Error: 1288 SQLSTATE: HY000 (ER_NON_UPDATABLE_TABLE)

  Message: The target table %s of the %s is not updateable

- Error: 1289 SQLSTATE: HY000 (ER_FEATURE_DISABLED)

  Message: The '%s' feature was disabled; you need MySQL built with '%s' to have it working

- Error: 1290 SQLSTATE: HY000 (ER_OPTION_PREVENTS_STATEMENT)

  Message: The MySQL server is running with the %s option so it cannot execute this statement

- Error: 1291 SQLSTATE: HY000 (ER_DUPLICATED_VALUE_IN_TYPE)

  Message: Column '%s' has duplicated value '%s' in %s

- Error: 1292 SQLSTATE: HY000 (ER_TRUNCATED_WRONG_VALUE)

  Message: Truncated wrong %s value: '%s'

- Error: 1293 SQLSTATE: HY000 (ER_TOO_MUCH_AUTO_TIMESTAMP_COLS)

  Message: Incorrect table definition; There can only be one TIMESTAMP column with CURRENT_TIMESTAMP in DEFAULT or ON UPDATE clause

- Error: 1294 SQLSTATE: HY000 (ER_INVALID_ON_UPDATE)

  Message: Invalid ON UPDATE clause for '%s' column

- Error: 1295 SQLSTATE: HY000 (ER_UNSUPPORTED_PS)

  Message: This command is not supported in the prepared statement protocol yet

- Error: 1296 SQLSTATE: HY000 (ER_GET_ERRMSG)

  Message: Got NDB error %d '%s'

- Error: 1297 SQLSTATE: HY000 (ER_GET_TEMPORARY_ERRMSG)

  Message: Got temporary NDB error %d '%s'

- Error: 1298 SQLSTATE: HY000 (ER_UNKNOWN_TIME_ZONE)

  Message: Unknown or incorrect time zone: '%s'

- Error: 1299 SQLSTATE: HY000 (ER_WARN_INVALID_TIMESTAMP)

  Message: Invalid TIMESTAMP value in column '%s' at row %ld

- Error: 1300 SQLSTATE: HY000 (ER_INVALID_CHARACTER_STRING)

  Message: Invalid %s character string: '%s'

- Error: 1301 SQLSTATE: HY000 (ER_WARN_ALLOWED_PACKET_OVERFLOWED)

  Message: Result of %s() was larger than max_allowed_packet (%ld) - truncated

- Error: 1302 SQLSTATE: HY000 (ER_CONFLICTING_DECLARATIONS)

  Message: Conflicting declarations: '%s%s' and '%s%s'

Client error information comes from the following files:

- The Error values and the symbols in parentheses correspond to definitions in the include/errmsg.h MySQL source file.
- The Message values correspond to the error messages that are listed in the libmysql/errmsg.c file. %d or %s represent numbers or strings that are substituted into the messages %when they are displayed.

Because updates are frequent, it is possible that these files contain additional error information not listed here.

- Error: 2000 (CR_UNKNOWN_ERROR)

  Message: Unknown MySQL error

- Error: 2001 (CR_SOCKET_CREATE_ERROR)

  Message: Can't create UNIX socket (%d)

- Error: 2002 (CR_CONNECTION_ERROR)

  Message: Can't connect to local MySQL server through socket '%s' (%d)

- Error: 2003 (CR_CONN_HOST_ERROR)

  Message: Can't connect to MySQL server on '%s' (%d)

- Error: 2004 (CR_IPSOCK_ERROR)

  Message: Can't create TCP/IP socket (%d)

- Error: 2005 (CR_UNKNOWN_HOST)

  Message: Unknown MySQL server host '%s' (%d)

- Error: 2006 (CR_SERVER_GONE_ERROR)

  Message: MySQL server has gone away

- Error: 2007 (CR_VERSION_ERROR)

  Message: Protocol mismatch; server version = %d, client version = %d

- Error: 2008 (CR_OUT_OF_MEMORY)

  Message: MySQL client ran out of memory

- Error: 2009 (CR_WRONG_HOST_INFO)

  Message: Wrong host info

- Error: 2010 (CR_LOCALHOST_CONNECTION)

  Message: Localhost via UNIX socket

- Error: 2011 (CR_TCP_CONNECTION)

  Message: %s via TCP/IP

- Error: 2012 (CR_SERVER_HANDSHAKE_ERR)

  Message: Error in server handshake

- Error: 2013 (CR_SERVER_LOST)

  Message: Lost connection to MySQL server during query

- Error: 2014 (CR_COMMANDS_OUT_OF_SYNC)

  Message: Commands out of sync; you can't run this command now

- Error: 2015 (CR_NAMEDPIPE_CONNECTION)

  Message: Named pipe: %s

- Error: 2016 (CR_NAMEDPIPEWAIT_ERROR)

  Message: Can't wait for named pipe to host: %s pipe: %s (%lu)

- Error: 2017 (CR_NAMEDPIPEOPEN_ERROR)

  Message: Can't open named pipe to host: %s pipe: %s (%lu)

- Error: 2018 (CR_NAMEDPIPESETSTATE_ERROR)

  Message: Can't set state of named pipe to host: %s pipe: %s (%lu)

- Error: 2019 (CR_CANT_READ_CHARSET)

  Message: Can't initialize character set %s (path: %s)

- Error: 2020 (CR_NET_PACKET_TOO_LARGE)

  Message: Got packet bigger than 'max_allowed_packet' bytes

- Error: 2021 (CR_EMBEDDED_CONNECTION)

  Message: Embedded server

- Error: 2022 (CR_PROBE_SLAVE_STATUS)

  Message: Error on SHOW SLAVE STATUS:

- Error: 2023 (CR_PROBE_SLAVE_HOSTS)

  Message: Error on SHOW SLAVE HOSTS:

- Error: 2024 (CR_PROBE_SLAVE_CONNECT)

  Message: Error connecting to slave:

- Error: 2025 (CR_PROBE_MASTER_CONNECT)

  Message: Error connecting to master:

- Error: 2026 (CR_SSL_CONNECTION_ERROR)

  Message: SSL connection error

- Error: 2027 (CR_MALFORMED_PACKET)

  Message: Malformed packet

- Error: 2028 (CR_WRONG_LICENSE)

  Message: This client library is licensed only for use with MySQL servers having '%s' license

- Error: 2029 (CR_NULL_POINTER)

  Message: Invalid use of null pointer

- Error: 2030 (CR_NO_PREPARE_STMT)

  Message: Statement not prepared

- Error: 2031 (CR_PARAMS_NOT_BOUND)

  Message: No data supplied for parameters in prepared statement

- Error: 2032 (CR_DATA_TRUNCATED)

  Message: Data truncated

- Error: 2033 (CR_NO_PARAMETERS_EXISTS)

  Message: No parameters exist in the statement

- Error: 2034 (CR_INVALID_PARAMETER_NO)

  Message: Invalid parameter number

- Error: 2035 (CR_INVALID_BUFFER_USE)

  Message: Can't send long data for non-string/non-binary data types (parameter: %d)

- Error: 2036 (CR_UNSUPPORTED_PARAM_TYPE)

  Message: Using unsupported buffer type: %d (parameter: %d)

- Error: 2037 (CR_SHARED_MEMORY_CONNECTION)

  Message: Shared memory: %s

- Error: 2038 (CR_SHARED_MEMORY_CONNECT_REQUEST_ERROR)

  Message: Can't open shared memory; client could not create request event (%lu)

- Error: 2039 (CR_SHARED_MEMORY_CONNECT_ANSWER_ERROR)

  Message: Can't open shared memory; no answer event received from server (%lu)

- Error: 2040 (CR_SHARED_MEMORY_CONNECT_FILE_MAP_ERROR)

  Message: Can't open shared memory; server could not allocate file mapping (%lu)

- Error: 2041 (CR_SHARED_MEMORY_CONNECT_MAP_ERROR)

  Message: Can't open shared memory; server could not get pointer to file mapping (%lu)

- Error: 2042 (CR_SHARED_MEMORY_FILE_MAP_ERROR)

  Message: Can't open shared memory; client could not allocate file mapping (%lu)

- Error: 2043 (CR_SHARED_MEMORY_MAP_ERROR)

  Message: Can't open shared memory; client could not get pointer to file mapping (%lu)

- Error: 2044 (CR_SHARED_MEMORY_EVENT_ERROR)

  Message: Can't open shared memory; client could not create %s event (%lu)

- Error: 2045 (CR_SHARED_MEMORY_CONNECT_ABANDONED_ERROR)

  Message: Can't open shared memory; no answer from server (%lu)

- Error: 2046 (CR_SHARED_MEMORY_CONNECT_SET_ERROR)

  Message: Can't open shared memory; cannot send request event to server (%lu)

- Error: 2047 (CR_CONN_UNKNOW_PROTOCOL)

  Message: Wrong or unknown protocol

- Error: 2048 (CR_INVALID_CONN_HANDLE)

  Message: Invalid connection handle

- Error: 2049 (CR_SECURE_AUTH)

  Message: Connection using old (pre-4.1.1) authentication protocol refused (client option 'secure_auth' enabled)

- Error: 2050 (CR_FETCH_CANCELED)

  Message: Row retrieval was canceled by mysql_stmt_close() call

- Error: 2051 (CR_NO_DATA)

  Message: Attempt to read column without prior row fetch

- Error: 2052 (CR_NO_STMT_METADATA)

  Message: Prepared statement contains no metadata

第13章 MySQL の拡張

13.1. MySQL の内部情報

この章では、MySQL コードを扱うときに知る必要がある事項について説明します。MySQL の開発に貢献したい、開発中の非公式バージョンのコードにアクセスしたい、あるいは開発過程に興味がある、という場合は、項2.3.4.「開発ソースツリーからのインストール」 の指示を参照してください。MySQL の内部情報に興味がある場合は、当社の internals メーリングリストに加入してください。このメーリングリストはそれほど投稿数は多くありません。加入方法の詳細については、項1.7.1.1.「MySQL メーリングリスト」 を参照してください。MySQL AB の開発者は全員 internals リストに加入しており、MySQL コードに取り組んでる人々をサポートします。コードについて質問したり、MySQL プロジェクトに提供したいパッチを送信する場合には、このリストを遠慮なくご使用ください。

13.1.1. MySQL のスレッド

MySQL サーバが作成するスレッドを以下に示します。

- TCP/IP 接続スレッドは、すべての接続要求を受け取り、新しく専用のスレッドを作成して、接続ごとの認証および SQL クエリの処理を行う。
- Windows NT では、名前付きパイプハンドラスレッドが、名前付きパイプ接続要求に対して TCP/IP 接続スレッドと同じ処理を実行する。
- シグナルスレッドは、すべてのシグナルを処理する。また、通常はアラームを処理して process_alarm() を呼び出し、長時間アイドル状態が続いている接続をタイムアウトさせる。
- mysqld が -DUSE_ALARM_THREAD を指定してコンパイルされている場合、アラーム処理専用スレッドが作成される。このスレッドは、sigwait() で問題が発生しているシステムか、または開発者が専用のシグナル処理スレッドを使用せずに thr_alarm() コードをアプリケーションで使用したい場合にのみ使用する。
- --flush_time=# オプションを使用する場合、専用スレッドが作成され、指定された間隔ですべてのテーブルをフラッシュする。
- すべての接続はそれ自身のスレッドを持つ。
- INSERT DELAYED を使用されるすべてのテーブルはそれ自身のスレッドを持つ。
- --master-host オプションを使用した場合、スレーブレプリケーションスレッドが起動し、マスタの更新を読み込み、適用する。

mysqladmin processlist は、接続スレッド、INSERT DELAYED スレッド、およびレプリケーションスレッドのみを表示します。

13.1.2. MySQL テストスイート

最近までは、全範囲を対象とする当社のテストスイートは独自に開発した顧客データを使用しており、そのために公開されていませんでした。テストプロセスの中で唯一公開されていたのが、sql-bench ディレクトリにある Perl DBI/DBD ベンチマークの crash-me テスト、および tests ディレクトリにあるその他の諸々のテストでした。標準化された公開テストス

イートがなかったため、当社製品のユーザや開発者は、MySQL コードに対して回帰試験を実施することが困難でした。この問題を解決するために、バージョン 3.23.29 から、Unix のソースディストリビューションおよびバイナリディストリビューションに、新しく作成したテストシステムを追加しました。このテストは Unix で実行できます。また、サーバを Cygwin でコンパイルした場合は、Cygwin 環境の Windows 上で実行できます。ネイティブな Windows 環境では現在は動作しません。

現在のテストケースは MySQL 全体をテストするわけではありませんが、SQL を処理するコードにおける明白なバグや OS/ライブラリの問題は検出できるはずです。またレプリケーションのテストは徹底的に行っています。最終的には、コードを 100% カバーするようなテストを作成することを目標としています。そのためにテストスイートを拡充するためのみなさんからのご協力をお待ちしています。特に、お使いのシステムにとってクリティカルな機能を検査するテストをご提供していただければ、今後リリースされるすべての MySQL バージョンでお使いのアプリケーションの動作が保証されることになります。

13.1.2.1. MySQL テストスイートの実行

テストシステムは、テスト言語インタープリタ (mysqltest)、すべてのテストを実行するシェルスクリプト (mysql-test-run)、テスト専用言語で記述された実際のテストケース、および予想されるテスト結果から構成されます。ビルドしてからシステムでテストスイートを実行するには、ソースルートから make test または mysql-test/mysql-test-run と入力します。バイナリディストリビューションをインストールした場合は、cd でインストールルートディレクトリ (/usr/local/mysql など) に移動し、scripts/mysql-test-run を実行します。すべてのテストが正常終了する必要があります。正常終了しないケースがある場合、その原因を調べ、それが MySQL のバグであれば問題点を報告してください。 See 項 13.1.2.3. 「MySQL テストスイートのバグの報告」。

テストスイートを実行しようとするマシン上で mysqld が動作している場合、それが 9306 および 9307 のポートをどちらも使用していないのであれば、それを停止する必要はありません。どちらかのポートが使用されている場合は、mysql-test-run を編集してマスタポートまたはスレーブポート、あるいはその両方を、使用できるポートに変更する必要があります。

mysql-test/mysql-test-run test_name を使用して、個々のテストケースを 1 件ずつ実行できます。

あるテストが失敗した場合、--force オプションを指定して mysql-test-run を実行し、ほかにも失敗するテストがあるかどうかを調べる必要があります。

13.1.2.2. MySQL テストスイートの拡張

mysqltest 言語を使用して独自にテストケースを作成できます。残念ながら、その完全なマニュアルはまだ作成されていません。ただ、現在のテストケースを参考にすることができます。独自にテストケースを作成する際の注意点を以下に示します。

- 作成したテストは mysql-test/t/*.test に置く。
- テストケースは、; で終端されたステートメントで構成され、mysql コマンドラインクライアントへの入力と同様である。ステートメントは、それが内部コマンド (sleep など) として認識される場合を除いて、デフォルトでクエリとして MySQL サーバに送信される。
- SELECT、SHOW、EXPLAIN など、結果セットを生成するすべてのクエリは、先に @/path/to/result/file を指定する必要がある。ファイルには予想される結果を記述する必要がある。結果ファイルを簡単に生成する方法としては、mysql-test ディレクトリから mysqltest -r < t/test-case-name.test を実行し、必要に応じて生成された結果ファイルを編集して予想される出力に変更する。この場合、表示されない文字を追加または削除しないように注意する必要がある。必ずテ

キストの変更または行の削除、あるいはその両方だけを実行するようにする。行を挿入する必要がある場合、必ずフィールドをハードタブで区切り、行の最後にハードタブを置くようにする。od -c を使用すると、編集中にテキストエディタで何も壊していないことを確認できる。mysqltest -r の出力を編集する必要があるのはバグが見つかった場合だけなので、そのような必要性が生じないことを祈っている。

- テストスイートとの整合性を取るには、結果ファイルを mysql-test/r ディレクトリに置いて、名前を test_name.result にする必要がある。テストで複数の結果ファイルを生成する場合は、test_name.a.result、test_name.b.result のように名前を付ける必要がある。
- ステートメントがエラーを返す場合、そのステートメントの前に --error error-number を指定する。ここで、エラー番号には、',' で区切られたエラー番号のリストを指定できる。
- レプリケーションのテストケースを作成している場合、テストファイルの先頭行に source include/master-slave.inc; と記述する必要がある。マスタとスレーブを切り替えるには、connection master; および connection slave; を使用する。もう一方の接続で何らかの処理を実行する必要がある場合、マスタには connection master1; を実行し、スレーブには connection slave1; を実行する。
- ループの中で何らかの処理を実行する必要がある場合、以下のようなコードを作成する。

```
let $1=1000;
while ($1)
{
 # do your queries here
 dec $1;
}
```

- クエリ間でスリープするには、sleep コマンドを使用する。秒数は小数点以下まで指定できるので、たとえば 1.3 秒スリープする場合は sleep 1.3; を実行する。
- テストケースのためにオプションを追加してスレーブを実行するには、mysql-test/t/test_name-slave.opt にコマンドライン形式で記述する。マスタの場合は、mysql-test/t/test_name-master.opt に記述する。
- テストスイートに質問がある場合、またはテストスイートに提供するテストケースがある場合、MySQL 内部情報のメーリングリストにメールを送信する。See 項1.7.1.1.「MySQL メーリングリスト」。 このメーリングリストは添付ファイルを受け付けないので、関連するファイルは ftp://support.mysql.com/pub/mysql/Incoming/ にアップロードする必要がある。

13.1.2.3. MySQL テストスイートのバグの報告

お使いの MySQL バージョンがテストスイートを正常終了できない場合、以下の処理を実行する必要があります。

- どこで異常が発生しているかをできるかぎり調査するまで、バグレポートを送信しないこと。バグレポートを送信する場合は、使用しているシステムおよび MySQL のバージョンに関する情報を収集するための mysqlbug スクリプトを使用する。 See 項1.7.1.3.「バグまたは問題を報告する方法」。
- mysql-test-run の出力および mysql-test/r ディレクトリのすべての .reject ファイルの内容を必ず送信する。
- テストスイートのテストが異常終了する場合、そのテストを単独で実行したときも異常終了するかどうかを調べる。

```
cd mysql-test
```

```
mysql-test-run --local test-name
```

単独で実行しても異常終了する場合、--with-debug を指定して MySQL を実行し、--debug オプション付きで mysql-test-run を実行する。これでも異常終了する場合は、当社でトレースファイル var/tmp/master.trace を調べたいので、ftp://support.mysql.com/pub/mysql/secret にそのファイルをアップロードする。このとき、必ず使用しているシステムの詳細な説明、mysqld バイナリのバージョン、および mysqld バイナリのコンパイル方法を連絡する。

- --force オプションを指定して mysql-test-run を実行し、ほかのテストが異常終了するかどうかも調べる。
- 自分で MySQL をコンパイルした場合、マニュアルを参照して使用しているプラットフォーム上で MySQL をコンパイルする方法を調べる。できれば、当社がコンパイルしたバイナリを http://www.mysql.com/downloads/ からダウンロードして使用する。当社が提供する標準のバイナリはすべてテストスイートを正常終了するはずである。
- Result length mismatch または Result content mismatch などのエラーが発生する場合、これはテストの出力と予想される出力が正確に一致しなかったことを意味する。この場合、MySQL のバグの可能性がある。あるいは、使用している mysqld のバージョンが特定の環境下で多少異なる結果を出力した可能性がある。

  異常終了したテスト結果は、結果ファイルと同じベース名に拡張子 .reject を付けた名前のファイルに出力される。テストケースを実行して異常終了した場合、これらの 2 つのファイルをパラメータとして diff を実行する必要がある。両者の違いがわからない場合、それぞれをパラメータとして od -c を実行し、そのサイズを調べる。
- テストが全体的に異常終了する場合、mysql-test/var/log ディレクトリのログファイルを調べて、異常の原因に関するヒントを探す。
- デバッグ機能を組み込んで MySQL をコンパイルしていた場合、--gdb または --debug、あるいはその両方のオプションを指定して mysql-test-run を実行することによってデバッグできる。 See 項E.1.2.「トレースファイルの作成」。

  デバッグ機能を組み込まずに MySQL をコンパイルしていた場合、デバッグ機能を組み込むようにコンパイルしてから再実行する必要がある。それには、--with-debug オプションを指定して configure を実行する。 See 項2.3.「MySQL ソースディストリビューションのインストール」。

13.2. MySQL への新しい関数の追加

MySQL に新しい関数を追加する方法は 2 つあります。

- ユーザ定義関数 (UDF) インタフェースを使用して関数を追加できる。ユーザ定義関数は、CREATE FUNCTION ステートメントおよび DROP FUNCTION ステートメントを使用して、動的に追加および削除できる。 See 項13.2.1.「CREATE FUNCTION/DROP FUNCTION の構文」。
- ネイティブ (組み込み) MySQL 関数として関数を追加できる。ネイティブ関数は、コンパイルして mysqld サーバに組み込むことによって、永続的に使用できるようになる。

それぞれの方法に長所および短所があります。

- ユーザ定義関数を作成する場合、サーバのほかにオブジェクトファイルをインストールする必要がある。関数をコンパイルしてサーバに組み込んだ場合はその必要はない。

- UDF は、MySQL バイナリディストリビューションに追加できる。ネイティブ関数の場合、ソースディストリビューションを修正する必要がある。
- MySQL ディストリビューションをアップグレードする場合、既存の UDF はそのまま使用できる。ネイティブ関数の場合、アップグレードするたびに修正作業を繰り返す必要がある。

どちらの方法を使用して新しい関数を追加する場合も、ABS() や SOUNDEX() などのネイティブ関数と同じように使用できます。

13.2.1. CREATE FUNCTION/DROP FUNCTION の構文

```
CREATE [AGGREGATE] FUNCTION function_name RETURNS {STRING|REAL|INTEGER}
    SONAME shared_library_name

DROP FUNCTION function_name
```

ユーザ定義関数 (UDF) は、ABS() や CONCAT() などの MySQL ネイティブ (組み込み) 関数と同様に動作する新しい関数を追加して MySQL を拡張するための手段です。

AGGREGATE は、MySQL バージョン 3.23 で新しく追加されました。AGGREGATE 関数は SUM や COUNT() などの MySQL のネイティブグループ関数とまったく同じように動作します。

CREATE FUNCTION は、関数の名前、型、および共有ライブラリ名を mysql.func システムテーブルに保存します。関数を作成および削除するには、mysql データベースに対する INSERT 特権および DELETE 特権を持っている必要があります。

mysqld を起動する際に --skip-grant-tables オプションが指定されていなかった場合、すべてのアクティブな関数はサーバが起動するたびに再ロードされます。オプションが指定されていた場合、UDF の初期化は行われず、それを使用することはできません (アクティブな関数とは CREATE FUNCTION でロードされたが DROP FUNCTION による削除は行われていない関数) 。

ユーザ定義関数の作成手順については、項13.2.「MySQL への新しい関数の追加」 を参照してください。UDF のメカニズムが機能するには、関数を C または C++ で記述する必要があります。また、オペレーティングシステムでダイナミックローディングがサポートされていること、および mysqld を (静的ではなく) 動的にコンパイルしている必要があります。

注意: AGGREGATE を実行するには、type カラムを含む mysql.func テーブルが必要です。このテーブルがない場合、スクリプト mysql_fix_privilege_tables を実行して、このテーブルを作成する必要があります。

13.2.2. 新しいユーザ定義関数の追加

UDF メカニズムが機能するには、関数を C または C++ で記述し、オペレーティングシステムでダイナミックローディングがサポートされている必要があります。MySQL ソースディストリビューションには、5 つの関数を定義する sql/udf_example.cc ファイルが含まれています。UDF の呼び出し例として、このファイルを参考にしてください。mysqld が UDF 関数を使用できるようにするには、--with-mysqld-ldflags=-rdynamic オプションを指定して MySQL を設定する必要があります。その理由は、多くのプラットフォーム (Linux を含む) では、静的リンクされたプログラムから (dlopen() を使用して) ダイナミックライブラリをロードできます。これは --with-mysqld-ldflags=-all-static を使用している場合でも実行できます。しかし、mysqld からシンボルにアクセスする必要がある UDF を使用する場合 (default_charset_info を使用する sql/udf_example.cc の metaphone の例を参照) 、プログラムを -rdynamic を指定してリンクする必要があるからです (

man dlopen 参照）。

プリコンパイルされたバージョンのサーバを使用している場合は、MySQL-Max を使用します。MySQL-Max はダイナミックローディングをサポートしています。

SQL ステートメントで使用する関数には、それぞれ対応する C（または C++）関数を定義する必要があります。以下の説明では、``xxx" をサンプルの関数名として使用します。SQL と C/C++ の区別をするために、XXX()（大文字）は SQL 関数呼び出しを、xxx()（小文字）は C/C++ 関数呼び出しを、それぞれ表すものとします。

XXX() のインタフェースを実装するために作成する C/C++ 関数を以下に示します。

- xxx()（必須）

  関数のメイン。ここで関数の戻り値を計算する。SQL の型と C/C++ 関数の戻り値の型の対応を以下に示す。

SQL の型	C/C++ の型
STRING	char *
INTEGER	long long
REAL	double

- xxx_init()（省略可能）

  xxx() の初期化関数。以下の処理を実行できる。

  - XXX() に渡す引数の数を確認する。
  - 引数が要求している型であることを確認する。または、関数が呼び出されたときに、引数を強制的に要求している型に変換するように MySQL に指示する。
  - 関数で必要とするメモリを割り当てる。
  - 結果の最大長を指定する。
  - 最大の小数点以下桁数（REAL 関数で使用）を指定する。
  - 結果として NULL を返すことができるかどうかを指定する。

- xxx_deinit()（省略可能）

  xxx() の後処理関数。初期化関数が割り当てたメモリを解放する必要がある。

SQL ステートメントが XXX() を呼び出すと、MySQL は初期化関数 xxx_init() を呼び出して、引数の確認やメモリ割り当てなどの必要なセットアップを実行させます。xxx_init() がエラーを返した場合、SQL ステートメントの処理はエラーメッセージを出力して中断され、関数のメインと後処理関数の呼び出しは行われません。正常終了した場合、関数 xxx() がレコードごとに 1 回呼び出されます。すべてのレコードの処理が終了したら、後処理関数 xxx_deinit() が呼び出され、必要なクリーンアップ処理を実行します。

集計関数（SUM() など）の場合は、以下の関数も用意する必要があります。

- xxx_reset() (必須)

  合計値をリセットし、引数を新しいグループの初期値として使用する。

- xxx_add() (必須)

  引数を既存の合計値に加算する。

集計 UDF を使用する場合、MySQL は以下のように動作します。

1. xxx_init() を呼び出して集計関数に結果を保存するためのメモリを割り当てる。
2. GROUP BY 式に従ってテーブルを並べ替える。
3. 新しいグループの先頭のレコードに対して、xxx_reset() 関数を呼び出す。
4. 同じグループに属する新しいレコードごとに、xxx_add() 関数を呼び出す。
5. グループが変わった場合、または最後のレコードの処理が終わった場合、xxx() を呼び出して集計値を取得する。
6. すべてのレコードの処理が終了するまで 3 から 5 の手順を繰り返す。
7. xxx_deinit() を呼び出して、UDF に割り当てられていたメモリを解放する。

上記の関数はすべて (関数だけでなく、初期化関数および後処理関数も) スレッドセーフで作成する必要があります。これは、変化する可能性があるグローバル変数やスタティック変数にメモリを割り当てることはできないことを意味します。メモリが必要な場合は、xxx_init() で割り当て、xxx_deinit() で解放する必要があります。

13.2.2.1. 単純関数の場合の UDF の呼び出し手順

関数は以下に示すように宣言する必要があります。注意: 戻り値の型およびパラメータは、CREATE FUNCTION ステートメントで SQL 関数 XXX() の戻り値を STRING、INTEGER、または REAL のどの型で返すと宣言するかによって異なります。

STRING 関数の場合は以下のように宣言します。

```
char *xxx(UDF_INIT *initid, UDF_ARGS *args,
        char *result, unsigned long *length,
        char *is_null, char *error);
```

INTEGER 関数の場合は以下のように宣言します。

```
long long xxx(UDF_INIT *initid, UDF_ARGS *args,
            char *is_null, char *error);
```

REAL 関数の場合は以下のように宣言します。

```
double xxx(UDF_INIT *initid, UDF_ARGS *args,
            char *is_null, char *error);
```

初期化関数および終了関数は以下のように宣言します。

```
my_bool xxx_init(UDF_INIT *initid, UDF_ARGS *args, char *message);

void xxx_deinit(UDF_INIT *initid);
```

initid パラメータは、上記の 3 つの関数すべてに渡されます。UDF_INIT 構造体を参照しており、関数間のデータの受け渡しに使用されます。UDF_INIT 構造体のメンバを以下に示します。初期化関数では、必要に応じてメンバの値を変更します (メンバのデフォルト値を使用する場合はそのままにしておきます)。

- my_bool maybe_null

  xxx_init() は、xxx() が NULL を返す可能性がある場合は maybe_null を 1 に設定する必要がある。maybe_null と宣言されている引数が存在する場合は、デフォルト値は 1 である。

- unsigned int decimals

  小数点以下桁数。デフォルト値は、関数に渡される引数の最大の小数点以下桁数 (たとえば、1.34、1.345、および 1.3 を渡された場合、1.345 の小数点以下桁数が 3 で一番大きいので、デフォルト値は 3 になる)。

- unsigned int max_length

  戻り値の文字列表現の最大長。関数の戻り値の型によってデフォルト値は異なる。文字列関数の場合、デフォルトは最長の引数の長さ。整数関数の場合、デフォルトは 21 桁。実数関数の場合、デフォルトは 13 に initid->decimals で示される小数点以下桁数を加算した値 (数値関数の場合、この長さには符号や小数点が含まれる)。

  BLOB 型の値を返す場合、このメンバの値を 65K または 16M に設定できる。このメモリは割り当てられないが、一時的にデータを格納する必要が生じた場合、どのカラム型を使用するかを判断するために使用できる。

- char *ptr

  関数が独自の用途に使用できるポインタ。たとえば、関数は initid->ptr を使用して、割り当てられたメモリを関数間で受け渡すことができる。xxx_init() では、以下のようにメモリを割り当て、その値をこのポインタに代入する。

  ```
  initid->ptr = allocated_memory;
  ```

  xxx() および xxx_deinit() は、initid->ptr を参照してメモリを使用したり、解放します。

13.2.2.2. 集計関数の場合の UDF の呼び出し手順

ここでは、集計 UDF 関数を作成する際に定義する必要があるさまざまな関数について説明します。

注意: 以下に示す関数は MySQL 4.1.1 では必要ありません。また使用されることもありません。MySQL 4.0 および MySQL 4.1.1 の両方で動作するようなコードを作成する場合にこれらの関数を定義します。

```
char *xxx_reset(UDF_INIT *initid, UDF_ARGS *args,
                char *is_null, char *error);
```

この関数は、MySQL が新しいグループの最初の行を見つけたときに呼び出されます。この関数では、内部の集計変数をリセットし、渡された引数をグループの最初の引数として設定する必要があります。

多くの場合、この処理はすべての変数をリセットすることで内部的に実装します (たとえば、xxx_clear() を呼び出してから xxx_add() を呼び出します) 。

次の関数は、MySQL 4.1.1 以降でのみ必要です。

```
char *xxx_clear(UDF_INIT *initid, char *is_null, char *error);
```

この関数は、MySQL が集計結果をリセットする必要があるときに呼び出されます。この関数は新しいグループが始まるたびに呼び出されますが、一致するレコードがなかったクエリの値をリセットするために呼び出される場合もあります。is_null は、xxx_clear() を呼び出す前に CHAR(0) を参照するように設定されます。

何らかの異常が発生した場合、error の参照先に 1 バイトを保存できます。

```
char *xxx_add(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

この関数は、同じグループに属する 2 番目以降のすべてのレコードについて呼び出されます。この関数では、UDF_ARGS の値を内部集計変数に加算する必要があります。

xxx() 関数は、単純 UDF 関数を定義するときと同じように宣言する必要があります。 See 項13.2.2.1.「単純関数の場合の UDF の呼び出し手順」。

この関数は、グループのすべてのレコードが処理されたときに呼び出されます。通常はこの関数で args 変数にアクセスする必要はまったくなく、内部集計変数に基づいて値を返します。

xxx_reset() および xxx_add() では、引数の処理はすべて、通常の UDF と同じように実行する必要があります。 See 項13.2.2.3.「引数の処理」。

xxx() では、戻り値の処理を、通常の UDF と同じように実行する必要があります。 See 項13.2.2.4.「戻り値およびエラー処理」。

is_null および error へのポインタ引数は、xxx_reset()、xxx_clear()、xxx_add()、および xxx() への呼び出しですべて同じです。この引数を使用して、エラーが発生したこと、または xxx() 関数が NULL を返す必要があるかどうかを記憶できます。注意: *error には文字列は格納できません。これは 1 バイトフラグです。

is_null は、グループごとにリセットされます (xxx_clear() を呼び出す前でも) 。error はリセットされることはありません。

isnull または error が xxx() の後に設定された場合、MySQL はグループ関数の結果として NULL を返します。

13.2.2.3. 引数の処理

args パラメータは、UDF_ARGS 構造体を参照します。そのメンバを以下に示します。

- unsigned int arg_count

  引数の数。作成した関数を呼び出すときに引数の数が決まっている場合は、初期化関数でこの値をチェックする。たとえば、以下のように指定する。

  ```
  if (args->arg_count != 2)
  {
  ```

```
  strcpy(message,"XXX() requires two arguments");
  return 1;
}
```

- enum Item_result *arg_type

  各引数の型。型を表す値として設定できるのは、STRING_RESULT、INT_RESULT、および REAL_RESULT である。

  引数が指定した型であることを確認し、一致しない場合にエラーを返すには、初期化関数で arg_type 配列をチェックする。たとえば、以下のように指定する。

```
if (args->arg_type[0] != STRING_RESULT ||
  args->arg_type[1] != INT_RESULT)
{
  strcpy(message,"XXX() requires a string and an integer");
  return 1;
}
```

  関数の引数を特定の型で渡すように要求する代わりに、初期化関数を使用して arg_type の各要素に希望する型を設定する方法もある。これにより、MySQL は、xxx()を呼び出すたびに強制的に引数をその型に変換する。たとえば、最初の 2 つの引数を文字列と整数で渡すように指定するには、xxx_init() で以下の処理を行う。

```
args->arg_type[0] = STRING_RESULT;
args->arg_type[1] = INT_RESULT;
```

- char **args

  args->args は、関数が呼び出されたときに渡された引数の一般的な性質について初期化関数に情報を伝達する。i 番目の引数が定数の場合、args->args[i] は引数の値への参照である（値にアクセスする正しい方法については以下の手順を参照）。非定数の場合、args->args[i] は 0 である。定数引数は、3、4*7-2、または SIN(3.14) など、定数だけを使用する式である。非定数引数は、カラム名、または非定数引数を使用して呼び出される関数など、レコードごとに変化する可能性がある値を参照する式である。

  関数を呼び出すたびに、args->args に、現在処理中のレコードに関して渡される実際の引数が設定される。

  関数は i 番目の引数を以下のように参照する。

  - STRING_RESULT 型の引数は、文字列へのポインタとその長さとして渡され、バイナリデータまたは可変長のデータを処理できる。実際の文字列は args->args[i] で参照し、文字列の長さは args->lengths[i] と表す。文字列がヌルで終端されていると仮定することは不適切である。
  - INT_RESULT 型の引数の場合、args->args[i] をキャストして long long 型の値を取得する必要がある。

```
long long int_val;
int_val = *((long long*) args->args[i]);
```

  - REAL_RESULT 型の引数の場合、args->args[i] をキャストして double 型の値を取得する必要がある。

```
double    real_val;
```

```
real_val = *((double*) args->args[i]);
```

- unsigned long *lengths

  初期化関数の呼び出し時は、lengths 配列の要素は対応する引数の最大文字列長を示す。この値は変更せず、そのままにしておく必要がある。関数を呼び出すたびに、lengths に、現在処理中のレコードに関して渡される文字列引数の実際の長さが設定される。（初期化関数の呼び出し時は）引数が INT_RESULT 型または REAL_RESULT 型である場合も、lengths には引数の最大長が格納される。

13.2.2.4. 戻り値およびエラー処理

初期化関数は、エラーがない場合は 0、エラーが発生した場合は 1 を返す必要があります。エラーが発生した場合、xxx_init() は message パラメータにヌルで終端されたエラーメッセージを格納する必要があります。このメッセージがクライアントに返されます。メッセージバッファの長さは MYSQL_ERRMSG_SIZE 文字ですが、標準の端末画面の幅に合わせて、80 文字未満でメッセージを作成するように努力する必要があります。

long long 関数および double 関数の場合、関数 xxx() の戻り値が関数値になります。文字列関数の場合、結果を参照するポインタを返し、その文字列長を length 引数に格納する必要があります。

戻り値の内容と長さは、以下のように設定します。

```
memcpy(result, "result string", 13);
*length = 13;
```

calc 関数には、255 バイトの result バッファが渡されます。返す値がこの大きさに収まる場合は、そのメモリ割り当てについて心配する必要はありません。

文字列関数が 255 バイトより長い文字列を返す必要がある場合、xxx_init() 関数または xxx() 関数で malloc() を使用してそのための領域を割り当て、xxx_deinit() 関数でそれを解放する必要があります。割り当てたメモリを UDF_INIT 構造体の ptr スロットに保存しておけば、将来 xxx() が呼び出されたときにそのメモリを再利用できます。 See 項13.2.2.1.「単純関数の場合の UDF の呼び出し手順」。

関数で戻り値が NULL であることを示すには、is_null に 1 を設定します。

```
*is_null = 1;
```

関数がエラーを返すことを示すには、error に 1 を設定します。

```
*error = 1;
```

xxx() が、いずれかのレコードの処理で *error に 1 を設定した場合、カレントレコードおよび XXX() を呼び出したステートメントが処理するそれ以降のレコードについて、関数値が NULL になります(それ以降のレコードでは xxx() の呼び出しも行われません）。注意:MySQL 3.22.10 より古いバージョンでは、*error および *is_null の両方に値を設定する必要があります。

```
*error = 1;
*is_null = 1;
```

13.2.2.5. ユーザ定義関数のコンパイルおよびインストール

UDF を実装するファイルは、サーバが動作するホスト上でコンパイルし、インストールする必要があります。ここでは、MySQL ソースディストリビューションに含まれるサンプル UDF ファイル udf_example.cc を使用して、その手順について説明します。このファイルには以下の関数が定義されています。

- metaphon() は、文字列引数の metaphon 文字列を返す。これは soudex 文字列と同種の文字列で、英語向けに精度が高められている。
- myfunc_double() は、その引数に含まれる文字の ASCII 値の合計を、その引数の長さの合計で割った値を返す。
- myfunc_int() は、その引数の長さの合計を返す。
- sequence([const int]) は、番号が指定された場合はその番号から始まるシーケンス、指定されなかった場合は 1 から始まるシーケンスを返す。
- lookup() は、ホスト名に対応する IP 番号を返す。
- reverse_lookup() は、IP 番号に対応するホスト名を返す。文字列 "xxx.xxx.xxx.xxx" または 4 つの数値をパラメータとして呼び出すことができる。

ダイナミックにロード可能なファイルは、以下のようなコマンドを使用して、共有オブジェクトファイルとしてコンパイルする必要があります。

```
shell> gcc -shared -o udf_example.so myfunc.cc
```

MySQL ソースツリーの sql ディレクトリで以下のコマンドを実行するだけで、システムに合った適切なコンパイラオプションを知ることができます。

```
shell> make udf_example.o
```

make が表示するコンパイルコマンドをほとんどそのまま使用できます。ただし、行末付近にある -c オプションは指定しないでください。また、行末に -o udf_example.so を追加してください (一部のシステムでは、-c オプションをコマンドに残す必要があります)。

UDF を含む共有オブジェクトをコンパイルしたら、それをインストールして、MySQL に通知する必要があります。udf_example.cc をコンパイルして共有オブジェクトを作成すると、udf_example.so のような名前のファイルが生成されます (正確な名前はプラットフォームによって異なる可能性があります)。このファイルを /usr/lib などのダイナミックリンカ ld の検索対象パスにコピーするか、または共有オブジェクトを置いたディレクトリをリンカ設定ファイル (たとえば /etc/ld.so.conf) に追加します。

多くのシステムでは、UDF 関数ファイルを置いたディレクトリを参照するように環境変数 LD_LIBRARY または LD_LIBRARY_PATH を設定することもできます。システムで使用する必要がある変数については、dlopen のマニュアルページを参照してください。それらの変数を mysql.server または mysqld_safe の起動スクリプトで設定し、mysqld を再起動する必要があります。

ライブラリをインストールしたら、以下のコマンドを使用して、新しく追加した関数について mysqld に通知します。

```
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";
```

関数を削除するには、DROP FUNCTION を使用します。

```
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;
```

CREATE FUNCTION および DROP FUNCTION の各ステートメントは、mysql データベースのシステムテーブル func を更新します。関数の名前、型、および共有ライブラリ名はこのテーブルに保存されます。関数を作成および削除するには、mysql データベースに対する INSERT 特権および DELETE 特権を持っている必要があります。

すでに作成されている関数を CREATE FUNCTION を使用して追加してはいけません。関数を再インストールする必要がある場合は、まず DROP FUNCTION で関数を削除してから CREATE FUNCTION で再インストールします。たとえば新しいバージョンの関数を再コンパイルした場合に、この手順で再インストールして mysqld に新しいバージョンの関数を追加します。再インストールしなければ、サーバは古いバージョンの関数を使い続けます。

mysqld を起動する際に --skip-grant-tables オプションが指定されていなかった場合、アクティブな関数はサーバが起動するたびに再ロードされます。オプションが指定されていた場合、UDF の初期化は行われず、それを使用することはできません（アクティブな関数とは CREATE FUNCTION でロードされたが DROP FUNCTION による削除は行われていない関数）。

13.2.3. 新しいネイティブ関数の追加

ここでは、新しいネイティブ関数を追加する手順について説明します。注意: この手順には MySQL ソースコードの修正が含まれるので、バイナリディストリビューションにネイティブ関数を追加することはできません。MySQL をソースディストリビューションからコンパイルする必要があります。注意:（新しいバージョンがリリースされたときなど）別のバージョンの MySQL に移行する場合、新しいバージョンでも同じ手順を繰り返してネイティブ関数を追加する必要があります。

MySQL の新しいネイティブ関数を追加するには、以下の手順に従います。

1. sql_functions[] 配列に関数名を定義する行を lex.h に追加する。
2. 関数のプロトタイプが単純な場合（引数の数が 0、1、2、または 3 個しかない）、lex.h で sql_functions[] 配列の 2 番目の引数として SYM(FUNC_ARG#)（# は引数の数）を指定し、item_create.cc に関数オブジェクトを作成する関数を追加する必要がある。この例として、"ABS" および create_funcs_abs() を参照すること。

   関数のプロトタイプが複雑な場合（引数の数が可変の場合など）、sql_yacc.yy に 2 行追加する必要がある。まず、yacc が定義する必要があるプリプロセッサシンボルを表す行を追加する（ファイルの先頭部分）。次に、関数のパラメータを定義し、パラメータの指定方法を定義する行を simple_expr 解析規則に追加する。この参考例として、sql_yacc.yy の ATAN が出現する部分を参照すること。

3. item_func.h で、関数の戻り値の型（数値または文字列）に応じて、Item_num_func または Item_str_func を継承するクラスを宣言する。

4. item_func.cc で、定義している関数の型（数値型または文字列型）に応じて、以下のいずれかの宣言を追加する。

```
double   Item_func_newname::val()
longlong Item_func_newname::val_int()
String  *Item_func_newname::Str(String *str)
```

   標準のアイテム（Item_num_func など）からオブジェクトを継承する場合、定義する必要があるのはおそらく上記のいずれか 1 つだけであり、残りの関数の定義は親オブジェクトにまかせることができる。たとえば、Item_str_func クラスは ::str() が返す値に対して atof() を実行する val() 関数を定義している。

5. おそらく、以下のオブジェクト関数も定義する必要がある。

```
void Item_func_newname::fix_length_and_dec()
```

   この関数は、少なくとも渡された引数に基づいて max_length を計算する必要がある。max_length は、関数が返す可能性がある最大文字数である。関数が NULL 値を返すことができない場合、この関数で maybe_null = 0 も設定する必要がある。関数は、関数の引数が NULL を返すことができるかどうかを、その引数の maybe_null 変数を調べることによって、確認できる。この実装例については、Item_func_mod::fix_length_and_dec を参照すること。

上記の関数はすべてスレッドセーフで作成する必要があります（言い換えると、mutex ロックを使用せずに関数でグローバルまたはスタティックな変数を使用してはいけません）。

NULL を返す場合、::val()、::val_int()、または ::str() で、null_value を 1 に設定して、0 を返す必要があります。

::str() オブジェクト関数の場合、さらにいくつかの注意事項があります。

- String *str 引数は、結果の格納に使用できる文字列バッファを提供する（String 型の詳細については、sql_string.h ファイルを参照すること）。
- ::str() 関数は、結果が NULL の場合は (char*) 0 を、それ以外の値の場合は結果が格納された文字列を返す必要がある。
- 現在のすべての文字列関数は、どうしても必要な場合を除いて、メモリ割り当ては一切行わないように定義されている。

13.3. MySQL への新しいプロシージャの追加

MySQL では、クライアントに送信する前にクエリのデータにアクセスして、変更できるプロシージャを、C++ を使用して定義できます。データの変更は、1 行ごと、または GROUP BY レベルで実行できます。

MySQL バージョン 3.23 には、プロシージャで実行できる処理を示すサンプルプロシージャが用意されています。

また、mylua を参照することをおすすめします。これは、LUA 言語を使用して実行時に mysqld にプロシージャをロードするツールです。

13.3.1. プロシージャの分析

analyse([max elements,[max memory]])

このプロシージャは、sql/sql_analyse.cc で定義されています。クエリから返された結果を調べて、その分析結果を返します。

- max elements (デフォルトは 256) は、analyse がカラムごとに検出した、重複を除いた値の最大数。analyse はこの値を使用して、ENUM 型が最適なカラム型かどうかを調べる。
- max memory (デフォルトは 8192) は、analyse が重複を除いた値をすべて検出する際にカラムごとに割り当てる必要がある最大メモリ容量。

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max elements,[max memory]])
```

13.3.2. プロシージャの作成

現在、プロシージャの作成に関する唯一のドキュメントはソースです。

以下のファイルを参照することで、プロシージャに関するすべての情報を入手できます。

- sql/sql_analyse.cc
- sql/procedure.h
- sql/procedure.cc
- sql/sql_select.cc

付録 A. 問題と一般的なエラー

この章では、ユーザが遭遇するいくつかの一般的な問題とエラーメッセージについて説明しています。どんな問題か見分ける方法と、問題を解決するために行わなければならないことがわかります。また、一般的な問題の適切な解決方法についても記載されています。

A.1. 問題の原因を突き止める方法

問題に遭遇した場合、まず、問題の原因となっているプログラムまたは装置を突き止めます。

- 以下の症状がある場合は、ハードウェア (メモリ、マザーボード、CPU、ハードディスク) の問題かカーネルの問題である可能性あり。
  - キーボードが動作しない。これは通常、CapsLock キーを押すことで確認できる。CapsLock ライトが変わらない場合は、キーボードを取り替える必要がある (これを行う前に、コンピュータを再起動し、キーボードに接続されているすべてのケーブルを確認する) 。
  - マウスポインタが動かない。
  - マシンがリモートマシンの ping に応答しない。
  - 複数の無関係のプログラムが正しく動作しない。
  - システムが予期せず再起動した場合 (誤ったユーザレベルのプログラムは、決してシステムを停止できない) 。

  この場合、まずケーブルをすべて確認し、その後いくつかの診断ツールを実行してハードウェアを確認する。 また、問題が解決される可能性のある、使用しているオペレーティングシステム用の修正プログラム、アップデート、または Service Pack があるかどうかを確認する。 すべてのライブラリ (glibc など) が最新であることも確認する。

  メモリ問題を早期に検出するためには ECC メモリ内蔵のマシンが有効である。
- キーボードがロックされている場合、別のユーザからマシンにログインし、kbd_mode -a を実行してロックを解除できる場合がある。
- 問題の原因を突き止めるためには、システムログファイル (/var/log/messages など) を調べる。問題が MySQL にあると考えられる場合は、MySQL のログファイルも調べる必要がある。 See 項4.10.4.「バイナリログ」。
- ハードウェアの問題ではないと考えられる場合には、問題の原因となるプログラムを突き止める必要がある。

  top、ps、taskmanager や同様のプログラムを使用して、すべての CPU を占有してしまっているプログラムや、マシンをロックしているプログラムを調べる。
- top、df や同様のプログラムを使用して、メモリ、ディスク領域、開いているファイルや他の重要なリソースが不足しているかどうかを確認する。
- 暴走プロセスが問題となっている場合は、まずその強制終了を試みる。強制終了されない場合は、オペレーティングシステムにバグがある可能性がある。

他の可能性をすべて調査し、問題の原因となっているのが MySQL サーバか MySQL クライアントであると結論づいた場

合には、弊社メーリングリストまたはサポートチームにバグレポートをお送りください。バグレポートには、システムの動作、および発生していると考えられることを、詳細に記載してください。また、問題の原因が MySQL であると考えられる理由についても記入してください。この章のすべての状況を考慮に入れてください。システムを調査したときに、問題がどのように発生したかを正確に記入してください。プログラムまたはログファイルの出力またはエラーメッセージを記入する場合は、'カット & ペースト' を使用してください。

動作していないプログラムおよびすべての症状を詳細に記述するようにしてください。過去、"システムが動作しない" としか記述されていないバグレポートを多く受け取りました。これでは、何が問題なのかわかりません。

プログラムが失敗する場合は、以下を確認すると役立ちます。

- そのプログラムがセグメンテーションフォルトしてコアダンプを作成しているか。
- プログラムが CPU を占有しているか。top でチェックする。何か重い処理を行っている可能性があるため、プログラムをしばらく動作させる。
- 問題の原因が mysqld サーバである場合は、mysqladmin -u root ping または mysqladmin -u root processlist を実行できるか。
- MySQL サーバに接続しようとすると、クライアントプログラムにどのようなメッセージが表示されるか (たとえば、mysql で試す) 。 クライアントが動かなくなるか。プログラムから何か出力があるか。

バグレポートをお送りいただく際には、このマニュアルに記載されている原則に従ってください。 See 項1.7.1.2.「質問またはバグの報告」。

A.2. MySQL 使用時によくあるエラー

このセクションでは、頻繁に発生するいくつかのエラーについて説明します。エラーの内容と問題の解決方法を説明します。

A.2.1. Access denied エラー

See 項4.3.12.「Access denied エラーの原因」。 See 項4.3.6.「権限システムはどのように機能するか」。

A.2.2. MySQL server has gone away エラー

このセクションでは、関連する Lost connection to server during query エラーもカバーしています。

MySQL server has gone away エラーの最も一般的な原因は、サーバがタイムアウトして接続がクローズしたことです。デフォルトでは、何も起きない状態が 8 時間続くと、サーバは接続をクローズします。この時間は、mysqld 開始時に wait_timeout 変数を設定することで変更できます。

MySQL server has gone away エラーが発生する一般的なもう一つの原因としては、MySQL とのコネクション上で ``close" を発行し、クローズしたコネクションでクエリを実行しようとしたことが考えられます。

スクリプトがある場合は、クライアントが自動再接続を実行するためのクエリを再発行するだけで解決します。

この場合、通常以下のエラーコードが取得されます (OS 依存です) 。

エラーコード	説明
CR_SERVER_GONE_ERROR	クライアントがサーバに問い合わせを送信できませんでした。
CR_SERVER_LOST	クライアントがサーバに書き込みを行った際エラーは発生しませんでしたが、問い合わせに対して完全な回答 (または何らかの回答) が返ってきませんでした。

誰かが kill #スレッドID# で実行中のスレッドを強制終了した場合も、このエラーが発生します。

mysqladmin version を実行し、使用可能時間を調べることによって、MySQL が強制終了されていないことを確認できます。問題が mysqld のクラッシュであれば、クラッシュの理由を見つけ出すことに注力してください。 この場合、まずクエリを再発行して、MySQL が再び強制終了されるかどうかをチェックする必要があります。 See 項A.4.1.「MySQL が何度もクラッシュする場合に行うこと」。

間違ったクエリか大きすぎるクエリをサーバに送信した場合も、このエラーが発生する可能性があります。mysqld は大きすぎるか異常のあるパケットを取得すると、クライアントに何か問題が発生したとみなし、接続をクローズします。大きなクエリが必要な場合 (たとえば、大きな BLOB カラムを使用している場合)、mysqld を -O max_allowed_packet=# オプション (デフォルト 1MB) で起動して、クエリ制限を引き上げることができます。拡張メモリは要求に応じて割り当てられます。そのため、mysqld は、ユーザが大きなクエリを発行するときや、mysqld が大きな結果レコードを返す必要のあるときだけ、メモリを増やして割り当てます。

クライアントが 4.0.8 より古く、サーバが 4.0.8 以上の場合、またはその逆の場合、16MB 以上のパケットを送信すると接続が失われます。

この問題についてバグレポートを作成する際には、必ず以下の情報を記入してください。

- MySQL が強制終了されたかどうか (この情報は、hostname.err ファイル にある)。 See 項A.4.1.「MySQL が何度もクラッシュする場合に行うこと」。
- 特定のクエリが、mysqld と、クエリ実行前に CHECK TABLE によるチェックが実行された関連テーブルを強制終了する場合、このテストケースを実行することができるか。 See 項E.1.6.「テーブルが破損した場合にテストケースを作成する」。
- MySQL サーバにおける wait_timeout 変数の値は何か。 この値は、mysqladmin variables で確認することができる。
- mysqld を --log で実行し、発行したクエリがログに出力されているかどうかチェックしたか。

See 項1.7.1.2.「質問またはバグの報告」。

A.2.3. Can't connect to [local] MySQL server エラー

Unix上の MySQL は mysqld サーバに次の2つの方法で接続することができます。Unix ソケット、これはファイルシステム上のファイル (デフォルトでは /tmp/mysqld.sock) を通して接続します。または TCP/IPで、これはポート番号を通して接続します。 Unix ソケットは、TCP/IP より高速ですが、同じコンピュータのサーバに接続する場合にしか使用できません。Unix ソケットは、ホスト名を指定しない場合、または特別なホスト名 localhost を指定する場合に使用されます。

Windows では、mysqld サーバが 9x/Me で動作している場合、TCP/IP を介してのみ接続できます。サーバが NT/2000/XP で動作しており、mysqld が --enable-named-pipe で起動している場合は、名前付きパイプと接続することもできます。名

前付きパイプの名前は MySQL です。 mysqld に接続する際にホスト名を与えない場合は、MySQL クライアントは最初に名前付きパイプに接続しようとします。これが正しく機能しない場合は、TCP/IP ポートに接続します。ホスト名として . を使用することにより、Windows の名前付きパイプを強制的に使用できます。

エラー (2002) Can't connect to ... は、通常、MySQL サーバがシステム上で動作していないか、間違ったソケットファイルや TCP/IP ポートを使用して、mysqld サーバに接続しようとした場合に発生します。

サーバ上で mysqld というプロセスが動作しているかどうか、(ps か、Windows の場合はタスクマネージャを使用して) 最初に確認してください。mysqld プロセスがなければ、これを起動してください。 See 項2.4.2.「MySQL サーバの起動に関する問題」。

mysqld プロセスが動作している場合は、さまざまな接続を試してサーバを確認できます (ポート番号とソケットのパス名は、当然セットアップ時と異なる可能性があります) 。

```
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
```

hostnameコマンドは、フォワードクォートではなくバッククォートで囲んでください。これによって、hostname の出力 (つまり現在のホスト名) が mysqladmin コマンドに代入されます。

Can't connect to local MySQL server エラーが発生する理由として、以下のことが考えられます。

- mysqld が動作していない。
- MIT-pthreads を使用するシステム上で実行している。 ネイティブスレッドを持たないシステムで実行している場合、mysqld は MIT-pthreads パッケージを使用する。See 項2.2.3.「MySQL がサポートしているオペレーティングシステム」。ただし、すべてのMIT-pthreads バージョンが Unix ソケットをサポートしているわけではない。ソケットサポートのないシステムでは、サーバに接続する際に、常にホスト名を明示的に指定する必要がある。以下のコマンドを使用して、サーバへの接続を確認する。

  ```
  shell> mysqladmin -h `hostname` version
  ```

- 誰かが、mysqld が使用する Unix ソケット (デフォルトでは /tmp/mysqld.sock) を削除した。MySQL ソケットを削除する cron ジョブ (たとえば、/tmp ディレクトリから古いファイルを削除するジョブ) を行っている可能性がある。mysqladmin version を実行し、mysqladmin が使用するソケットが本当に存在するかどうかを常にチェックすることができる。この場合の修正方法は、mysqld.sock を削除しないように cron ジョブを変更するか、ソケットを別の場所に移動させることである。 See 項A.4.5.「MySQL ソケットファイル /tmp/mysql.sock の保護または変更方法」。
- --socket=/path/to/socket オプションを指定して、mysqld サーバを開始した。サーバのソケットパス名を変更する場合は、MySQL クライアントに新しいパスを通知する必要がある。パスを通知するには、ソケットパスを引数としてクライアントに提供する。 See 項A.4.5.「MySQL ソケットファイル /tmp/mysql.sock の保護または変更方法」。
- Linux を使用中にスレッドが 1 つ消滅した (コアダンプした) 。この場合、(たとえば、新しい MySQL サーバを実行する前に mysql_zap スクリプトを使用して) 他の mysqld スレッドを強制終了する必要がある。 See 項A.4.1.「MySQL が何度もクラッシュする場合に行うこと」。
- ソケットファイルを保持しているディレクトリ、またはソケットファイル自体への読み取り権限と書き込み権限がない

可能性がある。この場合、アクセス可能なディレクトリを使用するように、ディレクトリまたはファイルに対する権限を変更するか、mysqld を再起動する必要がある。

エラーメッセージ Can't connect to MySQL server on some_hostname が発生した場合は、以下を行って問題を突き止めることができます。

- telnet your-host-name tcp-ip-port-number を実行してサーバが立ち上がっているかどうかを確認し、Enter キーを数回押す。このポートで MySQL サーバが動作している場合は、動作中の MySQL サーバのバージョン番号を含むレスポンスが返ってくる。telnet: Unable to connect to remote host: Connection refused などのエラーが発生する場合は、サーバは所定のポートで動作していない。
- ローカルマシンの mysqld デーモンに接続し、mysqladmin variables で mysqld が使用するように設定された TCP/IP ポート（変数 port）を確認する。
- mysqld サーバが、--skip-networking オプションで起動されていないか確認する。

A.2.4. Client does not support authentication protocol エラー

MySQL 4.1 では、パスワードハッシュアルゴリズムに基づく認証プロトコルが使用されていますが、これは旧クライアントが使用しているものと互換性がありません。 サーバを 4.1 にアップグレードすると、旧クライアントで接続しようとした場合に、以下のエラーメッセージが発生する可能性があります。

```
shell> mysql
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

この問題を解決するには、以下のいずれかを行ってください。

- 全クライアントプログラムをアップグレードして、4.1.1 以降のクライアントライブラリを使用するようにする。
- 4.1 より前のクライアントから接続する場合は、旧パスワードでユーザアカウントを使用する。
- 4.1 より前のクライアントを必要とするユーザをリセットし、旧パスワードを使用するようにする。

```
mysql> UPDATE user SET Password = OLD_PASSWORD('mypass')
    -> WHERE Host = 'some_host' AND User = 'some_user';
mysql> FLUSH PRIVILEGES;
```

- サーバが旧パスワードハッシュアルゴリズムを使用するように指定する。

  1. --old-passwords を指定して mysqld を開始する。
  2. ロングパスワードを持っているユーザすべてのパスワードを設定する。これらのユーザは、以下で調べることができる。

     ```
     SELECT * FROM mysql.user WHERE LEN(password) > 16;
     ```

パスワードハッシュと認証の背景については、項4.3.11.「MySQL 4.1 のパスワードハッシュ」 を参照してください。

A.2.5. Host '...' is blocked エラー

以下のエラーが発生した場合

```
Host 'hostname' is blocked because of many connection errors.
Unblock with 'mysqladmin flush-hosts'
```

これは、mysqld が 'hostname' ホストから多くの接続エラー (max_connect_errors) を受けた場合に発生します。max_connect_errors 大量発生後、mysqld は何か問題 (クラッカーからの攻撃など) が発生したと判断し、このホストからの接続を拒否するようにします。これを解除するには、mysqladmin flush-hosts コマンドを実行します。

デフォルトでは、接続エラーが 10 回発生すると、mysqld はそのホストを拒否します。 この値は、以下のようにサーバを開始することで簡単に変更できます。

```
shell> mysqld_safe -O max_connect_errors=10000 &
```

特定のホストに対してこのエラーメッセージが発生する場合は、まず、そのホストからの TCP/IP 接続に問題がないか確認してください。TCP/IP 接続が機能していない場合は、max_connect_errors 変数の値を増やしても効果はありません。

A.2.6. Too many connections エラー

MySQL に接続しようとして Too many connections エラーが発生する場合は、すでに mysqld サーバに接続している max_connections クライアントが存在しています。

デフォルト (100) より多い接続を行う場合は、max_connections 変数の値を 100 より大きくして、mysqld を再起動する必要があります。

実際は、mysqld では (max_connections+1) クライアントの接続が許可されています。最後の接続は、SUPER 特権のあるユーザ用に予約されています。一般ユーザにこの特権を与えないことによって (一般ユーザにこの特権は必要ありません)、この特権のある管理者はログインして、SHOW PROCESSLIST を使用して問題を見つけることができます。 See 項4.6.8.6.「SHOW PROCESSLIST」。

MySQL 接続の最大数は、スレッドライブラリが特定のプラットフォームでどの程度まで有用であるかに依存します。Linux または Solaris では、使用している RAM のサイズと、クライアントが何を実行しているかによって、500 〜 1000 の同時接続をサポートできます。

A.2.7. Some non-transactional changed tables couldn't be rolled back エラー

ROLLBACK を実行しようとしたときに Warning: Some non-transactional changed tables couldn't be rolled back が発生した場合、トランザクションで使用したテーブルの中にトランザクションをサポートしていないものがあることを示しています。これら非トランザクションテーブルは、ROLLBACK ステートメントからの影響を受けません。

最も一般的には、このエラーが発生するのは、mysqld バイナリでサポートされていない型のテーブルを作成しようとしたときです。 mysqld がテーブル型をサポートしていない場合 (またはスタートアップオプションでテーブル型が無効化されている場合)、代わりに、要求したものと最も類似したテーブル型 (おそらく MyISAM) が作成されます。

以下のコマンドでテーブルのテーブル型を確認できます。

SHOW TABLE STATUS LIKE 'table_name'。 See 項4.6.8.2.「SHOW TABLE STATUS」。

以下のコマンドで mysqld バイナリがサポートする拡張子を確認できます。

SHOW VARIABLES LIKE 'have_%'。 See 項4.6.8.4.「SHOW VARIABLES」。

A.2.8. Out of memory エラー

クエリを発行し、以下のようなエラーが発生した場合

```
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
```

エラーが MySQL クライアント mysql を参照していることに注意してください。このエラーの原因は、クライアントにすべての結果を格納するだけのメモリがないことです。

問題を解決するには、まず、クエリが正しいことを確認してください。そのクエリは大量のレコードを返すものでしょうか？そうであれば、mysql --quick を使用できます。これは、結果セットを取り出すために mysql_use_result() を使用します。これによってクライアントの負荷が小さくなります（サーバ側では負荷が大きくなります）。

A.2.9. Packet too large エラー

MySQL クライアントまたは mysqld サーバが max_allowed_packet バイトより大きいパケットを受け取った場合、Packet too large エラーが発生し、接続がクローズされます。

MySQL 3.23 で使用できる最も大きなパケットは 16M です（クライアントおよびサーバプロトコルの制限によります）。MySQL 4.01 以上では、パケットの大きさは、サーバ上のメモリ容量でのみ制限されます（理論上は最大 2G）。

1 つの通信パケットは、MySQL サーバに送信される単一の SQL ステートメントか、クライアントに送信される単一行です。

MySQL クライアントまたは mysqld サーバが max_allowed_packet バイトより大きいパケットを受け取った場合は、Packet too largeエラーが発生し、接続がクローズされます。クライアントの中には、通信パケットが大きすぎると Lost connection to MySQL server during query エラーを発生するものもあります。

クライアントとサーバには、共に独自の max_allowed_packet 変数があります。大きなパケットを扱う場合は、クライアントとサーバ両方の変数を増やす必要があります。

メモリは必要な場合にのみ割り当てられるため、この変数を増やしても安全です。この変数は、クライアントとサーバ間の不正なパケットを捕捉したり、メモリ不足になってしまうような大きなパケットを誤って使用したりしないための予防策です。

mysql クライアントを使用している場合は、クライアントを mysql --set-variable=max_allowed_packet=8M で開始することで、より大きなバッファを指定することができます。 他のクライアントには、この変数を設定する別の方法があります。 MySQL 4.0 以降、--set-variable は使用されなくなっています。代わりに --max-allowed-packet=8M を使用してください。

オプション設定ファイルを使用すると、mysqld で max_allowed_packet をより大きなサイズに設定することができます。たとえば、MEDIUMBLOB の許容範囲いっぱいのデータをテーブルに入れる場合、--set-variable=max_allowed_packet=16M オプションを指定してサーバを起動します。

大きな BLOB を使用している場合、大きなパケットに関する見慣れない問題が発生する可能性がありますが、これは、mysqld に、そのクエリを処理できるだけの十分なメモリへのアクセス権がないことが原因です。その場合は、mysqld_safe スクリプトの始めに ulimit -d 256000 を追加して、mysqld を再起動してください。

A.2.10. 通信エラー/Aborted connection

MySQL 3.23.40 から、mysqld を --warnings で開始した場合のみ、Aborted connection エラーが発生します。

エラーログに以下のようなエラーが含まれている場合は、

```
010301 14:38:23  Aborted connection 854 to db: 'users' user: 'josh'
```

See 項4.10.1.「エラーログ」。

以下が発生したと考えられます。

- クライアントプログラムが終了前に mysql_close() を呼び出さなかった。
- クライアントが何の要求もせずに、wait_timeout または interactive_timeout より多くの時間スリープ状態であった。See 項4.6.8.4.「SHOW VARIABLES」。
- クライアントプログラムが、転送中に突然終了した。

上記のことが発生した場合、サーバ変数 Aborted_clients の値が増えます。

以下の場合、サーバ変数 Aborted_connects の値が増えます。

- 接続パケットに正しい情報が含まれていない場合
- ユーザにデータベースに接続する権限がない場合
- ユーザが間違ったパスワードを使用した場合
- 接続パケットの取得に、connect_timeout で指定されている秒数より多く要した場合 See 項4.6.8.4.「SHOW VARIABLES」。

上記のことは、不正ユーザがデータベースに侵入しようとしている可能性を示しています。

中止されたクライアントおよび中止された接続エラーのその他の原因

- Linux での Ethernet プロトコルの使用（半二重と全二重）。 多くの Linux Ethernet ドライバには、このバグがある。2 つのマシン間で ftp を使用して大きなファイルを転送し、このバグについてテストする必要がある。 転送が burst-pause-burst-pause ... モードで行われている場合は、Linux 二重シンドロームに陥っている。 唯一の解決策は、ネットワークカードとハブ/スイッチの二重モードを、全二重または半二重に切り替え、その結果をテストし最も良い設定を行うことである。
- 読み取りの中断の原因になるスレッドライブラリの問題。
- TCP/IP の設定間違い。

- 不良 Ethernet、ハブ、スイッチ、ケーブルなど。これは、ハードウェアを交換してみることでしか正しく診断できない。
- max_allowed_packet が小さすぎるか、クエリが mysqld に割り当てられたメモリより多くのメモリを必要としている。 See 項A.2.9.「Packet too large エラー」。

A.2.11. The table is full エラー

このエラーが発生するケースとしては、以下のようなさまざまなケースが考えられます。

- 旧バージョン（3.23.0 より前）の MySQL を使用していて、メモリ内のテンポラリテーブルが tmp_table_size バイトより大きくなったとき。 この問題を回避するには、-O tmp_table_size=# オプションを使用して mysqld でテンポラリテーブルサイズを大きくするか、問題のクエリを発行する前に、SQL オプション SQL_BIG_TABLES を使用する。 See 項5.5.6.「SET 構文」。

  また、--big-tables オプションを指定して mysqld を開始することもできる。 これは、すべてのクエリに SQL_BIG_TABLES を使用しているのとまったく同じ。

  MySQL バージョン 3.23 では、メモリ内のテンポラリテーブルが tmp_table_size より大きくなると、サーバがそのテーブルを自動的にディスクベースの MyISAM テーブルに変換した。
- InnoDB テーブルを使用しており、InnoDB のテーブルスペースが不足している。この問題を解決するには、InnoDB のテーブルスペースを拡張する。
- サイズが 2GB のファイルしかサポートしていない OS で ISAM または MyISAM テーブルを使用しており、データファイルまたはインデックスファイルが制限に達した。
- MyISAM テーブルを使用しており、必要なデータまたはインデックスサイズが、MySQL がポインタを割り当てたものより大きくなっている（MAX_ROWS を CREATE TABLE に指定しない場合は、MySQL は 4G のデータを保持できる分だけポインタを割り当てる）。

  最大データとインデックスサイズを確認できる。確認するには、以下のコマンドを実行するか、

  ```
  SHOW TABLE STATUS FROM database LIKE 'table_name';
  ```

  myisamchk -dv database/table_name を使用する。

  これで問題がある場合は、以下の方法で解決できる。

  ```
  ALTER TABLE table_name MAX_ROWS=1000000000 AVG_ROW_LENGTH=nnn;
  ```

  この場合、MySQL はレコード数だけに基づいて必要なスペースを最適化することができないので、BLOB/TEXT フィールドをもつテーブルに AVG_ROW_LENGTH を指定することが必要になる。

A.2.12. Can't create/write to file エラー

以下のようなクエリのエラーが発生する場合

```
Can't create/write to file '\\sqla3fe_0.ism'.
```

MySQL が、特定のテンポラリディレクトリに結果セット用のテンポラリファイルを作成できないことを示しています (上記のエラーは、Windows での典型的なエラーメッセージですが、Unix でのエラーメッセージもよく似たメッセージになります) 。 解決するには、--tmpdir=path を指定して mysqld を起動するか、オプション設定ファイルに以下のコードを追加します。

```
[mysqld]
tmpdir=C:/temp
```

c:\\temp ディレクトリが存在していると仮定しています。 See 項4.1.2.「my.cnf オプション設定ファイル」。

また perror で、取得するエラーコードを確認します。1つの原因としてディスクがいっぱいになっていることも考えられます。

```
shell> perror 28
Error code  28:  No space left on device
```

A.2.13. クライアントでの Commands out of sync エラー

クライアントコードの中に Commands out of sync; you can't run this command now がある場合は、間違った順序でクライアント関数を呼び出しています。

たとえば、mysql_free_result() を呼び出す前に mysql_use_result() を使って新しいクエリを実行しようとすると、上記のことが起こる可能性があります。 また、データを返す 2つのクエリを、その間に mysql_use_result() や mysql_store_result() を呼び出さないで実行した場合にも起こる可能性があります。

A.2.14. Ignoring user エラー

次のエラーが発生する場合

Found wrong password for user: 'some_user@some_host'; ignoring user

mysqld が再起動されたとき、またはアクセス権テーブルが再ロードされたときに、user テーブルに無効なパスワードのエントリが見つかったことを示しています。結果として、このエントリはアクセス権システムから無視されます。

この問題の考えられる原因と修正方法

- 旧 user テーブルで新バージョンの mysqld を実行している可能性がある。 mysqlshow mysql user を実行し、パスワードフィールドが 16 文字より短いかどうかを調べることによって、このことを確認できる。短い場合、scripts/add_long_password スクリプトを実行して、この状態を修正できる。
- ユーザのパスワードは旧パスワード (8 文字長) であるが、mysqld が --old-protocol オプションで開始されなかった。user テーブルのユーザを新規パスワードで更新するか、mysqld を --old-protocol を指定して再起動する。
- PASSWORD() 関数を使用せずに、user テーブルにパスワードを指定した。mysql を使用して user テーブルのユーザを新規パスワードで更新する。このとき、必ず PASSWORD() 関数を使用すること。

```
mysql> UPDATE user SET password=PASSWORD('your password')
    ->           WHERE user='XXX';
```

A.2.15. Table 'xxx' doesn't exist エラー

エラー Table 'xxx' doesn't exist または Can't find file: 'xxx' (errno: 2) が発生する場合、xxx という名前のカレントデータベースにテーブルがないことを示しています。

MySQL では、データベースとテーブルの格納にディレクトリとファイルを使用するため、データベースとテーブル名は、大文字と小文字の区別がある（ケース依存）ことに注意してください（Windows では、データベースとテーブル名に大文字小文字の区別がありません。しかし、クエリ内の特定テーブルへの参照では大文字と小文字を完全に合わせる必要があります）。

SHOW TABLES を使用して、カレントデータベースにあるテーブルを確認できます。 See 項4.6.8.「SHOW 構文」。

A.2.16. Can't initialize character set xxx エラー

以下のエラーが発生した場合

```
MySQL Connection Failed: Can't initialize character set xxx
```

以下のことが考えられます。

- キャラクタセットがマルチバイトのキャラクタセットであるが、クライアントではキャラクタセットがサポートされていない。

  この場合、クライアントを --with-charset=xxx または --with-extra-charsets=xxx で再コンパイルする。 See 項2.3.3.「一般的な configure オプション」。

  標準 MySQL バイナリはすべて --with-extra-character-sets=complex でコンパイルされている。これによって、マルチバイトのキャラクタセットすべてがサポートされる。 See 項4.7.1.「データおよびソート用キャラクタセット」。

- キャラクタセットが単純なキャラクタセットで、それが mysqld に組み込まれておらず、キャラクタセットの定義ファイルがクライアントの予想する場所にない。

  この場合、以下のようにする。

  - キャラクタセットをサポートするように、クライアントを再コンパイルする。 See 項2.3.3.「一般的な configure オプション」。
  - クライアントに、キャラクタセットの定義ファイルの場所を通知する。多くのクライアントに対して、--character-sets-dir=path-to-charset-dir オプションを使用して通知するできる。
  - 文字定義ファイルを、クライアントがその定義ファイルがあると予想しているパスにコピーする。

A.2.17. File Not Found エラー

MySQL から ERROR '...' not found (errno: 23)、Can't open file: ... (errno: 24)、または errno 23 または errno 24 とともに他のエラーを受け取った場合、MySQL に十分なファイル記述子が割り当てられていないことを示しています。perror ユーティリティを使用して、エラー番号が何を意味しているか、その内容を取得することができます。

```
shell> perror 23
File table overflow
```

```
shell> perror 24
Too many open files
shell> perror 11
Resource temporarily unavailable
```

ここでの問題は、mysqld が非常に多くのファイルを同時に開こうとしていることです。mysqld が同時に多くのファイルを開かないように指定するか、mysqld で使用できるファイル記述子の数を増やすことができます。

mysqld で同時に開くファイルを少なくするために、mysqld_safe に -O table_cache=32 オプション (初期値は 64) を指定して、テーブルキャッシュを小さくすることができます。max_connections の値を減らすことで、オープンファイル数 (初期値は 90) も少なくなります。

mysqld で使用できるファイル記述子の数を変更するために、mysqld_safe に --open-files-limit=# オプション、または mysqld に -O open-files-limit=# オプションを使用することができます。 See 項4.6.8.4.「SHOW VARIABLES」。 これを行う最も簡単な方法は、オプション設定ファイルにオプションを追加することです。 See 項4.1.2.「my.cnf オプション設定ファイル」。 これをサポートしていない旧バージョンの mysqld を使用している場合は、mysqld_safe スクリプトを編集することができます。スクリプトにはコメントアウトされた ulimit -n 256 行があります。'#' 文字を削除してこの行のコメントを解除し、数字 256 を変更することができます。これによって、 mysqld で使用できるファイル記述数も変わります。

ulimit (および open-files-limit) によって、ファイル記述子の数を増やすことができます。しかし、オペレーティングシステムで定められた上限までしか増やせません。また 'ハード' リミットが存在し、ルートとして mysqld_safe または mysqld を開始した場合に限り上書きすることができます (この場合、--user=... オプションも使用する必要があります) 。各プロセスで利用できるファイル記述子の数の OS 制限を増やす必要がある場合は、そのオペレーティングシステムのドキュメントを参照してください。

tcsh シェルを実行している場合は、ulimit は機能しません。 tcsh は、現在の制限について問い合わせると、間違った値を返します。この場合、mysqld_safe を sh で開始する必要があります。

A.3. インストール関連の問題

A.3.1. MySQL クライアントライブラリにリンク時の問題

プログラムをリンクするときに、mysql_ で始まる未参照シンボルに関する以下のようなエラーが発生する場合

```
/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'
```

これは、リンク行の最後に -Lpath-to-the-mysql-library-lmysqlclient を追加すると解決できます。

uncompress または compress 関数で undefined reference エラーが発生する場合は、リンク行の最後に -lz を追加して再度実行してください。

システムにあるはずの関数 (例: connect) に対して undefined reference エラーが発生する場合は、該当する関数を man ページで確認し、そのライブラリをリンク行に追加するかどうか判断してください。

以下のように、システムに存在しない関数で undefined reference エラーが発生する場合

```
mf_format.o(.text+0x201): undefined reference to `__lxstat'
```

これは、通常、使用しているシステムと 100% 互換でないシステム上でライブラリがコンパイルされていることを示しています。この場合、最新の MySQL ソースディストリビューションをダウンロードし、それを独自にコンパイルしてください。 See 項2.3.「MySQL ソースディストリビューションのインストール」。

プログラムを実行しようとしているが、mysql_ で始まるエラーか、mysqlclient ライブラリが見つからない未参照シンボルのエラーが発生する場合は、システムが libmysqlclient.so 共有ライブラリを見つけられないということを示しています。

これを解決するには、以下のいずれかの方法で、関数が組み込まれている共用ライブラリを検索するように、システムに指示します。

- LD_LIBRARY_PATH 環境変数に libmysqlclient.so が存在するディレクトリのパスを追加する。
- LD_LIBRARY 環境変数に libmysqlclient.so が存在するディレクトリのパスを追加する。
- libmysqlclient.so をシステムが検索する場所（ たとえば /lib ）にコピーし、ldconfig を実行して共有ライブラリ情報を更新する。

この問題を解決するもう 1つの方法として、-static を使用するか、コードをリンクする前に MySQL の動的ライブラリを削除して、プログラムを静的にリンクする方法があります。2 番目のケースでは、別のプログラムが動的ライブラリを使用していないことを確認する必要があります。

A.3.2. 一般ユーザで MySQL を実行する方法

MySQL サーバ mysqld は、どんな Unix ユーザでも開始、実行できます。 mysqld を Unix ユーザ user_name で実行できるように変更するには、以下を行ってください。

1. サーバが動作している場合は、それを停止します（ mysqladmin shutdown を使用します ）。
2. データベースのディレクトリとファイルを変更して、その中のファイルを読み込んだり書き込んだりする権限を user_name に与えます（ 場合によっては Unix root ユーザとして行う必要があります ）。

   ```
   shell> chown -R user_name /path/to/mysql/datadir
   ```

   MySQL データディレクトリ内のディレクトリまたはファイルがシンボリックリンクの場合は、リンクに従い、そのファイルやディレクトリが指しているディレクトリとファイルも変更する必要があります。chown -R では、シンボリックリンク先まで変更できません。
3. サーバを user_name で開始するか、MySQL バージョン 3.22 以降を使用している場合は、mysqld をUnix root ユーザで開始して、--user=user_name オプションを使用します。mysqld は、接続を許可する前に、Unix ユーザ user_name での実行に切り替わります。
4. システム起動時にサーバを特定のユーザ名で自動的に開始するには、ユーザ名を指定する user 行を、/etc/my.cnf オプション設定ファイルかサーバのデータディレクトリにある my.cnf オプション設定ファイルの [mysqld] グループに追加します。たとえば、以下のようにします。

   ```
   [mysqld]
   user=user_name
   ```

この時点で、mysqld プロセスは Unix ユーザ user_name として正常に動作します。ただし、アクセス権テーブルの内容は変わりません。デフォルトでは (アクセス権テーブルのインストールスクリプト mysql_install_db の実行直後) 、mysql データベースへのアクセスまたはデータベースの作成や破棄の権限を持つ唯一のユーザは MySQL ユーザ root です。これらのアクセス権を変更しなかった場合、そのアクセス権はそのまま保持されます。root 以外の Unix ユーザとしてログインしている場合も、MySQL root ユーザとしてMySQLにアクセスできます。そのために行うことは、-u root オプションをクライアントプログラムに指定するだけです。

コマンドラインで -u root を指定して、root としてMySQL にアクセスすることと、Unix root ユーザ、または実際には別の Unix ユーザとして MySQL を実行することとは何の関係もありません。MySQL のアクセス権とユーザ名は、Unix ユーザ名とは完全に別のものです。Unix ユーザ名と唯一関連があるのは、クライアントプログラムを呼び出すときに -u オプションを指定しない場合に、クライアントが MySQL ユーザ名として Unix ログイン名を使用して接続を試みるということだけです。

Unix ボックスそのものが保護されていない場合は、少なくとも、アクセステーブルの MySQL root ユーザにパスワードを設定しておく必要があります。 そうしなければ、マシンにアカウントのあるユーザは、mysql -u root db_name を実行して思い通りのことを行えます。

A.3.3. ファイルアクセス権の問題

ファイルアクセス権に問題がある場合、たとえばテーブルを作成する際に mysql が以下のエラーメッセージを発行した場合

```
ERROR: Can't find file: 'path/with/filename.frm' (Errcode: 13)
```

mysqld 起動時に、環境変数 UMASK が正しく設定されていなかった可能性があります。umask 初期値は 0660 です。以下のように mysqld_safe を実行して、この動作を変更することができます。

```
shell> UMASK=384  # = 600 in octal
shell> export UMASK
shell> /path/to/mysqld_safe &
```

デフォルトでは、MySQL はアクセス権タイプ 0700 のデータベースと RAID ディレクトリを作成します。UMASK_DIR 変数を設定して、この動作を変更することができます。この変数を設定すると、UMASK と UMASK_DIR が組み合わされて新しいディレクトリが作成されます。たとえば新しいディレクトリすべてにグループアクセス権を与える場合は、以下を実行します。

```
shell> UMASK_DIR=504  # = 8進数で 770
shell> export UMASK_DIR
shell> /path/to/mysqld_safe &
```

MySQL バージョン 3.23.25 以降では、UMASK と UMASK_DIR の値がゼロで始まる場合、MySQL はその値を8進数と仮定します。

See 付録 F. 環境変数。

A.4. 管理関連の問題

A.4.1. MySQL が何度もクラッシュする場合に行うこと

MySQL バージョンはすべて、リリース前に多くのプラットフォームでテストされています。これは MySQL にバグがないというわけではありませんが、バグがあったとしても、非常に少なく見つけるのが困難です。何か問題がある場合に、システムをクラッシュさせているものを正確に調査することは、この問題の早期解決に役立ちます。

まず、問題は mysqld デーモンが死んでいることにあるのか、または問題がクライアントに関係しているのかを判断する必要があります。mysqladmin version を実行して、mysqld サーバが稼動していた時間を確認することができます。mysqld が死んでいる場合は、ファイル mysql-data-directory/`hostname`.err を見ることによりその原因を見つけることができます。 See 項4.10.1.「エラーログ」。

システムの中には、このファイルに mysqld が死んだ際のスタックトレースがあるものがあり、resolve_back_stack で問題を解決できます。See 項E.1.4.「スタックトレースの使用」。.err ファイルに書き込まれている変数値は、常に 100 パーセント正確であるとは限りません。

MySQL のクラッシュの多くは、インデックスファイルやデータファイルの破壊が原因です。 MySQL は、すべての SQL ステートメントの後で write() システムコールを呼び出し、ディスク上のデータを更新します。クライアントには、その後その結果が通知されます (delay_key_write で実行している場合で、データの書き込みだけの場合にはこの限りではありません)。 つまり、OS によってフラッシュされていないデータがディスクに確実に書き込まれるため、mysqld がクラッシュした場合でもデータは安全です。--flush を指定して mysqld を開始することによって、各 SQL コマンド実行後、強制的にすべてをディスクと同期させることができます。

つまり、以下のことがなければ、通常は破壊されたテーブルを得ることはありません。

- 誰かまたは何かが、更新中に mysqld またはマシンを強制終了した。
- 更新中に mysqld が死ぬようなバグを見つけた。
- テーブルを適切にロックしないで、mysqld 外でデータまたはインデックスファファイルを操作している。
- 優れたファイルシステムロック (通常、lockd デーモンによって処理される) をサポートしていないシステム上で、同じデータに対して、多くの mysqld サーバを実行しているか、--skip-external-locking で複数サーバを実行している。
- mysqld を混乱させるような不正なデータを含んだ、破損したインデックスまたはデータファイルがある。
- データストレージのコードにバグを発見した。これはおそらくあり得ないことだが、可能性としてはある。この場合、修正されたテーブルのコピーに対して ALTER TABLE を使用して、ファイルタイプを別のデータストレージエンジンに変更することができる。

クラッシュの原因を見つけるのは非常に難しいので、まず、他で動作しているものが自分の環境でクラッシュするかどうかを確認してください。以下を試してください。

mysqladmin shutdown で mysqld デーモンを停止し、全テーブルで myisamchk --silent --force */*.MYI を実行した後、mysqld デーモンを再起動する。必ず、クリーンな状態から実行すること。 See 章 4. データベース管理。

- mysqld --log を使用して、ログの情報から、特定のクエリがサーバを終了させるかどうかを判断する。バグの 95% が特定のクエリに関係している。通常、これは、MySQL 再実行直前のログファイルにある最後のクエリの 1 つである。See 項4.10.2.「一般クエリログ」。 クエリを実行する直前にすべてのテーブルを確認しても、クエリの 1 つで MySQL を何度も強制終了させることができる場合、バグを突き止められているので、バグレポートを発行する必要が

ある。 See 項1.7.1.3.「バグまたは問題を報告する方法」。

- 問題の再現に使用するテストケースを作成する。 See 項E.1.6.「テーブルが破損した場合にテストケースを作成する」。
- 組み込まれている mysql-test テストと MySQL ベンチマークを実行する。See 項13.1.2.「MySQL テストスイート」。これによって MySQL を正確にテストできる。ベンチマークにアプリケーションをシミュレートするコードを追加することもできる。ベンチマークは、ソースディストリビューションまたはバイナリディストリビューションの bench ディレクトリか、MySQL インストールディレクトリの sql-bench ディレクトリにある。
- fork_test.pl と fork2_test.pl を実行する。
- デバッグ用に MySQL を設定している場合は、何か問題が発生した際に、考えられるエラー情報の収集が非常に容易になる。 configure に --with-debug オプションか --with-debug=full オプションを指定して MySQL を再設定し、その後再コンパイルする。 See 項E.1.「MySQL サーバのデバッグ」。
- デバッグ用に MySQL を設定すると、安全なメモリアロケータが組み込まれ、エラーを見つけることができる。また、何が発生しているかについて、多くの情報が提供される。
- オペレーティングシステムの最新のパッチを適用しているかどうか確認する。
- mysqld にオプション --skip-external-locking を指定する。システムの中には、lockd ロックマネージャが適切に動作しないものがある。--skip-external-locking オプションによって、mysqld に外部ロックを使用しないように指示することができる（つまり、同じデータで 2 つの mysqld サーバを実行することができない。myisamchk を使用する場合は注意が必要だが、テストとしてオプションを使用する場合は有益である）。
- mysqld が動作しているように見えるのに応答がなかった時、mysqladmin -u root processlist を実行したか。mysqld は消滅しているように見えても、消滅していない場合がある。すべての接続が使用中であるか、内部ロック問題の可能性がある。 mysqladmin processlist は、通常、このような場合であっても接続を行うことができ、現在の接続数やその状況に関して有益な情報を取得できる。
- 他のクエリを実行中に、別のウィンドウで mysqladmin -i 5 status または mysqladmin -i 5 -r status を実行し、統計を生成する。
- 以下を実行してみる。

  1. gdb から（または別のデバッガで）、mysqld を起動する。 See 項E.1.3.「mysqld のデバッグ（gdb 使用）」。
  2. テストスクリプトを実行する。
  3. 3つの最下位レベルで、バックトレースとローカル変数を出力する。gdb では、mysqld が gdb 内部でクラッシュした場合、以下のコマンドでこの出力を実行できる。

     ```
     backtrace
     info local
     up
     info local
     up
     info local
     ```

     gdb では、info threads により、存在しているスレッドを調べ、thread #（# はスレッド ID）で特定のスレッドにスイッチすることができる。

- Perl スクリプトでアプリケーションをシミュレートし、MySQL をクラッシュさせたり不正な動作をさせたりする。
- バグレポートを送付する。See 項1.7.1.3.「バグまたは問題を報告する方法」。詳細なレポートにすること。MySQL は多くの人が使用しているため、クラッシュがあなたのコンピュータだけにある何らかの原因に起因している可能性がある（たとえば、特定のシステムライブラリに関連するエラー）。
- 可変長レコードを持つテーブルに問題があり、BLOB/TEXT カラム（BLOB/TEXT カラムのみ）を使用していない場合、ALTER TABLE で VARCHAR すべてを CHAR に変更することができる。これによって、MySQL は固定サイズのレコードを使用する。 固定サイズのレコードは余分な領域を少し取るだけだが、破壊に対して非常に大きな耐性がある。

  現在の可変長レコードのコードは、少なくとも 3 年間は何の問題もなく MySQL AB で使用されてきた。しかし本来、可変長レコードはエラーが発生しやすく、上記のことが役立つかどうか試してみるのも有益である。

A.4.2. 忘れたルートパスワードをリセットする方法

MySQL に root パスワードを設定しないと、root として接続した場合に、サーバがパスワードを一切要求しなくなります。ユーザごとに常にパスワードを設定することをお勧めします。 See 項4.3.2.「MySQL のクラッカー対策」。

root パスワードを設定した後で、それを忘れてしまった場合、以下の手順で新しいパスワードを設定することができます。

1. mysqld サーバに kill（kill -9 以外）を送り、mysqld サーバを停止します。pid が .pid ファイルに格納されています。このファイルは通常、MySQL データベースディレクトリにあります。

   ```
   shell> kill `cat /mysql-data-directory/hostname.pid`
   ```

   これを行うには、Unix root ユーザまたは mysqld が実行しているユーザと同じでなければなりません。
2. --skip-grant-tables オプションを指定して mysqld を再起動します。
3. mysqladmin password コマンドで新しいパスワードを設定します。

   ```
   shell> mysqladmin -u root password 'mynewpassword'
   ```

4. これで、適切に mysqld を停止し再起動するか、以下のように、アクセス権テーブルをロードできるようになります。

   ```
   shell> mysqladmin -h hostname flush-privileges
   ```

5. この後、新しいパスワードを使用して接続できます。

もう一つの方法として、mysql クライアントを使用して新しいパスワードを設定することができます。

1. 上述したように、mysqld を停止して、--skip-grant-tables オプションを指定して再起動します。
2. 以下のように、mysqld サーバに接続します。

   ```
   shell> mysql -u root mysql
   ```

3. mysql クライアントで以下のコマンドを実行します。

```
mysql> UPDATE user SET Password=PASSWORD('mynewpassword')
    ->           WHERE User='root';
mysql> FLUSH PRIVILEGES;
```

4. この後、新しいパスワードを使用して接続できます。

5. これで、適切に mysqld を停止し、再起動できます。

A.4.3. フルディスク時の MySQL の動作

ディスクフル状態が発生した場合、MySQL では以下のことを行います。

- 1 分ごとに 1 回、現在のレコードを書き込むために十分な空き領域があるかどうか確認する。十分な空き領域がある場合は、何も問題が発生しなかったように処理を続行する。
- 6 分ごとにログファイルにエントリを書き込み、ディスクフル状態を警告する。

問題を軽減するために、以下を行うことができます。

- 処理を続行するには、全レコードを挿入できるディスク領域を確保する必要がある。
- スレッドを停止するには、mysqladmin kill をスレッドに送信する必要がある。 スレッドは、次にディスクがチェックされたとき (1分) に停止する。
- 他のスレッドが、ディスクフル状態の原因となったテーブルを待機している可能性がある。複数の ``ロック" スレッドがある場合、ディスクフル状態で待機しているスレッドを 1 つ強制終了すると、他のスレッドの処理を続行できる。

上記の動作の例外は、REPAIR または OPTIMIZE を使用しているときか、インデックスが LOAD DATA INFILE または ALTER TABLE ステートメントの後にバッチで作成されるときです。

上記のコマンドはすべて、大きなテンポラリファイルを使用する場合があるので、資源が残り少なくなったシステムに問題を引き起こす可能性があります。上記の操作を実行中に MySQL がディスクフル状態になると、大きなテンポラリファイルが削除され、テーブルはクラッシュしたものとしてマークされます (旧テーブルが変更されずに残る ALTER TABLE は除きます) 。

A.4.4. MySQL がテンポラリファイルを格納する場所

MySQL は、テンポラリファイルを格納するディレクトリのパス名として、TMPDIR 環境変数の値を使用します。TMPDIR を設定していない場合、MySQL はシステムのデフォルトを使用します。システムのデフォルトは、通常 /tmp または /usr/tmp です。テンポラリファイルディレクトリを含むファイルシステムが非常に小さい場合、mysqld_safe を編集して、領域が十分あるファイルシステムのディレクトリを指すように TMPDIR を設定する必要があります。mysqld に対して --tmpdir オプションを使用しても、テンポラリディレクトリを設定できます。

MySQL はすべてのテンポラリファイルを隠しファイルとして作成します。これによって、mysqld が強制終了されるとテンポラリファイルは確実に削除されます。隠しファイル使用の不利な点は、テンポラリファイルディレクトリのあるファ

イルシステムをいっぱいにするような大きなテンポラリファイルが見えないことです。

ソート時 (ORDER BY または GROUP BY により) 、MySQL は通常、1つまたは2つのテンポラリファイルを使用します。必要な最大ディスク領域は、以下のとおりです。

```
(ソートされたものの長さ + sizeof(データベースポインタ))
* マッチするレコードの数
* 2
```

sizeof(データベースポインタ) は、通常 4 ですが、大きいテーブルに対応するため、将来的に増える可能性があります。

SELECT クエリの中には、MySQL が テンポラリ SQL テーブルを作成するものがあります。これらは隠しファイルではなく、SQL_* という名前になります。

ALTER TABLE では、元のテーブルと同じディレクトリにテンポラリテーブルが作成されます。

MySQL 4.1 以降を使用している場合、コロン : (Windows の場合はセミコロン ;) で区切られたパスのリストを --tmpdir に設定することにより、複数の物理ディスク間で負荷を分散させることができます。この物理ディスクは、ラウンドロビン方式で使用されます。 注意: これらのパスは、同一ディスクの複数のパーティションではなく、異なる物理ディスクである必要があります。

tmpdir を設定してメモリベースのファイルシステムを指定することは可能ですが、MySQL サーバがスレーブの場合はできません。スレーブの場合、コンピュータの再起動用に、いくつかのテンポラリファイルが必要 (テンポラリテーブルまたは LOAD DATA INFILE のレプリケーションため) です。そのため、コンピュータの再起動で消去されるメモリベースの tmpdir は適しません。ディスクベースの tmpdir が必要です。

A.4.5. MySQL ソケットファイル /tmp/mysql.sock の保護または変更方法

すべての人が MySQL 通信ソケット /tmp/mysql.sock を削除できるということが問題であれば、Unix の多くのバージョンにおいて、/tmp ファイルシステムに sticky ビットを設定することにより、そのファイルシステムを保護できるようになっています。root としてログインし、以下を行います。

```
shell> chmod +t /tmp
```

これによって、/tmp ファイルシステムが保護され、ファイルの所有者かスーパーユーザ (root) しかファイルを削除できなくなります。

ls -ld /tmp を実行して、sticky ビットが設定されているかどうかを確認できます。 最後のアクセス権ビットが t であれば、ビットは設定されています。

MySQL がソケットファイルを使用または置く場所を、以下の方法で変更できます。

- グローバルオプション設定ファイルまたはローカルオプション設定ファイルのパスを指定する。 たとえば、/etc/my.cnf に置く場合

  ```
  [client]
  socket=path-for-socket-file

  [mysqld]
  socket=path-for-socket-file
  ```

See 項4.1.2.「my.cnf オプション設定ファイル」。

- コマンドラインで --socket=path-for-socket-file オプションを使用して、mysqld_safe と大部分のクライアントに対してソケットを置く場所を指定する。
- ソケットファイルのパスを、MYSQL_UNIX_PORT 環境変数で指定する。
- configure を --with-unix-socket-path=path-for-socket-file オプションとともに使用して、パスを定義する。 See 項2.3.3.「一般的な configure オプション」。

以下のコマンドで、ソケットの動作をテストできます。

```
shell> mysqladmin --socket=/path/to/socket version
```

A.4.6. タイムゾーンの問題

SELECT NOW() が現地時間ではなく GMT の値を返す場合、現在のタイムゾーンに TZ 環境変数を設定する必要があります。サーバが動作している環境 (たとえば mysqld_safe または mysqld_safe) に対して設定を行ってください。 See 付録 F. 環境変数。

A.5. クエリ関連の問題

A.5.1. 検索時のケース依存

デフォルトでは、MySQL 検索では大文字と小文字は区別されませんが (ケース非依存) 、中には、ケース依存のキャラクタセット (czech) もあります。 つまり、col_name LIKE 'a%' で検索すると、A または a で始まる全カラムの値が検出されます。これをケース依存にする場合は、INSTR(col_name, "A")=1 としてプリフィックスをチェックします。または、カラム値が確実に "A" でなければならない場合は、STRCMP(col_name, "A") = 0 を使用します。

簡単な比較演算 (>=、>、=、<、<=、ソートおよびグループ化) は、各文字の ``ソート値" に基づいています。同じソート値のある文字 (E、e および e など) は、同じ文字として扱われます。

旧 MySQL バージョンでは、LIKE 比較は、各文字の大文字の値 (E == e but E <> e) で実行されました。新しい MySQL バージョンでは、LIKE は他の比較演算と同じように動作します。

カラムを常にケース依存で扱う場合は、BINARY として宣言します。 See 項6.5.3.「CREATE TABLE 構文」。

Big5 と呼ばれるエンコードで中国語データを使用している場合、すべての文字カラムを BINARY にします。Big5 エンコード文字のソート順序は ASCII コードの順序に基づいているため、これが機能します。

A.5.2. DATE カラム使用時の問題

DATE 値の書式は 'YYYY-MM-DD' です。標準 SQL では、他の書式は使用できません。この書式は、UPDATE 式と SELECT ステートメントの WHERE 節で使用してください。たとえば、以下のように使用します。

```
mysql> SELECT * FROM tbl_name WHERE date >= '1997-05-05';
```

日付が数値コンテキストで使用されている場合は、利便性を考えて、MySQL は日付を数値に自動的に変換します (逆の場

合も同様です）。更新時や WHERE 節で ``柔軟な" 文字列書式を使用できるので、日付を TIMESTAMP、DATE または DATETIME カラムと比較することもできます（柔軟な書式とは、どんな句読文字もパート間の区切り記号として使用できる書式のことを言います。たとえば、'1998-08-15' と '1998#08#15' は同一です）。また MySQL は、区切り記号を含まない文字列（たとえば '19980815' ）も、それが日付として理解できる場合は変換することができます。

特別な '0000-00-00' 日付は、'0000-00-00' として格納し取り出すことができます。MyODBC を介して '0000-00-00' 日付を使用している場合、MyODBC バージョン 2.40.12 以上では自動的に NULL に変換されます。

MySQL は上述の変換を実行するため、以下のステートメントを使用します。

```
mysql> INSERT INTO tbl_name (idate) VALUES (19970505);
mysql> INSERT INTO tbl_name (idate) VALUES ('19970505');
mysql> INSERT INTO tbl_name (idate) VALUES ('97-05-05');
mysql> INSERT INTO tbl_name (idate) VALUES ('1997.05.05');
mysql> INSERT INTO tbl_name (idate) VALUES ('1997 05 05');
mysql> INSERT INTO tbl_name (idate) VALUES ('0000-00-00');

mysql> SELECT idate FROM tbl_name WHERE idate >= '1997-05-05';
mysql> SELECT idate FROM tbl_name WHERE idate >= 19970505;
mysql> SELECT MOD(idate,100) FROM tbl_name WHERE idate >= 19970505;
mysql> SELECT idate FROM tbl_name WHERE idate >= '19970505';
```

ただし、以下は動作しません。

```
mysql> SELECT idate FROM tbl_name WHERE STRCMP(idate,'19970505')=0;
```

STRCMP() は文字列関数なので、idate が文字列に変換され、文字列比較が実行されます。'19970505' は日付に変換されず、日付比較も実行されません。

MySQLは、日付が正しいかどうかという非常に限定された確認しか行いません。'1998-2-31' のような不正な日付を格納すると、間違った日付が格納されます。

MySQL は、日付を圧縮して保存するため、結果バッファに適合しない特定の日付は格納できません。日付の受け入れ規則は、以下のとおりです。

- MySQL が任意の日付を格納し取り出せる場合、DATE と DATETIME カラムは間違った日付を受けつける。
- 0 ー 31 のすべての値は、どんな日付にも受け入れられる。これは、3 つの別々のフィールドで年、月、日を保存する Web アプリケーションでは非常に便利である。
- 日または月フィールドをゼロにすることもできる。これは、DATE カラムに生年月日を格納する場合で、日付の一部しか知らないときに便利である。

日付を適切な値に変換できない場合は、0 が DATE フィールドに格納され、0000-00-00 として取り出されます。データベースが行うことは、ユーザが格納した日付と同じものを取り出すことなので（日付が論理的に正しくない場合でも）、これはスピードと利便性両方の問題になります。 我々は、日付を確認するのはサーバではなくアプリケーションの責任であると考えています。

A.5.3. NULL 値の問題

NULL 値というものを SQL 初心者はよく混乱します。SQL 初心者は、多くの場合、NULL が空文字 "" と同じであると考えてしまいます。これは違います。たとえば、以下のステートメントは完全に別のものです。

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ("");
```

どちらのステートメントも、値を phone カラムに挿入しています。しかし、最初のステートメントは NULL 値を挿入し、2 つ目は空文字を挿入しています。最初のステートメントは ``電話番号が不明" であると考えることができ、2番目は``電話を持っていない" と考えることができます。

SQL では、NULL 値は、他の値と比較すると（NULLでも）常に偽になります。NULL を含む式は、演算子と式に含まれている関数のドキュメントに特に断りがなければ、常に NULL 値を生成します。以下の例では、全カラムが NULL を返します。

```
mysql> SELECT NULL,1+NULL,CONCAT('Invisible',NULL);
```

NULL のカラム値を検索したい場合、=NULL テストは使用できません。どんな式でも expr = NULL は偽なので、以下のステートメントはレコードを返しません。

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

NULL 値を検出するには、IS NULL テストを使用します。 以下から、NULL の電話番号と空の電話番号の検索方法がわかります。

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
mysql> SELECT * FROM my_table WHERE phone = "";
```

MySQL バージョン 3.23.2 以降を使用し、かつ MyISAM、InnoDB、BDB テーブル型を使用している場合に限り、NULL 値を持つことができるカラムのインデックスを追加することができます。 以前のバージョンや別のテーブル型では、NOT NULL などのカラムを宣言する必要があります。これは、NULL をインデックス化されたカラムに挿入できないということでもあります。

LOAD DATA INFILE でデータを読み取ると、空のカラムは '' で更新されます。カラムに NULL 値が必要な場合は、テキストファイルで \N を使用してください。状況によっては、リテラル文字 'NULL' も使用されます。 See 項6.4.8.「LOAD DATA INFILE 構文」。

ORDER BY を使用する際、降順でソートするように DESC を指定すると、NULL 値が最初または最後に表示されます。例外: MySQL バージョン 4.0.2 から 4.0.10 では、ソート順序に関わらず NULL 値は 1 番目にソートされます。

GROUP BY を使用すると、すべての NULL 値が同じと見なされます。

COUNT()、MIN()、SUM() などの集約（要約）関数では、NULL 値は無視されます。例外は COUNT(*) です。この関数は、個々のカラム値ではなくレコードをカウントします。 たとえば、以下のステートメントでは 2つのカウントが行われます。 最初は、テーブルにあるレコード数のカウントです。2 番目は age カラムにある非 NULL 値のカウントです。

```
mysql> SELECT COUNT(*), COUNT(age) FROM person;
```

NULL 処理を補うために、IS NULL と IS NOT NULL 演算子と IFNULL() 関数を使用することができます。

カラム型の中には、NULL 値が特別に扱われるものがあります。テーブルの最初のカラム TIMESTAMP に NULL を挿入す

ると、現在の日付と時刻が挿入されます。AUTO_INCREMENT カラムに NULL を挿入すると、順番の次の番号が挿入されます。

A.5.4. alias の問題

エイリアスを使用して、GROUP BY、ORDER BY または HAVING 部分のカラムを参照することができます。エイリアスを使用して、カラムにわかりやすい名前を付けることもできます。

```
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;
```

標準 SQL では、WHERE 節のエイリアスを参照することはできません。これは、WHERE コードが実行される際、カラム値がまだ設定されていない場合があるためです。たとえば、以下のクエリは無効です。

```
SELECT id,COUNT(*) AS cnt FROM table_name WHERE cnt > 0 GROUP BY id;
```

WHERE ステートメントを実行して GROUP BY 部分に追加するレコードを決定し、HAVING を実行して結果セットからどのレコードを使用するか決定します。

A.5.5. 関連テーブルからのレコードの削除

MySQL では、サブクエリをサポートしていません (バージョン 4.1 より前) 。また、DELETE ステートメントでの 2 つ以上のテーブルの使用もサポートしていません (バージョン 4.0 より前) 。そのため、2 つの関連テーブルからレコードを削除するには、以下の方法を使用してください。

1. メインテーブルで WHERE 条件に基づいて、レコードを SELECT する。
2. 同じ条件に基づいて、メインテーブルのレコードを DELETE する。
3. DELETE FROM related_table WHERE related_column IN (selected_rows)。

related_column のクエリの合計文字数が 1,048,576 (max_allowed_packetのデフォルト値) を超える場合、分割して、複数回 DELETE ステートメントを実行してください。related_column がインデックスの場合は、100 ー 1000 の related_column ID を削除するだけなので、通常 DELETE が非常に速くなります。related_column がインデックスでない場合、速度は IN 節の引数の数に依存しません。

A.5.6. 不整合レコードの問題解決

多くのテーブルが含まれる複雑なクエリを使用していて、それがレコードを返さない場合、以下の手順でクエリの問題を見つける必要があります。

1. EXPLAIN でクエリをテストし、明らかに問題と思われる箇所があるか確認する。 See 項5.2.1.「EXPLAIN 構文 (SELECT に関する情報の取得) 」。
2. WHERE 節で使用されるフィールドだけを選択する。
3. レコードが返るまで、クエリから 1 つ 1 つテーブルを削除する。 テーブルが大きい場合に有効なのは、クエリで LIMIT 10 を使用する方法である。

4. クエリから最後に削除されたテーブルに対してレコードが一致したカラムに、 SELECT を実行する。

5. FLOAT または DOUBLE カラムを小数と比較している場合、'=' は使用できない。浮動小数点の値は正確な値ではない。この問題は大部分のコンピュータ言語で共通。 たいていの場合、FLOAT を DOUBLE に変更すると問題は解決する。 See 項A.5.7.「浮動小数点比較の問題」。

6. それでも問題が解決できない場合は、mysql test < query.sql で実行できる小さなテストを作成して、問題を表示する。 mysqldump --quick database tables > query.sql で、テストファイルを作成できる。エディタでファイルを開き、数が多ければいくつかの挿入行を削除する。そして、ファイルの最後に SELECT ステートメントを追加する。

   まだ問題が残っているかどうか、以下の方法でテストしてください。

```
shell> mysqladmin create test2
shell> mysql test2 < query.sql
```

   mysqlbug を使用して、テストファイルを汎用 MySQL メーリングリストに投稿してください。 See 項1.7.1.1.「MySQL メーリングリスト」。

A.5.7. 浮動小数点比較の問題

浮動小数点数は、コンピュータアーキテクチャ内部では正確な値として格納されないため、混乱を引き起こす場合があります。通常画面に表示される値は、正確な値ではありません。

フィールドタイプ FLOAT、DOUBLE、および DECIMAL が該当します。

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

結果は正確です。最初の 5 つのレコードは比較テストにパスしないように見えますがパスできます。コンピュータアーキテクチャによりますが、数の差異は、小数点第 10 位くらいで現れるためです。

結果は浮動小数点数なので、ROUND( ) (または同様の関数) を使用して問題を解決することはできません。たとえば、以下のとおりです。

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

カラム 'a' 内の数は以下のようになります。

```
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 | -51.3999999999999986 |  0.00 |
+------+----------------------+-------+
```

コンピュータアーキテクチャによって、同じ結果が表示されたりされなかったりします。 CPU ごとに浮動小数点数の評価方法が異なります。たとえばマシンの中には、両方の引数を 1 で掛け合わすと '正確な' 結果を得られるものがあります。たとえば以下のようになります。

警告: あなたのアプリケーションでは、この方法を決して信用しないでください。これは間違った方法の例です。

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

上記の例が動作するように見える理由は、テストが実行されたマシンで、CPU 浮動小数点演算が複数の数字を偶然同じ数字に四捨五入してしまうことにあります。しかし、CPU がそうしなければならないという規則はないため、この例は信用できません。

浮動小数点数比較を行う正しい方法は、まず数の許容範囲を決定し、続いて許容範囲の数に対して比較を行うことです。たとえば、浮動小数点数は同一と見なすということにすれば、それらが 10000 分の 1 (0.0001) の精度で同じである場合、比較が以下のように行われます。

```
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)
```

逆に、数が同じになるレコードを取得したい場合は、テストは次のようになります。

```
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. オプティマイザ関連の問題

MySQL はオプティマイザを使用して、クエリを解決する最も優れた方法を見つけ出します。多くの場合、MySQL は最も可能性のあるクエリを見積もることができますが、場合によってはデータに関する十分な情報が手元にないので、経験に基づく '推測' を行う必要があります。

ここでは、MySQL が正しく理解しない場合を扱います。

MySQL が '正しく' 最適化するのに役立つツールは、以下のとおりです。

- EXPLAIN。 See 項5.2.1.「EXPLAIN 構文 (SELECT に関する情報の取得) 」。
- ANALYZE TABLE。 See 項4.6.2.「ANALYZE TABLE 構文」。
- USE INDEX, FORCE INDEX および IGNORE INDEX。 See 項6.4.1.「SELECT 構文」。
- グローバルおよびテーブルレベルの STRAIGHT JOIN。 See 項6.4.1.「SELECT 構文」。
- スレッド固有の変数の設定。 See 項4.6.8.4.「SHOW VARIABLES」。

A.6.1. テーブルスキャンを回避する方法

MySQL がテーブルスキャンを使用してクエリを解決した場合、EXPLAIN では type カラム内に ALL が表示されます。これは、通常以下の場合に起こります。

- テーブルが小さく、キー走査よりもテーブルスキャンを行う方が速い場合。 これは、レコードの数が 10 未満で、レコードの長さが短いテーブルではよくあるケースである。
- ON または WHERE 節に、インデックスを張ったカラムを利用する条件がない場合。
- インデックスを張ったカラムと変数を比較中に、MySQL が (インデックスツリーに基づいて) 、変数がテーブルの大部分の範囲に及んでいるためテーブルスキャンの方が速いと予測した場合。 See 項5.2.4.「MySQL による WHERE 節

の最適化」。

- 別のカラムを介して、下位の基数 (= 多くの一致レコード) でキーを使用している場合。この場合、MySQL は、そのキーを使用して多くのキールックアップが行われると見なし、テーブルスキャンの方が早いと見なす。

大きなテーブルの '不正な' テーブルスキャンを回避するには、以下のことを行います。

- スキャンテーブルに対して ANALYZE TABLE を使用し、キーの分布を更新する。 See 項4.6.2.「ANALYZE TABLE 構文」。
- スキャンテーブルに対して FORCE INDEX を使用し、テーブルのスキャンは、特定のインデックスの使用と比較すると非常に処理の負荷が高いことを、MySQL に通知する。 See 項6.4.1.「SELECT 構文」。

```
SELECT * FROM t1,t2 force index(index_for_column) WHERE t1.column=t2.column;
```

- --max-seeks-for-key=1000 を指定して mysqld を起動するか SET MAX_SEEKS_FOR_KEY=1000 を行って、オプティマイザに、キースキャンで 1000 を越すキー検索が行われないことを告げる。

A.7. テーブル定義関連の問題

A.7.1. ALTER TABLE の問題

ALTER TABLE によって、テーブルが現在のキャラクタセットに変更されます。 ALTER TABLE 中に duplicate key error が発生する場合、新しいキャラクタセットが 2 つのキーを同じ値にマップしているか、テーブルが壊れています。テーブルが壊れている場合は、そのテーブルで REPAIR TABLE を実行してください。

以下のようなエラーで ALTER TABLE が強制終了される場合

```
Error on rename of './database/name.frm' to './database/B-a.frm' (Errcode: 17)
```

原因として、MySQL が前回の ALTER TABLE でクラッシュしており、使用されていない A-something または B-something という名前の古いテーブルがあることが考えられます。この場合、MySQL データディレクトリに移り、A- または B- で始まる名前のファイルをすべて削除してください (削除しないで別の場所に移動することもできます)。

ALTER TABLE は、以下のように動作します。

- 変更要求のあった A-xxx という名前の新しいテ-ブルを作成する。
- 旧テーブルのすべてのレコードが A-xxx にコピーされる。
- 旧テーブル名が B-xxx に変更される。
- A-xxx が旧テーブル名に変更される。
- B-xxx が削除される。

名前変更操作に何か問題が発生した場合は、MySQL は変更を取り消します。致命的な問題が発生した場合 (もちろん起こ

ってはならないことですが）、MySQL は旧テーブルを B-xxx のままにしておく可能性がありますが、システムレベルでの簡単な名前変更でデータは元に戻ります。

A.7.2. テーブルのカラム順序を変更する方法

SQL の主な目的は、データストレージから要求した情報を抽出することです。必ず、データを取り出す順序を指定する必要があります。たとえば、以下のようになります。

```
SELECT col_name1, col_name2, col_name3 FROM tbl_name;
```

上記は、col_name1、col_name2、col_name3 の順序でカラムを返します。

```
SELECT col_name1, col_name3, col_name2 FROM tbl_name;
```

上記の場合、col_name1、col_name3、col_name2 の順序でカラムを返します。

カラム順序は、以下のように変更できます。

1. 正しい順序のカラムで新しいテーブルを作成する。
2. INSERT INTO new_table SELECT fields-in-new_table-order FROM old_table を実行する。
3. old_table を廃棄するか、名前変更する。
4. ALTER TABLE new_table RENAME old_table。

カラムを追加、移動、削除する場合、カラムが返される順序と位置が同じにならないので、アプリケーションで、SELECT * を使用してその位置に依存するカラムを絶対に取り出さないでください。データベース構造に簡単な変更を加えるだけでも、アプリケーションエラーが発生します。 もちろん SELECT * は、クエリのテストに最適です。

A.7.3. テンポラリテーブルの問題

以下、テンポラリテーブルの制限を示します。

- テンポラリテーブルの型は、HEAP、ISAM、MyISAM、 MERGE、InnoDB のみ。
- テンポラリテーブルは同じクエリで 2 回以上使用できない。 たとえば、以下は動作しない。

  ```
  mysql> SELECT * FROM temporary_table, temporary_table AS t2;
  ```

- TEMPORARY テーブルで RENAME は使用できない。 ALTER TABLE org_name RENAME new_name は使用できる。

付録 B. 提供されたプログラム

MySQL の多くのユーザによって、非常に便利なサポートツールとアドオンが提供されました。

MySQL Web サイト (またはミラー) から利用できるソフトウェアの一覧は以下のとおりです。

MySQL 関連ソフトウェアのオンライン一覧には、http://www.mysql.com/portal/software/ からアクセスできます。 コミュニティ機能により、入力も行えるようになっています。

Perl DBI/DBD インタフェースで MySQL サポートを行う場合は、Data-Dumper、DBI、および DBD-mysql ファイルを取得し、それらをインストールする必要があります。 See 項2.7.「Perl インストールについてのコメント」。

注意: ここで紹介するプログラムは、自由にダウンロードして使用することができます。 プログラムの著作権は、それぞれの所有者に属しています。 ライセンスと規約の詳細については、各製品のドキュメントを参照してください。 MySQL AB は、本章における情報の正当性、またはここに示されているプログラムが正確に動作するかどうかについては、一切の責任を負いません。

B.1. API

- Perl モジュール
  - http://www.mysql.com/Downloads/Contrib/KAMXbase1.2.tar.gz .dbf ファイルと MySQL テーブル間の変換を行う。 Pratap Pereira <pereira@ee.eng.ohio-state.edu> が作成し、Kevin A. McGrail <kmcgrail@digital1.peregrinehw.com> が拡張した Perl モジュール。 このコンバータは、MEMO フィールドを処理することができる。
  - http://www.mysql.com/Downloads/Contrib/HandySQL-1.1.tar.gz HandySQL は MySQL アクセスモジュールである。 Perl に埋め込まれた C インタフェースを提供し、標準 DBI より約 20 % 速い。
- OLEDB
  - http://www.mysql.com/Downloads/Win32/MyOLEDB3.exe SWSoft の MyOLEDB 3.0 インストールパッケージ。
  - http://www.mysql.com/Downloads/Win32/mysql-oledb-3.0.0.zip MyOLEDB 3.0 用ソース。
  - http://www.mysql.com/Downloads/Win32/MySamples.zip MyOLEDB 用サンプルとドキュメント。
  - http://www.mysql.com/Downloads/Win32/MyOLEDB.chm MyOLEDB 用ヘルプファイル。
  - http://www.mysql.com/Downloads/Win32/libmyodbc.zip MyOLEDB のビルドに使用する静的 MyODBC ライブラリ。 MyODBC コードに準拠している。
- C++
  - http://www.mysql.com/Downloads/Contrib/mysql-c++-0.02.tar.gz MySQL C++ ラッパライブラリ。Roland Haenel、<rh@ginster.net> が作成。
  - http://www.mysql.com/Downloads/Contrib/MyDAO.tar.gz MySQL C++ API。Satish <spitfire@pn3.vsnl.net.in> が作成。Roland Haenel の C++ API と Ed Carp の MyC ライブラリからアイディアを得る。

  - http://www.mysql.com/products/mysql++/ MySQL C++ API (単なるラッパライブラリではない) 。最初の作成者は <kevina@clark.net>。現在 MySQL AB で Sinisa によって保守されている。
  - http://nelsonjr.homepage.com/NJrAPI/ MySQL をサポートする C++ データベース非依存ライブラリ。
- Delphi
  - http://www.mysql.com/Downloads/Contrib/DelphiMySQL2.zip libmysql.dll に対する Delphi インタフェース。 <bsilva@umesd.k12.or.us> が作成。
  - http://www.mysql.com/Downloads/Contrib/Udmysql.pas Delphi で使用するための libmysql.dll 用ラッパ。Reiner Sombrowsky が作成。
  - http://www.fichtner.net/delphi/mysql.delphi.phtml MySQL に対する Delphi インタフェース (ソースコード付き) 。Matthias Fichtner が作成。
  - http://www.productivity.org/projects/tmysql/ TmySQL。MySQL を Delphi で使用するためのライブラリ
  - https://sourceforge.net/projects/zeoslib/ Zeos ライブラリは、MySQL、PostgreSQL、Interbase、MS SQL、Oracle および DB2 用の Delphi ネイティブデータベースとデータベースコンポーネントのセットである。Database Explorer および Database Designer などの開発ツールも含まれている。
  - http://www.mysql.com/Downloads/Contrib/JdmMysqlDriver-0.1.0.tar.gz MySQL 用 VisualWorks 3.0 Smalltalk ドライバ。<joshmiller@earthlink.net> が作成。
  - http://www.mysql.com/Downloads/Contrib/Db.py キャッシュ機能のあるPython モジュール。 <gandalf@rosmail.com> が作成。
  - http://www.mysql.com/Downloads/Contrib/MySQLmodule-1.4.tar.gz MySQL 用 Python インタフェース。Joseph Skinner <joe@earthlight.co.nz> が作成。Joerg Senekowitsch <senekow@ibm.net> が修正。
  - http://www.mysql.com/Downloads/Contrib/mysqltcl-1.53.tar.gz MySQL 用 Tcl インタフェース。msqltcl-1.50.tar.gz に準拠している。 バージョン 2.0 と詳細については、http://www.xdobry.de/mysqltcl/ を参照。
  - http://www.mysql.com/Downloads/Contrib/MyC-0.1.tar.gz Visual Basic のような API。Ed Carp が作成。
  - http://www.mysql.com/Downloads/Contrib/Vdb-dflts-2.1.tar.gz これは、SQL データベースエンジンに対する汎用インタフェースを提供することを目的としたライブラリユーティリティセットの新バージョン。これによって、アプリケーションが 3 層アプリケーションになる。利点は、アプリケーションを変更せずに新しいバックエンド用ファイルを 1 つ実装することで、他のデータベース間のスイッチや他のデータベースへの移動が容易に行えることである。<damian@cablenet.net> が作成。
  - http://www.mysql.com/Downloads/Contrib/DbFramework-1.10.tar.gz DbFramework は、MySQL データベースを操作するクラスの集まりである。クラスは、CDIF データモデルのサブジェクトエリアを大まかに基準にしている。Paul Sharpe <paul@miraclefish.com> が作成。
  - http://www.mysql.com/Downloads/Contrib/pike-mysql-1.4.tar.gz pike 用 MySQL モジュール。Roxen Web サーバで使用。
  - http://www.mysql.com/Downloads/Contrib/squile.tar.gz guile がSQL データベースと対話できるようにする guile 用モジュール。Hal Roberts が作成。

- http://www.mysql.com/Downloads/Contrib/stk-mysql.tar.gz Stk 用インタフェース。Stk は、下部に Tcl 以外のスキームを持つ Tk ウィジェットである。 Terry Jones が作成。
- http://www.mysql.com/Downloads/Contrib/eiffel-wrapper-1.0.tar.gz Michael Ravits が作成した Eiffel ラッパ。
- http://www.mysql.com/Downloads/Contrib/SQLmy0.06.tgz MySQL 用 FlagShip Replaceable Database Driver (RDD) 。Alejandro Fernandez Herrero が作成。 Flagship RDD ホームページの URL は http://www.fship.com/rdds.html。
- http://www.mysql.com/Downloads/Contrib/mydsn-1.0.zip mydsn.dll のバイナリおよびソース。Mydsn を使用して、Coldfusion アプリケーションで MyODBC ドライバの DNS レジストリファイルを構築、削除する。Miguel Angel Solórzano が作成。
- http://www.mysql.com/Downloads/Contrib/MySQL-ADA95_API.zip MySQL API に対する ADA95 インタフェース。Francois Fabien が作成。
- http://www.mysql.com/Downloads/Contrib/MyTool-DLL_for_VB_and_MySQL.zip Visual Basic 用 MySQL C API 付き DLL。 Ken Menzel <kenm@icarz.com> が作成。
- http://www.mysql.com/Downloads/Contrib/MYSQLX.EXE 処理が遅い ODBC メソッドを省略して、IIS/ASP、VB、VC++ から直接 MySQL サーバにアクセスするための、MySQL ActiveX オブジェクト。全 MySQL フィールドタイプを完全サポート、完全更新可能でマルチスレッド化されている (バージョン 2001.1.1) 。SciBit http://www.scibit.com/ が作成。
- http://www.fastflow.it/mylua/ MyLUA ホームページ。LUA 言語を使用して、実行時にロードできる MySQL PROCEDURE を作成する方法が記載されている。
  - 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 LUA 4.0 を使用するための MySQL 3.23.32 用パッチ。Cristian Giussani が作成。
- http://www.mysql.com/Downloads/Contrib/patched_myodbc.zip (Omniform 4.0 をサポートするための) MyODBC ドライバのパッチ。 Thomas Thaele <tthaele@papenmeier.de> が作成。

B.2. コンバータ

- http://www.mysql.com/Downloads/Contrib/mssql2mysql.txt MS-SQL から MySQL へのコンバータ。Michael Kofler が作成。 mssql2mysql ホームページの URL は http://www.kofler.cc/mysql/mssql2mysql.html。
- http://www.mysql.com/Downloads/Contrib/dbf2mysql-1.14.tar.gz .dbf ファイルと MySQL テーブル間の変換を行う。Maarten Boekhold (<boekhold@cindy.et.tudelft.nl>)、William Volkman、Michael Widenius が作成。このコンバータには、MEMOフィールドの基本的な読み取り専用サポートも含まれている。
- http://www.mysql.com/Downloads/Contrib/dbf2mysql-1.13.tgz .dbf ファイルと MySQL テーブル間の変換を行う。Maarten Boekhold、<boekhold@cindy.et.tudelft.nl>、Michael Widenius が作成。 このコンバータは、MEMO フィールドを処理できない。
- http://www.mysql.com/Downloads/Contrib/dbf2mysql.zip Windows 上で、FoxPro .dbf ファイルと MySQL テーブル間の

変換を行う。 Alexander Eltsyn、<ae@nica.ru>、または <ae@usa.net> が作成。

- http://www.mysql.com/Downloads/Contrib/dbf2sql.zip foxpro テーブルから MySQL テーブルへのデータ転送に役立つ簡潔かつ簡単な prg。Danko Josic が作成。
- http://www.mysql.com/Downloads/Contrib/dump2h-1.20.gz mysqldump 出力から C ヘッダファイルへの変換を行う。Harry Brueckner <brueckner@mail.respublica.de> が作成。.
- http://www.mysql.com/Downloads/Contrib/exportsql.txt access_to_mysql.txt に類似したスクリプト (このスクリプトが完全に調整可能であることを除いて) 。優れた型変換 (TIMESTAMP フィールドの検出を含む) 機能を持つ。このスクリプトは、変換中に警告や提案を行い、テキストやバイナリデータ内の全特殊文字を引用符で囲む。また、mSQL バージョン 1 およびバージョン 2 に変換し、無償で使用できる。最新のバージョンについては、http://www.cynergi.net/exportsql/ を参照。Pedro Freire、<support@cynergi.net> が作成。注意: Access 2.0 では動作しない。
- http://www.mysql.com/Downloads/Contrib/access_to_mysql.txt エクスポートするテーブルがあるデータベースの Access モジュールに、この関数を貼り付ける。exportsql も参照のこと。Brian Andrews が作成。 注意: Access 2 では動作しない。
- http://www.mysql.com/Downloads/Contrib/importsql.txt exportsql.txt と正反対のことを行うスクリプト。つまり、ODBC を介して、データを MySQL から Access データベースにインポーする。exportsql と組み合わせると非常に便利である。これによって、 DB 設計と管理すべてに Access を使用し、実際の MySQL サーバと同期化できるからである。無償提供。アップデートについては http://www.netdive.com/freebies/importsql/ を参照。 NetDIVE の Laurent Bossavit が作成。 注意: Access 2.0 では動作しない。
- http://www.mysql.com/Downloads/Contrib/mdb2sql.bas Moshe Gurvich が作成した Access97 から MySQL へのコンバータ。
- http://www.mysql.com/Downloads/Contrib/msql2mysqlWrapper-1.0.tgz mSQL から MySQL への C ラッパ。<alfred@sb.net> が作成。
- http://www.mysql.com/Downloads/Contrib/sqlconv.pl 1つの MySQL テーブルから別のテーブルに、フィールドをまとめてコピーするのに使用する簡単なスクリプト。基本的に、mysqldump を実行し、sqlconv.pl にパイプすることができる。スクリプトは mysqldump 出力によってフィールドを解析、再配置するので、そのフィールドを新しいテーブルに挿入できるようになる。たとえば、作業中の別のサイトに新しいテーブルを作成する際、そのテーブルに少ししか差異がない場合 (つまり、フィールドの順序が異なっているなど) に使用する。 Steve Shreeve が作成。
- http://www.mysql.com/Downloads/Contrib/oracledump Oracle データベースを MySQL に変換する Perl プログラム。mysqldupm と同じ出力形式である。Johan Andersson が作成。
- http://www.mysql.com/Downloads/Contrib/excel2mysql Excel スプレッドシートを MySQL データベースにインポートする Perl プログラム。 Stephen Hurd <shurd@sk.sympatico.ca> が作成。
- http://www.mysql.com/Downloads/Contrib/T2S_100.ZIP。 テキストファイルを MySQL データベースに変換する Windows プログラム。Asaf Azulay が作成。

B.3. ユーティリティ

- http://worldcommunity.com/opensource/utilities/mysql_backup.html MySQL Backup は、MySQL 用のバックアップスクリプトである。Peter F. Brown が作成。
- http://www.mysql.com/Downloads/Contrib/mysql_watchdog.pl 可能なルックアップに対して MySQL デーモンをモニタする。Yermo Lamers が作成。 <yml@yml.com>。
- http://www.mysql.com/Downloads/Contrib/mysql_structure_dumper.tar.gz
- http://www.mysql.com/Downloads/Contrib/mysql_structure_dumper.tgz データベースの全テーブルの構造を出力。Thomas Wana が作成。
- http://www.mysql.com/Downloads/Contrib/mysqlsync。 中央マスタコピーと同期を取りながら MySQL データベースのリモートコピーを続ける Perl スクリプト。Mark Jeftovicが作成。 <markjr@easydns.com>。
- http://www.mysql.com/Downloads/Contrib/MySQLTutor-0.2.tar.gz。 MySQLTutor。初心者向け MySQL チュートリアル。
- http://www.mysql.com/Downloads/Contrib/MySQLDB.zip
- http://www.mysql.com/Downloads/Contrib/MySQLDB-readme.html。 Alok Singh が作成した MySQL 用 COM ライブラリ。
- http://www.mysql.com/Downloads/Contrib/mysql_replicate.pl レプリケーションを処理する Perl プログラム。 <elble@icculus.nsg.nwu.edu> が作成。
- http://www.mysql.com/Downloads/Contrib/dbcheck テーブルで isamchk を実行する前に、そのテーブルのバックアップを取る Perl スクリプト。 Elizabeth が作成。
- http://www.mysql.com/Downloads/Contrib/mybackup。
- http://www.mswanson.com/mybackup (mybackup のホームページ)。 全データベースをバックアップする mysqldump 用ラッパ。Marc Swanson が作成。
- http://www.mysql.com/Downloads/Contrib/mdu.pl.gz MySQL データベースの記憶域使用率を出力。

付録 C. 協力者

この付録では、MySQL を今日の形にするためにご協力頂いた開発者、貢献者およびサポータを記載しています。

C.1. MySQL AB の開発者

MySQL データベースソフトウェアに取り組むために MySQL AB に採用されている、または採用された開発者を、我々と作業を開始した大まかな順序で記載しています。また、各開発者の後に、その開発者が担当している仕事および開発者の業績を記載しました。すべての開発者がサポート業務に関わっています。

- Michael (Monty) Widenius
  - MySQL サーバ（mysqld）の主要開発者および主要作成者。
  - 文字列ライブラリの新規関数。
  - mysys ライブラリの大部分。
  - ISAM および MyISAM ライブラリ（インデックス圧縮と複数のレコード書式を持つ B-tree インデックスファイルハンドラ）。
  - HEAP ライブラリ。優れたフルダイナミックハッシュを持つメモリテーブルシステム。1981 年から使用され、1984 年頃に公開された。
  - replace プログラム（優れものである）。
  - Windows 95 用の ODBC ドライバである MyODBC。
  - MIT-pthread が MySQL サーバでも動作するようにバグを修正。さらに多くのユーティリティを持つ curses ベースのアプリケーションツールである Unireg を開発。
  - msqlperl、DBD/DBI、DB2mysql などの mSQL ツールの移植。
  - crash-me の大部分と MySQL ベンチマークの基礎。
- David Axmark
  - リファレンスマニュアル（texi2htmlへの拡張を含む）の最初の主要著者。
  - マニュアルからの自動 Web サイト更新。
  - 初期の Autoconf、Automake および Libtool サポート。
  - ライセンス。
  - 全テキストファイル部分（現在では、README だけが残っている。残りはマニュアルに取り込まれた）。
  - 多くの新機能のテスト。
  - 社内フリーソフトの法律顧問。
  - メーリングリストの保守管理者（完全に管理する時間はなかった）。

  - オリジナルの移植コード（10 年以上昔のもの）の開発。現在、mysys の一部だけが残っている。
  - Monty が新しい機能を動作させようとするときに、真夜中に電話をする相手。
  - チーフ "Open Sourcerer"（MySQL コミュニティ関連）。
- Jani Tolonen
  - mysqlimport
  - 多くのコマンドラインクライアントの拡張。
  - PROCEDURE ANALYSE()
- Sinisa Milivojevic
  - クライアント/サーバプロトコルの（zlib による）圧縮。
  - 字句アナライザフェーズの完全ハッシュ。
  - 複数行 INSERT。
  - mysqldump -e オプション。
  - LOAD DATA LOCAL INFILE。
  - SQL_CALC_FOUND_ROWS SELECT オプション。
  - --max-user-connections=... オプション。
  - net_read および net_write_timeout。
  - GRANT/REVOKE および SHOW GRANTS FOR。
  - 4.0 用の新規クライアント/サーバプロトコル。
  - 4.0の UNION。
  - 複数テーブル DELETE/UPDATE。
  - 4.1 の派生テーブル。
  - ユーザリソース管理。
  - MySQL++ C++ API と MySQLGUI クライアントの初期開発者。
- Tonu Samuel（過去の開発者）
  - VIO インタフェース（暗号化クライアント/サーバプロトコルの基礎）。
  - MySQL ファイルシステム（ファイルおよびディレクトリとして MySQL データベースを使用する方法）。
  - CASE 式。
  - MD5() および COALESCE() 関数。

    - MyISAM テーブルの RAID サポート。
- Sasha Pachev
    - レプリケーションを最初に導入 (バージョン 4.0 まで) 。
    - SHOW CREATE TABLE。
    - mysql-bench。
- Matt Wagner
    - MySQL テストスィート。
    - Web マスタ (2002 年まで) 。
    - 開発コーディネータ。
- Miguel Solorzano
    - Win32 の開発とリリースのビルド。
    - Windows NTサーバコード。
    - WinMySQLAdmin
- Timothy Smith (過去の開発者)
    - 動的キャラクタセットのサポート。
    - ビルドシステムの RPM およびその他の部分を構築。
    - libmysqld、埋め込みサーバの初期の開発者。
- Sergei Golubchik
    - 全文検索。
    - MERGE ライブラリにキーを追加。
- Jeremy Cole
    - この優れたマニュアルの校正と編集。
    - ALTER TABLE ...ORDER BY ...。
    - UPDATE ...ORDER BY ...。
    - DELETE ...ORDER BY ...。
- Indrek Siitan
    - Web インタフェースの設計/プログラミング。
    - ニュースレター作成者管理システム。

- Jorge del Conde
  - MySQLCC（MySQL コントロールセンタ）。
  - Win32 開発。
  - Web サイトポータルを最初に導入。
- Venu Anuganti
  - Connector/ODBC（MyODBC）3.51。
  - 4.1 の新規クライアント/サーバプロトコル（作成済ステートメント用）。
- Arjen Lentz
  - MySQL リファレンスマニュアルの更新者。
  - O'Reilly の印刷版マニュアルの準備。
- Alexander (Bar) Barkov、Alexey (Holyfoot) Botchkov および Ramil Kalimullin
  - 4.1 用の空間データ（GIS）と R-Tree の導入。
  - 4.1 用のユニコードとキャラクタセット、および 4.1 用のドキュメント。
- Oleksandr (Sanja) Byelkin
  - 4.0 のクエリキャッシュ
  - サブクエリの導入（4.1）。
- Aleksey (Walrus) Kishkin および Alexey (Ranger) Stroganov
  - ベンチマークの設計と分析。
  - MySQL テストスィートの保守。
- Zak Greant
  - オープンソースの支持者、MySQLコミュニティの関係者。
- Carsten Pedersen
  - MySQL 証明書プログラム。
- Lenz Grimmer
  - 製品（ビルドおよびリリース）エンジニアリング。
- Peter Zaitsev
  - SHA1()、AES_ENCRYPT() および AES_DECRYPT() 関数。
  - 各種機能のデバッグとクリーンアップ。

- Alexander (Salle) Keremidarski
  - サポート。
  - デバッグ。
- Per-Erik Martin
  - ストアドプロシージャ（5.0）とトリガの主要開発者。
- Jim Winstead
  - 主要 Web 開発者。
- Mark Matthews
  - Connector/J ドライバ（Java）。
- Peter Gulutzan

  SQL-99、SQL:2003 標準準拠
  - 既存の MySQL コード/アルゴリズムの文書化。
  - キャラクタセットの文書化。
- Guilhem Bichot
  - MySQL バージョン 4.0 からのレプリケーション。
  - DECIMAL 指数の固定処理。
  - mysql_tableinfo の作成者。
- Antony T. Curtis
  - OS/2 への MySQL データベースソフトウェアの移植。

C.2. MySQL への貢献者

MySQL AB は、MySQL サーバと MySQL マニュアルの全著作権を保有していますが、MySQL ディストリビューションに貢献していただいた方々に感謝申し上げます。 貢献者は以下のとおりです。順不同です。

- Gianmassimo Vigazzola <qwerg@mbox.vol.it> または <qwerg@tin.it>

  Win32/NT への最初の移植。
- Per Eric Olsson

  多少の建設的な批判と動的レコード形式に対する正確なテスト。
- Irena Pancirov <irena@mail.yacc.it>

Borland コンパイラを使用した Win32 の移植。mysqlshutdown.exe および mysqlwatch.exe。

- David J. Hughes

  シェアウェアの SQL データベースの作成に尽力。MySQL AB の前身である TcX で、mSQL を始めたが、目的を満足させるものではないことがわかったため、アプリケーションインタフェースビルダ Unireg への SQL インタフェースの作成に転換。mysqladmin および mysql クライアントは、mSQL 対応物に非常に影響を受けたプログラムである。 我々は、MySQL 構文が mSQL のスーパーセットになるよう非常に努力した。この API のアイディアは mSQL に基づいたもので、無料の mSQL プログラムを MySQL API に簡単に移植できるようにした。 MySQL ソフトウェアには、mSQL のコードは含まれていない。 ディストリビューションの 2 つのファイル (client/insert_test.c および client/select_test.c) は、mSQL ディストリビューションの対応ファイル (著作権未取得) に準拠しているが、mSQL から MySQL サーバにコードを変換するために必要な変更を示す例として修正されている。 (mSQL の著作権は David J. Hughes にある) 。

- Patrick Lynch

  http://www.mysql.com/ の取得を支援。

- Fred Lindberg

  MySQL メーリングリストを処理する qmail の設定と、MySQL メーリングリスト管理に対する多大な支援。

- Igor Romanenko <igor@frog.kiev.ua> mysqldump (以前は msqldump であったが、Monty によって移植、拡張された) 。 , Yuri Dario

  MySQL OS/2 の移植の継続と拡張。

- Tim Bunce

  mysqlhotcopy の作成者。

- Zarko Mocnik <zarko.mocnik@dem.si>

  スロベニア語のソート。

- "TAMITO" <tommy@tmtm.org>

  _MB キャラクタセットのマクロと、ujis および sjis のキャラクタセット。

- Joshua Chamas <joshua@chamas.com>

  同時挿入の基準、拡張日付構文、NT でのデバッグ、MySQL メーリングリストの回答。

- Yves Carlier <Yves.Carlier@rug.ac.be>

  ユーザのアクセス権を表示するプログラム mysqlaccess。

- Rhys Jones <rhys@wales.com> (および GWE Technologies Limited)

  初期 JDBC ドライバの 1 つ

- Dr Xiaokun Kelvin ZHU <X.Zhu@brad.ac.uk>

初期 JDBC ドライバの 1 つと、他の MySQL 関連 Java ツールのさらなる開発。

- James Cooper <pixel@organic.com>

  自分自身のサイトで検索可能なメーリングリストアーカイブのセットアップ。

- Rick Mehalick <Rick_Mehalick@i-o.com>

  MySQL サーバのグラフィカルな X クライアントである xmysql。

- Doug Sisk <sisk@wix.com>

  Red Hat Linux 対応 MySQL の RPM パッケージの提供。

- Diemand Alexander V. <axeld@vial.ethz.ch>

  Red Hat Linux-Alpha 対応 MySQL の RPM パッケージの提供。

- Antoni Pamies Olive <toni@readysoft.es>

  Intel および SPARC 対応の多くの MySQL クライアントの RPM バージョンの提供。

- Jay Bloodworth <jay@pathways.sde.state.sc.us>

  MySQL バージョン 3.21 対応 RPM バージョンの提供。

- David Sacerdote <davids@secnet.com>

  DNS ホスト名のセキュリティチェックの方針。

- Wei-Jou Chen <jou@nematic.ieo.nctu.edu.tw>

  中国語 (BIG5) キャラクタのサポート。

- Wei He <hewei@mail.ied.ac.cn>

  中国語 (GBK) キャラクタセット対応の多くの機能。

- Jan Pazdziora <adelton@fi.muni.cz>

  チェコ語のソート順序。

- Zeev Suraski <bourbon@netvision.net.il>

  FROM_UNIXTIME() の時間書式、ENCRYPT() 関数、bison のアドバイザ。 メーリングリストのアクティブメンバ。

- Luuk de Boer <luuk@wxs.nl>

  ベンチマークスィートを DBI/DBD に移植 (および拡張) 。crash-me とベンチマークの実行の大きな助けとなった。いくつかの新規日付関数。mysql_setpermissions スクリプト。

- Alexis Mikhailov <root@medinf.chuvashia.su>

  ユーザ定義可能関数 (UDF) 、CREATE FUNCTION および DROP FUNCTION。

- Andreas F. Bobak <bobak@relog.ch>

  UDF 関数の AGGREGATE 拡張。

- Ross Wakelin <R.Wakelin@march.co.uk>

  MySQL-Win32 用 InstallShield の設定の支援。

- Jethro Wright III <jetman@li.net>

  libmysql.dll ライブラリ。

- James Pereria <jpereira@iafrica.com>

  MySQL サーバを管理する Win32 GUI ツール Mysqlmanager。

- Curt Sampson <cjs@portal.ca>

  MIT-pthread の NetBSD/Alpha および NetBSD 1.3/i386 への移植。

- Martin Ramsch <m.ramsch@computer.org>

  MySQL チュートリアル内のサンプル。

- Steve Harvey

  mysqlaccess の安全性を高めた。

- Persistent Systems Private Limited http://www.pspl.co.in/konark/ のKonark IA-64 Centre。MySQL サーバの Win64 移植の支援。 , Albert Chin-A-Young.

  Tru64 用アップデートの設定、大きなファイルのサポート、優れた TCP ラッパのサポート。

- John Birrell

  OS/2 用の pthread_mutex() エミュレーション。

- Benjamin Pflugmann

  INSERTS を処理する拡張 MERGE テーブル。MySQL メーリングリストのアクティブメンバ。

- Jocelyn Fournier

  （特に MySQL 4.1 サブクエリコードの）無数のバグを発見しレポートした。

- Marc Liyanage

  Mac OS X パッケージを保守し、Mac OS X PKG の作成方法に関する非常に有益なフィードバックを提供。

- Robert Rutherford

  QNX 移植に関する非常に有益な情報とフィードバックを提供。

その他の貢献者、バグ発見者、テスタ-: 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

メーリングリストメンバから提供された多くのバグレポートおよびパッチ

MySQL メーリングリストの質問に回答いただいた方々に感謝いたします。

- Daniel Koch <dkoch@amcity.com>

  Irix セットアップ。

- Luuk de Boer <luuk@wxs.nl>

  ベンチマークの質問。

- Tim Sailer <tps@users.buoy.com>

  DBD-mysql の質問。

- Boyd Lynn Gerber <gerberb@zenez.com>

  SCO 関連の質問。

- Richard Mehalick <RM186061@shellus.com>

  xmysql 関連の質問と基本的なインストールの質問。

- Zeev Suraski <bourbon@netvision.net.il>

  Apache モジュール構成の質問（ログおよび権限）、PHP 関連の質問、SQL 文法関連の質問、その他一般的な質問。

- Francesc Guasch <frankie@citel.upc.es>

  一般的な質問。

- Jonathan J Smith <jsmith@wtp.net>

  Linux の OS 特定事項、SQL 構文、および何らかの作業を要するその他の事柄に関する質問。

- David Sklar <sklar@student.net>

  PHP および Perl からの MySQL の使用。

- Alistair MacDonald <A.MacDonald@uel.ac.uk>

  まだ指定されていないが、柔軟性があり Linux およびおそらく HP-UX も操作できる。 ユーザが mysqlbug を使用するように努力する予定。

- John Lyon <jlyon@imag.net>

  .rpm ファイルを使った、またはソースからのコンパイルによる Linux システムへの MySQL のインストールに関する質問。

- Lorvid Ltd. <lorvid@WOLFENET.com>

簡単な請求/ライセンス/サポート/著作権の問題。

- Patrick Sherrill <patrick@coconet.com>

  ODBC および VisualC++ インタフェースに関する質問。

- Randy Harmon <rjharmon@uptimecomputers.com>

  DBD、Linux、いくつかの SQL 構文に関する質問。

C.3. ドキュメント作成者および翻訳者

以下、MySQL ドキュメントの作成、および MySLQ のドキュメントまたはエラーメッセージの翻訳にご協力いただいた方々です。

- Paul DuBois

  このマニュアルを正確にわかりやすくするために、現在もご協力いただいている。 Monty と David の英語ドキュメントのリライトも含まれている。

- Kim Aldale

  Monty および David の初期の英語ドキュメントのリライト。

- Michael J. Miller Jr. <mke@terrapin.turbolift.com>

  初回の MySQL マニュアルの作成。FAQ (ずいぶんと前に MySQL マニュアルに変更) の多くのスペリングまたは言語を修正。

- Yan Cailin

  2000 年始めに、コード化された Big5 と HK (http://mysql.hitstar.com/) バージョンが準拠している中国語 (簡体字) に MySQL リファレンスマニュアルを初めて翻訳。 Personal home page at linuxdb.yeah.net。

- Jay Flaherty <fty@mediapulse.com>

  マニュアル内の Perl DBI/DBD セクションの大部分に携わる。

- Paul Southworth <pauls@etext.org>、Ray Loyzaga <yar@cs.su.oz.au>

  リファレンスマニュアルの校正。

- Therrien Gilbert <gilbert@ican.net>、 Jean-Marc Pouyot <jmp@scalaire.fr>

  フランス語のエラーメッセージ。

- Petr Snajdr、<snajdr@pvt.net>

  チェコ語のエラーメッセージ。

- Jaroslaw Lewandowski <jotel@itnet.com.pl>

ポーランド語のエラーメッセージ。

- Miguel Angel Fernandez Roiz

  スペイン語のエラーメッセージ。

- Roy-Magne Mo <rmo@www.hivolda.no>

  ノルウェー語のエラーメッセージとバージョン 3.21.# のテスト。

- Timur I. Bakeyev <root@timur.tatarstan.ru>

  ロシア語のエラーメッセージ。

- <brenno@dewinter.com> & Filippo Grassilli <phil@hyppo.com>

  イタリア語のエラーメッセージ。

- Dirk Munzinger <dirk@trinity.saar.de>

  ドイツ語のエラーメッセージ。

- Billik Stefan <billik@sun.uniag.sk>

  スロバキア語のエラーメッセージ。

- Stefan Saroiu <tzoompy@cs.washington.edu>

  ルーマニア語のエラーメッセージ。

- Peter Feher

  ハンガリー語のエラーメッセージ。

- Roberto M. Serqueira

  ポルトガル語のエラーメッセージ。

- Carsten H. Pedersen

  デンマーク語のエラーメッセージ。

- Arjen G. Lentz

  オランダ語のエラーメッセージ。以前の部分翻訳の完成（さらに、整合性とスペルについてもチェック）。

C.4. MySQL で使用されているライブラリと MySQL 付属のライブラリ

以下は、MySQL のコンパイルとインストールを容易にするために、MySQL サーバソースに追加したライブラリの作成者一覧です。これらライブラリを作成し、作業の簡易化にご尽力いただいた方々に感謝申し上げます。

- Fred Fish

優れた C デバッグとトレースライブラリ。Monty により、このライブラリに対して多くの細かい改善（速度と補足オプション）が行われた。

- Richard A. O'Keefe

  パブリックデーモンの文字列ライブラリ。

- Henry Spencer

  Regex ライブラリ。WHERE column REGEXP regexp で使用。

- Chris Provenzano

  ユーザレベルのポータブル pthread。（著作権から判断）この製品には、Chris Provenzano、the University of California, Berkeley および貢献者によって開発されたソフトウェアが含まれている。現在、社内で、Monty によりパッチが適用された バージョン 1_60_beta6 が使用されている（mit-pthreads/Changes-mysql参照）。

- Jean-loup Gailly および Mark Adler

  zlib ライブラリ（Windows の MySQL で使用）。

- Bjorn Benson

  MySQL を --debug で構成する際に使用される safe_malloc（メモリチェッカ）。

- フリーソフトウェア財団 (Free Software Foundation, FSF)

  readline ライブラリ（mysql コマンドラインクライアントが使用）。

- NetBSD 協会

  libedit パッケージ（mysql コマンドラインクライアントが任意に使用）。

C.5. MySQL をサポートするパッケージ

多くのユーザが MySQL とともに使用する重要な API、パッケージ、およびアプリケーションの作成者または保守者の一覧を以下に示します。

最新情報への更新を簡易にするため、この一覧にはすべてのパッケージは含まれていません。他のパッケージについては、http://www.mysql.com/portal/software のソフトウェアポータルを参照してください。

- Tim Bunce、Alligator Descartes

  DBD (Perl) インタフェース。

- Andreas Koenig <a.koenig@mind.de>

  MySQL サーバ用 Perl インタフェース。

- Jochen Wiedmann <wiedmann@neckar-alb.de>

Perl DBD::mysql モジュールの保守。

- Eugene Chan <eugene@acenet.com.sg>

  MySQL サーバ用 PHP の移植。

- Georg Richter

  MySQL 4.1 のテストとバグ検出。 MySQL 4.1 以上で使用するための新規 PHP 5.0 mysqli 拡張（API）。

- Giovanni Maruzzelli <maruzz@matrice.it>

  iODBC の移植（Unix ODBC）。

- Xavier Leroy <Xavier.Leroy@inria.fr>

  LinuxThreads（Linux 上で MySQL サーバが使用）の作成者。

C.6. MySQL の作成に使用したツール

以下、MySQL を作成するために使用したツールの一覧です。 ここで、これらのツールを作成した方々に感謝申し上げます。これらのツールがなければ、MySQL は今日の形になりませんでした。

- フリーソフトウェア財団 (Free Software Foundation, FSF)

  優れたコンパイラ（gcc）と優れたデバッガ（gdb）および libc ライブラリを提供。このライブラリの strto.c を使用して、Linux で動作するいくつかのコードを作成した。

- フリーソフトウェア財団 と Xemacs 開発チーム

  MySQL AB でほとんどすべてのユーザが使用した優れた工ディタ環境を提供。

- Julian Seward

  MySQL のバグを検出に役だった非常に優れたメモリチェッカ valgrind（これがなければバグ検出は困難であった）の作成者。

- Dorothea Lutkehaus および Andreas Zeller

  gdb に対する優れたグラフィカルフロントエンドである DDD（データ表示デバッガ）を提供。

C.7. MySQL のサポータ

MySQL AB は、MySQL サーバと MySQL マニュアルの全著作権を所有していますが、新機能の開発に対する融資、MySQL サーバ開発用のハードウェアの提供など、MySQL サーバの開発を助成していただいた以下の会社に感謝申し上げます。

- VA Linux / Andover.net

資金提供を行う。

- NuSphere

  MySQL マニュアルの編集。

- Stork Design studio

  1998 〜 2000 年の MySQL Web サイト。

- Intel

  Windows および Linux プラットフォーム上での開発に貢献。

- Compaq

  Linux/Alpha 上での開発に貢献。

- SWSoft

  埋め込み mysqld バージョンでの開発。

- FutureQuest

  --skip-show-database。

付録 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 項1.6.「MySQL の今後（TODO）」.

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!

The date mentioned with a release version is the date of the last BitKeeper ChangeSet that this particular release has been based on, not the date when the packages have been made available. The binaries are usually made available a few days after the date of the tagged ChangeSet - building and testing all packages takes some time.

D.1. Changes in release 5.0.0 (Development)

For the time being, version 5.0 is only available in source code. See 項2.3.4.「開発ソースツリーからのインストール」.

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, that is, global and local type.
- Deprecated the update log (no longer supported). It is fully replaced by the binary log.
- User variable names are now case insensitive: if you do SET @a=10; then SELECT @A; will now return 10. Of course, the content of the variable is still case sensitive; only the name of this variable is case insensitive.

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 and derived tables (unnamed views). See 項6.4.2.「サブクエリ構文」.
- 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 項6.4.3.「INSERT 構文」.
- A newly designed GROUP_CONCAT() aggregate function. See 項6.3.7.「GROUP BY 節で使用する関数と修飾子」.
- Extensive Unicode (UTF8) support.
- Character sets can be defined per column, table, and database.

- New key cache for MyISAM tables with many tunable parameters. You can have multiple key caches, preload index into caches for batches...
- BTREE index on HEAP tables.
- Support for OpenGIS spatial types (geographical data). See 章 10. MySQL における空間情報の機能.
- SHOW WARNINGS shows warnings for the last command. See 項4.6.8.9. 「SHOW WARNINGS | ERRORS」.
- Faster binary protocol with prepared statements and parameter binding. See 項11.1.4. 「C API のプリペアドステートメント」.
- You can now issue multiple statements with a single C API call and then read the results in one go. See 項11.1.8. 「C API における複数クエリの実行の取り扱い」.
- Create Table: CREATE [TEMPORARY] TABLE [IF NOT EXISTS] table2 LIKE table1.
- Server based HELP command that can be used in the mysql command line client (and other clients) to get help for SQL statements.

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 4.1.2 (not released yet)

Functionality added or changed:

Bugs fixed:

- Packaging: Fixed a bug in the Mac OS PKG postinstall script (mysql_install_db was called with an obsolete argument).
- Packaging: Added missing file mysql_create_system_tables to the server RPM package. This bug was fixed for the 4.1.1 RPMs by updating the MySQL-server RPM from MySQL-server-4.1.1-0 to MySQL-server-4.1.1-1. The other RPMs were not affected by this change.
- Fixed a bug in myisamchk and CHECK TABLE that sometimes resulted in a spurious error Found key at page ..... that points to record outside datafile for a table with a FULLTEXT index.
- Fixed a hang in full-text indexing of EUC-JP (ujis) data.
- Fixed a crash in full-text indexing of UTF-8 data.
- Replication: a rare race condition in the slave SQL thread that could lead to an incorrect complaint that the relay log is corrupted. (Bug#2011)
- Replication: if an administrative command on a table (OPTIMIZE TABLE, REPAIR TABLE etc) was run on the slave, this could sometimes stop the slave SQL thread (this did not lead to any corruption; one just had to type START SLAVE to get replication going again). (Bug#1858)
- Replication: in the slave SQL thread, a multi-table UPDATE could produce an incorrect complaint that some record was not found in one table, if the UPDATE was preceded by a INSERT ... SELECT. (Bug#1701)

D.2.2. Changes in release 4.1.1 (01 Dec 2003)

Functionality added or changed:

- Added IGNORE option for DELETE statement.
- The MySQL source distribution now also includes the MySQL Internals Manual internals.texi.
- Added mysql_set_server_option() C API client function to allow multiple statement handling in the server to be enabled or disabled.
- The mysql_next_result() C API function now returns -1 if there are no more result sets.
- Renamed CLIENT_MULTI_QUERIES connect option flag to CLIENT_MULTI_STATEMENTS. To allow for a transition period, the old option will continue to be recognized for a while.
- Require DEFAULT before table and database default character set. This enables us to use ALTER TABLE table_name ... CHARACTER SET=... to change the character set for all CHAR, VARCHAR, and TEXT columns in a table.
- Added MATCH ... AGAINST(... WITH QUERY EXPANSION) and the ft_query_expansion_limit server variable.
- Removed unused ft_max_word_len_for_sort server variable.
- Full-text search now supports multi-byte character sets and the Unicode utf8 character set. (The Unicode ucs2 character set is not yet supported.)
- Phrase search in MATCH ... AGAINST (... IN BOOLEAN MODE) no longer matches partial words.
- Added aggregate function BIT_XOR() for bitwise XOR operations.
- Replication over SSL now works.
- The START SLAVE statement now supports an UNTIL clause for specifying that the slave SQL thread should be started but run only until it reaches a given position in the master's binary logs or in the slave's relay logs.
- Produce warnings even for single-row INSERT statements, not just for multiple-row INSERT statements. Previously, it was necessary to set SQL_WARNINGS=1 to generate warnings for single-row statements.
- Added delimiter (\d) command to the mysql command-line client for changing the statement delimiter (terminator). The default delimiter is semicolon.
- CHAR, VARCHAR, and TEXT columns now have lengths measured in characters rather than in bytes. The character size depends on the column's character set. This means, for example, that a CHAR(n) column for a multi-byte character set will take more storage than before. Similarly, index values on such columns are measured in characters, not bytes.
- LIMIT no longer accepts negative arguments (they used to be treated as very big positive numbers before).
- The DATABASE() function now returns NULL rather than the empty string if there is no database selected.
- Added --sql-mode=NO_AUTO_VALUE_ON_ZERO option to suppress the usual behavior of generating the next sequence number when zero is stored in an AUTO_INCREMENT column. With this mode enabled, zero is stored as zero; only storing NULL generates a sequence number.

- Warning: Incompatible change! Client authentication now is based on 41-byte passwords in the user table, not 45-byte passwords as in 4.1.0. Any 45-byte passwords created for 4.1.0 must be reset after running the mysql_fix_privilege_tables script.
- Added MySQL Server option and global variable 'secure-auth' that disallows authentication for accounts that have old (pre-4.1.1) passwords.
- Added MySQL command line client option 'secure-auth'. If this option is set, client will refuse to send password in old (pre-4.1.1) format.
- Warning: Incompatible change! Renamed the C API mysql_prepare_result() function to mysql_get_metadata() as the old name was confusing.
- Added DROP USER 'username'@'hostname' statement to drop an account that has no privileges.
- The interface to aggregated UDF functions has changed a bit. You must now declare a xxx_clear() function for each aggregate function XXX().
- The CONCAT_WS() function no longer skips empty strings.
- Added new ADDTIME(), DATE(), DATEDIFF(), LAST_DAY(), MAKEDATE(), MAKETIME(), MICROSECOND(), SUBTIME(), TIME(), TIMEDIFF(), TIMESTAMP(), UTC_DATE(), UTC_TIME(), UTC_TIMESTAMP(), and WEEKOFYEAR() functions.
- Added new syntax for ADDDATE() and SUBDATE(). The second argument now may be a number representing the number of days to be added to or subtracted from the first date argument.
- Added new type values DAY_MICROSECOND, HOUR_MICROSECOND, MINUTE_MICROSECOND, SECOND_MICROSECOND, and MICROSECOND for DATE_ADD(), DATE_SUB(), and EXTRACT().
- Added new %f microseconds format specifier for DATE_FORMAT() and TIME_FORMAT().
- All queries in which at least one SELECT does not use indexes properly now are written to the slow query log when long log format is used.
- It is now possible to create a MERGE table from MyISAM tables in different databases. Formerly, all the MyISAM tables had to be in the same database, and the MERGE table had to be created in that database as well.
- Added new COMPRESS(), UNCOMPRESS(), and UNCOMPRESSED_LENGTH() functions.
- When doing SET sql_mode='mode' for a complex mode (like ANSI), we now update the sql_mode variable to include all the individual options implied by the complex mode.
- Added the OLAP (On-Line Analytical Processing) function ROLLUP, which provides summary rows for each GROUP BY level.
- Added SQLSTATE codes for all server errors.
- Added mysql_sqlstate() and mysql_stmt_sqlstate() C API client functions that return the SQLSTATE error code for the last error.
- TIME columns with hour values greater than 24 were returned incorrectly to the client.

- ANALYZE, OPTIMIZE, REPAIR, and FLUSH statements are now stored in the binary log and thus replicated to slaves. This logging does not occur if the optional NO_WRITE_TO_BINLOG keyword (or its alias LOCAL) is given. Exceptions are that FLUSH LOGS, FLUSH MASTER, FLUSH SLAVE, and FLUSH TABLES WITH READ LOCK are not logged in any case. For a syntax example, see 項4.6.4.「FLUSH 構文」.
- New global variable RELAY_LOG_PURGE to enable or disable automatic relay log purging.
- LOAD DATA now produces warnings that can be fetched with SHOW WARNINGS.
- Added support for syntax CREATE TABLE table2 (LIKE table1) that creates an empty table table2 with a definition that is exactly the same as table1, including any indexes.
- CREATE TABLE table_name (...) TYPE=storage_engine now generates a warning if the named storage engine is not available. The table is still created as a MyISAM table, as before.
- Most subqueries are now much faster than before.
- Added PURGE BINARY LOGS as an alias for PURGE MASTER LOGS.
- Disabled the PURGE LOGS statement that was added in in version 4.1.0. The statement now should be issued as PURGE MASTER LOGS or PURGE BINARY LOGS.
- Added SHOW BDB LOGS as an alias for SHOW LOGS.
- Added SHOW MASTER LOGS (which had been deleted in version 4.1.0) as an alias for SHOW BINARY LOGS.
- Added Slave_IO_State and Seconds_Behind_Master columns to the output of SHOW SLAVE STATUS. Slave_IO_State indicates the state of the slave I/O thread, and Seconds_Behind_Master indicates the number of seconds by which the slave is late compared to the master.
- --lower-case-table-names=1 now also makes aliases case insensitive. (Bug#534)

Bugs fixed:

- Fixed merging types and length of result set columns for UNION operations. The types and lengths now are determined taking into account values for all SELECT statements in the UNION, not just the first SELECT.
- Fixed a bug in privilege handling that caused connections from certain IP addresses to be assigned incorrect database-level privileges. A connection could be assigned the database privileges of the previous successful authentication from one of those IP addresses, even if the IP address username and database name were different. (Bug#1636)
- Error-handling functions were not called properly when an error resulted from [CREATE | REPLACE| INSERT] ... SELECT statements.
- HASH, BTREE, RTREE, ERRORS, and WARNINGS no longer are reserved words. (Bug#724)
- Fix for bug in ROLLUP when all tables were const tables. (Bug#714)
- Fixed a bug in UNION that prohibited NULL values from being inserted into result set columns where the first SELECT of the UNION retrieved NOT NULL columns.

- Fixed name resolution of columns of reduced subqueries in unions. (Bug#745)
- Fixed memory overrun in subqueries in select list with WHERE clause bigger than outer query WHERE clause. (Bug#726)
- Fixed a bug that caused MyISAM tables with FULLTEXT indexes created in 4.0.x to be unreadable in 4.1.x.
- Fixed a data loss bug in REPAIR TABLE ... USE_FRM when used with tables that contained TIMESTAMP columns and were created in 4.0.x.
- Fixed reduced subquery processing in ORDER BY/GROUP BY clauses. (Bug#442)
- Fixed name resolution of outer columns of subquery in INSERT/REPLACE statements. (Bug#446)
- Fixed bug in marking columns of reduced subqueries. (Bug#679)
- Fixed a bug that made CREATE FULLTEXT INDEX syntax illegal.
- Fixed a crash when a SELECT that required a temporary table (marked by Using temporary in EXPLAIN output) was used as a derived table in EXPLAIN command. (Bug#251)
- Fixed a rare table corruption bug in DELETE from a big table with a new (created by MySQL-4.1) full-text index.
- LAST_INSERT_ID() now returns 0 if the last INSERT statement didn't insert any rows.
- Fixed missing last character in function output. (Bug#447)
- Fixed a rare replication bug when a transaction spanned two or more relay logs, and the slave was stopped while executing the part of the transaction that was in the second or later relay log. Then replication would resume at the beginning of the second or later relay log, which was incorrect. (It should resume at BEGIN, in the first relay log.) (Bug#53)
- CONNECTION_ID() now is properly replicated. (Bug#177)
- The new PASSWORD() function in 4.1 is now properly replicated. (Bug#344)
- Fixed a bug with double freed memory.
- Fixed a crashing bug in UNION operations that involved temporary tables.
- 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 an uninitialized array of reference pointers.
- Fixed a bug in the USER() function caused by an error in the size of the allocated string.
- Fixed a crashing bug when attempting to create a table containing a spatial (GIS) column with a storage engine that does not support spatial types.
- Fixed a crashing bug in UNION caused by the empty select list and a non-existent column being used in some of the individual SELECT statements.
- Fixed a replication bug with a 3.23 master and a 4.0 slave: The slave lost the replicated temporary tables if FLUSH LOGS was issued on the master. (Bug#254)

- Fixed a security bug: A server compiled without SSL support still allowed connections by users that had the REQUIRE SSL option specified for their accounts.
- When an undefined user variable was used in a updating query on the master (such as INSERT INTO t VALUES(@a), where @a had never been set by this connection before), the slave could replicate the query incorrectly if a previous transaction on the master used a user variable of the same name. (Bug#1331)
- Fixed bug with prepared statements: Using the ? prepared statement parameter as the argument to certain functions or statement clauses caused a server crash when mysql_prepare() was invoked. (Bug#1500)
- Fixed bug with prepared statements: after call to mysql_stmt_prepare placeholders became allowed in all consequent statements, even if they are not prepared (Bug#1946)

D.2.3. Changes in release 4.1.0 (03 Apr 2003: Alpha)

Functionality added or changed:

- New more secure client authentication based on 45-byte passwords in the user table.
- New CRC32() function to compute cyclic redundancy check value.
- 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; you 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 mysql_get_server_version() C API client function.
- Fixed bug in libmysqlclient that fetched column 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 項4.6.7.「KILL 構文」.
- REPAIR TABLE now can be killed. See 項4.6.7.「KILL 構文」.
- Allow empty index lists to be specified for USE INDEX, IGNORE INDEX, and FORCE INDEX.

- DROP TEMPORARY TABLE now drops only temporary tables and doesn't end transactions.
- Added support for UNION in derived tables.
- Warning: Incompatible change! TIMESTAMP is now returned as a string of type 'YYYY-MM-DD HH:MM:SS' 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.
- New CONVERT(... USING ...) syntax for converting string values between character sets.
- 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 章 10. MySQL における空間情報の機能.
- SHOW [COUNT(*)] WARNINGS shows warnings from the last command.
- One can specify a column type for a colum 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).
- Added new VARIANCE(expr) function 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 storage engine is now included in the standard binaries, adding transactions, row-level locking, and foreign keys. See 項7.5. 「InnoDB テーブル」.
- A query cache, offering vastly increased performance for many applications. By caching complete result sets, later

identical queries can return instantly. See 項6.9.「MySQL クエリキャッシュ」.

- Improved full-text indexing with boolean mode, truncation, and phrase searching. See 項6.8.「MySQL 全文検索」.
- Enhanced MERGE tables, now supporting INSERT statements and AUTO_INCREMENT. See 項7.2.「MERGE テーブル」.
- UNION syntax in SELECT. See 項6.4.1.2.「UNION 構文」.
- Multiple-table DELETE statements. See 項6.4.5.「DELETE 構文」.
- libmysqld, the embedded server library. See 項11.1.15.「組み込み MySQL サーバライブラリ libmysqld」.
- Additional GRANT privilege options for even tighter control and security. See 項4.4.1.「GRANT および REVOKE の構文」.
- Management of user resources in the GRANT system, particularly useful for ISPs and other hosting providers. See 項4.4.7.「ユーザリソースの制限」.
- Dynamic server variables, allowing configuration changes to be made without having to stop and restart the server. See 項5.5.6.「SET 構文」.
- Improved replication code and features. See 項4.11.「MySQL のレプリケーション」.
- 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 4.0.17 (not released yet)

Functionality added or changed:

- Added four new modes to WEEK(..., mode) function. See WEEK(date: (mode)). (Bug#1178)
- Changed the default Windows service name for mysqld from MySql to MySQL. This should not affect usage, because service names are not case sensitive.
- When you install mysqld as a service on Windows systems, mysqld will read startup options in option files from the option group with the same name as the service name. (Except when the service name is MySQL).

Bugs fixed:

- Code cleanup: Fixed a few code defects (potential memory leaks, null pointer dereferences, uninitialized variables). Thanks to Reasoning Inc. for informing us about these findings.
- Filesort was never shown in EXPLAIN if query contained an ORDER BY NULL clause. (Bug#1335)
- Fixed invalidation of whole query cache on DROP DATABASE. (Bug#1898)

- Fixed bug in range optimizer that caused wrong results for some unlikely AND/OR queries. (Bug#1828)
- Fixed a crash in ORDER BY when ordering by expression and identifier. (Bug#1945)
- Fixed a crash in an open HANDLER when an ALTER TABLE was executed in a different connection. (Bug#1826)
- Fixed a bug in trunc* operator of full-text search which sometimes caused MySQL not to find all matched rows.
- Fixed bug in zero prepending to DECIMAL column type.
- Fixed optimizer bug, introduced in 4.0.16, when REF access plan was preferred to more efficient RANGE on another column.
- Fixed problem when installing a MySQL server as a Windows service using a command of the form mysqld --install mysql --defaults-file=path-to-file.
- Fixed an incorrect result from a query that uses only const tables (such as one-row tables) and non-constant expression (such as RAND()). (Bug#1271)
- Fixed bug when the optimizer did not take SQL_CALC_FOUND_ROWS into account if LIMIT clause was present. (Bug#1274)
- mysqlbinlog now asks for a password at the console when the -p or --password option is used with no argument. This is consistent with the way that other clients such mysqladmin and mysqldump already behave. Note: A consequence of this change is that it is no longer possible to invoke mysqlbinlog as mysqlbinlog -p pass_val (with a space between the -p option and the following password value). (Bug#1595)
- Fixed bug accidentally introduced in 4.0.16 where the slave SQL thread deleted its replicated temporary tables when STOP SLAVE was issued.
- In a ``chain'' replication setup A->B->C, if 2 sessions on A updated temporary tables of the same name at the same time, the binary log of B became incorrect, resulting in C becoming confused. (Bug#1686)
- In a ``chain'' replication setup A->B->C, if STOP SLAVE was issued on B while it was replicating a temporary table from A, then when START SLAVE was issued on B, the binary log of B became incorrect, resulting in C becoming confused. (Bug#1240)
- When MASTER_LOG_FILE and MASTER_LOG_POS were not specified, CHANGE MASTER used the coordinates of the slave I/O thread to set up replication, which broke replication if the slave SQL thread lagged behind the slave I/O thread. This caused the slave SQL thread to lose some events. The new behavior is to use the coordinates of the slave SQL thread instead. See 項4.11.8.1.「CHANGE MASTER TO」. (Bug#1870)
- Now if integer is stored or converted to TIMESTAMP or DATETIME value checks of year, month, day, hour, minute and second ranges are performed and numbers representing illegal timestamps are converted to 0 value. This behavior is consistent with manual and with behavior of string to TIMESTAMP/DATETIME conversion. (Bug#1448)
- Fixed bug when BIT_AND() and BIT_OR() group functions returned incorrect value if SELECT used a temporary table and no rows were found. (Bug#1790)
- Fixed bug with BIT_AND() still returning signed value for an empty set in some cases. (Bug#1972)
- Fixed bug with ^ (XOR) and >> (bit shift) still returning signed value in some cases. (Bug#1993)

- Replication: a rare race condition in the slave SQL thread, which could lead to a wrong complain that the relay log is corrupted. (Bug#2011)
- Replication: if an administrative command on a table (OPTIMIZE TABLE, REPAIR TABLE etc) was run on the slave, this could sometimes stop the slave SQL thread (this did not led to any corruption; one just had to type START SLAVE to get replication going again). (Bug#1858)
- Replication: in the slave SQL thread, a multi-table UPDATE could produce a wrong complain that some record was not found in one table, if the UPDATE was preceded by a INSERT ... SELECT. (Bug#1701)

D.3.2. Changes in release 4.0.16 (17 Oct 2003)

Functionality added or changed:

- Write memory allocation information to error log when doing mysqladmin debug. This works only on systems that support the mallinfo() call (like newer Linux systems).
- Added the following new server variables to allow more precise memory allocation: range_alloc_block_size, query_alloc_block_size, query_prealloc_size, transaction_alloc_block_size, and transaction_prealloc_size.
- mysqlbinlog now reads option files. To make this work one must now specify --read-from-remote-server when reading binary logs from a MySQL server. (Note that using a remote server is deprecated and may disappear in future mysqlbinlog versions).
- Block SIGPIPE signals also for non-threaded programs. The blocking is moved from mysql_init() to mysql_server_init(), which is automatically called on the first call to mysql_init().
- Added --libs_r and --include options to mysql_config.
- New `> prompt for mysql. This prompt is similar to the '> and "> prompts, but indicates that an identifier quoted with backticks was begun on an earlier line and the closing backtick has not yet been seen.
- Updated mysql_install_db to be able to use the local machine's IP address instead of the host name when building the initial grant tables if skip-name-resolve has been specified. This option can be helpful on FreeBSD to avoid thread-safety problems with the FreeBSD resolver libraries. (Thanks to Jeremy Zawodny for the patch.)
- A documentation change: Added a note that when backing up a slave, it is necessary also to back up the master.info and relay-log.info files, as well as any SQL_LOAD-* files located in the directory specified by the --slave-load-tmpdir option. All these files are needed when the slave resumes replication after you restore the slave's data.

Bugs fixed:

- Fixed a spurious error ERROR 14: Can't change size of file (Errcode: 2) on Windows in DELETE FROM table_name without a WHERE clause or TRUNCATE TABLE table_name, when table_name is a MyISAM table. (Bug#1397)
- Fixed a bug that resulted in thr_alarm queue is full warnings after increasing the max_connections variable with SET GLOBAL. (Bug#1435)

- Made LOCK TABLES to work when Lock_tables_priv is granted on the database level and Select_priv is granted on the table level.
- Fixed crash of FLUSH QUERY CACHE on queries that use same table several times (Bug#988).
- Fixed core dump bug when setting an enum system variable (such as SQL_WARNINGS) to NULL.
- Extended the default timeout value for Windows clients from 30 seconds to 1 year. (The timeout that was added in MySQL 4.0.15 was way too short). This fixes a bug that caused ERROR 2013: Lost connection to MySQL server during query for queries that lasted longer than 30 seconds, if the client didn't specify a limit with mysql_options(). Users of 4.0.15 on Windows should upgrade to avoid this problem.
- More ``out of memory'' checking in range optimizer.
- Fixed and documented a problem when setting and using a user variable within the same SELECT statement. (Bug#1194).
- Fixed bug in overrun check for BLOB values with compressed tables. This was a bug introduced in 4.0.14. It caused MySQL to regard some correct tables containing BLOB values as corrupted. (Bug#770, Bug#1304, and maybe Bug#1295)
- SHOW GRANTS showed USAGE instead of the real column-level privileges when no table-level privileges were given.
- When copying a database from the master, LOAD DATA FROM MASTER dropped the corresponding database on the slave, thus erroneously dropping tables that had no counterpart on the master and tables that may have been excluded from replication using replicate-*-table rules. Now LOAD DATA FROM MASTER no longer drops the database. Instead, it drops only the tables that have a counterpart on the master and that match the replicate-*-table rules. replicate-*-db rules can still be used to include or exclude a database as a whole from LOAD DATA FROM MASTER. A database will also be included or excluded as a whole if there are some rules like replicate-wild-do-table=db1.% or replicate-wild-ignore-table=db1.%, as is already the case for CREATE DATABASE and DROP DATABASE in replication. (Bug#1248)
- Fixed a bug where mysqlbinlog crashed with a segmentation fault when used with the -h or --host option. (Bug#1258)
- Fixed a bug where mysqlbinlog crashed with a segmentation fault when used on a binary log containing only final events for LOAD DATA. (Bug#1340)
- Fixed compilation problem when compiling with OpenSSL 0.9.7 with disabled old DES support (If OPENSSL_DISABLE_OLD_DES_SUPPORT option was enabled).
- Fixed a bug when two (or more) MySQL servers were running on the same machine, and they were both slaves, and at least one of them was replicating some LOAD DATA INFILE command from its master. The bug was that one slave MySQL server sometimes deleted the SQL_LOAD-* files (used for replication of LOAD DATA INFILE and located in the slave-load-tmpdir directory, which defaults to tmpdir) belonging to the other slave MySQL server of this machine, if these slaves had the same slave-load-tmpdir directory. When that happened, the other slave could not replicate LOAD DATA INFILE and complained about not being able to open some SQL_LOAD-* file. (Bug#1357)
- If LOAD DATA INFILE failed for a small file, the master forgot to write a marker (a Delete_file event) in its binary log, so the slave could not delete 2 files (SQL_LOAD-*.info and SQL_LOAD-*.data from its tmpdir. (Bug#1391)
- On Windows, the slave forgot to delete a SQL_LOAD-*.info file from tmpdir after successfully replicating a LOAD DATA

INFILE command. (Bug#1392)

- When a connection terminates, MySQL writes DROP TEMPORARY TABLE statements to the binary log for all temporary tables which the connection had not explicitely dropped. MySQL forgot to backquote the database and table names in the statement. (Bug#1345)
- On some 64-bit machines (some HP-UX and Solaris machines), a slave installed with the 64-bit MySQL binary could not connect to its master (it connected to itself instead). (Bug#1256, Bug#1381)
- Code was introduced in MySQL 4.0.15 for the slave to detect that the master had died while writing a transaction to its binary log. This code reported an error in a legal situation: When the slave I/O thread was stopped while copying a transaction to the relay log, the slave SQL thread would later pretend that it found an unfinished transaction. (Bug#1475)

D.3.3. Changes in release 4.0.15 (03 Sep 2003)

IMPORTANT:

If you are using this release on Windows, you should upgrade at least your clients (any program that uses libmysql.lib) to 4.0.16 or above. This is because the 4.0.15 release had a bug in the Windows client library that causes Windows clients using the library to die with a Lost connection to MySQL server during query error for queries that take more than 30 seconds. This problem is specific to Windows; clients on other platforms are unaffected.

Functionality added or changed:

- mysqldump now correctly quotes all identifiers when communicating with the server. This assures that during the dump process, mysqldump will never send queries to the server that result in a syntax error. This problem is not related to the mysqldump program's output, which was not changed. (Bug#1148)
- Change result set metadata information so that MIN() and MAX() report that they can return NULL (this is true because an empty set will return NULL). (Bug#324)
- Produce an error message on Windows if a second mysqld server is started on the same TCP/IP port as an already running mysqld server.
- The mysqld server variables wait_timeout, net_read_timeout, and net_write_timeout now work on Windows. One can now also set timeouts for read and writes in Windows clients with mysql_options().
- Added option --sql-mode=NO_DIR_IN_CREATE to make it possible for slaves to ignore INDEX DIRECTORY and DATA DIRECTORY options given to CREATE TABLE. When this is mode is on, SHOW CREATE TABLE will not show the given directories.
- SHOW CREATE TABLE now shows the INDEX DIRECTORY and DATA DIRECTORY options, if they were specified when the table was created.
- The open_files_limit server variable now shows the real open files limit.
- MATCH ... AGAINST() in natural language mode now treats words that are present in more than 2,000,000 rows as stopwords.
- The Mac OS X installation disk images now include an additional MySQLStartupItem.pkg package that enables the

automatic startup of MySQL on system bootup. See 項2.1.3.「Mac OS X への MySQL のインストール」.

- Most of the documentation included in the binary tarball distributions (.tar.gz) has been moved into a subdirectory docs. See 項2.2.5.「インストールレイアウト」.
- The manual is now included as an additional info file in the binary distributions. (Bug#1019)
- The binary distributions now include the embedded server library (libmysqld.a) by default. Due to a linking problem with non-gcc compilers, it was not included in all packages of the initial 4.0.15 release. The affected packages were rebuilt and released as 4.0.15a. See 項1.5.1.2.「組み込み MySQL サーバ」.
- MySQL can now use range optimization for BETWEEN with non-constant limits. (Bug#991)
- Replication error messages now include the default database, so that users can check which database the failing query was run for.
- A documentation change: Added a paragraph about how the binlog-do-db and binlog-ignore-db options are tested against the database on the master (see 項4.10.4.「バイナリログ」), and a paragraph about how replicate-do-db, replicate-do-table and analogous options are tested against the database and tables on the slave (see 項4.11.6.「レプリケーションスタートアップオプション」).
- Now the slave does not replicate SET PASSWORD if it is configured to exclude the mysql database from replication (using for example replicate-wild-ignore-table=mysql.%). This was already the case for GRANT and REVOKE since version 4.0.13 (though there was Bug#980 in 4.0.13 & 4.0.14, which has been fixed in 4.0.15).
- Rewrote the information shown in the State column of SHOW PROCESSLIST for replication threads and for MASTER_POS_WAIT() and added the most common states for these threads to the documentation, see 項4.11.3.「レプリケーションの実装の詳細」.
- Added a test in replication to detect the case where the master died in the middle of writing a transaction to the binlog; such unfinished transactions now trigger an error message on the slave.
- A GRANT command that creates an anonymous user (that is, an account with an empty username) no longer requires FLUSH PRIVILEGES for the account to be recognized by the server. (Bug#473)
- CHANGE MASTER now flushes relay-log.info. Previously this was deferred to the next run of START SLAVE, so if mysqld was shutdown on the slave after CHANGE MASTER without having run START SLAVE, the relay log's name and position were lost. At restart they were reloaded from relay-log.info, thus reverting to their old (incorrect) values from before CHANGE MASTER and leading to error messages (as the old relay log did not exist any more) and the slave threads refusing to start. (Bug#858)

Bugs fixed:

- Fixed buffer overflow in password handling which could potentially be exploited by MySQL users with ALTER privilege on the mysql.user table to execute random code or to gain shell access with the UID of the mysqld process (thanks to Jedi/Sector One for spotting and reporting this bug).
- Fixed server crash on FORCE INDEX in a query that contained "Range checked for each record" in the EXPLAIN output. (Bug#1172)

- Fixed table/column grant handling - proper sort order (from most specific to less specific, see 項4.3.10. 「アクセス制御の段階 2: 要求確認」) was not honored. (Bug#928)
- Fixed rare bug in MYISAM introduced in 4.0.3 where the index file header was not updated directly after an UPDATE of split dynamic rows. The symptom was that the table had a corrupted delete-link if mysqld was shut down or the table was checked directly after the update.
- Fixed Can't unlock file error when running myisamchk --sort-index on Windows. (Bug#1119)
- Fixed possible deadlock when changing key_buffer_size while the key cache was actively used. (Bug#1088)
- Fixed overflow bug in MyISAM and ISAM when a row is updated in a table with a large number of columns and at least one BLOB/TEXT column.
- Fixed incorrect result when doing UNION and LIMIT #,# when one didn't use braces around the SELECT parts.
- Fixed incorrect result when doing UNION and ORDER BY .. LIMIT # when one didn't use braces around the SELECT parts.
- Fixed problem with SELECT SQL_CALC_FOUND_ROWS ... UNION ALL ... LIMIT # where FOUND_ROWS() returned incorrect number of rows.
- Fixed unlikely stack bug when having a BIG expression of type 1+1-1+1-1... in certain combinations. (Bug#871)
- Fixed the bug that sometimes prevented a table with a FULLTEXT index from being marked as "analyzed".
- Fixed MySQL so that the column length (in C API) for the second column in SHOW CREATE TABLE is always larger than the data length. The only known application that was affected by the old behavior was Borland dbExpress, which truncated the output from the command. (Bug#1064)
- Fixed crash in comparisons of strings using the tis620 character set. (Bug#1116)
- Fixed ISAM bug in MAX() optimization.
- myisamchk --sort-records=N no longer marks table as crashed if sorting failed because of an inappropriate key. (Bug#892)
- Fixed a minor bug in MyISAM compressed table handling that sometimes made it impossible to repair compressed table in "Repair by sort" mode. "Repair with keycache" (myisamchk --safe-recover) worked, though. (Bug#1015)
- Fixed bug in propagating the version number to the manual included in the distribution files. (Bug#1020)
- Fixed key sorting problem (a PRIMARY key declared for a column that is not explicitly marked NOT NULL was sorted after a UNIQUE key for a NOT NULL column).
- Fixed the result of INTERVAL when applied to a DATE value. (Bug#792)
- Fixed compiling of the embedded server library in the RPM spec file. (Bug#959)
- Added some missing files to the RPM spec file and fixed some RPM building errors that occured on Red Hat Linux 9. (Bug#998)
- Fixed incorrect XOR evaluation in WHERE clause. (Bug#992)

- Fixed bug with processing in query cache merged tables constructed from more then 255 tables. (Bug#930)
- Fixed incorrect results from outer join query (e.g. LEFT JOIN) when ON condition is always false, and range search in used. (Bug#926)
- Fixed a bug causing incorrect results from MATCH ... AGAINST() in some joins. (Bug#942)
- MERGE tables do not ignore "Using index" (from EXPLAIN output) anymore.
- Fixed a bug that prevented an empty table from being marked as "analyzed". (Bug#937)
- Fixed myisamchk --sort-records crash when used on compressed table.
- Fixed slow (as compared to 3.23) ALTER TABLE and related commands such as CREATE INDEX. (Bug#712)
- Fixed segmentation fault resulting from LOAD DATA FROM MASTER when the master was running without the --log-bin option. (Bug#934)
- Fixed a security bug: A server compiled without SSL support still allowed connections by users that had the REQUIRE SSL option specified for their accounts.
- Fixed a random bug: Sometimes the slave would replicate GRANT or REVOKE queries even if it was configured to exclude the mysql database from replication (for example, using replicate-wild-ignore-table=mysql.%). (Bug#980)
- The Last_Errno and Last_Error fields in the output of SHOW SLAVE STATUS are now cleared by CHANGE MASTER and when the slave SQL thread starts. (Bug#986)
- A documentation mistake: It said that RESET SLAVE does not change connection information (master host, port, user, and password), whereas it does. The statement resets these to the startup options (master-host etc) if there were some. (Bug#985)
- SHOW SLAVE STATUS now shows correct information (master host, port, user, and password) after RESET SLAVE (that is, it shows the new values, which are copied from the startup options if there were some). (Bug#985)
- Disabled propagation of the original master's log position for events because this caused unexpected values for Exec_Master_Log_Pos and problems with MASTER_POS_WAIT() in A->B->C replication setup. (Bug#1086)
- Fixed a segfault in mysqlbinlog when --position=x was used with x being between a Create_file event and its fellow Append_block, Exec_load or Delete_file events. (Bug#1091)
- mysqlbinlog printed superfluous warnings when using --database, which caused syntax errors when piped to mysql. (Bug#1092)
- Made mysqlbinlog --database filter LOAD DATA INFILE too (previously, it filtered all queries except LOAD DATA INFILE). (Bug#1093)
- mysqlbinlog in some cases forgot to put a leading '#' in front of the original LOAD DATA INFILE (this command is displayed only for information, not to be run; it is later reworked to LOAD DATA LOCAL with a different filename, for execution by mysql). (Bug#1096)
- binlog-do-db and binlog-ignore-db incorrectly filtered LOAD DATA INFILE (it was half-written to the binary log). This resulted in a corrupted binary log, which could cause the slave to stop with an error. (Bug#1100)

- When, in a transaction, a transactional table (such as an InnoDB table) was updated, and later in the same transaction a non-transactional table (such as a MyISAM table) was updated using the updated content of the transactional table (with INSERT ... SELECT for example), the queries were written to the binary log in an incorrect order. (Bug#873)
- When, in a transaction, INSERT ... SELECT updated a non-transactional table, and ROLLBACK was issued, no error was returned to the client. Now the client is warned that some changes could not be rolled back, as this was already the case for normal INSERT. (Bug#1113)
- Fixed a potential bug: When STOP SLAVE was run while the slave SQL thread was in the middle of a transaction, and then CHANGE MASTER was used to point the slave to some non-transactional statement, the slave SQL thread could get confused (because it would still think, from the past, that it was in a transaction).

D.3.4. Changes in release 4.0.14 (18 Jul 2003)

Functionality added or changed:

- InnoDB now supports indexing a prefix of a column. This means, in particular, that BLOB and TEXT columns can be indexed in InnoDB tables, which was not possible before.
- A documentation change: Function INTERVAL(NULL, ...) returns -1.
- Enabled INSERT from SELECT when the table into which the records are inserted is also a table listed in the SELECT.
- Allow CREATE TABLE and INSERT from any UNION.
- The SQL_CALC_FOUND_ROWS option now always returns the total number of rows for any UNION.
- Removed --table option from mysqlbinlog to avoid repeating mysqldump functionality.
- Comment lines in option files can now start from the middle of a line, too (like basedir=c:\mysql # installation directory).
- Changed optimizer slightly to prefer index lookups over full table scans in some boundary cases.
- Added thread-specific max_seeks_for_key variable that can be used to force the optimizer to use keys instead of table scans even if the cardinality of the index is low.
- Added optimization that converts LEFT JOIN to normal join in some cases.
- A documentation change: added a paragraph about failover in replication (how to use a surviving slave as the new master, how to resume to the original setup). See 項4.11.9.「レプリケーション FAQ」.
- A documentation change: added warning notes about safe use of the CHANGE MASTER command. See 項4.11.8.1.「CHANGE MASTER TO」.
- MySQL now issues a warning (not an error, as in 4.0.13) when it opens a table that was created with MySQL 4.1.
- Added --nice option to mysqld_safe to allow setting the niceness of the mysqld process. (Thanks to Christian Hammers for providing the initial patch.) (Bug#627)
- Added --read-only option to cause mysqld to allow no updates except from slave threads or from users with the SUPER privilege. (Original patch from Markus Benning).

- SHOW BINLOG EVENTS FROM x where x is less than 4 now silently converts x to 4 instead of printing an error. The same change was done for CHANGE MASTER TO MASTER_LOG_POS=x and CHANGE MASTER TO RELAY_LOG_POS=x.
- mysqld now only adds an interrupt handler for the SIGINT signal if you start it with the new --gdb option. This is because some MySQL users encountered strange problems when they accidentally sent SIGINT to mysqld threads.
- RESET SLAVE now clears the Last_Errno and Last_Error fields in the output of SHOW SLAVE STATUS.
- Added max_relay_log_size variable; the relay log will be rotated automatically when its size exceeds max_relay_log_size. But if max_relay_log_size is 0 (the default), max_binlog_size will be used (as in older versions). max_binlog_size still applies to binary logs in any case.
- FLUSH LOGS now rotates relay logs in addition to the other types of logs it already rotated.

Bugs fixed:

- Comparison/sorting for latin1_de character set was rewritten. The old algorithm could not handle cases like "sä" > "ßa". See 項4.7.1.1.「ドイツ語キャラクタセット」. In rare cases it resulted in table corruption.
- Fixed a problem with the password prompt on Windows. (Bug#683)
- ALTER TABLE ... UNION=(...) for MERGE table is now allowed even if some underlying MyISAM tables are read-only. (Bug#702)
- Fixed a problem with CREATE TABLE t1 SELECT x'41'. (Bug#801)
- Removed some incorrect lock warnings from the error log.
- Fixed memory overrun when doing REPAIR on a table with a multi-part auto_increment key where one part was a packed CHAR.
- Fixed a probable race condition in the replication code that could potentially lead to INSERT statements not being replicated in the event of a FLUSH LOGS command or when the binary log exceeds max_binlog_size. (Bug#791)
- Fixed a crashing bug in INTERVAL and GROUP BY or DISTINCT. (Bug#807)
- Fixed bug in mysqlhotcopy so it actually aborts for unsuccessful table copying operations. Fixed another bug so that it succeeds when there are thousands of tables to copy. (Bug#812)
- Fixed problem with mysqlhotcopy failing to read options from option files. (Bug#808)
- Fixed bugs in optimizer that sometimes prevented MySQL from using FULLTEXT indexes even though it was possible (for example, in SELECT * FROM t1 WHERE MATCH a,b AGAINST("index") > 0).
- Fixed a bug with ``table is full'' in UNION operations.
- Fixed a security problem that enabled users with no privileges to obtain information on the list of existing databases by using SHOW TABLES and similar commands.
- Fixed a stack problem on UnixWare/OpenUnix.

- Fixed a configuration problem on UnixWare/OpenUNIX and OpenServer.
- Fixed a stack overflow problem in password verification.
- Fixed a problem with max_user_connections.
- HANDLER without an index now works properly when a table has deleted rows. (Bug#787)
- Fixed a bug with LOAD DATA in mysqlbinlog. (Bug#670)
- Fixed that SET CHARACTER SET DEFAULT works. (Bug#462)
- Fixed MERGE table behavior in ORDER BY ... DESC queries. (Bug#515)
- Fixed server crash on PURGE MASTER LOGS or SHOW MASTER LOGS when the binary log is off. (Bug#733)
- Fixed password-checking problem on Windows. (Bug#464)
- Fixed the bug in comparison of a DATETIME column and an integer constant. (Bug#504)
- Fixed remote mode of mysqlbinlog. (Bug#672)
- Fixed ERROR 1105: Unknown error that occurred for some SELECT queries, where a column that was declared as NOT NULL was compared with an expression that took NULL value.
- Changed timeout in mysql_real_connect() to use poll() instead of select() to work around problem with many open files in the client.
- Fixed incorrect results from MATCH ... AGAINST used with a LEFT JOIN query.
- Fixed a bug that limited the maximum value for mysqld variables to 4294967295 when they are specified on the command line.
- Fixed a bug that sometimes caused spurious ``Access denied'' errors in HANDLER ... READ statements, when a table is referenced via an alias.
- Fixed portability problem with safe_malloc, which caused MySQL to give "Freeing wrong aligned pointer" errors on SCO 3.2.
- ALTER TABLE ... ENABLE/DISABLE KEYS could cause a core dump when done after an INSERT DELAYED statement on the same table.
- Fixed problem with conversion of localtime to GMT where some times resulted in different (but correct) timestamps. Now MySQL should use the smallest possible timestamp value in this case. (Bug#316)
- Very small query cache sizes could crash mysqld. (Bug#549)
- Fixed a bug (accidentally introduced by us but present only in version 4.0.13) that made INSERT ... SELECT into an AUTO_INCREMENT column not replicate well. This bug is in the master, not in the slave. (Bug#490)
- Fixed a bug: When an INSERT ... SELECT statement inserted rows into a non-transactional table, but failed at some point (for example, due to a ``Duplicate key'' error), the query was not written to the binlog. Now it is written to the binlog, with its error code, as all other queries are. About the slave-skip-errors option for how to handle partially completed

queries in the slave, see 項4.11.6.「レプリケーションスタートアップオプション」. (Bug#491)

- SET FOREIGN_KEY_CHECKS=0 was not replicated properly. The fix probably will not be backported to 3.23.
- On a slave, LOAD DATA INFILE which had no IGNORE or REPLACE clause on the master, was replicated with IGNORE. While this is not a problem if the master and slave data are identical (a LOAD that produces no duplicate conflicts on the master will produce none on the slave anyway), which is true in normal operation, it is better for debugging not to silently add the IGNORE. That way, you can get an error message on the slave and discover that for some reason, the data on master and slave are different and investigate why. (Bug#571)
- On a slave, LOAD DATA INFILE printed an incomplete ``Duplicate entry '%-.64s' for key %d'" message (the key name and value were not mentioned) in case of duplicate conflict (which does not happen in normal operation). (Bug#573)
- When using a slave compiled with --debug, CHANGE MASTER TO RELAY_LOG_POS could cause a debug assertion failure. (Bug#576)
- When doing a LOCK TABLES WRITE on an InnoDB table, commit could not happen, if the query was not written to the binary log (for example, if --log-bin was not used, or binlog-ignore-db was used). (Bug#578)
- If a 3.23 master had open temporary tables that had been replicated to a 4.0 slave, and the binlog got rotated, these temporary tables were immediately dropped by the slave (which caused problems if the master used them subsequently). This bug had been fixed in 4.0.13, but in a manner which caused an unlikely inconvenience: if the 3.23 master died brutally (power failure), without having enough time to automatically write DROP TABLE statements to its binlog, then the 4.0.13 slave would not notice the temporary tables have to be dropped, until the slave mysqld server is restarted. This minor inconvenience is fixed in 3.23.57 and 4.0.14 (meaning the master must be upgraded to 3.23.57 and the slave to 4.0.14 to remove the inconvenience). (Bug#254)
- If MASTER_POS_WAIT() was waiting, and the slave was idle, and the slave SQL thread terminated, MASTER_POS_WAIT() would wait forever. Now when the slave SQL thread terminates, MASTER_POS_WAIT() immediately returns NULL (``slave stopped''). (Bug#651)
- After RESET SLAVE; START SLAVE;, the Relay_Log_Space value displayed by SHOW SLAVE STATUS was too big by four bytes. (Bug#763)
- If a query was ignored on the slave (because of replicate-ignore-table and other similar rules), the slave still checked if the query got the same error code (0, no error) as on the master. So if the master had an error on the query (for example, ``Duplicate entry'' in a multiple-row insert), then the slave stopped and warned that the error codes didn't match. (Bug#797)

D.3.5. Changes in release 4.0.13 (16 May 2003)

Functionality added or changed:

- PRIMARY KEY now implies NOT NULL. (Bug#390)
- The Windows binary packages are now compiled with --enable-local-infile to match the Unix build configuration.
- Removed timing of tests from mysql-test-run. time does not accept all required parameters on many platforms (for example, QNX) and timing the tests is not really required (it's not a benchmark anyway).

- SHOW MASTER STATUS and SHOW SLAVE STATUS required the SUPER privilege; now they accept REPLICATION CLIENT as well. (Bug#343)
- Added multi-threaded MyISAM repair optimization and myisam_repair_threads variable to enable it. See 項4.6.8.4.「SHOW VARIABLES」.
- Added innodb_max_dirty_pages_pct variable which controls amount of dirty pages allowed in InnoDB buffer pool.
- CURRENT_USER() and Access denied error messages now report the 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 an error when it opens a table that was created with MySQL 4.1.
- 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(0xfeff AS UNSIGNED) < CAST(0xff AS UNSIGNED). This will be the default behavior in MySQL 4.1. (Bug#152)
- 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)
- Changed UPDATE ... LIMIT to apply the limit to rows that were matched, whether or not they actually were changed. Previously the limit was applied as a restriction on the number of rows changed.
- Tuned optimizer to favor 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 項7.5.3.「InnoDB 起動オプション」.
- 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 behavior was convenient for some users, others may suffer from it. Now one has to explicitely 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:

- Logged Access denied error message had incorrect Using password value. (Bug#398)
- 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)
- mysql command line client no longer looks for * commands inside backtick-quoted strings.
- Fixed Unknown error when using UPDATE ... LIMIT. (Bug#373)
- Fixed problem with ANSI mode and GROUP BY with constants. (Bug#387)
- Fixed bug with UNION and OUTER JOIN. (Bug#386)
- Fixed bug if one used a multiple-table UPDATE and the query required a temporary table bigger than tmp_table_size. (Bug#286)
- Run mysql_install_db with the -IN-RPM option for the Mac OS X installation to not fail on systems with improperly configured hostname configurations.
- 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)
- Fixed core dump bug when using FORMAT('nan',#). (Bug#284)
- Fixed name resolution bug with HAVING ... COUNT(DISTINCT ...).
- Fixed incorrect result from truncation operator (*) in MATCH ... AGAINST() in some complex joins.
- Fixed a crash in REPAIR ... USE_FRM command, when used on read-only, nonexisting table or a table with a crashed index file.
- 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 that SET SQL_BIG_SELECTS=1 works as documented (This corrects a new bug introduced in 4.0)
- Fixed some serious bugs in UPDATE ... ORDER BY. (Bug#241)
- Fixed unlikely problem in optimizing 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.
- FULLTEXT index stopped working after ALTER TABLE that converts TEXT column to CHAR. (Bug#283)
- Fixed a security problem with SELECT and wildcarded select list, when user only had partial column SELECT privileges on the table.
- 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, for example, 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 incorrect 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. (Bug#183, Bug#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 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 overflow bug in MyISAM when a row is inserted into a table with a large number of columns and at least one BLOB/TEXT column. Bug was caused by incorrect calculation of the needed buffer to pack data.
- 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 information instead of information from the linked list of threads.
- Removed unnecessary Mac OS X helper files from server RPM. (Bug#144)
- Allow optimization of multiple-table update for InnoDB tables as well.
- Fixed a bug in multiple-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 incorrect 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.3.6. Changes in release 4.0.12 (15 Mar 2003: Production)

Functionality added or changed:

- mysqld no longer reads options from world-writeable config files.
- Integer values between 9223372036854775807 and 9999999999999999999 are now regarded as unsigned longlongs, not as floats. This makes these values work similar to values between 10000000000000000000 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 multiple-table updates with InnoDB tables. This bug occurred as, in many cases, InnoDB tables cannot 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 multiple-table UPDATE statements when user had all privileges on the database where tables are located and there were any entries in tables_priv table, that is, 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 incorrect results from the query.
- Fixed rare bug with incorrect initialization of AUTO_INCREMENT column, as a secondary column in a multi-column key (see 項3.6.9.「AUTO_INCREMENT の使用」), 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 (for example, 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 (for example, 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.3.7. 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 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 incorrect).
- 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 multiple-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 multiple-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 incorrect values when used in OUTER JOIN.
- MIN(key_column) and MAX(key_column) could return incorrect 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.3.8. 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 optimization that if one does GROUP BY ... ORDER BY NULL then result is not sorted.
- New --ft-stopword-file command-line option for mysqld to replace/disable the built-in stopword list that is used in full-text searches. See 項4.6.8.4.「SHOW VARIABLES」.
- 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 a stack size larger than 128K 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 column 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 incorrect 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 an AUTO_INCREMENT column 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 columns then the slave data is corrupted and you should reinitialize the affected tables from the master.
- Fixed possible memory overrun when sending a BLOB value larger than 16M to the client.
- Fixed incorrect error message when setting a NOT NULL column 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.3.9. 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 optimizing 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.3.10. 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 support 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 >= 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 NULL values in CASE ... WHEN ...

D.3.11. 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 HPUX 10.20.
- Fixed some optimization 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.3.12. 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 loose 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 optimization (see 項4.6.8.4.「SHOW VARIABLES」) is no longer used when inserting small (less than 100) number of rows.
- Optimization 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 項6.3.6.2.「その他の各種関数」.
- 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 row_count OFFSET 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..)
- CHANGE MASTER and RESET SLAVE now require that slave threads be both already stopped; these commands will return an error if at least one of these two threads is running.

Bugs fixed:

- Fixed number of found rows returned in multi table updates
- Make --lower-case-table-names default on Mac OS X as the default file system (HFS+) is case insensitive. See 項6.1.3.「名前におけるケース依存」.
- 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 also solved problems with LEFT JOIN.
- 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 multiple-table-update to optimize it, make it safer and more bug free.
- LOCK TABLES now works together with multiple-table-update and multiple-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.
- RESET SLAVE caused a crash if the slave threads were running.

D.3.13. 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() optimization when used with JOIN and ON expressions.
- Added support for reading of MySQL 4.1 table definition files.
- BETWEEN behavior changed (see 項6.3.1.2. 「比較演算子」). 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 項7.5.16. 「InnoDB の変更履歴」 in this manual.

Bugs fixed:

- Fixed bug in MAX() optimization 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 behavior 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 that 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 multiple-table updates.
- Fixed a newly introduced bug that caused ORDER BY ... LIMIT row_count to not return all rows.
- Fixed a bug in multiple-table deletes when outer join is used on an empty table, which gets first to be deleted.
- Fixed a bug in multiple-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.3.14. 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 reworked in a future release.
- Fixed a bug with constant expression (for example, 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 the my_option struct. This affected at least the 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 datatype 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 optimization 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.3.15. Changes in release 4.0.3 (26 Aug 2002: Beta)

- Fixed problem with types of user variables. (Bug#551)
- 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, for example, 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 initialized.
- 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 BIGINT values 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 behavior of IF(condition,column,NULL) so that it returns the value of the column type.
- Made safe_mysqld 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 multiple-table UPDATE: when updating a table, do_select() became confused about reading records from a cache.
- Fixed bug in multiple-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 multiple-table DELETE when tables are re-ordered in the table initialization method and ref_lengths are of different sizes.
- Fixed two bugs in SELECT DISTINCT with large tables.

- Fixed bug in query cache initialization 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 incorrect 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 read_rnd_buffer_size.
- Renamed some SQL variables, but old names will still work until 5.0. See 項2.5.2.「バージョン 3.23 から 4.0 へのアップグレード」.
- 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 項6.3.3.2.「数学関数」.
- 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.3.16. 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 項6.3.6.2.「その他の各種関数」.
- Added AES_ENCRYPT() and AES_DECRYPT() functions to perform encryption according to AES standard (Rijndael). See 項6.3.6.2.「その他の各種関数」.
- Added --single-transaction option to mysqldump, allowing a consistent dump of InnoDB tables. See 項4.9.7.「mysqldump（テーブル構造とデータのダンプ）」.
- Fixed bug in innodb_log_group_home_dir in SHOW VARIABLES.
- Fixed a bug in optimizer with merge tables when non-unique values are used in summing up (causing crashes).
- Fixed a bug in optimizer 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 multiple-table DELETE when optimizer uses only indexes.
- 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 (incorrect 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 UNION. 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 optimization.
- 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.
- Optimized InnoDB sort-buffer handling to take less memory.
- Fixed bug in multiple-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 like 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 optimization 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 multiple-table UPDATE statement.
- Fixed bug in multiple-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 項6.3.5.「キャスト関数」.
- 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 項4.4.7.「ユーザリソースの制限」.
- Added mysql_secure_installation to the scripts/ directory.

D.3.17. 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 tbl_name DISABLE KEYS and ALTER TABLE tbl_name 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() initialization so that RAND(N) and RAND(N+1) are more distinct.
- Fixed core dump bug in UPDATE ... ORDER BY.
- In 3.23, INSERT INTO ... SELECT always had IGNORE enabled. Now MySQL will stop (and possibly roll back) by default in case of an error unless you specify IGNORE.
- 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.3.18. 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 behavior now is case-insensitive.
- TRUNCATE TABLE and DELETE FROM tbl_name are now separate functions. One bonus is that DELETE FROM tbl_name 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 multiple-table DELETE operations.
- A new HANDLER interface to MyISAM tables.
- Added support for INSERT on MERGE tables. Patch from Benjamin Pflugmann.
- Changed WEEK(date,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.
- Optimized queries of type: SELECT DISTINCT * from tbl_name ORDER by key_part1 LIMIT row_count.
- 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 ORDER BY syntax to UPDATE and DELETE.
- SHOW INDEXES is now a synonym for SHOW INDEX.
- Added ALTER TABLE tbl_name DISABLE KEYS and ALTER TABLE tbl_name 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 optimization.
- 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 項2.5.2.「バージョン 3.23 から 4.0 へのアップグレード」.

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 3.23.59 (not released yet)

- If a query was ignored on the slave (because of replicate-ignore-table and other similar rules), the slave still checked if the query got the same error code (0, no error) as on the master. So if the master had an error on the query (for example, ``Duplicate entry'' in a multiple-row insert), then the slave stopped and warned that the error codes didn't match. This is a backport of the fix for MySQL 4.0. (Bug#797)
- mysqlbinlog now asks for a password at console when the -p/--password option is used with no argument. This is how the other clients (mysqladmin, mysqldump..) already behave. Note that one now has to use mysqlbinlog -p<my_password>; mysqlbinlog -p <my_password> will not work anymore (in other words, put no space after -p). (Bug#1595)
- On some 64-bit machines (some HP-UX and Solaris machines), a slave installed with the 64-bit MySQL binary could not connect to its master (it connected to itself instead). (Bug#1256, #1381)
- Fixed a Windows-specific bug present since MySQL version 3.23.57 and 3.23.58, which caused Windows slaves to crash when they started replication if a master.info file existed. (Bug#1720)

D.4.2. Changes in release 3.23.58 (11 Sep 2003)

- Fixed buffer overflow in password handling which could potentially be exploited by MySQL users with ALTER privilege on the mysql.user table to execute random code or to gain shell access with the UID of the mysqld process (thanks to Jedi/Sector One for spotting and reporting this bug).
- mysqldump now correctly quotes all identifiers when communicating with the server. This assures that during the dump process, mysqldump will never send queries to the server that result in a syntax error. This problem is not related to the mysqldump program's output, which was not changed. (Bug#1148)
- Fixed table/column grant handling - proper sort order (from most specific to less specific, see 項4.3.10.「アクセス制御の段階 2: 要求確認」) was not honored. (Bug#928)
- Fixed overflow bug in MyISAM and ISAM when a row is updated in a table with a large number of columns and at least one BLOB/TEXT column.
- Fixed MySQL so that field length (in C API) for the second column in SHOW CREATE TABLE is always larger than the data length. The only known application that was affected by the old behavior was Borland dbExpress, which truncated the output from the command. (Bug#1064)
- Fixed ISAM bug in MAX() optimization.
- Fixed Unknown error when doing ORDER BY on reference table which was used with NULL value on NOT NULL column. (Bug#479)

D.4.3. Changes in release 3.23.57 (06 Jun 2003)

- Fixed problem in alarm handling that could cause problems when getting a packet that is too large.
- Fixed problem when installing MySQL as a service on Windows when one gave 2 arguments (option file group name and service name) to mysqld.
- Fixed kill pid-of-mysqld to work on Mac OS X.
- SHOW TABLE STATUS displayed incorrect Row_format value for tables that have been compressed with myisampack. (Bug#427)
- SHOW VARIABLES LIKE 'innodb_data_file_path' displayed only the name of the first datafile. (Bug#468)
- Fixed security problem where mysqld didn't allow one to UPDATE rows in a table even if one had a global UPDATE privilege and a database SELECT privilege.
- 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 optimizing WHERE clause with a constant expression such as in WHERE 1 AND (a=1 AND b=1).
- Fixed problem on IA-64 with timestamps that 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 項7.5.3.「InnoDB 起動オプション」.

- 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 overflow bug in MyISAM when a row is inserted into a table with a large number of columns and at least one BLOB/TEXT column. Bug was caused by incorrect calculation of the needed buffer to pack data.
- The binary log was not locked during TRUNCATE table_name or DELETE FROM table_name statements, which could cause an INSERT to table_name to be written to the log before the TRUNCATE or DELETE statements.
- Fixed rare bug in UPDATE of InnoDB tables where one row could be updated multiple times.
- Produce an error for empty table and column names.
- Changed PROCEDURE ANALYSE() to report DATE instead of NEWDATE.
- Changed PROCEDURE ANALYSE(#) to restrict the number of values in an ENUM column to # also for string values.
- mysqldump no longer silently deletes the binary logs when invoked with the --master-data or --first-slave option; while this behavior was convenient for some users, others may suffer from it. Now one has to explicitly ask for binary logs to be deleted by using the new --delete-master-logs option.
- Fixed a bug in mysqldump when it was invoked with the --master-data option: The CHANGE MASTER TO statements that were appended to the SQL dump had incorrect coordinates. (Bug#159)

D.4.4. 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-writeable 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 incorrect results from the query.
- Fixed a bug in SHOW VARIABLES on 64-bit platforms. The bug was caused by incorrect 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() optimization 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.4.5. 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 NULL values in CASE ... WHEN ....
- Bugfix for --chroot (see 項D.4.6. 「Changes in release 3.23.54 (05 Dec 2002)」) 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 default file system (HFS+) 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.4.6. 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).
- Make it possible to use 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 optimizing 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.4.7. 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 HPUX 11.0.
- Multi-threaded stress tests for InnoDB.

D.4.8. Changes in release 3.23.52 (14 Aug 2002)

- Wrap BEGIN/COMMIT around transaction in the binary log. This makes replication honor transactions.
- Fixed security bug when having an empty database name in the user.db table.
- Changed initialization 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 optimizer (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.4.9. 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 optimization 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 optimization 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.4.10. 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.4.11. 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.4.12. 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 optimizer 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 resynchronizing 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.4.13. 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.4.14. 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.4.15. 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.4.16. 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 col_name=NULL on an indexed column that had NULL values.
- Fixed bug when doing LEFT JOIN ... ON (col_name = constant) WHERE col_name = 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 optimizer.
- 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.4.17. 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.4.18. 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.4.19. Changes in release 3.23.41 (11 Aug 2001)

- Added --sql-mode=value[,value[,value]] option to mysqld. See 項4.1.1.「mysqld コマンドラインオプション」.
- 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 optimizing bug in ORDER BY where some ORDER BY parts where 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.4.20. 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 tbl_name ... LIMIT 0 and UPDATE FROM tbl_name ... 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.4.21. 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 tbl_name 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.4.22. 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 where 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 optimizer so that queries like SELECT * FROM tbl_name,tbl_name2 ... ORDER BY key_part1 LIMIT row_count 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.4.23. 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.
- Initialize 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.4.24. 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.4.25. 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.4.26. Changes in release 3.23.34a

- Added extra files to the distribution to allow INNOBASE support to be compiled.

D.4.27. 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() optimization 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.4.28. 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 tbl_name to TRUNCATE TABLE tbl_name to use the same syntax as Oracle. Until 4.0 we will also allow TRUNCATE tbl_name 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 tbl_name 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 behavior, Linux seems to "leak" memory, as it's being

allocated to the directory entry cache instead of the disk cache.

D.4.29. 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.4.30. 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.

- HOUR() (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 optimized 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.4.31. 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 where reported as having the name UNOPENED in error messages.
- Fixed bug when running two simultaneous SHOW LOGS queries.

D.4.32. 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 optimize 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 endpoints.
- 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 項4.8.3.「mysqld_multi（複数の MySQL サーバを管理するプログラム）」.
- 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 tbl_name 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 optimizer 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.4.33. 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 項4.9.2.「mysql（コマンドラインツール）」, 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 row_count 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 Glomsrød)
- 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 tbl_name.
- 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 uninitialized 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.4.34. 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.4.35. 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 Igonitchev)
- 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 optimization 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 the 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 optimization 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 initialized 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.4.36. 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 (as 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 項7.2.「MERGE テーブル」.
- 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-alloced 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.4.37. 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.4.38. 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.
- Optimizer 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 behavior 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 optimizer 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.4.39. 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 (IA-64) in gen_lex_hash.c.
- Fixed memory leak in the client library when using host= in the my.cnf file.
- Optimized 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 tbl_name 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.4.40. 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 optimize the DELETE FROM tbl_name 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.4.41. Changes in release 3.23.20

- Fixed bug in 3.23.19; DELETE FROM tbl_name removed the .frm file.
- Added SHOW CREATE TABLE.

D.4.42. 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. See http://www.fsf.org/licenses/.
- 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.4.43. 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.4.44. 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 duplicated 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.4.45. 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.4.46. 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.

- Optimized 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 optimization 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 項5.4.3.「MySQL でのインデックスの使用」.
- 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.4.47. 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.4.48. 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.4.49. 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.4.50. Changes in release 3.23.11

- Allow the ALTER TABLE tbl_name ADD (field_list) syntax.
- Fixed problem with optimizer 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.4.51. Changes in release 3.23.10

- Fixed bug in 3.23.9 where memory wasn't properly freed when using LOCK TABLES.

D.4.52. 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 optimizer 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 optimizer 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 optimizer to prefer a range key instead of a ref key when the range key can uses 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 optimization 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.4.53. 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.4.54. 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 項2.6.2.「Linux の注意事項（すべての Linux バージョン）」.
- 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.4.55. 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 <= 24 and a DOUBLE if 24 < X <= 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.4.56. 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.4.57. 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.4.58. 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 tbl_name 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.4.59. Changes in release 3.23.2 (09 Aug 1999)

- Fixed range optimizer 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 tbl_name 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 optimization 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 optimizer bug (when using BLOB values) 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 = col_name even if the columns are not identically packed.
- Indexes are now used for WHERE col_name 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.4.60. Changes in release 3.23.1

- Fixed some compilation problems.

D.4.61. Changes in release 3.23.0 (05 Aug 1999: Alpha)

- A new storage engine library (MyISAM) with a lot of new features. See 項7.1.「MyISAM テーブル」.
- 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 'e' 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 optimizer 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 項2.5.4.「バージョン 3.21 から 3.22 へのアップグレード」.

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 optimizer 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 optimizer 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 optimizer 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=file_name 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.
- MySQL on Windows is now compiled with VC++ 6.0 instead of with VC++ 5.0.
- New installation setup for MySQL on 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 tbl_name WHERE key_column=col_name 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 optimized 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' optimization 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 7.1.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 (single-row) 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 tbl_name SET col_name=value, col_name=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.
- 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 tbl_name 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 optimizer 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 analyzer 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 optimize 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 dbug 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().
- Initialize 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 tbl_name 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 optimization; some queries are now resolved using only indexes. Until MySQL 4.0, this works only for numeric columns. See 項5.4.3. 「MySQL でのインデックスの使用」.
- 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 optimization 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 optimizer.
- 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 tbl_name@db_name or db_name.tbl_name. 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 optimization of not_null_column IS NULL (needed for some Access queries).
- Allow STRAIGHT_JOIN to be used between two tables to force the optimizer 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-assembler 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 tbl_name 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 ont 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 where found.
- Fixed a bug in the range optimizer 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 tbl_name 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 optimizer (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 optimizer 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 e with an accent).
- 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 optimizer code when using many part keys on the same key: INDEX (Organization,Surname(35),Initials(35)).
- Added some tests to the table order optimizer 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 optimizer 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 optimization 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 recognize 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:SS') as argument.

- Fixed unlikely(?) key optimizer bug when using OR operators of key parts inside AND expressions.
- 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 for 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 behavior 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 optimizer 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 optimizer 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 col_name, SUM(expr) now returns NULL for col_name when there are matching rows.
- Fixed problem with comparing binary strings and BLOB values 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.
- Optimized 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 an SQL server. Tested with mSQL, PostgreSQL, Solid and MySQL.
- Fixed bug in range-optimizer 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 INT values 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 tbl_name ALTER COLUMN col_name 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 program 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 BLOBtypes, 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 datatype 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 optimizer. 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 SCO. See 項2.6.6.9.「SCO の注意事項」.

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 trigometric 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 optimization 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 optimizer (did only test = of the first argument).
- Added machine-dependent files for MIT-pthreads 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/os-freebsd.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:SS' 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 optimizer 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 tbl_name.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 tbl_name [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 optimizer 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:
  - ENUM

    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!
  - SET

    A string which may have one or many string values separated with ','. 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 optimizer that can resolve ranges when some keypart prefix is constant. Example:

```
mysql> SELECT * FROM tbl_name
    ->        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 optimizer 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 unoptimizable 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 YYMMDD.
- 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 optimizer 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 SELECT statements 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 gibc-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 BLOB values.
- 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, newline or '\' while writing tab-separated output. This is to allow printing of binary data in a portable format. To get the old behavior, 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 optimizer a little to get better results when searching on a key part.
- Added SELECT option STRAIGHT_JOIN to tell the optimizer 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 -ldbug -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 tbl_name 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 optimization 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 information 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.

付録 E. 他システムへの移植

この付録は、MySQL を他のオペレーティングシステムに移植する際に役立ちます。 現時点でサポートされているオペレーティングシステムの一覧を最初に確認してください。 See 項2.2.3.「MySQL がサポートしているオペレーティングシステム」。 MySQL の移植版を新たに作成した場合はお知らせください。このマニュアルと当社 Web サイト (http://www.mysql.com/) に新規移植版を掲載し、他のユーザに推奨させていただきます。

注意: 作成した MySQL 移植版は GPL ライセンスの下で自由に複製および配布することができますが、これによって MySQL の所有権が付与されるものではありません。

移植先のサーバには、有効な Posix スレッドライブラリが必要です。我々は、Solaris 2.5 については Sun の PThread (2.4 以前のバージョンにおけるネイティブスレッドのサポートは不十分でした)、Linux については Xavier Leroy (<Xavier.Leroy@inria.fr>) による LinuxThread を使用しています。

ネイティブスレッドに関する十分なサポートがない状態で新種の Unix に移植する際に難しいのは、MIT-pthread の移植であると考えられます。mit-pthreads/README と「Programming POSIX Threads」(http://www.humanfactor.com/pthreads/) を参照してください。

MySQL 4.0.2 まで、MySQL のディストリビューションには Chris Provenzano の MIT Pthread (MIT Pthread の Web ページ (http://www.mit.edu/afs/sipb/project/pthreads/) と プログラミング入門 (http://www.mit.edu:8001/people/proven/IAP_2000/)) パッチ適用バージョンが含まれていました。 これらは POSIX スレッドを持たない一部のオペレーティングシステムでも使用可能です。 See 項2.3.6.「MIT-pthreads に関する注意事項」。

別のユーザレベルスレッドパッケージである FSU Pthread (http://moss.csc.ncsu.edu/~mueller/pthreads/ を参照) を使用することもできます。 この実装は SCO への移植に使用されています。

これらの問題のテストや実例については、mysys ディレクトリの thr_lock.c および thr_alarm.c プログラムを参照してください。

サーバとクライアントのどちらにも C++ コンパイラが必要です。我々は、多くのプラットフォームで gcc を使用しています。上記以外で有効と確認されたコンパイラは、SPARCworks、Sun Forte、Irix cc、HP-UX aCC、IBM AIX xlC_r)、Intel ecc、Compaq cxx) です。

クライアントのみコンパイルするには、./configure --without-server を使用します。

サーバのみのコンパイルは、現時点ではサポートされておらず、格別の理由がない限り追加されないものと思われます。

Makefile または configure スクリプトを変更する必要がある場合、GNU Automake および Autoconf も必要です。 See 項2.3.4.「開発ソースツリーからのインストール」。

最も基本的なファイルからすべてを再作成するために必要な手順。

```
/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
```

移植に関して問題が起きた場合、MySQL をデバッグしなければならないことがあります。 See 項E.1.「MySQL サーバのデバッグ」。

注意: mysqld のデバッグを開始する前に、テストプログラム mysys/thr_alarm および mysys/thr_lock を起動してください。これによって、スレッドのインストールが実行される可能性も多少あります。

E.1. MySQL サーバのデバッグ

MySQL の新機能を使用している場合、以下を指定して mysqld の実行を試みることができます。--skip-new を指定すると、安全でない可能性のある新機能がすべて無効になります。--safe-mode を指定すると、問題を引き起こす可能性のある最適化の大部分が無効になります。 See 項A.4.1.「MySQL が何度もクラッシュする場合に行うこと」。

mysqld が起動しない場合、いずれかの my.cnf ファイルがセットアップの妨げとなっていないか確認する必要があります。 my.cnf 引数を mysqld --print-defaults でチェックし、これらの引数が使用されないように、mysqld --no-defaults ... で起動することができます。

mysqld がCPU またはメモリを消費し始めたか ``ハング'' した場合、mysqladmin processlist status を使用して、時間のかかるクエリを実行しているユーザを確認することができます。mysqladmin -i10 processlist status をウィンドウで実行すると有益なケースもあります。たとえば、新しいクライアントに接続できないときにパフォーマンスの問題が起きている場合です。

コマンド mysqladmin debug を実行すると、使用中のロック、メモリの使用容量、およびクエリの用途に関する情報が mysql ログファイルにダンプされ、問題解決につながることがあります。また、デバッグのために MySQL をコンパイルしていなくても、有益な情報を得られることもあります。

一部のテーブルの処理速度が低下する問題が起きた場合、OPTIMIZE TABLE または myisamchk でテーブルの最適化を試みてください。See 章 4. データベース管理。 また、遅いクエリを EXPLAIN でチェックする必要があります。

このマニュアルにおける OS 固有のセクションを参照し、使用環境に特有な問題について確認する必要もあります。 See 項2.6.「オペレーティングシステム固有の注意事項」。

E.1.1. MYSQL のコンパイル (デバッグ用)

極めて固有な問題が起きた場合は、いつでも MySQL のデバッグを試みることができます。デバッグするには、--with-debug または --with-debug=full オプションを指定して MySQL を設定する必要があります。 MySQL がコンパイルされたかどうかをチェックするには、mysqld --help を実行します。--debug オプションが表示されたらデバッグは有効です。この場合、mysqladmin ver を実行すると、mysqld バージョンが mysql ... --debug として表示されます。

gcc または egcs の使用時には、以下の configure 行が推奨されています。

```
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
```

これによって、libstdc++ ライブラリと C++ 例外に関する問題が回避され (多くのコンパイラでは、スレッドコードで C++ 例外が発生します)、すべてのキャラクタセットのサポート付きで MySQL がコンパイルされます。

メモリオーバーランエラーの疑いがある場合、--with-debug=full オプションを指定して MySQL を設定し、メモリ割当 (SAFEMALLOC) チェッカをインストールできます。ただし、SAFEMALLOC を指定して実行すると非常に遅くなるので、パフォーマンスの問題が生じたときには mysqld を --skip-safemalloc オプションで起動してください。これによって、malloc() と free() に対する呼び出しのたびにメモリオーバーランのチェックが無効になります。

mysqld が --with-debug 付きのコンパイル時にクラッシュしなくなった場合、コンパイラバグまたはタイミングバグが MySQL で検出されたと考えられます。この場合、上記の CFLAGS および CXXFLAGS 変数に -g を追加し、--with-debug を指定しない方法を試みることができます。mysqld がクラッシュした場合、少なくとも gdb を使用してアタッチするか、コアファイル上で gdb を実行して何が起きたか確認することは可能です。

デバッグ用に MySQL を設定すると、多くの特別なチェック関数が自動的に有効になり、これらによって mysqld の状態が監視されます。 ``予期しない" 状態が検出された場合、stderr にエントリが書き込まれ、mysqld_safe によってエラーログに送信されます。つまり、予期しない問題が MySQL で発生したときにソースディストリビューションを使用している場合、まず MySQL をデバッグ用に設定する必要があります。その後は MySQL メーリングリストにメールを送信し、ヘルプを依頼してください。 See 項1.7.1.1.「MySQL メーリングリスト」。 ご利用の MySQL バージョンに関するバグレポートまたは質問については、mysqlbug を使用してください。

Windows 版 MySQL ディストリビューションでは、mysqld.exe がデフォルトでコンパイルされています (トレースファイルのサポート付き) 。

E.1.2. トレースファイルの作成

mysqld サーバが起動しないか mysqld サーバですぐにクラッシュが起きる場合、トレースファイルを作成して問題の検出を試みることができます。

そのためには、mysqld をデバッグ用にコンパイルしておく必要があります。そのようにコンパイルされているか確認するには、mysqld -V を実行します。バージョン番号の末尾が -debug である場合、トレースファイルのサポート付きでコンパイルされています。

/tmp/mysqld.trace (Windows では C:\mysqld.trace) 内のトレースログを指定して mysqld サーバを起動します。

mysqld --debug

Windows では、--standalone フラグを使用し、mysqld がサービスとして起動されないようにする必要もあります。

コンソールウィンドウで以下を実行します。

```
mysqld --debug --standalone
```

この後、mysql.exe コマンドラインツールを 2 番目のコンソールウィンドウで実行し、問題を再現することができます。mysqladmin shutdown では、mysqld サーバを強制終了することができます。

注意: トレースファイルは非常に大きなサイズになります。 トレースファイルのサイズを小さくしたい場合、たとえば以下のように指定することができます。

mysqld --debug=d,info,error,query,general,where:O,/tmp/mysqld.trace

(この場合、最も興味深いタグの付いた情報のみが /tmp/mysqld.trace に出力されます。)

これに関するバグレポートを作成する場合、不具合があると思われる行のみをトレースファイルから選択し、適切なメーリングリスト宛てに送信してください。不具合の場所を特定できない場合には、トレースファイルと完全なバグレポート

を ftp://support.mysql.com/pub/mysql/secret/ まで ftp で送信してください。MySQL の開発者側で確認させていただきます。

トレースファイルは、Fred Fish による DBUG パッケージで作成されています。 See 項E.3.「DBUG パッケージ」。

E.1.3. mysqld のデバッグ（gdb 使用）

多くのシステムでは、mysqld がクラッシュした際に、mysqld を gdb から起動することで、より多くの情報を取得することもできます。

Linux で使用される旧バージョンの gdb の一部では、mysqld スレッドをデバッグする際に run --one-thread を使用しなければなりません。この場合、一度にアクティブにできるスレッドは 1 つだけです。速やかに gdb 5.1 にアップグレードすることをお奨めします。このバージョンでは、スレッドデバッグの処理が大幅に改善されるからです。

gdb で mysqld を実行する場合、スタックトレースを --skip-stack-trace で無効化し、gdb 内部で segfault を捕捉できるようにする必要があります。

MySQL 4.0.14 以降では、mysqld に対して --gdb オプションを使用してください。このオプションを使用すると、SIGINT（mysqld を ^C で終了し、ブレークポイントを設定するために必要）の割り込みハンドラがインストールされ、スタックトレースとコアファイル処理が無効になります。

gdb で MySQL をデバッグするのが非常に困難な場合があります。たとえば、古いスレッドに対するメモリが gdb によって解放されず、デバッグ中に大量の新規接続を連続して確立しなければならない場合です。この問題を回避するには、-O thread_cache_size= 'max_connections +1' を指定して mysqld を起動します。多くの場合、-O thread_cache_size=5' を指定するだけで状況は大幅に改善します。

SIGSEGV シグナルで mysqld がクラッシュした場合、Linux 上でコアダンプを取得するには、--core-file オプションを指定して mysqld を起動します。 このコアファイルを使用してバックトレースを作成し、mysqld がクラッシュした理由を特定することもできます。

```
shell> gdb mysqld core
gdb>   backtrace full
gdb>   exit
```

See 項A.4.1.「MySQL が何度もクラッシュする場合に行うこと」。

gdb 4.17.x 以降を Linux で使用している場合、.gdb ファイルを以下の情報と共にカレントディレクトリにインストールする必要があります。

```
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
```

gdb でスレッドのデバッグに問題が起きた場合、gdb 5.x をダウンロードして代用してみてください。gdb の新規バージョンでは、スレッドの処理が飛躍的に改善されています。

以下の例は、mysqld をデバッグする方法を示します。

```
shell> gdb /usr/local/libexec/mysqld
gdb> run
...
backtrace full # Do this when mysqld crashes
```

mysqlbug で生成された上記の出力内容をメールに取り込み、通常の MySQL メーリングリスト宛てに送信してください。 See 項1.7.1.1.「MySQL メーリングリスト」。

mysqld がハングした場合、strace や /usr/proc/bin/pstack のようなシステムツールを使用し、mysqld がハングした場所を調べることができます。

```
strace /tmp/log libexec/mysqld
```

Perl DBI インタフェースを使用している場合、デバッグ情報を有効にするには、trace メソッドを使用するか DBI_TRACE 環境変数を設定します。 See 項11.5.2.「DBI インタフェース」。

E.1.4. スタックトレースの使用

一部のオペレーティングシステムでは、mysqld が突然クラッシュするとスタックトレースがエラーログに記録されます。この情報を使用すると、mysqld がクラッシュした場所（および理由）を確認することができます。See 項4.10.1.「エラーログ」。 スタックトレースを取得する場合、mysqld のコンパイルを -fomit-frame-pointer オプション付き gcc で行わないでください。 See 項E.1.1.「MYSQL のコンパイル（デバッグ用）」。

エラーファイルの内容が以下のとおりであるとします。

```
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
Attemping 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
```

mysqld がクラッシュした場所を特定するには、以下を実行します。

1. 上記の番号をファイル（たとえば mysqld.stack）にコピーする。

2. mysqld サーバのシンボルファイルを作成する。

```
nm -n libexec/mysqld > /tmp/mysqld.sym
```

注意: 多くの MySQL バイナリディストリビューション (この情報がバイナリ自体の内部に含まれている "デバッグ" パッケージは除外) には、上記のファイルが mysqld.sym.gz という名称で同梱されています。 ここでは、このファイルを以下のように簡単に解凍できます。

```
gunzip < bin/mysqld.sym.gz > /tmp/mysqld.sym
```

3. resolve_stack_dump -s /tmp/mysqld.sym -n mysqld.stack を実行する。

実行すると、mysqld がクラッシュした場所が出力されます。mysqld がクラッシュした原因をこの方法で特定できない場合には、バグレポートを作成し、上記コマンドの出力結果をバグレポートに含める必要があります。

ただし、多くの場合、スタックトレースだけでは問題の理由を特定できません。バグを見つけるか次善策を講じるには、多くの場合、mysqld のクラッシュを引き起こしたクエリのほか、可能であれば問題を再現できるテストテースについても知る必要があります。 See 項1.7.1.3.「バグまたは問題を報告する方法」。

E.1.5. mysqld におけるエラーの原因をログファイルを使用して特定する

注意: --log を指定して mysqld を起動する前に、すべてのテーブルを myisamchk でチェックしてください。 See 章 4. データベース管理。

mysqld がクラッシュまたはハングした場合、--log を指定して mysqld を起動する必要があります。mysqld が再度クラッシュした場合、ログファイルの末尾を参照し、mysqld のクラッシュを引き起こしたクエリの有無を確認できます。

ファイル名なしで --log を使用している場合、ログは 'hostname'.log としてデータベースディレクトリに格納されます。多くの場合、ログファイル内の最後のクエリが原因で mysqld がクラッシュしています。ただし、実際にそうであったかどうかを確認するため、mysqld を再起動し、検出されたクエリを mysql コマンドラインツールから実行する必要があります (可能な場合)。この方法が有効な場合、完了しなかった複雑なクエリをすべてテストする必要もあります。

時間がかかる SELECT ステートメントのすべてに対してコマンド EXPLAIN を実行し、インデックスが mysqld によって適切に使用されているかどうかを確認することもできます。 See 項5.2.1.「EXPLAIN 構文 (SELECT に関する情報の取得)」。

実行時間の長いクエリを検索するには、--log-slow-queries を指定して mysqld を実行します。 See 項4.10.5.「スロークエリログ」。

テキスト mysqld restarted がエラーログファイル (通常は hostname.err という名称) に含まれている場合、mysqld のクラッシュを引き起こしたクエリが検出されたと考えられます。この場合、myisamchk (see 章 4. データベース管理) でテーブルをすべてチェックします。また、MySQL ログファイル内のクエリをテストし、いずれかが無効かどうかを確認します。そのようなクエリが見つかった場合、最新バージョンの MySQL へのアップグレードを試みてください。アップグレードしても問題が解決せず、mysql メールアーカイブにも有益な情報がない場合、MySQL メーリングリストにバグをレポートする必要があります。 このメーリングリストが掲載されている http://lists.mysql.com/ には、オンラインリストアーカイブもリンクされています。

myisam-recover を指定して mysqld を起動した際、MyISAM テーブルが '正しく閉じられなかった' または 'クラッシュした' とマークされている場合、そのテーブルは自動的にチェックされ、修復が試行されます。この場合、hostname.err ファイ

ルに 'Warning: Checking table ...' と書き込まれます。また、テーブルの修復が必要であれば、Warning: Repairing table が続いて書き込まれます。これらのエラーが大量に発生した際、直前に mysqld が突然クラッシュしていない場合は異常であり、さらに調査しなければなりません。 See 項4.1.1.「mysqld コマンドラインオプション」。

mysqld が突然クラッシュするのは、もちろん正常ではありません。ただし、その場合には、Checking table... メッセージを調べるのではなく、mysqld がクラッシュした原因を探してください。

E.1.6. テーブルが破損した場合にテストケースを作成する

テーブルが破損しているか、更新コマンドの後に mysqld が必ずエラーになる場合、この問題が再現されるかテストするには以下を実行します。

- MySQL デーモンを強制終了する（ mysqladmin shutdown を使用 ）。
- テーブルのバックアップを作成する（ 修復による悪影響が生じた場合に備えるため ）。
- すべてのテーブルを myisamchk -s database/*.MYI でチェックする。問題の起きたテーブルがあれば myisamchk -r database/table.MYI で修復する。
- テーブルの 2 次バックアップを作成する。
- 空き容量を増やす必要がある場合、古いログファイルを MySQL データディレクトリから削除または移動する。
- mysqld を --log-bin で起動する。 See 項4.10.4.「バイナリログ」。 mysqld のクラッシュを引き起こすクエリを検索する場合、--log --log-bin を使用する。
- テーブルがクラッシュした場合、mysqld server を終了する。
- バックアップをリストアする。
- --log-bin を指定しないで mysqld サーバを起動する。
- mysqlbinlog update-log-file | mysql を指定してコマンドを再実行する。 更新ログは、hostname-bin.# という名称で MySQL データベースディレクトリに保存される。
- テーブルが再度破損したか、上記のコマンドで mysqld が終了する場合、再現可能なバグが検出されている。そのようなバグは、容易に修正することが可能。テーブルとバイナリログを ftp://support.mysql.com/pub/mysql/secret/ FTP で送信し、当社のバグシステム（ http://bugs.mysql.com/ ）に入力する。 サポートユーザの場合、<support@mysql.com> で MySQL チームに問題を通知し、迅速な修正を依頼することもできる。

スクリプト mysql_find_rows で一部の更新ステートメントのみを実行し、問題を絞り込むこともできます。

E.2. MySQL クライアントのデバッグ

MySQL クライアントを統合デバッグパッケージによってデバッグするには、--with-debug または --with-debug=full を指定して MySQL を設定する必要があります。 See 項2.3.3.「一般的な configure オプション」。

クライアントを実行する前に、MYSQL_DEBUG 環境変数を指定してください。

```
shell> MYSQL_DEBUG=d:t:O,/tmp/client.trace
shell> export MYSQL_DEBUG
```

その結果、クライアントによってトレースファイルが /tmp/client.trace に生成されます。

ユーザ自身のクライアントコードで問題が起きた場合、サーバに接続し、有効と確認されているクライアントでクエリを実行してみてください。そのためには、mysql をデバッグモードで実行します (デバッグを有効にして MySQL をコンパイルしてあることが前提) 。

```
shell> mysql --debug=d:t:O,/tmp/client.trace
```

バグレポートをメール送信する場合、上記によって有益な情報を得ることができます。 See 項1.7.1.3.「バグまたは問題を報告する方法」。

'有効と思われる' コードでクライアントがクラッシュした場合、mysql.h インクルードファイルが mysql ライブラリファイルと一致するかどうかを確認する必要があります。 一般的な誤りとしては、古いバージョンの MySQL の mysql.h ファイルを新しい MySQL ライブラリと共に使用することがあります。

E.3. DBUG パッケージ

MySQL サーバと多くの MySQL クライアントは、DBUG パッケージ (オリジナル開発者は Fred Fish) によってコンパイルされています。デバッグ用に MySQL を設定した場合、デバッグされている内容に関するトレースファイルを取得できます。 See 項E.1.2.「トレースファイルの作成」。

デバッグパッケージを使用するには、--debug="..." または -#... オプションを指定してプログラムを起動します。

多くの MySQL プログラムにはデフォルトのデバッグ文字列があり、--debug に対してオプションを指定しなかった場合に使用されます。デフォルトのトレースファイルは通常、Unix では /tmp/programname.trace、Windows では \programname.trace です。

デバッグコントロール文字列は、以下のようにコロンで区切られた一連のフィールドです。

```
<field_1>:<field_2>:...:<field_N>
```

各フィールドの構成要素は、フラグ文字列 (必須) と、後続する "," およびコンマ区切り修飾子一覧 (任意) です。

```
flag[,modifier,modifier,...,modifier]
```

現時点で認識されるフラグ文字は以下のとおりです。

フラグ	説明
d	現在の状態に対して DBUG_<N> マクロからの出力を有効にする。キーワードを伴う DBUG マクロに関する出力のみが選択されるキーワード一覧が後続する場合がある。キーワード一覧が空白になっている場合、すべてのマクロに関する出力であることを示す。
D	各デバッガ出力行の後の遅延。引数は遅延する時間 (1/10 秒単位の数) 。マシンの性能によって異なる。つまり、-#D,20 は 2 秒遅延することを示す。
f	デバッグとトレースの一方または両方を制限し、指定された関数の一覧にプロファイルを設定する。注意: 一覧が空白になっている場合、すべての関数が無効になる。適切な "d" または "t" フラグの指定が依然として必要。このフラグによって関数の動作が制限されるのは、関数が有効になっている場合に限られる。

F	デバッグまたはトレース出力の各行に対応したソースファイル名を識別する。
i	デバッグまたはトレース出力の各行に対応した PID またはスレッド ID を持つプロセスを識別する。
g	プロファイルを有効にする。プログラムのプロファイルに使用できる情報を含むファイル 'dbugmon.out' を作成する。一覧の関数のみに対してプロファイルを選択するキーワードの一覧が後続する場合がある。一覧が空白になっている場合、すべての関数が考慮される。
F	デバッグまたはトレース出力の各行に対応したソースファイル行番号を識別する。
n	デバッグまたはトレース出力の各行に対応した現在の関数のネスト深度を印刷する。
N	dbug 出力の各行に番号を設定する。
o	デバッガ出力ストリームを指定ファイルにリダイレクトする。デフォルト出力は stderr である。
O	o と同様。ただし、書き込みのたびにファイルが実際にフラッシュされる。必要に応じて、ファイルは閉じられ、各書き込みの間に再度開かれる。
p	デバッガの操作を指定プロセスに制限する。プロセスは、DBUG_PROCESS マクロで識別され、実行されるデバッガ操作の一覧に含まれるものと一致しなければならない。
P	デバッグまたはトレース出力の各行に対応した現在のプロセス名を印刷する。
r	新しいステータスに移行する際に、以前のステータスの関数ネストレベルを継承しない。.出力の開始位置が左マージンである場合に有効。
S	0 以外の値が _sanity() から返されるまで、デバッグ対象の各関数で関数 _sanity(_file_,_line_) を実行する（メモリリークを検出するために safemalloc と共に使用されることが多い）。
t	関数呼び出しの有効化/トレース行の終了。最大トレースレベルを数値で指定する一覧（修飾子が 1 つだけ含まれている）が後続する場合がある。このレベルを超えると、デバッグマクロとトレースマクロのいずれに対しても出力は実行されない。デフォルトはコンパイル時間オプションである。

シェルコマンドラインに表示されるデバッグコントロール文字列（"-#" の一般的な用途は、アプリケーションプログラムに対するコントロール文字列の導入）の例を以下に示します。

```
-#d:t
-#d:f,main,subr1:F:L:t,20
-#d,input,output,files:n
-#d:t:i:O,\\mysqld.trace
```

MySQL で出力するための一般的なタグ（d オプション付き）は、enter、exit、error、warning、info、loop です。

E.4. ロック方法

ISAM/MyISAM および HEAP テーブルのテーブルロック、BDB テーブルのページレベルロック、InnoDB テーブルの行レベルロックのみが、MySQL で現在サポートされています。 See 項5.3.1.「MySQL のテーブルロック方法」。 INSERT ステートメント間に競合がない場合（レコードまたはデータの削除による空き領域を埋めるのではなく、テーブルファイル末尾に追加する場合はいつでも）、MyISAM テーブルでは INSERT と SELECT をロックなしで自由に組み合わせることができます。

バージョン 3.23.33 からは、システムにおけるテーブルロックの競合を Table_locks_waited および Table_locks_immediate 環境変数を確認して分析できるようになりました。

テーブル型を行レベルロックと共に使用するかどうかを決定するには、アプリケーションの処理内容とデータの選択/更新パターンを確認する必要があります。

行ロックの利点:

- 多数のスレッド内の異なったレコードにアクセスする際にロックの競合が少ない。
- ロールバックの変更が少ない。
- 単一レコードを長時間ロックすることができる。

行ロックの欠点:

- ページレベルロックまたはテーブルロックよりも多くのメモリを消費する。
- テーブルの広範囲で実行する場合、多数のロックが必要になるため、ページレベルロックまたはテーブルロックよりも処理速度が低下する。
- データの大部分で GROUP BY を頻繁に実行するか、テーブル全体を頻繁にスキャンする必要がある場合、他のロックよりも明らかに効率が悪化する。
- 高いロックレベルを使用すると、タイプの異なるロックをサポートし、行レベルロックの場合と同様にロックオーバーヘッドが減少するようにアプリケーションを調整するのも容易である。

以下の場合、ページレベル/行レベルロックよりもテーブルロックの方が適しています。

- 大部分が読み込み
- 厳密なキーでの読み込みおよび更新。この場合、1 つのキー読み込みで取得できる 1 つのレコードを更新または削除する。

```
UPDATE table_name SET column=value WHERE unique_key#
DELETE FROM table_name WHERE unique_key=#
```

- SELECT と INSERT（ならびに少数の UPDATE および DELETE ステートメント）を組み合わせる。
- 多数のスキャンまたは GROUP BY をテーブル全体で実行する（書き込みなし）。

行レベル/ページレベルロック以外の他のオプション:

バージョン管理（MySQL では並列 INSERT に使用）。多数の読み取りと同時に単一の書き込みが可能です。つまり、データベース/テーブルでは、ユーザがアクセスを開始した時期に基づき、異なるビューが同じデータに関してサポートされます。これはタイムトラベルにたとえることも、書き込み時コピー、コピーオンデマンドと呼ぶこともできます。

多くの場合、コピーオンデマンドは、ページレベルまたは行レベルのロックよりもはるかに優れています。ただし、最悪の場合、通常のロックよりもはるかに大量のメモリを消費します。

行レベルロックの代わりに、アプリケーションレベルロック（MySQL での get_lock/release_lock と同様）を使用すること

ができます。言うまでもなく、このロックが役立つのは正常に動作するアプリケーションに限られます。

多くの場合、アプリケーションに最適なロックタイプは経験に基づき推測できます。しかし、指定されたロックタイプが別のロックタイプよりも好ましいと判断するのは一般的に極めて困難です。すべてがアプリケーションに依存しており、アプリケーションにおける異なった部分には異なったロックタイプが必要とされるからです。

MySQL でのロックに関するヒントを以下に示します。

多くの Web アプリケーションでは、多くの選択、一部の削除、更新が主としてキーで実行され、挿入が特定のテーブルに対して実行されます。 MySQL の基本設定は、上記に適した状態になっています。

同一テーブル内のレコードを大量に確認する必要のある更新と選択を併用しない場合、同時ユーザは問題になりません。

同一テーブルで挿入と削除を併用する場合、INSERT DELAYED が役立つことがあります。

また、LOCK TABLES を使用して処理速度を向上させることもできます (単一ロックでの多数の更新は、ロックなしの更新よりもはるかに高速です) 。複数のテーブルに分割するのも有効な対策です。

MySQL でのテーブルロックに処理速度の問題がある場合、テーブルの一部をInnoDB または BDB テーブルに変換して解決できることもあります。 See 項7.5.「InnoDB テーブル」。 See 項7.6.「BDB または BerkeleyDB テーブル」。

このマニュアルの最適化に関するセクションでは、アプリケーションの調整方法の多種多様な局面について取り上げています。 See 項5.2.13.「その他の最適化のヒント」。

E.5. RTS スレッドに関するコメント

RTS スレッドパッケージと MySQL の併用を試みたところ、以下の問題が発生しました。

これらのパッケージでは、多数の POSIX 呼び出しが使用されるほか、すべての関数のラッパを作成するのに非常に時間がかかります。そこで、スレッドライブラリを最新の POSIX 仕様に変更した方が簡単なのではないかと考えるようになりました。

一部のラッパは作成済みです。詳細については、mysys/my_pthread.c を参照してください。

少なくとも以下を変更する必要があります。

pthread_get_specific では 1 つの引数が使用されます。 sigwait では 2 つの引数が使用されます。 多くの関数 (少なくとも pthread_cond_wait、pthread_cond_timedwait) からエラー時にエラーコードが返される必要があります。ここでは -1 が返され、errno が設定されます。

もう 1 つの問題は、ユーザレベルのスレッドによって ALRM シグナルが使用され、そのために多くの関数 (read、write、open...) が中断されることです。 MySQL では、中断されたすべての関数における割り込みに対して再試行する必要がありますが、確認するのは容易ではありません。

解決されていない最大の問題は以下のとおりです。

スレッドレベルのアラームを取得するため、mysys/thr_alarm.c を変更し、pthread_cond_timedwait() でアラームとアラームの間に待機しましたが、エラー EINTR で中断されました。原因を調べるためにスレッドライブラリのデバッグを試みたものの、簡単な解決策は見つかりませんでした。

MySQL と RTS スレッドの併用を試みる場合、以下を行うことをお奨めします。

- MySQL で使用される関数をスレッドライブラリから POSIX に変更する。 変更するのに時間はかからない。
- -DHAVE_rts_threads ですべてのライブラリをコンパイルする。
- thr_alarm をコンパイルする。
- 実装に若干の差異がある場合、my_pthread.h と my_pthread.c の変更により修正することができる。
- thr_alarm を実行する。実行時に ``警告''、``エラー''、中断メッセージのいずれも表示されなかった場合、正常に実行されています。Solaris で正常に実行された場合の例を以下に示します。

```
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. 異なったスレッドパッケージ間の差異

MySQL は、使用されるスレッドパッケージによって大きく左右されます。そのため、MySQL に適したプラットフォーム

を選択する場合、スレッドパッケージは極めて重要です。

スレッドパッケージには、少なくとも以下の 3 種類があります。

- 単一プロセス内のユーザスレッド。スレッドの切り替えはアラームによって管理され、スレッドライブラリがすべての非スレッドセーフ関数をロックによって管理する。実行中のスレッドがデータを待機しなければならない場合、読み取り、書き込み、選択などの操作は通常、別のスレッドに切り替えるスレッド固有の select で管理される。ユーザスレッドパッケージが標準ライブラリ (古いバージョンの FreeBSD および BSDI スレッド) に統合されている場合、そのパッケージに必要なオーバーヘッドは、すべての安全でない呼び出し (MIT-pthread、FSU Pthread、RTS thread) にマッピングしなければならないスレッドパッケージよりも少なくて済む。一部の環境 (たとえば SCO) では、すべてのシステムコールがスレッドセーフなので、マッピングは非常に容易である (SCO における FSU Pthread)。欠点：マッピングされたすべての呼び出しに必要な時間はごくわずかであり、あらゆる状況に対処するのが極めて難しい。通常、スレッドパッケージによって処理されないシステムコールもいくつか存在する (MIT-pthread、ソケットなど)。スレッドのスケジュールは常に最適とは限らない。

- 個別プロセス内のユーザスレッド。スレッド切り替えはカーネルによって行われ、すべてのデータはスレッド間で共有される。スレッドパッケージは、スレッド間でデータが共有されるように標準スレッド呼び出しを管理する。この方法は LinuxThread で使用されている。欠点：プロセスが大量。スレッド作成が遅い。1 つのスレッドがクラッシュした場合、通常は残りのスレッドがハングしているので、それらをすべて強制終了した後に再起動しなければならない。スレッド切り替えには若干のコストがかかる。

- カーネルスレッド。スレッド切り替えは、スレッドライブラリまたはカーネルによって行われ、極めて高速である。すべて 1 つのプロセスで処理されるが、一部のシステムでは ps によって異なったスレッドが表示されることがある。1 つのスレッドが中止されると、プロセス全体が中止される。多くのシステムはスレッドセーフであり、必要とされるオーバーヘッドはごくわずかである。Solaris、HP-UX、AIX、OSF/1 にはカーネルスレッドがある。

一部のシステムでは、システムライブラリ内のユーザレベルスレッドによってカーネルスレッドが管理される。そのような場合、スレッド切り替えはスレッドライブラリによってのみ可能であり、カーネルは実際には ``スレッド対応" ではない。

付録 F. 環境変数

MySQL で直接または間接的に使用されるすべての環境変数の一覧を以下に示します。これらの変数の多くは、このマニュアルの別の個所にも記載されています。

注意: コマンドラインのオプションはオプション設定ファイルと環境変数で指定された値に優先し、オプション設定ファイル内の値は環境変数で指定された値に優先します。

MySQL の動作を修正する場合、環境変数ではなくオプション設定ファイルを使用した方が好ましいことがよくあります。See 項4.1.2.「my.cnf オプション設定ファイル」。

変数	説明
CXX	configure 実行時に C++ コンパイラに設定する。
CC	configure 実行時に C コンパイラに設定する。
CFLAGS	configure 実行時に使用する C コンパイラのフラグ。
CXXFLAGS	configure 実行時に使用する C++ コンパイラのフラグ。
DBI_USER	Perl DBI のデフォルトユーザ名。
DBI_TRACE	Perl DBI のトレース時に使用。
HOME	mysql ヒストリファイルのデフォルトパスは $HOME/.mysql_history である。
LD_RUN_PATH	libmysqlclient.so の場所を指定するために使用。
MYSQL_DEBUG	デバッグ時のデバッグ-トレースオプション。
MYSQL_HISTFILE	mysql ヒストリファイルのパス。
MYSQL_HOST	mysql コマンドラインクライアントによって使用されるデフォルトホスト名。
MYSQL_PS1	mysql コマンドラインクライアントで使用するコマンドプロンプト。 See 項4.9.2.「mysql（コマンドラインツール）」。
MYSQL_PWD	mysqld 接続時のデフォルトパスワード。使用するとセキュリティが確保されないことに注意する。
MYSQL_TCP_PORT	デフォルト TCP/IP ポート。
MYSQL_UNIX_PORT	デフォルトソケット。localhost への接続に使用。
PATH	MySQL プログラムを検索するためにシェルで使用。
TMPDIR	テンポラリテーブルまたはテンポラリファイルが作成されるディレクトリ。
TZ	ローカルタイムゾーンに設定される。 See 項A.4.6.「タイムゾーンの問題」。
UMASK_DIR	ディレクトリ作成時のユーザディレクトリ作成マスク。UMASK との AND 演算が実行されることに注意する。
UMASK	ファイル作成時のユーザファイル作成マスク。
USER	mysqld への接続時に使用する、Windows 上のデフォルトユーザ。

付録 G. MySQL の正規表現

正規表現（regex）は、複雑な検索を指定するための有効な方法の 1 つです。

MySQL では、Henry Spencer の正規表現が実装されていますが、その目的は POSIX 1003.2 に適合することです。MySQL では、拡張バージョンが使用されています。

ここでは簡単な説明にとどめ、詳細については省略しています。詳細な情報を入手するには、ソースディストリビューションに含まれている Henry Spencer の regex(7) マニュアルページを参照してください。 See 付録 C. 協力者。

正規表現では、一連の文字列を記述します。最も単純な regexp は、特殊文字を含まない形式です。たとえば、regexp hello は hello のみにマッチします。

単純でない形式の正規表現は、特殊な構成要素を含んでおり、複数の文字列にマッチします。たとえば、regexp hello|word は文字列 hello または文字列 word にマッチします。

さらに複雑な例は regexp B[an]*s です。文字列 Bananas、Baaaaas、Bs のいずれにもマッチするほか、これら以外で先頭が B、末尾が s になっており、かつ、先頭と末尾の間に a または n が含まれている（文字数は問わない）文字列のすべてにマッチします。

正規表現では、以下の特殊な文字または構成要素を使用することができます。

- ^

  文字列の先頭にマッチ。

```
mysql> SELECT "fo\nfo" REGEXP "^fo$";         -> 0
mysql> SELECT "fofo" REGEXP "^fo";            -> 1
```

- $

  文字列の末尾にマッチ。

```
mysql> SELECT "fo\no" REGEXP "^fo\no$";       -> 1
mysql> SELECT "fo\no" REGEXP "^fo$";          -> 0
```

- .

  あらゆる文字（改行も含む）にマッチ。

```
mysql> SELECT "fofo" REGEXP "^f.*";           -> 1
mysql> SELECT "fo\nfo" REGEXP "^f.*";         -> 1
```

- a*

  連続する 0 文字以上の a にマッチ。

```
mysql> SELECT "Ban" REGEXP "^Ba*n";           -> 1
mysql> SELECT "Baaan" REGEXP "^Ba*n";         -> 1
mysql> SELECT "Bn" REGEXP "^Ba*n";            -> 1
```

- a+

  連続する 1 文字以上の a にマッチ。

```
mysql> SELECT "Ban" REGEXP "^Ba+n";           -> 1
mysql> SELECT "Bn" REGEXP "^Ba+n";            -> 0
```

- a?

  0 文字または 1 文字の a にマッチ。

```
mysql> SELECT "Bn" REGEXP "^Ba?n";            -> 1
mysql> SELECT "Ban" REGEXP "^Ba?n";           -> 1
mysql> SELECT "Baan" REGEXP "^Ba?n";          -> 0
```

- de|abc

  連続する de または abc にマッチ。

```
mysql> SELECT "pi" REGEXP "pi|apa";           -> 1
mysql> SELECT "axe" REGEXP "pi|apa";          -> 0
mysql> SELECT "apa" REGEXP "pi|apa";          -> 1
mysql> SELECT "apa" REGEXP "^(pi|apa)$";      -> 1
mysql> SELECT "pi" REGEXP "^(pi|apa)$";       -> 1
mysql> SELECT "pix" REGEXP "^(pi|apa)$";      -> 0
```

- (abc)*

  文字列 abc の 0 回以上の繰り返しにマッチ。

```
mysql> SELECT "pi" REGEXP "^(pi)*$";          -> 1
mysql> SELECT "pip" REGEXP "^(pi)*$";         -> 0
mysql> SELECT "pipi" REGEXP "^(pi)*$";        -> 1
```

- {1} , {2,3}

  直前のパターンが多数出現する場合、それらにマッチする regexp をさらに一般的な方式で記述することができます。

  - a*

    a{0,} に書き換えることが可能。

  - a+

    a{1,} に書き換えることが可能。

  - a?

    a{0,1} に書き換えることが可能。

  より正確に表現すると、パターンの直後のかっこ内に整数 i が 1 つあってカンマがない場合、パターンに適合する i 回のシーケンスにマッチします。パターンの直後のかっこ内に整数 i とカンマが 1 つずつある場合、パターンに適合する i 回以上のシーケンスにマッチします。 パターンの直後のかっこ内に 2 つの整数 (i と j) がある場合、パターンに適合する i 〜 j 回のシーケンスにマッチします。

2 つの引数は、0 から RE_DUP_MAX (デフォルトは 255) までの範囲に含まれている必要があります。引数が 2 つある場合、2 つ目は 1 つ目以上でなければなりません。

- [a-dX] , [^a-dX]

  a、b、c、d、X のいずれかに該当する (^ が使用されている場合、これらのいずれにも該当しない) 文字にマッチします。] をリテラルとして含めるには、[の直後に記述する必要があります。- をリテラルとして含めるには、最初または最後に記述する必要があります。したがって、[0-9] は任意の 10 進数値にマッチします。意味が定義されていない文字が [] 内に存在する場合、その文字は特別な意味を持たず、自分自身にのみマッチします。

```
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.]]

  照合要素の文字シーケンス。シーケンスは、かっこ内のリストに含まれる単一の要素です。かっこ表現に複数文字の照合要素が含まれている場合、2 文字以上にマッチします。たとえば、照合シーケンスに照合要素 ch が含まれていると、正規表現 [[.ch.]]*c は chchcc の最初の 5 文字にマッチします。

- [=character_class=]

  等価クラスであり、それ自身を含め、それと等価なすべての照合要素の文字列を意味します。

  たとえば、o と (+) が等価クラスのメンバである場合、[[=o=]]、[[=(+)=]]、[o(+)] はすべて同義語です。等価クラスを範囲の終了位置にすることはできません。

- [:character_class:]

  かっこ表現において [: と :] で囲まれた文字クラス名は、そのクラスに属するすべての文字のリストを意味します。標準文字クラス名は以下のとおりです。

名称	名称	名称
alnum	digit	punct
alpha	graph	space
blank	lower	upper
cntrl	print	xdigit

  これらは、ctype(3) マニュアルページで定義された文字クラスを意味します。ロケールにより、他の文字クラスが提供されることもあります。文字クラスを範囲のエンドポイントにすることはできません。

```
mysql> SELECT "justalnums" REGEXP "[[:alnum:]]+";      -> 1
mysql> SELECT "!!" REGEXP "[[:alnum:]]+";              -> 0
```

- [[:<:]] , [[:>:]]

それぞれ単語の始めと終わりの NULL 文字列にマッチします。単語文字シーケンスのうち、前後に別の単語文字が存在しないものが単語と定義されています。単語文字は alnum 文字 (ctype(3)で定義) またはアンダースコア (_) です。

```
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
```

付録 H. GNU General Public License

Version 2, June 1991

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.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. 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.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.

7. 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.

8. 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.

9. 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.

10. 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.

11. 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

12. 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.

13. 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., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 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.

目次

シンボル

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y