Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

MySQL 5.5参考手册

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

This translation was done by MySQL partner GreatLinux, Beijing, People's Republic of China. GreatLinux Inc. 北京万里开源软件有限公司在全国范围提供MySQL产品相关的商务及 技术方面的咨询与支持服务。垂询请致电：8610-65694500，或发送邮件至：sales@greatlinux.com.

摘要

这是MySQL参考手册 它涉及MySQL 5.5至5.5.2-alpha版本。

文档生成于： 2010-11-15

目录

前言
1. 一般信息

- 1.1. 关于本手册
- 1.2. 本手册采用的惯例
- 1.3. MySQL AB概述
- 1.4. MySQL数据库管理系统概述
  - 1.4.1. MySQL的历史
  - 1.4.2. MySQL的的主要特性
  - 1.4.3. MySQL稳定性
  - 1.4.4. MySQL表最大能达到多少
  - 1.4.5. 2000年兼容性
- 1.5. MaxDB数据库管理系统概述
  - 1.5.1. 什么是MaxDB？
  - 1.5.2. MaxDB的历史
  - 1.5.3. MaxDB的特性
  - 1.5.4. 许可和支持
  - 1.5.5. MaxDB和MySQL之间的特性差异
  - 1.5.6. MaxDB和MySQL之间的协同性
  - 1.5.7. 与MaxDB有关的链接
- 1.6. MySQL发展大事记
  - 1.6.1. MySQL 5.5的新特性
- 1.7. MySQL信息源
  - 1.7.1. MySQL邮件列表
  - 1.7.2. IRC（在线聊天系统）上的MySQL社区支持
  - 1.7.3. MySQL论坛上的MySQL社区支持
- 1.8. MySQL标准的兼容性

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

1.8.1. MySQL遵从的标准是什么
1.8.2. 选择SQL模式
1.8.3. 在ANSI模式下运行MySQL
1.8.4. MySQL对标准SQL的扩展
1.8.5. MySQL与标准SQL的差别
1.8.6. MySQL处理约束的方式

2. 安装MySQL

2.1. 一般安装问题

2.1.1. MySQL支持的操作系统
2.1.2. 选择要安装的MySQL分发版
2.1.3. 怎样获得MySQL
2.1.4. 通过MD5校验和或GnuPG验证软件包的完整性
2.1.5. 安装布局

2.2. 使用二进制分发版的标准MySQL安装
2.3. 在Windows上安装MySQL

2.3.1. Windows系统要求
2.3.2. 选择安装软件包
2.3.3. 用自动安装器安装MySQL
2.3.4. 使用MySQL安装向导
2.3.5. 使用配置向导
2.3.6. 通过非安装Zip文件安装MySQL
2.3.7. 提取安装档案文件
2.3.8. 创建选项文件
2.3.9. 选择MySQL服务器类型
2.3.10. 首次启动服务器
2.3.11. 从Windows命令行启动MySQL
2.3.12. 以Windows服务方式启动MySQL
2.3.13. 测试MySQL安装
2.3.14. 在Windows环境下对MySQL安装的故障诊断与排除
2.3.15. 在Windows下升级MySQL
2.3.16. Windows版MySQL同Unix版MySQL对比

2.4. 在Linux下安装MySQL
2.5.在Mac OS X中安装MySQL
2.6. 在NetWare中安装MySQL
2.7. 在其它类Unix系统中安装MySQL
2.8. 使用源码分发版安装MySQL

2.8.1. 源码安装概述
2.8.2. 典型配置选项
2.8.3. 从开发源码树安装
2.8.4. 处理MySQL编译问题
2.8.5. MIT-pthreads注意事项
2.8.6. 在Windows下从源码安装MySQL
2.8.7. 在Windows下编译MySQL客户端

2.9. 安装后的设置和测试

2.9.1. Windows下安装后的过程
2.9.2. Unix下安装后的过程

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

2.9.3. 使初始MySQL账户安全

2.10. 升级MySQL

2.10.1. 从5.0版升级
2.10.2. 升级授权表
2.10.3. 将MySQL数据库拷贝到另一台机器

2.11. 降级MySQL
2.12. 具体操作系统相关的注意事项

2.12.1. Linux注意事项
2.12.2. Mac OS X注意事项
2.12.3. Solaris注意事项
2.12.4. BSD注意事项
2.12.5. 其它Unix注意事项
2.12.6. OS/2注意事项

2.13. Perl安装注意事项

2.13.1. 在Unix中安装Perl
2.13.2. 在Windows下安装ActiveState Perl
2.13.3. 使用Perl DBI/DBD接口的问题

3. 教程

3.1. 连接与断开服务器
3.2. 输入查询
3.3. 创建并使用数据库

3.3.1. 创建并选择数据库
3.3.2. 创建表
3.3.3. 将数据装入表中
3.3.4. 从表检索信息

3.4. 获得数据库和表的信息
3.5. 在批处理模式下使用**my**sql
3.6. 常用查询的例子

3.6.1. 列的最大值
3.6.2. 拥有某个列的最大值的行
3.6.3. 列的最大值：按组
3.6.4. 拥有某个字段的组间最大值的行
3.6.5. 使用用户变量
3.6.6. 使用外键
3.6.7. 根据两个键搜索
3.6.8. 根据天计算访问量
3.6.9. 使用AUTO_INCREMENT

3.7. 孪生项目的查询

3.7.1. 查找所有未分发的孪生项
3.7.2. 显示孪生对状态的表

3.8. 与Apache一起使用MySQL

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

4. MySQL程序概述

4.1. MySQL程序概述
4.2. 调用MySQL程序
4.3. 指定程序选项

4.3.1. 在命令行上使用选项
4.3.2. 使用选项文件
4.3.3. 用环境变量指定选项
4.3.4. 使用选项设置程序变量

5. 数据库管理

5.1. MySQL服务器和服务器启动脚本

5.1.1. 服务器端脚本和实用工具概述
5.1.2. mysq**ld-max**扩展**My**SQL服务器
5.1.3. mysqld_safe：MySQL服务器启动脚本
5.1.4. mysql.server：MySQL服务器启动脚本
5.1.5. mysqld_multi：管理多个MySQL服务器的程序

5.2. mysqlmanager：MySQL实例管理器

5.2.1. 用MySQL实例管理器启动MySQL服务器
5.2.2. 连接到MySQL实例管理器并创建用户账户
5.2.3. MySQL实例管理器命令行选项
5.2.4. MySQL实例管理器配置文件
5.2.5. MySQL实例管理器识别的命令

5.3. mysqld：MySQL服务器

5.3.1. **mysqld**命令行选项
5.3.2. SQL服务器模式
5.3.3. 服务器系统变量
5.3.4. 服务器状态变量

5.4. mysql_fix_privilege_tables：升级MySQL系统表
5.5. MySQL服务器关机进程
5.6. 一般安全问题

5.6.1. 通用安全指南
5.6.2. 使MySQL在攻击者面前保持安全
5.6.3. Mysqld安全相关启动选项
5.6.4. LOAD DATA LOCAL安全问题

5.7. MySQL访问权限系统

5.7.1. 权限系统的作用
5.7.2. 权限系统工作原理
5.7.3. MySQL提供的权限
5.7.4. 与MySQL服务器连接
5.7.5. 访问控制, 阶段1：连接核实
5.7.6. 访问控制, 阶段2：请求核实
5.7.7. 权限更改何时生效
5.7.8. 拒绝访问错误的原因

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

5.7.9. MySQL 4.1中的密码哈希处理

5.8. MySQL用户账户管理

5.8.1. MySQL用户名和密码
5.8.2. 向MySQL增加新用户账户
5.8.3. 从MySQL删除用户账户
5.8.4. 限制账户资源
5.8.5. 设置账户密码
5.8.6. 使你的密码安全
5.8.7. 使用安全连接

5.9. 备份与恢复

5.9.1. 数据库备份
5.9.2. 示例用备份与恢复策略
5.9.3. 自动恢复
5.9.4. 表维护和崩溃恢复
5.9.5. myisamchk：MyISAM表维护实用工具
5.9.6. 建立表维护计划
5.9.7. 获取关于表的信息

5.10. MySQL本地化和国际应用

5.10.1. 数据和排序用字符集
5.10.2. 设置错误消息语言
5.10.3. 添加新的字符集
5.10.4. 字符定义数组
5.10.5. 字符串比较支持
5.10.6. 多字节字符支持
5.10.7. 字符集问题
5.10.8. MySQL服务器时区支持

5.11. MySQL日志文件

5.11.1. 错误日志
5.11.2. 通用查询日志
5.11.3. 二进制日志
5.11.4. 慢速查询日志
5.11.5. 日志文件维护

5.12. 在同一台机器上运行多个MySQL服务器

5.12.1. 在Windows下运行多个服务器
5.12.2. 在Unix中运行多个服务器
5.12.3. 在多服务器环境中使用客户端程序

5.13. MySQL查询高速缓冲

5.13.1. 查询高速缓冲如何工作
5.13.2. 查询高速缓冲SELECT选项
5.13.3. 查询高速缓冲配置
5.13.4. 查询高速缓冲状态和维护

6. MySQL中的复制

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

6.1. 复制介绍
6.2. 复制实施概述
6.3. 复制实施细节

6.3.1. 复制主线程状态
6.3.2. 复制从I/O线程状态
6.3.3. 复制从SQL线程状态
6.3.4. 复制传递和状态文件

6.4. 如何设置复制
6.5. 不同MySQL版本之间的复制兼容性
6.6. 升级复制设置

6.6.1. 将复制升级到5.0版

6.7. 复制特性和已知问题
6.8. 复制启动选项
6.9. 复制FAQ
6.10. 复制故障诊断与排除
6.11. 通报复制缺陷
6.12. 多服务器复制中的Auto-Increment

7. 优化

7.1. 优化概述

7.1.1. MySQL设计局限与折衷
7.1.2. 为可移植性设计应用程序
7.1.3. 我们已将MySQL用在何处？
7.1.4. MySQL基准套件
7.1.5. 使用自己的基准

7.2. 优化SELECT语句和其它查询

7.2.1. EXPLAIN语法（获取SELECT相关信息）
7.2.2. 估计查询性能
7.2.3. SELECT查询的速度
7.2.4. MySQL怎样优化WHERE子句
7.2.5. 范围优化
7.2.6. 索引合并优化
7.2.7. MySQL如何优化IS NULL
7.2.8. MySQL如何优化DISTINCT
7.2.9. MySQL如何优化LEFT JOIN和RIGHT JOIN
7.2.10. MySQL如何优化嵌套Join
7.2.11. MySQL如何简化外部联合
7.2.12. MySQL如何优化ORDER BY
7.2.13. MySQL如何优化GROUP BY
7.2.14. MySQL如何优化LIMIT
7.2.15. 如何避免表扫描
7.2.16. INSERT语句的速度
7.2.17. UPDATE语句的速度
7.2.18. DELETE语句的速度
7.2.19. 其它优化技巧

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

7.3. 锁定事宜

7.3.1. 锁定方法
7.3.2. 表锁定事宜

7.4. 优化数据库结构

7.4.1. 设计选择
7.4.2. 使你的数据尽可能小
7.4.3. 列索引
7.4.4. 多列索引
7.4.5. MySQL如何使用索引
7.4.6. MyISAM键高速缓冲
7.4.7. MyISAM索引统计集合
7.4.8. MySQL如何计算打开的表
7.4.9. MySQL如何打开和关闭表
7.4.10. 在同一个数据库中创建多个表的缺陷

7.5. 优化MySQL服务器

7.5.1. 系统因素和启动参数的调节
7.5.2. 调节服务器参数
7.5.3. 控制查询优化器的性能
7.5.4. 编译和链接怎样影响MySQL的速度
7.5.5. MySQL如何使用内存
7.5.6. MySQL如何使用DNS

7.6. 磁盘事宜

7.6.1. 使用符号链接

8. 客户端和实用工具程序

8.1. 客户端脚本和实用工具概述
8.2. myisampack：生成压缩、只读MyISAM表
8.3. mysql：MySQL命令行工具

8.3.1. 选项
8.3.2. **mysql**命令
8.3.3. 怎样从文本文件执行SQL语句
8.3.4. **mysql**技巧

8.4. mysqlaccess：用于检查访问权限的客户端
8.5. mysqladmin：用于管理MySQL服务器的客户端
8.6. mysqlbinlog：用于处理二进制日志文件的实用工具
8.7. mysqlcheck：表维护和维修程序
8.8. mysqldump：数据库备份程序
8.9. mysqlhotcopy：数据库备份程序
8.10. mysqlimport：数据导入程序
8.11. mysqlshow－显示数据库、表和列信息
8.12. myisamlog：显示MyISAM日志文件内容
8.13. perror：解释错误代码
8.14. replace：字符串替换实用工具
8.15. mysql_zap：杀死符合某一模式的进程

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

9. 语言结构

9.1. 文字值

9.1.1. 字符串
9.1.2. 数值
9.1.3. 十六进制值
9.1.4. 布尔值
9.1.5. 位字段值
9.1.6. NULL值

9.2. 数据库、表、索引、列和别名

9.2.1. 识别符限制条件
9.2.2. 识别符大小写敏感性

9.3. 用户变量
9.4. 系统变量

9.4.1. 结构式系统变量

9.5. 注释语法
9.6. MySQL中保留字的处理

10. 字符集支持

10.1. 常规字符集和校对
10.2. MySQL中的字符集和校对
10.3. 确定默认字符集和校对

10.3.1. 服务器字符集和校对
10.3.2. 数据库字符集和校对
10.3.3. 表字符集和校对
10.3.4. 列字符集和校对
10.3.5. 字符集和校对分配示例
10.3.6. 连接字符集和校对
10.3.7. 字符串文字字符集和校对
10.3.8. 在SQL语句中使用COLLATE
10.3.9. COLLATE子句优先
10.3.10. BINARY操作符
10.3.11. 校对确定较为复杂的一些特殊情况
10.3.12. 校对必须适合字符集
10.3.13. 校对效果的示例

10.4. 字符集支持影响到的操作

10.4.1. 结果字符串
10.4.2. CONVERT()
10.4.3. CAST()
10.4.4. SHOW语句

10.5. Unicode支持
10.6. 用于元数据的UTF8
10.7. 与其它DBMS的兼容性
10.8. 新字符集配置文件格式

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

10.9. 国家特有字符集
10.10. MySQL支持的字符集和校对

10.10.1. Unicode字符集
10.10.2. 西欧字符集
10.10.3. 中欧字符集
10.10.4. 南欧与中东字符集
10.10.5. 波罗的海字符集
10.10.6. 西里尔字符集
10.10.7. 亚洲字符集

11. 列类型

11.1. 列类型概述

11.1.1. 数值类型概述
11.1.2. 日期和时间类型概述
11.1.3. 字符串类型概述

11.2. 数值类型
11.3. 日期和时间类型

11.3.1. DATETIME、DATE和TIMESTAMP类型
11.3.2. TIME类型
11.3.3. YEAR类型
11.3.4. Y2K事宜和日期类型

11.4. String类型

11.4.1. CHAR和VARCHAR类型
11.4.2. BINARY和VARBINARY类型
11.4.3. BLOB和TEXT类型
11.4.4. ENUM类型
11.4.5. SET类型

11.5. 列类型存储需求
11.6. 选择正确的列类型
11.7. 使用来自其他数据库引擎的列类型

12. 函数和操作符

12.1. 操作符

12.1.1. 操作符优先级
12.1.2. 圆括号
12.1.3. 比较函数和操作符
12.1.4. 逻辑操作符

12.2. 控制流程函数
12.3. 字符串函数

12.3.1. 字符串比较函数

12.4. 数值函数

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

12.4.1. 算术操作符
12.4.2. 数学函数

12.5. 日期和时间函数
12.6. MySQL使用什么日历？
12.7. 全文搜索功能

12.7.1. 布尔全文搜索
12.7.2. 全文搜索带查询扩展
12.7.3. 全文停止字
12.7.4. 全文限定条件
12.7.5. 微调MySQL全文搜索

12.8. Cast函数和操作符
12.9. 其他函数

12.9.1. 位函数
12.9.2. 加密函数
12.9.3. 信息函数
12.9.4. 其他函数

12.10. 与GROUP BY子句同时使用的函数和修改程序

12.10.1. GROUP BY（聚合）函数
12.10.2. GROUP BY修改程序
12.10.3. 具有隐含字段的GROUP BY

13. SQL语句语法

13.1. 数据定义语句

13.1.1. ALTER DATABASE语法
13.1.2. ALTER TABLE语法
13.1.3. CREATE DATABASE语法
13.1.4. CREATE INDEX语法
13.1.5. CREATE TABLE语法
13.1.6. DROP DATABASE语法
13.1.7. DROP INDEX语法
13.1.8. DROP TABLE语法
13.1.9. RENAME TABLE语法

13.2. 数据操作语句

13.2.1. DELETE语法
13.2.2. DO语法
13.2.3. HANDLER语法
13.2.4. INSERT语法
13.2.5. LOAD DATA INFILE语法
13.2.6. REPLACE语法
13.2.7. SELECT语法
13.2.8. Subquery语法
13.2.9. TRUNCATE语法
13.2.10. UPDATE语法

13.3. MySQL实用工具语句

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

13.3.1. DESCRIBE语法（获取有关列的信息）
13.3.2. USE语法

13.4. MySQL事务处理和锁定语句

13.4.1. START TRANSACTION, COMMIT和ROLLBACK语法
13.4.2. 不能回滚的语句
13.4.3. 会造成隐式提交的语句
13.4.4. SAVEPOINT和ROLLBACK TO SAVEPOINT语法
13.4.5. LOCK TABLES和UNLOCK TABLES语法
13.4.6. SET TRANSACTION语法
13.4.7. XA事务

13.5. 数据库管理语句

13.5.1. 账户管理语句
13.5.2. 表维护语句
13.5.3. SET语法
13.5.4. SHOW语法
13.5.5. 其它管理语句

13.6. 复制语句

13.6.1. 用于控制主服务器的SQL语句
13.6.2. 用于控制从服务器的SQL语句

13.7. 用于预处理语句的SQL语法

14. 插件式存储引擎体系结构

14.1. 前言
14.2. 概述
14.3. 公共MySQL数据库服务器层
14.4. 选择存储引擎
14.5. 将存储引擎指定给表
14.6. 存储引擎和事务
14.7. 插入存储引擎
14.8. 拔出存储引擎
14.9. 插件式存储器的安全含义

15. 存储引擎和表类型

15.1. MyISAM存储引擎

15.1.1. MyISAM启动选项
15.1.2. 键所需的空间
15.1.3. MyISAM表的存储格式
15.1.4. MyISAM表方面的问题

15.2. InnoDB存储引擎

15.2.1. InnoDB概述
15.2.2. InnoDB联系信息
15.2.3. InnoDB配置
15.2.4. InnoDB启动选项

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

15.2.5. 创建InnoDB表空间
15.2.6. 创建InnoDB表
15.2.7. 添加和删除InnoDB数据和日志文件
15.2.8. InnoDB数据库的备份和恢复
15.2.9. 将InnoDB数据库移到另一台机器上
15.2.10. InnoDB事务模型和锁定
15.2.11. InnoDB性能调节提示
15.2.12. 多版本的实施
15.2.13. 表和索引结构
15.2.14. 文件空间管理和磁盘I/O
15.2.15. InnoDB错误处理
15.2.16. 对InnoDB表的限制
15.2.17. InnoDB故障诊断与排除

15.3. MERGE存储引擎

15.3.1. MERGE表方面的问题

15.4. MEMORY (HEAP)存储引擎
15.5. BDB (BerkeleyDB)存储引擎

15.5.1. BDB支持的操作系统
15.5.2. 安装BDB
15.5.3. BDB启动选项
15.5.4. BDB表的特性
15.5.5. 修改BDB所需的事宜
15.5.6. 对BDB表的限制
15.5.7. 使用BDB表时可能出现的错误

15.6. EXAMPLE存储引擎
15.7. FEDERATED存储引擎

15.7.1. 安装FEDERATED存储引擎
15.7.2. FEDERATED存储引擎介绍
15.7.3. 如何使用FEDERATED表
15.7.4. FEDERATED存储引擎的局限性

15.8. ARCHIVE存储引擎
15.9. CSV存储引擎
15.10. BLACKHOLE存储引擎

16. 编写自定义存储引擎

16.1. 前言
16.2. 概述
16.3. 创建存储引擎源文件
16.4. 创建handlerton
16.5. 对处理程序进行实例化处理
16.6. 定义表扩展
16.7. 创建表
16.8. 打开表
16.9. 实施基本的表扫描功能

16.9.1. 实施store_lock()函数
16.9.2. 实施external_lock()函数

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

16.9.3. 实施rnd_init()函数
16.9.4. 实施info()函数
16.9.5. 实施extra()函数
16.9.6. 实施rnd_next()函数

16.10. 关闭表
16.11. 为存储引擎添加对INSERT的支持
16.12. 为存储引擎添加对UPDATE的支持
16.13. 为存储引擎添加对DELETE的支持
16.14. API引用

16.14.1. bas_ext
16.14.2. close
16.14.3. create
16.14.4. delete_row
16.14.5. delete_table
16.14.6. external_lock
16.14.7. extra
16.14.8. info
16.14.9. open
16.14.10. rnd_init
16.14.11. rnd_next
16.14.12. store_lock
16.14.13. update_row
16.14.14. write_row

17. MySQL簇

17.1. MySQL簇概述
17.2. MySQL簇的基本概念
17.3. 多计算机的简单基础知识

17.3.1. 硬件、软件和联网
17.3.2. 安装
17.3.3. 配置
17.3.4. 首次启动
17.3.5. 加载示例数据并执行查询
17.3.6. 安全关闭和重启

17.4. MySQL簇的配置

17.4.1. 从源码创建MySQL簇
17.4.2. 安装软件
17.4.3. MySQL簇的快速测试设置
17.4.4. 配置文件

17.5. MySQL簇中的进程管理

17.5.1. 用于MySQL簇的MySQL服务器进程使用
17.5.2. **ndbd**，存储引擎节点进程
17.5.3. **ndb_mgmd**，“管理服务器”进程
17.5.4. **ndb_mgm**，“管理客户端”进程
17.5.5. 用于MySQL簇进程的命令选项

17.6. MySQL簇的管理

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

17.6.1. MySQL簇的启动阶段
17.6.2. "管理客户端"中的命令
17.6.3. MySQL簇中生成的事件报告
17.6.4. 单用户模式
17.6.5. MySQL簇的联机备份

17.7. 使用与MySQL簇的高速互连

17.7.1. 配置MySQL簇以使用SCI套接字
17.7.2. 理解簇互连的影响

17.8. MySQL簇的已知限制
17.9. MySQL簇发展的重要历程

17.9.1. MySQL 5.0中的MySQL簇变化
17.9.2. 关于MySQL簇的MySQL 5.1发展历程

17.10. MySQL簇常见问题解答
17.11. MySQL簇术语表

18. 分区

18.1. MySQL中的分区概述
18.2. 分区类型

18.2.1. RANGE分区
18.2.2. LIST分区
18.2.3. HASH分区
18.2.4. KEY分区
18.2.5. 子分区
18.2.6. MySQL分区处理NULL值的方式

18.3. 分区管理

18.3.1. RANGE和LIST分区的管理
18.3.2. HASH和KEY分区的管理
18.3.3. 分区维护
18.3.4. 获取关于分区的信息

19. MySQL中的空间扩展

19.1. 前言
19.2. OpenGIS几何模型

19.2.1. Geometry类的层次
19.2.2. 类Geometry
19.2.3. 类Point
19.2.4. 类Curve
19.2.5. 类LineString
19.2.6. 类Surface
19.2.7. 类Polygon
19.2.8. 类GeometryCollection
19.2.9. 类MultiPoint
19.2.10. 类MultiCurve
19.2.11. 类MultiLineString

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

19.2.12. 类MultiSurface
19.2.13. 类MultiPolygon

19.3. 支持的空间数据格式

19.3.1. 著名的文本(WKT)格式
19.3.2. 著名的二进制(WKB)格式

19.4. 创建具备空间功能的MySQL数据库

19.4.1. MySQL空间数据类型
19.4.2. 创建空间值
19.4.3. 创建空间列
19.4.4. 填充空间列
19.4.5. 获取空间数据

19.5. 分析空间信息

19.5.1. Geometry格式转换函数
19.5.2. Geometry函数
19.5.3. 从已有Geometry创建新Geometry的函数
19.5.4. 测试几何对象间空间关系的函数
19.5.5. 关于几何最小边界矩形（MBR）的关系
19.5.6. 测试几何类之间空间关系的函数

19.6. 优化空间分析

19.6.1. 创建空间索引
19.6.2. 使用空间索引

19.7. MySQL的一致性和兼容性

19.7.1. 尚未实施的GIS特性

20. 存储程序和函数

20.1. 存储程序和授权表
20.2. 存储程序的语法

20.2.1. CREATE PROCEDURE和CREATE FUNCTION
20.2.2. ALTER PROCEDURE和ALTER FUNCTION
20.2.3. DROP PROCEDURE和DROP FUNCTION
20.2.4. SHOW CREATE PROCEDURE和SHOW CREATE FUNCTION
20.2.5. SHOW PROCEDURE STATUS和SHOW FUNCTION STATUS
20.2.6. CALL语句
20.2.7. BEGIN ... END复合语句
20.2.8. DECLARE语句
20.2.9. 存储程序中的变量
20.2.10. 条件和处理程序
20.2.11. 光标
20.2.12. 流程控制构造

20.3. 存储程序、函数、触发程序和复制：常见问题
20.4. 存储子程序和触发程序的二进制日志功能

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

21. 触发程序

21.1. CREATE TRIGGER语法
21.2. DROP TRIGGER语法
21.3. 使用触发程序

22. 视图

22.1. ALTER VIEW语法
22.2. CREATE VIEW语法
22.3. DROP VIEW语法
22.4. SHOW CREATE VIEW语法

23. INFORMATION_SCHEMA信息数据库

23.1. INFORMATION_SCHEMA表

23.1.1. INFORMATION_SCHEMA SCHEMATA表
23.1.2. INFORMATION_SCHEMA TABLES表
23.1.3. INFORMATION_SCHEMA COLUMNS表
23.1.4. INFORMATION_SCHEMA STATISTICS表
23.1.5. INFORMATION_SCHEMA USER_PRIVILEGES表
23.1.6. INFORMATION_SCHEMA SCHEMA_PRIVILEGES表
23.1.7. INFORMATION_SCHEMA TABLE_PRIVILEGES表
23.1.8. INFORMATION_SCHEMA COLUMN_PRIVILEGES表
23.1.9. INFORMATION_SCHEMA CHARACTER_SETS表
23.1.10. INFORMATION_SCHEMA COLLATIONS表
23.1.11. INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY表
23.1.12. INFORMATION_SCHEMA TABLE_CONSTRAINTS表
23.1.13. INFORMATION_SCHEMA KEY_COLUMN_USAGE表
23.1.14. INFORMATION_SCHEMA ROUTINES表
23.1.15. INFORMATION_SCHEMA VIEWS表
23.1.16. INFORMATION_SCHEMA TRIGGERS表
23.1.17. 其他INFORMATION_SCHEMA表

23.2. SHOW语句的扩展

24. 精度数学

24.1. 数值的类型
24.2. DECIMAL数据类型更改
24.3. 表达式处理
24.4. 四舍五入
24.5. 精度数学示例

25. API和库

25.1. libmysqld，嵌入式MySQL服务器库

25.1.1. 嵌入式MySQL服务器库概述
25.1.2. 使用libmysqld编译程序
25.1.3. 使用嵌入式MySQL服务器时的限制
25.1.4. 与嵌入式服务器一起使用的选项
25.1.5. 嵌入式服务器中尚需完成的事项(TODO)
25.1.6. 嵌入式服务器示例

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

25.1.7. 嵌入式服务器的许可

25.2. MySQL C API

25.2.1. C API数据类型
25.2.2. C API函数概述
25.2.3. C API函数描述
25.2.4. C API预处理语句
25.2.5. C API预处理语句的数据类型
25.2.6. C API预处理语句函数概述
25.2.7. C API预处理语句函数描述
25.2.8. C API预处理语句方面的问题
25.2.9. 多查询执行的C API处理
25.2.10. 日期和时间值的C API处理
25.2.11. C API线程函数介绍
25.2.12. C API嵌入式服务器函数介绍
25.2.13. 使用C API时的常见问题
25.2.14. 创建客户端程序
25.2.15. 如何生成线程式客户端

25.3. MySQL PHP API

25.3.1. 使用MySQL和PHP的常见问题

25.4. MySQL Perl API
25.5. MySQL C++ API

25.5.1. Borland C++

25.6. MySQL Python API
25.7. MySQL Tcl API
25.8. MySQL Eiffel Wrapper
25.9. MySQL程序开发实用工具

25.9.1. msql2mysql：转换mSQL程序以用于MySQL
25.9.2. mysql_config：获取编译客户端的编译选项

26. 连接器

26.1. MySQL Connector/ODBC

26.1.1. MyODBC介绍
26.1.2. 关于ODBC和MyODBC的一般信息
26.1.3. 如何安装MyODBC
26.1.4. 在Windows平台上从二进制版本安装MyODBC
26.1.5. I在Unix平台上从二进制版本安装MyODBC
26.1.6. 在Windows平台上从源码版本安装MyODBC
26.1.7. 在Unix平台上从源码版本安装MyODBC
26.1.8. 从BitKeeper开发源码树安装MyODBC
26.1.9. MyODBC配置
26.1.10. 与MyODBC连接相关的事宜
26.1.11. MyODBC和Microsoft Access
26.1.12. MyODBC和Microsoft VBA及ASP
26.1.13. MyODBC和第三方ODBC工具
26.1.14. MyODBC通用功能

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

26.1.15. 基本的MyODBC应用步骤
26.1.16. MyODBC API引用
26.1.17. MyODBC数据类型
26.1.18. MyODBC错误代码
26.1.19. MyODBC与VB：ADO、DAO和RDO
26.1.20. MyODBC与Microsoft.NET
26.1.21. 感谢

26.2. MySQL Connector/NET

26.2.1. 前言
26.2.2. 下载并安装MySQL Connector/NET
26.2.3. Connector/NET体系结构
26.2.4. 使用MySQL Connector/NET
26.2.5. MySQL Connector/NET变更史

26.3. MySQL Connector/J

26.3.1. 基本的JDBC概念
26.3.2. 安装 Connector/J
26.3.3. JDBC引用
26.3.4. 与J2EE和其他Java框架一起使用 Connector/J
26.3.5. 诊断 Connector/J方面的问题
26.3.6. Changelog

26.4. MySQL Connector/MXJ

26.4.1. 前言
26.4.2. 支持平台：
26.4.3. Junit测试要求
26.4.4. 运行Junit测试
26.4.5. 作为JDBC驱动程序的一部分运行
26.4.6. 在Java对象中运行
26.4.7. MysqldResource API
26.4.8. 在JMX代理(custom)中运行
26.4.9. 部署在标准的JMX代理环境下 (JBoss)
26.4.10. 安装

27. 扩展MySQL

27.1. MySQL内部控件

27.1.1. MySQL线程
27.1.2. MySQL测试套件

27.2. 为MySQL添加新函数

27.2.1. 自定义函数接口的特性
27.2.2. CREATE FUNCTION/DROP FUNCTION语法
27.2.3. 添加新的自定义函数
27.2.4. 添加新的固有函数

27.3. 为MySQL添加新步骤

27.3.1. 步骤分析

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

27.3.2. 编写步骤

A. 问题和常见错误

A.1. 如何确定导致问题的原因
A.2. 使用MySQL程序时的常见错误

A.2.1. 拒绝访问
A.2.2. 无法连接到[local] MySQL服务器
A.2.3. 客户端不支持鉴定协议
A.2.4. 输入密码时出现密码错误
A.2.5. 主机的host_name被屏蔽
A.2.6. 连接数过多
A.2.7. 内存溢出
A.2.8. MySQL服务器不可用
A.2.9. 信息包过大
A.2.10. 通信错误和失效连接
A.2.11. 表已满
A.2.12. 无法创建文件／写入文件
A.2.13. 命令不同步
A.2.14. 忽略用户
A.2.15. 表tbl_name不存在
A.2.16. 无法初始化字符集
A.2.17. 文件未找到

A.3. 与安装有关的事宜

A.3.1. 与MySQL客户端库的链接问题
A.3.2. 如何以普通用户身份运行MySQL
A.3.3. 与文件许可有关的问题

A.4. 与管理有关的事宜

A.4.1. 如何复位根用户密码
A.4.2. 如果MySQL依然崩溃，应作些什么
A.4.3. MySQL处理磁盘满的方式
A.4.4. MySQL将临时文件储存在哪里
A.4.5. 如何保护或更改MySQL套接字文件/tmp/mysql.sock
A.4.6. 时区问题

A.5. 与查询有关的事宜

A.5.1. 搜索中的大小写敏感性
A.5.2. 使用DATE列方面的问题
A.5.3. 与NULL值有关的问题
A.5.4. 与列别名有关的问题
A.5.5. 非事务表回滚失败
A.5.6. 从相关表删除行
A.5.7. 解决与不匹配行有关的问题
A.5.8. 与浮点比较有关的问题

A.6. 与优化器有关的事宜
A.7. 与表定义有关的事宜

A.7.1. 与ALTER TABLE有关的问题

Edited by Foxit PDF Editor
Copyright (c) by Foxit Corporation, 2003 - 2010
For Evaluation Only.

A.7.2. 如何更改表中的列顺序
A.7.3. TEMPORARY TABLE问题

A.8. MySQL中的已知事宜

A.8.1. MySQL中的打开事宜

B. 错误代码和消息

B.1. 服务器错误代码和消息
B.2. 客户端错误代码和消息

C. 感谢

C.1. MySQL AB处的开发人
C.2. MySQL贡献人
C.3. 资料员和译员
C.4. MySQL使用和包含的库
C.5. 支持MySQL的软件包
C.6. 用于创建MySQL的工具
C.7. MySQL支持人员

D. MySQL变更史

D.1. 5.1.x版中的变更情况（开发）

D.1.1. 5.1.2版中的变更情况（尚未发布）
D.1.2. 5.1.1版中的变更情况（尚未发布）

D.2. MyODBC的变更情况

D.2.1. MyODBC 3.51.12的变更情况
D.2.2. MyODBC 3.51.11的变更情况

E. 移植到其他系统

E.1. 调试MySQL服务器

E.1.1. 针对调试编译MySQL
E.1.2. 创建跟踪文件
E.1.3. 在gdb环境下调试m**ysqld**
E.1.4. 使用堆栈跟踪
E.1.5. 使用日志文件找出mysqld中的错误原因
E.1.6. 如果出现表崩溃，请生成测试案例

E.2. 调试MySQL客户端
E.3. DBUG软件包
E.4. 关于RTS线程的注释
E.5. 线程软件包之间的差异

F. 环境变量
G. MySQL正则表达式
H. MySQL中的限制

H.1. 联合的限制

I. 特性限制

I.1. 对存储子程序和触发程序的限制
I.2. 对服务器端光标的限制
I.3. 对子查询的限制
I.4. 对视图的限制
I.5. 对XA事务的限制

J. GNU通用公共许可
K. MySQL FLOSS许可例外
索引

图形清单

14.1. MySQL插件式存储引擎的体系结构
14.2. 存储引擎比较
16.1. MySQL体系结构

表格清单

26.1. 连接属性
26.2. 转换表
26.3. 用于ResultSet.getObject()的MySQL类型和Java类型
26.4. MySQL对Java编码名称的翻译

示例清单

26.1. 从DriverManager获得连接
26.2. 使用java.sql.Statement执行SELECT查询
26.3. 存储程序示例
26.4. 使用Connection.prepareCall()
26.5. 注册输出参数
26.6. 设置CallableStatement输入参数
26.7. 检索结果和输出参数值
26.8. 使用Statement.getGeneratedKeys()检索AUTO_INCREMENT列的值
26.9. 使用SELECT LAST_INSERT_ID()检索AUTO_INCREMENT列的值
26.10. 在可更新的ResultSets中检索AUTO_INCREMENT列的值
26.11. 设置Unix环境下的CLASSPATH
26.12. 与J2EE应用服务器一起使用连接池
26.13. 重试逻辑的事务示例

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

前言

这是关于5.1版至5.1.2-alpha版MySQL数据库系统的参考手册。该手册不适用于旧版本MySQL软件，这是因为在MySQL 5.1和以前的版本存在很多功能性差异和其他差异。如果正在使用MySQL软件的较旧版本，请参阅*MySQL 5.0*参考手册，该手册涵盖了MySQL 5.0，或参阅*MySQL 4.1*参考手册，该手册涵盖了MySQL 4.1以及MySQL的所有早期版本。在手册的文本中，通过引用发布版本号（5.1.x），注明了MySQL 5.1的二级版本。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第1章：一般信息

目录

1.1. 关于本手册
1.2. 本手册采用的惯例
1.3. MySQL AB概述
1.4. MySQL数据库管理系统概述

- 1.4.1. MySQL的历史
- 1.4.2. MySQL的的主要特性
- 1.4.3. MySQL稳定性
- 1.4.4. MySQL表最大能达到多少
- 1.4.5. 2000年兼容性

1.5. MaxDB数据库管理系统概述

- 1.5.1. 什么是MaxDB？
- 1.5.2. MaxDB的历史
- 1.5.3. MaxDB的特性
- 1.5.4. 许可和支持
- 1.5.5. MaxDB和MySQL之间的特性差异
- 1.5.6. MaxDB和MySQL之间的协同性
- 1.5.7. 与MaxDB有关的链接

1.6. MySQL发展大事记

- 1.6.1. MySQL 5.1的新特性

1.7. MySQL信息源

- 1.7.1. MySQL邮件列表
- 1.7.2. IRC（在线聊天系统）上的MySQL社区支持
- 1.7.3. MySQL论坛上的MySQL社区支持

1.8. MySQL标准的兼容性

- 1.8.1. MySQL遵从的标准是什么
- 1.8.2. 选择SQL模式
- 1.8.3. 在ANSI模式下运行MySQL
- 1.8.4. MySQL对标准SQL的扩展
- 1.8.5. MySQL与标准SQL的差别
- 1.8.6. MySQL处理约束的方式

MySQL®软件提供了十分快速的多线程、多用户、牢靠的SQL（结构化查询语言）数据库服务器。 MySQL服务器定位于任务关键型、重负荷生产系统，并能嵌入在大量部署的软件中。MySQL是MySQL AB的注册商标。

MySQL软件采用双许可方式。用户可根据GNU通用公共许可（http://www.fsf.org/licenses/）条款，将MySQL软件作为开放源码产品使用，或从MySQL AB公司购买标准的商业许可证。关于我方许可策略的更多信息，请参见http://www.mysql.com/company/legal/licensing/。

在下面的清单中，介绍了本手册感兴趣的一些特殊部分。

- 关于MySQL数据库服务器的讨论，请参见1.4.2 “MySQL的的主要特性”。
- 关于安装说明，请参见第2章：安装MySQL。

- 关于将MySQL数据库软件移植到新体系结构或操作系统方面的技巧，请参见附录E：移植到其他系统。
- 关于从5.0版向上升级的更多信息，请参见2.10.1节，“从5.0版升级”。
- 关于MySQL数据库服务器的教程说明，请参见第3章：教程。
- 关于SQL示例和标准程序信息，请参见标准程序目录（在本发布版中，为sql-bench）。
- 关于新特性和缺陷更正的历史信息，请参见附录D：*MySQL*变更史。
- 关于当前已知缺陷和错误特性的列表，请参见A.8 “MySQL中的已知事宜”。
- 关于未来计划，请参见1.6 “MySQL发展大事记”。
- 关于本项目所有贡献人的清单，请参见 附录C：感谢。

重要说明：

请将错误报告（通常称为缺陷）以及问题和评论发送到http://bugs.mysql.com。请参见1.7.1.3 “如何通报缺陷和问题”。

如果在MySQL服务器中发现敏感的安全缺陷，请使用电子邮件立刻通知我们：security@mysql.com。

1.1. 关于本手册

这是关于5.1版至5.1.2-alpha版MySQL数据库系统的参考手册。该手册不适用于旧版本MySQL软件，这是因为在MySQL 5.1和以前的版本存在很多功能性差异和其他差异。如果正在使用MySQL软件的较旧版本，请参阅*MySQL 5.0*参考手册，该手册涵盖了MySQL 5.0，或参阅*MySQL 4.1*参考手册，该手册涵盖了MySQL 3.22、3.23、4.0和4.1系列。在手册的文本中，通过引用发布版本号（5.1.x），注明了MySQL 5.1的二级版本。

由于本手册是作为参考手册而编制的，在本手册中未提供关于SQL或关联数据库概念的一般说明。在本手册中，也不包含如何使用操作系统或命令行解释器方面的信息。

MySQL数据库软件始终在发展，参考手册也会相应地频繁更新。本手册的最新版本以在线方式提供，请使用http://dev.mysql.com/doc/上的搜索表单。也提供多重其他格式，包括HTML、PDF、和Windows CHM版本。

主要文档是DocBook XML文件的集合。对于HTML版本和其他格式，它们是使用DocBook XSL stylesheets自动生成的。

如果你有任何关于本手册应增加内容或更正内容方面的建议，请将其发送给文档编制团队：http://www.mysql.com/company/contact/。

本手册最初是由David Axmark和Michael “Monty” Widenius编制的。由MySQL文档编制团队负责维护，团队成员包括Paul DuBois、Stefan Hinz、Mike Hillyer和Jon Stephens。关于中多其他贡献人，请参见附录C：感谢。

本手册的版权归瑞典公司MySQL AB所有。MySQL®和MySQL徽标均是MySQL AB的注册商标。本手册中引用的其他商标和注册商标是相应所有人的财产，在本手册中仅将其用于辨识目的。

1.2. 本手册采用的惯例

本手册采用了特定的印刷惯例：

- 这类风格的文本用于SQL语句，数据库、表和列名称，C和Perl代码，以及环境变量。例如： 要想重新加载授权表，请使用FLUSH PRIVILEGES语句。

这类风格的文本用于指明键入的数如信息。

- 这类风格的文本用于指明可执行程序和脚本的名称，例如**mysql**（MySQL命令行客户端程序）和**mysqld**（MySQL服务器执行程序）。
- 这类风格的文本用于变量输入，应使用你选择的值替换它。

- 文件名和目录名采取下述方式："全程my.cnf位于目录/etc下"。
- 字符序列采取下述方式："要想使用通配符，请使用字符%"。
- 这类风格的文本用于强调。
- 这类风格的文本用于表头，并用于传递强调信息。

当出现准备在特定程序中执行的命令时，该程序将由位于命令前的提示符指明。例如，shell>指明命令将从注册外壳程序中执行，mysql>指明命令将从**mysql**客户端程序中执行：

```
shell> type a shell command here（在此输入shell命令）
mysql> type a mysql statement here（在此输入mysql语句）
```

"shell"是命令解释程序。在Unix平台上，它通常是程序，如**sh**、**csh**或**bash**。在Windows平台下，等效程序为**command.com**或**cmd.exe**，通常运行在控制台窗口中。

输入示例中显示的命令或语句时，不要输入示例中给出的提示符。

数据库、表和；列名必须代入语句中。为了指明该代入是必要的，在本手册中使用了*db_name*、*tbl_name*和*col_name*。例如，你将见到如下所示的语句：

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

这意味着，如果你输入了类似的语句，应提供你的数据库、表和列名，如下例所示：

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

SQL关键字不区分大小写，因此即可为大写也可为小写。在本手册中采用大写。

在语法介绍中，方括号（"["和"]"）用于指明可选字或子句。例如，在下面的语句中，IF EXISTS是可选的：

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

当某一语法成分由多个可选项组成时，可选项应用竖线"|"分开。当可能选择一组选择中的某一成员时，可选项将列在方括号（"["和"]"）中。

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

当必须选择一组选择中的某一成员时，可选项将列在大括号（"{"和"}"）中。

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

省略号（...）表明省略了语句的某一选择，通常是为了提供复杂语法的简短表述。例如，INSERT ... SELECT是后跟SLECT语句的INSERT语句的简短形式。

省略号还能指明语句的前部分语法元素可重复。在下面的示例中，可给定多个*reset_option*值，第1个值后每一个可由逗号分开：

```
RESET reset_option [,reset_option] ...
```

对于用来设置shell变量的命令，采用Bourne shell语法给出。例如，用于设置环境变量的序列和运行命令的序列，与下述Bourne shell语法给出的类似：

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

如果你正在使用**csh**或**tcsh**，必须使用略有不同的命令。应执行与下例所示类似的序列：

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

1.3. MySQL AB概述

MySQL AB是由MySQL创始人和主要开发人创办的公司。MySQL AB最初是由David Axmark、Allan Larsson和Michael"Monty"Widenius在瑞典创办的。

我们致力于开发MySQL数据库软件，并向新用户宣传推广它。MySQL AB拥有MySQL源代码、MySQL徽标和（注册）商标、以及本手册的版权。请参见

1.4 "MySQL数据库管理系统概述"。

MySQL的核心价值取向指明了我们对MySQL和开发源码的贡献。

这些核心价值取向规定了MySQL AB与MySQL服务器软件的协作方式：

- 成为世界上最好和使用最广泛的数据库。
- 面向所有人，而且所有人均能支付得起。
- 使用简单。
- 在保持快速和安全的同时不断改进。
- 使用和改进充满乐趣。
- 不存在缺陷。

这就是MySQL AB公司及其雇员的核心价值取向。

- 我们同意开放源码理念，并支持开放源码群。
- 我们的目标是成为最佳公民。
- 我们倾向于那些与我们有共同价值取向和思想倾向的合作伙伴。
- 我们将回复电子邮件并提供支持。
- 我们是一家与其他方联系在一起的"虚拟"公司。
- 我们反对软件专利。

在MySQL的网站（http://www.mysql.com/）上，给出了关于MySQL和MySQL的最新信息。

顺便提及一下，公司名中的"AB"是瑞典语"aktiebolag"或"股份公司"的首字母缩写。可将其翻译为"MySQL有限公司"。事实上，MySQL有限公司和MySQLGmbH均是MySQL AB子公司的名称。它们分别位于美国和德国。

1.4. MySQL数据库管理系统概述

1.4.1. MySQL的历史
1.4.2. MySQL的的主要特性
1.4.3. MySQL稳定性
1.4.4. MySQL表最大能达到多少
1.4.5. 2000年兼容性

MySQL是最流行的开放源码SQL数据库管理系统，它是由MySQL AB公司开发、发布并支持的。MySQL AB是由多名MySQL开发人创办的一家商业公司。它是一家第二代开放源码公司，结合了开放源码价值取向、方法和成功的商业模型。

在MySQL的网站（http://www.mysql.com/）上，给出了关于MySQL和MySQL的最新信息。

- MySQL是一种数据库管理系统。

数据库是数据的结构化集合。它可以是任何东西，从简单的购物清单到画展，或企业网络中的海量信息。要想将数据添加到数据库，或访问、处理计算机数据库中保存的数据，需要使用数据库管理系统，如MySQL服务器。计算机是处理大量数据的理想工具，因此，数据库管理系统在计算方面扮演着关键的中心角色，或是作为独立的实用工具，或是作为其他应用程序的组成部分。

- MySQL是一种关联数据库管理系统。

  关联数据库将数据保存在不同的表中，而不是将所有数据放在一个大的仓库内。这样就增加了速度并提高了灵活性。MySQL的SQL指得是"结构化查询语言"。SQL是用于访问数据库的最常用标准化语言，它是由ANSI/ISO SQL标准定义的。SQL标准自1986年以来不断演化发展，有数种版本。在本手册中，"SQL-92"指得是1992年发布的标准，"SQL:1999"指得是1999年发布的标准，"SQL:2003"指得是标准的当前版本。我们采用术语"SQL标准"标示SQL标准的当前版本。

- MySQL软件是一种开放源码软件。

  "开放源码"意味着任何人都能使用和改变软件。任何人都能从Internet下载MySQL软件，而无需支付任何费用。如果愿意，你可以研究源码并进行恰当的更改，以满足你自己的需求。MySQL软件采用了GPL（GNU通用公共许可证），http://www.mysql.com/company/legal/licensing/）。

- MySQL数据库服务器具有快速、可靠和易于使用的特点。

  如果它正是你所寻找的，不妨一试。MySQL服务器还有一套实用的特性集合，这些特性是通过与我们用户的密切合作而开发的。在我们的基准测试主页上，给出了MySQL服务器和其他数据库管理器的比较结果。请参见7.1.4 "MySQL基准套件"。

  MySQL服务器最初是为处理大型数据库而开发的，与已有的解决方案相比，它的速度更快，多年以来，它已成功用于众多要求很高的生产环境。尽管MySQL始终在不断发展，但目前MySQL服务器已能提供丰富和有用的功能。它具有良好的连通性、速度和安全性，这使的MySQL十分适合于访问Internet上的数据库。

- MySQL服务器工作在客户端/服务器模式下，或嵌入式系统中。

  MySQL数据库软件是一种客户端/服务器系统，由支持不同后端的1个多线程SQL服务器，数种不同的客户端程序和库，众多管理工具和广泛的应用编程接口API组成。

  我们还能以嵌入式多线程库的形式提供MySQL服务器，你可以将其链接到你的应用程序，从而获得更小、更快、和更易管理的产品。

- 有大量可用的共享MySQL软件。

  你所喜欢的应用程序和语言均支持MySQL数据库服务器，这种情况十分可能。

"MySQL"的正式发音是"My Ess Que Ell"（而不是"my sequel"）,但我们并不介意你的发音方式是"my sequel"或其他当地方式。

1.4.1. MySQL的历史

我们最初的出发点是，使用mSQL来连接我们的表，这类表采用了我们的快速低层面（ISAM）子程序。然而，经过一些测试后，我们得出结论，mSQL的速度或灵活性不足以满足我们的要求。其结果是，为我们的数据库提供了新的SQL接口，但API接口与mSQL的几乎一样。设计该API的目的在于，允许将为mSQL编写的第三方代码方便地移植到MySQL。

MySQL名称的起源不明。10多年来，我们的基本目录以及大量库和工具均采用了前缀"my"。不过，共同创办人Monty Widenius的女儿名字也叫"My"。时至今日，MySQL名称的起源仍是一个迷，即使对我们也一样。

MySQL Dolphin（我方徽标）的名称为"Sakila"，它是由MySQL AB公司的创办人从用户在"Dolphin命名"比赛中提供的众多建议中选定的。该名称是由来自非洲斯威士兰的开放源码软件开发人Ambrose Twebaze提出的。根据Ambrose的说法，按斯威士兰的本地语言，女性化名称Sakila源自SiSwati。Sakila也是坦桑尼亚、Arusha地区的一个镇的镇名，靠近Ambrose的母国乌干达。

1.4.2. MySQL的的主要特性

下面介绍了MySQL数据库软件的一些重要特性。关于当前特性和即将提供特性的更多信息，，请参见1.6节，"MySQL发展大事记"。

· 内部构件和可移植性

- 使用C和C++编写
- 用众多不同的编译器进行了测试
- 能够工作在众多不同的平台上。请参见2.1.1 "MySQL支持的操作系统"。
- 使用GNU Automake、Autoconf和Libtool进行移植。
- 提供了用于C、C++、Eiffel、Java、Perl、PHP、Python、Ruby和Tcl的API。请参见第25章：API和库。
- 采用核心线程的完全多线程 如果有多个CPU，它能方便地使用这些CPU。
- 提供了事务性和非事务性存储引擎。
- 使用了极快的"B树"磁盘表（MyISAM）和索引压缩。
- 添加另一个存储引擎相对简单。如果打算为内部数据库添加一个SQL接口，该特性十分有用。
- 极快的基于线程的内存分配系统。
- 通过使用优化的"单扫描多连接"，能实现极快的连接。
- 存储器中的哈希表用作临时表。
- SQL函数是使用高度优化的类库实现的，运行很快。通常，在完成查询初始化后，不存在存储器分配。
- 采用Purify（商业内存溢出检测器）以及GPL工具Valgrind（http://developer.kde.org/~sewardj/）测试了MySQL代码。
- 服务器可作为单独程序运行在客户端/服务器联网环境下。它也可作为库提供，可嵌入（链接）到独立的应用程序中。这类应用程序可单独使用，也能在网络环境下使用。

- 列类型
  - 众多列类型： 带符号/无符号整数，1、2、3、4、8字节长，FLOAT，DOUBLE，CHAR，VARCHAR，TEXT，BLOB，DATE，TIME，DATETIME，TIMESTAMP，YEAR，SET，ENUM，以及OpenGIS空间类型。请参见第11章：列类型。
  - 定长和可变长度记录。
- 语句和函数
  - 在SELECT和查询的WHERE子句中，提供完整的操作符和函数支持。例如：

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

  - 对SQL GROUP BY和ORDER BY子句的全面支持。支持聚合函数(COUNT(), COUNT(DISTINCT ...)，AVG()，STD()，SUM()，MAX()，MIN()和GROUP_CONCAT())。
  - 支持LEFT OUTER JOIN和RIGHT OUTER JOIN，采用标准的SQL和ODBC语法。
  - 按照标准SQL的要求，支持表别名和列别名。
  - DELETE、INSERT、REPLACE和UPDATE返回更改（影响）的行数。连接到服务器时，可通过设置标志返回匹配的行数。
  - MySQL的SHOW命令可用于检索关于数据库、数据库引擎、表和索引的信息。EXPLAIN命令可用于确定优化器处理查询的方式。
  - 函数名与表名或列名不冲突。例如，ABS是有效的列名。唯一的限制在于，调用函数时，函数名和随后的符号"("之间不得有空格。请参见9.6 "MySQL中保留字的处理"。

  - 可以将不同数据库的表混合在相同的查询中（就像MySQL 3.22中那样）。

- 安全
  - 十分灵活和安全的权限和密码系统，允许基于主机的验证。连接到服务器时，所有的密码传输均采用加密形式，从而保证了密码安全。

- 可伸缩性和限制
  - 处理大型数据库： 我们使用了MySQL服务器和含5千万条记录的数据库。我们还听说，有些用户将MySQL用于含60000个表和约50亿行的数据库。
  - 每个表可支持高达64条索引（在MySQL 4.1.2之前为32条）。每条索引可由1～16个列或列元素组成。最大索引宽度为1000字节（在MySQL 4.1.2之前为500）。索引可使用具备CHAR、VARCHAR、BLOB或TEXT列类型的列前缀。

- 连接性
  - 在任何平台上，客户端可使用TCP/IP协议连接到MySQL服务器。在Windows系统的NT系列中（NT、2000、XP或2003），客户端可使用命名管道进行连接。在Unix系统中，客户端可使用Unix域套接字文件建立连接。
  - 在MySQL 4.1和更高的版本中，如果是以"--shared-memory"选项开始，Windows服务器还支持共享内存连接。客户端可使用"--protocol=memory"选项，通过共享内存建立连接。
  - Connector/ODBC (MyODBC)接口为使用ODBC（开放式数据库连接性）连接的客户端程序提供了MySQL支持。例如，可以使用MS Access连接到你的MySQL服务器。客户端可运行在Windows或Unix平台上。提供了MyODBC源。支持所有的ODBC 2.5函数，以及众多其他函数。请参见第26章：连接器。
  - Connector/J接口为使用JDBC连接的Java客户端程序提供了MySQL支持。客户端可运行在Windows或Unix平台上。提供了Connector/J源码。请参见第26章：连接器。

- 本地化
  - 服务器可使用多种语言向客户端提供错误消息。请参见5.10.2节，"设置错误消息语言"。
  - 对数种不同字符集的全面支持，包括latin1 (cp1252)、german、big5、ujis等。例如，在表名和列名中允许使用斯堪的纳维亚字符'å'、'ä'和'ö'。从MySQL 4.1开始，提供了Unicode支持。
  - 所有数据均以所选的字符集保存。正常字符串列的比较不区分大小写。
  - 分类是根据所选的字符集（默认情况下，使用瑞典校对）进行的。启动MySQL服务器时，可更改该项设置。要想查看高级分类的示例，请参见Czech分类代码。MySQL服务器支持众多不同的字符集，这类字符集可在编译时和运行时指定。

- 客户端和工具
  - MySQL服务器提供了对SQL语句的内部支持，可用于检查、优化和修复表。通过**mysqlcheck**客户端，可在命令行上使用这类语句。MySQL还包括**myisamchk**，这是一种很快的命令行实用工具，可用于在MyISAM表上执行这类操作。请参见第5章：数据库管理。
  - 对于所有MySQL程序，均能通过"-help"或"-?"选项调用，以获取联机帮助信息。

1.4.3. MySQL稳定性

本节回答了如下问题："MySQL服务器有多稳定？"，以及"在本项目中我能依靠MySQL服务器吗"？ 我们将尝试阐明这类问题，并回答很多潜在用户关心的某些重要问题。本节所给出的信息基于通过邮件列表收集的数据，在确定问题和通报使用类型方面，邮件列表十分有用。

最初的代码可回溯至20世纪80年代初。它提供了稳定的编码基数，最初存储引擎使用的ISAM表格式仍保持向后兼容性。在MySQL AB公司的前身TcX，自1996年中期以来，MySQL代码在多个项目中工作良好，未出现任何问题。当MySQL数据库软件首次向更广泛的公众发布时，我们的用户很快发现了一些未经测试的代码段。自那以后，尽管每个新版本具有很多新的特性，但每次新发布的版本均存在少量的移植性问题。

每次发布的MySQL服务器均是可用的。仅当用户尝试源自"灰色区域"的代码时才会出现问题。当然，新用户不了解"灰色区域"是什么。因此，在本节中，我们介绍了目前已知的这类区域。本节所作的介绍主要针对MySQL服务器3.23版和更高版本。在最新的版本中，更正了所有已知和通报的缺陷，但"缺陷"一节所列的除外，这类缺陷与设计有关。请参

见A.8节，“MySQL中的已知事宜”。

MySQL服务器采用了多层设计和独立模块。在此列出了一些较新的模块，并指明了它们的测试情况。

- Replication（稳定）

大量使用复制功能的服务器均处于生产模式下，结果良好。在MySQL 5.x中，将继续增强复制功能。

- InnoDB表（稳定）

自3.23.49版以来，InnoDB事务存储引擎一直很稳定。InnoDB正用于大型、重负荷生产系统。

- BDB表（稳定）

Berkeley DB码十分稳定，但在MySQL服务器中，我们仍在改进BDB事务存储引擎。

- 全文本搜索（稳定）

全文本搜索的使用范围十分广泛。在MySQL 4.0和4.1中，增加了重要的特性增强。

- MyODBC 3.51（稳定）

MyODBC 3.51采用了ODBC SDK 3.51，并广泛用于生产活动中。某些出现的情况看上去与应用程序相关，与ODBC驱动程序或底层数据库服务器无关。

1.4.4. MySQL表最大能达到多少

MySQL 3.22限制的表大小为4GB。由于在MySQL 3.23中使用了MyISAM存储引擎，最大表尺寸增加到了65536TB（$256^7 - 1$字节）。由于允许的表尺寸更大，MySQL数据库的最大有效表尺寸通常是由操作系统对文件大小的限制决定的，而不是由MySQL内部限制决定的。

InnoDB存储引擎将InnoDB表保存在一个表空间内，该表空间可由数个文件创建。这样，表的大小就能超过单独文件的最大容量。表空间可包括原始磁盘分区，从而使得很大的表成为可能。表空间的最大容量为64TB。

在下面的表格中，列出了一些关于操作系统文件大小限制的示例。这仅是初步指南，并不是最终的。要想了解最新信息，请参阅关于操作系统的文档。

操作系统	文件大小限制
Linux 2.2-Intel 32-bit	2GB (LFS: 4GB)
Linux 2.4+	(using ext3 filesystem) 4TB
Solaris 9/10	16TB
NetWare w/NSS filesystem	8TB
win32 w/ FAT/FAT32	2GB/4GB
win32 w/ NTFS	2TB（可能更大）
MacOS X w/ HFS+	2TB

在Linux 2.2平台下，通过使用对ext2文件系统的大文件支持（LFS）补丁，可以获得超过2GB的MyISAM表。在Linux 2.4平台下，存在针对ReiserFS的补丁，可支持大文件（高达2TB）。目前发布的大多数Linux版本均基于2.4内核，包含所有所需的LFS补丁。使用JFS和XFS，petabyte（千兆兆）和更大的文件也能在Linux上实现。然而，最大可用的文件容量仍取决于多项因素，其中之一就是用于存储MySQL表的文件系统。

关于Linux中LFS的详细介绍，请参见Andreas Jaeger的“Linux中的大文件支持”页面：http://www.suse.de/~aj/linux_lfs.html。

Windows用户请注意： FAT和VFAT (FAT32)不适合MySQL的生产使用。应使用NTFS。

在默认情况下，MySQL创建的MyISAM表允许的最大尺寸为4GB。你可以使用SHOW TABLE STATUS语句或**myisamchk -dv** *tbl_name*检查表的最大尺寸。请参见13.5.4节，“SHOW语法”。

如果需要使用大于4GB的MyISAM表（而且你的操作系统支持大文件），可使用允许AVG_ROW_LENGTH和MAX_ROWS选项的CREATE TABLE语句。请参见13.1.5节，“CREATE TABLE语法”。创建了表后，也可以使用ALTER TABLE更改这些选项，以增加表的最大允许容量。请参见13.1.2节，“ALTER TABLE语法”。

处理MyISAM表文件大小的其他方式：

- 如果你的大表是只读的，可使用**myisampack**压缩它。**myisampack**通常能将表压缩至少50％，因而，从结果上看，可获得更大的表。此外，**myisampack**还能将多个表合并为1个表。请参见8.2节，“myisampack：生成压缩、只读MyISAM表”。

- MySQL包含一个允许处理MyISAM表集合的MERGE库，这类MyISAM表具有与单个MERGE表相同的结构。请参见15.3节，“MERGE存储引擎”。

1.4.5. 2000年兼容性

MySQL服务器本身不存在2000年（Y2K）兼容性问题：

- MySQL服务器采用了Unix的时间功能，对于TIMESTAMP值，可处理的日期至2037年。对于DATE和DATETIME值，可接受的日期可至9999年。

- 所有的MySQL日期函数均是在1个源文件sql/time.cc中实现的，并经过了恰当编码以确保2000年安全。

- 在MySQL 3.22和以后的版本中，YEAR列类型能够在1个字节内保存0年以及1901～2155年，并能使用两位或四位数字显示它们。所有的两位数字年份均被视为介于1970～2069年之间，这意味着，如果你在YEAR列中保存了01，MySQL服务器会将其当作2001年。

通过下面的简单演示示例，表明MySQL服务器在处理直至9999年的DATE或DATETIME值方面不存在问题，在处理2030年以前的TIMESTAMP值方面也不存在问题：

```
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 (0.01 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),
    -> ('2040-01-01','2040-01-01 00:00:00',20400101000000),
```

```
    -> ('9999-12-31','9999-12-31 23:59:59',99991231235959);
Query OK, 14 rows affected (0.01 sec)
Records: 14  Duplicates: 0  Warnings: 2

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 |
| 2040-01-01 | 2040-01-01 00:00:00 | 00000000000000 |
| 9999-12-31 | 9999-12-31 23:59:59 | 00000000000000 |
+------------+---------------------+----------------+
14 rows in set (0.00 sec)
```

最后2个TIMESTAMP列的值为0，这是因为年份值（2040，9999）超出了TIMESTAMP的最大范围。TIMESTAMP数据类型用于保存当前时间，在32位机器上，支持的取值范围是19700101000000～20300101000000（带符号值）。在64位机器上，TIMESTAMP能处理的值达2106（无符号值）。

尽管MySQL服务器本身不存在2000年安全问题，但如果使用了存在Y2K问题的应用程序，也会遇到问题。例如，很多早期的应用程序采用2位数值（两可性）而不是4位数值来保存和处理年份数据。这类问题可能会被使用"00"或"99"的应用程序合并为"丢失"值指示符。很不幸，这类问题或许很难更正，这是因为不同的应用程序是由不同的程序员编写的，每位程序员可能使用了不同的惯例集和日期处理函数。

因此，尽管MySQL服务器不存在Y2K问题，但应用程序须提供无歧义的输入值。关于MySQL服务器在处理含2位年份数值的两可性日期输入数据方面的作用，请参见11.3.4节，"Y2K事宜和日期类型"。

1.5. MaxDB数据库管理系统概述

1.5.1. 什么是MaxDB？
1.5.2. MaxDB的历史
1.5.3. MaxDB的特性
1.5.4. 许可和支持
1.5.5. MaxDB和MySQL之间的特性差异
1.5.6. MaxDB和MySQL之间的协同性
1.5.7. 与MaxDB有关的链接

MaxDB是一种大型高效的企业数据库。数据库管理通过了SAP认证。

MaxDB是数据库管理系统的新名称，以前称为SAP DB。2003年，SAP AG和MySQL AB确立了合作伙伴关系，并将数据库系统重命名为MaxDB。自此以后，MaxDB的开发一直由SAP开发者团队负责，就像以前那样。

MySQL AB与MaxDB团队在SAP处保持着密切的合作，以不断改进MaxDB产品。两者的联合努力包括：开发新的固有驱动程序，以便能够在开发源码社区中更有效地使用MaxDB，并不断改善各种文档，以拓展MaxDB的用户基数。此外，MySQL和MaxDB数据库的协同性也被视为一项重要因素，例如，新的MaxDB同步管理器支持从MaxDB到MySQL的数据同步。

MaxDB数据库管理系统和MySQL数据库管理系统未共享公用编码基数。MaxDB和MySQL数据库管理系统是由MySQL AB公司提供的独立产品。

MySQL AB为MaxDB提供了全面的专业服务组合。

1.5.1. 什么是MaxDB？

MaxDB是兼容ANSI SQL-92（入门级）、由SAP AG提供的关联数据库管理系统（RDBMS)，也可由MySQL AB提供。MaxDB能够满足企业级应用的要求： 安全性，可伸缩性，高度并行性，以及强大的性能。它能运行在所有主要的操作系统下。多年的经历表明，它能运行，并能在24x7的运作中运行数以TB计的数据。

数据库开发是于1977年在柏林技术大学作为一个研究项目开始的。在20世纪80年代早期，它发展成为数据库产品，随后归Nixdorf、Siemens Nixdorf、Software AG所有，目前归SAP AG所有。在这一发展历程中，它先后被命名为VDN、Reflex、Supra 2、DDB/4、Entire SQL-DB-Server和ADABAS D。1997年，SAP从软件AP手中接管了该软件，并将其重新命名为SAP DB。自2000年10月起，依GNU通用公共许可的名义发布了众多的SAP DB源码（请参见附录J：GNU通用公共许可）。

2003年，SAP AG和MySQL AB确立了合作伙伴关系，并将数据库系统重命名为MaxDB。

1.5.2. MaxDB的历史

MaxDB的历史可追溯至SAP DB、SAP AG的DBMS，也就是说，MaxDB是SAP DB的重命名和增强版本。多年来，MaxDB已成功用于mySAP业务套件的小型、中性和大型安装实例，以及需要企业级DBMS的其他要求苛刻的SQL应用（涉及用户数、事务工作量、以及数据库的大小）。

除了第三方数据库系统外，如Oracle、Microsoft SQL Server以及IBM DB2，SAP DB意味着另一种选择。2000年10月，SAP AG依 GNU GPL许可（请参见附录J：GNU通用公共许可）发布了SAP DB，从而使得其成为开放源码软件。

目前，MaxDB已被世界各地约3500个SAP客户使用。不仅如此，在SAP的IT部门内，大多数安装在Unix和Linux平台上的DBMS均依赖于MaxDB。MaxDB正转向重负荷联机事务处理（OLTP），用户数以千计，数据库的大小从数百GB到数TB。

2003年，SAP和MySQL确立了合作伙伴关系，并达成了开发合作协议。作为其结果，自7.5版发布以来（2003年11月），SAP的数据库系统SAP DB以MySQL的MaxDB名义提供。

MaxDB 7.5版是SAP DB 7.4编码基数的直接改进。因此，MaxDB软件7.5版可用于SAP DB 7.2.04版和更高版本的直接升级。

与以往相同，目前，位于SAP AG的前SAP DB开发团队仍负责MaxDB的开发和支持。MySQL AB与位于SAP的MaxDB团队密切合作，致力于改进MaxDB产品，请参见1.5节，“MaxDB数据库管理系统概述”。SAP AG和MySQL AB均负责MaxDB的销售和分发。MaxDB和MySQL服务器的提升促进了企业协作，从而使得两种产品系列均从中受益。

与SAP解决方案一起提供之前，或放在MySQL站点供下载之前，MaxDB受SAP AG全面质量保证计划的控制。

1.5.3. MaxDB的特性

MaxDB是一种大型、通过SAP认证的开放源码数据库，可用于OLTP和OLAP，它具有高的可靠性、可用性和可伸缩性，以及相当完善的特性集。它定位于大型mySAP商业套件环境，以及需要最大企业级数据库功能的其他应用，此外，它还补充了MySQL数据库服务器。

MaxDB是采用客户端/服务器模式运作的产品。开发它的目的在于满足OLTP和数据仓库/OLAP/决策支持方面的安装需求。优点：

- 简单的配置和管理： 基于GUI（图形化用户界面）的安装管理器和数据库管理器，可作为DBMS操作的单个管理工具。
- 不间断操作，无需计划的停机时间，也不需要持久性维护：自动空间管理，无需重组。
- 精心设计的备份和恢复能力：联机备份和增量备份，恢复向导以指导你完成整个恢复步骤。

- 支持大量用户，数**TB**的数据库大小，以及苛刻的工作量要求： 高的可靠性，性能和可伸缩性
- 高可用性： 簇支持，待机配置，热待机配置

1.5.4. 许可和支持

使用MySQL AB提供的其他产品的许可证，可使用MaxDB。因此，可在GNU通用公共许可下以及商业许可下使用MaxDB。关于许可的更多信息，请访问http://www.mysql.com/company/legal/licensing/。

MySQL AB负责为非SAP客户提供MaxDB技术支持。MaxDB支持可在各种层面上提供（基本，银质和金质），将无限的电子邮件/Web支持扩展为对业务关键系统的全天候电话支持。

当MaxDB与Sap应用程序（如SAP NetWeaver和mySAP商业套件）一起使用时，MySQL AB还能为其提供许可证和支持。关于能满足您需求的许可和支持方面的更多信息，http://www.mysql.com/company/contact/MySQL AB。

我们也提供咨询和培训服务。MySQL将定期提供MaxDB课程，关于课程表，请参见http://www.mysql.com/training/。

1.5.5. MaxDB和MySQL之间的特性差异

MaxDB是MySQL AB公司通过SAP认证的数据库。MaxDB数据库服务器补充了MySQL AB产品系列。某些MaxDB特性在MySQL数据库服务器上不可用，反之亦然。

下面简要介绍了MaxDB和MySQL的主要差别，但并不完全。

- MaxDB是采用客户端/服务器模式运作的系统。MySQL能够作为客户端/服务器系统运行，也能作为嵌入式系统运行。
- MaxDB或许不能运行在MySQL支持的所有平台上。
- MaxDB采用了针对客户端／服务器通信的专有网络协议。MySQL采用了TCP/IP（采用或未采用SSL加密）、套接字协议（类似Unix的系统下）或命名管道（Windows NT系列下）。
- MaxDB支持存储程序。对于MySQL，在5.0版本中实现了存储程序。MaxDB还支持通过SQL扩展进行的触发程序编程，该功能计划在MySQL 5.1中实现。MaxDB包含针对存储程序语言的调试程序，能够将多个嵌套式触发程序串联在一起，而且每个动作和行均支持多个触发程序。
- MaxDB的发布采用了基于文本、图形或Web的用户界面。MySQL的发布仅采用基于文本的用户界面：图形化用户界面（MySQL控制中心、MySQL管理器）与主发布版本是单独提供的。针对MySQL的基于Web的用户界面是由第三方提供的。
- MaxDB支持多种也被MySQL支持的编程接口。为了使用MaxDB进行开发，还提供了MaxDB ODBC驱动程序，SQL数据库连通（SQLDBC），JDBC驱动程序，Perl和Python模块，以及MaxDB PHP扩展（通过使用PHP来访问MySQL MaxDB数据库）。第三方编程接口： 支持OLE DB、ADO、DAO、RDO、以及.NET和ODBC。MaxDB支持嵌入式SQL和C/C++。
- MaxDB包含MySQL不具备的管理特性： 按时间、事件和告警进行规划安排，并能在达到告警阈值时将消息发送给数据库管理器。

1.5.6. MaxDB和MySQL之间的协同性

MaxDB和MySQL是独立的数据库管理服务器。系统间的协同性是可能的，通过相应的方式，系统能够彼此交换数据。要想在MaxDB和MySQL之间交换数据，可使用系统的导入和导出工具，或MaxDB同步管理器。对于导入和导出工具，可在手动模式下传输数据（很少出现）。MaxDB同步管理器提供了更快的数据传输功能。

MaxDB装载器可用于导出数据和对象定义。装载器能够以MaxDB内部二进制格式和文本格式（CSV）导出数据。对于以文本格式从MaxDB导出的数据，可使用**mysqldump**数据库备份程序将其重新导入到MySQL中。要想将MySQL数据导入到MaxDB，可使用**mysqldump**创建INSERT语句或SELECT ... INTO OUTFILE语句以创建文本文件（CSV）。使用MaxDB装载器装载由MySQL生成的数据文件。

可以使用MaxDB装载器和MySQL工具**mysqldump**，在系统间交换数据定义。由于两种系统使用的SQL"方言"略有差异，而且MaxDB拥有目前尚不被MySQL支持的特性（如SQL约束），我们建议以手动方式调整定义文件。**Mysqldump**工具提供了"--compatible-name = maxdb"选项来生成与MaxDB兼容的输出，以便使移植更为简单。

作为MaxDB 7.6的组成部份，发布了MaxDB同步管理器。同步管理器支持数个MaxDB实例间的异步复制。但是，也设计规划了协同特性，因此，同步管理器支持复制到MySQL服 MySQL

务器的操作，以及来自　　服务器的复制操作。

在首次发布的版本中，同步管理器支持将数据插入到MySQL。这意味着，在开始时仅支持从MaxDB到MySQL的复制。在2005年的安排中，将增加把数据从MySQL服务器导出到同步管理器的功能，因而增加了对从MySQL到MaxDB的复制支持。

1.5.7. 与MaxDB有关的链接

关于MaxDB信息的主页位于http://www.mysql.com/products/maxdb。在该页面上，详细介绍了MaxDB数据库管理系统的特性，并提供了指向文档的多个超级链接。

除了本章给出的介绍外，MySQL参考手册不含任何MaxDB文档。MaxDB有自己的文档，称为MaxDB库。MaxDB库可从下述网址获得：http://dev.mysql.com/doc/maxdb/index.html。

MySQL AB运行着一个关于MaxDB的社区邮件列表，请参见http://lists.mysql.com/maxdb。该列表给出了生动活泼的社区讨论。很多核心开发人员均为其提供了相应的贡献。产品发布将被发送至该列表。

MaxDB的Web论坛网址是http://forums.mysql.com/。该论坛主要处理关于MaxDB的问题，而不是关于SAP应用程序的问题。

1.6. MySQL发展大事记

1.6.1. MySQL 5.1的新特性

在本节中，介绍了MySQL发展历程中的重要事件，包括各种MySQL版本中已实现的主要特性或规划中的特性。在下节中，介绍了各发布系列的相关信息。

当前的生产版本系列是MySQL 5.0，据称它能稳定地用于生产环境，如2005年10月发布的5.0.15版。以前的生产版本系列是MySQL 4.1，据称它也能稳定地用于生产环境，如2004年10月发布的4.1.7版。“生产状态”意味着未来的5.0和4.1开发仅限于修正缺陷。对于较早的MySQL 4.0和3.23系列，仅会对关键缺陷进行更正。

对于MySQL 5.0和5.1系列，相关的MySQL开发正在积极进行当中，并会为后者增加新的特性。

从1个版本系列升级到下一个版本系列之前，请参见2.10节，“升级MySQL”的介绍。

在下面的表格中，归纳了要求最迫切的特性，以及实施了这些特性或计划实施这些特性的版本：

特性	**MySQ**系列
Foreign keys	3.23（针对InnoDB存储引擎）
Unions	4.0
Subqueries	4.1
R-trees	4.1（针对MyISAM 存储引擎）
Stored procedures	5.0
Views	5.0
Cursors	5.0
XA transactions	5.0
Foreign keys	5.1（在3.23中实施，对于InnoDB）
Triggers	5.0和5.1
Full outer joins	5.1
Constraints	5.1（在3.23中实施，对于InnoDB）
Partitioning	5.1
Pluggable Storage Engine API	5.1
Row-Based Replication	5.1

1.6.1. MySQL 5.1的新特性

关于我们打算在MySQL 5.1中增加的特性列表，请参见1.6节，“MySQL发展大事记”。随着5.1版的不断发展，我们将在本节增加更多详细信息。

另请参见第18章：分区。

1.7. MySQL信息源

1.7.1. MySQL邮件列表
1.7.2. IRC（在线聊天系统）上的MySQL社区支持
1.7.3. MySQL论坛上的MySQL社区支持

1.7.1. MySQL邮件列表

1.7.1.1. MySQL邮件列表
1.7.1.2. 请教问题或通报缺陷
1.7.1.3. 如何通报缺陷和问题
1.7.1.4. 在邮件列表上回答问题的指南

在本节中介绍了MySQL邮件列表，并给出了使用邮件列表的指南。订购邮件列表后，将以电子邮件消息的形式收到所有已记录的信息。你也可以将自己的问题和解答发送至邮件列表。

1.7.1.1. MySQL邮件列表

要想订购本节所介绍的邮件列表或取消订购，请访问http://lists.mysql.com/。对于大多数邮件列表，可选择能够获取单独消息的正规列表版本，或选择按天发布的包含大量消息的文摘版本。

不要将订购信息或取消订购的信息发送到邮件列表，原因在于，这类消息将自动分发给数千位其他用户。

在你的所在地，可能有很多MySQL邮件列表的订户。如果是这样，该地点可能会有本地邮件列表，这样，从lists.mysql.com发出的消息将被传播到本地列表。在这类情形下，请与你的系统管理员联系，添加或删除本地MySQL列表。

如果希望将邮件列表的信息传送到邮件程序的邮箱中，请根据消息标题设置过滤器。可以使用列表ID： 或投递至： 识别列表消息的标题。

MySQL列表包含：

- 通告

该列表用于通告新的MySQL版本和相关程序。这是1种低容量列表，所有的MySQL用户均应订购它。

- mysql

这是关于一般MySQL讨论的主要列表。请注意，对于某些主题来说，在更专门的列表中会得到更好地讨论。如果将问题张贴到了错误的列表，可能不会得到回答。

- 缺陷

该列表面向那些希望随时了解自上次MySQL版本发布以来已通报事宜的人员，或希望积极参与缺陷寻找和更正进程的人员。请参见1.7.1.3节，“如何通报缺陷和问题”。

- 内部构件

该列表面向那些与MySQL代码打交道的人员。它也是讨论MySQL开发并张贴补丁的论坛。

- mysqldoc

该列表面向那些与MySQL文档打交道的人员： MySQL AB公司的人员，译者，以及其他社区成员。

- 基准

该列表面向任何对性能事宜感兴趣的人员。讨论主要集中在数据库性能方面（不限于MySQL），也包括更广的类别，如内核性能、文件系统、磁盘系统等。

- packagers（包装程序）

该列表主要讨论包装和分发MySQL方面的问题。这是供分发版维护人员交流MySQL打包事宜的论坛，为的是确保在所有支持的平台和操作系统上，MySQL的外观和感觉尽可能类似。

- java

该列表主要讨论MySQL服务器和Java方面的问题。它主要讨论JDBC驱动程序，包括MySQL Connector/J。

- win32

该列表涵盖了在Microsoft操作系统环境下（如Windows 9x, Me, NT, 2000, XP和2003）与MySQL软件有关的所有主题，

- myodbc

该列表涵盖了与使用ODBC连接到MySQL服务器有关的所有主题。

- gui-tools

该列表涵盖了与MySQL GUI工具有关的所有主题，包括MySQL管理员以及MySQL控制中心图形客户端。

- cluster

该列表主要讨论MySQL簇。

- dotnet

该列表主要讨论MySQL服务器和.NET平台方面的问题。它与MySQL Connector/Net提供人的关系最密切。

- plusplus

该列表涵盖了使用C++ API进行MySQL编程的所有主题。

- perl

该列表涵盖了与Perl对MySQL支持、以及DBD::mysql有关的所有主题。

如果无法从MySQL邮件列表或论坛获得问题解答，一种选择是购买MySQL AB的支持服务。这样，你就能与MySQL开发人员直接联系。

下面介绍了一些英语以外其他语言的MySQL邮件列表。这些邮件列表不是由MySQL AB运营的。

- <mysql-france-subscribe@yahoogroups.com>

法语邮件列表。

- <list@tinc.net>

朝鲜语邮件列表。发送电子邮件订购mysql your@email.address。

- <mysql-de-request@lists.4t2.com>

德语邮件列表。发送电子邮件订购mysql-de your@email.address。在http://www.4t2.com/mysql/站点上，可找到关于该邮件列表的更多信息。

- <mysql-br-request@listas.linkway.com.br>

葡萄牙语邮件列表。发送电子邮件订购mysql-br your@email.address。

- <mysql-alta@elistas.net>

西班牙语邮件列表。发送电子邮件订购mysql your@email.address。

1.7.1.2. 请教问题或通报缺陷

张贴缺陷报告或问题之前，请:

- 首先搜索MySQL在线手册，http://dev.mysql.com/doc/。我们经常更新该手册，以使该手册保持最新，其中包含相应的解决方案和新发现的问题。变更史（http://dev.mysql.com/doc/mysql/en/News.html）可能更有用，原因在于，在较新的版本中可能包含对你所提出问题的解决方案。
- 搜索缺陷数据库，http://bugs.mysql.com/，查找该缺陷是否已通报或更正。
- 搜索MySQL邮件列表档案，http://lists.mysql.com/。
- 你也可以使用http://www.mysql.com/search/来搜索MySQL AB网站上的所有网页（包含手册）。

如果无法在手册或档案中找到答案，请与本地MySQL专家协商。如果仍无法获得解答，在与我们联系之前，请按照介绍发送电子邮件至MySQL邮件列表，具体内容见下一节。

1.7.1.3. 如何通报缺陷和问题

通报缺陷的正常地址是http://bugs.mysql.com/，它也是我方缺陷数据库的地址。这是1个公共数据库，任何人都能浏览它并进行相应的搜索。如果登录到系统，可输入新的报告。

编写良好的缺陷报告需要耐心，但在第1时间正确地完成它不仅能节省我们的时间，也能节省你自己的时间。良好的缺陷报告应包含对缺陷的完整测试情况，以便我们能够在下个版本中更正该缺陷。本节介绍的内容用于帮助你正确地编写报告，从避免将你的时间浪费在对我们帮助不大或没有帮助的事上，

我们鼓励任何人使用**mysqlbug**脚本来生成缺陷报告（或通报问题）。**Mysqlbug**可在脚本目录下找到（源码分发版），也能在MySQL安装目录的bin子目录下找到（二进制分发版）。如果不能使用**mysqlbug**（例如，如果你正在Windows平台上运行），应包括本节所述的所有必要信息（更重要的是，应介绍操作系统和MySQL版本），这点十分重要。

通过自动确定下述信息，**mysqlbug**脚本能够帮助你生成报告，但是，如果遗漏了某些重要事项，请将其包含在消息中。请认真阅读本节，并确保在你的报告中包含了本节所述的所有信息。

在张贴问题前，最好使用MySQL服务器的最新生产版或开发版对问题进行测试。通过在所含的测试范例上使用“mysql test < script_file”，或运行缺陷报告中所含的Shell或Perl脚本，任何人均应能重复该缺陷。

对于在缺陷数据库（http://bugs.mysql.com/）中张贴的所有缺陷，均会被纳入或记录在下一个MySQL版本中。如果只需要少量更改就能更正问题，我们或许会给出更正该问题的补丁。

如果发现MySQL中存在敏感的安全缺陷，请发送电子邮件至security@mysql.com。

如果有1份可重复的缺陷报告，请将其提交到缺陷数据库，http://bugs.mysql.com/。注意，即使在该情况下，也应首先运行**mysqlbug**脚本以找出与你的系统有关的信息，这是一个不错的习惯。对于任何我们能再现的缺陷，在下一个MySQL版本中修正它的机会很大。

要想通报其他问题，请使用MySQL邮件列表。

请注意，我们可能会对包含过多信息的消息做出响应，但不太会对包含过少信息的消息做出回应。人们常会省略掉一些事实，因为他们认为自己知道了故障的原因，并想当然地认为这类细节无关紧要。良好的原则是： 如果你对陈述某事犹豫不定，请陈述之。如果我们要求你提供初始报告中缺少的信息，在报告中编写多行信息源比等候回复要快，麻烦也更小。

在缺陷报告中，最常犯的错误包括：（a）未包含所使用MySQL的版本号，以及（b）未完全描述安装了MySQL服务器的平台（包括平台类型，以及版本号）。这是高度相关的信息，如果没有它，99％的缺陷报告无用。我们遇到这类问题，“为何它对我没用”？ 随后，我们发现在该MySQL版本中，所请求的特性尚未实施，或在较新的MySQL版本中已更正了报告中描述的缺陷。有些时候，错误与平台相关，在这类情况下，如果不知道操作系统和平台的版本号，我们几乎不可能更正任何问题。

如果你是从源码编译MySQL的，如果与问题有关，还应提供有关编译器的信息。问题经常出在编译器，但人们却认为问题与MySQL有关。大多数编译均处于不断的开发过程中，并会变得越来越好。为了确定问题是否与你的编译器有关，我们需要知道你所使用的编译器。注意，所有的编译问题均应被当作缺陷并予以通报。

在你的报告中包含良好的问题描述时，报告最有帮助。也就是说，应给出示例，指明导致问题的所有事项，并准确描述问题本身。最好的报告应包含完整的示例，这类示例应阐明再现缺陷或问题的方式。请参见E.1.6节，“如果出现表崩溃，请生成测试案例”。

如果程序产生了错误消息，也应将其包含在你的报告中，这点很重要。如果我们打算使用程序搜索档案，最好是通报的错误消息与程序生成的错误消息准确匹配。（即使是字母的大小写也应考虑在内）。永远不要尝试从记忆中再现错误消息，而是应将整个消息拷贝并粘贴到报告中。

如果遇到与Connector/ODBC (MyODBC)有关的问题，请生成1份跟踪文件，并与报告一起发送给我们。请参见26.1.1.9节，“如何通报MyODBC问题或缺陷”。

请记住，很多阅读你报告的人员会使用80列的显示器。使用**mysql**命令行工具生成报告或示例时，如果输出内容可能会超过这类显示器的可用宽度，应使用“--vertical”选项（或“\G”语句终结符），例如EXPLAIN SELECT语句，请参见本节后面给出的示例。

请在你的报告中包含下述信息：

- 你所使用的MySQL分发版的版本号（例如MySQL 4.0.12）。通过执行**mysqladmin version**，即可了解正在运行版本。**Mysqladmin**程序位于MySQL安装目录的bin子目录下。
- 出现问题的机器的制造商和型号。
- 操作系统的名称和版本。如果你使用的是Windows操作系统，通常能通过双击“我的电脑”图标并点击“帮助/关于Windows”菜单来了解操作系统的名称和版本。对于大多数Unix操作系统，可通过执行命令uname –a获取这类信息。
- 某些时候，内存容量（实际内存和虚拟内存）也有关系。如果怀疑它，也应包含这类数值。
- 如果你正在使用的是MySQL软件的源码分发版，还须提供所使用编译器的名称和版本。如果使用的是二进制分发版，需要提供其名称。
- 如果在编译过程中出现问题，应给出准确的错误消息，出错文件中的不良代码，以及该代码附近的数行内容。
- 如果**mysqld**停止运行，还应通报导致**mysqld**崩溃的查询。通常，能够通过运行启用了查询日志功能的**mysqld**找出它，然后在**mysqld**崩溃后查找日志。请参见E.1.5节，“使用日志文件找出mysqld中的错误原因”。
- 如果数据库表与问题有关，还应包含**mysqldump --no-data** *db_name tbl_name*的输出。这是一种了解数据库中表相关信息的简单易行而且功能强大的方式。该信息能帮助我们建立与你所遇到的情况相匹配的场景。
- 对于与SELECT语句的速度有关的缺陷或问题，总应包含“EXPLAIN SELECT ...”的输出，以及SELECT语句生成的行数（至少）。对于每个涉及的表，应包含SHOW CREATE TABLE *tbl_name*的输出。你所提供的关于具体情况的信息越多，得到帮助的可能性就越大。

下面给出了一个良好缺陷报告的示例。应使用**mysqlbug**脚本张贴它。本例采用了**mysql**命令行工具。对于输出内容可能会超过80列显示器可用宽度的语句，应使用“\G”语句终结符。

```
mysql> SHOW VARIABLES;

mysql> SHOW COLUMNS FROM ...\G

       <output from SHOW COLUMNS>

mysql> EXPLAIN SELECT ...\G

       <output from EXPLAIN>

mysql> FLUSH STATUS;

mysql> SELECT ...;

       <A short version of the output from SELECT,

       including the time taken to run the query>

mysql> SHOW STATUS;

       <output from SHOW STATUS>
```

- 如果在运行**mysqld**时出现错误或问题，应提供导致异常的输入脚本。该脚本应包含任何所需的源文件。越能再现具体情况的脚本越好。如果能够创建可再现的测试范例，请将其张贴到http://bugs.mysql.com/，它将得到优先对待。

如果你不能提供脚本，至少应在你的邮件中包含**mysqladmin variables extended-status processlist**的输出，以提供关于系统执行情况的某些信息。

- 如果不能生成包含数行内容的测试范例，或者如果测试表过大以至于无法发送到邮件列表（超过10行），应使用**mysqldump**转储表，并创建描述问题的README文件。

使用**tar**和**gzip**或**zip**创建文件的压缩包档案，并使用FTP将档案传输到ftp://ftp.mysql.com/pub/mysql/upload/。然后将问题提交到我们的缺陷数据库中，http://bugs.mysql.com/。

- 如果你认为MySQL服务器生成了奇怪的查询结果，不仅应包含结果，还应给出你对该结果的看法，以及支持观点的基础。

- 提供问题的示例时，最好使用实际情况下已有的变量名、表名等，而不是新名称。问题可能与变量名或表名有关。或许这类情况很罕见，但安全总比道歉强。归根结底，对你来说，提供关于实际情况的示例要简单些，当然对我们也更好。如果你的数据不打算展示给其他人，请使用FTP将其传输到ftp://ftp.mysql.com/pub/mysql/upload/。如果信息是高度保密的，而且你甚至不打算向我们展示，请使用其他名称给出示例，但请注意，这应是最后的选择。

- 如果可能，应包含相关程序的所有选项。例如，应指明启动**mysqld**服务器时使用的选项，以及用来运行MySQL客户端程序的选项。对于程序（如**mysqld**和**mysql**）选项以及**configure**脚本的选项，通常是解答问题的关键，关系十分密切。包含它们总不是坏主意。如果使用了任何模块，如Perl或PHP等，还应给出它们的版本。

- 如果你的问题与权限系统有关，请给出**mysqlaccess**的输出，**mysqladmin reload**的输出，以及进行连接时获得的所有错误消息。测试权限时，首先应运行**mysqlaccess**。接下来，执行**mysqladmin reload version**，并与导致问题的程序相连。**mysqlaccess**可在MySQL安装目录的bin子目录下找到。

- 如果你有关于某一缺陷的补丁，也请将它包含在内。但不要认为该补丁是我们所需的全部，如果未提供补丁所更正缺陷的必要信息（如测试范例），不要假定我们会使用它。我们可能会通过补丁发现问题，或者不能理解该补丁，如果是这样，我们不会使用该补丁。

如果我们不能准确核实补丁的目的，将不会使用它。测试范例会对我们有所帮助。请指明该补丁能处理所有的问题。如果我们发现补丁不能工作的临界情况（即使很罕见），它可能是无用的。

- 关于缺陷是什么、出现原因、以及缺陷导因的猜测通常是错的。即使是MySQL团队，在未使用调试器判定缺陷真实原因的情况下，也不能妄加猜测。

- 请在你的缺陷报告中指明，你已参阅了参考手册并寄出了档案，以便让其他人知道你已作了自行解决问题的尝试。

- 如果遇到解析错误，请仔细检查语法。如果不能找出错误出现在那里，很可能是因为你使用的MySQL服务器版本不支持你使用的语法。如果你使用的是http://dev.mysql.com/doc/上提供的当前版本和手册，不要包含你所使用的语法，MySQL服务器不支持你的查询。在这种情况下，唯一的选择是自行实施语法，或发送电子邮件至<licensing@mysql.com>，并寻求实施方案。

如果手册中涵盖了你所使用的语法，但你使用的是旧版本MySQL服务器，请检查MySQL变更史，以查看语法的实施时间。在这种情况下，可以选择升级到较新的MySQL服务器版本。请参见附录D：MySQL变更史。

- 如果问题在于数据崩溃，或访问特殊表时出错，首先应使用CHECK TABLE和REPAIR TABLE或**myisamchk**进行检查并尝试修复。请参见第5章：数据库管理。

如果你使用的操作系统是Windows，请使用SHOW VARIABLES LIKE 'lower_case_table_names'命令核实“lower_case_table_names”的值。

- 如果经常获得崩溃的表，请尝试找出发生的时间和原因。在这种情况下，MySQL数据目录下的错误日志可能会包含关于它的一些信息。（这是名称中包含.err后缀的文件）。请参见5.11.1节，“错误日志”。在你的缺陷报告中，请包含该文件提供的相关信息。如果在更新期间，未杀死更新进程，正常情况下，**mysqld**不会造成表损坏。如果你能够找到**mysqld**停止的原因，我们会更容易地为你提供更正它的补丁。请参见A.1节，“如何确定导致问题的原因”。

- 如果可能，请下载并安装最新版本的MySQL服务器，并检查你的问题是否得到解决。所有版本的MySQL软件均经过彻底测试，并应能无故障运行。我们致力于尽可能地向后兼容，你也应能够毫不困难地在不同的MySQL版本间进行切换。请参见2.1.2节，“选择要安装的MySQL分发版”。

如果你是享受支持服务的客户，请将缺陷报告交叉张贴在mysql-support@mysql.com，以获得更高的优先级，并将其张贴到恰当的邮件列表，以查看是否有人遇到了类似问题（或解决了问题）。

关于通报MyODBC中存在缺陷的更多信息，请参见26.1.1.9节，“如何通报MyODBC问题或缺陷”。

关于某些常见问题的解决方案，请参见附录A：问题和常见错误。

将答案单独发送给你而不是发送到邮件列表时，良好的礼节是，对回答进行归纳总结并将结果发送到邮件列表，以便其他人也能从你所收到、并解决了问题的回应中受益。

1.7.1.4. 在邮件列表上回答问题的指南

如果你认为自己的解答会引起广泛关注，可以将其张贴到邮件列表，而不是直接回复请教的个人。尽量使你的解答具有普遍性，以便除初始发起人之外的其他人也能从中受益。将解答张贴到邮件列表时，请确认你的解答不是已有答案的复制品。

在你的回复中，应尽量归纳问题的基本部分，没有必要一定引用全部初始信息。

在要在打开HTML模式的情况下从浏览器张贴邮件信息。很多用户不使用浏览器来阅读邮件。

1.7.2. IRC（在线聊天系统）上的MySQL社区支持

除了各种MySQL邮件列表外，在IRC（在线聊天系统）上，也能发现有经验的社区成员。以下是目前我们已知的最好的网络／渠道：

- **Freenode**（请参见http://www.freenode.net/以查找服务器信息）

o #mysql，主要针对MySQL问题，也欢迎其他数据库和一般的SQL问题。与MySQL一起使用PHP、Perl或C方面的问题也很常见。

如果你正在寻找URC客户端软件，以便连接到IRC网络，请访问xChat（http://www.xchat.org/）。X-Chat（GPL许可）即能用于Unix平台，也适用于Windows平台（免费的面向Windows的X-Chat可从站点http://www.silverex.org/download/上下载）。

1.7.3. MySQL论坛上的MySQL社区支持

最新的社区支持资源是位于下述站点的论坛：http://forums.mysql.com。

有各种可用论坛，分为以下大类：

- 移植
- MySQL用法
- MySQL连接器
- 编程语言
- 工具
- 第三方应用程序
- 存储引擎
- MySQL技术
- SQL标准
- 业务

1.8. MySQL标准的兼容性

1.8.1. MySQL遵从的标准是什么
1.8.2. 选择SQL模式
1.8.3. 在ANSI模式下运行MySQL
1.8.4. MySQL对标准SQL的扩展
1.8.5. MySQL与标准SQL的差别

1.8.6. MySQL处理约束的方式

在本节中，介绍了MySQL与ANSI/ISO SQL标准的关系。MySQL服务器有很多对SQL标准的扩展之处，这里介绍了它们是什么，以及使用它们的方法。你也能了解关于MySQL服务器缺失功能的信息，以及如何处理某些差异的方法。

SQL标准自1986年以来不断演化发展，有数种版本。在本手册中，“SQL-92”指得是1992年发布的标准，“SQL:1999”指得是1999年发布的标准，“SQL:2003”指得是标准的当前版本。我们采用术语“SQL标准”标示SQL标准的当前版本。

我们的目标是在没有良好理由的情况下不限制MySQL服务器的可用性。即使我们没有足够的资源就每种可能的应用进行开发，我们始终愿意帮助那些在新领域使用MySQL服务器的人员，并向他们提供建议。

对于该产品，我们的一项主要目标是，继续致力于与SQL标准的兼容性，但不以牺牲速度和可靠性为代价。如果它们能显著增加拥有大量用户基数的MySQL服务器的可用性，我们无惧于为SQL添加扩展，也无惧于为非SQL特性提供支持。MySQL服务器4.0中的HANDLER接口即是该策略的例子。请参见13.2.3节，“HANDLER语法”。

我们将继续支持事务性和非事务性数据库，以满足任务关键型全天候应用，以及高负载Web或日志应用。

MySQL服务器最初是为小型计算机系统上中等规模的数据库设计的（100万-1亿行，或每个表的大小为100MB）。目前，MySQL服务器能处理TB级别的数据库，也能在针对便携式设备或嵌入式设备的精简版本中使用。MySQL服务器的精简设计使得双向开发成为可能，不会在源码树中产生任何冲突。

目前，我们并未定位于实时支持，虽说MySQL复制特性提供了强大的功能。

在众多第三方簇解决方案中均有数据库簇支持特性，自4.1.2版以来，对于我们所需的NDB簇技术集成方案，同样请参见第17章：*MySQL*簇。

我们也正着手在数据库服务器中提供XML支持。

1.8.1. MySQL遵从的标准是什么

我们致力于支持全套ANSI/ISO SQL标准，但不会以牺牲代码的速度和质量为代价。

ODBC级别0-3.51。

1.8.2. 选择SQL模式

MySQL服务器能够工作在不同的SQL模式下，并能针对不同的客户端以不同的方式应用这些模式。这样，应用程序就能对服务器操作进行量身定制以满足自己的需求。

这类模式定义了MySQL应支持的SQL语法，以及应该在数据上执行何种确认检查。这样，就能在众多不同的环境下、与其他数据库服务器一起更容易地使用MySQL。

可以使用“--sql-mode="modes""选项，通过启动**mysqld**来设置默认的SQL模式。从MySQL 4.1开始，也能在启动之后，使用ET [SESSION|GLOBAL] sql_mode='modes'语句，通过设置sql_mode变量更改模式。

关于设置服务器模式的更多信息，请参见5.3.2节，“SQL服务器模式”。

1.8.3. 在ANSI模式下运行MySQL

你可以使用“--ansi”启动选项，要求**mysqld**使用ANSI模式。请参见5.3.1节，“**mysqld**命令行选项”。

在ANSI模式下运行服务器与使用该选项启动它的效果一样（在一行上指定“--sql_mode”值）：

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

在MySQL 4.1中，能够用下述两条语句实现相同的效果（在一行上指定“sql_mode”值）：

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

请参见1.8.2节，“选择SQL模式”。

在MySQL 4.1.1中，也能用下述语句设置sql_mode选项：

```
SET GLOBAL sql_mode='ansi';
```

在本例中，将sql_mode变量的值设置为与ANSI模式相关的所有选项。你可以检查其结果，如下所示：

```
mysql> SET GLOBAL sql_mode='ansi';
mysql> SELECT @@global.sql_mode;
        -> 'REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,
            IGNORE_SPACE,ANSI';
```

1.8.4. MySQL对标准SQL的扩展

MySQL服务器包含一些其他SQL DBMS中不具备的扩展。注意，如果使用了它们，将无法把代码移植到其他SQL服务器。在某些情况下，你可以编写包含MySQL扩展的代码，但仍保持其可移植性，方法是用“/*... */”注释掉这些扩展。在本例中，MySQL服务器能够解析并执行注释中的代码，就像对待其他MySQL语句一样，但其他SQL服务器将忽略这些扩展。例如：

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

如果在字符“!”后添加了版本号，仅当MySQL的版本等于或高于指定的版本号时才会执行注释中的语法：

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

这意味着，如果你的版本号为3.23.02或更高，MySQL服务器将使用TEMPORARY关键字。

下面按类别介绍了各种MySQL扩展。

· 磁盘上的数据组织

MySQL服务器会将每个数据库映射到MySQL数据目录下的1个目录中，并将数据库中的表映射到数据库目录下的文件名。它具有下述含义：

o 如果操作系统的文件名区分大小写（如大多数Unix系统），当MySQL服务器运行在这类操作系统上时，数据库名和表名也区分大小写。请参见9.2.2节，“识别符大小写敏感性”。

o 你可以使用标准的系统命令来备份、重命名、移动、删除、并拷贝由MyISAM或ISAM存储引擎管理的表。例如，要想重命名MyISAM表，可重命名表对应的.MYD、.MYI、以及.frm文件。

数据库、表、索引、列或别名能够以数字开头（但或许不能全部由数字构成）。

· 通用语言语法

o 可以使用“”“”或“’”括住字符串，而不仅是“’”。

o 在字符串中使用“\”作为转义字符。

o 在SQL语句中，可以使用“*db_name.tbl_name*”语法访问不同数据库中的表。某些SQL服务器提供了相同的功能，但调用该用户空间除外。MySQL服务器不支持表空间，如下述语句中使用的那样： CREATE TABLE ralph.my_table...IN my_tablespace.

· SQL语句的语法

- ANALYZE TABLE，CHECK TABLE，OPTIMIZE TABLE，以及REPAIR TABLE语句。
- CREATE DATABASE和DROP DATABASE语句。请参见13.1.3节，“CREATE DATABASE语法”。
- DO语句。
- EXPLAIN SELECT获取如何联合表的介绍。
- FLUSH和RESET语句。
- SET语句。请参见13.5.3节，“SET语法”。
- SHOW语句。请参见13.5.4节，“SHOW语法”。
- 使用LOAD DATA INFILE。在很多情况下，该语法与Oracle的LOAD DATA INFILE兼容。请参见13.2.5节，“LOAD DATA INFILE语法”。
- RENAME TABLE的使用。请参见13.1.9节，“RENAME TABLE语法”。
- 使用REPLACE取代DELETE + INSERT。请参见13.2.6节，“REPLACE语法”。
- 在ALTER TABLE语句中使用CHANGE col_name、DROP col_name、或DROP INDEX、IGNORE或RENAME。在ALTER TABLE语句中使用多个ADD、ALTER、DROP或CHANGE子句。请参见13.1.2节，“ALTER TABLE语法”。
- 使用索引名，字段前缀上的索引，并在CREATE TABLE语句中使用INDEX或KEY。请参见13.1.5节，“CREATE TABLE语法”。
- 与CREATE TABLE一起使用TEMPORARY或IF NOT EXISTS。
- 与DROP TABLE一起使用IF EXISTS。
- 使用单个DROP TABLE语句，能够舍弃多个表。
- UPDATE和DELETE语句的ORDER BY和LIMIT子句。
- INSERT INTO ... SET col_name = ... syntax.
- INSERT和REPLACE语句的DELAYED子句。
- INSERT、REPLACE、DELETE和UPDATE语句的LOW_PRIORITY子句。
- 在SELECT语句中使用INTO OUTFILE和STRAIGHT_JOIN。请参见13.2.7节，“SELECT语法”。
- SELECT语句中的SQL_SMALL_RESULT选项。
- 不需要在GROUP BY部分命名所有选择的列。对于某些十分特殊但相当正常的查询，它能提供更好的性能。请参见12.10节，“与GROUP BY子句同时使用的函数和修改程序”。
- 可以与GROUP BY一起指定ASC和DESC。
- 能够在带有“:=”赋值操作符的语句中设置变量。

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

- 列类型
- 列类型MEDIUMINT、SET、ENUM、以及不同的BLOB和TEXT类型。

o 列属性AUTO_INCREMENT、BINARY、NULL、UNSIGNED以及ZEROFILL。

· 函数和操作符

o 为了使其他SQL环境下的用户更容易入手，MySQL服务器对很多函数均支持别名特性。例如，所有的字符串函数均支持标准SQL语法和ODBC语法。

o MySQL服务器能够理解"||"和"&&"操作符，将其当作逻辑OR和AND，就像在C编程语言中那样。在MySQL服务器中，||和OR是同义词，&&和AND也是同义词。由于采用了该优异的语法体系，MySQL服务器不支持SQL针对字符串连接的"||"操作符，而采用了CONCAT()取而代之。由于CONCAT()能够接受任意数目的参量，很容易将使用"||"操作符的情况转换为MySQL服务器支持的类型。

o 请在有多于1个元素的场合下使用COUNT(DISTINCT list)。

o 默认情况下，所有的字符串比较均区分大小写，其分类顺序由当前字符集确定（默认为cp1252 Latin1）。如果你不喜欢该点，应使用BINARY属性或BINARY cast声明列，这样，就会使用基本的字符代码值进行比较，而不是词汇顺序。

o "%"操作符等同于MOD()。也就是说"N % M"等同于MOD(N,M)。Cyuyan的程序员支持"%"，而且它也是为了兼容PostgreSQL而使用的。

o 在列比较中，可在SELECT语句的FROM左侧使用=、<>、<=、<、>=、>、<<、>>、<=>、AND、OR或LIKE操作符。例如：

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

o 返回最近AUTO_INCREMENT值的LAST_INSERT_ID()函数。请参见12.9.3节，"信息函数"。

o 允许在数值列上使用LIKE。

o REGEXP和NOT REGEXP扩展了常规的表达式操作符。

o 具有1个或2个以上参量的CONCAT()或CHAR()。（在MySQL服务器中，这些函数可以有任意数目的参量）。

o
BIT_COUNT()、CASE、ELT()、FROM_DAYS()、FORMAT()、IF()、PASSWORD()、ENCRYPT()、MD5()、ENCODE()、DECODE()、PERIOD_ADD()、PERIOD_DIFF()、TO_DAYS()、以及WEEKDAY()函数。

o 使用TRIM()来调整子字符串。标准SQL仅支持单个字符的删除。

GROUP BY函数STD()、BIT_OR()、BIT_AND()、BIT_XOR()、以及GROUP_CONCAT()。请参见12.10节，"与GROUP BY子句同时使用的函数和修改程序"。

1.8.5. MySQL与标准SQL的差别

1.8.5.1. 子查询
1.8.5.2. SELECT INTO TABLE
1.8.5.3. 事务和原子操作
1.8.5.4. 存储程序和触发程序
1.8.5.5. 外键
1.8.5.6. 视图
1.8.5.7. '--'作为注释起始标记

我们试图使MySQL服务器遵从ANSI SQL标准和ODBC SQL标准，但在某些情况下MySQL服务器执行的操作有所不同：

· 对于VARCHAR列，存储值时删除了尾部空间。（在MySQL 5.0.3中更正）。请参见A.8节，"MySQL中的已知事宜"。

· 在某些情况下，定义表或更改其结构时，将CHAR列转换为VARCHAR列。（在MySQL 5.0.3中更正）。请参见13.1.5.1节，"沉寂的列规格变更"。

· 删除表时，不自动取消关于表的权限。必须明确发出REVOKE语句，以撤销针对表的权限。请参见13.5.1.3节，"GRANT和REVOKE语法"。

· CAST()函数不支持对REAL或BIGINT的抛弃。请参见12.8节，"Cast函数和操作符"。

- 标准SQL要求，SELECT语句中的HAVING子句能够引用GROUP BY子句中的列。在MySQL 5.0.2之前，不能完成该功能。

1.8.5.1. 子查询

MySQL 4.1支持子查询和导出表。"子查询"指的是嵌套在另一语句中的SELECT语句。"导出表"（未命名视图）是另一语句的FROM子句中的子查询。请参见13.2.8节，"Subquery语法"。

从MySQL 4.1版起，可以使用联合或其他方法重写大多数子查询。关于如何完成该任务的更多信息，请参见13.2.8.11节，"对于较早的MySQL版本，采用联合方法重写子查询"。

1.8.5.2. SELECT INTO TABLE

MySQL服务器不支持Sybase SQL扩展： SELECT ... INTO TABLE ....。但MySQL服务器支持标准的SQL语法INSERT INTO ... SELECT ...，它基本上相同。请参见13.2.4.1节，"INSERT ... SELECT语法"。

```
INSERT INTO tbl_temp2 (fld_id)
    SELECT tbl_temp1.fld_order_id
    FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

作为备选方式，可以使用SELECT INTO OUTFILE ...或CREATE TABLE ... SELECT。

从5.0版开始，MySQL支持SELECT ... INTO，以及用户变量。在使用光标和局部变量的存储程序中也可以使用相同的语法。请参见20.2.9.3节，"SELECT ... INTO语句"。

1.8.5.3. 事务和原子操作

MySQL服务器（3.23至该系列的最高版本，所有4.0版本，以及更高版本）支持采用InnoDB和BDB事务存储引擎的事务。InnoDB提供了全面的ACID兼容性。请参见第15章：存储引擎和表类型。

MySQL服务器中的其他非事务性存储引擎（如MyISAM）遵从不同的数据完整性范例，称之为"原子操作"。按照事务术语，MyISAM表总能高效地工作在AUTOCOMMIT=1模式下。原子操作通常能提供可比较的完整性以及更好的性能。

由于MySQL服务器支持两种范例，因而你能决定是否利用原子操作的速度更好地服务于你的应用程序，或使用事务特性。该选择可按表进行。

正如所阐述的那样，事务性和非事务性表类型之间的权衡主要取决于性能。事务性表对内存和磁盘空间的要求更高，CPU开销也更大。另一方面，多种事务性表类型，如InnoDB，也能提供很多显著特性。MySQL服务器的模块化设计允许同时使用不同的存储引擎，以满足不同的要求，并在所有情形下，提供最佳性能。

但是，即便使用非事务性MyISAM表，你将如何使用MySQL服务器的特性来保持严格的完整性呢？这些特性与事务性表类型相比又如何呢？

1. 如果应用程序采用了特定的编写方式，依赖于在关键情况下能够调用ROLLBACK而不是COMMIT，那么事务性类型更方便。使用事务，还能确保未完成的更新或崩溃的活动不被提交到数据库，能为服务器提供自动回滚的机会，并保存你的数据库。

如果使用非事务性表，MySQL服务器几乎在所有情况下均允许你解决潜在的问题，方式是在更新前进行简单检查，并运行检查数据库一致性的简单脚本，如果出现不一致性，该脚本能自动修复它或给出告警。注意，仅使用MySQL日志或增加额外日志，通常能完美地更正表，同时不会造成数据完整性损失。

2. 在很多情况下，能够对关键的事务更新进行重写，使之成为"原子"类型。一般而言，所有由事务解决的完整性问题均能用LOCK TABLES或原子更新解决，从而确保了服务器不会自动中断，后者是事务性数据库系统的常见问题。

3. 为了安全使用MySQL服务器，无论是否使用事务性表，仅需启用备份和二进制日志功能。这样，你就能解决使用其他事务性数据库系统时遇到的任何问题。无论使用的数据库系统是什么，启用备份总是个好主意。

事务范型有自己的优点和不足之处。很多用户和应用程序开发人员喜欢这类简单性，在出现问题时或必要时，通过代码解决问题。但是，即使你是原子操作范型的新手，或更熟悉事务，也请考虑非事务性表的速度益处，与经过优化调整的最快的事务性表相比，它的速度快3～5倍。

在完整性具有最高重要性的情况下，即使是对非事务性表，MySQL也能提供事务级别的可靠性和安全性。如果使用LOCK TABLES锁定了表，所有更新均将被暂时中止直至完整性

检查完成。如果你获得了对某一表的READ LOCAL锁定（与写锁定相对），该表允许在表尾执行并行插入，当其他客户端执行插入操作时，允许执行读操作。新插入的记录不会被有读锁定属性的客户端看到，直至解除了该锁定为止。使用INSERT DELAYED，能够将插入项置于本地队列中，直至锁定解除，不会让客户端等待插入完成。请参见13.2.4.2节，“INSERT DELAYED语法”。

从我们赋与其名称的意义上，“原子”绝非不可思议的。它仅意味着，你能确信在每个特性更新运行的同时，其他用户不能干涉它，而且不会出现自动回滚（如果你不小心，对于事务性表，这种情况可能发生）。MySQL服务器还能保证不存在脏读。

下面列出了使用非事务性表的一些技术：

- 对于需要事务的循环，通常能使用LOCK TABLES进行编码，不需要光标来更新正在处理的记录。
- 要想避免使用ROLLBACK，可采取下述策略：

1. 使用LOCK TABLES锁定所有希望访问的表。
2. 执行更新前，测试必须为真的条件。
3. 如果一切正常，执行更新。
4. 使用UNLOCK TABLES解除锁定。

与使用具有回滚可能性的事务性表相比，它通常具有更快的速度，虽然并非始终如此。该解决方案唯一不能处理的情形是，在更新中途杀死了线程。在这种情况下，将释放所有锁定，但某些更新可能尚未执行。

- 也可以使用函数在单一操作中更新记录。采用下述技术，能获得效率很高的应用程序。

o 根据其当前值更改列。

o 仅更新出现实际变化的列。

例如，当我们更新某些客户信息时，仅更新已更改的客户数据，与原始行相比，仅测试已更改的数据或依赖于已更改数据的数据是否未出现变化。对于已更改数据的测试，它是通过UPDATE语句的WHERE子句完成的。如果记录未更新，将向客户端发出消息：“一些你改变的数据已被其他用户更改”。接下来，我们在窗口中给出了旧行和新行，以便用户决定使用哪个版本。

这给出了与列锁定类似的结果，但效果更好，使用相对于其当前值的值，仅更新了某些列。这意味着，典型的UPDATE语句与下面给出的类似：

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


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

它很有效，即使其他客户端更改了pay_back或money_owed_to_us列中的值，也能使用。

- 在很多情况下，用户希望将LOCK TABLES和／或ROLLBACK用于管理唯一ID。可以在不使用锁定功能或回滚的情况下，使用AUTO_INCREMENT列以及LAST_INSERT_ID() SQL函数或mysql_insert_id() C API函数，更有效地处理之。请参见12.9.3节，“信息函数”。请参见25.2.3.36节，“mysql_insert_id()”。

我们通常能使用代码来处理行级锁定方面的需求。在某些情况下，实际上不需要它，InnoDB表支持行级锁定。通过MyISAM表，能够在表中使用标志列，并完成类似下面的操作：

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

如果找到行，而且原始行中的row_flag不是1，对于受影响的行数，MySQL返回1。

你可以认为MySQL将前述查询更改为：

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

1.8.5.4. 存储程序和触发程序

对于MySQL，在5.0版本中实现了存储程序。请参见第20章：存储程序和函数。

从5.0.2版开始，在MySQL中实现了基本的触发器功能，计划在MySQL 5.1中进一步发展它。请参见第21章：触发程序。

1.8.5.5. 外键

在MySQL服务器3.23.44和更高版本中，InnoDB存储引擎支持对外键约束的检查功能，这些约束包括CASCADE、ON DELETE和ON UPDATE。请参见15.2.6.4节，“FOREIGN KEY约束”。

对于InnoDB之外的其他存储引擎，MySQL服务器能够解析CREATE TABLE语句中的FOREIGN KEY语法，但不能使用或保存它。未来将进行扩展，能够将这类信息保存到表规范文件中，以便能被**mysqldump**和ODBC检索。稍后，还将为MyISAM表实现外键约束。

外键增强为数据库开发人员提供了多项益处：

- 假定关联设计恰当，外键约束使得程序员更难将不一致性引入数据库。
- 数据库服务器具有集中式约束检查功能，因而没有必要在应用程序一侧执行这类检查。这样，就消除了不同应用程序使用不同方式检查约束的可能性。
- 使用级联更新和删除，简化了应用程序代码。
- 设计恰当的外键有助于以文档方式记录表间的关系。

请记住，这些好处是以数据库服务器为执行必要检查而需的额外开销为代价的。服务器额外检查会影响性能，对于某些应用程序，该特性不受欢迎，应尽量避免。（出于该原因，在一些主要的商业应用程序中，在应用程序级别上实施了外键逻辑）。

MySQL允许数据库开发人员选择要使用的方法。如果你不需要外键，并希望避免与强制引用完整性有关的开销，可选择另一种表类型取而代之，如MyISAM。（例如，MyISAM存储引擎为仅执行INSERT和SELECT操作的应用程序提供了极快的性能，这是因为插入能和检索同时进行）。请参见7.3.2节，“表锁定事宜”。

如果你不打算利用引用完整性检查具备的优点，请记住下述要点：

- 不存在服务器端外键关联检查时，应用程序本身必须处理这类关联事宜。例如，将行按恰当顺序插入表时应谨慎，并应避免产生孤立的子记录。必须能够在多记录插入操作期间更正出现的错误。
- 如果ON DELETE是应用程序所需的唯一引用完整性功能，请注意，从MySQL服务器4.0起，可以使用多表DELETE语句，用单一语句从多个表中删除行。请参见13.2.1节，“DELETE语法”。
- 从具有外键的表删除记录时，在缺少ON DELETE的情况下，一种解决方式是为应用程序增加恰当的DELETE语句。实际上，它与使用外键同样快，而且移植性更好。

注意，使用外键在某些情况下会导致问题。

- 外键支持能处理很多引用完整性事宜，但仍需要仔细设计键的关系，以避免循环规则或不正确的级联删除组合。
- DBA需要创建关联拓扑，这会使从备份中恢复单独表变得困难，该类情形并不罕见。（加载依赖其他表的表时，MySQL允许你临时禁止外键检查，从而降低了该难度）。请参见15.2.6.4节，“FOREIGN KEY约束”。在MySQL 4.1.1以前。重新加载时，**mysqldump**能够生成自动利用该性能的转储文件。

注意，SQL中的外键用于检查和强制引用完整性，而不是联合表。如果打算用SELECT语句获取多个表的结果，可在表之间执行联合操作：

```
SELECT * FROM t1, t2 WHERE t1.id = t2.id;
```

请参见13.2.7.1节，“JOIN语法”。请参见3.6.6节，“使用外键”。

ODBC应用程序常使用不带“ON DELETE ...”的FOREIGN KEY语法来生成自动WHERE子句。

1.8.5.6. 视图

在MySQL服务器5.0版中实现了视图功能（包括可更新视图）。在5.0.1和更高版本中，提供了二进制版的视图功能。请参见第22章：视图。

View（视图）十分有用，它允许用户像单个表那样访问一组关系（表），而且仅允许对它们的这类访问。视图也能限制对行的访问（特定表的子集）。对于列控制的访问，可使用MySQL服务器中的高级权限系统。请参见5.7节，“MySQL访问权限系统”。

在设计视图的过程中，我们的宏伟目标是，在SQL的范围内尽可能与关联数据库系统的“Codd's Rule #6”兼容。“所有理论上可更新的视图，实际上也应是可更新的”。

1.8.5.7. '--'作为注释起始标记

一些其他SQL数据库采用“--”作为注释开始标志。MySQL服务器采用“#”作为注释起始字符。对于MySQL服务器，也能使用C风格的注释：/*该处为注释*/。请参见9.5节，“注释语法”。

MySQL服务器3.23.3和更高版本支持“--”注释风格，但要求注释后面跟1空格（或控制字符，如新行）。之所以要求使用空格，是为了防止与自动生成SQL查询有关的问题，它采用了类似下面的代码，其中，自动为“!payment!”插入“payment”的值：

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

考虑一下，如果“payment”的值为负数如“-1”时会出现什么情况：

```
UPDATE account SET credit=credit--1
```

在SQL中“credit--1”是合法的表达式，但是，如果“--1”被解释为注释开始，部分表达式将被舍弃。其结果是，表达式的意义与预期的意义完全不同。

```
UPDATE account SET credit=credit
```

该语句不会对值作任何更改！这表明，允许注释以“--”开始会产生严重后果。

采用MySQL服务器3.23.3和更高版本中的这类注释方法，“credit--1”实际上很安全。

另一个安全特性是，**mysql**命令行客户端将删除所有以“--”开头的行。

仅当使用高于3.23.3的MySQL时，下述信息才有意义：

如果有1个文本文件形式的SQL程序，该文件包含“--”注释，应按下述方式使用**replace**实用工具，将其转换为使用“#”字符的注释：

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

而不是通常的：

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

你也可以编辑注释文件，将“--”注释更改为“#”注释：

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

使用下述命令将其改回去：

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

1.8.6. MySQL处理约束的方式

1.8.6.1. PRIMARY KEY和UNIQUE索引约束
1.8.6.2. 对无效数据的约束
1.8.6.3. ENUM和SET约束

使用MySQL，你可以使用允许回滚的事务表，以及不允许回滚的非事务表。因此，在MySQL中的约束处理功能与其他DBMS中的略有不同。在非事务性表中插入或更新大量行时，当出现错误以至于不能回滚所作的变更时，必须处理该情况。

其基本原理在于，在解析将要执行的语句的同时，MySQL服务器会尽量为检测到的问题生成错误信息，并会在执行语句的同时尽量恢复出现的错误。在大多数情况下我们均是这样作的，但不包括所有情况。

出现错误时，MySQL可选择中途中止语句，或尽可能恢复并继续执行语句。默认情况下，服务器将采取后一种路线。这意味着，服务器可能会强制将非法值变为最接近的合法值（例如）。

从MySQL 5.0.2开始，提供了数种SQL模式，使用它们，能够对如何接受可能为不良数据值的方式进行更好的控制，也能在出现错误时，对是否继续执行语句或放弃语句进行控制。使用这些选项，能够将MySQL服务器配置为更为传统的风格，类似于拒绝不恰当输入的其他DBMS。可以在运行时设置SQL模式，这样，各客户端就能选择与其需求最为贴切的行为。请参见5.3.2节，“SQL服务器模式”。

在以下部分，介绍了使用不同约束类型的情况。

1.8.6.1. PRIMARY KEY和UNIQUE索引约束

通常情况下，当你试图INSERT或UPDATE会导致主键、唯一键或外键冲突的行时，将出现错误。如果你正在使用事务性存储引擎时，如InnoDB，MySQL会自动回滚语句。如果你正在使用非事务性存储引擎，MySQL将在出错的行上停止执行语句，剩余的行也不再处理。

如果你希望忽略这类键冲突，可使用MySQL支持的、用于INSERT和UPDATE的IGNORE关键字。在这种情况下，MySQL将忽略任何键冲突，并继续处理下一行。请参见13.2.4节，“INSERT语法”。请参见3.2.10节，“UPDATE语法”。

使用mysql_info() C API函数，能够获取关于实际插入或更新行数的信息。请参见25.2.3.34节，“mysql_info()”。在MySQL 4.1和更高版本中，也能使用SHOW WARNINGS语句。请参见13.5.4.22节，“SHOW WARNINGS语法”。

目前，只有InnoDB表支持外键。请参见15.2.6.4节，“FOREIGN KEY约束”。计划在MySQL 5.1中实施对MyISAM表的外键支持。

1.8.6.2. 对无效数据的约束

在MySQL 5.0.2之前，MySQL对非法或不当值并不严厉，而且为了数据输入还会强制将它们变为合法值。在MySQL 5.0.2和更高版本中，保留了以前的默认行为，但你可以为不良值选择更传统的处理方法，从而使得服务器能够拒绝并放弃出现不良值的语句。本节介绍了MySQL的默认行为（宽大行为），新的严格的SQL模式，以及它们的区别。

如果你未使用严格模式，下述情况是真实的。如果将“不正确”的值插入到列，如将NULL值插入非NULL列，或将过大的数值插入数值列，MySQL会将这些列设置为“最可能的值”，而不是生成错误信息。

- 如果试图将超范围的值保存到数值列，MySQL服务器将保存0（最小的可能值）取而代之，或最大的可能值。
- 对于字符串，MySQL或保存空字符串，或将字符串尽可能多的部分保存到列中。
- 如果打算将不是以数值开头的字符串保存到数值列，MySQL将保存0。
- MySQL允许将特定的不正确日期值保存到DATE和DATETIME列（如“2000-02-31”或“2000-02-00”）。其观点在于，验证日期不是SQL服务器的任务。如果MySQL能保存日期值并准确检索相同的值，MySQL就能按给定的值保存它。如果日期完全不正确（超出服务器能保存的范围）将在列中保存特殊的日期值“0000-00-00”取而代之。

· 如果试图将NULL值保存到不接受NULL值的列，对于单行INSERT语句，将出现错误。对于多行INSERT语句或INSERT INTO ... SELECT语句，MySQL服务器会保存针对列数据类型的隐含默认值。一般情况下，对于数值类型，它是0，对于字符串类型，它是空字符串(")，对于日期和时间类型是"zero"。在13.1.5节，"CREATE TABLE语法"一节中，讨论了隐含的默认值。

· 如果INSERT语句未为列指定值，如果列定义包含明确的DEFAULT子句，MySQL将插入默认值。如果在定义中没有这类DEFAULT子句，MySQL会插入列数据类型的隐含默认值。

采用前述规则的原因在于，在语句开始执行前，无法检查这些状况。如果在更新了数行后遇到这类问题，我们不能仅靠回滚解决，这是因为存储引擎可能不支持回滚。中止语句并不是良好的选择，在该情况下，更新完成了"一半"，这或许是最差的情况。对于本例，较好的方法是"仅可能做到最好"，然后就像什么都未发生那样继续。

在MySQL 5.0.2和更高版本中，可以使用STRICT_TRANS_TABLES或STRICT_ALL_TABLES SQL模式，选择更严格的处理方式。请参见5.3.2节，"SQL服务器模式"。

STRICT_TRANS_TABLES的工作方式：

· 对于事务性存储引擎，在语句中任何地方出现的不良数据值均会导致放弃语句并执行回滚。

· 对于非事务性存储引擎，如果错误出现在要插入或更新的第1行，将放弃语句。（在这种情况下，可以认为语句未改变表，就像事务表一样）。首行后出现的错误不会导致放弃语句。取而代之的是，将调整不良数据值，并给出告警，而不是错误。换句话讲，使用STRICT_TRANS_TABLES后，错误值会导致MySQL执行回滚操作，如果可以，所有更新到此为止。

要想执行更严格的检查，请启用STRICT_ALL_TABLES。除了非事务性存储引擎，它与STRICT_TRANS_TABLES等同，即使当不良数据出现在首行后的其他行，所产生的错误也会导致放弃语句。这意味着，如果错误出现在非事务性表多行插入或更新过程的中途，仅更新部分结果。前面的行将完成插入或更新，但错误出现点后面的行则不然。对于非事务性表，为了避免这种情况的发生，可使用单行语句，或者在能接受转换警告而不是错误的情况下使用STRICT_TRANS_TABLES。要想在第1场合防止问题的出现，不要使用MySQL来检查列的内容。最安全的方式（通常也较快）是，让应用程序负责，仅将有效值传递给数据库。

有了严格的模式选项后，可使用INSERT IGNORE或UPDATE IGNORE而不是不带IGNORE的INSERT或UPDATE，将错误当作告警对待。

1.8.6.3. ENUM和SET约束

ENUM和SET列提供了定义仅能包含给定值集合的列的有效方式。但是，从MySQL 5.0.2起，ENUM和SET不是实际约束。其原因与不重视NOT NULL的原因一样。请参见1.8.6.2节，"对无效数据的约束"。

ENUM列总有1个默认值。如果未指定默认值，对于包含NULL的列，默认值为NULL；否则，第1个枚举值将被当作默认值。

如果在ENUM列中插入了不正确的值，或者，如果使用IGNORE将值强制插入了ENUM列，会将其设置为保留的枚举值0，对于字符串情形，将显示为空字符串。请参见11.4.4节，"ENUM类型"。

如果在SET列中插入了不正确值，该值将被忽略。例如，如果列能包含值"a"、"b"和"c"，并赋值"a,x,b,y"，结果为"a,b"。请参见11.4.5节，"SET类型"。

从5.0.2开始，可以对服务器进行配置，以使用严格的SQL模式。请参见5.3.2节，"SQL服务器模式"。启用严格模式后，ENUM或SET列的定义可作为对输入至列的值的约束。如果值不满足下述条件，将出现错误：

· ENUM值必须是在列定义中给出的值之一，或内部的数字等同物。该值不能是错误值（即，0或空字符串）。对于定义为ENUM('a','b','c')的列，诸如''、'd'和'ax'等，均是非法的，并将被拒。

· SET值必须是空字符串，或由1个或多个在列定义中给出的且用逗号隔开的值组成。 对于定义为SET('a','b','c')的列，诸如'd'和'a,b,c,d'等，均是非法的，并将被拒。

如果使用了INSERT IGNORE或UPDATE IGNORE，在严格模式下，可抑制无效值导致的错误。在这种情况下，将生成警告而不是错误。对于ENUM，值将作为错误成员(0)插入。对于SET，会将给定值插入，但无效的子字符串将被删除。例如，'a,x,b,y'的结果是'a,b'，就像前面介绍的那样。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第2章：安装MySQL

目录

2.1. 一般安装问题

- 2.1.1. MySQL支持的操作系统
- 2.1.2. 选择要安装的MySQL分发版
- 2.1.3. 怎样获得MySQL
- 2.1.4. 通过MD5校验和或GnuPG验证软件包的完整性
- 2.1.5. 安装布局

2.2. 使用二进制分发版的标准MySQL安装
2.3. 在Windows上安装MySQL

- 2.3.1. Windows系统要求
- 2.3.2. 选择安装软件包
- 2.3.3. 用自动安装器安装MySQL
- 2.3.4. 使用MySQL安装向导
- 2.3.5. 使用配置向导
- 2.3.6. 通过非安装Zip文件安装MySQL
- 2.3.7. 提取安装档案文件
- 2.3.8. 创建选项文件
- 2.3.9. 选择MySQL服务器类型
- 2.3.10. 首次启动服务器
- 2.3.11. 从Windows命令行启动MySQL
- 2.3.12. 以Windows服务方式启动MySQL
- 2.3.13. 测试MySQL安装
- 2.3.14. 在Windows环境下对MySQL安装的故障诊断与排除
- 2.3.15. 在Windows下升级MySQL
- 2.3.16. Windows版MySQL同Unix版MySQL对比

2.4. 在Linux下安装MySQL
2.5.在Mac OS X中安装MySQL
2.6. 在NetWare中安装MySQL
2.7. 在其它类Unix系统中安装MySQL
2.8. 使用源码分发版安装MySQL

- 2.8.1. 源码安装概述
- 2.8.2. 典型配置选项
- 2.8.3. 从开发源码树安装
- 2.8.4. 处理MySQL编译问题
- 2.8.5. MIT-pthreads注意事项
- 2.8.6. 在Windows下从源码安装MySQL
- 2.8.7. 在Windows下编译MySQL客户端

2.9. 安装后的设置和测试

- 2.9.1. Windows下安装后的过程
- 2.9.2. Unix下安装后的过程
- 2.9.3. 使初始MySQL账户安全

2.10. 升级MySQL

- 2.10.1. 从5.0版升级
- 2.10.2. 升级授权表
- 2.10.3. 将MySQL数据库拷贝到另一台机器

2.11. 降级MySQL
2.12. 具体操作系统相关的注意事项

- 2.12.1. Linux注意事项
- 2.12.2. Mac OS X注意事项

2.12.3. Solaris注意事项
2.12.4. BSD注意事项
2.12.5. 其它Unix注意事项
2.12.6. OS/2注意事项

2.13. Perl安装注意事项

2.13.1. 在Unix中安装Perl
2.13.2. 在Windows下安装ActiveState Perl
2.13.3. 使用Perl DBI/DBD接口的问题

本章描述怎样获得并安装**MySQL**：

1. 要了解支持哪些平台。请注意并非所有支持的平台同等适合运行MySQL。在部分平台上会比其它平台更有效、稳定。详情请参见2.1.1节，“MySQL支持的操作系统”。

2. 确定你应该使用的分发的版本。可获得**MySQL**的多个版本，大部分有多种分发版。你可以选择包含二进制（预编译）代码或源码形式的预打包分发。如果不能确定，则使用二进制分发版。对于想看我们的最新开发并帮助我们测试新代码的用户，我们提供了公共访问接口，可以访问我们最新的源码树。为了确定你应该使用的分发的版本和类型，请参见2.1.2节，“选择要安装的MySQL分发版”。

3. 下载想安装的分发版。对于能从其获得**MySQL**的站点列表，请参见2.1.3节，“怎样获得MySQL”。用来验证分发版完整性的指令参见2.1.4节，“通过MD5校验和或GnuPG验证软件包的完整性”。

4. 安装分发版。针对二进制代码和源码分发版的安装说明在2.2节，“使用二进制分发版的标准MySQL安装”中讲述。针对二进制代码和源码分发版的安装说明在2.8节，“使用源码分发版安装MySQL”中讲述。

注释：如果你想要将当前版本的MySQL升级到最新版本，而不是首次安装MySQL，关于升级过程和升级前应考虑的问题请参见2.10节，“升级MySQL”。

如果遇到安装问题，2.12节，“具体操作系统相关的注意事项”中提供了解决平台相关问题的信息。

5. 进行安装后的设置。安装MySQL后，请阅读2.9节，“安装后的设置和测试”。本节介绍了重要的信息，可以确保MySQL服务器正确工作。本节还描述了如何使尚未有密码的初始MySQL用户账户的安全。不管你是用二进制代码还是源码分发版来安装**MySQL**，这些过程均适用。

如果要运行MySQL基准脚本，必须使用支持MySQL的Perl。请参见2.13节，“Perl安装注意事项”。

2.1. 一般安装问题

2.1.1. MySQL支持的操作系统
2.1.2. 选择要安装的MySQL分发版
2.1.3. 怎样获得MySQL
2.1.4. 通过MD5校验和或GnuPG验证软件包的完整性
2.1.5. 安装布局

在安装MySQL前，你应该：

1. 确定是否MySQL将运行在你的平台上。

2. 选择安装的分发版。

3. 下载分发版并验证其完整性。

本节包含执行上述步骤必需的信息。之后，你可以按照本章中后面几节的说明来安装选择的分发版。

2.1.1. MySQL支持的操作系统

本节列出了可以运行MySQL的操作系统。

我们使用GNU Autoconf，因此将**MySQL**移植到所有使用Posix线程和C++编译器的现代系统是可能的。(要求服务器支持线程。如果只是编译客户端代码，则只需要C++编译器)。我们主要在Linux(SuSE和Red Hat)、FreeBSD和Sun Solaris(版本8和9)上使用并开发本软件。

已经报告**MySQL**可以在下列操作系统/线程包的组合上成功地进行编译。注意，对于很多操作系统，原生的线程仅支持最新的版本。

- 有原生线程的AIX 4.x和5.x。请参见2.12.5.3节，“IBM-AIX注意事项”。
- Amiga。

- 包括MIT-pthreads包的BSDI 2.x。请参见2.12.4.4节，“BSD/OS 2.x版注意事项”。
- 有原生线程的BSDI3.0,3.1和4.x。请参见2.12.4.4节，“BSD/OS 2.x版注意事项”。
- 有原生线程的Digital Unix 4.x。请参见2.12.5.5节，“Alpha-DEC-UNIX(Tru64)版注意事项”.。
- 包括MIT-pthreads包的FreeBSD 2.x。请参见2.12.4.1节，“FreeBSD注意事项”。
- 有原生线程的FreeBSD 3.x和4.x。请参见2.12.4.1节，“FreeBSD注意事项”。
- 有LinuxThreads的FreeBSD 4.x。请参见2.12.4.1节，“FreeBSD注意事项”。
- 有DCE线程或MIT-pthreads包的HP-UX 10.20。请参见2.12.5.1节，“HP-UX 10.20版注意事项”。
- 有原生线程的HP-UX 11.x。请参见2.12.5.2节，“HP-UX 11.x版注意事项”。
- 有LinuxThreads 0.7.1+或glibc 2.0.7+，适合各种CPU的Linux 2.0+。请参见2.12.1节，“Linux注意事项”。
- Mac OS X。请参见2.12.2节，“Mac OS X注意事项”。
- NetBSD 1.3/1.4 Intel和NetBSD 1.3 Alpha(需要GNU make)。请参见2.12.4.2节，“NetBSD注意事项”。
- Novell NetWare 6.0。请参见2.6节，“在NetWare中安装MySQL”。
- 有原生线程的OpenBSD > 2.5。包括MIT-pthreads包的OpenBSD < 2.5。请参见2.12.4.3节，“OpenBSD 2.5版注意事项”。
- OS/2 Warp 3, FixPack 29和OS/2 Warp 4, FixPack 4. 参见2.12.6节，“OS/2注意事项”。
- 有最新FSU Pthreads移植包的SCO OpenServer 5.0.X。请参见2.12.5.8节，“SCO UNIX和OpenServer 5.0.x版注意事项”。
- SCO UnixWare 7.1.x。请参见2.12.5.9节，“SCO UnixWare 7.1.x和OpenUNIX 8.0.0版注意事项”。
- SCO Openserver 6.0.x。请参见2.12.5.10节，“SCO OpenServer 6.0.x版注意事项”。
- 有原生线程的SGI Irix 6.x。请参见2.12.5.7节，“SGI Irix注意事项”。
- SPARC和x86上有原生线程的Solaris 2.5和以上版本。请参见2.12.3节，“Solaris注意事项”。
- 包括MIT-pthreads包的SunOS 4.x。请参见2.12.3节，“Solaris注意事项”。
- Tru64 Unix。请参见2.12.5.5节，“Alpha-DEC-UNIX(Tru64)版注意事项”。
- Windows 9x、Me、NT、2000、XP和2003。请参见2.3节，“在Windows上安装MySQL”。

并非所有支持的平台同等适合运行MySQL。根据以下因素确定某个平台适合高负荷关键任务MySQL服务器的程度:

- 线程库的稳定性。可能某个平台的反应不错，但MySQL的稳定性取决于它所调用的线程库，不管其它部分是否完善。
- 内核和线程库利用均衡多处理器(SMP)系统的能力。换句话说，当进程创建一个线程时，应允许线程运行原进程所运行CPU之外的其它处理器。
- 内核和线程库运行在短范围频繁获取和释放互斥体而不需要过多地交换内容的多个线程的能力。如果执行pthread_mutex_lock()时产生的CPU时间太短，则会严重影响MySQL。如果不关注该问题，增加过多的CPU实际会降低MySQL的速度。
- 常规文件系统的稳定性和性能。
- 如果数据库表很大，文件系统处理大文件的能力和处理效率。
- 我们在该平台上使用MySQL AB的经验水平。如果我们熟悉一个平台，我们可以对具体平台进行优化并固定编译时间。我们还可以提供建议如何更好地为MySQL配置系统。
- 我们在类似配置下所完成的内部测试的数量。
- 在同一平台类似配置下成功运行MySQL的用户数。如果该数字很高，则说明遇到具体平台问题的几率要小得多。

根据前面的标准，允许MySQL的最好的平台是x86,安装SuSE Linux，使用2.4或2.6内核，和ReiserFS(或类似Linux分发版)和安装了 Solaris(2.7-9)的SPARC。FreeBSD排第三位，但我们真正希望一旦线程库得到改进，它也可以提高排名。从某一观点我们还希望提高其它MySQL目前正在上面编译、允许的平台的排名，但稳定性和性能上可以要求不同等级。这需要我们与MySQL所依赖的操作系统和库组件开发人员共同努力。如果你对改进某个组件感兴趣，可能影响其开发，需要更多的关于MySQL如何能运行得更好的说明，请向我们的MySQL internals发送邮件。请参见1.7.1.1节，“The MySQL邮件列表”。

请注意前面的对比并不是说一个操作系统总体上比另一个操作系统要好。我们只是讨论选择一个OS来专门运行MySQL。记住了这一点，如果考虑更多的因素，对比结果可能会不同。在某些情况下，一个OS比另一个OS好的原因只是我们已经为具体平台做出了更多的努力来进行测试和优化。我们只是陈述我们的观点帮助你选择运行MySQL的平台。

2.1.2. 选择要安装的MySQL分发版

2.1.2.1. 选择要安装的MySQL版本
2.1.2.2. 选择分发格式
2.1.2.3. 如何发布更新以及何时发布更新
2.1.2.4. 发布原则：发行版本中没有已知的缺陷
2.1.2.5. MySQL AB编译的MySQL二进制版本

准备安装MySQL时，你应确定使用哪个版本。MySQL的开发有几个发布系列，可以选择最适合你要求的一个版本。确定了安装的版本后，便可以选择分发版。有二进制或源码发布格式。

2.1.2.1. 选择要安装的MySQL版本

首先要作出决策，你是否想要使用最新的开发版本或最终的稳定版本。在MySQL开发过程中，同时存在多个发布系列，每个发布处在成熟度的不同阶段：

- MySQL 5.2是最新开发的发布系列，是将执行新功能的系列。不久的将来可以使用Alpha发行，以便感兴趣的用户进行广泛的测试。
- MySQL 5.1是当前稳定(产品质量)发布系列。只针对漏洞修复重新发布；没有增加会影响稳定性的新功能。
- MySQL 5.0是前一稳定(产品质量)发布系列。只针对严重漏洞修复和安全修复重新发布；没有增加会影响该系列的重要功能。
- MySQL 4.0和3.23是旧的稳定(产品质量)发布系列。该版本不再使用，新的发布只用来修复特别严重的漏洞（以前的安全问题）。

我们不认为有完全的冻结版，因为任何版本均需要对漏洞进行修复和其它修复。对于"某种程度的冻结"，我们是指我们可以在产品发布中增加一些不会影响当前工作的小东西。自然，前一系列的相关漏洞修复会移植到后面的系列。

通常, 如果你是第一次开始使用**MySQL**或想要将它移植到一些还没有二进制分发版的系统上，我们推荐使用最终的稳定版本。目前是MySQL 5.1。我们已经使用MySQL基准对所有MySQL发布进行了检查，包括来自开发系列的发布，在发布前还使用基准套件来测试它。

如果你正在运行一个老的系统并且想要升级，但是又不想冒险进行非无缝升级，应该升级到最新版本中你正使用的相同的发布系列（只有版本号的最后部分比你使用的新）。我们已经试着仅修复致命缺陷，对那个版本进行了很小的相对安全的改动。

如果你想要使用产品发布系列中所没有的新功能，你可以使用开发系列的版本。请注意开发发布不如产品发布稳定。

如果你想要使用包含所有最新补丁和漏洞修复的最新源码，可以使用我们的BitKeeper库。这些库还没有发布，但可以预览代码，将来的发布基于这些代码。

MySQL的命名机制使用由3个数字和一个后缀组成的版本号。例如，像**mysql-5.0.9-beta**的版本号这样解释：

- 第1个数字(**5**)是主版本号，描述了文件格式。所有版本5的发行都有相同的文件格式。
- 第2个数字(**0**)是发行级别。主版本号和发行级别组合到一起便构成了发行序列号。
- 第3个数字(**9**)是在此发行系列的版本号，随每个新分发版递增。通常你需要已经选择的发行(release)的最新版本(版本)。

每次更新后，版本字符串的最后一个数字递增。如果相对于前一个版本增加了新功能或有微小的不兼容性，字符串的第二个数字递增。如果文件格式改变，第一个数字递增。

后缀显示发行的稳定性级别。通过一系列后缀显示如何改进稳定性。可能的后缀有：

- **alpha**表明发行包含大量未被彻底测试的新代码。已知的缺陷应该在新闻小节被记录。请参见附录D：*MySQL*变更史。在大多数alpha版本中也有新的命令和扩展。alpha版本也可能有主要代码更改等开发。但我们在发布前一定对其进行测试。
- **beta**意味着该版本功能是完整的，并且所有的新代码被测试了，没有增加重要的新特征，应该没有已知的缺陷。当alpha版本至少一个月没有出现报导的致命漏洞，并且没有计划增加导致已经实施的功能不稳定的新功能时，版本则从alpha版变为**beta**版。

在以后的**beta**版、发布版或产品发布中，所有API、外部可视结构和SQL命令列均不再更改。

- **rc**是发布代表；是一个发行了一段时间的beta版本，看起来应该运行正常。只增加了很小的修复。(发布代表即以前所称的gamma 版)
- 如果没有后缀，这意味着该版本已经在很多地方运行一段时间了，而且没有非平台特定的缺陷报告。只增加了关键漏洞修复修复。这就是我们称为一个产品（稳定）或"通用"版本的东西。

MySQL的命名机制于其它产品稍有不同。一般情况，我们可以很放心地使用已经投放市场两周而没有被相同发布系列的新版本所代替的版本。

MySQL所有版本都经过我们的标准测试和基准测试运行，以保证它们可相当安全地使用。因为标准测试不断扩充以检测以前发现的缺陷，测试套件一直在改进之中。

所有版本都至少已经用下列套件进行了测试：

- 一个内部测试套件

mysql-test目录包含一整套测试案例。我们针对每个二进制服务器进行这些测试。关于该测试套件的详细信息参见27.1.2节，“MySQL测试套件”。

- MySQL基准套件

它运行一定范围的普通查询。它也是一个测试，检测最新的优化处理是否真的使代码更快。请参见7.1.4节，“MySQL基准套件”。

- crash-me测试

这尝试决定数据库支持什么功能和它的能力与限制是什么。请参见7.1.4节，“MySQL基准套件”。

另一个测试是在内部生产环境中使用最新**MySQL**版本，至少在一台机器上。我们有超过100GB的数据可供使用。

2.1.2.2. 选择分发格式

选择了MySQL的安装版本后，要做的第二项决策是你是使用源码分发版还是二进制分发版。大多数情况，如果你的平台上已经有了一个二进制分发版，你可能使用二进制分发版。大多数平台可以使用原格式二进制分发版，例如Linux使用RPM文件，Mac OS X使用DMG安装软件包。也可以使用Zip文件或压缩**tar**文件。

选择二进制分发版的理由：

- 二进制分发版比源码分发版更容易安装。
- 为了满足不同用户的需求，我们提供了两种不同的二进制版本：一个是编译过的带非事务储存引擎(小的快速库)，另一个配置了最重要的扩展选项，例如事务安全表。两个版本均从相同的源码分发版编译而来。所有本地MySQL客户端均可以连接任一版本的MySQL服务器。

扩展MySQL库分发版标有-max后缀，配置了与**mysqld-max**相同的选项。请参见5.1.2节，“mysq**ld-max**扩展**My**SQL服务器”。

如果想要使用MySQL-Max RPM，必须首先安装标准MySQL-server RPM。

在某些环境下，最好安装源码分发版MySQL：

- 你想要在某个明显的位置安装MySQL。标准二进制分发版可以安装到任何地方，但你想更加灵活地将MySQL组件放到某个地方。
- 你想要将**mysqld**配置一些标准二进制分发版中未包括的一些特殊特征。下面是一些你想要使用的最常用的选项：
  - --with-innodb(在所有MySQL 5.1库版本中默认使用)
  - --with-berkeley-db(只适合部分平台)
  - --with-libwrap
  - --with-名d-z-libs(适合部分二进制)
  - --with-debug[=full]
- 你想要将**mysqld**配置一些标准二进制分发版中已经包括的一些特殊特征。例如，一般编译过的分发版支持所有字符集。如果你想要一个更小的MySQL服务器，可以重新编译只支持你需要的字符集。
- 你有一个专用编译器(例如pgcc)或想要使用更好地优化适用你的处理器的编译器选项。二进制分发版编译的选项适合同一处理器族的各种处理器。
- 你想要使用BitKeeper库中的最新源码，以访问所有最新漏洞修复。例如，如果你发现了漏洞并将它报告给MySQL开发小组，漏洞修复传递给源码库，你便可以访问它。发布实际发行后才会出现漏洞修复。
- 如果你想要阅读(或修改)构成**MySQL**的C和C++代码，你应该拿到源码分发版。源码总是最终的手册。
- 源码分发版也比二进制的分发版包含更多的测试和实例。

2.1.2.3. 如何发布更新以及何时发布更新

MySQL进展的相当快，我们想要与其它**MySQL**用户分享它。当我们有一个看来其它人似乎需要的非常有用的功能时，我们就试着制作一个发行版本。

我们也尝试帮助那些需要很容易实现的功能的用户，我们也关注我们授权的用户想要什么，我们更特别关注我们支持的客户想要什么，并且尽力帮助他们。

没有人一定要下载一个新版本，新闻小节中将告诉你新版本是否有一些你确实想要的东西。请参见附录D：*MySQL*变更史。

当更新**MySQL**时，我们使用下列策略：

- 将发布每个系列。每次发布时，版本的最后一个数字为同一系列前一版本的基础上加1。
- 稳定的测试过的产品版本每年准备出现1-2次，但是如果发现小缺陷，只有缺陷修复的一个版本将被发行。
- 工作版本/对旧版本的漏洞修复文件每4-8周出现一次。
- 对一些平台的二进制分发版，主要版本由我们制作。其它人可以为其它系统制作二进制分发版，但是可能不多见。
- 我们一旦识别并修正了非紧急但烦人的缺陷，则立即将其发布，可以从公共BitKeeper库中立即得到修复代码，并将包含到下一发布版中。
- 在任何情况下，如果在一个发行中有一个致命缺陷，我们将尽快制作一个新版本来修复。（我们希望其它公司也这样做！）

2.1.2.4. 发布原则：发行版本中没有已知的缺陷

我们投入了大量的时间和工作来使我们的发布没有缺陷。我们从来没有发布过含有已知致命重复性缺陷的单一MySQL版本。(“致命”缺陷指能在正常应用中导致MySQL瘫痪的缺陷，对于正常查询产生错误答案，或有安全问题）。

我们已经将所有公开问题、缺陷和由设计决策决定的事宜记入文件。请参见A.8节，“MySQL中的已知事宜”。

我们的目标是修复一切可以修复的东西，而不会使稳定的MySQL版本变得不稳定。在某些情况，这意味着我们可以在开发版本中修复问题，而不是在稳定的 (产品) 版本。自然，我们会将这些问题记入文档，以便用户能知道。

下面描述了我们如何操作：

- 我们通过我们的客户支持列表、在http://bugs.mysql.com/ 缺陷数据库和MySQL外部邮件列表来监控缺陷。
- 当前版本中所有被报导的缺陷被输入缺陷数据库。
- 当我们修复缺陷，我们总是为其设计一次测试案例，并将其包括进测试系统中，以确保不会漏检使缺陷再现。(所有修复的缺陷中大约90%的具有测试案例）。
- 为添加到MySQL中的所有新功能创建测试案例。
- 我们开始构建新的MySQL发布前，我们确保修复了MySQL版本(3.23.x、4.0.x、4.1.x、5.0.x等等)中所有被报导的重复性缺陷。如果某些内容不能修复(由于MySQL内部的一些设计决策)，我们在本手册中记录下来。请参见A.8节，“MySQL中的已知事宜”。
- 我们在所有支持二进制的平台(15+平台)上构建并运行我们的测试套件和基准套件。
- 如果在某个平台上测试或基准套件失败，我们不会公布二进制。如果问题是由于源码中的普通缺陷，我们将进行修复并在所有系统上构建并测试。
- 构建和测试过程需要2-3天。如果在该过程中我们收到致命缺陷相关报告(例如，会造成内核转储)，我们将修复该问题并重新启动构建过程。
- 在http：//dev.mysql.com/上公布二进制后，我们则向mysql发出公告消并announce邮件列表。请参见1.7.1.1节，“The MySQL邮件列表”。公告消息包含所有发布的更改列表和已知问题。只有部分发版不需要已知的问题部分。
- 为了让我们的用户快速访问最新MySQL功能，我们每4-8周产生一个新的MySQL发布。每天构建源码快照，可以从http://downloads.mysql.com/snapshots.php获得。
- 如果，尽管经过我们最大的努力，我们在发布后仍收到缺陷报告，即在某个具体平台上出现严重问题，我们将立即进行修复，并为该平台构建一个新的 'a'版本。由于我们的大用户群，可以很快地查出并解决此类问题。
- 我们为保证稳定版本所做的跟踪记录不错。在最近150个发布中，我们只需要对其中不到10个重新构建。其中有3个案例，缺陷为我们的构建机器上的glibc 库，花了很长时间来跟踪。

2.1.2.5. MySQL AB编译的MySQL二进制版本

作为MySQL AB的服务，我们提供了一套二进制MySQL分发版，已经在我们的站点上所列的系统或支持MySQL、友好地为我们提供访问的机器上的系统上编译过。

除了具体平台安装软件包格式提供的二进制分发版，我们还以压缩**tar**文件(.tar.gz文件)为多种平台提供了二进制分发版。请参见2.2节，“使用二进制分发版的标准MySQL安装”。

对于Windows分发版，参见2.3节，“在Windows上安装MySQL”。

使用脚本 Build-tools/Do-compile生成这些分发版，可以编译源码并使用**scripts/make_binary_distribution**创建二进制tar.gz文件。

通过下面的编译器和选项来配置并构建二进制分发版。可以查看各个库二进制**tar**文件分发版的脚本**bin/mysqlbug**内的变量COMP_ENV_INFO和 CONFIGURE_LINE来查看这些信息。

- 在MySQL AB开发系统上构建下面的二进制：

  · Linux 2.4.xx x86中的**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.x x86中的**icc**(Intel C++ Compiler 8.1或更高版本)：

  CC=icc CXX=icpc CFLAGS="-O3 -unroll2 -ip -mp -no-gcc -restrict" CXXFLAGS="-O3 -unroll2 -ip -mp -no-gcc -restrict" ./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 --enable-assembler --disable-shared --with-client-ldflags=-all-static --with-mysqld-ldflags=-all-static --with-embedded-server --with-innodb

  请注意Intel 编译器8.1版和更高版有不同的"纯"C(icc)驱动程序和C++(icpc)驱动程序；如果你使用**icc** 8.0版本或较早的版本来构建MySQL，需要设置CXX=icc。

  · Linux 2.4.xx Intel Itanium 2中的**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中的**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中的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.x.xx ppc中的**gcc** 2.95.4：

  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-embedded-server --with-innodb

  · Linux 2.4.xx s390中的**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)中的**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中的**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-inno

  · Sun Solaris 8 SPARC中的**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 64-bit中的**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 --with-named-z-libs=no --with-named-curses-libs=-lcurses --disable-shared

  · Sun Solaris 9 SPARC中的**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中的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中的**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中的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中的**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 --enable-thread-safe-client --enable-local-infile --with-named-z-libs=no --disable-shared

- IBM AIX 5.2.0 ppc中的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-embedded-server --with-innodb

- HP-UX 10.20 pa-risc1.1中的**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 11.00 pa-risc中的aCC(HP ANSI C++ B3910B A.03.50):

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-embedded-server --with-innodb

- HP-UX 11.11 pa-risc2.0 64bit中的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中的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

- HP-UX 11.22 ia64 64bit中的aCC(HP aC++/ANSI C B3910B A.05.50):

CC=cc CXX=aCC CFLAGS="+DD64 +DSitanium2" CXXFLAGS="+DD64 +DSitanium2" ./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-embedded-server --with-innodb

- Apple Mac OS X 10.2 powerpc中的**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中的**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

- 使用LinuxThreads的FreeBSD 4.7 i386中的**gcc** 2.95.4:

CFLAGS="-DHAVE_BROKEN_REALPATH -D__USE_UNIX98 -D_REENTRANT -D_THREAD_SAFE -I/usr/local/include/pthread/linuxthreads" CXXFLAGS="-DHAVE_BROKEN_REALPATH -D__USE_UNIX98 -D_REENTRANT -D_THREAD_SAFE -I/usr/local/include/pthread/linuxthreads" ./configure --prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data --libexecdir=/usr/local/mysql/bin --enable-thread-safe-client --enable-local-infile --enable-assembler --with-named-thread-libs="-DHAVE_GLIBC2_STYLE_GETHOSTBYNAME_R -D_THREAD_SAFE -I/usr/local/include/pthread/linuxthreads -L/usr/local/lib -llthread -llgcc_r" --disable-shared --with-embedded-server --with-innodb

- QNX Neutrino 6.2.1 i386中的**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

下面的二进制是在由其它用户提供给MySQL AB的第三方系统上构建而成。这是免费赠送；MySQL AB没有完全控制这些系统，因此我们只能对在这些系统上构建的二进制提供有限的支持。

- SCO Unix 3.2v5.0.7 i386中的**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 UnixWare 7.1.4 i386中的**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 --with-readline

- CO OpenServer 6.0.0 i386中的**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 --with-readline

- Compaq Tru64 OSF/1 V5.1 732 alpha中的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-named-thread-libs="-lpthread -lmach -lexc -lc" --disable-shared --with-mysqld-ldflags=-all-static

- SGI Irix 6.5 IP32中的**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/sparc64 5.0中的**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

在MySQL AB过去提供的二进制软件包中应用了下面的编译选项。这些二进制不再进行更新，但是在这里列出了这些编译选项供参考。

- Linux 2.2.xx SPARC中的**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 x686中的**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中的**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(及以上版本)sun4u中的**egcs** 1.0.3a或2.90.27或**gcc** 2.95.2和更新版:

- 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中的**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中的**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 **gcc** 2.7.2:

中的

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

- AIX 4.2中的**gcc** 2.7.2.2：

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

如果你有更好的上述配置选项，可以向MySQL internals邮件列表发送邮件。请参见1.7.1.1节，“The MySQL邮件列表”。

在我们的站点中列出的MySQL 5.1发布的RPM分发版是由MySQL AB产生的。

如果你想要编译MySQL调试版本，应当在前面的**configure**命令中加上--with-debug或--with-debug=full，并去掉-fomit-frame-pointer选项。

2.1.3. 怎样获得**MySQL**

检查MySQL下载页(http://dev.mysql.com/downloads/)获取当前版本和下载指令信息。关于MySQL下载镜像站点的完整的最新列表，参见 http://dev.mysql.com/downloads/mirrors.html。你可以在此发现MySQL镜像站点和如何报告有问题的或过时的镜像站点的信息。

我们的主镜像站点位于http://mirrors.sunsite.dk/mysql/。

2.1.4. 通过**MD5**校验和或**GnuPG**验证软件包的完整性

2.1.4.1. 验证MD5校验和
2.1.4.2. 通过GnuPG进行签名检查
2.1.4.3. 使用RPM进行签名检查

下载了适合你的需求的MySQL安装软件包并在安装前，你应当确保它的完整性，没有被篡改。MySQL AB提供了3种完整性检查方法：

- MD5校验和
- 使用GnuPG、GNU Privacy Guard对签名进行加密
- 对于RPM软件包，使用内嵌式RPM完整性验证机制

下面的章节描述了如何使用这些方法。

如果你注意到MD5校验和与GPG签名不匹配，首先应尝试多次下载相关安装软件包，可以从其它镜像站点。如果你反复尝试仍不能成功验证安装软件包的完整性，请将此类问题通知给我们，包括完整的安装软件包名和你使用的下载站点，在<webmaster@mysql.com>或<build@mysql.com>。不要使用缺陷-报告机制来报告下载问题。

2.1.4.1. 验证**MD5**校验和

下载MySQL安装软件包后，你应当确保它的MD5校验和与提供的MySQL下载页上的相匹配。每个安装软件包有唯一的校验和，可以用下面的命令来验证，其中package_name是你下载的安装软件包的名称：

```
shell> md5sum package_name
```

示例：

```
shell> md5sum mysql-st和ard-5.1.2-alpha-linux-i686.tar.gz
aaab65abbec64d5e907dcd41b8699945  mysql-st和ard-5.1.2-alpha-linux-i686.tar.gz
```

你应当验证结果校验和(十六进制数字字符串)与下载页上相关安装软件包下面显示的校验和相匹配。

注释：一定要验证归档文件(例如，.zip或.tar.gz文件)的校验和，而不是其中包括的文件。

请注意并非所有操作系统支持**md5sum**命令。在一些操作系统上，只是称为**md5**，另一些根本不装载它。在Linux中，它是**GNU Text Utilities**安装软件包的一部分，适合各种平台。可以从http：//www.gnu.org/software/textutils/下载源码。如果你已经安装了OpenSSL，还可以使用命令**openssl md5 package_name**。http：//www.fourmilab.ch/md5/提供了DOS/Windows使用的**md5**命令行实用工具。**winMd5Sum**是一个图形MD5检查工具，可以从http：//www.nullriver.com/index/products/winmd5sum获得。

2.1.4.2. 通过**GnuPG**进行签名检查

验证安装软件包完整性和真实性的另一个方法是使用加密签名。这比使用MD5校验和更可靠，但是需要更多的工作。

MySQL AB用**GnuPG**(GNU Privacy Guard)对下载MySQ软件包进行签名。**GnuPG**是开放源码，不同于Phil Zimmermann的闻名的Pretty Good Privacy (**PGP**)。关于**GnuPG**和如何获得并安装到你的系统的详细信息，请参见http：//www.gnupg.org/。大多数Linux分发版装有默认安装的**GnuPG**。关于**GnuPG**的详细信息，参见 http：//www.openpgp.org/。

要想验证具体安装软件包的签名，你首先需要获得MySQL AB公共GPG构建密钥的拷贝。可以从http：//www.keyserver.net/下载密钥。你想要获得的密钥名为build@mysql.com。另外，可以从下面的文本直接剪切并粘贴密钥：

```
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
QJEXM6FSbi0LLtZciNlYsafwAPEOMDKpMqAK6IyisNtPvaLd8lH0bPAnWqcyefep
rv0sxxqUEMcM3o7wwgfN83POkDasDbs3pjwPhxvhz6//62zQJ7Q7TXlTUUwgUGFj
a2FnZSBzaWduaW5nIGtleSAod3d3Lm15c3FsLmNvbSkgPGJ1aWxkQG15c3FsLmNv
bT6IXQQTEQIAHQUCPj6jDAUJCWYBgAULBwoDBAMVAwIDFgIBAheAAAoJEIxxjTtQ
cuH1cY4AnilUwTXn8MatQOiG0a/bPxrvK/gCAJ4oinSNZRYTnblChwFaazt7PF3q
zIhMBBMRAgAMBQI+PqPRBYMJZgC7AAoJEElQ4SqycpHyJOEAn1mxHijft00bKXvu
cSo/pECUmppiAJ41M9MRVj5VcdH/KN/KjRtW6tHFPYhMBBMRAgAMBQI+QoIDBYMJ
YiKJAAoJELb1zU3GuiQ/lpEAoIhpp6BozKI8p6eaabzF5MlJH58pAKCu/ROofK8J
Eg2aLos+5zEYrB/LsrkCDQQ+PqMdEAgA7+GJfxbMdY4wslPnjH9rF4N2qfWsEN/l
xaZoJYc3a6M02WCnHl6ahT2/tBK2w1QI4YFteR47gCvtgb6O1JHffOo2HfLmRDRi
Rjd1DTCHqeyX7CHhcghj/dNRlW2Z0l5QFEcmV9U0Vhp3aFfWC4Ujfs3LU+hkAWzE
7zaD5cH9J7yv/6xuZVw411x0h4UqsTcWMu0iM1BzELqX1DY7LwoPEb/O9Rkbf4fm
Le11EzIaCa4PqARXQZc4dhSinMt6K3X4BrRsKTfozBu74F47D8Ilbf5vSYHbuE5p
/1oIDznkg/p8kW+3FxuWrycciqFTcNz215yyX39LXFnlLzKUb/F5GwADBQf+Lwqq
a8CGrRfsOAJxim63CHfty5mUc5rUSnTslGYEIOCR1BeQauyPZbPDsDD9MZ1ZaSaf
anFvwFG6Llx9xkU7tzq+vKLoWkm4u5xf3vn55VjnSd1aQ9eQnUcXiL4cnBGoTbOW
I39EcyzgslzBdC++MPjcQTcA7p6JUVsP6oAB3FQWg54tuUo0Ec8bsM8b3Ev42Lmu
QT5NdKHGwHsXTPtl0klk4bQk4OajHsiy1BMahpT27jWjJlMiJc+IWJ0mghkKHt92
6s/ymfdf5HkdQ1cyvsz5tryVI3Fx78XeSYfQvuuwqp2H139pXGEkg0n6KdUOetdZ
Whe70YGNPw1yjWJT1IhMBBgRAgAMBQI+PqMdBQkJZgGAAAoJEIxxjTtQcuH17p4A
n3r1QpVC9yhnW2cSAjq+kr72GX0eAJ4295kl6NxYEuFApmr1+0uUq/SlsQ==
=YJkx
-----END PGP PUBLIC KEY BLOCK-----
```

你可以使用**gpg --import**将构建密钥导入你的个人公共GPG钥匙圈（keyring）。例如，如果你把密钥保存到mysql_pubkey.asc文件中，导入命令应为：

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

关于公共密钥如何工作的详细信息，请参阅GPG文档。

下载并导入公共构建密钥后，下载期望的MySQL安装软件包和相应的签名，也可以从下载页上获得。签名文件名与分发版文件相同，有 an .asc扩展名。例如：

分发文件	mysql-st和ard-5.1.2-alpha-linux-i686.tar.gz
签名文件	mysql-st和ard-5.1.2-alpha-linux-i686.tar.gz.asc

确保两个文件保存在同一目录下，运行下面的命令来验证分发文件的签名：

```
shell> gpg --verify package_name.asc
```

示例：

```
shell> gpg --verify mysql-standard-5.1.2-alpha-linux-i686.tar.gz.asc
gpg: Signature made Tue 12 Jul 2005 23:35:41 EST using DSA key ID 5072E1F5
gpg: Good signature from "MySQL Package signing key (www.mysql.com) <build@mysql.com>"
```

Good signature消息表示所有内容都很正确。你可以忽略任何insecure memory警告。

2.1.4.3. 使用RPM进行签名检查

RPM软件包没有单独的签名。RPM软件包内置GPG签名和MD5校验和。可以运行下面的命令来验证安装软件包：

```
shell> rpm --checksig package_name.rpm
```

示例：

```
shell> rpm --checksig MySQL-server-5.1.2-alpha-0.i386.rpm
MySQL-server-5.1.2-alpha-0.i386.rpm:  md5 gpg OK
```

注释：如果你正使用RPM 4.1，并且出现关于(GPG) NOT OK (MISSING KEYS: GPG#5072e1f5)的抱怨，尽管你已经将MySQL公共构建密钥导入到你自己的GPG钥匙圈，你必须首先将密钥导入到RPM钥匙圈中。RPM 4.1不再使用你的个人GPG钥匙圈(或GPG本身)。RPM 4.1维护自己的钥匙圈，因为它是系统范围的应用程序，而用户GPG公共钥匙圈是具体的用户文件。要想将MySQL公共密钥导入RPM钥匙圈，要首先获得前面章节描述的密钥。然后使用**rpm --import**导入密钥。例如，如果你的公共密钥保存在mysql_pubkey.asc文件中，应使用下述命令导入公共密钥：

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

如果你需要获得MySQL公共密钥，参见2.1.4.2节，“通过GnuPG进行签名检查”。

2.1.5. 安装布局

这节描述安装MySQL AB提供的二进制代码和源码分发时创建的默认目录布局。如果你安装其它供应商提供的分发，应使用其它布局。

在Windows中，MySQL 5.1的默认安装目录是C:\Program Files\MySQL\MySQL Server 5.1。(一些Windows用户宁愿安装到原来的默认安装目录 C：\mysql。然而，子目录布局仍然相同）。安装目录包括以下子目录：

目录	目录内容
bin	客户端程序和**mysqld**服务器
data	日志文件，数据库
Docs	文档
examples	示例程序和脚本
include	包含(头)文件
lib	库
scripts	实用工具脚本
share	错误消息文件

使用MySQL AB's Linux RPM分发进行安装后，将在以下系统目录产生文件：

目录	目录内容

/usr/bin	客户端程序和脚本
/usr/sbin	**mysqld**服务器
/var/lib/mysql	日志文件，数据库
/usr/share/doc/packages	文档
/usr/include/mysql	包含(头)文件
/usr/lib/mysql	库
/usr/share/mysql	错误消息和字符集文件
/usr/share/sql-bench	基准程序

在Unix中，可以在你选择的安装位置解压并安装**tar**文件二进制分发(typically /usr/local/mysql)并在该位置创建以下目录：

目录	目录内容
bin	客户端程序和**mysqld**服务器
data	日志文件，数据库
docs	文档，ChangeLog
include	包含(头)文件
lib	库
scripts	**mysql_install_db**
share/mysql	错误消息文件
sql-bench	基准程序

配置并编译完源码分发后，便开始安装。默认情况下，可以将文件安装到/usr/local，即在下面的子目录中：

目录	目录内容
bin	客户端程序和脚本
include/mysql	包含(头)文件
info	Info格式的文档
lib/mysql	库
libexec	**mysqld**服务器
share/mysql	错误消息文件
sql-bench	基准程序和crash-me测试
var	数据库和日志文件

在一个安装目录内，源码安装的布局在下列方面不同于二进制安装：

- **mysqld**服务器被安装在“`libexec`”目录而不是“`bin`”目录内。
- 数据目录是“`var`”而非“`data`”。
- **mysql_install_db**被安装在“`bin`”目录而非“`scripts`”内。
- 头文件和库目录是“`include/mysql`”和“`lib/mysql`”而非“`include`”和“`lib`”。

执行源码分发根目录下的scripts/make_binary_distribution脚本，你可以用编译过的源码分发版创建你自己的二进制安装。

2.2. 使用二进制分发版的标准MySQL安装

后面几章包括如何在安装软件包格式适合的平台上平台安装MySQL。(即执行“二进制安装”）。也可以在多种平台上安装MySQL二进制分发版。关于适合所有平台的软件包的通用安装指令参见2.7节，“在其它类Unix系统中安装MySQL”。

关于二进制分发版的使用和如何获得的详细信息请参见2.1节，“一般安装问题”。

2.3. 在Windows上安装MySQL

2.3.1. Windows系统要求
2.3.2. 选择安装软件包
2.3.3. 用自动安装器安装MySQL

2.3.4. 使用MySQL安装向导
2.3.5. 使用配置向导
2.3.6. 通过非安装Zip文件安装MySQL
2.3.7. 提取安装档案文件
2.3.8. 创建选项文件
2.3.9. 选择MySQL服务器类型
2.3.10. 首次启动服务器
2.3.11. 从Windows命令行启动MySQL
2.3.12. 以Windows服务方式启动MySQL
2.3.13. 测试MySQL安装
2.3.14. 在Windows环境下对MySQL安装的故障诊断与排除
2.3.15. 在Windows下升级MySQL
2.3.16. Windows版MySQL同Unix版MySQL对比

MySQL AB已经提供了Windows中安装的MySQL 3.21以上版本，并提供了每天下载MySQL的比率。本节描述在Windows中安装MySQL的过程。

使用Windows版本MySQL安装器，结合GUI配置向导，可以自动安装MySQL，创建选项文件，启动服务器并使默认用户账户安全。

如果你要升级已有的4.1.5版以前的MySQL,你必须执行以下步骤:

1. 获得并安装分发版。
2. 根据需要设置选项文件。
3. 选择想要使用的服务器。
4. 启动服务器。
5. 为初始MySQL账户指定密码。

该过程还适合安装软件包内不包括安装器的MySQL安装。

Windows版MySQL 5.1有3种分发格式:

- 二进制分发版包含一个设置程序，可以安装你需要的任何内容，因此可以立即启动服务器。
- 源码分发版包含所有使用Visual Studio 2003编译器来构建可执行程序的代码和支持文件。

一般来讲，你应当使用二进制分发版。它比其它的分发版使用起来要简单，不再需要其它工具来启动并运行MySQL。

本节描述了如何使用二进制分发版在Windows中安装MySQL。要想使用源码分发版安装，参见2.8.6节，“在Windows下从源码安装MySQL”。

2.3.1. Windows系统要求

要想在Windows中运行MySQL，你需要:

- 32位Windows操作系统，例如9x、Me、NT、2000、XP或Windows Server 2003。

基于Windows NT的操作系统(NT，2000,XP，2003)，将MySQL服务器做为服务来运行。强烈建议使用基于Windows NT的操作系统。请参见2.3.12节，“以Windows服务方式启动MySQL”。

- TCP/IP协议支持。
- Windows版本MySQL二进制分发版，可以从http：//dev.mysql.com/下载/下载。请参见2.1.3节，“怎样获得MySQL”。

注释：如果你从FTP下载分发版，我们建议使用充分的FTP客户端以保证续传，避免下载过程中文件被破坏。

- 可以读取 .zip文件的工具，以解压分发文件。
- 硬盘上有足够的空间保证根据你的需求来解包、安装和创建数据库(一般建议至少有200兆字节)。

你还可以有以下可选需求:

- 如果你计划通过ODBC连接MySQL服务器，你还需要一个连接器/ODBC驱动程序。请参见26.1节，“MySQL Connector/ODBC”。
- 如果表需要占用的空间大于4GB，则在NTFS或更新的文件系统上安装MySQL。当创建表时不要忘记使用MAX_ROWS和 AVG_ROW_LENGTH。请参见13.1.5节，“CREATE TABLE语法”。

2.3.2. 选择安装软件包

在Windows中安装MySQL时，有3种MySQL 5.1安装软件包可供选择：

- 基本安装：该安装软件包的文件名类似于mysql-essential-5.1.2-alpha-win32.msi，包含在Windows中安装MySQL所需要的最少的文件，包括配置向导。该安装软件包不包括可选组件，例如嵌入式服务器和基准套件。
- 完全安装：该安装软件包的文件名类似于mysql-5.1.2-alpha-win32.zip，包含在Windows中安装MySQL所需要的全部文件，包括配置向导。该安装软件包包括可选组件，例如嵌入式服务器和基准套件。
- 非自动安装文件：该安装软件包的文件名类似于mysql-noinstall-5.1.2-alpha-win32.zip，包含完整安装包中的全部文件，只是不包括配置向导。该安装软件包不包括自动安装器，必须手动安装和配置。

对于大多数用户，建议选择基本安装。

你的选择会影响你后面必须遵从的安装过程。如果你选择基本安装或完全安装，参见2.3.3节，“用自动安装器安装MySQL”。如果你选择非自动安装MySQL，参见2.3.6节，“通过非安装Zip文件安装MySQL”。

2.3.3. 用自动安装器安装MySQL

在Windows中安装MySQL时，新用户可以使用MySQL安装帮助和MySQL Configuration Wizard（配置向导）。MySQL安装和配置的方式是使新用户可以立即开始使用MySQL。

基本安装和完全安装中均包括MySQL安装帮助和配置向导，建议在大多数标准MySQL安装中选择。例外情况包括想在单个服务器上安装多个实例的用户和想完全控制服务器配置的高级用户。

2.3.4. 使用MySQL安装向导

2.3.4.1. 前言
2.3.4.2. 下载并启动MySQL安装向导
2.3.4.3. 选择安装类型
2.3.4.4. 定制安装对话框
2.3.4.5. 配置对话框
2.3.4.6. MySQL安装向导所作的更改
2.3.4.7. 升级MySQL

2.3.4.1. 前言

MySQL安装帮助是MySQL服务器的安装器，使用最新的Microsoft Window安装器技术。结合使用MySQL安装帮助和配置向导，用户安装并 配置完MySQL服务器后便可以直接使用。

MySQL安装帮助是MySQL 5.1服务器分发的标准安装器。使用MySQL安装帮助安装MySQL之前，用户需要手动关闭并卸载已经安装的以前版本的MySQL。关于对以前版本进行升级的详细信息请参见2.3.4.7节，“升级MySQL”。

最近版本的Windows包含了改进版本的Microsoft Windows安装器(MSI)。MSI已经成为Windows 2000、Windows XP和Windows Server 2003应用程序安装的事实标准。MySQL安装帮助中使用了该技术，使安装过程更加灵活、顺利。

Microsoft Windows安装器引擎随着Windows XP的更新而更新；使用以前Windows版本的用户可以参考Microsoft 知识库文章查阅关于升级到最新版Windows安装器引擎的资料。

此外，Microsoft最近已经引入了WiX(Windows 安装器 XML)工具包。这是Microsoft公认的开放源码项目。我们转换到WiX是因为它是一个开放源码项目，可以使用脚本更加灵活地处理整个Windows安装过程。

对MySQL装帮助的改进依赖于各用户的支持和反馈。如果你发现MySQL安装帮助缺少对你很重要的某些功能，或如果你发现某个缺陷，请使用我们的MySQL缺陷系统来索取功能或报告问题。

2.3.4.2. 下载并启动MySQL安装向导

可以从http://dev.mysql.com/downloads/下载MySQL服务器安装软件包。如果你下载的安装软件包在Zip文件中，你需要先提取文件。

启动帮助的过程取决于下载的安装软件包的内容。如果有setup.exe文件，双击启动安装过程。如果有.msi文件，双击启动安装过程。

2.3.4.3. 选择安装类型

有3种安装类型：**Typical(**典型安装**)**、**Complete(**完全安装**)**和**Custom(**定制安装**)**。

Typical(典型安装**)**安装只安装MySQL服务器、**mysql**命令行客户端和命令行实用程序。命令行客户端和实用程序包括**mysqldump**、**myisamchk**和其它几个工具来帮助你管理MySQL服务器。

Complete(完全安装**)**安装将安装软件包内包含的所有组件。完全安装软件包包括的组件包括嵌入式服务器库、基准套件、支持脚本和文档。

Custom(定制安装**)**安装允许你完全控制你想要安装的软件包和安装路径。关于定制安装的详细信息请参见2.3.4.4节，“定制安装对话框”。

如果你选择**Typical(**典型安装**)**或**Complete(**完全安装**)**安装并点击Next按钮，你将进入确认窗口确认选择并开始安装。如果你选择定制安装并点击Next按钮，你将进入定制安装对话框，参见2.3.4.4节，“定制安装对话框”。

2.3.4.4. 定制安装对话框

如果你想要更改MySQL安装帮助安装的安装路径或具体组件，应当选择**Custom(**定制安装**)**安装。

所有可用组件列入定制安装对话框左侧的树状视图内。未安装的组件用红色 X 图标表示；已经安装的组件有灰色图标。要想更改组件，点击该组件的图标并从下拉列表中选择新的选项。

可以点击安装路径右侧的Change...按钮来更改默认安装路径。

选择了安装组件和安装路径后，点击Next按钮进入确认对话框。

2.3.4.5. 配置对话框

选择了安装类型和可选安装组件后，则进入确认对话框。该对话框中将显示安装类型和安装路径供检查。

如果你满意设定值要想安装MySQL，点击Install（安装）按钮。要想更改设定值，点击Back按钮。要想退出MySQL安装帮助不再安装 MySQL，点击Cancel按钮。

完成安装后，将出现注册选项和MySQL网址。注册后便可以访问forums.mysql.com处的MySQL论坛，可以在bugs.mysql.com报告缺陷，并可以注册为时事新闻订户。在安装器的最后一个窗口内，提供了安装摘要，并提供选项以启动MySQL Configuration Wizard（配置向导），可以用来创建配置文件，安装MySQL服务并进行安全配置。

2.3.4.6. MySQL安装向导所作的更改

点击Install（安装）按钮后，MySQL安装帮助开始安装过程，并对系统进行下面章节描述的更改。

更改注册表

在典型安装中，MySQL安装帮助在HKEY_LOCAL_MACHINE\SOFTWARE\MySQL AB创建Windows注册键值。

MySQL安装帮助根据正安装的服务器的主要版本创建一个键值，例如 MySQL服务器5.1。它包含两个字符串值，Location和Version。Location字符串包含安装目录。在默认安装中，它包含C:\Program Files\MySQL\MySQL Server 5.1\。Version字符串包含发布号。例如，安装MySQL Server 5.1.2-alpha，键值包含一个5.1.2-alpha值。

这些注册键值用来帮助外部工具识别MySQL服务器的安装位置，不需要扫描整个硬盘来确定MySQL服务器的安装路径。运行服务器时不需要注册键值，使用noinstall Zip文件时不创建注册键值。

更改启动菜单

MySQL安装帮助在Windows 启动菜单中创建一条新的条目，使用MySQL菜单，根据安装的MySQL的主版本来命名。例如，如果安装了MySQL 5.1, MySQL安装帮助在启动菜单中创建MySQL Server 5.1部分。

将在新启动菜单部分创建下面的条目：

- MySQL命令行客户端：这是**mysql**命令行客户端的快捷方式，对其进行配置以连接为root用户。当连接时快捷方式提示输入root用户密码。
- MySQL服务器实例配置向导：这是MySQL Configuration Wizard（配置向导）的快捷方式。使用该快捷方式来配置新安装的服务器，或重新配置已有的服务器。
- MySQL文档：可以连接到保存在MySQL服务器安装目录下的文档。采用基本安装方式安装MySQL服务器时，该选项不可用。

更改文件系统

默认情况下，MySQL安装帮助将MySQL 5.1服务器安装到C:*Program Files*\MySQL\MySQL Server *5.1*，其中*Program Files*是系统应用程序的默认位置，*5.1*是MySQL服务器的主要版本。这是建议的MySQL服务器的新安装位置，替换了前面的默认位置c：\mysql。

默认情况下，所有MySQL应用程序保存到目录C:*Program Files*\MySQL下，其中*Program Files*是应用程序在Windows中的默认安装位置。开发机器上的典型MySQL安装应为：

```
C:\Program Files\MySQL\MySQL Server 5.1
C:\Program Files\MySQL\MySQL Administrator 1.0
C:\Program Files\MySQL\MySQL Query Browser 1.0
```

该方法使管理和维护具体系统上安装的MySQL应用程序更加容易。

2.3.4.7. 升级MySQL

使用MSI的升级功能，MySQL安装帮助可以自动执行服务器升级。这意味着安装新版本前，不需要手动卸载前面安装的程序。安装新版本前，安装器自动关闭并卸载前面安装的MySQL服务。

只有在主版本号和次要版本号相同的安装之间进行升级时，才能自动进行升级。例如，可以自动从MySQL 4.1.5升级到MySQL 4.1.6,但是不能从MySQL 5.0升级到MySQL 5.1。

参见2.3.15节，“在Windows下升级MySQL”。

2.3.5. 使用配置向导

2.3.5.1. 前言
2.3.5.2. 启动MySQL配置向导
2.3.5.3. 选择维护选项
2.3.5.4. 选择配置类型
2.3.5.5. 服务器类型对话框
2.3.5.6. 数据库使用情况对话框
2.3.5.7. InnoDB表空间对话框
2.3.5.8. 并发连接对话框
2.3.5.9. 联网选项对话框
2.3.5.10. 字符集对话框
2.3.5.11. 服务选项对话框
2.3.5.12. 安全选项对话框
2.3.5.13. 配置对话框
2.3.5.14. my.ini文件的位置
2.3.5.15. 编辑my.ini文件

2.3.5.1. 前言

MySQL Configuration Wizard（配置向导）可以帮助自动配置Windows中的服务器。MySQL Configuration Wizard（配置向导）问你一系列问题，然后将回答放到模板中生成一个my.ini文件，该文件与你的安装一致。

MySQL Configuration Wizard（配置向导）包含在MySQL 5.1服务器中，目前只适用于Windows用户。

MySQL Configuration Wizard（配置向导）在很大程度上是MySQL AB经过多年从许多用户收到的反馈的结果。然而，如果你发现它缺少某些对你很重要的功能，或如果你发现一个缺陷，请使用我们的MySQL 缺陷系统来索取功能或报告问题。

2.3.5.2. 启动MySQL配置向导

一般情况当MySQL安装帮助退出时，从MySQL安装帮助启动MySQL Configuration Wizard（配置向导）。还可以点击Windows启动菜单中MySQL服务器实例配置向导条目中的MySQL部分来启动MySQL Configuration Wizard（配置向导）。

并且，还可以进入MySQL安装bin目录直接启动MySQLInstanceConfig.exe文件。

2.3.5.3. 选择维护选项

如果MySQL Configuration Wizard（配置向导）检查到my.ini文件，你可以选择重新配置已有的服务器，或通过删除my.ini文件并停止、卸载MySQL服务来卸载服务器实例。

要想重新配置已有的服务器，选择Re-configure Instance选项并点击Next按钮。已有的my.ini文件重新命名为my*timestamp*.ini.bak，其中*timestamp*是my.ini文件创建是的日期和时间。要想卸载已有的服务器实例，选择Remove Instance选项并点击Next按钮。

如果选择了Remove Instance选项，则进入确认窗口。点击Execute按钮：MySQL Configuration Wizard（配置向导）停止并卸载MySQL服务，然后删除my.ini文件。服务器安装和自己的data文件夹不删除。

如果选择了Re-configure Instance选项，则进入配置类型对话框，可以选择你想要配置的安装类型。

2.3.5.4. 选择配置类型

当启动MySQL Configuration Wizard（配置向导）重新安装MySQL，或为已有安装选择Re-configure Instance选项，则进入配置类型对话框。

可以选择两种配置类型：Detailed Configuration（详细配置）和Standard Configuration（标准配置）。Standard Configuration（标准配置）选项适合想要快速启动MySQL而不必考虑服务器配置的新用户。详细配置选项适合想要更加细粒度控制服务器配置的高级用户。

如果你是MySQL的新手，需要配置为单用户开发机的服务器，Standard Configuration（标准配置）应当适合你的需求。选择Standard Configuration（标准配置）选项，则 MySQL Configuration Wizard（配

置向导）自动设置所有配置选项，但不包括服务选项和安全选项。

Standard Configuration（标准配置）设置选项可能与安装MySQL的系统不兼容。如果系统上已经安装了MySQL和你想要配置的安装，建议选择详细配置。

要想Standard Configuration（标准配置），请分别参阅2.3.5.11节，“服务选项对话框”和2.3.5.12节，“安全选项对话框”的服务选项和安全选项。

2.3.5.5. 服务器类型对话框

可以选择3种服务器类型，选择哪种服务器将影响到MySQL Configuration Wizard（配置向导）对内存、硬盘和过程或使用的决策。

- Developer Machine(开发机器)：该选项代表典型个人用桌面工作站。假定机器上运行着多个桌面应用程序。将MySQL服务器配置成使用最少的系统资源。
- Server Machine（服务器）：该选项代表服务器，MySQL服务器可以同其它应用程序一起运行，例如FTP、email和web服务器。MySQL服务器配置成使用适当比例的系统资源。
- Dedicated MySQL Server Machine（专用MySQL服务器）：该选项代表只运行MySQL服务的服务器。假定运行没有运行其它应用程序。MySQL服务器配置成使用所有可用系统资源。

2.3.5.6. 数据库使用情况对话框

通过Database Usage（数据库使用）对话框，你可以指出创建MySQL表时使用的表处理器。通过该选项，你可以选择是否使用InnoDB储存引擎，以及InnoDB占用多大比例的服务器资源。

- Multifunctional Database（多功能数据库）：选择该选项，则同时使用InnoDB和MyISAM储存引擎，并在两个引擎之间平均分配资源。建议经常使用两个储存引擎的用户选择该选项。
- Transactional Database Only（只是事务处理数据库）：该选项同时使用InnoDB和MyISAM 储存引擎，但是将大多数服务器资源指派给InnoDB储存引擎。建议主要使用InnoDB只偶尔使用MyISAM的用户选择该选项。
- Non-Transactional Database Only（只是非事务处理数据库）：该选项完全禁用InnoDB储存引擎，将所有服务器资源指派给MyISAM储存引擎。建议不使用InnoDB的用户选择该选项。

2.3.5.7. InnoDB表空间对话框

有些用户可能想要将InnoDB表空间文件放到不同的位置，而不放到MySQL服务器数据目录。如果你的系统有较大的空间或较高性能的储存设备（例如RAID储存系统），则最好将表空间文件单独放到一个位置。

要想更改InnoDB表空间文件的默认位置，从驱动器下拉列表选择一个新的驱动器，并从路径下拉列表选择新的路径。要想创建路径，点击 ...按钮。

如果你要更改已有服务器的配置，更改路径前你必须点击Modify按钮。此时启动服务器之前，你必须将已有表空间文件移到新位置。

2.3.5.8. 并发连接对话框

限制所创建的与MySQL服务器之间的并行连接数量很重要，以便防止服务器耗尽资源。在Concurrent Connections（并行连接）对话框中，可以选择服务器的使用方法，并根据情况限制并行连接的数量。还可以手动设置并行连接的限制。

- Decision Support（决策支持）(DSS)/OLAP：如果服务器不需要大量的并行连接可以选择该选项。假定最大连接数目设置为100,平均并行连接数为20。
- Online Transaction Processing（联机事务处理）(OLTP)：如果你的服务器需要大量的并行连接则选择该选项。最大连接数设置为500。
- Manual Setting（人工设置）：选择该选项可以手动设置服务器并行连接的最大数目。从前面的下拉框中选择并行连接的数目，如果你期望的数目不在列表中，则在下拉框中输入最大连接数。

2.3.5.9. 联网选项对话框

在Networking Options（网络选项）对话框中可以启用或禁用TCP/IP网络，并配置用来连接MySQL服务器的端口号。

默认情况启用TCP/IP网络。要想禁用TCP/IP网络，取消选择Enable TCP/IP Networking选项旁边的检查框。

默认使用3306端口。要想更访问MySQL使用的端口，从下拉框选择一个新端口号或直接向下拉框输入新的端口号。如果你选择的端口号已经被占用，将提示确认选择的端口号。

2.3.5.10. 字符集对话框

MySQL服务器支持多种字符集，可以设置适用于所有表、列和数据库的默认服务器字符集。使用Character Set（字符集对话框）来更改 MySQL服务器的默认字符集。

- Standard Character Set（标准字符集）：如果想要使用Latin1做为默认服务器字符集，则选择该选项。Latin1用于英语和许多西欧语言。
- Best Support For Multilingualism（支持多种语言）：如果想要使用UTF8做为默认服务器字符集，则选择该选项。UTF8可以 将不同语言的字符储存为单一的字符集。

- Manual Selected Default Character Set/Collation（人工选择的默认字符集/校对规则）：如果想要手动选择服务器的默认字符集，请选择该项。从下拉列表中选择期望的字符集。

2.3.5.11. 服务选项对话框

在基于Windows NT的平台上，可以将MySQL服务器安装成服务。安装成服务，系统启动时可以自动启动MySQL服务器，甚至出现服务故障时可以随Windows自动启动。

默认情况，MySQL Configuration Wizard（配置向导）将MySQL服务器安装为服务，服务名为MySQL。如果你不想安装服务，取消Install As Windows Service选项旁边的选择框。可以从下拉框选择新的服务名或在下拉框输入新的服务名来更改服务名。

要想将MySQL服务器安装为服务，但是不自动启动，不选中Launch the MySQL Server Automatically选项旁边的检查框。

2.3.5.12. 安全选项对话框

强烈建议为你的MySQL服务器设置一个root密码，默认情况MySQL Configuration Wizard（配置向导）要求你设置一个root密码。如果你不想设置root密码，不选中Modify Security Settings（修改安全设定值）选项旁边的选择框。

要想设置root密码，在New root password(输入新密码)和Confirm（确认）两个框内输入期望的密码。如果重新配置已有的服务器，你还需要Current root password（当前root密码）框内输入已有的root密码。

要想防止通过网络以root登录，选中Root may only connect from localhost（只允许从本机登陆连接root）选项旁边的框。这样可以提高root账户的安全。

要想创建一个匿名用户账户，选中Create An Anonymous Account（创建匿名账户）选项旁边的框。创建匿名账户会降低服务器的安全，并造成登录和许可困难。因此不建议。

2.3.5.13. 配置对话框

MySQL Configuration Wizard（配置向导）的最后一个对话框是Confirmation（确认）对话框。要想启动配置过程，点击Execute。要想返回要想到前面的对话框，点击Back按钮。要想不配置服务器即退出MySQL Configuration Wizard（配置向导），点击Cancel按钮。

点击Execute按钮后，MySQL Configuration Wizard（配置向导）执行一系列的任务，执行过程将显示在屏幕上。

MySQL Configuration Wizard（配置向导）首先使用MySQL AB开发人员和工程师准备的模板根据你的选择确定配置文件选项。该模板的名称为my-template.ini，位于服务器安装目录中。

MySQL Configuration Wizard（配置向导）将这些选项写入到一个my.ini文件中。my.ini文件的最终位置显示在写配置文件任务的旁边。

如果选择为MySQL服务器创建一个服务，MySQL Configuration Wizard（配置向导）则创建并启动服务。如果你重新配置已有的服务，MySQL Configuration Wizard（配置向导）则根据你的配置更改重新启动服务。

如果选择设置root密码，MySQL Configuration Wizard（配置向导）则连接服务器，设置新root密码并应用你选择的其它安全设定值。

MySQL Configuration Wizard（配置向导）完成任务后，则显示一个概要。点击Finish按钮退出MySQL Configuration Wizard（配置向导）。

2.3.5.14. **my.ini**文件的位置

MySQL Configuration Wizard（配置向导）将my.ini文件放到MySQL服务器的安装目录中。这样可以帮助将配置文件与具体服务器实例关联起来。

要想确保MySQL服务器知道从哪里查找my.ini文件，即做为服务安装的一部分传递给MySQL服务器的部分：--defaults-file="*C:\Program Files\MySQL\MySQL Server 5.1*\my.ini"，其中*C:\Program Files\MySQL\MySQL Server 5.1* 被MySQL服务器的安装路径替代。

--defaults-file指导MySQL服务器读取具体的配置文件。

2.3.5.15. 编辑**my.ini**文件

要想修改my.ini文件，用文本编辑器打开并进行必要的修改。你还可以用MySQL Administrator实用工具修改服务器配置。

MySQL客户端和实用程序，例如**mysql**命令行客户端和**mysqldump**不能将my.ini文件放到服务器安装目录中。要想配置客户端和实用工具，根据Windows 版本，在C：\Windows或C：\WINNT目录下创建一个新的my.ini文件。

2.3.6. 通过非安装**Zip**文件安装**MySQL**

想使用非自动安装包的用户可以使用本节的指令手动安装MySQL。从Zip文档安装MySQL的过程为：

1. 将文档提取到期望的安装目录中。

2. 创建一个选项文件。
3. 选择MySQL服务器类型。
4. 启动MySQL服务器。
5. 使默认用户账户安全。

下面一节描述了该过程。

2.3.7. 提取安装档案文件

要想手动安装MySQL，其步骤为：

1. 如果你从以前的版本升级，开始升级过程前请参阅2.3.15节，“在Windows下升级MySQL”。
2. 如果你使用基于Windows NT的操作系统，例如Windows NT、Windows 2000、Windows XP或Windows Server 2003,确保登录的用户名具有管理员权限。
3. 选择安装位置。传统安装中MySQL服务器安装到C：\mysql，MySQL Installation Wizard（安装帮助）将MySQL安装到C:\Program Files\MySQL。如果你不想将MySQL安装到C：\mysql，你必须在启动过程或在选项文件中指定安装目录的路径。请参见2.3.8节，“创建选项文件”。
4. 使用zip文件工具，将安装文件提取到选定的安装位置。使用某些工具，可以将文档提取到选择的安装位置的文件夹中。在这种情况下，你可以将子文件夹中的内容移动到选择的安装位置。

2.3.8. 创建选项文件

如果想要指定运行服务器的启动选项，可以在命令行中指出或放到一个选项文件中。对于服务器每次启动都用到的选项，你会发现使用选项文件来指定MySQL配置非常方便。特别是在以下环境中：

- 安装或数据目录位置不是默认位置(C:\Program Files\MySQL\MySQL Server 5.1和C:\Program Files\MySQL\MySQL Server 5.1\data)。
- 需要调节服务器设定值。

档MySQL服务器在Windows中启动时，它从两个文件中寻找选项：Windows目录中的my.ini文件和C：\my.cnf 文件。Windows目录典型名称为C：\WINDOWS或C：\WINNT。可以使用下面的命令从WINDIR环境变量值确定自己的确切位置：

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

MySQL首先从my.ini文件中寻找选项，然后从my.cnf文件中寻找。为了避免混淆，最好只使用一个文件。如果你的PC使用了一个引导加载器，其中C：不是引导盘，唯一的选则是使用my.ini文件。不管你使用哪个选项文件，必须为纯文本文件。

还可以使用MySQL分发中的示例选项文件。在安装目录中查找以下文件：my-small.cnf、my-medium.cnf、my-large.cnf和my-huge.cnf，你可以重新命名并复制到相应位置用作基本配置文件。

可以用任何文本编辑器来创建并修改选项文件，例如记事本。例如，如果MySQL安装在E：\mysql，数据目录安装在E:\mydata\data，你可以 创建含有[mysqld]部分的选项文件，指定basedir和datadir参数的值：

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

请注意在选项文件中使用正斜线儿不是反斜线指定Windows路径名。如果使用反斜线，必须用双反斜线：

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

在Windows下，MySQL安装器将数据目录直接放到安装MySQL的目录下。如果你想要使用其它位置的数据目录，应当将data目录中的全部内容复制到新的目录下。例如，如果MySQL安装到C:\Program Files\MySQL\MySQL Server 5.1中，数据目录默认在C:\Program Files\MySQL\MySQL Server 5.1\data。如果想要使用E:\mydata做为数据目录，你必须做两件事情：

1. 将整个data目录和全部内容从C:\Program Files\MySQL\MySQL Server 5.1\data移动到E:\mydata。

2. 每次启动服务器时，使用--datadir选项来指定新的数据目录。

2.3.9. 选择MySQL服务器类型

下面的表显示了Windows下可用的MySQL 5.1服务器:

二进制	描述
mysqld- debug	已经编译过，可以进行完全调试和自动内存分配检查，用于表InnoDB和BDB。
mysqld	优化的二进制，支持InnoDB。
mysqld-nt	优化的二进制，支持Windows NT，2000 XP，有命名管道。
mysqld-max	优化的二进制，支持InnoDB和BDB表。
mysqld-max-nt	同**mysqld-max**，但是已经编译过，支持命名管道。

已经对所有前面的二进制进行了优化以适用现代Intel处理器，但是应当工作在Intel i386-类或更高的处理器上。

所有Windows MySQL 5.1服务器支持数据库目录符号链接。

MySQL支持所有Windows平台的TCP/IP。**mysqld-nt**和mysql-max-nt服务器支持Windows NT、2000、XP和2003的命名管道。然而，在所有平台下默认使用TCP/IP。(在许多Windows配置中，命名管道比TCP/IP慢）。

命名管道在以下条件下使用:

- 如果使用--enable-named-pipe选项启动服务器，则启用了命名管道。需要明确使用该选项，因为当使用命名管道关闭MySQL服务器时，某些用户会遇到问题。
- 只有**mysqld-nt**或**mysqld-max-nt**服务器，并且服务器运行在支持命名管道的Windows版本(NT、2000、XP、2003)，才能够使用命名管道连接。
- 这些服务器可以运行在Windows 98 或Me中，但是必须安装TCP/IP协议；此时不可以使用命名管道连接。
- 这些服务器不能在Windows 95中运行。

注释：参考手册中的大多数例子使用**mysqld**做为服务器名。如果你选择使用不同的服务器，例如**mysqld-nt**，则在相应例子中的命令中进行相应的替换。

2.3.10. 首次启动服务器

本节中的信息主要适用选择NoinstallMySQL安装版本，或想要手动配置和测试MySQL而不用GUI工具的情况。

在Windows 95,98,或Me中，MySQL客户端总使用TCP/IP连接服务器。(这样允许网络中的所有机器连接MySQL服务器）。因此，在启动MySQL之前，你必须确保机器支持TCP/IP。可以从Windows CD-ROM中找到TCP/IP。

请注意如果使用旧版本的Windows 95(例如，OSR2)，很可能你使用了旧的Winsock包；MySQL需要Winsock 2。可以从http://www.microsoft.com/获得最新的Winsock。Windows 98包含新的Winsock 2库，因此不需要更新库。

在基于NT的系统中，例如 Windows NT、2000、XP或2003,客户端可以有两个选择。可以使用TCP/IP，如果服务器支持命名管道连接，也可以使用命名管道。要想使MySQL使用Windows NT 4中的TCP/IP,你必须安装服务pack 3(或更新的补丁)。

如果用--shared-memory选项启动，Windows版MySQL还支持共享内存连接。客户端可以通过--protocol=memory选项通过共享内存进行连接。

关于运行哪个服务器二进制的信息，参见2.3.9节，“选择MySQL服务器类型”。

本节对MySQL服务器启动进行了概述。下面章节详细说明了从命令行或Windows 服务启动MySQL服务器的具体信息。

这些章节的例子假定MySQL安装到默认位置C:\Program Files\MySQL\MySQL Server 5.1。如果你将MySQL安装到了其它位置，则要更改例子中的路径名。

在控制台窗口(或“DOS window”)中，最好根据命令提示进行测试。这样可以让服务器在window中容易看见的地方显示状态消息。如果配置出现错误，通过这些消息，可以更容易地识别并修复问题。

要想启动服务器，输入命令:

```
C: \> C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld --console
```

支持InnoDB的服务器启动时，你应当能看见下面的消息:

```
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
```

服务器完成启动序列后，你应当能看见下述内容，表示服务器准备好，可以进行客户端连接：

```
mysqld: ready for connections
Version: '5.1.2-alpha'  socket: ''  port: 3306
```

服务器继续向控制台写入诊断输出。你可以打开新的控制台窗口运行客户端程序。

如果省略--console选项，服务器向数据目录(默认为C:\Program Files\MySQL\MySQL Server 5.1\data)中的错误日志写入诊断输出。错误日志文件的扩展名为.err。

注释：MySQL授权表中的账户一开始没有密码。服务器启动后，你应当根据2.9节，“安装后的设置和测试”中的说明设置密码。

2.3.11. 从Windows命令行启动MySQL

可以从命令行手动启动MySQL服务器。可以在任何版本的Windows中实现。

要想从命令行启动**mysqld**服务器，你应当启动控制台窗口(或“DOS window”)并输入命令：

```
C: \> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld
```

根据系统中MySQL安装位置的不同，前面的例子中使用的路径也不同。

在非NT版本的Windows中，在后台启动**mysqld**。也就是，服务器启动后，你应当可以看见后面的命令提示。如果你用该方法在Windows NT、2000、XP或2003中启动服务器，服务器则在前台运行，在服务器退出前，不会出现命令提示。因此，当服务器运行时，应当打开另一个控制台窗口来运行客户端程序。

下述命令可以停止MySQL服务器：

```
C: \> C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin -u root shutdown
```

该命令调用MySQL管理工具**mysqladmin**连接服务器并告知它关闭服务。该命令使用MySQL root用户连接，这是MySQL权限系统中的默认管理账户。请注意MySQL权限系统中的用户完全独立于Windows下的登录用户。

如果**mysqld**不能启动，检查错误日志看看服务器是否写入了显示问题原因的任何消息。错误日志位于C:\Program Files\MySQL\MySQL Server 5.1\data目录中。是带后缀.err的文件。还可以尝试将服务器启动为**mysqld --console**；此时，可以从窗口中获得一些有用的信息，可以帮助解决问题。

最后选项是用--standalone --debug启动**mysqld**。此时，**mysqld** 写入日志文件C：\mysqld.trace，应当包含**mysqld**不启动的原因。请参见E.1.2节，“创建跟踪文件”。

使用**mysqld --verbose --help**显示**mysqld**的所有选项。

2.3.12. 以Windows服务方式启动MySQL

在NT家族 (Windows NT，2000,XP，2003)中，建议将MySQL安装为Windows服务，当Windows启动、停止时，MySQL也自动启动、停止。还可以从命令行使用**NET**命令，或使用图形**Services**工具来控制AMySQL服务器。

在Windows控制面板(Windows 2000,XP和Server 2003的管理工具下面)中可以看见**Services**工具(Windows **Service Control Manager**)。建议从命令行安装活卸载服务器时关闭**Services**工具。这样可以防止许多错误。

安装MySQL Windows服务之前，应当使用下面的命令先停止当前正运行的服务器：

```
C: \> C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqladmin -u root shutdown
```

注释：如果MySQL root用户账户有密码，你需要调用命令 **C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqladmin -u root -p shutdown**并根据提示输入密码。

该命令调用MySQL管理工具**mysqladmin**，连接服务器并告知它关闭服务。该命令使用MySQL root用户连接，这是MySQL权限系统中的默认管理账户。请注意MySQL权限系统中的用户完全独立于Windows下的登录用户。

使用该命令安装服务器

```
C: \> mysqld --install
```

如果只使用服务器名安装**mysqld**时有问题，尝试使用全路径名来安装。例如：

```
C: \> C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld --install
```

你还可以将MySQL bin目录的路径加到Windows系统PATH环境变量中：

- 右击Windows桌面上的My Computer（我的计算机）图标，选择Properties（属性）。
- 然后从出现的System Properties菜单中选择Advanced（高级）标签，然后点击Environment Variables（环境变量）按钮。
- 选择System Variables（系统变量）下面的Path（路径），然后点击Edit（编辑）按钮。将弹出Edit System Variable（编辑系统变量）对话框。
- 将光标放到标记Variable Value（变量值）处的文本的后面。(使用**End**键确保光标位于文本的最后)。然后输入MySQL bin目录 的完整路径(例如，C:\Program Files\MySQL\MySQL Server 5.1\bin)，请注意应使用分号将该路径与该域内的其它值隔离开。一次关闭该对话框和各个对话框，点击OK直到打开的所有对话框被关闭。你现在可以在DOS提示符下从系统的任何目录输入MySQL可执行程序名来调用MySQL可执行程序，而不必提供路径名。包括服务器、MySQL客户端和所有MySQL命令行实用程序，例如**mysqladmin**和**mysqldump**。
- 请注意在同一个机器上运行多个MySQL服务器时，不需要在Windows PATH中加入MySQL bin目录。

警告：编辑系统PATH时必须特别注意；意外删除或修改已有路径值会造成系统故障甚至瘫痪。

服务安装命令不会启动服务器。在本节后面将给出相关指令。

安装服务时，可以使用MySQL 5.1中的下面的参数：

- 可以直接在--install选项后面指定服务名。默认服务名是MySQL。
- 如果给出了服务名，后面可以跟一个选项。一般情况，应当为defaults-file=*file_name*，指定选项文件的名称，服务器启动时应当从中读取选项。

可以使用单个选项代替--defaults-file，但是不提倡这样做。--defaults-file更加灵活一些，因为你可以将多个服务器启动选项放入命名的选项文件中。

- 可以在服务名后面指定一个--local-service选项。这样服务器运行时使用LocalService Windows账户，只具有有限的系统权限。该账户只适用于Windows XP或更新版本。如果服务名后面同时跟有--defaults-file和--local-service，二者可以按任何顺序排放。

对于Windows MySQL服务器，根据下面规则确定服务器使用的服务名和选项文件：

- 如果在服务安装命令中，没有在--install选项后面指定了服务名或使用默认服务名(MySQL)，服务器则使用MySQL服务名并从标准选项文件的[mysqld]组读取选项。
- 如果在服务安装命令中，在--install选项后面指定的服务名不是默认服务名(MySQL)。则从具有相同服务名的组中读取选项，并从标准选项文件读取选项。

服务器还从标准选项文件的[mysqld]组读取选项。你可以使用[mysqld]组中的选项用于所有MySQL 服务，还可以使用具有相同服务名的组，用于该服务名所对应的服务器。

- 如果在服务安装命令中，在服务名后面指定了--defaults-file选项，服务器只从命名文件的[mysqld]组中读取选项，忽略标准选项文件。

对于更复杂的例子可以考虑使用命令：

```
C: \> C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld --install MySQL --defaults-file=C:\my-opts.cnf
```

该命令中，--install选项后面给出了默认服务名(MySQL)。如果未给出--defaults-file选项，该命令可以让服务器从标准选项文件的[mysqld]组中读数。由于提供了--defaults-file选项，服务器只从命名文件的[mysqld]组读取选项。

还可以在启动MySQL服务之前，在Windows **Services**工具中指定启动参数选项。

MySQL服务器安装为服务后，Windows启动时自动启动服务。还可以从**Services**工具直接启动服务，或使用命令**NET START MySQL**。**NET**命令忽略大小写。

做为服务运行时，**mysqld**不能访问控制台窗口，因此看不见任何消息。如果**mysqld**未启动，检查错误日志看看服务器是否写入了显示问题原因的任何消息。错误日志位于MySQL数据目录中(例

如，C:\Program Files\MySQL\MySQL Server 5.1\data)，是带后缀.err的文件。

如果MySQL服务器安装为服务，并且服务正运行，Windows关闭时自动停止服务。还可以通过Services工具、**NET STOP MySQL**命令或 **mysqladmin shutdown**命令手动停止服务器。

如果不想在引导过程中自动启动服务，还可以将服务设置为手动启动模式。实现的方法是使用--install-manual选项而非--install选项：

```
C: \> C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld --install-manual
```

要想卸载服务器，如果服务正运行则应首先使用命令**NET STOP MYSQL**停止服务。然后使用--remove选项卸载服务：

```
C: \> C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld --remove
```

如果**mysqld**不是服务，可以从命令行启动它。详细说明参见2.3.11节，“从Windows命令行启动MySQL”。

如果在安装过程中遇到问题，请参阅2.3.14节，“在Windows环境下对MySQL安装的故障诊断与排除”。

2.3.13. 测试MySQL安装

可以通过以下命令测试MySQL服务器是否工作：

```
C:\> C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqlshow

C:\> C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqlshow -u root mysql

C:\> C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqladmin version status proc

C:\> C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql test
```

如果**mysqld**对客户端程序TCP/IP连接的响应较慢，可能是DNS问题。此时，使用--skip-name-resolve选项启动**mysqld**，在MySQL授权表的Host列只使用localhost和IP号。

可以通过--pipe或--protocol=PIPE选项强制MySQL客户端使用命名管道连接代替TCP/IP连接，或指定.(阶段)做为主机名。使用--socket选项指定管道名。

2.3.14. 在Windows环境下对MySQL安装的故障诊断与排除

首次安装、运行MySQL时，你可能会遇到一些错误，使MySQL服务器不能启动。本节的目的是帮助你诊断并纠正这些错误。

解决服务器问题时你的第一资料是错误日志。MySQL服务器使用错误日志来记录服务器不能启动的信息。错误日志位于my.ini文件指定的数据目录中。默认数据目录位于C:\Program Files\MySQL\MySQL Server 5.1\dat。请参见5.11.1节，“错误日志”。

另一个错误相关信息源是MySQL服务启动时控制台上显示的消息。将**mysqld**安装为服务后，从命令行通过**NET START mysql**命令来查看MySQL服务启动相关的错误消息。请参见2.3.12节，“以Windows服务方式启动MySQL”。

下面距离说明了首次安装并启动MySQL服务时会遇到的最常见的错误消息：

- `System error 1067 has occurred.`
- `Fatal error: Can't open privilege tables: Table 'mysql.host' doesn't exist`
- 当MySQL服务器找不到MySQL权限数据库或关键文件时会出现这些消息。当MySQL基或数据目录没有安装在默认位置(C：\mysql和 C:\Program Files\MySQL\MySQL Server 5.1\data)，而是安装到其它位置时通常会遇到该问题。

发生该问题的一种情况是对MySQL进行升级或安装到了新位置，但是没有对配置文件进行更新以对应新的安装位置。此外还可能是新旧配置文件冲突，当升级MySQL时，一定要删除或重新命名旧的配置文件。

如果你没有将MySQL安装到C:\Program Files\MySQL\MySQL Server 5.1目录下，而是安装到了其它目录下，你需要通过使用配置(my.ini)文件让MySQL服务器知道安装目录。my.ini文件应位于Windows目录下，典型为C：\WINNT或C：\WINDOWS。可以在命令提示符下输入下面的命令从WINDIR环境变量值中确定确切位置：

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

可以通过文本编辑器（例如记事本）来创建并修改选项文件。例如，如果MySQL安装在E：\mysql下，数据目录在D:\MySQLdata，你可以创建选项文件并设置[mysqld]来指定basedir和datadir参数的值：

```
[mysqld]

# set basedir to your installation path

basedir=E:/mysql

# set datadir to the location of your data directory

datadir=D:/MySQLdata
```

请注意应使用(正)斜线而不是反斜线在选项文件中指定Windows路径名。如果使用反斜线，则必须使用双斜线：

```
[mysqld]
# set basedir to your installation path
basedir=C:\\Program Files\\MySQL\\MySQL Server 5.1
# set datadir to the location of your data directory
datadir=D:\\MySQLdata
```

如果在MySQL配置文件中更改datadir值，在重新启动MySQL服务器之前，必须移动已有MySQL数据目录中的内容。

参见2.3.8节，“创建选项文件”。

- `Error: Cannot create Windows service for MySql. Error: 0`

- 当你没有事先停止并卸载已有MySQL服务，并且使用MySQL Configuration Wizard（配置向导）重新安装或升级MySQL时，会遇到该问题。发生的原因是，当配置向导试图安装服务时，它发现已经有一个同名的服务。

解决该问题的一个方案是使用配置向导时选择mysql之外的其它服务名。这样可以正确安装新服务，但保留了原来的服务。但是这样不好，最好是卸掉不再使用的旧服务。

要想永远卸掉旧的MySQL服务，通过管理权限用户在命令行执行下面的命令：

```
C:\>sc delete mysql
[SC] DeleteService SUCCESS
```

如果Windows版本中没有sc工具，可以从 http://www.microsoft.com/windows2000/techinfo/reskit/tools/existing/delsrv-o.asp 下载delsrv工具并使用delsrvMySQL语法。

2.3.15. 在Windows下升级MySQL

本节列出了在Windows中升级MySQL应采取的步骤。

1. 进行升级前你应先备份当前的MySQL安装。请参见5.9.1节，“数据库备份”。
2. 从http://dev.mysql.com/downloads/下载最新Windows版MySQL。
3. 升级MySQL前，必须停止服务器。

如果服务器安装为服务，必须在命令提示符下在命令行中用下面的命令停止服务：

```
C: \> NET STOP MYSQL
```

如果MySQL服务器不是服务，使用下面的命令停止服务器：

```
C: \> C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqladmin -u root shutdown
```

4. 当从以前至的4.1.5版本升级到MySQL 5.1时,或从Zip文件安装的MySQL升级到MySQL Installation Wizard（安装帮助）安装的MySQL时，你必须手动卸载前面安装的MySQL服务。

要想卸载MySQL服务，使用下面的命令：

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

如果你不想卸载已有的服务，**MySQL Installation Wizard**（安装帮助）则不会正确安装新的**MySQL**服务。

5. 如果你使用MySQL Installation Wizard（安装帮助），按照2.3.4节，“使用MySQL安装向导”的描述来启动帮助。
6. 如果用Zip文件安装MySQL，应提取文件。你可以覆盖已有的MySQL安装(通常位于C：\mysql)，或将它安装到其它目录，例如 C：\mysql4。建议覆盖原有的安装。
7. 重新启动服务器。如果你运行MySQL服务，则使用**NET START MySQL**，还可以直接调用**mysqld**。
8. 关于在Windows之外的操作系统升级MySQL的详细信息，请参阅2.10节，“升级MySQL”。
9. 如果遇到错误，参见 2.3.14节，“在Windows环境下对MySQL安装的故障诊断与排除”。

2.3.16. Windows版MySQL同Unix版MySQL对比

已经证明，Windows版MySQL很稳定。Windows版MySQL的功能与相应的Unix版相同，只有以下例外：

- **Windows 95**和线程

Windows 95创建一个线程时大约需要200字节的主内存。MySQL的每个连接都会创建一个新线程，因此如果你的服务器正处理许多连接，你不应当在Windows 95中运行**mysqld**。

- 有限的端口数目

Windows系统有大约4,000个端口供客户端连接，某个端口的连接关闭后，在能够重新利用该端口前，需要2至4分钟。在客户端频繁连接并从服务器上断开时，在可以重新使用关闭的端口前，有可能用完了所有可用的端口。如果发生这种情况，MySQL服务器不会响应，即使它仍在运行。请注意机器上运行的其它应用程序也可以使用端口，此时可供MySQL使用的端口数要少。

详细信息参见http://support.microsoft.com/default.aspx?scid=kb;en-us;196271 。

- 并行读

MySQL依靠pread()和pwrite()系统调用来同时使用INSERT和SELECT。目前我们使用互斥来竞争pread()和pwrite()。我们将来想用虚拟接口替换文件级接口，因此要想更快，我们可以在NT、2000和XP上使用readfile()/writefile()接口。目前MySQL 5.1可以打开的文件的限制数目为2,048,意味着在Windows NT，2000,XP和2003上可以运行的并行线程不如Unix上多。

- 阻塞读

MySQL为每个连接使用阻塞读取，如果启用了命名管道连接，其含义如下：

o 连接不会在8小时后自动断开，而在Unix版MySQL中会发生。

o 如果连接被挂起，不杀掉MySQL则不会将其中断。

o **mysqladmin kill**不会杀掉睡眠中的连接。

o 只要有睡眠连接，**mysqladmin shutdown**不会中断。

我们计划在将来修复该问题。

- **ALTER TABLE**

执行ALTER TABLE语句时，将该表锁定不让其它线程使用。在Windows中，你不能删除正被另一个线程使用的文件。在将来，我们会找出办法解决该问题。

- **DROP TABLE**

在Windows中对一个被MERGE表应用的表执行DROP TABLE不会实现，因为MERGE处理器将表从MySQL上层映射隐藏起来。由于Windows不允许删除打开的文件，必须在删除表之前首先清空所有MERGE表(使用FLUSH TABLES)或删掉MERGE表。

- **DATA DIRECTORY and INDEX DIRECTORY**

在Windows中将忽略DATA DIRECTORY和INDEX DIRECTORY选项，因为Windows不支持符号连接。在具有非功能realpath()调用的系统中，这些选项也被忽略。

- **DROP DATABASE**

你不能删掉正被线程使用的数据库。

- 从**Task Manager**（任务管理器）杀掉**MySQL**

你不能从Task Manager（任务管理求）或使用Windows 95的shutdown工具来杀掉MySQL。你必须通过**mysqladmin shutdown**停止它。

- 大小写名

由于Windows对文件名大小写不敏感。因此在Windows中MySQL数据库名和表名对大小写也不敏感。唯一的限制是在同一个语句中，必须同时使用大写或小写指定数据库名和表名。请参见9.2.2节，“识别符大小写敏感性”。

- **'\'**路径名间隔符

Windows中的路径名用'\'符间隔开，在MySQL中还是转义字符。如果你使用LOAD DATA INFILE或SELECT ... INTO OUTFILE，用'/'符使用Unix-类文件名：

```
mysql> LOAD DATA INFILE 'C:/tmp/skr.txt' INTO TABLE skr;
mysql> SELECT * INTO OUTFILE 'C:/tmp/skr.txt' FROM skr;
```

你还可以使用双'\'符：

```
mysql> LOAD DATA INFILE 'C:\\tmp\\skr.txt' INTO TABLE skr;
mysql> SELECT * INTO OUTFILE 'C:\\tmp\\skr.txt' FROM skr;
```

- 管道问题。

管道不能在Windows命令行提示符下可靠地工作。如果管道包括字符^Z/CHAR(24)，Windows认为遇到了文件末尾并中断程序。

这主要是按照如下所示使用二进制日志的主要问题：

```
C: \>mysqlbinlog binary-log-name | mysql --user=root
```

如果使用日志时出现问题，怀疑是由于^Z / CHAR(24)字符，你可以使用下面的程序：

```
C:\> mysqlbinlog binary-log-file --result-file=/tmp/bin.sql
C:\> mysql --user=root --execute "source /tmp/bin.sql"
```

后面的命令还可以用来可靠读取任何包含二进制数据的SQL文件。

- **Access denied for user**错误

如果你试图运行MySQL客户端程序来连接同一机器上运行的服务器，但是遇到错误Access denied for user '*some-user*'@'unknown' to database 'mysql'，这意味着MySQL不能正确识别你的主机名。

要解决该问题，你应当创建一个名为\windows\hosts包含下面信息的文件：

```
127.0.0.1       localhost
```

这儿有一些公开问题，提供给想要帮助我们改进Windows中的MySQL的人们：

- 增加宏来使用Windows提供的更快的线程安全增/减方法。

2.4. 在Linux下安装MySQL

建议在Linux中使用RPM包来安装MySQL。MySQL RPM目前已经嵌入到SuSE Linux 7.3系统中，但是应当能在大多数支持**rpm**和使用glibc的Linux版本中工作。要想获得RPM软件包，参见2.1.3节，“怎样获得MySQL”。

MySQL AB不提供与具体平台相关的RPM；具体平台相关的RPM和通用RPM之间的区别是具体平台相关RPM为目标平台而构建，为动态连接。而通用RPM与Linux线程之间是静态连接。

注释：通常由其它供应商提供MySQL的RPM分发版。其特征和功能与MySQL AB所构建的不同，该手册中的指令不一定适合安装其它供应商提供的MySQL的RPM分发版。此时应咨询供应商的说明。

如果RPM文件有问题(例如，如果你收到错误“Sorry, the host '*xxxx*' could not be looked up”)，参见2.12.1.2节，“Linux二进制分发版说明”。

在大多数情况，你只需要安装MySQL-server和MySQL-client软件包来安装MySQL。在标准安装中不需要其它的包。如果你想要运行含更多功能的MySQL-Max服务器，还应当安装MySQL-Max RPM。必须先安装M MySQL-server RPM再安装该模块。请参见5.1.2节，“mysq**ld-max**扩展**My**SQL服务器”。

如果安装MySQL软件包时出现从属错误(例如，“error: removing these packages would break dependencies: libmysqlclient.so.10 is needed by ..”)，你还应当安装包MySQL-shared-compat，其中包括两个向后兼容的共享库(MySQL 4.0为libmysqlclient.so.12，MySQL 3.23为libmysqlclient.so.10)。

许多Linux分发版包含了MySQL 3.23，通常动态连接应用程序以节省硬盘空间。如果这些共享库在单独的包内(例如，MySQL-shared)，只需要安装这些包并升级MySQL服务器和客户端软件包(为静态连接，不依赖共享库)。对于包含与MySQL服务器相同软件包中的共享库的分发版(例如，Red Hat Linux)，你可以安装3.23 MySQL-shared RPM，或使用MySQL-shared-compat包。

可以使用以下RPM包：

- MySQL-server-*VERSION*.i386.rpm

MySQL服务器。你需要该选项，除非你只想连接运行在另一台机器上的MySQL服务器。 注释：在MySQL 4.0.10之前，服务器RPM文件被称为MySQL-*VERSION*.i386.rpm。也就是说，名称中不含有-server。

- MySQL-Max-*VERSION*.i386.rpm

MySQL-Max服务器。该服务器具有其它功能，这是MySQL-server RPM不具备的。你必须首先安装MySQL-server RPM，因为MySQL-Max RPM依赖于它。

- MySQL-client-*VERSION*.i386.rpm

标准MySQL客户端程序。你可能总是要安装该软件包。

- MySQL-bench-*VERSION*.i386.rpm

测试和基准。需要Perl和DBD：：mysql模块。

- MySQL-devel-*VERSION*.i386.rpm

库和包含文件，如果你想要编译其它MySQL客户端，例如Perl模块，则需要。

- MySQL-shared-*VERSION*.i386.rpm

该软件包包含某些语言和应用程序需要动态装载的共享库(libmysqlclient.so*)，使用MySQL。

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

该软件包包括MySQL 3.23和MySQL 4.0的共享库。如果你安装了应用程序动态连接MySQL 3.23，但是你想要升级到MySQL 4.0而不想打破库的从属关系，则安装该软件包而不要安装MySQL-shared。从MySQL 4.0.13起包含该安装软件包。

- MySQL-embedded-*VERSION*.i386.rpm

嵌入式MySQL服务器库(从MySQL 4.0起)。

- MySQL-*VERSION*.src.rpm

包含以前所有软件包的源码。可用来在其它架构上重建RPM(例如，Alpha或SPARC)。

要想看到RPM软件包内的所有文件(例如，MySQL-server RPM)，运行：

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

要想执行标准最小安装，运行：

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

要想只安装客户端软件包，运行：

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

RPM提供在安装软件包前验证其完整性和真实性的功能。如果你要想了解关于该功能的更多信息，参见2.1.4节，“通过MD5校验和或GnuPG验证软件包的完整性”。

服务器RPM将数据放入/var/lib/mysql目录。RPM还为mysql用户创建登录账户(如果还没有)，用来运行MySQL服务器，并在/etc/init.d/创建相应条目，以便在引导时自动启动服务器。(这意味着如果你已经安装并且已经更改启动脚本，你可能要复制脚本，以便你安装新RPM时 不会丢掉脚本)。关于系统启动时如何自动启动MySQL的详细信息，参见 2.9.2.2节，“自动启动和停止MySQL”。

如果想要在旧的不支持/etc/init.d(直接或通过符号连接)初始化脚本的Linux分发版中安装MySQL RPM，你应当创建一个指向初始化脚本实际安装的位置的符号连接。例如，如果安装位置为/etc/rc.d/init.d，安装RPM前使用这些命令创建/etc/init.d，作为指向该位置的符号连接：

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

目前主要Linux分发版应当支持使用/etc/init.d的新目录布局，因为需要LSB(Linux标准基)兼容性。

如果你安装的RPM文件包括MySQL-server，安装后MySQL**d**服务器应当能启动并运行。你应当能使用MySQL启动。

如果出现问题，可以从二进制安装部分找到详细信息。2.7节，“在其它类Unix系统中安装MySQL”。

注释：MySQL授权表中的账户开始没有密码。启动服务器后，应当按照2.9节，“安装后的设置和测试”中的说明为它们设置密码。

2.5.在Mac OS X中安装MySQL

可以在Mac OS X 10.2.x(“Jaguar”)和以上版本上Mac OS X使用二进制安装软件包的PKG格式代替二进制分发版来安装MySQL。请注意该软件包不支持旧版本的Mac OS X(例如，10.1.x)。

安装软件包位于硬盘映象(.dmg)文件中，必须首先双击搜索起中的图标来安装该文件。应当安装图像并显示其内容。

要想获得MySQL，参见2.1.3节，“怎样获得MySQL”。

注释：在继续安装前，一定要使用MySQL管理器应用程序(在Mac OS X服务器上)或通过命令行的**mysqladmin shutdown**关闭所有运行MySQL服务器实例。

要想安装MySQL PKG文件，双击软件包图标。将启动Mac OS X安装软件包安装器，将指导你完成MySQL安装。

Mac OS X安装软件包的安装器中有一个缺陷，你会在目标盘选择对话框中发现该错误消息：

```
You cannot install this software on this disk. (null)
```

如果出现该错误，只需要点击Go Back按钮返回前面的窗口。然后点击Continue再次选择目标盘，此时应当可以正确地选择目标盘。我们已经向Apple报告了该缺陷，它正在调查该问题。

MySQL的Mac OS X PKG安装到/usr/local/mysql-*VERSION*，并且还会安装一个符号连接，/usr/local/mysql，指向新的位置。如果有/usr/local/mysql目录，首先将它改名为/usr/local/mysql.bak。安装完后，安装器执行**mysql_install_db**在MySQL数据库中创建授权表。

安装布局与**tar**文件二进制分发版类似；所有MySQL二进制代码位于目录/usr/local/mysql/bin中。默认情况，MySQL套接字文件创建为/tmp/mysql.sock。请参见2.1.5节，“安装布局”。

MySQL安装需要Mac OS X的mysql用户账户。在Mac OS X 10.2及以上版本中，默认情况应存在mysql用户账户。

如果正运行Mac OS X服务器，需要安装MySQL。随Mac OS X服务器版本安装的MySQL版本如下表所示：

Mac OS X服务器版本	**MySQL**版本
10.2-10.2.2	3.23.51
10.2.3-10.2.6	3.23.53
10.3	4.0.14
10.3.2	4.0.16
10.4.0	4.1.10a

本节只覆盖官方MySQL Mac OS X PKG版本。一定要阅读Apple关于MySQL安装的帮助信息：运行“帮助视图”应用程序，选择“Mac OS X服务器”帮助，搜索“MySQL”，阅读标题为“安装MySQL”的内容。

对于Mac OS X服务器上已经安装的MySQL版本，如果早于4.0，应特别注意应当使用**mysqld_safe**而不是**safe_mysqld**来启动**mysqld**。

如果你前面在Mac OS X上安装了从http://www.entropy.ch下载的Marc Liyanage的MySQL软件包，你可以只使用该页给出的二进制安装布局按照更新说明来安装。

如果你从Marc的3.23.xx 版本或从Mac OS X的MySQL服务器版本升级到官方MySQL PKG，你还需要将已有MySQL授权表转换为当前格式，因为已经加了许多新的安全权限。请参见2.10.2节，“升级授权表”。

如果你想在系统启动过程中自动启动MySQL，你还需要安装MySQL Startup Item。在MySQL 5.1中,它是Mac OS X安装盘映像的一部分，是一个独立的安装软件包。只需要双击MySQLStartupItem.图标并按照说明来安装。

请注意Startup Item只需要安装一次！不需要在以后每次升级MySQL软件包时都安装它。

MySQL 5.1的Startup Item安装在/Library/StartupItems/MySQLCOM。(在MySQL 4.1.2以前的版本,安装位置为/Library/StartupItems/MySQL，但与Mac OS X服务器安装的MySQL Startup Item冲突)。Startup Item安装时在系统配置文件/etc/hostconfig中加入变量MySQLCOM=-YES-。如果你想要disable自动启动MySQL，只需要将该变量更改为MySQLCOM=-NO-。

在Mac OS X服务器上，默认MySQL安装使用/etc/hostconfig文件中的变量MySQL。MySQL AB Startup Item安装器将其设置为MySQL=-NO-禁用该变量。这样可以避免启动时间与MySQL AB Startup Item使用的MySQLCOM变量冲突。但是，不会关闭运行的MySQL服务器。你应当自己关闭。

安装完后，可以在终端窗口运行下面的命令来启动MySQL。你必须具有管理员权限来执行该任务。

如果你已经安装的Startup Item：

shell> **sudo /Library/StartupItems/MySQLCOM/MySQLCOM start**

(Enter your password, if necessary)

(Press Control-D or enter "exit" to exit the shell)

如果不使用Startup Item，输入下面的命令序列：

shell> **cd /usr/local/mysql**

shell> **sudo ./bin/mysqld_safe**

(Enter your password, if necessary)

(Press Control-Z)

shell> **bg**

(Press Control-D or enter "exit" to exit the shell)

你应当能够连接MySQL服务器，例如，通过运行/usr/local/mysql/bin/mysql来连接。

注释：MySQL授权表中的账户开始没有密码。启动服务器后，应当按照2.9节，“安装后的设置和测试”中的说明来设置密码。

你可以为外壳源文件起一个别名，使它更容易访问常用的程序，例如从命令行运行**mysql**和**mysqladmin**。**tcsh**语法为：

```
aliasMySQL /usr/local/mysql/bin/mysql
aliasMySQLadmin /usr/local/mysql/bin/mysqladmin
```

对于**bash**，使用：

```
aliasMySQL=/usr/local/mysql/bin/mysql
aliasMySQLadmin=/usr/local/mysql/bin/mysqladmin
```

最好是在PATH环境变量中加入/usr/local/mysql/bin。例如，如果外壳命令是**tcsh**，在$HOME/.tcshrc文件中加入下面一行：

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

如果根目录中没有.tcshrc文件，则用文本编辑器创建该文件。

如果你对已有的安装进行升级，请注意安装新MySQL PKG时不会卸载旧的安装目录。遗憾的是，Mac OS X安装器尚未提供功能来正确升级前面安装的软件包。

要想在新安装中使用已有的数据库，你需要将旧数据目录中的内容复制到新的数据目录。执行该操作时，确保关闭新、旧服务器。将以前安装的MySQL数据库文件拷贝过来并成功启动新的服务器后，应当卸载旧的文件以节省硬盘空间。并且，还应当卸掉位于/Library/Receipts/mysql-*VERSION*.pkg的旧版本Package Receipt目录。

2.6. 在NetWare中安装MySQL

将MySQL装入到NetWare首先由Novell实现。Novell客户应当满意，NetWare 6.5中捆绑了SQL二进制，并具有自动商业使用许可，用于运行该版本NetWare的所有服务器。

NetWare的MySQL使用NetWare的Metrowerks CodeWarrior和专用跨编译版GNU自动工具编译而成。

可以从http://dev.mysql.com/downloads/获得最新的二进制软件包NetWare。请参见2.1.3节，“怎样获得MySQL”。

为了使MySQL驻留在主机上，NetWare服务器必须满足以下要求：

- 安装的NetWare 6.5 或NetWare 6.0的Latest Support Pack（最新支持补丁）。
- 系统必须满足Novell的最低要求以运行相关版本的NetWare。
- MySQL数据及其二进制，必须安装到NSS卷内；不支持传统卷。

要想在NetWare下安装MySQL，遵从下面的步骤：

1. 如果你对前面的安装进行升级，停止MySQL服务器。应使用下面的命令从服务器控制台完成：

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

3. 从客户端登录到目标服务器上，访问安装MySQL的位置。
4. 将二进制软件包Zip文件提取到服务器上。一定要使用Zip文件中的路径。简单地将文件提取到SYS：\是很安全的。

如果你从前面的安装升级，你可能需要复制数据目录(例如，SYS:MYSQL\DATA)和my.cnf（如果你已经定制）。然后你可以删除旧版本的MySQL。

5. 你可以将目录重新命名为更加一致和易于使用的名称。在本手册的例子中使用SYS：MYSQL来指向安装目录。

请注意在NetWare上安装MySQL不会检测到是否已经安装了NetWare发布之外的MySQL版本。因此，如果你已经在SYS：\MYSQL安装了从网站上下载的最新MySQL版本(例如，MySQL 4.1或更新版本)，在升级NetWare服务器之前，必须重新命名文件夹；否则，SYS：\MySQL中的文件会被NetWare Support Pack中的版本所覆盖。

6. 在服务器控制台中，加入含有 MySQL NLM的目录的搜索路径。例如：

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

8. 初始化数据目录和授权表，如果需要，应在服务器控制台执行**mysql_install_db**。

9. 在服务器控制台使用**mysqld_safe**启动MySQL服务器。

10.要完成安装，还应当在autoexec.ncf中加入以下命令。例如，如果MySQL安装在SYS：MYSQL中，你想让MySQL自动启动，可以加入以下行：

```
11.         #Starts the MySQL 5.1.x database server
12.        SEARCH ADD SYS:MYSQL\BIN
13.        MYSQLD_SAFE
```

如果MySQL运行在NetWare 6.0上,我们强烈建议你在命令行中使用--skip-external-locking选项：

```
#Starts the MySQL 5.1.x database server
SEARCH ADD SYS:MYSQL\BIN
MYSQLD_SAFE --skip-external-locking
```

还需要使用CHECK TABLE和REPAIR TABLE来替代**myisamchk**，因为**myisamchk**使用外部锁。已经知道外部锁在NetWare 6.0中有问题；在NetWare 6.5中已经解决了该问题。

NetWare中的**mysqld_safe**提供了窗口保留功能。当卸载(关闭)**mysqld_safe** NLM时，默认情况窗口不会消失。相反，会提示用户输入：

```
*<NLM has terminated; Press any key to close the screen>*
```

如果你想让NetWare自动关闭窗口，在**mysqld_safe**中使用--autoclose选项。例如：

```
#Starts the MySQL 5.1.x database server
SEARCH ADD SYS:MYSQL\BIN
MYSQLD_SAFE --autoclose
```

14.当首次安装或从前面的版本升级MySQL 5.1版时，应下载并安装最新的相应Perl模块和PHP扩展名：

- Perl for NetWare: http://forge.novell.com/modules/xfcontent/downloads.php/perl/Modules/
- PHP for NetWare: http://forge.novell.com/modules/xfcontent/downloads.php/php/Modules/

(MySQL 4.1的PHP 5扩展名适用于MySQL 5.1)。

在5.1.3节，“mysqld_safe：MySQL服务器启动脚本”中详细描述了NetWare中的**mysqld_safe**的行为。

如果服务器上已经安装了MySQL，一定要检查autoexec.ncf中已有的MySQL启动命令，如果需要则编辑或删除这些命令。

注释：MySQL授权表中的初始账户没有密码。启动服务器后，你应当使用2.9节，“安装后的设置和测试”中的说明来设置密码。

2.7. 在其它类Unix系统中安装MySQL

本节介绍在各种平台上安装**tar**压缩文件格式的MySQL二进制分发版(文件有.tar.gz 扩展名)。详细目录参见2.1.2.5节，“MySQL AB编译的MySQL二进制版本”。

要想获得MySQL，参见2.1.3节，“怎样获得MySQL”。

MySQL**tar**文件二进制分发版的文件名格式为mysql-*VERSION*-*OS*.tar.gz，其中*VERSION*是版本号(例如，5.1.2-alpha)，*OS*表示分发版安装的操作系统类型(例如，pc-linux-i686)。

除了这些通用软件包，我们还为各平台提供了具体的二进制安装软件包。关于如何安装的详细信息参见2.2节，“使用二进制分发版的标准MySQL安装”。

你需要下面的工具来安装MySQL **tar**文件二进制分发版：

- GNU gunzip解压缩分发版。
- 一个适当的tar解包分发版。 GNU tar已知可以。一些操作系统中含有预安装的**tar**，已知有问题。例如，已知Mac OS X **tar**和Sun **tar**对于长文件名有问题。在Mac OS X中，你可以使用预安装的**gnutar**程序。在其它系统中若**tar**不完善，你应当首先安装GNU **tar**。

如果你遇到问题，在把问题寄到MySQL邮件目录时，请一定使用**mysqlbug**，就算问题不是一个缺陷，**mysqlbug**也会收集系统信息，将有助于其它人解决你的问题。不使用**mysqlbug**，你将减少得到你问题的一个解决方案的可能性！在你解包分发版后，你会在“bin”目录下找到**mysqlbug**。请参见1.7.1.3节，“如何通报缺陷和问题”。

必须执行以便安装并使用**MySQL**二进制分发版的基本命令是：

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 --user=mysql**

shell> **chown -R root .**

shell> **chown -R mysql data**

shell> **chgrp -R mysql .**

shell> **bin/mysqld_safe --user=mysql &**

注释：该过程未设置MySQL账户的密码。完成下面的过程后，继续执行想2.9节，“安装后的设置和测试”。

下面为安装二进制分发版的更详细的描述：

1. 为**mysqld**增加一个登录用户和组：

2. shell> **groupadd mysql**

3. shell> **useradd -g mysql mysql**

这些命令将增加MySQL组和MySQL用户。不同版本的Unix中，**useradd**和**groupadd**的语法可能会稍有不同。还可以称为**adduser**和**addgroup**。

你可能会将这些用户和组命名为mysql之外的名称。如果是这样，在下面的步骤中替换为相应的名称。

4. 挑选一个你想解开分发包的目录，进入该目录。在下面的例子中，我们将分发解包在“/usr/local”下（因此，下列说明假设你有权限在“/usr/local”中创建文件和目录，如果该目录被保护，你需要以root实施安装）。

5. shell> **cd /usr/local**

6. 从2.1.3节，“怎样获得MySQL”中列出的站点之一获得一个分发文件。对于一个给定的版本，所有平台的二进制分发版均从相同的MySQL源码分发版构建。

7. 解包分发版，将创建安装目录。然后生成到该目录的一个符号链接：

8. shell> **gunzip <** */path/to/mysql-VERSION-OS***.tar.gz | tar xvf -**

9. shell> **ln -s** *full-path-to-mysql-VERSION-OS* **mysql**

tar命令创建mysql-*VERSION-OS*目录。ln命令生成到该目录的一个符号链接。这让你更容易地把安装目录指定为/usr/local/mysql。

使用GNU **tar**，则不再需要gunzip。可以用下面的命令替换第一行来解包和提取分发版：

shell> **tar zxvf** ***/path/to/mysql-VERSION-OS*****.tar.gz**

10. 进入安装目录：

11. shell> **cd mysql**

你会在mysql目录下发现几个文件和子目录，对安装目的最重要的是“bin”和“scripts”子目录。

- bin

这个目录包含客户端程序和服务器，你应该把这个目录的完整路径加到PATH环境变量，以便shell能正确的找到MySQL程序。请参见附录F：环境变量。

- scripts

这个目录包含mysql_install_db脚本，用来初始化mysql数据库的 授权表，其中贮存了服务器访问允许。

12. 如果还没有安装MySQL，必须创建MySQL授权表：

```
13.        shell> scripts/mysql_install_db --user=mysql
```

如果你用root运行命令，应当使用--user选项。选项的值应与你在第一步为运行服务器所创建的登录账户相同。如果你用该用户登录来运行命令，可以省略--user选项。

创建或升级授权表后，需要手动重新启动服务器。

14.将程序二进制的所有权改为root，数据目录的所有权改为运行**mysqld** 的用户。如果位于安装目录(/usr/local/mysql)下，命令应为：

15. shell> **chown -R root .**

16. shell> **chown -R mysql data**

17. shell> **chgrp -R mysql .**

第一个命令将文件的所有属性改为root用户。第二个命令将数据目录的所有属性改为mysql用户。第三个命令将组属性改为mysql组。

18.如果你喜欢在引导机器时自动启动MySQL，可以拷贝support-files/mysql.server文件到系统有启动文件的地方。更多的信息可以在support-files/mysql.server脚本中和2.9.2.2节，“自动启动和停止MySQL”中找到。

19.如果安装了DBI和DBD::mysql Perl模块，可以使用**bin/mysql_setpermission**脚本增加新账户。关于说明，参见 2.13节，“Perl安装注意事项”。

20.如果你想使用**mysqlaccess**而在某个非标准的地点有MySQL分发版，你必须改变地点，在那儿**mysqlaccess**期望找到**mysql**客户端。编辑“bin/mysqlaccess”脚本，大约在第18行，寻找类似下面的行：

```
21.        $MYSQL     = '/usr/local/bin/mysql';    # path to mysql executable
```

改变路径以反映出**mysql**实际存储在你系统上的地点。如果不这样做，当运行**mysqlaccess**时，你将遇到一个broken pipe的错误。

在所有东西被解包并且安装以后，你应该初始化并且测试你的分发版。

可以用下列命令启动MySQL服务器：

```
shell> bin/mysqld_safe --user=mysql &
```

关于**mysqld_safe**的详细信息参见5.1.3节，“mysqld_safe：MySQL服务器启动脚本”。

注释：MySQL授权表中的账户开始没有密码。启动服务器后，应当使用2.9节，“安装后的设置和测试”中的说明来设置密码。

2.8. 使用源码分发版安装MySQL

2.8.1. 源码安装概述
2.8.2. 典型配置选项
2.8.3. 从开发源码树安装
2.8.4. 处理MySQL编译问题
2.8.5. MIT-pthreads注意事项
2.8.6. 在Windows下从源码安装MySQL
2.8.7. 在Windows下编译MySQL客户端

安装源码之前，首先检查我们的二进制是否适合你的平台，是否可以工作。我们做了大量的工作确保我们的二进制用最合适的选项来构建。

要获得MySQLa 源码分发版，参见2.1.3节，“怎样获得MySQL”。

MySQL源码分发版为**tar**压缩文件，文件名格式为mysql-*VERSION*.tar.gz，其中 *VERSION*是版本号，例如5.1.2-alpha。

你需要下列工具构造并安装**MySQL**源码：

- GNU gunzip解压缩该分发版。
- 一个适当的tar解包分发版。GNU tar已知可以。一些操作系统中含有预安装的**tar**，已知有问题。例如，已知Mac OS X **tar**和Sun **tar**对于长文件名有问题。在Mac OS X中，你可以使用预安装的**gnutar**程序。在其它系统中若**tar**不完善，你应当首先安装GNU **tar**。
- 一个能工作的ANSI C++编译器。**gcc**> = 2.95.2、**egcs**> = 1.0.2或**egcs 2.91.66**、SGI C++和SunPro C++是一些已知能工作的编译器。当使用**gcc**时，不需要libg++。gcc2.7.x有一个缺陷，使得它可能不能编译一些完全合法的C++文件，例如“sql/sql_base.cc”。如果你仅有gcc2.7.x，你必须升级gcc才能编译MySQL。已知**gcc** 2.8.1在某些平台上有一些问题，因此如果平台上有新的编译器，则不要使用**gcc** 2.8.1。

编译MySQL 3.23.x时，建议使用**gcc** 2.95.2或更新版。

- 一个优秀的**make**程序。总是推荐GNU **make**且有时必须使用。如果你有问题，我们推荐试一试GNU **make**3.75或更新版。

如果你使用的**gcc**版本足够新，可以识别-fno-exceptions选项，则使用该选项非常重要。否则，你编译二进制时出现问题。我们建议你同时使用-felide-constructors和-fno-rtti选项。当有疑问时，执行下面操作：

```
CFLAGS="-O3" CXX=gcc CXXFLAGS="-O3 -felide-constructors \
       -fno-exceptions -fno-rtti" ./configure \
       --prefix=/usr/local/mysql --enable-assembler \
       --with-mysqld-ldflags=-all-static
```

在大多数系统中，可以得到快速、稳定的二进制。

如果你遇到问题，在把问题邮寄到MySQL邮件目录时，请总是使用**mysqlbug**，就算问题不是一个缺陷，**mysqlbug**收集系统信息，将有助于其它人解决你的问题。不使用**mysqlbug**，你将减少得到问题的解决方案的可能性！在解包分发版后，你会在“bin”目录下找到**mysqlbug**。请参见1.7.1.3节，“如何通报缺陷和问题”。

2.8.1. 源码安装概述

你必须执行的安装MySQL源码分发版的基本命令是：

```
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> cp support-files/my-medium.cnf /etc/my.cnf
shell> cd /usr/local/mysql
shell> bin/mysql_install_db --user=mysql
shell> chown -R root  .
shell> chown -R mysql var
shell> chgrp -R mysql .
shell> bin/mysqld_safe --user=mysql &
```

如果从一个源码RPM开始，那么执行如下命令：

```
shell> rpmbuild --rebuild --clean MySQL-VERSION.src.rpm
```

这样你制作一个可以安装的二进制RPM。对于旧版本的RPM，你可能需要替换用**rpm**替换命令**rpmbuild**。

注释：该过程不为任何MySQL账户设置密码。完成下面的过程后，根据2.9节，“安装后的设置和测试”进行安装后的设置和测试。

以下是安装MySQL源码分发版的更详细的描述：

1. 为**mysqld**增加一个登录用户和组：

```
2.          shell> groupadd mysql
3.          shell> useradd -g mysql mysql
```

这些命令将增加MySQL组和 MySQL用户。不同版本的Unix中，**useradd** 和**groupadd**的语法可能会稍有不同。还可以称为**adduser**和**addgroup**。

你可能会将这些用户和组命名为mysql之外的名称。如果是这样，在下面的步骤中替换为相应的名称。

4. 挑选一个你想解开分发包的目录，进入该目录。
5. 从2.1.3节，“怎样获得MySQL”中列出的站点之一获得一个分发文件。
6. 在当前目录下解包分发版：

```
7.          shell> gunzip < /path/to/mysql-VERSION.tar.gz | tar xvf -
```

该命令创建mysql-*VERSION* 目录。

使用GNU **tar**，则不再需要gunzip。你可以用下面的命令来解包和提取分发：

```
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
```

8. 进入解包分发版的顶级目录：

```
9.          shell> cd  mysql-VERSION
```

请注意目前必须从顶级目录配置和构建MySQL。你不能在其它目录中构建。

10.配置发布版并且编译：

```
11.       shell> ./configure --prefix=/usr/local/mysql
12.       shell> make
```

当你运行**configure**时，你可能想要指定一些选项，运行**./configure --help**得到一个选项列表。2.8.2节，“典型配置选项”讨论一些很有用的选项。

如果**configure**失败，你将向MySQL邮件目录发送包含你认为能帮你解决该问题的包含“config.log”中所有行的邮件，也要包括**configure**的最后几行输出。用**mysqlbug**脚本邮寄错误报告。请参见1.7.1.3节，“如何通报缺陷和问题”。

如果编译失败，参见2.8.4节，“处理MySQL编译问题”寻求帮助。

13.安装分发版：

```
14.       shell> make install
```

如果你想要设置一个选项文件，使用support-files目录中的一个作为模板。例如：

```
shell> cp support-files/my-medium.cnf /etc/my.cnf
```

可能你需要用root用户运行这些命令。

如果想要配置为支持InnoDB表，应当编辑/etc/my.cnf文件，去掉innodb_...开始选项行前面的#符，并将选项值改为你想要的。请参见4.3.2节，“使用选项文件”和15.2.3节，“InnoDB配置”。

15.进入安装目录：

```
16.       shell> cd /usr/local/mysql
```

17.如果还没有安装的MySQL，必须创建MySQL授权表：

```
18.       shell> bin/mysql_install_db --user=mysql
```

如果用root运行命令，应当使用--user选项。选项的值应与你在第一步为运行服务器所创建的登录账户相同。如果用该用户登录来运行命令，可以省略--user选项。

用**mysql_install_db**创建MySQL授权表后，需要手动重新启动服务器

19.将程序二进制的所有权改为root，数据目录的所有权改为你运行**mysqld**的用户。如果位于安装目录(/usr/local/mysql)下，命令应为：

```
20.       shell> chown -R root  .
21.       shell> chown -R mysql var
22.       shell> chgrp -R mysql .
```

第一个命令将文件的所有属性改为root用户。第二个命令将数据目录的所有属性改为mysql用户。第三个命令将组属性改为mysql组。

23.如果你喜欢在引导机器时自动启动MySQL，你可以拷贝support-files/mysql.server文件到你系统有启动文件的地方。更多的信息可以在support-files/mysql.server脚本中和2.9.2.2节，“自动启动和停止MySQL”中找到。

24.如果安装了DBI和DBD::mysql Perl模块，可以使用**bin/mysql_setpermission**脚本增加新账户。关于说明，参见2.13节，“Perl安装注意事项”。

在所有东西被安装以后，你应当使用该命令初始化并且测试你的分发版：

```
shell> /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

如果这个命令提示mysqld daemon ended并马上失败，那么你能在数据目录中的文件“hostname'.err”找到一些信息。

关于**mysqld_safe**的详细信息参见5.1.3节，“mysqld_safe：MySQL服务器启动脚本”。

注释：MySQL授权表中的账户开始没有密码。启动服务器后，你应当使用2.9节，“安装后的设置和测试”中的说明来设置密码。

2.8.2. 典型配置选项

configure脚本给你一个很大的控制权来控制如何配置MySQL源码分发版。一般使用**configure**命令行的选项来进行。你也可以用某些环境变量作用于**configure**。请参见附录F：环境变量。要查看**configure**支持的选择列表，可以运行命令：

```
shell> ./configure --help
```

- 如果只编译MySQL客户端库和客户端程序而不是服务器，使用--without-server选项：
- shell> **./configure --without-server**

如果你没有C++编译器，则不能编译**mysql**(有一个客户端程序需要C++)。在这种情况下，你可以去掉configure里面测试C++编译器的代码，然后运行带有 --without-server选项的**./configure**。编译步骤步将仍然尝试构造**mysql**，但是你可以忽略任何关于“mysql.cc”的警告。（如果**make**停止，试试**make -k**告诉它即使出错继续剩下的构造步骤）。

- 如果你想要构建嵌入式MySQL库 (libmysqld.a),你应当使用--with-embedded-server选项。

- 如果你不想要位于“/usr/local/var”目录下面的日志(log)文件和数据库，使用类似于下列configure命令的一个：
- shell>**./configure--prefix=/usr/local/mysql**
- shell>**./configure--prefix=/usr/local –**
- **localstatedir=/usr/local/mysql/data**

第一个命令改变安装前缀以便将所有内容安装到“/usr/local/mysql”下面而非 默认的“/usr/local”。第二个命令保留默认安装前缀，但是覆盖了数据库目录默认目录(通常是“/usr/local/var”)并且把它改为/usr/local/mysql/data。编译完MySQL后，可以通过选项文件更改这些选项。请参见4.3.2节，“使用选项文件”。

- 如果你正在使用Unix并且你想要MySQL的套接字位于 默认位置以外的某个地方(通常在目录“/tmp”或“/var/run”，使用象这样的**configure**的命令：
- shell> **./configure\-- with-unix-socket-**
- **path=/usr/local/mysql/tmp/mysql.sock**

注意给出的文件名必须是一个绝对路径。你还可以在以后使用MySQL选项文件更改mysql.sock的位置。请参见A.4.5节，“如何保护或更改MySQL套接字文件/tmp/mysql.sock”。

- 如果你想编译静态链接程序(例如，制作二进制分发版、获得更快的速度或与解决与RedHat分发版的一些问题)，像这样运行**configure**：

- **shell>./configure --with-client-ldflags=-all-static**

- **--with-mysqld-ldflags=-all-static**

- 如果你正在使用gcc并且没有安装libg++或libstdc++，你可以告诉**configure**使用gcc作为C++编译器：
- shell> **CC=gcc CXX=gcc ./configure**

当你使用gcc作为C++编译器用时，它将不试图链接libg++或libstdc++。即使你安装了这些库，这样也很好，因为过去使用MySQL时，它们的部分版本会出现一些奇怪的问题。

下面列出了一些经常使用的编译器和环境变量设定值。

o **gcc** 2.7.2：

o
```
CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors"
```

o **egcs** 1.0.3a：

o
```
CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors \
-fno-exceptions -fno-rtti"
```

o **gcc** 2.95.2：

o
```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro \
-felide-constructors -fno-exceptions -fno-rtti"
```

o pgcc 2.90.29 或更新版：

o
```
CFLAGS="-O3 -mpentiumpro -mstack-align-double" CXX=gcc \
CXXFLAGS="-O3 -mpentiumpro -mstack-align-double \
```

```
o                    -felide-constructors -fno- exceptions -fno-rtti"
```

在大多数情况，在**configure**行使用前面列出的选项并加入下面的选项，可以得到合理的优化的MySQL二进制：

```
--prefix=/usr/local/mysql --enable-assembler \
--with-mysqld-ldflags=-all-static
```

换句话说，对于所有最近的**gcc**版，整个**configure**行应为：

```
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
```

我们在MySQL网址http://www.mysql.com/提供的二进制均已经编译过并进行了优化，对于大多数用户应当比较完善。请参见2.1.2.5节，“MySQL AB编译的MySQL二进制版本”。其中有一些配置设定值，可以选用来构建更快的二进制，但是只适用高级用户。请参见7.5.4节，“编译和链接怎样影响MySQL的速度”。

如果构造失败并且产生有关编译器或链接器的错误，不能创建共享库“libmysqlclient.so.#”(“#”是一个版本号), 你可以通过为**configure**指定--disable-shared选项来解决此问题。在这种情况下，**configure**将不构造一个共享libmysqlclient.so.#库。

- 默认地，MySQL使用latin1(cp1252)字符集。为了改变 默认设置，使用--with-charset选项：
- shell> **./configure -- with-charset=***CHARSET*

*CHARSET*可以是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或win1251ukr。请参见5.10.1节，“数据和排序用字符集”。

可以指定默认校对规则。MySQL默认使用latin1_swedish_ci校对规则。要想更改，使用--with-collation选项：

```
shell> ./configure --with-collation=COLLATION
```

要想更改字符集和校对规则，要同时使用--with-charset和--with-collation选项。 校对规则必须是字符集的合法校对规则。(使用SHOW COLLATION语句来确定每个字符集使用哪个校对规则)。

如果想在服务器和客户端之间变换字符，你应该看一下SET OPTION CHARACTER SET命令。请参见13.5.3节，“SET语法”。

警告：如果你在创建了任何数据库表之后改变字符集，你将必须对每个表运行**myisamchk -r -q --set-character-set=***charset*，否则索引可能不正确地排序。（如果安装了MySQL，创建一些表，然后重新配置MySQL使用一个不同的字符集并且重新安装它，这时就可能发生这种情况。）

通过**configure**选项--with-extra-charsets=*LIST*，你可以定义在服务器中再定义哪些字符集。*LIST* 指下面任何一项：

o 空格间隔的一系列字符集名

o complex -，以包括不能动态装载的所有字符集

o all –，以将所有字符集包括进二进制

- 为了配置MySQL带有调试代码，使用--with-debug选项：
- shell> **./configure --with-debug**

这导致包含一个安全的内存分配器，能发现一些错误和并提供关于正在发生的事情的输出。请参见E.1节，“调试MySQL服务器”。

· 如果客户端程序使用线程，还必须使用--enable-thread-safe-client配置选项来编译线程安全版的MySQL客户端库。将创建一个libmysqlclient_r库，可以用来连接应用程序线程。请参见25.2.15节，“如何生成线程式客户端”。

· 可以使用--with-big-tables选项构建支持大数据库表的MySQL。

该选项使变量用来将表行数保存到unsigned long long中而不是unsigned long。结果是允许表有大约1.844E+19 ($(2^{32})^2$)行而不是2^{32} (~4.295E+09) 行。在以前，要完成该功能必须手动将-DBIG_TABLES传递给编译器。

属于特定系统的选项可在本章特定系统的小节里找到。请参见2.12节，“具体操作系统相关的注意事项”

2.8.3. 从开发源码树安装

注意：如果你想帮助我们测试新代码，应当阅读本节。如果你只想在系统上安装并运行MySQL，应当使用标准发布分发版(二进制或源码分发版)。

要想获得我们的最新开发源码树，应使用说明：

1. 从http://www.bitmover.com/bk-client.shar下载BitKeeper免费客户端。

2. 在Unix上安装免费客户端：

```
3.          shell> sh bk-client.shar
4.          shell> cd bk_client-1.1
5.          shell> make all
6.          shell> PATH=$PWD:$PATH
```

在Windows上的安装方法：

- 从http://cygwin.com下载并安装Cygwin。
- 确保**gcc**和**make**已经安装到Cygwin。可以用**which gcc**和 **which make**命令进行测试。如果任何一个没有安装，运行 Cygwin安装包管理器，选择**gcc**、**make**并安装。
- 在Cygwin中，运行：

```
·                   shell> sh bk-client.shar
·                   shell> cd bk_client-1.1
```

然后编辑Makefile并将行s $(CC) $(CFLAGS) -o sfio -lz sfio.c更改为：

```
$(CC) $(CFLAGS) -o sfio sfio.c -lz
```

现在运行**make**命令并设置路径：

```
shell> make all
shell> PATH=$PWD:$PATH
```

7. 安装BitKeeper免费客户端后，首先进入工作目录，然后使用以下命令在本地复制MySQL 5.1：

```
8.          shell> sfioball -r+ bk://mysql.bkbits.net/mysql-5.1 mysql-5.1
```

一般情况，你不需要自己构建文档，因为我们已经在http://dev.mysql.com/doc/提供了许多格式。我们每天更新你可以下载的格式(HTML、PDF等)，因此你没有必要从MySQLdoc树中的DocBook XML基本格式来创建。如果你想要复制文档库，使用下面的命令：

```
shell> sfioball -r+ bk://mysql.bkbits.net/mysqldoc mysqldoc
```

在前面的例子中，在当前目录的MySQL-5.1/子目录中设置源码树。

如果你位于防火墙后面，只能初始化HTTP连接，还可以通过HTTP使用BitKeeper。

如果需要使用代理服务器，设置环境变量http_proxy指向代理：

```
shell> export http_proxy="http://your.proxy.server:8080/"
```

当复制库时用http://替换bk：//。例如：

```
shell> sfioball -r+ http://mysql.bkbits.net/mysql-5.1 mysql-5.1
```

下载源码树可能需要一段时间，取决于连接速度。请耐心等待。

9. 要想更新本地MySQL 5.1库复制，使用命令：

```
10.       shell> update bk://mysql.bkbits.net/mysql-5.1
```

11.你需要GNU **make**、**autoconf** 2.58 (或更新版)、**automake** 1.8、**libtool** 1.5和**m4**来运行后面的命令。尽管许多操作系统安装了**make**，很可能编译失败，出现许多奇怪的缺陷消息。因此，强烈建议你使用GNU **make**(有时名为**gmake**)。

幸运的是，大量的操作系统预装了GNU工具链或提供了可安装软件包。你可以随时从下面的地址下载：

- http://www.gnu.org/software/autoconf/
- http://www.gnu.org/software/automake/

- http://www.gnu.org/software/libtool/
- http://www.gnu.org/software/m4/
- http://www.gnu.org/software/make/

要想配置MySQL 5.1,你还需要GNU **bison** 1.75或更新版。旧版本的**bison**会报告错误:

```
sql_yacc.yy:#####: fatal error: maximum table size (32767) exceeded
```

注释:实际上并没有超出最大表空间;这是由旧版本的**bison**的缺陷造成的该问题。

下面的例子显示了配置源码树需要的典型命令。第一个cd命令将位置改为树的顶级目录;用相应目录名替换MySQL-5.1。

```
shell> cd mysql-5.1
shell> bk -r edit
shell> aclocal; autoheader
shell> libtoolize --automake --force
shell> automake --force --add-missing; autoconf
shell> (cd innobase; aclocal; autoheader; autoconf; automake)
shell> (cd bdb/dist; sh s_all)
shell> ./configure  # Add your favorite options here
shell> make
```

对于下面的命令,可以使用**BUILD/autorun.sh**作为快捷方式:

```
shell> aclocal; autoheader
shell> libtoolize --automake --force
shell> automake --force --add-missing; autoconf
shell> (cd innobase; aclocal; autoheader; autoconf; automake)
shell> (cd bdb/dist; sh s_all)
```

使用将目录改为innobase和bdb/dist目录的命令行来配置InnoDB和 Berkeley DB (BDB)储存引擎。如果你不需要InnoDB或BDB支持,你可以省略这些命令行。

如果在该阶段你遇到一些奇怪的错误,请确认你是否已经安装了 **libtool**。

标准配置脚本位于BUILD/子目录。你会发现使用BUILD/compile-pentium-debug脚本比使用前面的外壳命令要方便得多。要想在不同的架构上编译,可以修改脚本,即去掉Pentium相关标志。

12.构建完成后,运行**make install**。在产品机器上运行时应注意;该命令会覆盖你正使用的版本。如果你安装了另一个MySQL,我们建议你使用 与你的产品服务器上不同的--prefix、--with-tcp-port和--unix-socket-path选项值来运行**./configure**。

13.强制运行新安装和尝试会破坏新功能。应先运行**make test**。请参见27.1.2节,"MySQL测试套件"。

14.如果你已经进入**make**阶段而分发版不编译,请报告给我们的http://bugs.mysql.com/中的缺陷数据库。如果你已经安装了GNU工具的最新版本,如果视图处理配置文件时出错,请报告给我们。但是,如果你执行aclocal并且遇到command not found error或类似问题,不要报告给我们。而是应确保安装了所有必要的工具,并且PATH变量设置正确,保证外壳命令可以找到它们。

15.复制完库(sfioball)获得源码树后,你应当定期更新库(update)。

16.你可以检查树的更改史,浏览源码树中的BK/_ChangeLog文件并查看ChangeSet描述找到所有不同点。要想检查特殊的更改集,必须使用**sfioball**命令提取源码树的两次不同的修订,然后使用外部**diff**命令来比较。如果你看到有疑问的奇怪的不同之处或代码,一定要向MySQL internals邮件列表发送邮件。请参见1.7.1.1节,"The MySQL邮件列表"。并且,如果你有更好的思想,请向上述地址发送邮件。

17.BitKeeper免费客户端预装了源码。免费客户端可用的唯一文档是源码本身。

你还可以在线浏览更改集、注释和源码。要想浏览MySQL 5.1的相关信息,请到http://mysql.bkbits.net:8080/mysql-5.1。

2.8.4. 处理**MySQL**编译问题

所有MySQL程序在Solaris或Linux上使用**gcc**编译并且没有任何警告。在其它系统上,由于系统包含文件的差别可能会发生警告。对于使用MIT-pthreads时发生的警告。请参见2.8.5节,"MIT-pthreads注意事项"。其它问题,检查下面的表。

许多问题的解决方案涉及重新配置。如果你确实需要重新配置，注意下列事项：

- 如果**configure**在它已经被运行了以后运行，它可以使用先前收集的信息。这个信息存储在"config.cache"里面。当**configure**启动时，它寻找该文件而且如果它存在，假定信息仍然是正确的，读入它的内容。当你重新配置时，该假设无效。
- 每次运行**configure**的时候，必须运行**make**再重新编译。然而，你可能想要把先前构造的老的目标文件删除，因为它们使用不同的配置选项编译而成。

为了防止使用旧的配置信息或目标文件，重新运行**configure**前运行这些命令：

shell> **rm config.cache**

shell> **make clean**

另外，你可以运行**make distclean**。

下表描述了一些最常发生的编译MySQL的问题：

- 如果在编译"sql_yacc.cc"时，遇到如下错误，可能是存储器或交换空间溢出：

```
Internal compiler error: program cc1plus got fatal signal 11
Out of virtual memory
Virtual memory exhausted
```

该问题是gcc要求大量的内存编译带有嵌入函数(inline function)的"sql_yacc.cc"。试试以--with-low-memory选项运行**configure**：

shell> **./configure -- with-low-memory**

如果你正在使用**gcc**，该选项使得将-fno-inline加到编译行，如果你正使用其它的编译器，则加入-O0。你应该试一试--with-low-memory选项，即使你有特别多的存储器和交换空间，而你认为不可能运行得溢出。这个问题甚至会在很慷慨的硬件配置的系统上出现，通常用--with-low-memory选项修正它。

- 默认情况，**configure**选则**c++**作为编译器并用-lg++选项的GNU **c++**链接。如果你正在使用**gcc**，这个特性在配置期间导致如下问题：

```
configure: error: installation or configuration problem:
C++ compiler cannot create executables.
```

也可能在编译期间看到**g++**、libg++或libstdc++相关的问题。

这些问题的一个原因是你可能没有**g++**，或可能有**g++**但无libg++或libstdc++。看一下"config.log"文件。它应该包含c++编译器不能工作的准确原因！为了解决这些问题，可以使用gcc作为C++编译器。试试设置环境变量CXX为"gcc -O3"。例如：

shell> **CXX="gcc -O3" ./configure**

可以工作，因为**gcc**象**g++**一样编译C++源码，但 默认地它不链接libg++或libstdc++。

解决这些问题的其它方法当然是安装**g++**、libg++和libstdc++。然而，我们建议不要在MySQL中使用libg++或libstdc++，因为只会增加**mysqld**二进制空间而不会带来任何好处。部分版本的库在过去曾经给用户带来一些奇怪的问题。

- 如果你的编译以下面任何错误而失败，必须升级**make**版本到**GNUmake**：

```
making all in mit-pthreads
make: Fatal error in reader: Makefile, line 18:
Badly formed macro assignment
```

或：

```
make: file `Makefile' line 18: Must be a separator (:
```

或：

```
pthread.h: No such file or directory
```

已知Solaris和FreeBSD的make程序有问题。

已知GNU make 3.75能工作。

- 如果你想要定义C或C++编译器所使用的标志，把标志加到CFLAGS和CXXFLAGS环境变量中即可。也可以使用CC和CXX来指定编译器名字。例如：

```
shell> CC=gcc
shell> CFLAGS=-O3
shell> CXX=gcc
shell> CXXFLAGS=-O3
shell> export CC CFLAGS CXX CXXFLAGS
```

对于已经知道在不同系统上有用的标志定义列表，参见2.1.2.5节，“MySQL AB编译的MySQL二进制版本”。

- 如果你遇到象如下的一条错误消息，则需要升级**gcc**编译器：

```
client/libmysql.c:273: parse error before `__attribute__'
```

gcc2.8.1已知可以工作，但是我们推荐使用**gcc** 2.95.2或**egcs** 1.0.3a。

- 如果编译**mysqld**时显示例如下面的那些错误, **configure**没有正确地检测传到accept()、getsockname()或getpeername()最后参数的类型：

```
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);
```

为了修正它，编辑“config.h”文件(它由configure生成)。寻找这些行：

```
/* Define as the base type of the last arg to accept */
#define SOCKET_SIZE_TYPE XXX
```

- 更改XXX为size_t或int，取决于你的操作系统。（注意：每次运行**configure**都必须这样做，因为**configure**重新生成“config.h”）。
- “sql_yacc.cc”文件由“sql_yacc.yy”生成。通常构造过程不需要创造“sql_yacc.cc”，因为MySQL有一个已经生成的拷贝，然而，如果你确实需要再创建它，可能会碰到这个错误：

```
"sql_yacc.yy", line xxx fatal: default action causes potential...
```

这是一个**yacc**版本不完善的迹象。你可能需要安装**bison**（GNU的**yacc**)并使用它。

- 在Debian Linux 3.0上, 如果你编译的MySQL 5.1要支持Berkeley DB，需要安装gawk代替默认的mawk。
- 如果你需要调试mysqld或MySQL客户端，运行**configure**，使用--with-debug选项，然后重新编译并且将客户端程序链接到新的客户端库。请参见E.2节，“调试MySQL客户端”。
- 如果你在Linux(例如，SuSE Linux 8.1或Red Hat Linux 7.3)下遇到类似下面的编译错误：

```
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
```

默认情况，**configure**脚本试图使用**g++** GNU C++编译器来确定正确的参数。如果未安装**g++**，将会产生错误的结果。有两种方法可以解决该问题：

- 确保安装了GNU C++ **g++**。在某些Linux分发版上，需要安装的软件包叫做gpp；在其它分发版上名为**gcc-c++**。
- 使用**gcc**作为C++ 编译器，将CXX环境变量设置为**gcc**：

```
export CXX="gcc"
```

请注意之后需要再次运行**configure**。

2.8.5. MIT-pthreads注意事项

这节描述使用MIT-pthreads时所涉及的一些问题。

在Linux上，应该不使用MIT-pthreads而是安装LinuxThreads！参见2.12.1节，“Linux注意事项”。

如果你的系统不提供原生的线程支持，将需要使用MIT-pthreads包构造**MySQL**。这包括大多数FreeBSD系统、SunOS 4.x、Solaris 2.4和更早版本及其它，参见2.1.1节，“MySQL支持的操作系统”。

MIT-pthreads不是MySQL 5.1源码分发版的一部分。如果你需要该安装包，需要单独从http://www.mysql.com/Downloads/Contrib/pthreads-1_60_beta6-mysql.tar.gz下载。

下载后，将源文件提取到MySQL源码目录的顶级目录。将创建新的mit-pthreads子目录。

- 在大多数系统上，你能通过使用**configure**并用--with-mit-threads选项来强迫运行MIT-pthreads：
- shell> **./configure -- with-mit-threads**

当使用MIT-pthreads时，不支持在一个非源码目录构造，因为我们想要使我们对代码的改变减到最小。

· 决定是否使用MIT-pthreads的检查仅在处理服务器代码的配置过程期间发生。如果已经用--without-server配置了分发版并只构造客户端代码，客户端将不知道MIT-pthreads是否正在被使用并且是否使用 默认的Unix套接字连接。因为在某些平台上Unix套接字文件不能在MIT-pthreads下面工作，这意味着当你运行客户端程序时，你需要使用-h或--host。

· 当使用MIT-pthreads编译MySQL时，因为性能原因，系统锁定 默认为禁止使用。你可以用--external-locking选项告诉服务器使用系统锁定。只是在相同数据文件上运行两个MySQL服务器时采需要，因此不推荐。

- 有时pthreadbind()命令不能绑定一个套接字但没有任何错误消息(至少在Solaris上)，结果是所有到服务器的连接均失败。例如：
- shell> **mysqladmin version**
- mysqladmin: connect to server at '' failed;
- error: 'Can't connect to mysql server on localhost (146)'

解决它的方法是杀死**mysqld**服务器并且重启它。这只有当我们强迫服务器停止并马上进行重启时在发生。

- 使用MIT-pthreads，不能用SIGINT（break)中断sleep()系统调用。这只有在运行**mysqladmin --sleep**时才能注意到。在中断起作用并且进程停止之前必须等待sleep()终止。
- 当链接时，你可能会收到这样的警告消息(至少在Solaris上)；它们可以被忽视：
- 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

- 一些其它的警告也可被忽略：
- implicit declaration of function `int strtoll(...)'
- implicit declaration of function `int strtoul(...)'
- 我们还没有让readline在MIT-pthreads上工作。（这不需要，但是可能某些人会感兴趣。）

2.8.6. 在**Windows**下从源码安装**MySQL**

2.8.6.1. 使用VC++构建MySQL
2.8.6.2. 从最新的开发源码创建Windows源码安装包

下面的说明描述了如何在Windows下从5.1版源码构建MySQL二进制。说明用于从包含最新开发源码的标准源码分发版或从BitKeeper树来构建二进制。

注释：本文档中的说明严格限于在Windows下从最新源码分发版或从BitKeeper树来测试MySQL的用户。作为产品使用时，MySQL AB不建议使用你自己从源码构建的MySQL服务器。一般情况，最好使用MySQL AB预编译的对性能进行优化的适用Windows的MySQL二进制分发版。安装二进制分发版的说明参见2.3节，“在Windows上安装MySQL”。

要想在Windows中从源码构建MySQL，Windows系统中应有下面的编译器和资源：

· Visual Studio 2003编译器系统(VC++ 7.0)。

· 3到5GB的硬盘空间。

· Windows 2000或更高版本。

确切的系统需求见：http://msdn.microsoft.com/vstudio/productinfo/sysreqs/default.aspx

还需要适用Windows的MySQL源码分发版。有两种方法可以获得MySQL 5.1 源码分发版：

1. 获得MySQL AB打包的源码分发版软件包。预打包的源码分发版可从 http://dev.mysql.com/downloads/获得。

2. 你可以用最新BitKeeper开发源码树构建源码分发版。如果你计划构建，必须在Unix系统中创建安装软件包，并将它转移到Windows系统。(这样操作的理由是部分配置和构建步骤需要只能在Unix中工作的工具）。BitKeeper方法需要：

- 运行Unix或类似Unix的系统，例如Linux。
- 该系统上安装的BitKeeper 3.0。关于如何下载和安装BitKeeper的说明参见2.8.3节，“从开发源码树安装”。

如果你使用Windows源码分发版，你可以直接跳到2.8.6.1节，“使用VC++构建MySQL”。要想从BitKeeper树构建，继续2.8.6.2节，“从最新的开发源码创建Windows源码安装包”。

如果你发现某些组件不能按预料的工作，或者你有好的建议可以改进目前的在Windows中的构建过程，请向win32邮件列表发送消息。请参见1.7.1.1节，“The MySQL邮件列表”。

2.8.6.1. 使用VC++构建MySQL

注释：MySQL 4.1和以上版本的VC++工作区文件与Microsoft Visual Studio 2003版本兼容，MySQL AB人员已经在每个版本之前进行了测试。

按照以下步骤构建MySQL：

1. 创建工作目录(例如，C：\workdir)。
2. 使用**WinZip**或其它可以读取.zip文件的Windows工具将源码分发版解压缩到上述目录。
3. 启动Visual Studio。
4. 从File菜单，选择Open Workspace。
5. 打开工作目录中的MySQL.dsw工作区。
6. 从Build 菜单，选择Set Active Configuration菜单。
7. 点击窗口选择mysqld - Win32 Debug并点击OK。
8. 按**F7**开始构建调试服务器、库和客户端应用程序。
9. 按同样方法编译发布版本。
10. 程序和库的调试版本位于client_debug和lib_debug目录。程序和库的发布版本位于client_release和lib_release目录。请注意如果你想要构建调试版本和发布版本，可以从Build菜单选择Build All选项。
11. 测试服务器。使用前面的说明构建服务器时，默认MySQL基本目录和数据目录位于C:\mysql和C:\mysql\data。如果你想要使用源码树根目录和数据目录作为基本目录和数据目录来测试服务器，需要告诉服务器其路径名。可以在命令行用--basedir和--datadir选项来完成，或将相应选项放入选项文件(在Windows目录或C：\my.cnf中的my.ini文件)。如果想要使用的数据目录在其它地方，可以指定其路径名。
12. 根据你想要使用的服务器，从client_release或client_debug目录启动服务器。常用服务器启动说明见2.3节，“在Windows上安装MySQL”。如果想要使用其它基本目录或数据目录，需要相应地更改指令。
13. 当服务器根据你的配置独立运行或作为服务时，尝试从client_release或client_debug目录中的MySQL交互式命令行实用工具连接它。

如果构建的程序工作正常，停止服务器。按照以下步骤安装MySQL：

1. 创建安装MySQL的目录。例如，要安装到C：\mysql，使用命令：

```
2.          C:\> mkdir C:\mysql
3.          C:\> mkdir C:\mysql\bin
4.          C:\> mkdir C:\mysql\data
5.          C:\> mkdir C:\mysql\share
6.          C:\> mkdir C:\mysql\scripts
```

如果想要编译其它客户端程序并连接到MySQL，你还需要创建几个目录：

```
C: \> mkdir C: \mysql\include
C: \> mkdir C: \mysql\lib
C: \> mkdir C: \mysql\lib\debug
C: \> mkdir C: \mysql\lib\opt
```

如果想要对MySQL进行基准检查，创建目录：

```
C: \> mkdir C: \mysql\sql-bench
```

基准检查需要Perl支持。请参见2.13节，“Perl安装注意事项”。

7. 将下面的目录从workdir目录复制到C：\mysql目录：

```
C:\> cd \workdir
C:\workdir> copy client_release\*.exe C:\mysql\bin
C:\workdir> copy client_debug\mysqld.exe C:\mysql\bin\mysqld-debug.exe
C:\workdir> xcopy scripts\*.* C:\mysql\scripts /E
C:\workdir> xcopy share\*.* C:\mysql\share /E
```

如果想要编译其它客户端程序并连接到MySQL，还应当复制几个库和头文件：

```
C:\workdir> copy lib_debug\mysqlclient.lib C:\mysql\lib\debug
C:\workdir> copy lib_debug\libmysql.* C:\mysql\lib\debug
C:\workdir> copy lib_debug\zlib.* C:\mysql\lib\debug
C:\workdir> copy lib_release\mysqlclient.lib C:\mysql\lib\opt
C:\workdir> copy lib_release\libmysql.* C:\mysql\lib\opt
C:\workdir> copy lib_release\zlib.* C:\mysql\lib\opt
C:\workdir> copy include\*.h C:\mysql\include
C:\workdir> copy libmysql\libmysql.def C:\mysql\include
```

如果你想要对MySQL进行基准测试，你还应当：

```
C:\workdir> xcopy sql-bench\*.* C:\mysql\bench /E
```

按照二进制Windows分发版相同的方法设置并启动服务器。请参见2.3节，“在Windows上安装MySQL”。

2.8.6.2. 从最新的开发源码创建Windows源码安装包

要想从当前的BitKeeper源码树创建Windows源码安装包，使用下面的说明。请注意必须在运行Unix或类Unix操作系统的系统上执行该过程。例如，已知该过程在Linux上工作得很好。

1. 复制MySQL 5.1的BitKeeper源码树。关于如何复制源码树的详细信息，相关说明参见2.8.3节，“从开发源码树安装”。
2. 配置和构建分发版，得到可工作的服务器二进制。实现的一个方法是在源码树顶级目录运行下面的命令：

```
shell> ./BUILD/compile-pentium-max
```

4. 确保构建过程成功完成后，从源码树顶级目录运行下面的实用工具脚本：

```
shell> ./scripts/make_win_src_distribution
```

该脚本创建在Windows 系统中使用的源码安装包。可以根据你的需求为脚本提供不同的选项。它接受下面的选项：

- --help

显示帮助消息。

- --debug

打印关于脚本操作的信息，不创建包。

- --tmp

指定临时位置。

- --suffix

安装软件包后缀名。

- --dirname

复制文件的目录名(媒介之间)。

- --silent

不打印正处理文件的详细列表。

- --tar

创建tar.gz安装软件包而不是.zip安装软件包。

默认情况，**make_win_src_distribution**创建Zip-格式的文件，文件名为mysql-*VERSION*-win-src.zip，其中*VERSION*代表MySQL源码树的版本。

6. 将创建的Windows源码安装包复制或上载到Windows机器上。要想编译，使用2.8.6.1节，“使用VC++构建MySQL”中的说明。

2.8.7. 在Windows下编译MySQL客户端

在源码文件中，应当在MySQL.h之前包括my_global.h：

```
#include <my_global.h>
#include <mysql.h>
```

my_global.h包括你在Windows中编译程序所需要的Windows兼容文件(例如 windows.h)。

可以将代码连接到动态libmysql.lib库，它只是一个包装器以便根据需要装载libmysql.dll，或连接静态mysqlclient.lib库。

MySQL客户端库编译为线程库，因此你应当将代码编译为多线程。

2.9. 安装后的设置和测试

2.9.1. Windows下安装后的过程
2.9.2. Unix下安装后的过程
2.9.3. 使初始MySQL账户安全

安装完MySQL后，有一些问题你应当处理。例如，在Unix中，你应当初始化数据目录并创建MySQL授权表。对于所有平台，一个重要安全问题是 授权表中的初始账户没有密码。你应当指定密码以防止未授权访问MySQL服务器。你可以创建时区表以识别命名的时区。(目前，只在Unix上装了这些表。不久将在Windows中解决该问题）。

下面章节包括适用Windows系统和Unix系统的安装后的过程。另一节，2.9.2.3节，“启动MySQL服务器以及其故障诊断和排除”，适用所有平台；它描述了启动服务器时遇到问题时如何解决。2.9.3节，“使初始MySQL账户安全”也适用所有平台。你应当按照说明确保你已经正确地为你的MySQL账户指定密码来保护其安全。

如果你准备创建其它用户账户，你可以在5.7节，“MySQL访问权限系统”和5.8节，“MySQL用户账户管理”找到关于MySQL访问控制系统和账户管理的信息。

2.9.1. Windows下安装后的过程

在Windows中，不需要创建数据目录和授权表。MySQL Windows分发版包括在数据目录下的MySQL数据库中的一套预初始化的账户的 授权表。不要运行Unix中使用的**mysql_install_db**脚本。但是，如果你没有使用Windows Installation Wizard（安装帮助）来安装MySQL，应当为账户指定密码。请参见2.3.4.1节，“前言”。在2.9.3节，“使初始MySQL账户安全”。

设置密码前，你可能想要运行一些客户端程序来确保你能够连接服务器，并且操作正确。确保服务器在运行(参见2.3.10节，“首次启动服务器”)，然后发出下面的命令来验证你可以从服务器获取信息。输出应当类似于：

```
C:\> C:\mysql\bin\mysqlshow
+-----------+
| Databases |
+-----------+
| mysql     |
| test      |
+-----------+
```

```
C:\> C:\mysql\bin\mysqlshow mysql
Database: mysql
+--------------+
|    Tables    |
+--------------+
| columns_priv |
| db           |
| func         |
| host         |
| tables_priv  |
| user         |
+--------------+

C:\> C:\mysql\bin\mysql -e "SELECT Host,Db,User FROM db" mysql
+------+-------+------+
| host | db    | user |
+------+-------+------+
| %    | test% |      |
+------+-------+------+
```

如果你正运行支持服务的Windows版本，你想让MySQL服务器在Windows启动时自动运行，参见2.3.12节，“以Windows服务方式启动MySQL”。

2.9.2. Unix下安装后的过程

2.9.2.1. 与运行mysql_install_db有关的问题
2.9.2.2. 自动启动和停止MySQL
2.9.2.3. 启动MySQL服务器以及其故障诊断和排除

在Unix上安装MySQL后，需要初始化 授权表、启动服务器，并确保服务器工作正常。你还要让服务器随系统的启动和停止自动启动和停止。应当为授权表中的账户指定密码。

在Unix中，由**mysql_install_db**设置 授权表。在某些安装中，该程序自动运行：

- 如果你使用RPM分发版在Linux上安装MySQL，服务器RPM运行**mysql_install_db**。
- 如果你使用PKG分发版在Mac OS X上安装MySQL，安装器运行**mysql_install_db**。

否则，你需要自己运行**mysql_install_db**。

下面的过程描述了如何初始化授权表 (如果还没有初始化)并启动服务器。还推荐了一些你可以用来测试服务器是否可以访问以及是否工作正确的命令。关于自动启动和停止服务器的信息，参见2.9.2.2节，“自动启动和停止MySQL”。

你完成过程并让服务器运行后，你应当为**mysql_install_db**创建的账户指定密码。2.9.3节，“使初始MySQL账户安全”中列出了相关说明。

在本节的例子中，服务器用MySQL登录账户的用户ID运行。假定存在这样的账户。如果不存在，要么创建账户，或用其它已有的用来运行服务器的登录账户来替代。

1. 进入MySQL的顶级安装目录，此处为*BASEDIR*：

2. `shell> cd BASEDIR`

*BASEDIR*可能为/usr/local/mysql 或/usr/local。以下步骤假定你位于该目录。

3. 根据需要，运行**mysql_install_db** 程序设置含有确定如何让用户连接服务器的权限的初始MySQL授权表。如果你使用的分发版类型不能运行程序，你需要执行该步骤。

典型，只有首次安装MySQL时，才需要运行**mysql_install_db**，因此如果你升级已有的安装你可以跳过该步骤，但**mysql_install_db**不会覆盖已有的 授权表，因此可以在任何环境下安全运行。

要想初始化授权表，根据**mysql_install_db**是位于bin还是scripts目录下，使用下面的一个命令：

```
shell> bin/mysql_install_db --user=mysql
```

```
shell> scripts/mysql_install_db --user=mysql
```

mysql_install_db脚本创建数据目录、拥有所有数据库权限的mysql数据库和可以用来测试MySQL的test数据库。脚本为root账户和匿名用户账户创建 授权表条目。账户一开始没有密码。2.9.3节，“使初始MySQL账户安全”中描述了初始权限。简单说，这些权限允许MySQL root用户执行任何操作，允许任何人使用test名创建或使用数据库或用test_启动。

一定要确保由mysql登录账户拥有数据库目录和文件，以便在以后运行时 服务器具有读、写访问权限。为此，如果你用root用户运行**mysql_install_db**，应当使用--user选项。否则，当以mysql登录时，应当执行脚本，你可以省略命令中的--user选项。

mysql_install_db在mysql数据库中创建几个表，包括user、db、host、 tables_priv、columns_priv和func以及其它。5.7节，“MySQL访问权限系统”中有完整的列表和描述。

如果你不想要test数据库，启动服务器后，可以用**mysqladmin -u root drop test**卸掉。

如果有**mysql_install_db**相关问题，参见2.9.2.1节，“与运行mysql_install_**db**有关的问题”。

有一些选择运行**mysql_install_db**脚本,正如MySQL分发版中所提供的：

- 如果想让初始权限与标准默认值不同，你可以在运行前修改**mysql_install_db**。然而，最好是在设置 授权表后使用GRANT和REVOKE来更改权限。换句话说，你可以运行**mysql_install_db**，然后通过MySQL root用户使用mysql -u rootMySQL来连接服务器，以便发出GRANT和REVOKE命令。

如果你想要在几个机器上用相同的权限安装MySQL，可以将GRANT和REVOKE语句放入一个文件中，以脚本方式执行文件，运行**mysql_install_db**之后使用mysql。例如：

```
shell> bin/ mysql_install_db --user=mysql
shell> bin/mysql -u root < your_script_file
```

这样，你就可以避免在每台机器上手动发出命令。

- 完全可以重新创建授权表。如果你刚刚知道如何使用GRANT和REVOKE，并且运行**mysql_install_db** 之后进行了许多修改，想要关闭表重新启动，你可能想要这样操作。

要想重新创建授权表，从含有MySQL数据库的目录中移走所有 .frm、.MYI和 .MYD文件。(这是数据目录下面的mysql目录，当你运行**mysqld --help** 时，列为datadir值)。然后再次运行**mysql_install_db**脚本。

- 可以使用--skip-grant-tables选项手动启动**mysqld**，并使用**mysql**自己增加权限信息：

```
shell> bin/mysqld_safe --user=mysql --skip-grant-tables &
shell> bin/mysql mysql
```

手动从**mysql**执行**mysql_install_db**中的SQL命令。确保后面钥运行**mysqladmin flush-privileges**或**mysqladmin reload**，让服务器重载授权表。

请注意不使用**mysql_install_db**，不仅需要手动安装授权表，还需要先创建。

4. 启动MySQL服务器：

```
5.                    shell> bin/mysqld_safe --user=mysql &
```

一定要让MySQL服务器使用非权限(non-root) 登录账户运行。为此，如果你以系统root运行mysql_safe，应当使用--user选项。否则，你应当用mysql登录到系统来执行脚本，这样可以省略命令中的--user选项。

A.3.2节，“如何以普通用户身份运行MySQL”中给出了非特权用户运行MySQL的说明。

如果在执行该步前你忘记了创建授权表，当你启动服务器时，在错误日志文件中将出现下面的消息：

```
mysqld: Can't find file: 'host.frm'
```

如果启动服务器时遇到其它问题，查阅2.9.2.3节，“启动MySQL服务器以及其故障诊断和排除”。

6. 使用**mysqladmin**验证服务器在运行中。以下命令提供了简单的测试，可检查服务器是否已经启动并能响应连接：

```
7.            shell> bin/mysqladmin version
8.            shell> bin/mysqladmin variables
```

mysqladmin version的输出根据平台和MySQL版本的不同而稍有变化，但是应当类似于：

```
shell> bin/mysqladmin version
mysqladmin  Ver 8.41 Distrib 5.1.2-alpha, for pc-linux-gnu on i686
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          5.1.2-alpha-Max
Protocol version        10
Connection              Localhost via UNIX socket
UNIX socket             /var/lib/mysql/mysql.sock
Uptime:                 14 days 5 hours 5 min 21 sec

Threads: 1  Questions: 366  Slow queries: 0
Opens: 0  Flush tables: 1  Open tables: 19
Queries per second avg: 0.000
```

要想看还可以怎样使用**mysqladmin**，用-- help选项调用它。

9. 验证可以关闭服务器：

```
10.        shell> bin/mysqladmin -u root shutdown
```

11.验证是否可以重启服务器。可以直接使用**mysqld_safe**或调用**mysqld**。例如：

```
12.        shell> bin/mysqld_safe --user=mysql --log &
```

如果mysqld_safe失败，参见2.9.2.3节，“启动MySQL服务器以及其故障诊断和排除”。

13.进行一些简单的测试，验证你可以从服务器查询信息。输出应当类似于：

```
14.        shell> bin/mysqlshow
15.        +-----------+
16.        | Databases |
17.        +-----------+
18.        | mysql     |
19.        | test      |
20.        +-----------+
21.
22.        shell> bin/mysqlshow mysql
23.        Database: mysql
24.        +---------------------------+
25.        |          Tables           |
26.        +---------------------------+
27.        | columns_priv              |
28.        | db                        |
29.        | func                      |
30.        | help_category             |
31.        | help_keyword              |
32.        | help_relation             |
33.        | help_topic                |
34.        | host                      |
35.        | proc                      |
36.        | procs_priv                |
```

```
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+

shell> bin/mysql -e "SELECT Host,Db,User FROM db" mysql
+------+--------+------+
| host | db     | user |
+------+--------+------+
| %    | test   |      |
| %    | test_% |      |
+------+--------+------+
```

53. 在sql-bench目录(在MySQL安装目录下)中有一个基准套件，可以用来比较MySQL在不同平台上的执行情况。基准套件在Perl中编写。它使用Perl DBI模块来为各种数据库提供一个与数据库无关的接口，并且还需要其它Perl模块来运行基准套件。必须安装以下模块：

```
DBI
DBD::mysql
Data::Dumper
Data::ShowTable
```

可以从CPAN(http://www.cpan.org/)获得这些模块。请参见2.13.1节，“在Unix中安装Perl”。

sql-bench/Results目录包含了在不同数据库和平台上的各种运行结果。要想进行测试，执行命令：

```
shell> cd sql-bench
shell> perl run-all-tests
```

如果没有sql-bench目录，你可能使用RPM文件安装了MySQL，没有使用源码RPM。(源码RPM包括sql-bench benchmark目录）。 此时，必须先安装基准套件后才能使用。有一个单独的mysql-bench-VERSION-i386.rpm基准RPM文件，其中包含了基准代码和数据。

如果你有源码分发版，其tests子目录中也有一些测试可供运行。例如，要运行auto_increment.tst，从源码分发版的顶级目录执行该命令：

```
shell> mysql -vvf test < ./tests/auto_increment.tst
```

期望的结果被显示在“ ./tests/auto_increment.res”文件中。

58. 至此，你应当可以运行服务器了。然而，初始MySQL账户均没有密码，因此应当使用2.9.3节，“使初始MySQL账户安全”中的说明来指定密码。

MySQL 5.1安装过程在MySQL数据库中创建时区表。但是，必须手动装载表。相关说明参见5.10.8节，“MySQL服务器时区支持”。

2.9.2.1. 与运行**mysql_install_db**有关的问题

mysql_install_db 脚本的目的是生成新的MySQL授权表。它不覆盖已有的MySQL授权表，并且它不影响任何其它数据。

如果你想要重新创建授权表，首先停止**mysqld**服务器（如果它正运行）。然后重新命名数据目录下的MySQL目录并保存，然后运行**mysql_install_db**。例如：

```
shell> mv mysql-data-directory/mysql mysql-data-directory/mysql-old
shell> mysql_install_db --user=mysql
```

本节列出了运行**mysql_install_db**时你可能遇到的问题：

- **mysql_install_db fails to install the grant tables**

你会发现**mysql_install_db**不能安装 授权表，显示下面的消息后终止：

```
Starting mysqld daemon with databases from XXXXXX
mysqld ended
```

在这种情况下，你应该很小心地检验日志文件！日志文件应该位于目录“xxxxxx”，用错误消息命名，并且应该指出为什么**mysqld**没启动。如果你不理解发生的事情，邮寄一份错误报告，包含日志文件！参见1.7.1.3节，“如何通报缺陷和问题”。

- 已经有一个**amysqld**进程在运行

表示服务器在运行，这种情况下可能已经创建了授权表。如果如此，则不再需要运行**mysql_install_db**，因为只需要运行一次(当你首次安装MySQL时)。

- 当一个服务器正运行时，安装第二个服务器不工作

这只有在当你已经有已存在的**MySQL**安装但是想要把新安装放在一个不同的地方时才会发生。例如，你可能已经有了一个产品安装，但为了测试想要同时运行2个安装。通常当你试着运行第二个服务器时，发生的问题是它试图和第一个使用同样的套接字和端口。在这种情况下，你将遇到错误消息：

```
Can't start server: Bind on TCP/IP port:
Address already in use
Can't start server: Bind on unix socket...
```

关于设置多个服务器的说明，参见5.12节，“在同一台机器上运行多个MySQL服务器”。

- 你没有“ **/tmp** ”的写权限

如果你没有写权限在默认位置(在“/tmp”里)创建一个Unix套接字文件，或没有在“/tmp”创建临时文件的许可，在运行**mysql_install_db**或**mysqld**服务器时，你将遇到一个错误。

你可以在开始**mysql_install_db**或**mysqld**之前执行以下命令指定一个不同的Unix套接字文件位置和临时目录：

```
shell> TMPDIR=/some_tmp_dir/
shell> MYSQL_UNIX_PORT=/some_tmp_dir/mysql.sock
shell> export TMPDIR MYSQL_UNIX_PORT
```

*some_tmp_dir*应该是你有写许可的某个目录的全路径。

然后，你应当能够用这些命令运行**mysql_install_db**并启动服务器：

```
shell> bin/mysql_install_db --user=mysql
shell> bin/mysqld_safe --user=mysql &
```

如果**mysql_install_db**位于scripts目录下，首先修改命令scripts/mysql_install_db。

参见A.4.5节，“如何保护或更改MySQL套接字文件/tmp/mysql.sock”。请参见附录F：环境变量。

2.9.2.2. 自动启动和停止MySQL

通常你可以用以下方法启动**mysqld**服务器：

- 直接调用**mysqld**。该方法适合任何平台。

- 作为Windows服务运行MySQL服务器。可以在支持服务的Windows版本(例如 NT、2000、XP和2003)上实现。可以将服务设置为在Windows 启动时自动启动服务器，或根据需要启动的手动服务。相关说明参见2.3.12节，“以Windows服务方式启动MySQL”。

- 调用**mysqld_safe**，可以为**mysqld**确定正确的选项然后使用这些选项来运行。该脚本适用于基于BSD Unix的系统。请参见5.1.3节，“mysqld_safe：MySQL服务器启动脚本”。

- 调用**mysql.server**。该脚本主要用于使用系统V-style运行目录的系统的启动和关闭，它通常安装到mysql下。**mysql.server**脚本调用**mysqld_safe**来启动服务器。请参见5.1.4节，“mysql.server：MySQL服务器启动脚本”。

- 你可以在Mac OS X上安装一个单独的MySQL Startup Item安装包来使系统启动时自动启动MySQL。Startup Item调用**mysql.server**来启动服务器。详细介绍参见 2.5节，“在Mac OS X上安装MySQL”。

mysql.server和**mysqld_safe**脚本和Mac OS X Startup Item可以用来手动启动服务器，或自动启动系统。**mysql.server**和Startup Item还可以用来停止服务器。

mysql.server脚本可以被用来启动或停止服务器，通过用start或stop参数调用它：

```
shell> mysql.server start
shell> mysql.server stop
```

在**mysql.server**启动服务器之前，它把目录改变到**MySQL**安装目录，然后调用**safe_mysqld**。如果你想要作为一些特定的用户运行服务器，在/etc/my.cnf选项文件的[**mysqld**]组增加相应user选项，如本节后面所示。(如果你有在一个非标准的地点安装的二进制分发版，你可能需要编辑**mysql.server**。修改它，运行**safe_mysqld**前，cd到正确的目录。注意如果你修改**mysql.server**，那么某个时候升级**MySQL**时，你的修改版本将被覆盖，因此你应该做一个你可重新安装的编辑过的版本的拷贝）。

mysql.server stop通过向服务器发出一个信号停止它。你可手动执行**mysqladmin shutdown**关闭服务器。

要想在服务器上自动启动和停止MySQL，应在"/etc/rc * 文件中适当的地方增加启动、停止命令。

如果你使用Linux服务器RPM安装软件包(MySQL-server-VERSION.rpm)，**mysql.server**脚本安装在/etc/init.d目录中，名为MySQL。你不需要手动安装它。关于Linux RPM软件包的详细信息参见2.4节，"在Linux下安装MySQL"。

一些供应商提供的RPM软件包安装的启动脚本名字不同，例如**mysqld**。

如果从不自动安装**mysql.server**的源码分发版或二进制分发版格式来安装MySQL，可以手动安装它。可以在MySQL安装目录下或MySQL 源码树的support-files目录中找到脚本。

要想手动安装**mysql.server**，用名称**mysql**将它复制到/etc/init.d目录，然后将它变为可执行文件。只需要将位置更改为**mysql.server**is所在并执行这些命令的相应目录：

```
shell> cp mysql.server /etc/init.d/mysql
shell> chmod +x /etc/init.d/mysql
```

旧的Red Hat系统使用/etc/rc.d/init.d目录，不使用/etc/init.d。相应地调节前面的命令。也可以首先创建指向/etc/rc.d/init.d的符号连接/etc/init.d：

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

安装脚本后，用来激活它以便在系统启动时运行所需要的命令取决于你的操作系统。在Linux中，你可以使用chkconfig：

```
shell> chkconfig --addMySQL
```

在一些Linux系统中，还需要下面的命令来完全激活MySQL脚本：

```
shell> chkconfig --level 345MySQL on
```

在FreeBSD中，启动脚本通常应当位于/usr/local/etc/rc.d/。手册的rc(8)页内说明只有该目录脚本的基本名匹配*.sh shell文件名模式，脚本才会执行。目录内的其它文件或目录将被忽略掉。换句话说，在FreeBSD中，应当将mysql.server脚本安装为 /usr/local/etc/rc.d/mysql.server.sh以便自动启动。

前面设置的另一种情况是，一些操作系统启动时也使用/etc/rc.local或/etc/init.d/boot.local 来启动其它服务。要想使用该方法启动MySQL，你可以在相应启动文件后面追加一条命令：

```
/bin/sh -c 'cd /usr/local/mysql; ./bin/mysqld_safe --user=mysql &'
```

对于其它系统，查阅操作系统的文档来查看安装启动脚本的方法。

你也可以在一个全局"/etc/my.cnf"文件中增加**mysql.server**的选项。一个典型的"/etc/my.cnf"文件可能看起来像这样：

```
[mysqld]
datadir=/usr/local/mysql/var
socket=/var/tmp/mysql.sock
port=3306
user=mysql

[mysql.server]
basedir=/usr/local/mysql
```

mysql.server脚本使用下列变量：basedir、datadir和pid-file。定义后，必须将它们放到选项文件中，不能放到命令行。**mysql.server**只识别start和stop命令行参数。

下面的表显示了服务器和每个启动脚本从选项文件读取哪个选项组：

脚本	选项组

mysqld	[mysqld], [server], [mysqld-major-version]
mysql.server	[mysqld], [mysql.server], [server]
mysqld_safe	[mysqld], [server], [mysqld_safe]

[mysqld-major-version]意味着名为[mysqld-5.0]的组，[mysqld-5.1]用于版本为5.0.x、5.1.x等的服务器。该特性可以用来指定只被给定发布系列的服务器读取的选项。

为了向后兼容，**mysql.server**还读取[mysql_server]组，**mysqld_safe**还读取[safe_mysqld]组。然而，当使用MySQL 5.1时，你应当更新选项文件，使用[mysql.server]和[mysqld_safe]组。

参见4.3.2节，“使用选项文件”。

2.9.2.3. 启动MySQL服务器以及其故障诊断和排除

如果启动服务器时有问题，可以尝试：

- 指定你使用的储存引擎需要的任何特殊选项。
- 确保服务器知道从哪里找到数据目录。
- 确保服务器可以使用数据目录。数据目录和内容的所有权和允许必须设置成服务器可以访问和修改它们。
- 检查错误日志查看服务器为何不启动。
- 验证服务器想要使用的网络接口可用。

一些储存引擎有一些选项可以控制其行为。你可以创建一个my.cnf文件并为计划使用的引擎设置启动选项。如果你将要使用支持事务处理表 (InnoDB，BDB)的储存引擎，应确保启动服务器之前按照你的期望对它们进行了配置：

- 如果你正使用InnoDB表，参阅InnoDB-specific启动选项。如果你未指定选项，InnoDB使用默认值作为配置选项。请参见15.2.3节，“InnoDB配置”。
- 如果你正使用BDB (Berkeley DB)表，你应当熟悉不同的BDB-specific启动选项。请参见15.5.3节，“BDB启动选项”。

当**mysqld**服务器启动时，它进入数据目录。在这里它可以找到数据库并写入日志文件。在Unix中，服务器还在数据目录中写pid(过程 ID)文件。

当编译服务器时确定数据目录。这是服务器默认寻找数据目录的位置。如果数据目录位于系统中的其它位置，服务器不能正确工作。用--verbose和--help选项调用**mysqld**你可以找出默认路径设定值。

如果默认值与你的系统中的MySQL安装布局不匹配，你可以在命令行中为**mysqld** 或**mysqld_safe**指定选项来覆盖它们。你还可以在选项文件中列出选项。

要想明显指定数据目录的位置，使用--datadir选项。一般情况下，你可以告诉**mysqld**基本目录的位置，MySQL安装在该目录下，并且它在该目录中寻找数据目录。你可以使用--basedir选项来实现。

要想检查指定路径选项的结果，用--verbose和--help选项调用**mysqld**。例如，如果你进入**mysqld**的安装目录，然后运行下面的命令，它显示启动服务器的结果，基本目录为/usr/local：

```
shell> ./mysqld --basedir=/usr/local --verbose --help
```

你可以指定其它选项，例如--datadir，但是请注意--verbose和--help必须为最后的选项。

一旦你确定了你想要的路径设定值，用--verbose和-- help启动服务器。

如果**mysqld**正在运行，执行下列命令你可以找出它所使用的路径设定值：

```
shell> mysqladmin variables
```

或：

```
shell> mysqladmin -h host_name variables
```

*host_name*是MySQL服务器主机的名称。

如果启动**mysqld**时遇到Errcode 13(意味着Permission denied)，这意味着数据目录或其内容的访问权限不允许服务器访问。此时，你需要更改所调用文件和目录的权限，使服务器有权使用它们。你还可以用root启动服务器，但是这样会造成安全问题，应当避免。

在Unix中，进入数据目录，检查数据目录和其内容的所有权，确保服务器可以访问。例如，如果数据目录是/usr/local/mysql/var，使用命令：

```
shell> ls -la /usr/local/mysql/var
```

如果数据目录或其文件或子目录不属于你运行服务器使用的账户，将所有权改为该账户：

```
shell> chown -R mysql /usr/local/mysql/var
shell> chgrp -R mysql /usr/local/mysql/var
```

如果服务器不能正确启动，检查错误日志文件，看看是否可以找到原因。日志文件位于数据目录(在Windows中一般为C:\Program Files\MySQL\MySQL Server 5.1\data，Unix二进制分发版为/usr/local/mysql/data，Unix源码分发版为/usr/local/var)。查找数据目录中的*host_name*.err和*host_name*.log文件，其中*host_name*是你的服务器主机名。然后检查文件的最后几行。在Unix中，可以使用tail来显示：

```
shell> tail host_name.err
shell> tail host_name.log
```

错误日志包含指示服务器不能启动的信息。例如，你可以看见日志中：

```
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
```

这意味着你没有用--bdb-no-recover选项启动**mysqld**，Berkeley DB恢复数据库时发现其日志文件有一些问题。要想继续，你应当将旧的Berkeley DB 日志文件从数据库目录移到其它地方，以后你可以在那儿检查它们。BDB日志文件以log.0000000001开头，按顺序命名。

如果你运行支持BDB 表的**mysqld**,**mysqld**启动时内核崩溃，该可能是由于BDB 恢复日志的问题。此时，你可以尝试用--bdb-no-recover启动**mysqld**。如果有帮助，你应当从数据目录移走所有BDB日志文件并尝试不用--bdb-no-recover选项重新启动**mysqld**。

如果出现下面的错误，说明其它程序(也许是另一个**mysqld**服务器)正使用**mysqld**正试图使用的TCP/IP端口或Unix 套接字文件：

```
Can't start server: Bind on TCP/IP port: Address already in use
Can't start server: Bind 在Unix中 socket...
```

使用**ps**来确定是否另有一个**mysqld**服务器正在运行。如果如此，关闭服务器重新启动**mysqld**。(如果另一个服务器正运行，你的确想要运行多个服务器，你可以在5.12节，“在同一台机器上运行多个MySQL服务器”中发现相关信息)。

如果没有其它服务器在运行，尝试执行命令 telnet your-host-name tcp-ip-port-number。(默认MySQL端口号是3306）。然后按两次Enter（回车）键。如果出现telnet: Unable to connect to remote host: Connection refused错误消息，其它程序正使用**mysqld**试图使用的 TCP/IP端口。你需要跟踪这是哪个程序并禁用它，或让**mysqld**用--port选项帧听其它端口。此时，当通过TCP/IP协议连接服务器时，你还需要为客户端程序指定端口号。

端口不能访问的另一个原因可能是防火墙正运行，阻挡了与它的连接。如果如此，修改防火墙设置允许对该端口的访问。

如果服务器已经启动但是你不能与它连接，你应当确保在/etc/hosts中有下面所示条目：

```
127.0.0.1       localhost
```

该问题只发生在没有工作线程库，并且MySQL必须配置为使用MIT-pthreads的系统。

如果你不能启动**mysqld**，你可以使用--debug选项尝试编写一个跟踪文件来找到问题。请参见E.1.2节，“创建跟踪文件”。

关于在Windows安装中排错的详细信息，参见2.3.14节，“在Windows环境下对MySQL安装的故障诊断与排除”。

2.9.3. 使初始**MySQL**账户安全

MySQL安装过程包括设置含有授权表的MySQL数据库：

- Windows分发版包含预初始化的授权表，可以自动安装。
- 在Unix中，用**mysql_install_db**程序来安装 授权表。可以通过一些安装方法来运行该程序。否则你需要手动执行。详细信息参见2.9.2节，“Unix下安装后的过程”。

授权表定义了初始MySQL用户账户和访问权限。按照以下步骤对这些账户进行设置：

- 用用户root 创建两个账户。这些账户为超用户账户，可以执行任何操作。初始root账户的密码为空，因此任何人可以用root账户不用任何密码来连接MySQL服务器，并具有所有权限。
  - 在Windows中，一个root账户用来从本机连接MySQL服务器，另一个允许从任何主机连接。
  - 在Unix中，两个root账户均用于从本机连接。必须从本机进行连接，一个账户主机名应指定为localhost，另一个账户为实际的主机名或IP号。
- 创建了两个匿名用户账户，每个账户的用户名均为空。匿名账户没有密码，因此任何人可以使用匿名账户来连接MySQL服务器。

- 在Windows中，一个匿名账户用来从本机进行连接。它具有所有权限，同root 账户一样。另一个可以从任何主机上连接，具有test数据库或其它以test开始的数据库的所有权限。
- 在Unix中，两个匿名账户均用于从本机连接。必须从本机进行连接，一个账户主机名应指定为localhost，另一个账户为实际的主机名或IP号。两个账户具有test数据库或其它以test开始的数据库的所有权限。

如前面所述，所有初始账户均没有密码。这意味着在你执行下述操作前，MySQL安装未受保护：

- 如果你想要防止客户端不使用密码用匿名用户来连接，你应当为匿名账户指定密码或删掉匿名账户。
- 你应当为MySQL root账户指定密码。

下面的说明描述了如何为初始MySQL账户设置密码，先为匿名账户设置然后为root账户设置。在例子中用实际密码替换"*newpwd*"。说明还包括如果你不想用匿名账户访问，如何删掉匿名账户。

你可能想要在以后设置密码，因此不需要在进一步的设置或测试中指定密码。但是，一定要在实际生产作业、使用安装前设置好密码。

要想为匿名账户指定密码，可以使用SET PASSWORD或UPDATE。在两种情况中，一定要使用PASSWORD()函数为密码加密。

在Windows中使用PASSWORD的方法：

```
shell> mysql -u root
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR ''@'%' = PASSWORD('newpwd');
```

在Unix中使用PASSWORD的方法：

```
shell> mysql -u root
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR ''@'host_name' = PASSWORD('newpwd');
```

用服务器主机名替换第二个SET PASSWORD语句中的*host_name*。这是指定的user表中的root non-localhost记录的Host列名。如果你不知道是哪个主机名，在SET PASSWORD之前执行下面的语句：

```
mysql> SELECT Host, User FROM mysql.user;
```

查找在User列有root和在Host列没有localhost的记录。然后在第二个SET PASSWORD语句中使用该Host值。

为匿名账户指定密码的另一种方法是使用UPDATE直接修改用户表。用root连接服务器，运行UPDATE语句为相应user表记录的Password列指定一个值。在Windows和Unix中的过程是相同的。下面的UPDATE语句同时为两个匿名账户指定密码：

```
shell> mysql -u root
mysql> UPDATE mysql.user SET Password = PASSWORD('newpwd')
    ->     WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

在user表中直接使用UPDATE更新密码后，必须让服务器用FLUSH PRIVILEGES重新读授权表。否则，重新启动服务器前，不会使用更改。

如果你宁愿删除匿名账户，操作方法是：

```
shell> mysql -u root
mysql> DELETE FROM mysql.user WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

可以在Windows和Unix中使用DELETE语句。在Windows中，如果你只想删掉具有与root相同权限的匿名账户，方法为：

```
shell> mysql -u root
mysql> DELETE FROM mysql.user WHERE Host='localhost' AND User='';
mysql> FLUSH PRIVILEGES;
```

该账户允许匿名访问，但是拥有全部的权限，因此删掉它可以提高安全。

你可以用几种方法为root账户指定密码。以下介绍了三种方法：

- 使用SET PASSWORD语句

- 使用**mysqladmin**命令行客户端程序
- 使用UPDATE语句

要想使用SET PASSWORD指定密码，用root连接服务器并执行两个SET PASSWORD语句。一定要使用PASSWORD()函数来加密密码。

在Windows中的语句：

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'%' = PASSWORD('newpwd');
```

在Unix中的语句：

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'host_name' = PASSWORD('newpwd');
```

用服务器主机名替换第二个SET PASSWORD语句中的*host_name*。这是你指定匿名账户密码的主机名。

要想使用**mysqladmin**为root账户指定密码，执行下面的命令：

```
shell> mysqladmin -u root password "newpwd"
shell> mysqladmin -u root -h host_name password "newpwd"
```

上述命令适用于Windows和Unix。用服务器主机名替换第二个命令中的*host_name*。不一定需要将密码用双引号引起来，但是你如果密码中包含空格或专用于命令解释的其它字符，则需要用双引号引起来。

你还可以使用UPDATE直接修改user表。下面的UPDATE语句可以同时为两个root账户指定密码：

```
shell> mysql -u root
mysql> UPDATE mysql.user SET Password = PASSWORD('newpwd')
    ->     WHERE User = 'root';
mysql> FLUSH PRIVILEGES;
```

UPDATE语句适用于Windows和Unix。

设置完密码后，当你连接服务器时你必须提供相应密码。例如，如果你想要用**mysqladmin** 关闭服务器，可以使用下面的命令：

```
shell> mysqladmin -u root -p shutdown
Enter password: (enter root password here)
```

注释：如果你设置了root密码后忘记了密码，从A.4.1节，“如何复位根用户密码”中查阅重设密码的方法。

你可以使用GRANT语句设置新账户。相关说明参见5.8.2节，“向MySQL增加新用户账户”。

2.10. 升级MySQL

2.10.1. 从5.0版升级
2.10.2. 升级授权表
2.10.3. 将MySQL数据库拷贝到另一台机器

做为一般原则，我们建议从一个发布系列升级到另一个发布系列时，你应当先升级到它的下一个系列而不要跳过。例如，如果你目前正运行MySQL 3.23，想要升级到较新的系列，要升级到MySQL 4.0而不要升级到5.0或5.1。

下面的项列出了升级时的相关信息：

- 从MySQL 5.0升级到5.1前,请阅读2.10.1节，“从5.0版升级”和附录D：*MySQL*变更史。其中提供了MySQL 5.1相对于MySQL 5.0的新特性或不同特性。如果你想要从MySQL 5.0以前的发布系列升级,应当依次升级到下一个发布系列，直到达到MySQL 5.0，然后再升级到MySQL 5.1。关于从MySQL 5.0升级的信息,参见MySQL *5.0* 参考手册；对于更早的发布，参见MySQL *4.1*参考手册。
- 在升级前应先备份数据库。

- 如果在Windows中运行MySQL服务器，参见2.3.15节，“在Windows下升级MySQL”。

- 从MySQL 5.0升级到5.1更改MySQL数据库中的 授权表；增加了列和表以支持新功能。为了充分利用这些特性，应确保授权表是最新的。升级授权表的过程参见2.10.2节，“升级授权表”。在升级前，你可能想要使用**mysqldump**来转储表；升级后，可以使用**mysql**或mysqlimport重建、重装 授权表来重载dump(备份)文件。

- 如果你正复制，关于升级复制设置的信息参见6.6节，“升级复制设置”。

- 如果安装包括**mysqld-max**服务器的MySQL-Max分发版，则后面要升级到非Max版的MySQL，**mysqld_safe**仍然试图运行旧的**mysqld-max**服务器。如果升级，应当手动删掉旧的**mysqld-max**服务器以确保**mysqld_safe**运行新的**mysqld**服务器。

只要你的MySQL版本属于相同的发布系列，总是可以在不同的版本之间的相同架构上移动MySQL格式文件和数据文件。目前的产品发布系列是5.1。如果运行MySQL时更改字符集，必须对所有MyISAM表运行**myisamchk -r -q --set-character-set=***charset*。否则，索引顺序将会出错，因为更改字符集也会更改排序。

如果你使用新版本时比较小心，在安装新的MySQL前可以重新命名旧的**mysqld**。例如，如果你想要从MySQL 5.0.13升级到5.1.10, 将当前的服务器从**mysqld**重新命名为**mysqld-5.0.13**。如果新的**mysqld**出现问题，只需要关闭并用旧的**mysqld**重新启动。

升级后，如果你遇到重新编译的客户端程序问题，例如Commands out of sync或不期望的内核转储，可能是编译程序时使用了旧的头文件或库文件。出现这种情况，应当检查mysql.h文件和libmysqlclient.a库的日期，以验证它们是否来自新的MySQL分发版。如果不是，用新的头文件和库重新编译程序。

如果出现问题，例如新**mysqld**服务器不启动或没有密码不能连接，验证是否使用了以前安装的旧my.cnf文件。可以用--print-defaults选项检查(例如，**mysqld --print-defaults**)。如果显示程序名之外的其它内容，说明有一个活动my.cnf文件影响了服务器或客户端操作。

当安装新的MySQL发布时，最好重建并重装Perl DBD：：mysql 模块。同样适用于其它MySQL接口，例如PHP mysql扩展名和Python MySQLdb模块。

2.10.1. 从**5.0**版升级

当从**5.0**升级到**5.0.10**或更高版本时请注意必须运行**mysql_fix_privilege_tables**(或在Windows中运行mysql_fix_privilege_tables.sql)。否则，不能创建保存的过程。相关过程参见2.10.2节，“升级授权表”。

2.10.2. 升级授权表

一些发布对授权表(MySQL数据库中的表)的结构进行了更改以增加新的权限或特性。当你更新到新版本 MySQL时，要想确保授权表最新，应当运行**mysql_fix_privilege_tables**脚本来更新 授权表。相关过程参见5.4节，“mysql_fix_privilege_tables：升级MySQL系统表”。

如果你从MySQL 4.1或更早版本升级，授权表升级过程为CREATE VIEW和SHOW VIEW权限增加了视图相关的列。这些权限位于全局和数据库 级。在这种情况下，MySQL 5.1版MySQL**_fix_privilege_tables**将user表中的Create_priv值复制到Create_view_priv和 Show_view_priv列。

2.10.3. 将**MySQL**数据库拷贝到另一台机器

你可以在支持相同浮点格式的不同架构之间为MyISAM表复制.frm、.MYI和.MYD文件。(MySQL关注所有字节交换问题）。请参见15.1节，“MyISAM存储引擎”。

如果你需要在不同的架构之间转移数据库，可以使用**mysqldump**创建含有SQL语句的文件。然后你可以将文件转移到其它机器上，并将它输入到MySQL客户端。

使用**mysqldump --help**来看有哪些选项可用。如果你正将数据移动到更新版本的MySQL，你应当使用**mysqldump –opt**来利用各种优化性能来产生更小、可以更快处理的转储文件。

在两台机器之间移动数据库的最简单(尽管不是最快)的方法是在数据库所在的机器上运行下面的命令：

```
shell> mysqladmin -h 'other_hostname' create db_name
shell> mysqldump --opt db_name | mysql -h 'other_hostname' db_name
```

如果你想要从远程机器通过慢速网络复制数据库，可以使用：

```
shell> mysqladmin create db_name
shell> mysqldump -h 'other_hostname' --opt --compress db_name | mysql db_name
```

还可以将结果保存到文件中，然后将文件转移到目标机器上并将文件装载到数据库中。例如，可以在源机器上使用下面的命令将数据库备份到文件中：

```
shell> mysqldump --quick db_name | gzip > db_name.contents.gz
```

(该例子中创建的文件是压缩格式）。将含有数据库内容的文件到目标机上并运行命令：

```
shell> mysqladmin create db_name
```

```
shell> gunzip < db_name.contents.gz | mysql db_name
```

还可以使用**mysqldump**和**mysqlimport**来转移数据库。对于大的表，比只是使用**mysqldump**要快得多。在下面的命令中，DUMPDIR代表用来保存**mysqldump**输出的目录全路径名。

首先，创建保存输出文件的目录并备份数据库：

```
shell> mkdir DUMPDIR

shell>mysqldump --tab=DUMPDIR db_name
```

然后将DUMPDIR目录中的文件转移到目标机上相应的目录中并将文件装载到MySQL：

```
shell> mysqladmin create db_name           # create database

shell> cat DUMPDIR/*.sql | mysql db_name   # create tables in database

shell> mysqlimport db_name DUMPDIR/*.txt   # load data into tables
```

不要忘记复制MySQL数据库，因为授权表保存在该数据库中。你可能需要在新机器上用MySQL root用户运行命令，直到产生MySQL数据库。

将mysql数据库导入目标机器后，执行**mysqladmin flush-privileges**，以便服务器重载授权表信息。

2.11. 降级MySQL

本节描述了旧版本MySQL比新版本工作得好的情况下，如何降级到旧的MySQL版本。

如果你在同一发布系列(例如，从 5.0.13 到5.0.12)内降级，一般规则是只需要在旧版本的顶部安装新的二进制。不需要对数据库进行任何操作。但是，最好是先进行备份。

下面的项列出了进行降级时应执行的操作：

- 阅读你将要降级的发布系列的升级部分，确定它没有你需要的功能。2.10节，“升级MySQL”。
- 如果该版本有降级部分，你也应当阅读。

只要MySQL版本属于相同的发布系列，你总是可以在不同的版本之间的相同架构上移动MySQL格式文件和数据文件。目前的产品发布系列是5.1。

如果你从一个发布系列降级到另一个发布系列，表储存格式可能不兼容。在这种情况下，你可以在降级嵌使用**mysqldump**来转储表。降级后，使用**mysql**或mysqlimport重载转储文件来重新创建表。请参见2.10.3节，“将MySQL数据库拷贝到另一台机器”。

表格式不向下兼容的一般迹象是降级时不能打开表。在这种情况下，使用下面的过程：

1. 停止你想要降级到的旧的MySQL服务器。
2. 重新启动将被降级的新的MySQL服务器。
3. 使用**mysqldump**创建一个dump（转储）文件来转储不能被旧服务器访问的所有表。
4. 停止新MySQL服务器，重新启动旧MySQL服务器。

将dump（转储）文件重载入旧服务器。表应当可访问。

2.12. 具体操作系统相关的注意事项

2.12.1. Linux注意事项
2.12.2. Mac OS X注意事项
2.12.3. Solaris注意事项
2.12.4. BSD注意事项
2.12.5. 其它Unix注意事项
2.12.6. OS/2注意事项

2.12.1. Linux注意事项

2.12.1.1. Linux操作系统注意事项
2.12.1.2. Linux二进制分发版说明
2.12.1.3. Linux源码分发版说明

2.12.1.4. Linux后期安装注意事项
2.12.1.5. Linux x86注意事项
2.12.1.6. Linux SPARC注意事项
2.12.1.7. Linux Alpha注意事项
2.12.1.8. Linux PowerPC注意事项
2.12.1.9. Linux MIPS注意事项
2.12.1.10. Linux IA-64注意事项

本节讨论发现的在Linux中出现的问题。前面几节描述了一般操作系统相关的问题，使用二进制或源码分发版时可能出现的问题和安装后的问题。后面几节讨论在具体Linux平台上出现的问题。

请注意这些问题的大多数出现在旧的Linux 版本中。如果你运行最新的版本，可能不会发现这些问题。

2.12.1.1. Linux操作系统注意事项

MySQL至少需要Linux 2.0版本。

警告：我们已经发现在SMP系统中Linux 2.2.14 和MySQL会出现一些奇怪的问题。我们还收到来自一些MySQL用户的报告说他们用内核2.2.14使用MySQL时遇到了严重的稳定性问题。如果你正使用该内核，应当升级到2.2.19 (或更新版)或到2.4内核。如果你有一个多CPU盒，应当考虑使用2.4，因为它能大大加速。你的系统将会更稳定。

当使用LinuxThreads时，你应当至少可以看见有三个**mysqld** 进程在运行。这些实际上是线程。有一个线程是LinuxThreads管理器，一个线程处理连接，另一个线程处理告警和信号。

2.12.1.2. Linux二进制分发版说明

MySQL的Linux-Intel二进制和RPM发布配置为最高的可能速度。我们一直在尝试使用可用的最快的稳定的编译器。

二进制发布用-staticis连接，说明一般情况你不需要关心系统库的版本。你也不需要安装LinuxThreads。用-staticis连接的程序稍微大于动态连接程序，但也稍微快一些(3-5%)。但是，静态连接程序的一个问题是你不能使用用户定义函数(UDF)。如果你将要写或使用UDF(只适用于C或C++ 编程人员)，你必须使用动态链接自己编译MySQL。

二进制分发版的一个已知问题是在使用libc的旧的Linux系统(例如Red Hat 4.x或Slackware)上，你会遇到一些(非致命)主机名解析问题。如果系统使用libc没有使用glibc2,你可能会遇到一些主机名解析和getpwnam()问题。这是因为glibc依靠一些外部库来执行主机名解析和getpwent(), 即使用-staticis编译也如此。这些问题出现在两个方面：

- 当运行**mysql_install_db**时，你会看见下面的错误消息：
- `Sorry, the host 'xxxx' could not be looked up`

你可以通过执行**mysql_install_db --force**来解决该问题，并不在**mysql_install_db**中执行**resolveip**测试。不利方面是 你不能在授权表中使用主机名：除了localhost，必须使用IP号。如果你正使用不支持—force的旧版本MySQL，必须使用文本编辑器手动卸载mysql_install中的resolveip测试。

- 当你尝试用--user选项运行**mysqld**时你还会看见下面的错误：
- `getpwnam: No such file or directory`

要解决该问题，使用su命令启动**mysqld**，不要指定--user选项。这样使系统自己更改**mysqld**进程的用户ID，**mysqld**不再需要这样做。

另一个解决办法可以解决两个问题，即不使用二进制分发版。获得MySQL源码分发版(RPM或tar.gz格式)并安装。

在一些Linux 2.2版本中，当客户端通过TCP/IP建立大量与**mysqld**服务器的新连接时，你可能会遇到错误Resource temporarily unavailable。该问题是Linux在你关闭TCP/IP套接字的时间和系统实际释放该套接字的时间之间有一个延迟。只有有限数目的TCP/IP时段有空间，因此如果客户端试图在短时间内建立许多新TCP/IP连接时，你会遇到源不可用错误。例如，当你通过TCP/IP运行MySQL test-connect基准测试时你会看见该错误。

我们已经多次向不同的 Linux 邮件列表询问该问题，但是一直没有找到合适的解决办法。唯一知道的"解决办法"是客户端使用永久连接，或，如果你在同一机器上运行数据库服务器和客户端，使用Unix套接字文件来连接，不要使用TCP/IP连接。

2.12.1.3. Linux源码分发版说明

下面关于glibc的注意事项只适用你自己构建MySQL时。如果在x86机器上运行Linux，在大多数情况下最好使用我们的二进制。我们将二进制连接到了我们能找到的打了最好补丁的glibc版本，并使用了最优的编译器选项，尽力使它适合高负荷服务器。对于典型用户，即使对于超过2GB限制的大量并行连接或表设置，在大多数情况下，我们的二进制仍然是最佳选择。阅读下面的内容后，如果你不清楚怎样做，先试用我们的二进制看它是否满足你的需求。如果你发现它不够完善，那么你可以尝试你自己的构建。在这种情况下，我们很希望能看到相关注释，以便我们下次可以构建更好的二进制。

MySQL在Linux上使用 LinuxThreads 。如果你正在使用一个没有glibc2的老的Linux版本，你必须在尝试编译**MySQL**前安装LinuxThreads。你可以从 http://dev.mysql.com/downloads/os-linux.html获得LinuxThreads。

注意：当你执行INSERT DELAYED时，所用的包括2.1.1及以前的glibc版本在pthread_互斥_timedwait()处理上有一个致命错误，如果你正在使用INSERT DELAYED，我们建议先升级glibc。

请注意Linux 内核和LinuxThreads库默认情况下最多可以处理1,024个线程。如果你计划使用超过1,000个并行连接，需要对LinuxThreads进行一些更改，如下所示：

- 将sysdeps/unix/sysv/linux/bits/local_lim.h中的PTHREAD_THREADS_MAX 增加到4096，将LinuxThreads/internals.h 中的STACK_SIZE减少到256KB。路径相对于glibc的根目录。(请注意如果STACK_SIZE为默认值2MB，MySQL有600-1000个连接时不稳定）。
- 重新编译LinuxThreads，生成新的libpthreads.a库，并重新连接MySQL。

可以从http://www.volano.com/linuxnotes.html 查阅关于LinuxThreads线程限制的详细信息。

还有一个问题会严重影响MySQL的性能，特别是在SMP系统中。在glibc 2.1的LinuxThreads中，对于拥有很多只是短时间内占有互斥体的线程的程序，不能很好地执行互斥。结果出现荒谬的现象：在许多情况下，如果你使用未修改的LinuxThreads连接MySQL，从SMP中去掉卸载处理器实际上会提高MySQL的性能。我们提供了一个glibc 2.1.3的补丁来纠正该行为(http://www.mysql.com/Downloads/Linux/linuxthreads-2.1-patch)。

在glibc 2.2.2中,MySQL使用修改后的互斥，这样甚至比打了补丁的glibc 2.1.3还要好得多。但是，需要注意的是在某些条件下，当前glibc 2.2.2中的互斥代码会高旋，影响了MySQL的性能。通过优化**mysqld**进程到最高优先级，可以降低在这种情况下出现这种问题的可能性。我们还可以通过补丁来纠正高旋问题，可以从http://www.mysql.com/Downloads/Linux/linuxthreads-2.2.2.patch下载。它集成高旋的纠正、线程最大数目和堆栈空间于一体。你需要用补丁patch -p0 </tmp/linuxthreads-2.2.2.patch在LinuxThreads目录下使用。我们 希望在将来的glibc 2.2发布中能以某种形式将它包括进来。在任何情况下，如果你连接glibc 2.2.2,仍然需要纠正STACK_SIZE和 PTHREAD_THREADS_MAX。我们希望在将来能将默认值纠正到某种程度上可以接受的值，适合高负荷MySQL设置，因此用来产生你自己的构建的 命令可以简化到到**./configure; make; make install**。

我们建议你使用这些补丁来构建专用的libpthreads.a静态版本，并只使用它同MySQL实现静态链接。我们知道这些补丁对于MySQL很安全 并大大改善了它的性能，但是我们还不能断言它在其它应用程序上的效果如何。如果你需要将其它使用LinuxThreads的应用程序同打了补丁的静态版本的库连接，或构建一个打了补丁的共享版本，并将它安装到系统中，你将自己承担风险。

如果你在安装MySQL的过程中遇到任何奇怪的问题，或一些常用实用工具被悬挂起来，很可能是库或编译器相关问题。在这种情况下，使用我们的二进制来解决。

如果你连接你自己的MySQL客户端程序，运行时你会看见下面的错误：

```
ld.so.1: fatal: libmysqlclient.so.#:
open failed: No such file or directory
```

使用下面的方法可以避免该问题：

- 用-Wl,r/full/path/to/libmysqlclient.so标记连接客户端，不要使用-Lpath)。
- 将libmysqclient.so复制到/usr/lib。
- 运行客户端程序前，将libmysqlclient.so所在目录的路径名增加到LD_RUN_PATH环境变量。

如果使用Fujitsu编译器 (fcc/FCC)，编译MySQ时可能会出现一些问题，因为Linux头文件主要面向**gcc**。以下的**configure**行应当结合**fcc/FCC**使用：

```
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
```

2.12.1.4. Linux后期安装注意事项

在MySQL安装目录或MySQL源码树下的support-files目录下可以找到**mysql.server**。你可以将它安装为/etc/init.d/mysql，以便自动启动和关闭MySQL。请参见2.9.2.2节，“自动启动和停止MySQL”。

如果MySQL不能打开足够的文件或连接，可能是你没有将Linux配置为处理足够的文件。

在Linux 2.2及以上版本中，你可以检查分配的文件的数目：

```
shell> cat /proc/sys/fs/file -max
shell> cat /proc/sys/fs/dquot-max
shell> cat /proc/sys/fs/super-max
```

如果有超过16MB的内存，应当在启动脚本中增加如下内容(例如，在SuSE Linux中：/etc/init.d/boot.local)：

```
echo 65536 > /proc/sys/fs/file-max
echo 8192 > /proc/sys/fs/dquot-max
echo 1024 > /proc/sys/fs/super-max
```

还可以用root从命令行运行echo命令，但是下一次重新启动计算机时这些设定值会丢掉。

另外，可以使用许多Linux分发版使用的sysctl工具设置启动参数(包括SuSE Linux 8.0和以后版本)。将下面的值放到文件 /etc/sysctl.conf中：

```
# Increase some values for MySQL
fs.file-max = 65536
fs.dquot-max = 8192
fs.super-max = 1024
```

你还应将以下内容加入/etc/my.cnf：

```
[mysqld_safe]
open-files-limit=8192
```

这样服务器连接和打开文件的总数目可以达到8,192。

LinuxThreads的STACK_SIZE常数控制在寻址空间中线程堆栈的占用空间。它需要足够大，以保证每个线程堆栈有足够的空间，但是应足够小，防止某些线程的堆栈运行全局**mysqld**数据。遗憾的是，根据我们的发现，如果你用mmap()映射目前正使用的寻址，Linux成功运行 后不会映射映射区，将会清空整个地址页上的数据而不会返回错误信息。因此，**mysqld**或其它线程应用程序的安全依靠创建线程的代码的"绅士"行为。用户必须采取措施以确保在任何时间运行线程的数目相对线程堆栈应足够低，以防止全局堆内存。使用**mysqld**时，你应当为max_connections变量设置合理的值强制该行为。

如果你自己构建MySQL，你可以为LinuxThreads打补丁以便更好地使用堆栈。请参见2.12.1.3节，"Linux源码分发版说明"。如果你不想为LinuxThreads打补丁，你应当将max_connections的值设置为不超过500。如果你有大的关键字缓冲区、大的堆内存表，或其它使**mysqld**分配大量内存的东西，或如果你用2GB的补丁运行2.2内核，max_connections的值应当更低。如果你正使用我们的二进制或RPM版本，可以安全地将max_connections设置为1500（假定没有大的关键字缓冲区或有大量数据的堆内存表。将LinuxThreads中的STACK_SIZE降低得越低，可以安全地创建越多的线程。我们推荐的值的范围为128KB到256KB。

如果你正使用大量的并行连接，在2.2内核中你可能会遇到一个"特性"，即通过对进程的分支或克隆子进程的行为进行罚分，来试图防止轰炸攻击。这样你增加并行客户端的数量时，MySQL不能正确响应。在单CPU系统中，我们已经发现该现象，即线程创建地很慢；连接MySQL的时间很长(长达1分钟)，并且关闭的时间也很长。在多CPU系统中，我们已经观察到随着客户端数目的增加，查询速度逐渐下降。在寻求解决办法的过程中，我们收到了一个用户的内核补丁，他声称该补丁可以解决他的问题。从http://www.mysql.com/Downloads/Patches/linux-fork.patch可以下载该补丁。我们已经在开发和生产系统上对该补丁进行了广泛的测试。它可以大大改善MySQL的性能，而不会造成任何问题，我们向仍然在2.2内核运行高负荷服务器的用户推荐它。

在2.4内核中已经对该问题进行了修复，因此如果你不满意当前系统的性能，不要为2.2 内核打补丁，在SMP系统中升级到2.4版要容易地多，升级不仅可以修复错误，还可以使SMP更好地加速。

我们已经在双CPU机器上在2.4内核中对MySQL进行了测试，发现MySQL的比例要好得多。1,000名客户查询时，还看不到速度有实质上的下降，MySQL的比例因子(最大吞吐量与一个客户端的吞吐量的比例)为180%。我们在四-CPU系统中也观察到了类似的结果：当客户数目上升到1,000名时还没有实质上的减慢, 比例因子为300%。根据这些结果，对于使用2.2内核的高负荷SMP服务器，我们绝对建议升级到 2.4内核。

我们已经发现，在2.4内核中应使用可能的最高优先级来运行**mysqld**进程，以获得最佳性能。可以通过为**mysqld_safe**加renice -20 $$命令来实现。在四-CPU系统中的测试中，提高优先级在400个客户时会使吞吐量增加60%。

我们目前还想搜集关于MySQL在four-way和eight-way系统上用2.4内核执行的性能的更详细信息。如果你访问了这些系统并进行了一些基准测试，请将结果用email发送到<benchmarks@mysql.com>。我们将对它们进行审查以包括进手册中。

如果用**ps**运行**mysqld**服务器进程时发现进程死掉了，一半是MySQL中有缺陷或数据库表崩溃了。请参见A.4.2节，"如果MySQL依然崩溃，应作些什么"。

如果**mysqld**死掉并给出SIGSEGV信号，要想在Linux中对内核进行调试，你可以用--core-file选项启动**mysqld**。注意你还可能需要通过向**mysqld_safe**添加**ulimit -c 1000000**或用--core-file-size=1000000启动**mysqld_safe**来加大内核文件的大小。请参见5.1.3节，"mysqld_safe：MySQL服务器启动脚本"。

2.12.1.5. Linux x86注意事项

MySQL需要5.4.12或更新版本的libc。已知它可以在libc 5.4.46中工作。glibc 2.0.6和更新版本应当也可以工作。Red Hat的glibc RPM有一些问题，因此如果你遇到问题，应看看是否有更新版。已知glibc 2.0.7-19和2.0.7-29 RPM可以工作。

如果你正使用Red Hat 8.0或更新的glibc 2.2.x库，你会看见**mysqld**执行gethostbyaddr()时会死掉。这是因为新的glibc库在执行该调用时需要大于128KB的堆栈空间。要想修复该问题，用--thread-stack=192K选项启动**mysqld**。(在MySQL 4以前的版本中应使用-O thread_stack=192K）。 在MySQL 4.0.10和以上版本中，该堆栈空间值即为默认值，因此你不会看见该问题。

如果你正使用**gcc** 3.0和以上版本编译MySQL，在编译MySQL前你必须先安装libstdc++v3库；如果不这样做，在联接过程中会遇到关于丢失__cxa_pure_virtual符号的错误。

在一些旧的Linux分发版中，**configure**会生成如下错误：

```
Syntax error in sched.h. Change _P to __P in the
/usr/include/sched.h file.
See the Installation chapter in the Reference Manual.
```

只需要按照错误消息的提示操作。对只有一条下划线的_P名再增加一条下划线，然后重新尝试。

编译时可能会出现警告。下面的警告可以忽略：

```
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'
```

如果**mysqld**启动时总是出现转储内核， 问题可能是你使用了旧的/lib/libc.a。尝试重新对其命名，然后删掉sql/**mysqld**并重新执行**make install**然后再次尝试。已经报导在部分Slackware安装中出现了该问题。

当链接**mysqld**时如果出现下面的错误，说明libg++.a安装不正确：

```
/usr/lib/libc.a(putc.o): In function `_IO_putc':
putc.o(.text+0x0): multiple definiti在of `_IO_putc'
```

你可以使用libg++.a运行**configure**来避免该问题：

```
shell> CXX=gcc ./configure
```

2.12.1.6. Linux SPARC注意事项

在部分执行过程中，readdir_r()会中断。现象是SHOW DATABASES语句总是返回空子集。可以在完成配置、进行编译前从config.h中删掉HAVE_READDIR_R来修复。

2.12.1.7. Linux Alpha注意事项

我们已经用我们的基准和测试套件在Alpha中测试了MySQL 5.1，看起来工作得不错。

我们目前在配置Alpha EV6处理器的Compaq DS20机器上，在SuSE Linux 7.0中对AXP、内核2.4.4-SMP、Compaq C编译器(V6.2-505)和Compaq C++编译器(V6.3-006)构建MySQL二进制软件包。

你可以从 http://www.support.compaq.com/alpha-tools/ 找到上述编译器。使用这些编译器，得到的MySQL性能比**gcc**要好9-14%。

在Alpha版MySQL中，我们在编译选项中使用-arch generic标记，可以确保二进制在所有Alpha处理器中运行。我们还采用静态编译以避免库问题。**configure**命令应为：

```
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
```

如果想要使用**egcs**，可以使用下面的**configure**行：

```
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
```

已知的在Linux-Alpha中运行MySQL的问题：

- 在gdb 4.18 中不能调试线程应用程序。你应当使用**gdb** 5.1。
- 使用**gcc**时如果你尝试静态链接**mysqld**，启动时映像会导致转储内核。换句话说，使用**gcc**时不要使用--with-mysqld-ldflags=-all-static。

2.12.1.8. Linux PowerPC注意事项

MySQL应当可以在使用最新glibc安装软件包(用glibc 2.0.7测试)的MkLinux上工作。

2.12.1.9. Linux MIPS注意事项

为了使MySQL在Qube2(Linux Mips)上工作，需要最新的glibc库。已知glibc-2.0.7-29C2可以工作。还必须使用**egcs** C++编译器(**egcs** 1.0.2-9、**gcc** 2.95.2或更新版)。

2.12.1.10. Linux IA-64注意事项

要想让MySQL在Linux IA-64中编译,我们用 **gcc** 2.96构建时使用下面的**configure**命令:

```
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
```

在IA-64中,MySQL客户端二进制使用共享库。这意味着如果你将我们的二进制分发版安装到/usr/local/mysql之外的其它位置，需要将libmysqlclient.so安装目录路径加到/etc/ld.so.conf文件或LD_LIBRARY_PATH环境变量中。

参见A.3.1节，“与MySQL客户端库的链接问题”。

2.12.2. Mac OS X注意事项

2.12.2.1. Mac OS X 10.x (Darwin)
2.12.2.2. Mac OS X Server 1.2 (Rhapsody)

在Mac OS X中，**tar**不能处理长文件名。如果你需要解包.tar.gz分发版，应使用**gnutar**。

2.12.2.1. Mac OS X 10.x (Darwin)

MySQL应当可以在Mac OS X 10.x(Darwin)中工作，不会有重大问题。

已知的问题有:

- 连接时间(wait_timeout、interactive_timeout和net_read_timeout)值不理想。

这可能是线程库中的信号处理问题，信号不能中断挂起的读命令，我们希望将来能对线程库进行更新以修复该问题。

在Darwin 6.3中用下面的**configure**行对Mac OS X二进制进行了编译:

```
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
```

参见2.5节，“在Mac OS X上安装MySQL”。

2.12.2.2. Mac OS X Server 1.2 (Rhapsody)

对于当前版本的Mac OS X服务器，编译MySQL之前不需要对操作系统进行更改。在服务器平台上的编译与客户端版本Mac OS X相同。

对于旧的版本(Mac OS X Server 1.2, a.k.a. Rhapsody)，在配置MySQL之前必须先安装pthread软件包。

参见2.5节，“在Mac OS X上安装MySQL”。

2.12.3. Solaris注意事项

2.12.3.1. Solaris 2.7/2.8注意事项
2.12.3.2. Solaris x86注意事项

在Solaris中，甚至在解压缩MySQL分发版时，你也会遇到问题，因为Solaris **tar**不能处理长文件名。这说明解压缩MySQL分发版时会遇见错误。

如果出现这种问题，必须使用GNU **tar**(**gtar**)来解压缩分发版。可以从http://dev.mysql.com/downloads/os-solaris.html找到预编译的Solaris版本。

Sun原生线程只在Solaris 2.5和更高版本中工作。对于版本2.4和更早版本，MySQL自动使用MIT-pthreads。请参见2.8.5节，“MIT-pthreads注意事项”。

如果你运行**configure**时遇到下面的错误，说明编译器安装得有问题：

```
checking for restartable system calls... configure: error can not
run test programs while cross compiling
```

在这种情况下，应当将编译器升级到较新的版本。还可以在config.cache文件中插入下面的行来解决该问题：

```
ac_cv_sys_restartable_syscalls=${ac_cv_sys_restartable_syscalls='no'}
```

如果在SPARC上使用Solaris，建议使用编译器**gcc** 2.95.2或3.2。可以从http://gcc.gnu.org/下载。请注意**egcs** 1.1.1和**gcc** 2.8.1不能在SPARC上可靠地工作。

当使用**gcc** 2.95.2时建议的**configure**行：

```
CC=gcc CFLAGS="-O3" \
CXX=gcc CXXFLAGS="-O3 -felide-constructors -fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql --with-low-memory \
    --enable-assembler
```

如果你使用UltraSPARC系统，在CFLAGS和CXXFLAGS环境变量中加上-mcpu=v8 -Wa,-xarch=v8plusa，性能可以提高4%。

如果你有Sun Forte 5.0(或更新版)编译器，可以运行：

```
CC=cc CFLAGS="-Xa -fast -native -xstrconst -mt" \
CXX=CC CXXFLAGS="-noex -mt" \
./configure --prefix=/usr/local/mysql --enable-assembler
```

要想用Sun Forte 编译器创建64-位二进制，使用下面的配置选项：

```
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
```

要想使用**gcc**创建64-位Solaris二进制，在CFLAGS和CXXFLAGS中增加-m64，并从**configure**行删掉--enable-assembler。

在MySQL基准测试中，在UltraSPARC上使用32位模式的Forte 5.0，同使用带-mcpu标记的**gcc** 3.2比较，速度提高了4%。

如果创建了64位的**mysqld**二进制，速度要比32-位二进制慢4%，但是可以处理更多的线程和内存。

当为x86_64使用Solaris 10时,你应当安装用forcedirectio选项保存InnoDB文件的文件系统。(默认情况不用该选项来安装)。如果不这样做，当在该平台上使用InnoDB 储存引擎时将会造成严重的性能下降。

如果fdatasync或sched_yield出现问题，你可以在**configure**行加LIBS=-lrt修复该问题。

对于早于WorkShop 5.3的编译器,你可能需要编辑**configure**脚本。应将该行：

```
#if !defined(__STDC__) || __STDC__ != 1
```

更该为:

```
#if !defined(__STDC__)
```

如果你用-Xc 选项打开__STDC__，Sun 编译器不能用Solaris pthreads.h 头文件编译。这是一个Sun 缺陷(中断的编译器或include文件)。

如果运行**mysqld**时出现下面的错误消息，原因是你企图不启用-mt multi-thread选项而用Sun编译器编译MySQL:

libc internal error: _rmutex_unlock: rmutex not held

将-mt加到CFLAGS和CXXFLAGS上并重新编译。

如果你正使用**gcc**的SFW版本(在Solaris 8安装包内)，运行**configure**之前你必须将/opt/sfw/lib加到环境变量LD_LIBRARY_PATH上。

如果你正使用从sunfreeware.com下载的**gcc**，会遇到许多问题。要想避免，应当在运行机器上重新编译**gcc**和GNU binutils。

如果你用**gcc**编译MySQL时遇到下面的错误，说明**gcc**配置不适合Solaris的版本:

```
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'
```

在这种情况下正确的做法是得到最新版本的**gcc**，并用你当前的**gcc**编译器编译。至少对于Solaris 2.5,几乎所有二进制版本的**gcc**有旧的不再使用的include文件,中断了所有使用线程的程序，也可能会中断其它程序。

Solaris不提供静态版本的系统库(libpthreads和libdl)，因此你不能用--static编译MySQL。如果你试图这样做，会遇到下面的错误:

```
ld: fatal: library -ldl: not found
undefined reference to `dlopen'
cannot find -lrt
```

如果你连接你自己的MySQL客户端程序，运行时会出现下面的错误:

```
ld.so.1: fatal: libmysqlclient.so.#:
open failed: No such file or directory
```

可以通过下面的方法避免该问题:

- 用-Wl,r/full/path/to/libmysqlclient.so标记而不要用-Lpath)连接客户端。
- 将libmysqclient.so复制到/usr/lib。
- 运行客户端程序前，将libmysqlclient.so目录的路径名加到LD_RUN_PATH环境变量上。

如果你未安装zlib用 -lz连接时**configure**出现问题，你有两个选则:

- 如果你想要使用压缩通信协议，你需要从ftp.gnu.org获得并安装zlib。
- 构建MySQL时用--with-named-z-libs=no选项运行**configure**。

如果你正使用**gcc**，向MySQL调用用户定义函数(UDF)时出现问题，尝试向UDF连接行添加-lgcc到。

如果想让MySQL自动启动，可以将support-files/mysql.server复制到/etc/init.d，并创建/etc/rc3.d/S99**mysql.server**符号连接。

如果有太多进程试图快速连接**mysqld**，你应当能再MySQL日志只看见错误:

```
Error in accept: Protocol error
```

可以尝试用--back_log=50选项启动服务器来解决该问题。(在MySQL 4以前的版本中使用-O back_log=50）。

Solaris不支持setuid() 应用程序所需的内核文件，因此如果你使用--user选项你不能从**mysqld**遇到内核文件。

2.12.3.1. Solaris 2.7/2.8注意事项

一般情况，可以在Solaris 2.7和2.8中使用Solaris 2.6二进制。大多数Solaris 2.6发布也适用Solaris 2.7和2.8。

MySQL应当能够自动检测到新的Solaris版本并解决下面的问题。

Solaris 2.7/2.8的include文件中有一些缺陷。当使用**gcc**时你会看见下面的错误：

```
/usr/include/widec.h:42: warning: `getwc' redefined
/usr/include/wchar.h:326: warning: this is the location of the previous
definition
```

如果出现该问题，修复的办法是将/usr/include/widec.h复制到.../lib/gcc-lib/os/gcc-version/include，并将41行从：

```
#if     !defined(lint) && !defined(__lint)
```

更改为：

```
#if     !defined(lint) && !defined(__lint) && !defined(getwc)
```

另外，可以直接编辑/usr/include/widec.h。用任何一种方法修复后，应当删掉config.cache并再次运行**configure**。

如果运行**make**时遇到下面的错误，这是因为**configure**不能检测到curses.h文件(可能是因为/usr/include/widec.h中的错误)：

```
In file included from mysql.cc:50:
/usr/include/term.h:1060: syntax error before `,'
/usr/include/term.h:1081: syntax error before `;'
```

可以用下面任何一种方法解决该问题：

- 用CFLAGS=-DHAVE_CURSES_H CXXFLAGS=-DHAVE_CURSES_H ./configure配置。
- 根据前面的讨论编辑/usr/include/widec.h并重新**configure**。
- 删掉config.h文件中的#define HAVE_TERM行并重新运行**make**。

如果链接客户端程序时连接器找不到-lz，问题可能是libz.so文件安装到了/usr/local/lib。可以用下面任何一种方法解决该问题：

- 向LD_LIBRARY_PATH增加/usr/local/lib。
- 增加到libz.so from /lib的连接。
- 如果使用Solaris 8,可以安装Solaris 8 CD分发版中的可选项zlib。
- 构建MySQL时用--with-named-z-libs=no选项运行**configure**。

2.12.3.2. Solaris x86注意事项

在x86 Solaris 8上, 如果使用strip删除调试符号，**mysqld**转储内核。

如果正在 Solaris x86上使用**gcc**或**egcs**，并且你碰到了在装载时的内核转储问题，应该使用下列**configure**命令：

```
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
```

用libstdc++库和与C++异常处理来避免这个问题。

如果这没有帮助，你应该编译一个调试版本并且用一个踪迹文件运行它或在gdb下运行它。请参见E.1.3节，“在gdb环境下调试mysqld”。

2.12.4. BSD注意事项

2.12.4.1. FreeBSD注意事项
2.12.4.2. NetBSD

注意事项
2.12.4.3. OpenBSD 2.5版注意事项
2.12.4.4. BSD/OS 2.x版注意事项
2.12.4.5. BSD/OS 3.x版注意事项
2.12.4.6. BSD/OS 4.x版注意事项

本节提供在各种BSD Unix中使用MySQL的相关信息。

2.12.4.1. FreeBSD注意事项

对于运行MySQL，推荐使用FreeBSD 4.x或更新版本，因为其线程包更加完整。要想遇到一个安全、稳定的系统，你应当只使用标记-RELEASE的FreeBSD内核。

最容易因此是比较受喜欢的安装方法是使用**mysql-server**和mysql-client移植，可从http://www.freebsd.org得到。

使用这些移植的益处包括：

- 一个全优化的工作在FreeBSD版本上的**MySQL**。
- 自动配置和构建。
- 启动脚本安装在/usr/local/etc/rc.d。

· 用pkg_info -L察看哪个文件被安装的能力

· 如果你在那台机器上不再想要**MySQL**，用pkg_delete完全删除它们。

建议在FreeBSD 2.x版上使用 MIT-pthreads，在版本3和以上版本用原生线程。在一些2.2.x的以后版本使用原生线程运行是可能的，但是你可能碰到**mysqld**关掉的问题。

遗憾的是，FreeBSD调用的某些函数还不能完全保证线程安全。特别令人注意的是，包括gethostbyname()函数， MySQL使用该函数将主机名转换为IP地址。在某些环境中，**mysqld**进程会突然造成100%的CPU负荷，不再响应。如果你遇到该问题，尝试使用--skip-name-resolve选项启动MySQL。

另外，还可以将FreeBSD 4.x中的MySQL同LinuxThreads库连接，这样可以避免一些原生FreeBSD线程执行时的问题。为了更好地将LinuxThreads同原生线程进行对比，参见Jeremy Zawodny的文章*FreeBSD or Linux for your MySQL Server?* 地址：http://jeremy.zawodny.com/blog/archives/000697.html 。

在FreeBSD使用LinuxThreads的已知问题有：

· 连接时间(wait_timeout、interactive_timeout和net_read_timeout)值不理想。现象是永久连接挂起较长时间，不能关闭，只有线程执行新命令时 'kill'线程方有效。

这可能是线程库中的信号处理问题，信号不能中断挂起的读命令。期望在FreeBSD 5.0中修复。

MySQL构建进程需要GNU make(**gmake**)工作。如果没有GNU **make**，必须在编译MySQL前先安装它。

建议的在FreeBSD中用**gcc**(2.95.2和以上版本)编译和安装MySQL的方法是：

```
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
cd /usr/local/mysql
bin/mysql_install_db --user=mysql
bin/mysqld_safe &
```

如果**configure**使用MIT-pthreads，应当阅读MIT-pthreads注意事项。请参见2.8.5节，“MIT-pthreads注意事项”。

如果你从**make install**遇到一个它不能找到“/usr/include/pthreads”的错误，**configure**没有检测出你需要MIT-pthreads。要修复该问题，移走config.cache，然后用--with-mit-threads选项重新运行**configure**。

确定让你的名字解析程序安装正确，否则当连接**mysqld**时，你可能会遇到解析延时或失败。保证在“/etc/hosts”文件中的localhost入口是正确的。“/etc/hosts”文件应该以下面一行开始：

```
127.0.0.1       localhost localhost.your.domain
```

已知FreeBSD的文件句柄限制默认值很低。请参见A.2.17节，“文件未找到”。使用--open-files-limit选项执行**mysqld_safe**启动服务器，或在/etc/login.conf为**mysqld**用户提高限制并用cap_mkdb /etc/login.conf重建它。如果你不使用默认名(即使用chpass mysqld-user-name)，还要保证为该用户设置在密码文件中的相应级别。请参见5.1.3节，“mysqld_safe：MySQL服务器启动脚本”。

FreeBSD将进程空间限制到512MB，即使系统有很多的可用RAM也如此。因此你可能会遇到如下所示错误：

```
Out of memory (Needed 16391 bytes)
```

在当前版本的FreeBSD(至少4.x和更高版本)中，你可以在/boot/loader.conf文件中增加下面的条目并重启机器增加该限制(不能在运行时间用**sysctl**命令更改这些设定值)：

```
kern.maxdsiz="1073741824" # 1GB
kern.dfldsiz="1073741824" # 1GB
kern.maxssiz="134217728" # 128MB
```

在旧版本的FreeBSD中，必须重新编译内核，以便更改进程的最大数据段空间。在这种情况下，应当查看LINT配置文件中的MAXDSIZ选项查看更详细的信息。

如果MySQL的当前日期有问题，设置TZ变量将会有帮助。请参见附录F：环境变量。

2.12.4.2. NetBSD注意事项

为了在NetBSD上编译，你需要GNU make，否则当make试图在C++文件上运行lint时，编译将失败。

2.12.4.3. OpenBSD 2.5版注意事项

在OpenBSD 2.5上，可以用下列选项编译带原生线程的**MySQL**：

```
CFLAGS=-pthread CXXFLAGS=-pthread ./configure --with-mit-threads=no
```

2.12.4.4. BSD/OS 2.x版注意事项

如果在编译**MySQL**时，遇到下列错误，说明虚拟内存的**ulimit**值太低了：

```
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
```

试试使用**ulimit -v 80000**并再次运行**make**。如果这还不行并且你正在使用**bash**，试试换到**csh**或**sh**；一些BSDI用户报告了使用**bash**和**ulimit**有问题。

如果你正在使用**gcc**，可能必须为**configure**使用--with-low-memory标志才能编译“sql_yacc.cc”。

如果MySQL的当前日期有问题，设置TZ变量将会有帮助。请参见附录F：环境变量。

2.12.4.5. BSD/OS 3.x版注意事项

升级到BSD/OS 3.1。如果不能升级，则安装BSDI patch M300-038。

在配置**MySQL**时，使用下列命令:

```
env CXX=shlicc++ CC=shlicc2 \
./configure \
    --prefix=/usr/local/mysql \
    --localstatedir=/var/mysql \
    --without-perl \
    --with-unix-socket-path=/var/mysql/mysql.sock
```

已知下面的项可以工作：

```
env CC=gcc CXX=gcc CXXFLAGS=-O3 \
./configure \
    --prefix=/usr/local/mysql \
```

```
    --with-unix-socket-path=/var/mysql/mysql.sock
```

如果你愿意，可以改变目录位置，或不指定任何位置而使用默认目录。

如果在重负载下的性能有问题，试试使用--skip-thread-priority选项执行**mysqld**！这将以相同优先级运行所有线程；在BSDI 3.1上，可以得到较好的性能(至少直到BSDI可以修正其线程调度程序)。

如果在编译时遇到virtual memory exhausted错误，试试使用**ulimit -v 80000**并再次运行**make**。如果这还不行并且你正在使用**bash**，试试换到**csh**或**sh**；一些BSDI用户报告了使用**bash**和**ulimit**会有问题。

2.12.4.6. BSD/OS 4.x版注意事项

BSDI 4.x有一些线程有关的缺陷。如果你想要在这上面使用**MySQL**，应该安装所有与线程相关的补丁，至少应该安装M400-023。

在一些 BSDI 4.x系统中，你可能会遇到共享库问题。现象是不能执行任何客户端程序，例如，**mysqladmin**。在这种情况下，需要重新配置，用disable-shared选项禁用库的共享以便进行配置。

一些客户在BSDI 4.0.1中遇到了严重问题，即**mysqld**二进制过一会儿不能打开表。这是因为一些库/系统相关错误使**mysqld**未得到指令便更改了当前目录。

修复方法是升级MySQL到最低3.23.34版或更高版，运行**configure**后，运行**make**前删掉config.h中的行#define HAVE_REALPATH。

请注意这说明你不能通过符号连接将数据库目录连接到另一个数据库目录，或通过符号连接将表连接到BSDI上的另一个数据库。(可以通过符号连接连接到另一个硬盘)。

2.12.5. 其它**Unix**注意事项

2.12.5.1. HP-UX 10.20版注意事项
2.12.5.2. HP-UX 11.x版注意事项
2.12.5.3. IBM-AIX注意事项
2.12.5.4. SunOS 4注意事项
2.12.5.5. Alpha-DEC-UNIX(Tru64)版注意事项
2.12.5.6. Alpha-DEC-OSF/1注意事项
2.12.5.7. SGI Irix注意事项
2.12.5.8. SCO UNIX和OpenServer 5.0.x版注意事项
2.12.5.9. SCO UnixWare 7.1.x和OpenUNIX 8.0.0版注意事项
2.12.5.10. SCO OpenServer 6.0.x版注意事项

2.12.5.1. HP-UX 10.20版注意事项

在HP-UX上编译**MySQL**时，有一些"小"问题。我们推荐您使用**gcc**而不是HP-UX自己的编译器，因为**gcc**可以生成更好的代码！

我们推荐在HP-UX上使用**gcc** 2.95，不使用高度优化标志(类似-O6)，因为这在HP-UX 上可能不安全。

下面的**configure**行在**gcc** 2.95中应当工作：

```
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
```

下面的**configure**行在**gcc** 3.1中应当工作：

```
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
```

2.12.5.2. HP-UX 11.x版注意事项

由于在标准 HP-UX库中有一些严重错误，试图在HP-UX 11.0上运行MySQL前应当安装下面的补丁：

```
PHKL_22840 Streams cumulative
PHNE_22397 ARPA cumulative
```

这样可以解决在线程应用程序中运行recv()时得到EWOULDBLOCK和执行accept()时得到EBADF的问题。

如果你在未打补丁的HP-UX 11.x系统中使用**gcc** 2.95.1，可能会遇到下面的错误：

```
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:
```

问题是HP-UX没有一致地定义pthreads_atfork()。它同/usr/include/sys/unistd.h：184和/usr/include/sys/pthreads.h：440中的原型冲突。

一个解决办法是将/usr/include/sys/unistd.h复制到MySQL/include并编辑unistd.h，对它进行更改以匹配pthreads.h中的定义。查找下列行：

```
extern int pthreads_atfork(void (*prepare)(), void (*parent)(),
                                          void (*child)());
```

将它更改为：

```
extern int pthreads_atfork(void (*prepare)(void), void (*parent)(void),
                                          void (*child)(void));
```

更改完后，下面的**configure**行应当工作：

```
CFLAGS="-fomit-frame-pointer -O3 -fpic" CXX=gcc \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti -O3" \
./configure --prefix=/usr/local/mysql --disable-shared
```

如果你使用HP-UX 编译器，可以使用下面的命令(已经用**cc** B.11.11.04进行了测试)：

```
CC=cc CXX=aCC CFLAGS=+DD64 CXXFLAGS=+DD64 ./configure \
    --with-extra-character-set=complex
```

你可以忽略下面类型的任何错误：

```
aCC: warning 901: unknown option: `-3': use +help for online
documentation
```

如果运行**configure**时遇到下面的错误，请验证在到HP-UX C和C++编译器的路径前是否有到K&R编译器的路径：

```
checking for cc option to accept ANSI C... no
configure: error: MySQL requires an ANSI C compiler (and a C++ compiler).
Try gcc. See the Installation chapter in the Reference Manual.
```

不能编译的另一个原因可能是你没有按照前面所述定义+DD64标记。

对于HP-UX 11，另一个办法是使用http://dev.mysql.com/downloads/提供的MySQL二进制，这是我们已经构建并测试过的。我们也收到了MySQL的HP-UX 10.20二进制可以在HP-UX 11上成功运行的报告。如果你遇到问题，一定要检查HP-UX补丁的版本。

2.12.5.3. IBM-AIX注意事项

Autoconf中没有了xlC自动检测，因此运行**configure**前，需要设置许多变量。以下例子使用IBM编译器：

```
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 \
                --sbindir='/usr/local/bin' \
                --libexecdir='/usr/local/bin' \
                --enable-thread-safe-client \
                --enable-large-files
```

可以从http://www-frec.bull.com/找到上述编译MySQL分发版的选项。

如果你将前面的**configure**行中的-O3改为-O2，必须删掉-qstrict选项。这是对IBM C编译器的限制。

如果你正在使用**gcc**或**egcs**编译**MySQL**，必须使用-fno-exceptions标志，因为**gcc/egcs**的异常处理不是对线程安全的！（已经用**egcs**1.1测试过）。IBM汇编程序也有一些已知问题，当使用**gcc**时可以生成坏的代码。

在 AIX 上使用**egcs**和**gcc 2.95**，我们推荐下列**configure**行：

```
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
```

为了成功编译，需要使用-Wa,-many选项。IBM知道该问题，但是由于工作区的原因，不急于修复它。我们不知道**gcc** 2.95是否需要-fno-exceptions选项,但是因为MySQL不使用例外，并且该选项可以产生快速的代码，我们建议使用**egcs**/**gcc**时使用该选项。

如果汇编码有问题，试试更改-mcpu=*xxx*选项以匹配你的CPU。一般情况可能需要使用power2、power或powerpc。另外，你可能需要使用604或604e。我们不能绝对肯定但怀疑power一般应安全，即使在power2机器上也安全。

如果你不知道你的CPU是什么样的，可以执行uname -m命令。可以产生类似000514676700的字符串，格式为xxyyyyyymmss，其中xx和ss总是00,yyyyyy是唯一的系统ID，mm是CPU Planar的ID。从http://www16.boulder.ibm.com/pseries/en_US/cmds/aixcmds5/uname.htm可以找到这些值的图表。

该图表可以给出你可以使用的机器类型和机器型号，确定CPU的类型。

如果遇到信号问题(**MySQL**在高负载下出人意料地死掉)，你可能已经发现了与线程和信号有关的一个OS 缺陷。在这种情况下，你可以用如下配置告诉**MySQL**不使用信号：

```
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
```

这不影响**MySQL**的性能，但是有副作用，就是不能用**mysqladmin kill**或**mysqladmin shutdown**杀死在一个连接上"睡眠"的客户端。相反，当客户端发出下一个命令时，它将死掉。

在AIX的一些版本上用libbind.a链接使得getservbyname（）内核转储，这是AIX的一个缺陷并且应该报告IBM。

对于AIX 4.2.1和**gcc**，进行下面的更改。

配置完后，编辑config.h和include/my_config.h，将下面的行：

```
#define HAVE_SNPRINTF 1
```

更改为：

```
#undef HAVE_SNPRINTF
```

最后需要在mysqld.cc中为initgroups ()添加：

```
#ifdef _AIX41
extern "C" int initgroups(const char *,int);
#endif
```

如果你需要为**mysqld**进程分配大量内存，只使用**ulimit -d unlimited**是不够的。还可以需要修改**mysqld_safe**，增加下面的行：

```
export LDR_CNTRL='MAXDATA=0x80000000'
```

关于使用大量内存的信息请查阅http://publib16.boulder.ibm.com/pseries/en_US/aixprggd/genprogc/lrg_prg_support.htm 。

2.12.5.4. SunOS 4注意事项

在SunOS 4上，需要用MIT-pthreads编译**MySQL**，反过来这意味着需要使用GNU make。

一些SunOS 4系统的动态库和**libtool**有问题。可以使用下列**configure**行来避免这个问题：

```
./configure --disable-shared --with-mysqld-ldflags=-all-static
```

当编译readline时，可能会遇到关于重复定义的警告，可以忽略。

当编译**mysqld**的时候，将会遇到implicit declaration of function警告，可以忽略。

2.12.5.5. Alpha-DEC-UNIX(Tru64)版注意事项

如果你在Digital Unix上使用**egcs** 1.1.2，应当升级到**gcc** 2.95.2,因为**egcs**在DEC上有一些严重缺陷！

当在Digital UNIX下编译线程程序时，文档推荐对于**cc**和**cxx**使用-pthread选项和-lmach -lexc库（除了-lpthread外)。你应该象下面这样运行**configure**：

```
CC="cc -pthread" CXX="cxx -pthread -O" \
./configure --with-named-thread-libs="-lpthread -lmach -lexc -lc"
```

当编译**mysqld**时，会看见下面的警告：

```
mysqld.cc: In function void handle_connections()':
mysqld.cc:626: passing long unsigned int *' as argument 3 of
accept(int,sockadddr *, int *)'
```

你可以放心地忽略这些警告，它们的发生是因为**configure**只能检测出错误，而不是警告。

如果直接从命令行启动服务器，当退出时，可能有死掉的问题。（当你退出时，重要进程收到一个SIGHUP信号)，如果如此，试着象这样启动服务器：

```
nohup mysqld [options] &
```

nohup使得后面的命令忽视从终端发来的任何SIGHUP信号了。另外，通过运行**safe_mysqld**启动服务器，可以让**nohup**调用**mysqld**。请参见5.1.3节，“mysqld_safe：MySQL服务器启动脚本”。

如果编译mysys/get_opt.c时遇到问题，只需要从启动文件删掉#define _NO_PROTO行。

如果使用Compaq的CC编译器，下面的**configure**行应当工作：

```
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
```

如果用共享库编译时**libtool**出现问题，应当能够在链接**mysql**时通过下述命令解决该问题：

```
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
```

2.12.5.6. Alpha-DEC-OSF/1注意事项

如果遇到编译问题并安装了DEC **CC**和**gcc**，尝试这样运行**configure**：

```
CC=cc CFLAGS=-O CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql
```

如果遇到“c_asm.h”文件的问题，可以创建并使用一个“哑(dummy)”“c_asm.h”文件：

```
touch include/c_asm.h
CC=gcc CFLAGS=-I./include \
CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql
```

请注意从http://ftp.support.compaq.com/public/unix/下载最新的DEC(Compaq)补丁可以修复**ld**程序的下面问题。

在OSF1 V4.0D和编译器"DEC C V5.6-071 on Digital UNIX V4.0 (Rev. 878)" 上，编译器有了一些奇怪的行为(未定义asm标志)。/bin/ld好象也被破坏(当链接**mysqld**时，发生_exit undefined问题)。在这个系统上，从OSF 4.0C的版本替换了/bin/ld后，我们设法用下列**configure**命令行编译**MySQL**：

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
```

用Digital编译器“C++ V6.1-029”，下列命令应该工作：

```
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"
```

在OSF1的某些版本中，alloca()函数被破坏，通过删除定义'HAVE_ALLOCA'的“config.h”的行更正它。

alloca()函数也可能在/usr/include/alloca.h里面有一个不正确的原型，源于此的警告可以忽略。

configure将自动地使用下列线程库： --with-named-thread-libs="-lpthread -lmach -lexc -lc"。

当使用**gcc**时，也可以试试这样运行**configure**：

```
CFLAGS=-D_PTHREAD_USE_D4 CXX=gcc CXXFLAGS=-O3 ./configure ....
```

如果你的信号有问题(**MySQL**出人意料地在高负载下死掉)，可能遇到了一个线程和信号有关的OS 缺陷。在这种情况下，你可以告诉**MySQL**按如下配置不使用信号：

```
CFLAGS=-DDONT_USE_THR_ALARM \
CXXFLAGS=-DDONT_USE_THR_ALARM \
./configure
```

这不影响**MySQL**性能，但是有副作用，就是不能用**mysqladmin kill**或**mysqladmin shutdown**杀死在一个连接上“睡眠”的客户端。相反，当客户端发出下一个命令时，它将死掉。

使用**gcc** 2.95.2,会遇到下面的编译错误：

```
sql_acl.cc:1456: Internal compiler error in `scan_region',
at except.c:2566
Please submit a full bug report.
```

要修复该问题，应当进入sql目录并剪切、复制最后一行**gcc**，但应将-O3 改为-O0 (或如果编译行没有任何-O选项，应在**gcc**后面增加-O0)。之后，可以回到顶级目录并重新运行**make**。

2.12.5.7. SGI Irix注意事项

如果使用Irix 6.5.3或更新版，如果用拥有CAP_SCHED_MGT权限的用户(例如root)运行**mysqld**或用下面的shell命令给**mysqld**服务器该权限，**mysqld**能够创建线程：

```
chcap "CAP_SCHED_MGT+epi" /opt/mysql/libexec/mysqld
```

可能需要在运行**configure**后且在编译前定义“config.h”里面的一些符号。

在一些Irix实现中，alloca()函数被破坏。如果**mysqld**服务器死于一些SELECT语句，把“config.h”定义HAVE_ALLOC和HAVE_ALLOCA_H的行删除即可。如果**mysqladmin create**不工作，把“config.h”定义HAVE_READDIR_R的行删除，你也可能必须删除HAVE_TERM_H行。

SGI推荐将本页上的所有补丁作为一个整体来安装：

http://support.sgi.com/surfzone/patches/patchset/6.2_indigo.rps.html

至少，应该安装最新的核心卷(rollup)、最新的rld卷和最新的libc卷。

很明确，对于pthreads支持，需要本页上所有的POSIX补丁：

http://support.sgi.com/surfzone/patches/patchset/6.2_posix.rps.html

如果在编译“mysql.cc”时，遇到类似于下面的错误：

```
"/usr/include/curses.h", line 82: error(1084):
invalid combinati在of type
```

那么应在**MySQL**源码树的顶级目录输入下列命令：

```
extra/replace bool curses_bool < /usr/include/curses.h > include/curses.h
make
```

应该也有安排上的问题报告。如果只有一个线程正在运行，事情会变慢的。通过启动另外一个客户端来避免它，这可以导致此后其它线程的执行速度增加2到10倍。这是Irix线程难以理解的问题；可能必须临时准备找出解决方案直到它能被修正。

如果你正在用gcc编译，可以使用下列**configure**命令：

```
CC=gcc CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql --enable-thread-safe-client \
    --with-named-thread-libs=-lpthread
```

在用原生Irix C和C++编译器7.3.1.2的Irix 6.5.11上,下面的项工作：

```
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
```

2.12.5.8. SCO UNIX和OpenServer 5.0.x版注意事项

当前的移植仅在"sco3.2v5.0.5","sco3.2v5.0.6"和"sco3.2v5.0.7"系统上进行了测试，在"sco 3.2v4.2"上的移植也有很大进展。OpenServer 5.0.8(Legend)具有原生线程，允许文件大于2GB。目前最大的文件可以达到2GB。

我们可以在OpenServer上用**gcc** 2.95.3用下面的**configure**命令编译MySQL。

```
CC=gcc CXX=gcc ./configure --prefix=/usr/local/mysql \
    --enable-thread-safe-client --with-innodb \
    --with-openssl --with-vio --with-extra-charsets=complex
```

从ftp://ftp.sco.com/pub/openserver5/opensrc/gnutools-5.0.7Kj可以得到**gcc**。

该开发系统需要在OpenServer 5.0.6O的penServer Execution Environment Supplement oss646B和oss656B，OpenSource库位于 gwxlibs。所有OpenSource工具位于opensrc目录。可以从ftp://ftp.sco.com/pub/openserver5/opensrc/获得。

我们建议使用最新的MySQL产品发布。

SCO提供的操作系统补丁：OpenServer 5.0.[0-6]：ftp://ftp.sco.com/pub/openserver5，OpenServer 5.0.7：ftp://ftp.sco.com/pub/openserverv5/507。

SCO提供的关于安全修复的信息：Server 5.0.x：ftp://ftp.sco.com/pub/security/OpenServer。

OpenSever 5.0.x系统上最大的文件的大小为2GB。

在OpenServer 5.0.x上，可供串缓冲区、clist和锁定记录分配的总内存不能超出60MB。

串缓冲区的分配单位为4096 字节的页，clists为70字节，锁定记录为64字节，因此为：

(NSTRPAGES * 4096) + (NCLIST * 70) + (MAX_FLCKREC * 64) <= 62914560

按照以下步骤来配置Database Services选项。如果你不确定应用程序是否需要，参见随应用程序提供的文档。

1. 用root登录。
2. 编辑/etc/conf/sdevice.d/suds文件启用SUDS驱动程序。将第2个域内的N更改为Y。
3. 使用mkdev aio或Hardware/Kernel Manager来启用对asynchronous I/O的支持，并重新连接内核。要想让用户锁定内存用于该类I/O，更新aiomemlock(F)文件。应当对该文件进行更新，包括进可以使用AIO的用户和可以锁定的最大数量的内存。
4. 许多应用程序使用setuid二进制，因此你只能指定单个用户 。请参见随应用程序提供的文档来看是否应用程序是这种情况。

完成该进程后，重新启动系统，创建包括这些更改的新内核。

默认情况，/etc/conf/cf.d/mtune中的条目设置为：

```
Value           Default         Min             Max
```

```
-----           -------         ---             ---
NBUF            0               24              450000
NHBUF           0               32              524288
NMPBUF          0               12              512
MAX_INODE       0               100             64000
MAX_FILE        0               100             64000
CTBUFSIZE       128             0               256
MAX_PROC        0               50              16000
MAX_REGION      0               500             160000
NCLIST          170             120             16640
MAXUP           100             15              16000
NOFILES         110             60              11000
NHINODE         128             64              8192
NAUTOUP         10              0               60
NGROUPS         8               0               128
BDFLUSHR        30              1               300
MAX_FLCKREC     0               50              16000
PUTBUFSZ        8000            2000            20000
MAXSLICE        100             25              100
ULIMIT          4194303         2048            4194303
* Streams Parameters
NSTREAM         64              1               32768
NSTRPUSH        9               9               9
NMUXLINK        192             1               4096
STRMSGSZ        16384           4096            524288
STRCTLSZ        1024            1024            1024
STRMAXBLK       524288          4096            524288
NSTRPAGES       500             0               8000
STRSPLITFRAC    80              50              100
NLOG            3               3               3
NUMSP           64              1               256
NUMTIM          16              1               8192
NUMTRW          16              1               8192
* Semaphore Parameters
SEMMAP          10              10              8192
SEMMNI          10              10              8192
SEMMNS          60              60              8192
SEMMNU          30              10              8192
SEMMSL          25              25              150
SEMOPM          10              10              1024
SEMUME          10              10              25
SEMVMX          32767           32767           32767
SEMAEM          16384           16384           16384
* Shared Memory Parameters
SHMMAX          524288          131072          2147483647
SHMMIN          1               1               1
```

```
SHMMNI          100             100             2000
FILE            0               100             64000
NMOUNT          0               4               256
NPROC           0               50              16000
NREGION         0               500             160000
```

我们建议将这些值设置为:

NOFILES应当为4096或2048。

MAXUP应当为2048。

要想更改内核，进入/etc/conf/bin并使用**./idtune** *name parameter*来更改。例如，要将SEMMS改为200，用root账户执行命令:

```
# cd /etc/conf/bin
# ./idtune SEMMNS 200
```

我们建议对该系统进行调节，但是使用的正确参数值取决于访问应用程序或数据库的用户数目和数据库空间(也就是使用的缓冲池)。下面的项影响/etc/conf/cf.d/stune中定义的内核参数:

SHMMAX(推荐设定值：128MB)和SHMSEG(推荐设定值：15)。这些参数影响创建用户缓冲池的MySQL数据库引擎。

NOFILES和MAXUP至少应达到2048。

MAXPROC至少应当设置到3000/4000(取决于用户数)或更大值。

建议使用下面的公式来计算SEMMSL、SEMMNS和SEMMNU的值:

```
SEMMSL = 13
```

发现13最适合进程和MySQL。

SEMMNS = SEMMSL * 系统上运行的db服务器数目。

将SEMMNS设置为SEMMSL值乘以你在该系统上一次运行的db服务器的(最大)数目。

```
SEMMNU = SEMMNS
```

将SEMMNU值设置为等于SEMMNS值。可以将该值设置为75%的SEMMNS，但是这是一个保守的估计。

至少需要安装"SCO OpenServer Linker and Application Development Libraries"或OpenServer开发系统来使用**gcc**。你不能只使用GCC Dev系统，而不安装它们。

你应当获得FSU PTHREADS安装软件包并先安装它。可以从http://moss.csc.ncsu.edu/~mueller/ftp/pub/PART/pthreads.tar.gz获得。你还可以从 ftp://ftp.zenez.com/pub/zenez/prgms/FSU-threads-3.14.tar.gz获得预编译的安装软件包。

FSU Pthreads能用带tcpip的SCO UNIX 4.2编译，或使用OpenServer 3.0或Open Desktop 3.0(OS 3.0 ODT 3.0)，安装带有使用一个GCC 2.5.X的SCO开发系统。对ODT或OS 3.0，将需要一个GCC 2.5.x的良好移植。没有一个良好的移植会有很多问题。对这个产品的移植需要SCO UNIX开发系统，没有它，缺少所需的库和链接器。还需要SCO-3.2v4.2-includes.tar.gz。该文件包含SCO 开发include 文件的更改这是构建MySQL所需要的。需要用这些修改的头文件替换已有系统的include文件。可以从 ftp://ftp.zenez.com/pub/zenez/prgms/SCO-3.2v4.2-includes.tar.gz获得。

为了在系统上构造FSU Pthreads，需要做的工作是运行GNU **make**。启动FSU-threads-3.14.tar.gz中的Makefile生成FSU-线程。

在“`thread/src`”目录下运行**./configure**并且选择SCO OpenServer选项。这个命令拷贝“`Makefile.SCO5`”到“`Makefile`”。然后运行**make**。

为了在默认的“`/usr/include`”目录安装，作为root登录，然后cd 到“`thread/src`”目录，并运行**make install**。

记得在制作**MySQL**时要使用GNU **make**。

注释：如果你不是作为root启动**safe_mysqld**，将可能每进程只有 默认的110个打开的文件。**mysqld**将在日志文件写下关于此的注解。

用SCO 3.2V4.2时,应当使用FSU PTHREADS 3.14或更新版。以下**configure**命令应当工作:

```
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"
```

你可能遇到include文件的某些问题。在这种情况下，你能在ftp://www.mysql.com/pub/mysql/Downloads/SCO/SCO-3.2v4.2-includes.tar.gz找到新的SCO特定的include文件。你应该在**MySQL**源码树的“`include`”目录下打开这个文件。

SCO开发注意事项:

- **MySQL**应该自动地检测FSU Pthreads并且用-lgthreads -lsocket -lgthreads选项链接**mysqld**。
- SCO开发库在FSU Pthreads是重入(reentrant)的。SCO宣称它的库函数是重入的，因此他们一定在FSU Pthreads中是重入的。在 OpenServer上的 FSU Pthreads 试图使用SCO方案制作重入的库。
- FSU Pthreads(至少在ftp::/ftp.zenez.com的版本)链接了GNU malloc，如果你遇到内存使用的问题，确定“`gmalloc.o`”包含在“`libgthreads.a`”和“`libgthreads.so`”中。

· 在FSU Pthreads中，下列系统调用是pthreads感知的：read()、write()、getmsg()、connect()、accept()、select()和wait()。

· CSSA-2001-SCO.35.2(补丁列为定制erg711905-dscr_remap安全补丁(版本 2.0.0))中断FSU线程并使**mysqld**不稳定。如果你想在 OpenServer 5.0.6机器上运行**mysqld**必须删除它。

· 如果你使用SCO OpenServer 5,可能需要用-DDRAFT7 in CFLAGS重新编译FSU pthreads。否则，**mysqld**启动时InnoDB会挂起。

· SCO在ftp://ftp.sco.com/pub/openserver5提供了OpenServer 5.0.x的操作系统补丁。

· SCO在ftp://ftp.sco.com/pub/security/sse和ftp://ftp.sco.com/pub/security/sse提供了OpenServer 5.0.x的安全修复和libsocket.so.2。

· Pre-OSR506安全修复。在ftp://stage.caldera.com/pub/security/openserver/或ftp://stage.caldera.com/pub/security/openserver/CSSA-2001-SCO.10/提供了libsocket.so.2和libresolv.so.1的telnetd修复，以及在pre-OSR506系统上的安装说明。

最好在编译/使用MySQL之前安装这些补丁。

Legend/ OpenServer 6.0.0有原生线程，没有2GB文件大小限制。

2.12.5.9. SCO UnixWare 7.1.x和OpenUNIX 8.0.0版注意事项

我们建议使用最新的MySQL产品发布。

我们可以用下面的**configure**命令在UnixWare 7.1.x中编译MySQL:

```
CC="cc" CFLAGS="-I/usr/local/include" \
CXX="CC" CXXFLAGS="-I/usr/local/include" \
./configure --prefix=/usr/local/mysql \
    --enable-thread-safe-client --with-berkeley-db=./bdb \
    --with-innodb --with-openssl --with-extra-charsets=complex
```

如果你想要使用**gcc**，必须使用**gcc** 2.95.3或更新版。

```
CC=gcc CXX=g++ ./configure --prefix=/usr/local/mysql
```

当构建MySQL时不使用UnixWare 7.1.4或OpenServer 6.0.0中的Berkeley DB版本。MySQL使用自己的Berkeley DB。用来构建静态和动态库的**configure**命令在*src_directory*/bdb/build_unix/中，但是它不在MySQL自己的BDB版本中。工作区如下所示。

1. 按常规配置MySQL。
2. 进入（cd）bdb/build_unix/
3. 将 -p Makefile复制(cp)到Makefile.sav
4. 使用同样的选项并运行**../dist/configure**。
5. 运行**gmake**。
6. cp -p Makefile.sav Makefile
7. 进入顶级源目录并运行**gmake**。

允许生成共享和动态库并工作。

SCO在ftp://ftp.sco.com/pub/unixware7提供了UnixWare 7.1.1的操作系统补丁,在ftp://ftp.sco.com/pub/unixware7/713/ 提供了UnixWare 7.1.3的操作系统补丁, UnixWare 7.1.4的见ftp://ftp.sco.com/pub/unixware7/714/, OpenUNIX 8.0.0的见 ftp://ftp.sco.com/pub/openunix8 。

SCO提供的安全修复相关信息：OpenUNIX见ftp://ftp.sco.com/pub/security/OpenUNIX，UnixWare见 ftp://ftp.sco.com/pub/security/UnixWare。

默认情况，在UnixWare 7.1.1系统中最大文件的大小为1GB，但是UnixWare 7.1.4中文件大小限制是1 TB（用VXFS）。一些OS实用程序 的限制是2GB。在UnixWare 7中可能的最大文件大小是1TB（用VXFS）。

在UnixWare 7.1.4中，你不需要采取措施来获得大文件支持，但在UnixWare 7.1.x的以前版本中需要启用大文件支持，可以运行fsadm。

```
# fsadm -Fvxfs -o largefiles /
# fsadm /         * Note
# ulimit unlimited
# cd /etc/conf/bin
# ./idtune SFSZLIM 0x7FFFFFFF     ** Note
# ./idtune HFSZLIM 0x7FFFFFFF     ** Note
# ./idbuild -B

* This should report "largefiles".
** 0x7FFFFFFF represents infinity for these values.
```

用shutdown重启系统。

默认情况，/etc/conf/cf.d/mtune中的条目设置为：

```
Value           Default         Min             Max
-----           -------         ---             ---
SVMMLIM         0x9000000       0x1000000       0x7FFFFFFF
HVMMLIM         0x9000000       0x1000000       0x7FFFFFFF
SSTKLIM         0x1000000       0x2000          0x7FFFFFFF
HSTKLIM         0x1000000       0x2000          0x7FFFFFFF
```

我们建议将这些值设置为：

```
SDATLIM 0x7FFFFFFF
HDATLIM 0x7FFFFFFF
SSTKLIM 0x7FFFFFFF
HSTKLIM 0x7FFFFFFF
SVMMLIM 0x7FFFFFFF
HVMMLIM 0x7FFFFFFF
SFNOLIM 2048
HFNOLIM 2048
```

我们建议对该系统进行调节，但是使用的正确参数值取决于访问应用程序或数据库的用户数目和数据库空间(也就是使用的缓冲池)。下面的项影响/etc/conf/cf.d/stune中定义的内核参数：

SHMMAX(推荐设定值：128MB)和SHMSEG(推荐设定值：15)。这些参数影响创建用户缓冲池的MySQL数据库引擎。

SFNOLIM和HFNOLIM至少应达到2048。

NPROC至少应当设置到3000/4000(取决于用户数)。

建议使用下面的公式来计算SEMMSL、SEMMNS和SEMMNU的值：

```
SEMMSL = 13
```

据发现13最适合进程和MySQL。

SEMMNS = SEMMSL * 系统上运行的db服务器数目。

将SEMMNS设置为SEMMSL值乘以你在该系统上一次运行的db服务器的(最大)数目。

SEMMNU = SEMMNS

将SEMMNU值设置为等于SEMMNS值。可以将该值设置为75%的SEMMNS，但是这是一个保守的估计。

2.12.5.10. SCO OpenServer 6.0.x版注意事项

OpenServer6的主要改进包括：

- 大文件支持，可达到1 TB。
- 多处理器支持，从4个处理器增加到32个处理器。
- 增加的内存支持，可达到64GB。
- 将UnixWare的功能扩展到OpenServer6。
- 性能的改进

OpenServer6.0.0具有：

- /bin，用于与OpenServer5.0.x相同行为的命令。
- /u95/bin用于标准的命令，例如大文件系统(LFS)支持。
- /udk/bin用于UnixWare 7.1.4中相同的命令，默认是LFS支持。

下面是在OpenServer6上设置PATH的指导。如果用户想使用传统的OpenServer5.0.x，则路径应当首先为/bin。如果用户需要LFS支持，则 路径应当为/u95/bin：/bin。如果用户首选UnixWare 7支持，则路径应为/udk/bin：/u95/bin：/bin：。

我们建议使用最新的MySQL产品发布。

我们可以在OpenServer 6.0.x用下面的**configure**命令编译MySQL：

```
CC="cc" CFLAGS="-I/usr/local/include" \
CXX="CC" CXXFLAGS="-I/usr/local/include" \
./configure --prefix=/usr/local/mysql \
    --enable-thread-safe-client --with-berkeley-db=./bdb \
    --with-innodb --with-openssl --with-extra-charsets=complex \
    --enable-readline
```

如果你想要使用**gcc**，必须使用**gcc**2.95.3或更新版。

```
CC=gcc CXX=g++ ./configure --prefix=/usr/local/mysql
```

当构建MySQL时不使用UnixWare 7.1.4或OpenServer 6.0.0中的Berkeley DB版本。MySQL使用自己的Berkeley DB。用来构建静态和动态库的**configure**命令在*src_directory*/bdb/build_unix/中，但是它不在MySQL自己的BDB版本中。工作区如下所示。

1. 按常规配置MySQL。
2. 进入（cd）bdb/build_unix/
3. 将 -p Makefile复制(cp)到Makefile.sav
4. 使用同样的选项并运行**../dist/configure**。
5. 运行**gmake**。
6. cp -p Makefile.sav Makefile
7. 进入顶级源目录并运行**gmake**。

允许生成共享和动态库并工作。OpenServer6.0.0 还需要MySQL源码树的补丁和适用于bdb/dist/config.guess的config.guess的补丁。你可以从ftp://ftp.zenez.com/pub/zenez/prgms/mysql-4.1.12-osr6-patches.tar.gz和ftp://ftp.zenez.com/pub/zenez/prgms/mysql-4.x.x-osr6-patches下载补丁。有一个README文件可以提供帮助。

SCO在ftp://ftp.sco.com/pub/openserver6提供了OpenServer6的操作系统补丁。

SCO在ftp://ftp.sco.com/pub/security/OpenServer提供关于安全修复的信息。

默认情况，在OpenServer6.0.0系统的最大文件大小为1TB。一些操作系统工具的限制是2GB。在UnixWare 7中最大文件大小可以是1TB（用 VXFS或HTFS）。

默认情况，/etc/conf/cf.d/mtune中的条目设置为：

```
Value           Default         Min             Max
-----           -------         ---             ---
SVMMLIM         0x9000000       0x1000000       0x7FFFFFFF
HVMMLIM         0x9000000       0x1000000       0x7FFFFFFF
SSTKLIM         0x1000000       0x2000          0x7FFFFFFF
HSTKLIM         0x1000000       0x2000          0x7FFFFFFF
```

我们建议将这些值设置为：

```
SDATLIM 0x7FFFFFFF
HDATLIM 0x7FFFFFFF
SSTKLIM 0x7FFFFFFF
HSTKLIM 0x7FFFFFFF
SVMMLIM 0x7FFFFFFF
HVMMLIM 0x7FFFFFFF
SFNOLIM 2048
HFNOLIM 2048
```

我们建议对该系统进行调节，但是使用的正确参数值取决于访问应用程序或数据库的用户数目和数据库空间(也就是使用的缓冲池)。下面的项影响/etc/conf/cf.d/stune中定义的内核参数：

SHMMAX(推荐设定值：128MB)和SHMSEG(推荐设定值：15)。这些参数影响创建用户缓冲池的MySQL数据库引擎。

SFNOLIM和HFNOLIM至少应达到2048。

NPROC至少应当设置到3000/4000(取决于用户数)。

建议使用下面的公式来计算SEMMSL、SEMMNS和SEMMNU的值：

```
SEMMSL = 13
```

发现13最适合进程和MySQL。

SEMMNS = SEMMSL * 系统上运行的db服务器数目。

将SEMMNS设置为SEMMSL值乘以你在该系统上一次运行的db服务器的(最大)数目。

SEMMNU = SEMMNS

将SEMMNU值设置为等于SEMMNS值。可以将该值设置为75%的SEMMNS，但是这是一个保守的估计。

2.12.6. OS/2注意事项

MySQL使用太多的打开文件。因此，应该增加类似下面的东西到“CONFIG.SYS”文件：

```
SET EMXOPT=-c -n -h1024
```

如果不这样做，将可能碰到下列错误：

```
File 'xxxx' not found (Errcode: 24)`
```

当OS/2 Warp 3上使用**MySQL**时，要求FixPack 29或以上版本。用OS/2 Wrap 4，要求FixPack 4或以上版本。这是Pthreads库的一个要求。**MySQL**必须安装在支持长文件名的一个分区上，如HPFS、FAT32等等。

“INSTALL.CMD”脚本必须从OS/2自己的“CMD.EXE”运行并且不能工作在替代外壳如“4OS2.EXE”。

"scripts/mysql-install-db"脚本已经被重新命名了：它称为"install.cmd"并且是一个REXX脚本，将安装 默认的**MySQL**安全配置并为**MySQL**创建WorkPlace Shell图标。

动态模块支持通过编译但没有充分测试，动态模块应该使用Pthreads运行时刻库编译。

```
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
```

注释：由于OS/2的限制，UDF模块名字的词干不能超过8个字符。模块存储在"/mysql2/udf"目录；safe-mysqld.cmd脚本将把这个目录放在BEGINLIBPATH环境变量中。当使用UDF模块时，指定的扩展名被忽略--它被假定为".udf"。例如，在 Unix上，共享模块可能被命名为"example.so"并且可以象这样从它装载函数：

mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'example.so';

在OS/2中, 模块被命名为example.udf，但是不能指定模块扩展名：

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'example';
```

2.13. Perl安装注意事项

2.13.1. 在Unix中安装Perl
2.13.2. 在Windows下安装ActiveState Perl
2.13.3. 使用Perl DBI/DBD接口的问题

Perl对**MySQL**的支持借助DBI/DBD客户端接口。接口要求Perl 5.6.0或以后版本。如果你的Perl版本更旧，接口将不能工作。

如果想要在Perl DBI中使用事务，需要有1.2216版或更新版DBD：：mysql。建议使用2.9003版或更新版。

如果你使用MySQL 4.1客户端库，必须使用DBD：：mysql 2.9003或更新版。

MySQL分发版不包含Perl支持。可以从http://search.cpan.org获得Unix需要的模块，或在Windows中使用ActiveState **ppm**程序。以下章节描述了如何做。

如果你想要运行MySQL基准脚本，必须安装Perl支持。请参见7.1.4节，"MySQL基准套件"。

2.13.1. 在Unix中安装Perl

MySQL Perl支持也要求你安装了**MySQL**客户端编程支持(库和头文件)。大多数安装中安装了必要的文件。但是，如果你在Linux中从RPM文件安装MySQL，确保安装了开发者RPM。客户端程序在客户端RPM中，但是客户端编程支持在开发者RPM。

如果你想要安装Perl支持，需要的文件能从http://search.cpan.org的CPAN (Comprehensive Perl Archive Network)获得。

在Unix中安装Perl模块最简单的方法是使用CPAN模块。例如：

```
shell> perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql
```

DBD：：mysql安装运行许多测试。测试中使用默认用户名和密码尝试连接本地MySQL服务器。(默认用户名是Unix中的登录名和Windows中的ODBC。默认密码"没有密码。") 如果你不能通过这些值连接服务器 (例如，如果你的账户有密码)，测试失败。可以使用force install DBD::mysql忽略失败的测试。

DBI需要Data::Dumper模块。应该可以安装；如果不能，应当在安装DBI前安装Data::Dumper模块。

还可以下载**tar**文件压缩格式的模块分发版，并手动构建模块。例如，解压缩和构建DBI分发版的过程为：

1. 将分发版解压缩到当前目录：

2.
```
shell> gunzip < DBI-VERSION.tar.gz | tar xvf -
```

该命令创建目录 DBI-*VERSION*。

3. 进入解压缩分发版的顶级目录：

4.
```
shell> cd DBI-VERSION
```

5. 构建分发版并编译:

```
6.          shell> perl Makefile.PL
7.          shell> make
8.          shell> make test
9.          shell> make install
```

make test命令很重要，因为它验证模块正在工作。注意：在你DBD::mysql安装期间运行该命令试验接口代码时，**MySQL**服务器必须正在运行，否则测试将失败。

无论何时你安装一个新版本的**MySQL**，重建并且重新安装DBD::mysql分发版是一个好主意, 特别是在升级**MySQL**后，观察所有异常，如DBI脚本失败。

如果你没有权限在系统目录下安装Perl模块，或如果你安装本地Perl模块，下列参考书可以帮助你：http://servers.digitaldaze.com/extensions/perl/modules.html#modules

在标题"Installing New Modules that Require Locally Installed Modules"下面查找。

2.13.2. 在**Windows**下安装**ActiveState Perl**

在Windows中，应当按照以下步骤安装带ActiveState Perl的MySQL DBD模块:

- 从http://www.activestate.com/Products/ActivePerl/获得ActiveState Perl并安装。
- 打开控制台窗口("DOS窗口)。
- 如果需要，设置HTTP_proxy变量。例如，你可以试试:
-
```
set HTTP_proxy=my.proxy.com:3128
```
- 启动PPM程序:
-
```
C: \> C: \perl\bin\ppm.pl
```
- 如果你还没有安装DBI，则安装:
-
```
ppm> install DBI
```
- 如果成功，运行下面的命令:
-
```
install\
ftp://ftp.de.uu.net/pub/CPAN/authors/id/JWIED/DBD-mysql-1.2212.x86.ppd
```

该进程应当在ActiveState Perl 5.6或更新版本中工作。

如果你不能使上述步骤工作，应该安装**MyODBC**驱动程序并且通过ODBC连接**MySQL**服务器:

```
use DBI;
$dbh= DBI->connect("DBI:ODBC:$dsn",$user,$password) ||
die "Got error $DBI::errstr when connecting to $dsn\n";
```

2.13.3. 使用**Perl DBI/DBD**接口的问题

如果Perl报告它不能发现../mysql/mysql.so模块，可能试因为Perl不能定位共享库libmysqlclient.so。

可以使用下面的方法修复该问题:

- 用perl Makefile.PL -static -config而不要用perl Makefile.PL编译DBD：：mysql分发版。
- 将libmysqlclient.so复制到其它共享库所在的目录 (可能为/usr/lib 或/lib)。
- 修改用来编译DBD：：mysql的-L选项，反应libmysqlclient.so的实际位置。
- 在Linux中，你可以将libmysqlclient.so的目录路径名增加到/etc/ld.so.conf文件。

- 将libmysqlclient.so的目录路径名增加到LD_RUN_PATH环境变量。一些系统使用LD_LIBRARY_PATH。

注意如果连接器没有找到另一个库，你还可能需要修改-L选项。例如，如果因为libc在/lib下并且连接命令指定了-L/usr/lib，连接器不能找到它，将-L选项更改为-L/lib或在连接命令中增加-L/lib。

如果你遇到下面的DBD：：mysql错误，你可能使用了**gcc**(或使用了用**gcc**编译的旧的二进制)：

```
/usr/bin/perl: can't resolve symbol '__moddi3'
/usr/bin/perl: can't resolve symbol '__divdi3'
```

当构建mysql.so库是在连接命令中增加-L/usr/lib/gcc-lib/... -lgcc(当编译Perl客户端时检查生成mysql.so的**make**的输出)。-L选项应指定libgcc.a在系统中的安装目录的路径名。

该问题的另一个原因是Perl和MySQL没有用**gcc**编译。在这种情况下，你可以用**gcc**编译二者来解决。

当你运行测试时你会从DBD：：mysql看见下面的错误：

```
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.
```

这意味着你需要在连接行包括-lz 压缩库。可以通过更改文件lib/DBD/mysql/Install.pm的下列行来实现：

```
$sysliblist .= " -lm";
```

更改为：

```
$sysliblist .= " -lm -lz";
```

然后，你必须运行**make realclean** 然后重新安装。

如果你想要在SCO上安装DBI，需要在DBI-*xxx* 和每个子目录中编辑Makefile。请注意下面的更新假定使用**gcc** 2.95.2或更新版：

```
旧:                                  新:
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

旧:
CCCFLAGS = -belf -dy -w0 -U M_XENIX -DPERL_SCO5 -I/usr/local/include

新:
CCFLAGS = -U M_XENIX -DPERL_SCO5 -I/usr/local/include
```

这些更改是必需的，因为用**icc**或**cc**编译的Perl动态装载器不装载DBI模块。

如果你想要在不支持动态链接(例如 SCO)的系统中使用Perl模块，可以生成一个包括DBI和DBD：：mysql的静态版本的Perl。方法是生成一个用DBI代码连接的Perl版本，并将它安装到当前Perl的顶级目录。然后构建一个含DBD代码连接的Perl版本并安装。

在SCO中，必须将下面的环境变量设置为：

```
LD_LIBRARY_PATH=/lib:/usr/lib:/usr/local/lib:/usr/progressive/lib
```

Or：

```
LD_LIBRARY_PATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:\
    /usr/progressive/lib:/usr/skunk/lib
LIBPATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:\
    /usr/progressive/lib:/usr/skunk/lib
MANPATH=scohelp:/usr/man:/usr/local1/man:/usr/local/man:\
    /usr/skunk/man:
```

首先，在DBD分发版的安装目录运行下述命令来创建一个包括静态连接的DBD模块的Perl：

```
shell> perl Makefile.PL -static -config
shell> make
shell> make install
shell> make perl
```

然后必须安装新的Perl。**make perl**的输出表示安装时需要执行的**make**命令。在SCO中，为**make -f Makefile.aperl inst_perl MAP_TARGET=perl**。

然后，使用刚刚创建的Perl，在DBD：：mysql分发版的安装目录运行下述命令来创建另一个Perl，应包括静态连接的DBD：：mysql：

```
shell> perl Makefile.PL -static -config
shell> make
shell> make install
shell> make perl
```

最后，应当安装新的Perl。**make perl**的输出指示了使用的命令。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com.。原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第3章：教程

目录

3.1. 连接与断开服务器
3.2. 输入查询
3.3. 创建并使用数据库

3.3.1. 创建并选择数据库
3.3.2. 创建表
3.3.3. 将数据装入表中
3.3.4. 从表检索信息

3.4. 获得数据库和表的信息
3.5. 在批处理模式下使用**my**sql
3.6. 常用查询的例子

3.6.1. 列的最大值
3.6.2. 拥有某个列的最大值的行
3.6.3. 列的最大值：按组
3.6.4. 拥有某个字段的组间最大值的行
3.6.5. 使用用户变量
3.6.6. 使用外键
3.6.7. 根据两个键搜索
3.6.8. 根据天计算访问量
3.6.9. 使用AUTO_INCREMENT

3.7. 孪生项目的查询

3.7.1. 查找所有未分发的孪生项
3.7.2. 显示孪生对状态的表

3.8. 与Apache一起使用MySQL

本章通过演示如何使用**mysql**客户程序创造和使用一个简单的数据库，提供一个MySQL的入门教程。**mysql**（有时称为"终端监视器"或只是"监视"）是一个交互式程序，允许你连接一个MySQL服务器，运行查询并察看结果。**mysql**可以用于批模式：你预先把查询放在一个文件中，然后告诉**mysql**执行文件的内容。本章将介绍使用**mysql**的两个方法。

要想查看由**mysql**提供的选择项目表，可以用--help选项来调用：

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

本章假定**mysql**已经被安装在你的机器上，并且有一个MySQL服务器可以连接。否则，请联络MySQ**L**管理员。（如果你是管理员，则需要查阅本手册的其它章节，例如第5章：数据库管理。）

本章描述建立和使用一个数据库的全过程。如果你仅仅对访问一个已经存在的数据库感兴趣，可以跳过描述怎样创建数据库及它所包含的表的章节。

由于本章是一个教程，省略了许多细节。关于这里所涉及的主题的详细信息，请查阅本手册的相关章节。

3.1. 连接与断开服务器

为了连接服务器，当调用**mysql**时，通常需要提供一个MySQL用户名并且很可能需要一个 密码。如果服务器运行在登录服务器之外的其它机器上，还需要指定主机名。联系管理员以找出进行连接所使用的参数 (即，连接的主机、用户名和使用的密码)。知道正确的参数后，可以按照以下方式进行连接:

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

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

host和user分别代表MySQL服务器运行的主机名和MySQL账户用户名。设置时替换为正确的值。******** 代表你的密码；当**mysql**显示Enter password:提示时输入它。

如果有效，你应该看见mysql>提示符后的一些介绍信息:

```
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: 5.1.2-alpha-standard


Type 'help;' or '\h' for help. Type '\c' to clear the buffer.


mysql>
```

mysql> 提示符告诉你mysql准备为你输入命令。

一些**MySQL**安装允许用户以匿名（未命名）用户连接到本地主机上运行的服务器。如果你的机器是这种情况，你应该能不带任何选项地调用**mysql**与该服务器连接:

```
shell> mysql
```

成功地连接后，可以在mysql>提示下输入QUIT (或\q)随时退出:

```
mysql> QUIT

Bye
```

在Unix中，也可以按control-D键断开服务器。

在下列章节的大多数例子都假设你连接到了服务器。由mysql>提示指明。

3.2. 输入查询

确保你连接上了服务器，如在先前的章节讨论的。连接上服务器并布代表选择了任何数据库，但这样就可以了。知道关于如何查询的基本知识，比马上跳至创建表、给他们装载数据并且从他们检索数据更重要。本节描述输入命令的基本原则，使用几个查询，你能尝试了解**mysql**是如何工作的。

这是一个简单的命令，要求服务器告诉它的版本号和当前日期。在mysql>提示输入如下命令并按回车键:

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

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

| VERSION()       | CURRENT_DATE |

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

| 5.1.2-alpha-log | 2005-10-11   |
```

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

1 row in set (0.01 sec)

mysql>
```

这询问说明**mysql**的几个方面:

· 一个命令通常由SQL语句组成，随后跟着一个分号。（有一些例外不需要分号。早先提到的QUIT是一个例子。后面我们将看到其它的例子。）

· 当发出一个命令时，**mysql**将它发送给服务器并显示执行结果，然后显示另一个mysql>显示它准备好接受其它命令。

· **mysql**用表格(行和列)方式显示查询输出。第一行包含列的标签，随后的行是查询结果。通常，列标签是你取自数据库表的列的名字。如果你正在检索一个表达式而非表列的值(如刚才的例子)，**mysql**用表达式本身标记列。

· **mysql**显示返回了多少行，以及查询花了多长时间，它给你提供服务器性能的一个大致概念。因为他们表示时钟时间(不是 CPU 或机器时间)，并且因为他们受到诸如服务器负载和网络延时的影响，因此这些值是不精确的。（为了简洁，在本章其它例子中不再显示"集合中的行"。）

能够以大小写输入关键词。下列查询是等价的:

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

mysql> select version(), current_date;

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

这是另外一个查询，它说明你能将**mysql**用作一个简单的计算器:

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

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

| SIN(PI()/4)      | (4+1)*5 |

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

| 0.70710678118655 |      25 |

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

1 row in set (0.02 sec)
```

至此显示的命令是相当短的单行语句。你可以在一行上输入多条语句，只需要以一个分号间隔开各语句:

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

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

| VERSION()       |

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

| 5.1.2-alpha-log |

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

1 row in set (0.00 sec)


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

| NOW()               |
```

```
+---------------------+
| 2005-10-11 15:15:00 |
+---------------------+
1 row in set (0.00 sec)
```

不必全在一个行内给出一个命令，较长命令可以输入到多个行中。**mysql**通过寻找终止分号而不是输入行的结束来决定语句在哪儿结束。（换句话说，**mysql**接受自由格式的输入：它收集输入行但直到看见分号才执行。）

这里是一个简单的多行语句的例子：

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+---------------+--------------+
| USER()        | CURRENT_DATE |
+---------------+--------------+
| jon@localhost | 2005-10-11   |
+---------------+--------------+
```

在这个例子中，在输入多行查询的第一行后，要注意提示符如何从mysql>变为->，这正是**mysql**如何指出它没见到完整的语句并且正在等待剩余的部分。提示符是你的朋友，因为它提供有价值的反馈，如果使用该反馈，将总是知道**mysql**正在等待什么。

如果你决定不想执行正在输入过程中的一个命令，输入\c取消它：

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

这里也要注意提示符，在你输入\c以后，它切换回到mysql>，提供反馈以表明**mysql**准备接受一个新命令。

下表显示出可以看见的各个提示符并简述它们所表示的**mysql**的状态：

提示符	含义
mysql>	准备好接受新命令。
->	等待多行命令的下一行。
'>	等待下一行，等待以单引号("'")开始的字符串的结束。
">	等待下一行，等待以双引号("")开始的字符串的结束。
`>	等待下一行，等待以反斜点(``)开始的识别符的结束。
/*>	等待下一行，等待以/*开始的注释的结束。

当你打算在一个单行上发出一个命令时，通常会"偶然"出现多行语句，但是没有终止分号。在这种情况中，**mysql**等待进一步输入：

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

```
    ->
```

如果出现这种情况(你认为输完了语句，但是只有一个->提示符响应)，很可能**mysql**正在等待分号。如果你没有注意到提示符的提示，在意识到你需要做什么之前，你可能会呆坐一会儿。输入一个分号完成语句，**mysql**将执行：

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

在字符串收集期间将出现 '> 和 "> 提示符（提示MySQL正等待字符串的结束）。在MySQL中，可以写由'''或'"'字符括起来的字符串 (例如，'hello'或"goodbye")，并且**mysql**允许输入跨越多行的字符串。当看到一个 '> 或 "> 提示符时，这意味着已经输入了包含以'''或'"'括号字符开始的字符串的一行，但是还没有输入终止字符串的匹配引号。这显示你粗心地省掉了一个引号字符。例如：

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

如果你输入SELECT语句，然后按**Enter**（回车）键并等待结果，什么都没有出现。不要惊讶，"为什么该查询这么长呢？"，注意">提示符提供的线索。它告诉你**mysql**期望见到一个未终止字符串的余下部分。（你看见语句中的错误吗？字符串"Smith丢掉了第二个引号。）

走到这一步，你该做什么？最简单的是取消命令。然而，在这种情况下，你不能只是输入\c，因为**mysql**作为它正在收集的字符串的一部分来解释它！相反，应输入关闭的引号字符(这样**mysql**知道你完成了字符串)，然后输入\c：

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

提示符回到mysql>，显示**mysql**准备好接受一个新命令了。

`> 提示符类似于 '> 和"> 提示符，但表示你已经开始但没有结束以`> 开始的识别符。

知道'>和">提示符的含义很重要，因为如果你错误地输入一个未终止的字符串，任何后面输入的行将要被**mysql**忽略--包括包含QUIT的行！这可能令人相当困惑，特别是如果取消当前命令前还不知道你需要提供终止引号。

3.3. 创建并使用数据库

3.3.1. 创建并选择数据库
3.3.2. 创建表
3.3.3. 将数据装入表中
3.3.4. 从表检索信息

知道怎样输入命令，便可以访问数据库了。

假定在你的家(你的"动物园")中有很多宠物，并且你想跟踪关于它们各种类型的信息。你可以通过创建表来保存你的数据并根据所需要的信息装载他们，然后你可以从表中检索数据来回答关于动物不同种类的问题。本节显示如何做到所有这些事情：

- 创建数据库
- 创建数据库表
- 装载数据到数据库表
- 以各种方法从表中检索数据
- 使用多个表

动物园数据库很简单(特意的)，但是不难把它想象成可能用到类似数据库的真实世界情况。例如，农夫可以使用这样的一个数据库来追踪家畜，或者兽医可以用它跟踪病畜记录。从MySQL网址上可以获得后面章节中将用到的含有部分查询和样例数据的动物园分发。有**tar**压缩格式 (http://downloads.mysql.com/docs/menagerie-db.tar.gz)和Zip压缩格式 (http://downloads.mysql.com/docs/menagerie-db.zip)。

使用SHOW语句找出服务器上当前存在什么数据库：

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

可能你的机器上的数据库列表是不同的，但是很可能有mysql和test数据库。mysql是必需的，因为它描述用户访问权限，test数据库经常作为用户试身手的工作区。

请注意如果没有SHOW DATABASES权限，则不能看见所有数据库。参见13.5.1.3节，"GRANT和REVOKE语法"。

如果test数据库存在，尝试访问它：

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

注意，USE，类似QUIT，不需要一个分号。（如果你喜欢，你可以用一个分号终止这样的语句；这无碍）USE语句在使用上也有另外一个特殊的地方：它必须在一个单行上给出。

你可列在后面的例子中使用test数据库(如果你能访问它)，但是你在该数据库创建的任何东西可以被访问它的其它人删除，因此，你应该询问MySQL管理员许可你使用自己的一个数据库。假定你想要调用你的menagerie，管理员需要执行这样一条命令：

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

这里your_mysql_name是分配给你的**MySQL**用户名，your_client_host是所连接的服务器所在的主机。

3.3.1. 创建并选择数据库

如果管理员在设置权限时为你创建了数据库，你可以开始使用它。否则，你需要自己创建数据库：

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

在Unix下，数据库名称是区分大小写的(不像SQL关键字)，因此你必须总是以menagerie访问数据库，而不能用Menagerie、MENAGERIE或其它一些变量。对表名也是这样的。（在Windows下，该限制不适用，尽管你必须在一个给定的查询中使用同样的大小写来引用数据库和表。但是，由于多种原因，作为最好的惯例，一定要使用与数据库创建时的同样的大小写。）

创建数据库并不表示选定并使用它，你必须明确地操作。为了使menagerie成为当前的数据库，使用这个命令：

```
mysql> USE menagerie

Database changed
```

数据库只需要创建一次，但是必须在每次启动**mysql**会话时在使用前先选择它。你可以根据上面的例子执行一个USE语句来实现。还可以在调用**mysql**时，通过命令行选择数据库，只需要在提供连接参数之后指定数据库名称。例如：

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

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

注意，刚才显示的命令行中的menagerie不是你的 密码。如果你想要在命令行上在-p选项后提供 密码，则不能插入空格(例如，如-pmypassword，不是-p mypassword)。但是，不建议在命令行输入密码，因为这样会暴露密码，能被在机器上登录的其它用户窥探到。

3.3.2. 创建表

创建数据库是很容易的部分，但是在这时它是空的，正如SHOW TABLES将告诉你的：

```
mysql> SHOW TABLES;

Empty set (0.00 sec)
```

较难的部分是决定你的数据库结构应该是什么：你需要什么数据库表，各数据库表中有什么样的列。

你将需要一个包含你每个宠物的记录的表。它可称为pet表，并且它应该包含，最少，每个动物的名字。因为名字本身不是很有趣，表应该包含另外的信息。例如，如果在你豢养宠物的家庭有超过一个人，你可能想要列出每个动物的主人。你可能也想要记录例如种类和性别的一些基本的描述信息。

年龄呢？那可能有趣，但是存储到一个数据库中不是一件好事情。年龄随着时间流逝而变化，这意味着你将要不断地更新你的记录。相反，存储一个固定值例如生日比较好，那么，无论何时你需要年龄，可以以当前日期和出生日期之间的差来计算它。**MySQL**提供了日期运算函数，因此这并不困难。存储出生日期而非年龄还有其它优点：

- 你可以使用数据库完成这样的任务，例如生成即将到来的宠物生日的提示。（如果你认为这类查询有点蠢，注意，这与从商务数据库来识别出不久要发给生日祝贺的客户是同一个问题，因为计算机帮助私人联络。）

- 你可以相对于日期而不止是当前日期来计算年龄。例如，如果你在数据库存储死亡日期，你能很容易地计算出一只宠物死时有多大。

你可能想到pet表中其它有用的其它类型信息，但是到目前为止这些已经足够了：名字、主人、种类，性别、出生和死亡日期。

使用一个CREATE TABLE语句指定你的数据库表的布局:

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

VARCHAR适合于name、owner和species列，因为列值是变长的。这些列的长度不必都相同，而且不必是20。你可以挑选从1到65535的任何长度，从中选择一个最合理的值。（如果选择得不合适，后来证明你需要一个更长的字段，**MySQL**提供一个ALTER TABLE语句。）

可以用多种类型的值来表示动物记录中的性别，例如，"m"和"f"，或"male"和"female"。使用单字符"m"和"f"是最简单的方法。

很显然，birth和death列应选用DATE数据类。

创建了数据库表后，SHOW TABLES应该产生一些输出:

```
mysql> SHOW TABLES;
+---------------------+
| Tables in menagerie |
+---------------------+
| pet                 |
+---------------------+
```

为了验证你的表是按你期望的方式创建，使用一个DESCRIBE语句:

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

你可以随时使用DESCRIBE，例如，如果你忘记表中的列的名称或类型时。

3.3.3. 将数据装入表中

创建表后，需要填入内容。通过LOAD DATA和INSERT语句可以完成该任务。

假定你的宠物纪录描述如下。（假定在**MySQL**中期望的日期格式是YYYY-MM-DD；这可能与你习惯的不同。）

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	

因为你是从一个空表开始的，填充它的一个简易方法是创建一个文本文件，每个动物各一行，然后用一个语句将文件的内容装载到表中。

你可以创建一个文本文件“`pet.txt`”，每行包含一个记录，用定位符(tab)把值分开，并且以CREATE TABLE语句中列出的列次序给出。对于丢失的值(例如未知的性别，或仍然活着的动物的死亡日期)，你可以使用NULL值。为了在你的文本文件中表示这些内容，使用\N（反斜线，字母N）。例如，Whistler鸟的记录应为(这里值之间的空白是一个定位符)：

name	owner	species	sex	birth	death
Whistler	Gwen	bird	\N	1997-12-09	\N

要想将文本文件“`pet.txt`”装载到pet表中，使用这个命令：

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet;
```

请注意如果用Windows中的编辑器（使用\r\n做为行的结束符）创建文件，应使用：

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet
    -> LINES TERMINATED BY '\r\n';
```

（在运行OS X的Apple机上，应使用行结束符'\r'。）

如果你愿意，你能明确地在LOAD DATA语句中指出列值的分隔符和行尾标记，但是默认标记是定位符和换行符。这对读入文件“`pet.txt`”的语句已经足够。

如果该语句失败，可能是你安装的MySQL不与使用默认值的本地文件兼容。关于如何更改请参见5.6.4节，“LOAD DATA LOCAL安全问题”。

如果想要一次增加一个新记录，可以使用INSERT语句。最简单的形式是，提供每一列的值，其顺序与CREATE TABLE语句中列的顺序相同。假定Diane把一只新仓鼠命名为Puffball，你可以使用下面的INSERT语句添加一条新记录：

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

注意，这里字符串和日期值均为引号扩起来的字符串。另外，可以直接用INSERT语句插入NULL代表不存在的值。不能使用LOAD DATA中所示的的\N。

从这个例子，你应该能看到涉及很多的键入用多个INSERT语句而非单个LOAD DATA语句装载你的初始记录。

3.3.4. 从表检索信息

3.3.4.1. 选择所有数据
3.3.4.2. 选择特殊行
3.3.4.3. 选择特殊列

3.3.4.4. 分类行
3.3.4.5. 日期计算
3.3.4.6. NULL值操作
3.3.4.7. 模式匹配
3.3.4.8. 计数行
3.3.4.9. 使用1个以上的表

SELECT语句用来从数据表中检索信息。语句的一般格式是:

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

what_to_select指出你想要看到的内容，可以是列的一个表，或*表示"所有的列"。which_table指出你想要从其检索数据的表。WHERE子句是可选项，如果选择该项，conditions_to_satisfy指定行必须满足的检索条件。

3.3.4.1. 选择所有数据

SELECT最简单的形式是从一个表中检索所有记录:

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

如果你想要浏览整个表，可以使用这种形式的SELECT，例如，刚刚装载了初始数据集以后。也有可能你想到Bowser的生日看起来不很对。查阅你原来的家谱，你发现正确的出生年是1989，而不是1979。

至少有两种修正方法:

- 编辑文件"pet.txt"改正错误，然后使用DELETE和LOAD DATA清空并重新装载表:

```
mysql> DELETE FROM pet;
mysql> LOAD DATA LOCAL INFILE 'pet.txt' INTO TABLE pet;
```

然而，如果这样操做，必须重新输入Puffball记录。

- 用一个UPDATE语句仅修正错误记录:

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

UPDATE只更改有问题的记录，不需要重新装载数据库表。

3.3.4.2. 选择特殊行

如上所示，检索整个表是容易的。只需要从SELECT语句中删掉WHERE子句。但是一般你不想看到整个表，特别地当表变得很大时。相反，你通常对回答一个具体的问题更感兴趣，在这种情况下在你想要的信息上进行一些限制。让我们看一些他们回答的有关你宠物的问题的选择查询。

可以从表中只选择特定的行。例如，如果你想要验证你对Bowser的生日所做的更改，按下述方法选择Bowser的记录：

```
mysql> SELECT * FROM pet WHERE name = 'Bowser';
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
```

输出证实正确的年份记录为1989，而不是1979。

字符串比较时通常对大小些不敏感，因此你可以将名字指定为"bowser"、"BOWSER"等，查询结果相同。

你可以在任何列上指定条件，不只仅仅是name。例如，如果你想要知道哪个动物在1998以后出生的，测试birth列：

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

可以组合条件，例如，找出雌性的狗：

```
mysql> SELECT * FROM pet WHERE species = 'dog' AND sex = 'f';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

上面的查询使用AND逻辑操作符，也有一个OR操作符：

```
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和OR可以混用，但AND比OR具有更高的优先级。如果你使用两个操作符，使用圆括号指明如何对条件进行分组是一个好主意：

```
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. 选择特殊列

如果你不想看到表中的所有行，就命名你感兴趣的列，用逗号分开。例如，如果你想要知道你的动物什么时候出生的，选择name和birth列：

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

找出谁拥有宠物，使用这个查询：

```
mysql> SELECT owner FROM pet;
+--------+
| owner  |
+--------+
```

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

请注意该查询只是简单地检索每个记录的owner列，并且他们中的一些出现多次。为了使输出减到最少，增加关键字DISTINCT检索出每个唯一的输出记录：

```
mysql> SELECT DISTINCT owner FROM pet;
+--------+
| owner  |
+--------+
| Benny  |
| Diane  |
| Gwen   |
| Harold |
+--------+
```

可以使用一个WHERE子句结合行选择与列选择。例如，要想查询狗和猫的出生日期，使用这个查询：

```
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. 分类行

你可能已经注意到前面的例子中结果行没有以特定的顺序显示。然而，当行按某种方式排序时，检查查询输出通常更容易。为了排序结果，使用ORDER BY子句。

这里是动物生日，按日期排序：

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

在字符类型列上，与所有其他比较操作类似，分类功能正常情况下是以区分大小写的方式执行的。这意味着，对于等同但大小写不同的列，并未定义其顺序。对于某一列，可以使用BINARY强制执行区分大小写的分类功能，如：ORDER BY BINARY *col_name*.

默认排序是升序，最小的值在第一。要想以降序排序，在你正在排序的列名上增加DESC（降序）关键字：

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

可以对多个列进行排序，并且可以按不同的方向对不同的列进行排序。例如，按升序对动物的种类进行排序，然后按降序根据生日对各动物种类进行排序（最年轻的动物在最前面），使用下列查询：

```
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关键字仅适用于在它前面的列名(birth)；不影响species列的排序顺序。

3.3.4.5. 日期计算

MySQL提供了几个函数，可以用来计算日期，例如，计算年龄或提取日期部分。

要想确定每个宠物有多大，可以计算当前日期的年和出生日期之间的差。如果当前日期的日历年比出生日期早，则减去一年。以下查询显示了每个宠物的出生日期、当前日期和年龄数值的年数字。

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

此处，YEAR()提取日期的年部分，RIGHT()提取日期的MM-DD (日历年)部分的最右面5个字符。比较MM-DD值的表达式部分的值一般为1或0，如果CURDATE()的年比birth的年早，则年份应减去1。整个表达式有些难懂，使用*alias* (age)来使输出的列标记更有意义。

尽管查询可行，如果以某个顺序排列行，则能更容易地浏览结果。添加ORDER BY name子句按照名字对输出进行排序则能够实现。

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

为了按age而非name排序输出，只要再使用一个ORDER BY子句：

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

可以使用一个类似的查询来确定已经死亡动物的死亡年龄。你通过检查death值是否是NULL来确定是哪些动物，然后，对于那些非NULL值的动物，需要计算出death和birth值之间的差：

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

查询使用death IS NOT NULL而非death != NULL，因为NULL是特殊的值，不能使用普通比较符来比较，以后会给出解释。参见3.3.4.6节，“NULL值操作”。

如果你想要知道哪个动物下个月过生日，怎么办？对于这类计算，年和天是无关的，你只需要提取birth列的月份部分。**MySQL**提供几个日期部分的提取函数，例如YEAR( )、MONTH( )和DAYOFMONTH( )。在这里MONTH()是适合的函数。为了看它怎样工作，运行一个简单的查询，显示birth和MONTH(birth)的值：

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

找出下个月生日的动物也是容易的。假定当前月是4月，那么月值是4，你可以找在5月出生的动物 (5月)，方法是：

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

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

如果当前月份是12月，就有点复杂了。你不能只把1加到月份数(12)上并寻找在13月出生的动物，因为没有这样的月份。相反，你应寻找在1月出生的动物(1月) 。

你甚至可以编写查询，不管当前月份是什么它都能工作。采用这种方法不必在查询中使用一个特定的月份，DATE_ADD( )允许在一个给定的日期上加上时间间隔。如果在NOW( )值上加上一个月，然后用MONTH()提取月份，结果产生生日所在月份:

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

完成该任务的另一个方法是加1以得出当前月份的下一个月(在使用取模函数(MOD)后，如果月份当前值是12，则"回滚"到值0):

```
mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MOD(MONTH(CURDATE()), 12) + 1;
```

注意，MONTH返回在1和12之间的一个数字，且MOD(something,12)返回在0和11之间的一个数字，因此必须在MOD( )以后加1，否则我们将从11月(11)跳到1月(1)。

3.3.4.6. NULL值操作

NULL值可能令人感到奇怪直到你习惯它。概念上，NULL意味着"没有值"或"未知值"，且它被看作与众不同的值。为了测试NULL，你不能使用算术比较 操作符例如=、<或!=。为了说明它，试试下列查询:

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+----------+-----------+----------+----------+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+----------+-----------+----------+----------+
|     NULL |      NULL |     NULL |     NULL |
+----------+-----------+----------+----------+
```

很显然你不能通过这些比较得到有意义的结果。相反使用IS NULL和IS NOT NULL操作符:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----------+---------------+
| 1 IS NULL | 1 IS NOT NULL |
+-----------+---------------+
|         0 |             1 |
+-----------+---------------+
```

请注意在**MySQL**中，0或 NULL意味着假而其它值意味着真。布尔运算的默认真值是1。

对NULL的特殊处理即是在前面的章节中，为了决定哪个动物不再是活着的，使用death IS NOT NULL而不使用death != NULL的原因。

在GROUP BY中，两个NULL值视为相同。

执行ORDER BY时，如果运行 ORDER BY ... ASC，则NULL值出现在最前面，若运行ORDER BY ... DESC，则NULL值出现在最后面。

NULL操作的常见错误是不能在定义为NOT NULL的列内插入0或空字符串，但事实并非如此。在NULL表示"没有数值"的地方有数值。使用IS [NOT] NULL则可以很容易地进行测试，如下所示：

```
mysql> SELECT 0 IS NULL, 0 IS NOT NULL, '' IS NULL, '' IS NOT NULL;

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

| 0 IS NULL | 0 IS NOT NULL | '' IS NULL | '' IS NOT NULL |

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

|         0 |             1 |          0 |              1 |

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

因此完全可以在定义为NOT NULL的列内插入0或空字符串，实际是NOT NULL。参见A.5.3节，“与NULL值有关的问题”。

3.3.4.7. 模式匹配

MySQL提供标准的SQL模式匹配，以及一种基于象Unix实用程序如**vi**、**grep**和**sed**的扩展正则表达式模式匹配的格式。

SQL模式匹配允许你使用“_”匹配任何单个字符，而“%”匹配任意数目字符(包括零字符)。在 MySQL中，SQL的模式默认是忽略大小写的。下面给出一些例子。注意使用SQL模式时，不能使用=或!=；而应使用LIKE或NOT LIKE比较操作符。

要想找出以“b”开头的名字：

```
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 |

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

要想找出以“fy”结尾的名字：

```
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  |

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

要想找出包含“w”的名字：

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

要想找出正好包含5个字符的名字，使用“_”模式字符：

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

由**MySQL**提供的模式匹配的其它类型是使用扩展正则表达式。当你对这类模式进行匹配测试时，使用REGEXP和NOT REGEXP操作符(或RLIKE和NOT RLIKE，它们是同义词)。

扩展正则表达式的一些字符是：

- '.'匹配任何单个的字符。
- 字符类“[...]”匹配在方括号内的任何字符。例如，“[abc]”匹配“a”、“b”或“c”。为了命名字符的范围，使用一个“-”。“[a-z]”匹配任何字母，而“[0-9]”匹配任何数字。
- “ * ”匹配零个或多个在它前面的字符。例如，“x*”匹配任何数量的“x”字符，“[0-9]*”匹配任何数量的数字，而“.*”匹配任何数量的任何字符。

- 如果REGEXP模式与被测试值的任何地方匹配，模式就匹配(这不同于LIKE模式匹配，只有与整个值匹配，模式才匹配)。
- 为了定位一个模式以便它必须匹配被测试值的开始或结尾，在模式开始处使用“^”或在模式的结尾用“$”。

为了说明扩展正则表达式如何工作，下面使用REGEXP重写上面所示的LIKE查询：

为了找出以“b”开头的名字，使用“^”匹配名字的开始：

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

如果你想强制使REGEXP比较区分大小写，使用BINARY关键字使其中一个字符串变为二进制字符串。该查询只

匹配名称首字母的小写'b'。

```
mysql> SELECT * FROM pet WHERE name REGEXP BINARY '^b';
```

为了找出以"fy"结尾的名字，使用"$"匹配名字的结尾:

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

为了找出包含一个"w"的名字，使用以下查询:

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

既然如果一个正则表达式出现在值的任何地方，其模式匹配了，就不必在先前的查询中在模式的两侧放置一个通配符以使得它匹配整个值，就像你使用了一个SQL模式那样。

为了找出包含正好5个字符的名字，使用"^"和"$"匹配名字的开始和结尾，和5个"."实例在两者之间:

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

你也可以使用"{n}"“重复n次”操作符重写前面的查询:

```
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  |
```

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

附录G：*MySQL*正则表达式 提供了关于正则表达式的句法的详细信息。

3.3.4.8. 计数行

数据库经常用于回答这个问题，“某个类型的数据在表中出现的频度?”例如，你可能想要知道你有多少宠物，或每位主人有多少宠物，或你可能想要对你的动物进行各种类型的普查。

计算你拥有动物的总数目与“在pet表中有多少行?”是同样的问题，因为每个宠物有一个记录。COUNT(*)函数计算行数，所以计算动物数目的查询应为：

```
mysql> SELECT COUNT(*) FROM pet;
+----------+
| COUNT(*) |
+----------+
|        9 |
+----------+
```

在前面，你检索了拥有宠物的人的名字。如果你想要知道每个主人有多少宠物，你可以使用COUNT( )函数：

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Benny  |        2 |
| Diane  |        2 |
| Gwen   |        3 |
| Harold |        2 |
+--------+----------+
```

注意，使用GROUP BY对每个owner的所有记录分组，没有它，你会得到错误消息：

```
mysql> SELECT owner, COUNT(*) FROM pet;
ERROR 1140 (42000): Mixing of GROUP columns (MIN(),MAX(),COUNT(),...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

COUNT( )和GROUP BY以各种方式分类你的数据。下列例子显示出进行动物普查操作的不同方式。

每种动物的数量：

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
+---------+----------+
| species | COUNT(*) |
+---------+----------+
| bird    |        2 |
| cat     |        2 |
| dog     |        3 |
```

```
| hamster |        1 |
| snake   |        1 |
+---------+----------+
```

每种性别的动物数量:

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
+------+----------+
| sex  | COUNT(*) |
+------+----------+
| NULL |        1 |
| f    |        4 |
| m    |        4 |
+------+----------+
```

（在这个输出中，NULL表示"未知性别"。）

按种类和性别组合的动物数量:

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

若使用COUNT( )，你不必检索整个表。例如,前面的查询，当只对狗和猫进行时，应为:

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

或，如果你仅需要知道已知性别的按性别的动物数目：

```
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. 使用1个以上的表

pet表追踪你有哪个宠物。如果你想要记录其它相关信息，例如在他们一生中看兽医或何时后代出生，你需要另外的表。这张表应该像什么呢？需要：

- 它需要包含宠物名字以便你知道每个事件属于哪个动物。
- 需要一个日期以便你知道事件是什么时候发生的。
- 需要一个描述事件的字段。
- 如果你想要对事件进行分类，则需要一个事件类型字段。

综合上述因素，event表的CREATE TABLE语句应为：

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
    -> type VARCHAR(15), remark VARCHAR(255));
```

对于pet表，最容易的方法是创建包含信息的用定位符分隔的文本文件来装载初始记录：

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

采用如下方式装载记录:

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

根据你从已经运行在pet表上的查询中学到的，你应该能执行对event表中记录的检索；原理是一样的。但是什么时候event表本身不能回答你可能问的问题呢？

当他们有了一窝小动物时，假定你想要找出每只宠物的年龄。我们前面看到了如何通过两个日期计算年龄。event表中有母亲的生产日期，但是为了计算母亲的年龄，你需要她的出生日期，存储在pet表中。说明查询需要两个表:

```
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 event.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         |
+--------+------+-----------------------------+
```

关于该查询要注意的几件事情:

- FROM子句列出两个表，因为查询需要从两个表提取信息。
- 当从多个表组合(联结)信息时，你需要指定一个表中的记录怎样能匹配其它表的记录。这很简单，因为它们都有一个name列。查询使用WHERE子句基于name值来匹配2个表中的记录。
- 因为name列出现在两个表中，当引用列时，你一定要指定哪个表。把表名附在列名前即可以实现。

你不必有2个不同的表来进行联结。如果你想要将一个表的记录与同一个表的其它记录进行比较，可以将一个表联结到自身。例如，为了在你的宠物之中繁殖配偶，你可以用pet联结自身来进行相似种类的雄雌配对:

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

在这个查询中，我们为表名指定别名以便能引用列并且使得每一个列引用与哪个表实例相关联更直观。

3.4. 获得数据库和表的信息

如果你忘记数据库或表的名字，或给定的表的结构是什么(例如，它的列叫什么)，怎么办？**MySQL**通过提供数据库及其支持的表的信息的几个语句解决这个问题。

你已经见到了SHOW DATABASES，它列出由服务器管理的数据库。为了找出当前选择了哪个数据库，使用DATABASE( )函数：

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| menagerie  |
+------------+
```

如果你还没选择任何数据库，结果是NULL。

为了找出当前的数据库包含什么表(例如，当你不能确定一个表的名字)，使用这个命令：

```
mysql> SHOW TABLES;
+---------------------+
| Tables in menagerie |
+---------------------+
| event               |
| pet                 |
+---------------------+
```

如果你想要知道一个表的结构，可以使用DESCRIBE命令；它显示表中每个列的信息：

```
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显示列名字，Type是列的数据类型，Null表示列是否能包含NULL值，Key显示列是否被索引而Default指定列的默认值。

如果表有索引，SHOW INDEX FROM tbl_name生成有关索引的信息。

3.5. 在批处理模式下使用mysql

在前面的章节中，你交互式地使用**mysql**输入查询并且查看结果。你也可以以批模式运行**mysql**。为了做到这些，把你想要运行的命令放在一个文件中，然后告诉**mysql**从文件读取它的输入：

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

如果在Windows下运行**mysql**，并且文件中有一些可以造成问题的特殊字符，可以这样操作：

```
C:\> mysql -e "source batch-file"
```

如果你需要在命令行上指定连接参数，命令应为：

```
shell> mysql -h host -u user -p < batch-file
Enter password: ********
```

当这样操作**mysql**时，则创建一个脚本文件，然后执行脚本。

如果你想在语句出现错误的时候仍想继续执行脚本，则应使用--force命令行选项。

为什么要使用一个脚本？有很多原因：

- 如果你需要重复运行查询(比如说，每天或每周)，可以把它编成一个脚本，则每次执行时不必重新键入。
- 可以通过拷贝并编辑脚本文件从类似的现有的查询生成一个新查询。
- 当你正在开发查询时，批模式也是很有用的，特别对多行命令或多语句命令序列。如果你犯了一个错误，你不必重新输入所有内容，只需要编辑脚本来改正错误，然后告诉**mysql**再次执行脚本。
- 如果你有一个产生多个输出的查询，你可以通过一个分页器而不是盯着它翻屏到屏幕的顶端来运行输出：

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

- 你可以捕捉文件中的输出以便进行进一步的处理：

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

- 你可以将脚本分发给另外的人，以便他们也能运行命令。
- 某些情况不允许交互地使用，例如, 当你从一个**cron**任务中运行查询时。在这种情况下，你必须使用批模式。

当你以批模式运行mysql时，比起你交互地使用它时，其默认输出格式是不同的(更简明些)。例如，当交互式运行SELECT DISTINCT species FROM pet时，输出应为：

```
+---------+
| species |
+---------+
```

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

但是当以批模式运行时，输出应为：

```
species
bird
cat
dog
hamster
snake
```

如果你想要在批模式中得到交互输出格式，使用mysql -t。为了回显以输出被执行的命令，使用mysql -vvv。

你还可以使用源代码或 \.命令从**mysql**提示符运行脚本：

```
mysql> source filename;
mysql> \. filename
```

3.6. 常用查询的例子

3.6.1. 列的最大值
3.6.2. 拥有某个列的最大值的行
3.6.3. 列的最大值：按组
3.6.4. 拥有某个字段的组间最大值的行
3.6.5. 使用用户变量
3.6.6. 使用外键
3.6.7. 根据两个键搜索
3.6.8. 根据天计算访问量
3.6.9. 使用AUTO_INCREMENT

下面是一些学习如何用**MySQL**解决一些常见问题的例子。

在一些例子中，使用数据库表"shop"来储存某个商人（经销商）的每件物品(物品号)的价格。假定每个商人对每项物品有一个固定价格，那么(物品，商人)即为该记录的主关键字。

启动命令行工具**mysql**并选择数据库：

```
shell> mysql your-database-name
```

（在大多数MySQL中，你可以使用test数据库）。

你可以使用以下语句创建示例表：

```
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);
```

执行语句后，表应包含以下内容:

```
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. 列的最大值

“最大的物品号是什么？”

```
SELECT MAX(article) AS article FROM shop;
+---------+
| article |
+---------+
|       4 |
+---------+
```

3.6.2. 拥有某个列的最大值的行

任务：找出最贵物品的编号、销售商和价格。

这很容易用一个子查询做到:

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

另一个解决方案是按价格降序排序所有行并用**MySQL**特定LIMIT子句只得到第一行:

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

```
LIMIT 1;
```

注:如果有多项最贵的物品(例如每个的价格为19.95)，LIMIT解决方案仅仅显示其中一个!

3.6.3. 列的最大值：按组

任务：每项物品的的最高价格是多少?

```
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. 拥有某个字段的组间最大值的行

任务：对每项物品，找出最贵价格的物品的经销商。

可以用这样一个子查询解决该问题:

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

3.6.5. 使用用户变量

你可以清空MySQL用户变量以记录结果，不必将它们保存到客户端的临时变量中。（参见 9.3节，“用户变量”.）。

例如，要找出价格最高或最低的物品的，其方法是:

```
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. 使用外键

在MySQL中，InnoDB表支持对外部关键字约束条件的检查。参见15.2节，“InnoDB存储引擎”。还可以参见1.8.5.5节，“外键”。

只是联接两个表时，不需要外部关键字。对于除InnoDB类型的表，当使用REFERENCES *tbl_name*(*col_name*)子句定义列时可以使用外部关键字，该子句没有实际的效果，只作为备忘录或注释来提醒，你目前正定义的列指向另一个表中的一个列。执行该语句时，实现下面很重要:

- MySQL不执行表*tbl_name* 中的动作，例如作为你正定义的表中的行的动作的响应而删除行；换句话说，该句法不会致使ON DELETE或ON UPDATE行为（如果你在REFERENCES子句中写入ON DELETE或ON UPDATE子句，将被忽略）。

- 该句法可以创建一个*column*；但不创建任何索引或关键字。
- 如果用该句法定义InnoDB表，将会导致错误。

你可以使用作为联接列创建的列，如下所示：

```
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');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'polo', 'blue', @last),
(NULL, 'dress', 'white', @last),
(NULL, 't-shirt', 'blue', @last);

INSERT INTO person VALUES (NULL, 'Lilliana Angelovska');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'dress', 'orange', @last),
(NULL, 'polo', 'red', @last),
(NULL, 'dress', 'blue', @last),
(NULL, 't-shirt', 'white', @last);

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

按照这种方式使用，REFERENCES子句不会显示在SHOW CREATE TABLE或DESCRIBE的输出中:

```
SHOW CREATE TABLE shirt\G
*************************** 1. row ***************************
Table: shirt
Create Table: CREATE TABLE `shirt` (
```

```
`id` smallint(5) unsigned NOT NULL auto_increment,
`style` enum('t-shirt','polo','dress') NOT NULL,
`color` enum('red','blue','orange','white','black') NOT NULL,
`owner` smallint(5) unsigned NOT NULL,
PRIMARY KEY  (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

在列定义中，按这种方式使用REFERENCES作为注释或"提示"适用于表MyISAM和BerkeleyDB。

3.6.7. 根据两个键搜索

可以充分利用使用单关键字的OR子句，如同AND的处理。

一个比较灵活的例子是寻找两个通过OR组合到一起的关键字:

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

该情形是已经优化过的。参见7.2.6节，"索引合并优化"。

还可以使用UNION将两个单独的SELECT语句的输出合成到一起来更有效地解决该问题。参见13.2.7.2节，"UNION语法"。

每个SELECT只搜索一个关键字，可以进行优化:

```
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';
```

3.6.8. 根据天计算访问量

下面的例子显示了如何使用位组函数来计算每个月中用户访问网页的天数。

```
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);
```

示例表中含有代表用户访问网页的年－月－日值。可以使用以下查询来确定每个月的访问天数:

```
SELECT year,month,BIT_COUNT(BIT_OR(1<<day)) AS days FROM t1
       GROUP BY year,month;
```

将返回:

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

```
| year | month | days |
+------+-------+------+
| 2000 |    01 |    3 |
| 2000 |    02 |    2 |
+------+-------+------+
```

该查询计算了在表中按年/月组合的不同天数，可以自动去除重复的询问。

3.6.9. 使用AUTO_INCREMENT

可以通过AUTO_INCREMENT属性为新的行产生唯一的标识：

```
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;
```

将返回：

```
+----+---------+
| id | name    |
+----+---------+
|  1 | dog     |
|  2 | cat     |
|  3 | penguin |
|  4 | lax     |
|  5 | whale   |
|  6 | ostrich |
+----+---------+
```

你可以使用LAST_INSERT_ID()SQL函数或mysql_insert_id() C API函数来查询最新的AUTO_INCREMENT值。这些函数与具体连接有关，因此其返回值不会被其它执行插入功能的连接影响。

注释：对于多行插入，LAST_INSERT_ID()和mysql_insert_id()从插入的第一行实际返回AUTO_INCREMENT关键字。在复制设置中，通过该函数可以在其它服务器上正确复制多行插入。

对于MyISAM和BDB表，你可以在第二栏指定AUTO_INCREMENT以及多列索引。此时，AUTO_INCREMENT列生成的值的计算方法为：MAX(*auto_increment_column*) + 1 WHERE prefix=*given-prefix*。如果想要将数据放

入到排序的组中可以使用该方法。

```
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;
```

将返回:

```
+--------+----+---------+
| grp    | id | name    |
+--------+----+---------+
| fish   |  1 | lax     |
| mammal |  1 | dog     |
| mammal |  2 | cat     |
| mammal |  3 | whale   |
| bird   |  1 | penguin |
| bird   |  2 | ostrich |
+--------+----+---------+
```

请注意在这种情况下（AUTO_INCREMENT列是多列索引的一部分），如果你在任何组中删除有最大AUTO_INCREMENT值的行，将会重新用到AUTO_INCREMENT值。对于MyISAM表也如此,对于该表一般不重复使用AUTO_INCREMENT值。

如果AUTO_INCREMENT列是多索引的一部分，MySQL将使用该索引生成以AUTO_INCREMENT列开始的序列值。。例如，如果animals表含有索引PRIMARY KEY (grp, id)和INDEX(id)，MySQL生成序列值时将忽略PRIMARY KEY。结果是，该表包含一个单个的序列，而不是符合grp值的序列。

要想以AUTO_INCREMENT值开始而不是1，你可以通过CREATE TABLE或ALTER TABLE来设置该值，如下所示:

```
mysql> ALTER TABLE tbl AUTO_INCREMENT = 100;
```

关于AUTO_INCREMENT的详细信息:

- 如何为列指定AUTO_INCREMENT属性：13.1.5节，“CREATE TABLE语法”和 13.1.2节，“ALTER TABLE语法”。
- AUTO_INCREMENT的动作取决于SQL模式：5.3.2节，“SQL服务器模式”。

- 找出含有最新AUTO_INCREMENT值的行：12.1.3节，“比较函数和操作符”。
- 设置将用到的AUTO_INCREMENT值: 13.5.3节，“SET语法” 。
- AUTO_INCREMENT和复制：6.7节，“复制特性和已知问题”.
- AUTO_INCREMENT相关的可用于复制的Server-system变量(auto_increment_increment和auto_increment_offset)：5.3.3节，“服务器系统变量”。

3.7. 孪生项目的查询

3.7.1. 查找所有未分发的孪生项
3.7.2. 显示孪生对状态的表

这个项目是Institute of Environmental Medicine at Karolinska Institutet Stockholm 和 the Section on Clinical Research in Aging and Psychology at the University of Southern California的合作项目。

该项目包括筛选部分，即通过电话回访在瑞典超过 65 岁的所有孪生。满足某种标准的孪生进入下一阶段。在下一阶段中，医生/护士小组将访问想参加的孪生。部分检查包括物理检查和神经、心理检查、实验室试验、神经成像、心理状况评估和家族历史搜集。并且，应根据医疗和环境风险因素来搜集数据。

可从以下链接找到孪生研究的更多信息：

http://www.mep.ki.se/twinreg/index_en.html

用一个用Perl和**MySQL**编写的web接口来管理项目的后面部分。

每天晚上所有会谈的数据被移入一个**MySQL**数据库。

3.7.1. 查找所有未分发的孪生项

下列查询用来决定谁进入项目的第二部分：

```
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 screening 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;
```

一些解释:

- CONCAT(p1.id, p1.tvab) + 0 AS tvid

我们想要在id和tvab的连接上以数字顺序排序。结果加0使得**MySQL**把结果变为一个数字。

- 列id

这标识一对孪生。它是所有表中的一个键。

- 列tvab

这标识孪生中的一个。它的值为1或2。

- 列ptvab

这是tvab的一个逆运算。当tvab是1，它是2，反之亦然。它用来保存输入并且使**MySQL**的优化查询更容易。

这个查询表明，怎样用联结(p1和p2)从同一个表中查找表。在例子中，这被用来检查孪生的一个是否在65岁前死了。如果如此，行不返回值。

上述所有孪生信息存在于所有表中。我们对id,tvab（所有表）和id,ptvab (person_data) 上采用键以使查询更快。

在我们的生产机器上(一台200MHz UltraSPARC)，这个查询返回大约 150-200 行并且时间不超过一秒。

表	行数
person_data	71074
lentus	5291
twin_project	5286
twin_data	2012
informant_data	663
harmony	381
postal_groups	100

3.7.2. 显示孪生对状态的表

每一次会面以一个称为event的状态码结束。下面显示的查询被用来显示按事件组合的所有孪生的表。这表明多少对孪生已经完成，多少对的其中之一已完成而另一个拒绝了，等等。

```
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 screening 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. 与Apache一起使用MySQL

还有一些项目，你可以从MySQL数据库鉴别用户，并且你还可以将日志文件写入MySQL数据库表。

你可以将以下内容放到Apache配置文件中，更改Apache日志格式，使MySQL更容易读取：

```
LogFormat \
        "\"%h\",%{%Y%m%d%H%M%S}t,%>s,\"%b\",\"%{Content-Type}o\",  \
        \"%U\",\"%{Referer}i\",\"%{User-Agent}i\""
```

要想将该格式的日志文件装载到MySQL，你可以使用以下语句：

```
LOAD DATA INFILE '/local/access_log' INTO TABLE tbl_name
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
```

所创建的表中的列应与写入日志文件的LogFormat行对应。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第4章：MySQL程序概述

目录

4.1. MySQL程序概述
4.2. 调用MySQL程序
4.3. 指定程序选项

4.3.1. 在命令行上使用选项
4.3.2. 使用选项文件
4.3.3. 用环境变量指定选项
4.3.4. 使用选项设置程序变量

本章简要概述了MySQL AB提供的命令行程序，并讨论了运行这些程序时如何指定选项。大多数程序具有其操作专用的选项，但指定选项的语法都近似。后面的几章更加详细地描述了各个程序，包括它们识别的选项。

MySQL AB还提供了3个GUI客户程序供MySQL服务器使用：

- MySQL管理器：该工具用于管理MySQL服务器、数据库、表以及用户。
- MySQL查询浏览器：该图形工具由MySQL AB提供，用于创建、执行以及优化对MySQL数据库的查询。
- MySQL移植工具包：该工具可以帮助你将计划和数据从其它关系数据库管理系统移植到MySQL。

4.1. MySQL程序概述

MySQL AB提供了几种类型的程序：

- MYSQL服务器和服务器启动脚本：
  - **mysqld**是MySQL服务器
  - **mysqld_safe**、**mysql.server**和**mysqld_multi**是服务器启动脚本
  - **mysql_install_db**初始化数据目录和初始数据库

在第5章：数据库管理中详细讨论了这些程序。

- 访问服务器的客户程序：
  - **mysql**是一个命令行客户程序，用于交互式或以批处理模式执行SQL语句。
  - **mysqladmin**是用于管理功能的客户程序。
  - **mysqlcheck**执行表维护操作。
  - **mysqldump**和**mysqlhotcopy**负责数据库备份。
  - **mysqlimport**导入数据文件。
  - **mysqlshow**显示信息数据库和表的相关信息。

在第8章：客户端和实用工具程序中详细讨论了这些程序。

- 独立于服务器操作的工具程序：
  - **myisamchk**执行表维护操作。
  - **myisampack**产生压缩、只读的表。
  - **mysqlbinlog**是处理二进制日志文件的实用工具。
  - **perror**显示错误代码的含义。

在第5章：数据库管理中详细讨论了**myisamchk**。在第8章：客户端和实用工具程序中详细讨论了其它程序。

大多数MySQL分发包括上述的全部程序，只是不包含那些与平台相关的程序。(例如，在Windows中不使用服务器启动脚本）。不同的只是RPM分发更加具体化。可能一个RPM是服务器分发，另一个RPM是客户程

序分发等等。如果你丢失了一个或多个程序，要想查阅关于分发类型和它们所包含内容的信息，参见第2章：安装MySQL。也可能需要安装一些其它内容。

4.2. 调用MySQL程序

要想从命令行调用MySQL程序(即从shell或命令提示)，应输入程序名，并随后输入指导操作发的选项或其它参量。下面的命令显示了一些程序调用的例子。"shell>"表示命令解释符提示；并不是输入的部分。你所看见的具体提示取决于命令解释符。典型提示符：**sh**或**bash**为$，**csh**或**tcsh**为%，Windows **command.com**或**cmd.exe**为C:\>。

```
shell> mysql test

shell> mysqladmin extended-status variables

shell> mysqlshow --help

shell> mysqldump --user=root personnel
```

以破折号开始的参数为选项参量。它们通常指定程序与服务器的连接类型或影响其操作模式。关于选项语法的描述参见4.3节，"指定程序选项"。

非选项参数(不以破折号开始的参量)可以为程序提供更详细的信息。例如，**mysql**程序将第一个非选项参量解释为数据库名，因此命令 mysql test表示你想要使用test数据库。

后面的章节描述了具体的程序，表示程序可以理解的选项，并描述了其它非选项参量的含义。

部分选项对部分程序是通用的。最常用的是指定连接参数的--host、--user和--password选项。它们指示MySQL服务器运行的主机和MySQL账户的用户名和 密码。所有MySQL客户程序可以理解这些选项；它们允许你指定连接哪个服务器，以及在该服务器上使用的 账户。

你也许会发现需要使用MySQL程序安装的bin目录的路径名来调用MySQL程序。如果你试图不在bin目录下运行MySQL程序，可能会遇到"程序未找到"错误。为了更方便地使用MySQL，可以将bin目录的路径名添加到PATH环境变量中。然后运行程序时只需要输入程序名，而不需要输入整个路径名。

关于设置PATH的指令的命令解释符请查阅相关文档。设置环境变量的语法与解释符有关。

4.3. 指定程序选项

4.3.1. 在命令行上使用选项
4.3.2. 使用选项文件
4.3.3. 用环境变量指定选项
4.3.4. 使用选项设置程序变量

可以用几种方式提供MySQL程序的选项：

- 在命令行中在程序名后面提供。这对于具体程序调用时使用的选项很普遍。
- 在程序启动时读取的选项文件中设置。这对于每次程序运行时使用的选项很普遍。
- 在环境变量中设置。这对每次程序运行时所使用的选项很有用，尽管实际上最常用选项文件。(5.12.2节，"在Unix中运行多个服务器"中讨论了环境变量会很有帮助的一种情况。描述了使用这些变量来指定服务器和客户程序的TCP/IP端口号和Unix套接字文件的各种技术）。

MySQL程序首先检查环境变量，然后检查选项文件，然后检查命令行来确定给出了哪些选项。如果多次指定一个选项，最后出现的选项占先。这说明环境变量具有最低的优先级，命令行选项具有最高优先级。

可以在选项文件中指定程序选项的默认值来让MySQL程序处理各选项。不需要在每次运行程序时输入选项，但可以根据需要通过命令行选项来覆盖默认值。

4.3.1. 在命令行上使用选项

在命令行中指定的程序选项遵从下述规则：

- 在命令名后面紧跟选项。
- 选项参量以一个和两个破折号开始，取决于它具有短名还是长名。许多选项有两种形式。例如，-?和--help是指导MySQL程序显示帮助消息的选项的短名和长名。
- 选项名对大小写敏感。-v和-V均有效，但具有不同的含义。(它们是--verbose和--version选项的短名）。
- 部分选项在选项名后面紧随选项值。例如，-h localhost或--host=localhost表示客户程序的MySQL服务器主机。选项值可以告诉程序MySQL服务器运行的主机名。
- 对于带选项值的长选项，通过一个'='将选项名和值隔离开来。对于带选项值的短选项，选项值可以紧随选项字母后面，或者二者之间可以用一个空格隔开。(-hlocalhost和-h localhost是等效的）。该规则的例外情况是指定MySQL密码的选项。该选项的形式可以为--password=*pass_val*或--password。在后一种情况(未给出 密码值)，程序将提示输入密码。也可以给出密码选项，短形式为-p*pass_val*或-p。然而，对于短形式，如果给出了 密码值，必须紧跟在选项后面，中间不能插入空格。这样要求的原因是如果选项后面有空格，程序没有办法来告知后面的参量是 密码值还是其它某种参量。因此，下面两个命令的含义完全不同：
- `shell> mysql -ptest`
- `shell> mysql -p test`

第一个命令让**mysql**使用密码test，但没有指定默认数据库。第二个命令让**mysql**提示输入 密码并使用test作为默认数据库。

部分选项控制可以开关的行为。例如，**mysql**客户端支持--column-names选项，确定是否在查询结果开头显示一行栏目名。默认情况，该选项被启用。但是可能在某些情况下你想要禁用它，例如将**mysql**的输出发送到另一个只希望看到数据而不希望看到开始的标题行的程序中。

要想禁用列名，可以使用下面的形式来指定选项：

```
--disable-column-names
--skip-column-names
--column-names=0
```

--disable和--skip前缀与=0后缀的效果相同：它们均关闭选项。

可以用下述方法"启用"选项：

```
--column-names
--enable-column-names
--column-names=1
```

如果选项有前缀--loose，如果程序未识别出选项不会提示错误退出，但是会发出一条警告：

```
shell> mysql --loose-no-such-option
mysql: WARNING: unknown option '--no-such-option'
```

当你从安装了多个MySQL的同一台机器上运行程序时，--loose前缀会很有用。当你在一个选项文件中列出选项时，该前缀会特别有用。有可能不能被程序的所有版本识别的选项可以冠以--loose前缀(或在选项文件中用loose)。不能识别选项的程序版本将会发出一条警告并忽视该选项。

对**mysql**偶尔有用的另一个选项是-e或--execute选项，可用来将SQL语句传递给服务器。该语句必须用引号引起来(单引号或双引号)。(然而，如果想要在查询中将值引起来，则对于查询应使用双引号，查询中引用的值应使用单引号）。当使用该选项时，语句被执行，然后**mysql**立即退出命令外壳。

例如，你可以用下面的命令获得用户账户列表：

```
shell> mysql -u root -p -e "SELECT User, Host FROM User" mysql
Enter password: ******
+------+-----------+
| User | Host      |
+------+-----------+
|      | gigan     |
| root | gigan     |
|      | localhost |
| jon  | localhost |
| root | localhost |
+------+-----------+
shell>
```

请注意mysql数据库名作为一个独立的参量传递。然而，相同的查询可能已经使用**mysql -u root -p -e "SELECT User**，**Host FROM mysql.User"**从外壳中执行。

可以按这种方式传递多个SQL语句，用分号隔开：

```
shell> mysql -u root -p --execute="SELECT Name FROM Country WHERE Name LIKE 'AU%';SELECT COUNT(*) FROM City" world
Enter password: ******
+-----------+
| Name      |
+-----------+
| Australia |
| Austria   |
+-----------+
+----------+
```

```
| COUNT(*) |
+----------+
|     4079 |
+----------+
```

请注意长形式(--execute)后面必须紧跟一个等号(=)。

-e选项也可以以类似方式用来将命令传递给MySQL簇的**ndb_mgm**管理客户端。相关例子参见17.3.6节，“安全关闭和重启”。

4.3.2. 使用选项文件

MySQL程序可以从选项文件(有时也称为配置文件)读取启动选项。选项文件提供了一种很方便的方式来指定常用的选项，因此不需要每次运行程序时从命令行输入。

下面的程序支持选项文件：**myisamchk**、**myisampack**、**mysql**、**mysql.server**、**mysqladmin**、**mysqlbinlog**、**mysqlcc**、**mysqlcheck**、**mysqld_safe**、**mysqldump**、**mysqld**、**mysqlhotcopy**、**mysqlimport**和**mysqlshow**。

注释：MySQL簇程序使用的选项文件参见17.4节，“MySQL簇的配置”。

在Windows中，MySQL程序从以下文件读取启动选项：

文件名	目的
WINDIR\my.ini	全局选项
C:\my.cnf	全局选项
INSTALLDIR\my.ini	全局选项
defaults-extra-file	用--defaults-extra-file=*path*指定的文件，如果有

*WINDIR*表示Windows目录的位置。通常为C:\WINDOWS或C:\WINNT。你可以使用下面的命令通过环境变量WINDIR的值确定其确切位置：

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

*INSTALLDIR*表示MySQL的安装目录。一般为C:*PROGRAMDIR*\MySQL\MySQL 5.1 Server，其中*PROGRAMDIR*表示程序目录(通常为Windows英文版的Program Files)，MySQL 5.1的安装是根据安装和配置向导完成的。参见2.3.5.14节，“my.ini文件的位置”。

在Unix中，MySQL程序从下面的文件读取启动选项：

文件名	目的
/etc/my.cnf	全局选项
$MYSQL_HOME/my.cnf	服务器相关选项
defaults-extra-file	用--defaults-extra-file=*path*指定的文件，如果有
~/.my.cnf	用户相关选项

MYSQL_HOME是一个环境变量，包含服务器相关的my.cnf文件驻留的目录路径。

如果未设置MYSQL_HOME，并且*DATADIR*中有一个my.cnf文件，*BASEDIR*中没有my.cnf文件，mysqld_safe将MYSQL_HOME设置为*DATADIR*。否则，如果未设置MYSQL_HOME并且在*DATADIR*中没有my.cnf，则mysqld_safe将MYSQL_HOME设置为*BASEDIR*。

典型情况二进制安装的目录为/usr/local/mysql/data或源代码安装的目录为/usr/local/var。请注意这是配置时指定的数据目录的位置，而不是 **mysqld**启动时用--datadir指定的。运行时使用--datadir对寻找选项文件的服务器没有效果，因为服务器在处理命令行参量之前寻找这些选项。

MySQL按照上述顺序寻找选项文件，并读存在的选项文件。如果你想要使用的某个选项文件不存在，则用明文文本编辑器创建。如果存在多个选项文件，文件中指定的后读取的选项要优先文件中指定的先读取的选项。

注释：在Unix平台上，MySQL忽略人人可写的配置文件。这是故意的，是一个安全措施。

任何可以在运行MySQL程序时在命令行给出的长选项也可以在选项文件中给出。要想列出程序的适用选项，用--help选项运行程序。

在选项文件中指定选项的语法类似于命令行语法，例外的是要忽略掉两个破折号。例如，命令行中的--quick或--host=localhost在选项文件中应指定为quick或host=localhost。要想在选项文件中指定--loose-*opt_name*形式的选项，应写为loose-*opt_name*。

选项文件中的空行被忽略掉。非空行可以采用下面任何形式：

- #注释，；注释

注释行以'#'或'; '开头。'#'注释也可以从行的中部开始。

· [*group*]

*group*是你想要设置选项的程序名或组名。在组行后面，任何*opt_name*或set-variable行适用于组名，直到选项文件结尾或给出其它组行。

· *opt_name*

等价于命令行中的--*opt_name*。

· *opt_name*=*value*

等价于命令行中的--*opt_name*=*value*。在选项文件中，'='字符附近可以有空格，而在命令行中是不允许的。你可以用单引号或双引号来引用值。如果值包含一个'#'注释字符或空格时很有用。

选项名和值前后的空白将自动删除掉。你可以在选项值中使用转义序列'\b'、'\t'、'\n'、'\r'、'\\'以及'\s'来表示退格、tab、换行符、回车以及空格字符。

在Windows中，如果某个选项值表示一个路径名，应使用'/'而不是'\'作为路径名间隔符来指定值。如果使用'\'，必须用双斜线'\\'，因为'\'在MySQL中为转义字符。

如果选项组名与程序名相同，则组内的选项专用于该程序。

所有客户程序(但不能被**mysqld**)读取[client]选项组。这样允许你指定适用于所有客户端的选项。例如，[client]是用于指定连接服务器的 密码的理想的组。(但应确保该选项文件只能由你自己读写，以便其他人不能发现你的密码）。一定不要随意在[client]组内放置选项，除非它可以被你使用的所有客户程序识别。如果你试图运行程序，如果程序不理解选项则会显示一条错误消息后退出。

从5.0系列的MySQL 5.0.4开始，可以在选项文件中使用!include指令来包括具体文件和!includedir来搜索选项文件的具体目录。例如，要包括文件/home/mydir/myopt.cnf，可以使用:

```
!include /home/me/myopt.cnf
```

要搜索所有以.cnf结尾的文件的目录/home/mydir并作为选项文件读取，应使用:

```
!includedir /home/mydir
```

请注意这些选项与节有关。例如，假定你想要使用my.cnf中的某些内容，如下所示:

```
[mysqld]
!include /home/mydir/myopt.cnf
```

在这种情况下，只为该服务器处理文件myopt.cnf，并且!include指令将被客户应用程序忽略。然而，如果你使用下面的部分:

```
[mysqldump]
!includedir /home/mydir/my-dump-option
```

则只有**mysqldump**为以.cnf结尾的文件检查目录/home/mydir/my-dump-option，服务器或其它客户应用程序均不检查。

注释：目前，在Unix操作系统中，所发现的使用!includedir指令包括的文件的文件名必须以.cnf为扩展名。在Windows中，该指令也为有.ini扩展名(包括.cnf)的文件做检查。

如果你想要创建只由一个具体**mysqld**服务器发布系列读取的选项组，选项组可以用[mysqld-5.0]、[mysqld-5.1]等名称。下面的组表示--new选项只能用于5.1.x 版本的MySQL服务器:

```
[mysqld-5.1]
new
```

下面是一个典型的全局选项文件:

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
key_buffer_size=16M
max_allowed_packet=8M

[mysqldump]
quick
```

在上述的选项文件中，设置key_buffer_size和max_allowed_packet变量的行使用了*var_name*=*value*语法。

下面是一个典型的用户选项文件：

```
[client]
# The following password will be sent to all standard MySQL clients
password="my_password"

[mysql]
no-auto-rehash
connect_timeout=2

[mysqlhotcopy]
interactive-timeout
```

如果你有一个源代码分发，可以从support-file目录中找到名为my-*xxxx*.cnf的示例选项文件。如果你有一个二进制分发，在MySQL安装目录的support-file目录中查找。在Windows中，示例选项文件也可以位于MySQL的安装目录(如果你不知道在哪里，查阅本节前面或第2章：安装MySQL)。目前有小、中等、大以及非常大的系统的示例选项文件。要想练习这些文件，在Windows中复制到C:\my.cnf或在Unix中复制到根目录的.my.cnf。

注释：在Windows中，可以不显示.cnf选项文件的扩展名。

所有支持选项文件的MySQL程序可以处理下面的命令行选项：

- --no-defaults

不读取任何选项文件。

- --print-defaults

打印从选项文件中获得的程序名和所有选项。

- --defaults-file=*path_name*

只使用给出的选项文件。*path_name*是文件的全路径名。

- --defaults-extra-file=*path_name*

在全局选项文件后但在用户选项文件前读该选项文件。*path_name*是文件的全路径名。

为了正确工作，每个选项必须紧随命令行中的命令名后，例外情况是--print-defaults可以紧随--defaults-file或--defaults-extra-file。

在shell脚本中，可以使用**my_print_defaults**程序来分析选项文件。下面的例子显示了当要求显示[client]和[mysql]组内发现的选项时**my_print_defaults**产生的输出：

```
shell> my_print_defaults client mysql
--port=3306
--socket=/tmp/mysql.sock
--no-auto-rehash
```

开发人员请注意：在C客户程序库中，只是通过处理命令行参量前面的所有匹配选项(即相应组内的选项)来处理选项文件。对于使用多次指定的最后的选项实例的程序，可以很好地工作。如果你有一个C或C++程序按这种方式处理多次指定的选项，但不读取选项文件，只需要添加两行来实现该功能。检查标准MySQL客户程序的源代码来看看如何做。

其它几个MySQL语言接口基于C客户库，它们中的一部分提供了访问选项文件的一种方式。包括Perl和Python。关于接口的详细信息参见相关文档。

4.3.3. 用环境变量指定选项

要想用环境变量指定选项，使用适用注释处理器的语法来设置变量。例如，在Windows或NetWare中，可以设置USER变量来指定MySQL账户名。要想实现，使用语法：

```
SET USER=your_name
```

在Unix中的语法取决于你的外壳。假定你想要使用MYSQL_TCP_PORT变量指定TCP/IP端口号。典型的语法为(例如**sh**、bash、**zsh**等等)：

```
MYSQL_TCP_PORT=3306
```

```
export MYSQL_TCP_PORT
```

第1个命令设置变量，export命令将变量导出到外壳环境，以便其值可供MySQL和其它进程访问。

csh和**tcsh**有类似的问题。运行这些外壳时，使用**setenv**使外壳变量适用环境：

```
setenv MYSQL_TCP_PORT 3306
```

可以在命令提示符下执行设置环境变量的命令，以便立即生效。这些设定值持续到你注销。要向让这些设定值在你每次登录时生效，将相应命令放入每次启动时命令解释符所读的启动文件中。在Windows中典型启动文件为AUTOEXEC.BAT，**bash**为.bash_profile，或者**tcsh**为.tcshrc。关于命令解释符的具体细节请查阅文档。

附录F：环境变量 了列出影响MySQL程序操作的所有环境变量。

4.3.4. 使用选项设置程序变量

许多MySQL程序有一些内部变量可以在运行时设置。程序变量的设置与带有值的其它长选项相同。例如，**mysql**有一个max_allowed_packet变量，可以控制其通信缓冲区的最大长度。要想为**mysql**将max_allowed_packet变量的值设置为16MB，使用下面的任何一个命令：

```
shell> mysql --max_allowed_packet=16777216
shell> mysql --max_allowed_packet=16M
```

第1个命令以字节指定值。第2个命令以兆字节指定值。变量值可以有一个后缀K、M或者G(可以为大写或小写)来表示千字节、兆字节或者十亿字节的单位。

在选项文件中，变量设定值没有引导破折号：

```
[mysql]
max_allowed_packet=16777216
```

或：

```
[mysql]
max_allowed_packet=16M
```

如果你喜欢，变量名的下划线可以为破折号。

注释：MySQL 5.1仍然能识别旧的语法--set-variable = *option*=*value*，但现在不赞成使用。

部分服务器变量可以在运行时设置。详情参见5.3.3.1节，“动态系统变量”。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com.。原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第5章：数据库管理

目录

5.1. MySQL服务器和服务器启动脚本

- 5.1.1. 服务器端脚本和实用工具概述
- 5.1.2. mysqld-max扩展MySQL服务器
- 5.1.3. mysqld_safe：MySQL服务器启动脚本
- 5.1.4. mysql.server：MySQL服务器启动脚本
- 5.1.5. mysqld_multi：管理多个MySQL服务器的程序

5.2. mysqlmanager：MySQL实例管理器

- 5.2.1. 用MySQL实例管理器启动MySQL服务器
- 5.2.2. 连接到MySQL实例管理器并创建用户账户
- 5.2.3. MySQL实例管理器命令行选项
- 5.2.4. MySQL实例管理器配置文件
- 5.2.5. MySQL实例管理器识别的命令

5.3. mysqld：MySQL服务器

- 5.3.1. mysqld命令行选项
- 5.3.2. SQL服务器模式
- 5.3.3. 服务器系统变量
- 5.3.4. 服务器状态变量

5.4. mysql_fix_privilege_tables：升级MySQL系统表
5.5. MySQL服务器关机进程
5.6. 一般安全问题

- 5.6.1. 通用安全指南
- 5.6.2. 使MySQL在攻击者面前保持安全
- 5.6.3. Mysqld安全相关启动选项
- 5.6.4. LOAD DATA LOCAL安全问题

5.7. MySQL访问权限系统

- 5.7.1. 权限系统的作用
- 5.7.2. 权限系统工作原理
- 5.7.3. MySQL提供的权限
- 5.7.4. 与MySQL服务器连接
- 5.7.5. 访问控制, 阶段1：连接核实
- 5.7.6. 访问控制, 阶段2：请求核实
- 5.7.7. 权限更改何时生效
- 5.7.8. 拒绝访问错误的原因
- 5.7.9. MySQL 4.1中的密码哈希处理

5.8. MySQL用户账户管理

- 5.8.1. MySQL用户名和密码
- 5.8.2. 向MySQL增加新用户账户
- 5.8.3. 从MySQL删除用户账户
- 5.8.4. 限制账户资源
- 5.8.5. 设置账户密码
- 5.8.6. 使你的密码安全
- 5.8.7. 使用安全连接

5.9. 备份与恢复

- 5.9.1. 数据库备份
- 5.9.2. 示例用备份与恢复策略
- 5.9.3. 自动恢复
- 5.9.4. 表维护和崩溃恢复
- 5.9.5. myisamchk：MyISAM表维护实用工具
- 5.9.6. 建立表维护计划
- 5.9.7. 获取关于表的信息

5.10. MySQL本地化和国际应用

- 5.10.1. 数据和排序用字符集
- 5.10.2. 设置错误消息语言
- 5.10.3. 添加新的字符集
- 5.10.4. 字符定义数组
- 5.10.5. 字符串比较支持
- 5.10.6. 多字节字符支持
- 5.10.7. 字符集问题

5.10.8. MySQL服务器时区支持

5.11. MySQL日志文件

5.11.1. 错误日志
5.11.2. 通用查询日志
5.11.3. 二进制日志
5.11.4. 慢速查询日志
5.11.5. 日志文件维护

5.12. 在同一台机器上运行多个MySQL服务器

5.12.1. 在Windows下运行多个服务器
5.12.2. 在Unix中运行多个服务器
5.12.3. 在多服务器环境中使用客户端程序

5.13. MySQL查询高速缓冲

5.13.1. 查询高速缓冲如何工作
5.13.2. 查询高速缓冲SELECT选项
5.13.3. 查询高速缓冲配置
5.13.4. 查询高速缓冲状态和维护

本章涵盖了MySQL安装管理主题，例如配置服务器、管理用户账户和备份。

5.1. MySQL服务器和服务器启动脚本

5.1.1. 服务器端脚本和实用工具概述
5.1.2. mysq**ld-max**扩展**My**SQL服务器
5.1.3. mysqld_safe：MySQL服务器启动脚本
5.1.4. mysql.server：MySQL服务器启动脚本
5.1.5. mysqld_multi：管理多个MySQL服务器的程序

MySQL服务器，即**mysqld**，是在MySQL安装中负责大部分工作的主程序。服务器随附了几个相关脚本，当你安装MySQL时它们可以执行设置操作，或者是帮助你启动和停止服务器的帮助程序。

本节提供了服务器和相关程序的概述，以及服务器启动脚本相关信息。关于配置服务器的信息参见5.3节，“mysqld：MySQL服务器”。

5.1.1. 服务器端脚本和实用工具概述

MySQL程序采用各种不同的选项。但每个MySQL程序提供一个--help选项,你可以用来查阅程序选项相关说明。例如，你可以试试**mysqld --help**。

你可以在命令行中或在选项文件中指定选项来替换所有标准程序中的默认选项。参见4.3节，“指定程序选项”。

下面简单描述了MySQL服务器和服务器相关程序：

· **mysqld**

SQL后台程序(即MySQL服务器)。要想使用客户端程序，该程序必须运行，因为客户端通过连接服务器来访问数据库。参见5.3节，“mysqld：MySQL服务器”。

· mysqld-max

包括更多特性的一个服务器版本。参见5.1.2节，“mysq**ld-max**扩展**My**SQL服务器”

· mysqld_safe

服务器启动脚本。如果**mysqld-max**存在,**mysqld_safe**试图启动它，否则启动**mysqld**。参见5.1.3节，“mysqld_safe：MySQL服务器启动脚本”。

· mysql.server

服务器启动脚本。该脚本用于使用包含为特定级别的运行启动服务的脚本的运行目录的系统。它调用**mysqld_safe**来启动MySQL服务器。参见5.1.4节，“mysql.server：MySQL服务器启动脚本”。

· mysqld_multi

服务器启动脚本，可以启动或停止系统上安装的多个服务器。参见5.1.5节，“mysqld_multi：管理多个MySQL服务器的程序”

· mysql_install_db

该脚本用默认权限创建MySQL授权表。通常只是在系统上首次安装MySQL时执行一次。参见2.9.2节，“Unix下安装后的过程”。

· **mysql_fix_ privilege_tables**

在升级安装后，如果新版本MySQL中的 授权表有更改，则使用该脚本来更改授权表。参见2.10.2节，“升级授权表”。

服务器主机上还运行其它几个程序：

- myisamchk

用来描述、检查、优化和维护MyISAM表的实用工具。在5.9.5节，“myisamchk：MyISAM表维护实用工具”中描述了**myisamchk**。

- make_binary_distribution

该程序可以生成编译过的MySQL的二进制版本。可以通过FTP上传到ftp.mysql.com的/pub/mysql/upload/，供其它MySQL用户使用。

- mysqlbug

MySQL 缺陷报告脚本。它可以用来向MySQL邮件系统发送缺陷报告。(你也可以访问http://bugs.mysql.com/在线创建缺陷报告文件。参见1.7.1.3节，“如何通报缺陷和问题”）。

5.1.2. mysqld-max扩展MySQL服务器

MySQL-Max服务器是**mysqld** MySQL服务器的一个版本，包含了更多的特性。

该分发版的使用取决于你的平台：

- 对于Windows，MySQL二进制分发版包括标准服务器 (mysqld.exe)和MySQL-Max服务器(**mysqld-max.exe**)，因此你不再需要专用分发版。只需要使用一个常规Windows分发版，可以从http://dev.mysql.com/downloads/获得。参见2.3节，“在Windows上安装MySQL”。

- 对于Linux，如果你使用RPM分发版安装MySQL，首先使用常规MySQL-server RPM来安装标准**mysqld**服务器。然后使用MySQL-Max RPM来安装**mysqld-max**服务器。MySQL-Max RPM假定你已经安装了常规服务器RPM。关于Linux RPM软件包的详细信息，参见2.4节，“在Linux下安装MySQL”。

- 所有其它MySQL-Max分发版包含一个**mysqld**服务器，但具有更多的特性。

你可以从MySQL AB网址http://dev.mysql.com/downloads/找到MySQL-Max二进制版本。

MySQL AB使用下面的**configure**选项构建MySQL-Max服务器：

- --with-server-suffix=-max

该选项为**mysqld**版本字符串添加一个-max后缀。

- --with-innodb

该选项启用InnoDB存储引擎支持。MySQL-Max服务器包括InnoDB支持。在MySQL 4.0及以上版本中，默认InnoDB包括在所有二进制分发版中，因此你不需要用MySQL-Max服务器只是用来获取InnoDB支持。

- --with-bdb

该选项启用Berkeley DB (BDB)存储引擎支持。

- --with-blackhole-storage-engine

该选项启用BLACKHOLE存储引擎支持。

- USE_SYMDIR

启用该定义来为Windows打开数据库符号链接支持。符号链接支持适用于所有Windows服务器，因此Max服务器不需要支持该特性。

- --with-ndbcluster

该选项启用NDB Cluster存储引擎支持。目前(5.1.2-alpha)只有Linux、Solaris和Mac OS X支持Cluster。已有一些用户报告在BSD 操作系统上成功使用了从源码构建的MySQL Cluster，但目前还没有得到官方支持。

MySQL-Max二进制分发版对于想要安装预编译程序的用户很方便。如果你使用源码分发版构建MySQL，你可以通过在配置时启用MySQL-Max二进制分发版构建所用的相同的特性来构建你自己的Max-like服务器。

MySQL-Max服务器包括BerkeleyDB (BDB)存储引擎，但并非所有平台支持BDB。

Solaris、Mac OS X和Linux(在大多数平台上)的MySQL-Max服务器包括NDB CLUSTER存储引擎支持。请注意必须用ndbcluster选项启动服务器，以便使服务器做为MySQL Cluster的一部分来运行。(详细信息参见17.4节，“MySQL簇的配置”）。

下面的表显示了MySQL-Max二进制在哪个平台上包括BDB和/或NDB CLUSTER支持：

系统	**BDB**支持	**NDB**支持
AIX 4.3	N	N
HP-UX 11.0	N	N
Linux-Alpha	N	Y
Linux-IA-64	N	N
Linux-Intel	Y	Y
Mac OS X	N	N
NetWare	N	N
SCO OSR5	Y	N
Solaris-SPARC	Y	Y

Solaris-Intel	N	Y
UnixWare	Y	N
Windows NT/2000/XP	Y	N

要想找出你的服务器支持哪个存储引擎，执行下面的语句：

```
mysql> SHOW ENGINES;
+------------+---------+------------------------------------------------------------+
| Engine     | Support | Comment                                                    |
+------------+---------+------------------------------------------------------------+
| MyISAM     | DEFAULT | Default engine as of MySQL 3.23 with great performance     |
| MEMORY     | YES     | Hash based, stored in memory, useful for temporary tables  |
| HEAP       | YES     | Alias for MEMORY                                           |
| MERGE      | YES     | Collection of identical MyISAM tables                      |
| MRG_MYISAM | YES     | Alias for MERGE                                            |
| ISAM       | NO      | Obsolete storage engine, now replaced by MyISAM            |
| MRG_ISAM   | NO      | Obsolete storage engine, now replaced by MERGE             |
| InnoDB     | YES     | Supports transactions, row-level locking, and foreign keys |
| INNOBASE   | YES     | Alias for INNODB                                           |
| BDB        | YES     | Supports transactions and page-level locking               |
| BERKELEYDB | YES     | Alias for BDB                                              |
| NDBCLUSTER | NO      | Clustered, fault-tolerant, memory-based tables             |
| NDB        | NO      | Alias for NDBCLUSTER                                       |
| EXAMPLE    | NO      | Example storage engine                                     |
| ARCHIVE    | YES     | Archive storage engine                                     |
| CSV        | NO      | CSV storage engine                                         |
| FEDERATED  | YES     | Federated MySQL storage engine                             |
| BLACKHOLE  | YES     | /dev/null storage engine (anything you write to it disappears) |
+------------+---------+------------------------------------------------------------+
18 rows in set (0.00 sec)
```

(另参见13.5.4.8节，“SHOW ENGINES语法”）。

你还可以使用下面的语句代替SHOW ENGINES，并检查你感兴趣的存储引擎的变量值：

```
mysql> SHOW VARIABLES LIKE 'have%';
+-----------------------+----------+
| Variable_name         | Value    |
+-----------------------+----------+
| have_archive          | YES      |
| have_bdb              | NO       |
| have_blackhole_engine | YES      |
| have_compress         | YES      |
| have_crypt            | YES      |
| have_csv              | YES      |
| have_example_engine   | NO       |
| have_federated_engine | NO       |
| have_geometry         | YES      |
| have_innodb           | YES      |
| have_isam             | NO       |
| have_ndbcluster       | DISABLED |
| have_openssl          | NO       |
| have_partition_engine | YES      |
| have_query_cache      | YES      |
| have_raid             | NO       |
| have_rtree_keys       | YES      |
| have_symlink          | YES      |
+-----------------------+----------+
```

```
18 rows in set (0.01 sec)
```

SHOW命令的精确输出随使用的MySQL版本(和启用的特性)的不同而有变化。第2列的值表示各特性支持的服务器级别，如下所示：

值	含义
YES	支持该特性并已经激活。
NO	不支持该特性。
DISABLED	支持该特性但被禁用。

NO值表示编译的服务器不支持该特性，因此在运行时不能激活。

出现DISABLED值是因为服务器启动时该特性被禁用，或没有给出启用它的所有选项。在后一种情况，*host_名.err*错误日志文件应包含该选项被禁用的原因。

如果服务器支持InnoDB或BDB存储引擎，你还可以看见DISABLED，但在运行启动时使用了--skip-innodb或--skip-bdb选项。对于NDB CLUSTER存储引擎，DISABLED表示服务器支持MySQL Cluster，但启动时未启用--ndb-cluster选项。

所有MySQL服务器支持MyISAM表，因为MyISAM是 默认存储引擎。

5.1.3. mysqld_safe：MySQL服务器启动脚本

在Unix和NetWare中推荐使用**mysqld_safe**来启动**mysqld**服务器。**mysqld_safe**增加了一些安全特性，例如当出现错误时重启服务器并向错误日志文件写入运行时间信息。本节后面列出了NetWare的特定行为。

注释：为了保持同旧版本MySQL的向后兼容性，MySQL二进制分发版仍然包括**safe_mysqld**作为**mysqld_safe**的符号链接。但是，你不应再依赖它，因为再将来将删掉它。

默认情况下，**mysqld_safe**尝试启动可执行**mysqld-max**（如果存在），否则启动**mysqld**。该行为的含义是：

- 在Linux中，MySQL-Max RPM依赖该**mysqld_safe**的行为。RPM安装可执行**mysqld-max**，使**mysqld_safe**从该点起自动使用可执行命令。
- 如果你安装包括**mysqld-max**服务器的MySQL-Max分发版，后面升级到非-Max的MySQL版本，**mysqld_safe**仍然试图运行旧的 **mysqld-max**服务器。升级时，你应手动删除旧的**mysqld-max**服务器以确保**mysqld_safe**运行新的**mysqld**服务器。

要想越过默认行为并显式指定你想要运行哪个服务器，为**mysqld_safe**指定--mysqld或--mysqld-version选项。

mysqld_safe的许多选项与**mysqld**的相同的。参见5.3.1节，“**mysqld**命令行选项”。

所有在命令行中为**mysqld_safe**指定的选项被传递给**mysqld**。如果你想要使用**mysqld**不支持的**mysqld_safe**的选项，不要在命令行中指定。相反，在选项文件的[mysqld_safe]组内将它们列出来。参见4.3.2节，“使用选项文件”。

mysqld_safe从选项文件的[mysqld]、[server]和[mysqld_safe]部分读取所有选项。为了保证向后兼容性，它还读取 [safe_mysqld]部分，尽管在MySQL 5.1安装中你应将这部分重新命名为[mysqld_safe]。

mysqld_safe支持下面的选项：

- --help

显示帮助消息并退出。

- --autoclose

(只在NetWare中)在NetWare中，**mysqld_safe**可以保持窗口。当你关掉**mysqld_safe** NLM时，窗口不按默认设置消失。相反，它提示用户输入：

```
*<NLM has terminated; Press any key to close the screen>*
```

如果你想让NetWare自动关闭窗口，在**mysqld_safe**中使用--autoclose选项。

- --basedir=*path*

MySQL安装目录的路径。

- --core-file-size=*size*

mysqld能够创建的内核文件的大小。选项值传递给**ulimit -c**。

- --datadir=*path*

数据目录的路径。

- --defaults-extra-file=*path*

除了通用选项文件所读取的选项文件名。如果给出，必须首选该选项。

- --defaults-file=*path*

读取的代替通用选项文件的选项文件名。如果给出，必须首选该选项。

- --ledir=*path*

包含**mysqld**程序的目录的路径。使用该选项来显式表示服务器位置。

- --log-error=*path*

将错误日志写入给定的文件。参见5.11.1节，“错误日志”。

- --mysqld=*prog_name*

想要启动的服务器程序名(在ledir目录)。如果你使用MySQL二进制分发版但有二进制分发版之外的数据目录需要该选项。

- --mysqld-version =*suffix*

该选项类似--mysqld选项，但你只指定服务器程序名的后缀。基本名假定为**mysqld**。例如，如果你使用--mysqld-version =max，**mysqld_safe**启动ledir目录中的**mysqld-max**程序。如果--mysqld-version的参数为空，**mysqld_safe**使用目录中的**mysqld**。

- --nice=*priority*

使用nice程序根据给定值来设置服务器的调度优先级。

- --no-defaults

不要读任何选项文件。如果给出，必须首选该选项。

- --open-files-limit=*count*

mysqld能够打开的文件的数量。选项值传递给 **ulimit -n**。请注意你需要用root启动**mysqld_safe**来保证正确工作!

- --pid-file=path

进程ID文件的路径。

- --port=*port_num*

用来帧听TCP/IP连接的端口号。端口号必须为1024或更大值，除非MySQL以root系统用户运行。

- --skip-character-set-client-handshake

忽略客户端发送的字符集信息，使用服务器的默认字符集。(选择该选项，MySQL的动作与MySQL 4.0相同）。

- --socket=path

用于本地连接的Unix套接字文件。

- --timezone=*zone*

为给定的选项值设置TZ时区环境变量。从操作系统文档查阅合法的时区规定格式。

- --user={*user_name* | *user_id*}

以用户名*user_name*或数字用户ID *user_id*运行**mysqld**服务器。(本文中的"用户"指系统登录账户，而不是 授权表中的MySQL用户）。

执行**mysqld_safe**时，必须先给出--defaults-file或--defaults-extra-option，或不使用选项文件。例如，该命令将不使用选项文件:

```
mysqld_safe --port=port_num --defaults-file=file_name
```

相反，使用下面的命令:

```
mysqld_safe --defaults-file=file_name --port=port_num
```

一般情况**mysqld_safe**脚本可以启动从源码或二进制MySQL分发版安装的服务器，即使这些分发版将服务器安装到稍微不同的位置。(参见2.1.5节，“安装布局”）。 **mysqld_safe**期望下面的其中一个条件是真的:

- 可以根据调用**mysqld_safe**的目录找到服务器和数据库。在二进制分发版中，**mysqld_safe**看上去在bin和data目录的工作目录下。对于源码分发版，为libexec和var目录。如果你从MySQL安装目录执行**mysqld_safe**应满足该条件(例如，二进制分发版为/usr/local/mysql)。
- 如果不能根据工作目录找到服务器和数据库，**mysqld_safe**试图通过绝对路径对它们定位。典型位置为/usr/local/libexec和/usr/local/var。实际位置由构建分发版时配置的值确定如果MySQL安装到配置时指定的位置，它们应该是正确的。

因为**mysqld_safe**试图通过工作目录找到服务器和数据库，只要你从MySQL安装目录运行**mysqld_safe**，可以将MySQL二进制分发版安装到其它位置:

```
shell> cd mysql_installation_directory
shell> bin/mysqld_safe &
```

如果**mysqld_safe**失败，即使从MySQL安装目录调用仍然失败，你可以指定--ledir和--datadir选项来指示服务器和数据库在你的系统中的安装目录。

一般情况，你不应编辑**mysqld_safe**脚本。相反，应使用命令行选项或my.cnf选项文件的[mysqld_safe]部分的选项来配置**mysqld_safe**。一般不需要编辑**mysqld_safe**来正确启动服务器。但是，如果你编辑，将来升级MySQL后会覆盖你修改的**mysqld_safe**版本，因此你应对你修改的版本进行备份以便将来重装。

在NetWare中，**mysqld_safe**是一个NetWare Loadable Module (NLM)，从原Unix shell脚本移植。它执行:

1. 检查系统和选项。
2. 检查MyISAM表。
3. 保持MySQL服务器窗口。
4. 启动并监视**mysqld**，如果因错误终止则重启。
5. 将**mysqld**的错误消息发送到数据目录中的*host_name.err* 文件。
6. 将**mysqld_safe**的屏幕输出发送到数据目录中的*host_name.safe*文件。

5.1.4. mysql.server：MySQL服务器启动脚本

在Unix中的MySQL分发版包括**mysql.server**脚本。它可以用于使用System V-style运行目录来启动和停止系统服务的系统，例如Linux和Solaris。它还用于MySQL的Mac OS X Startup Item。

mysql.server位于MySQL源码树MySQL安装目录下的support-files目录中。

如果你使用Linux 服务器RPM软件包(MySQL-server-VERSION.rpm)，**mysql.server**脚本将安装到/etc/init.d目录下，名为mysql。你不需要 手动安装。关于Linux RPM软件包的详细信息参见2.4节，“在Linux下安装MySQL”。

一些卖方提供的RPM软件包安装的启动脚本用其它名，例如**mysqld**。

如果你从不自动安装**mysql.server**的源码分发版或二进制分发版格式安装MySQL，也可以手动安装。相关说明参见2.9.2.2节，“自动启动和停止MySQL”。

mysql.server从 [mysql.server]和选项文件的[mysqld]部分读取选项。(为了保证向后兼容性，它还读取 [safe_mysqld]部分，尽管在MySQL 5.1安装中你应将这部分重新命名为[mysqld_safe]）。

5.1.5. mysqld_multi：管理多个MySQL服务器的程序

mysqld_multi可以管理多个帧听不同Unix套接字文件和TCP/IP端口的连接的**mysqld** 进程。它可以启动或停止服务器，或报告它们的当前状态。

程序寻找my.cnf中的[mysqld*N*]组(或--config-file选项指定的文件)。*N* 可以为任何正整数。在下面的讨论中该数字指选项组号，或*GNR*。组号区别各选项组，并用作**mysqld_multi**的参数来指定想要启动、停止哪个服务器或获取哪个服务器的状态报告。这些组中的选项与将用来启动**mysqld**的[mysqld]组中的相同。(例如，参见2.9.2.2节，“自动启动和停止MySQL”）。但是，当使用多个服务器时，需要每个服务器使用自己的选项值，例如Unix套接字文件和TCP/IP端口号。关于在多服务器环境中，每个服务器对应唯一选项的详细信息，参见5.12节，“在同一台机器上运行多个MySQL服务器”。

要想调用**mysqld_multi**，使用下面的语法：

```
shell> mysqld_multi [options] {start|stop|report} [GNR[,GNR] ...]
```

start、stop和report表示你想要执行的操作。你可以在单个服务器或多个服务器上执行指定的操作，取决于选项名后面的*GNR* 列。如果没有该列，**mysqld_multi**为选项文件中的所有服务器执行该操作。

每个*GNR*值代表一个选项组号或组号范围。*GNR*值应为选项文件中组名末尾的号。例如，组[mysqld17]的*GNR*为17。要想指定组号的范围，用破折号间隔开第1个和最后1个号。*GNR*值10-13代表组[mysqld10]到[mysqld13]。可以在命令行中指定多个组或组范围，用逗号间隔开。*GNR*列不能有空格字符(空格或tab)；空格字符后面的内容将被忽略掉。

该命令使用选项组[mysqld17]启动单个服务器：

```
shell> mysqld_multi start 17
```

该命令停止多个服务器，使用选项组[mysql8]和[mysqld10]至[mysqld13]：

```
shell> mysqld_multi stop 8,10-13
```

使用该命令列出设置选项文件的示例：

```
shell> mysqld_multi --example
```

mysqld_multi支持下面的选项：

- --config-file=*name*

指定选项文件名。这关系到**mysqld_multi**从哪里寻找[mysqld*N*]选项组。没有该选项，从通用my.cnf文件读所有选项。选项不影响 **mysqld_multi**从哪里读取自己的选项，总是从通用my.cnf文件的[mysqld_multi]组读取。

- --example

显示示例选项文件。

- --help

显示帮助消息并退出。

- --log=*name*

指定日志文件名。如果该文件存在，后面为日志输出。

- --mysqladmin=*prog_name*

用来停止服务器的**mysqladmin**二进制。

- --mysqld=*prog_name*

可用的**mysqld**二进制。请注意你还可以将该选项的值指定为**mysqld_safe**。选项被传递给 **mysqld**。确保在PATH环境变量设定值或**mysqld_safe**中有**mysqld**所在目录。

- --no-log

按照标准输出打印日志信息，不要写入日志文件。默认情况下，输出写入日志文件。

- --password=*password*

调用**mysqladmin**时使用的MySQL账户的密码。请注意该密码值不是可选项，不象其它MySQL程序。

- --silent

禁用警告。

- --tcp-ip

通过TCP/IP端口而不是Unix套接字文件来连接每个MySQL服务器。(如果找不到套接字文件， 服务器仍然可以运行，但只能通过 TCP/IP端口访问）。默认情况下，使用Unix套接字文件进行连接。该选项影响stop和report操作。

- --user=*user_name*

调用**mysqladmin**时使用的MySQL账户的用户名。

- --verbose

更详细。

- --version

显示版本信息并退出。

关于**mysqld_multi**的一些注解：

- 确保停止**mysqld**服务器(用**mysqladmin**程序)的MySQL账户在各个服务器中的用户名和密码相同。并且应确保账户具有SHUTDOWN权限。如果你想要管理的服务器的管理账户有许多不同的用户名或密码，你需要在每个服务器上创建一个账户，并具有相同的用户名和密码。例如，你可以执行下面的命令为每个服务器设置一个普通multi_admin账户：

```
shell> mysql -u root -S /tmp/mysql.sock -proot_password
mysql> GRANT SHUTDOWN ON *.*
    -> TO 'multi_admin'@'localhost' IDENTIFIED BY 'multipass';
```

参见5.7.2节，“权限系统工作原理”。你必须为每个**mysqld**服务器执行该操作。当连接时适当更改连接参数。请注意账户名的主机部分必须允许你用multi_admin从你想要运行**mysqld_multi**的主机进行连接。

- 如果你使用**mysqld_safe**来启动**mysqld**(例如，--mysqld=mysqld_safe)，--pid-file选项很重要。每个**mysqld**应有自己的进程ID文件。使用**mysqld_safe**而不使用**mysqld**的好处是**mysqld_safe**“守护”其**mysqld**进程，如果用kill –9发送的信号或由于其它原因（例如分段故障）进程终止，则重启进程。请注意**mysqld_safe**脚本需要你从某个位置启动它。这说明运行**mysqld_multi**前你必须进入某个目录。如果启动时有问题，请参见**mysqld_safe**脚本。特别是要检查下列行：

```
---------------------------------------------------------------
MY_PWD=`pwd`
# Check if we are starting this relative (for the binary release)
if test -d $MY_PWD/data/mysql -a -f ./share/mysql/english/errmsg.sys -a \
-x ./bin/mysqld
---------------------------------------------------------------
```

参见5.1.3节，“mysqld_safe：MySQL服务器启动脚本”。上述行执行的测试应成功，否则你可能遇到了问题。

- 每个**mysqld**的Unix套接字文件和TCP/IP端口号必须不同。

- 你可能想要为**mysqld**使用--user选项，但为此你需要用Unix root用户运行**mysqld_multi**脚本。选项文件中有选项不要紧；如果你不是超级用户，并且你用自己的Unix账户重启**mysqld**进程，你只会得到警告。

- 重要：确保**mysqld**进程启动所用Unix账户可以完全访问数据目录。不要使用Unix root账户，除非你知道你在做什么。

- 非常重要：使用**mysqld_multi**前，确保理解传递给**mysqld**服务器的选项的含义以及你为什么想要独立的**mysqld**进程。应清楚 在相同的数据目录下使用多个**mysqld**服务器的危险。使用单独的数据目录，除非你知道你在做什么。在线程系统中，在相同的数据目录下启动多个服务器不会得到超性能。参见5.12节，“在同一台机器上运行多个MySQL服务器”。

下面的示例显示了你如何设置选项文件来使用**mysqld_multi**。专门省去第1个和第5个[mysqld*N*]组来说明你的选项文件可以稍有不同。这样给你更大的灵活

性。**mysqld**程序重启或停止的顺序由它们在选项文件中的顺序决定。

```
# 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
```

参见4.3.2节，“使用选项文件”。

5.2. mysqlmanager：MySQL实例管理器

5.2.1. 用MySQL实例管理器启动MySQL服务器
5.2.2. 连接到MySQL实例管理器并创建用户账户
5.2.3. MySQL实例管理器命令行选项
5.2.4. MySQL实例管理器配置文件
5.2.5. MySQL实例管理器识别的命令

MySQL实例管理器(IM)是通过TCP/IP端口运行的后台程序，用来监视和管理MySQL数据库服务器实例。MySQL实例管理器 适合Unix-类操作系统和Windows。

可以在mysqld_safe脚本使用MySQL实例管理器来启动和停止MySQL服务器，甚至可以从一个远程主机。MySQL实例管理器还执行**mysqld_multi**脚本的功能(和大多数语法)。下面为MySQL实例管理器的详细描述。

5.2.1. 用**MySQL**实例管理器启动**MySQL**服务器

一般情况，用**mysql.server**脚本启动MySQL Database Server（MySQL数据库服务器），通常驻留在/etc/init.d/ 文件夹。默认情况下该脚本调用**mysqld_safe**脚本。但是，你可以在脚本中将use_mysqld_safe变量设置为0(零)以便使用MySQL实例管理器来启动服务器。

在这种情况下，Instance Manager的行为取决于MySQL配置文件中的选项。如果没有配置文件，MySQL实例管理器创建mysqld实例并试图用默认(编译嵌入的)配置来启动。这说明如果**mysqld**没有安装到 默认位置，IM不能猜出它的位置。如果你已经在非标准位置安装了MySQL服务器，你应使用配置文件。参见2.1.5节，“安装布局”。

如果有配置文件，IM将分析配置文件搜索[mysqld]部分(例如[mysqld]、[mysqld1]、[mysqld2]等)。每个部分指定一个实例。启动时IM将启动所有找到的实例。IM关闭时默认停止所有实例。

请注意有一个特殊选项mysqld-path(mysqld-path = *path-to-mysqld- binary*)，只能用IM识别。使用该变量让IM知道**mysqld**二进制驻留在哪儿。你还应该为服务器设置basedir和datadir选项。

启用MySQL实例管理器的典型MySQL服务器启动/关闭循环为:

- 用**/etc/init.d/mysql**脚本启动MySQL实例管理器。
- MySQL实例管理器启动所有实例并监视它们。
- 如果某个服务器实例失败，MySQL实例管理器重启它。
- 如果MySQL实例管理器被关闭(例如用**/etc/init.d/mysql stop**命令)，所有实例被MySQL实例管理器关闭。

5.2.2. 连接到**MySQL**实例管理器并创建用户账户

使用MySQL客户端-服务器协议来处理同MySQL实例管理器之间的通信。你不能使用标准**mysql**客户端程序和MySQL C API来连接IM。IM支持客户端工具和mysql-4.1或以后的版本所分发的库所用的MySQL客户端-服务器协议版本。

IM将用户信息保存到密码文件中。密码文件的默认位置为/etc/mysqlmanager.passwd。

密码应类似于:

```
petr:*35110DC9B4D8140F5DE667E28C72DD2597B5C848
```

要想生成密码用**--passwd**选项调用IM。则输出可以重定向到/etc/mysqlmanager.passwd文件以添加新用户。下面为示例命令。

```
./mysqlmanager --passwd >> /etc/mysqlmanager.passwd
Creating record for new user.
Enter user name: mike
Enter password: <password>
Re-type password: <password>
```

下面的行将加到/etc/mysqlmanager.passwd:

```
mike:*00A51F3F48415C7D4E8908980D443C29C69B60C9
```

如果/etc/mysqlmanager.passwd文件中没有该条，则不能连接IM。

5.2.3. **MySQL**实例管理器命令行选项

MySQL实例管理器支持许多命令行选项。执行**./mysqlmanager --help**命令可以简单列出。有下面的选项:

- --help，-?

显示帮助消息并退出。

- --bind-address=*name*

绑定地址用于连接。

- --default-mysqld-path=*name*

在Unix中，如果实例部分没有路径，则为寻找MySQL服务器二进制的地点。例如：default-mysqld-path = /usr/sbin/mysqld

- --defaults-file=*file_name*

从给定文件读Instance Manager和MySQL服务器设定值。所有Instance Manager更改的配置将加入该文件。只能用于Instance Manager的第一选项。

- --install

在Windows中，将Instance Manager安装为Windows服务。

- --log=*name*

IM日志文件的路径。结合**--run-as-service**选项使用。

- --monitoring-interval=*Seconds*

监视实例的间隔，单位为秒。Instance Manager将尝试连接每个监视的实例来检查它们是否是活动的/没有挂起。出现故障，IM将重启几次(实际上是多次)实例。可以用nonguarded选项为特定实例禁用该行为。如果未给定任何值， 默认使用20秒。

- --passwd，-P

编写passwd文件并退出。

- --password-file=*name*

从该文件中寻找Instance Manager用户和密码。默认文件是/etc/mysqlmanager.passwd。

- --pid-file=*name*

使用的进程ID文件。默认情况下，该文件文件名为mysqlmanager.pid。

- -- port=*port_num*

用于连接的端口号。(IANA分配的 默认端口号为2273)。

- --print-defaults

打印当前的默认值并退出。只能用作Instance Manager的第一选项。

- --remove

在Windows中，删掉Instance Manager Windows服务。假定前面已经用--install运行了Instance Manager。

- --run-as-service

使完善进程变为后台程序并启动。完善进程很简单，不易崩溃。出现故障后它将自己重启IM。

- --socket=*name*

Unix中用于连接的套接字文件。默认情况下，文件名为/tmp/mysqlmanager.sock。

- --standalone

在Windows中以单机模式运行Instance Manager。

- --user=*name*

启动并运行**mysqlmanager**的用户名。建议使用运行**mysqld**服务器的用户账户来运行**mysqlmanager**。

- --version, -V

输出版本信息并退出。

5.2.4. MySQL实例管理器配置文件

Instance Manager使用标准my.cnf文件。它使用[manager]部分为自己读取选项并读取[mysqld]部分来创建实例。[manager]部分包含上述列出的选项。下面为[manager]部分示例:

```
# MySQL Instance Manager options section
[manager]
default-mysqld-path = /usr/local/mysql/libexec/mysqld
socket=/tmp/manager.sock
pid-file=/tmp/manager.pid
password-file = /home/cps/.mysqlmanager.passwd
monitoring-interval = 2
port = 1999
bind-address = 192.168.1.5
```

MySQL实例管理器只在Unix中读取并管理/etc/my.cnf文件。在Windows中，MySQL实例管理器从Instance Manager的安装目录读取my.ini文件。用--defaults-file=*file_ name*选项可以更改默认选项文件的位置。

实例部分指定启动时给每个实例的选项。这些主要是普通MySQL服务器选项，但有一些IM-专用选项:

- mysqld-path = <path-to-mysqld-binary>

mysqld服务器二进制的路径。

- shutdown-delay = *Seconds*

IM应等待实例关闭的秒数。 默认为35秒。超过延迟时间后，IM假定实例正挂起并试图"kill –9"它。如果你使用带large表的InnoDB，你应当增加该值。

- nonguarded

如果你想要为某个实例禁用IM监视功能，应设置该选项。

下面给出了几个实例示例。

```
[mysqld]
mysqld-path=/usr/local/mysql/libexec/mysqld
socket=/tmp/mysql.sock
port=3307
server_id=1
skip-stack-trace
core-file
skip-bdb
log-bin
log-error
log=mylog
log-slow-queries

[mysqld2]
nonguarded
port=3308
server_id=2
mysqld-path= /home/cps/mysql/trees/mysql-5.1/sql/mysqld
socket     = /tmp/mysql.sock5
pid-file   = /tmp/hostname.pid5
datadir= /home/cps/mysql_data/data_dir1
language=/home/cps/mysql/trees/mysql-5.1/sql/share/english
log-bin
```

log=/tmp/fordel.log

5.2.5. MySQL实例管理器识别的命令

一旦你已经为MySQL实例管理器设置了一个密码文件并且IM正在运行，你可以连接它。你可以使用**mysql**客户端工具通过标准MySQL API来连接。以下是MySQL实例管理器目前可以接收的命令的清单和例子。

- START INSTANCE <instance_name>

该命令将试图启动一个实例:

```
mysql> START INSTANCE mysqld4;
Query OK, 0 rows affected (0,00 sec)
```

- STOP INSTANCE <instance_name>

将试图停止一个实例:

```
mysql> STOP INSTANCE mysqld4;
Query OK, 0 rows affected (0,00 sec)
```

- SHOW INSTANCES

显示所有载入的实例名:

```
mysql> show instances;
+---------------+---------+
| instance_name | status  |
+---------------+---------+
| mysqld3       | offline |
| mysqld4       | online  |
| mysqld2       | offline |
+---------------+---------+
3 rows in set (0,04 sec)
```

- SHOW INSTANCE STATUS <instance_name>

显示选定的实例的状态和版本信息:

```
mysql> SHOW INSTANCE STATUS mysqld3;
+---------------+--------+---------+
| instance_name | status | version |
+---------------+--------+---------+
| mysqld3       | online | unknown |
+---------------+--------+---------+
1 row in set (0.00 sec)
```

- SHOW INSTANCE OPTIONS <instance_name>

显示实例使用的选项:

```
mysql> SHOW INSTANCE OPTIONS mysqld3;
+---------------+--------------------------------------------------+
| option_name   | value                                            |
+---------------+--------------------------------------------------+
| instance_name | mysqld3                                          |
| mysqld-path   | /home/cps/mysql/trees/mysql-4.1/sql/mysqld       |
| port          | 3309                                             |
| socket        | /tmp/mysql.sock3                                 |
| pid-file      | hostname.pid3                                    |
| datadir       | /home/cps/mysql_data/data_dir1/                  |
| language      | /home/cps/mysql/trees/mysql-4.1/sql/share/english |
+---------------+--------------------------------------------------+
7 rows in set (0.01 sec)
```

- SHOW <instance_name> LOG FILES

该命令提供实例使用的所有日志文件。结果包含日志文件的路径和日志文件的大小。如果没有在配置文件中指定日志文件的路径 (例如log=/var/mysql.log)，IM试图猜出它的位置。如果IM不能猜出日志文件的位置，你应明确指定日志文件的位置。

```
mysql> SHOW mysqld LOG FILES;
+-------------+-----------------------------------+----------+
| Logfile     | Path                              | Filesize |
+-------------+-----------------------------------+----------+
| ERROR LOG   | /home/cps/var/mysql/owlet.err     | 9186     |
| GENERAL LOG | /home/cps/var/mysql/owlet.log     | 471503   |
| SLOW LOG    | /home/cps/var/mysql/owlet-slow.log | 4463     |
+-------------+-----------------------------------+----------+
3 rows in set (0.01 sec)
```

- SHOW <instance_name> LOG {ERROR | SLOW | GENERAL} size[,offset_from_end]

该命令查找指定的日志文件的一部分。因为大多数用户关注最新的日志消息，用size参数定义你想要从日志末尾开始索取的字节数。你可以指定可选offset_from_end参数从日志文件中部索取数据。下面的示例可以索取21个字节的数据，从日志文件末尾开始23个字节，2个字节用于结束:

```
mysql> SHOW mysqld LOG GENERAL 21, 2;
+---------------------+
| Log                 |
+---------------------+
| using password: YES |
+---------------------+
1 row in set (0.00 sec)
```

- SET instance_name.option_name=option_value

该命令编辑指定的实例的配置文件以更改/增加实例选项。IM假定配置文件位于/etc/my.cnf。你应检查文件是否存在，并具有合适的权限。

```
mysql> SET mysqld2.port=3322;
Query OK, 0 rows affected (0.00 sec)
```

在MySQL服务器重启前，对配置文件进行的更改不会生效。并且，执行FLUSH INSTANCES命令后，才会将这些更改保存到Instance Manager的实例设定值的本地缓存中。

- UNSET instance_name.option_name

该命令从实例的配置文件删除一个选项。

```
mysql> UNSET mysqld2.port;
Query OK, 0 rows affected (0.00 sec)
```

在MySQL服务器重启前，对配置文件进行的更改不会生效。并且，执行FLUSH INSTANCES命令后，才会将这些更改保存到Instance Manager的实例设定值的本地缓存中。

- FLUSH INSTANCES

该命令强制IM重读配置文件并清空内部结构。编辑配置文件后应执行该命令。该命令不重启实例：

```
mysql> FLUSH INSTANCES;
Query OK, 0 rows affected (0.04 sec)
```

5.3. mysqld：MySQL服务器

5.3.1. **mysqld**命令行选项
5.3.2. SQL服务器模式
5.3.3. 服务器系统变量
5.3.4. 服务器状态变量

mysqld是MySQL服务器。下面讨论MySQL服务器的配置：

- 服务器支持的启动选项
- 如何设置服务器SQL模式
- 服务器系统变量
- 服务器状态变量

5.3.1. mysqld命令行选项

当启动**mysqld**服务器时，你可以使用4.3节，“指定程序选项”中描述的方法指定程序选项。最常用的方法是在选项文件中或在命令行提供选项。但是，在大多数情况下，希望服务器每次运行时使用相同的选项。最好的方法是确保将它们列在选项文件中。参见4.3.2节，“使用选项文件”。

mysqld从[mysqld]和[server]组读取选项。**mysqld_safe**从[mysqld]、[server]、[mysqld_safe]和[safe_mysqld]组读取选项。**mysql.server**从[mysqld]和[mysql.server]组读取选项。嵌入式MySQL服务器通常从[server]、[embedded]和[*xxxxx_SERVER*]组读取选项，其中*xxxxx*是服务器嵌入的应用程序名。

mysqld接受许多命令行选项。执行**mysqld --help**可以简单列出来。要想看全部列表列，使用命令**mysqld --verbose --help**。

下面列出了一些最常用的服务器选项。其它的选项在其它地方描述：

- 影响安全的选项：参见5.6.3节，“Mysqld安全相关启动选项”。
- SSL-相关选项：参见5.8.7.6节，“SSL命令行选项”。
- 二进制日志控制选项：参见5.11.3节，“二进制日志”。
- 复制相关选项：参见6.8节，“复制启动选项”。
- 特定存储引擎相关选项：参见15.1.1节，“MyISAM启动选项”、15.5.3节，“BDB启动选项”和15.2.4节，“InnoDB启动选项”。

你还可以将变量名作为一个选项，设置服务器系统变量的值，如本节后面所述。

- --help，-?

显示简短的帮助消息并退出。使用--verbose和--help选项来看全部内容。

- --allow-suspicious-udfs

该选项控制是否用户定义的函数只有一个xxx符，用作可载入的主函数。默认情况下，该选项被关闭，只有至少有一个附属符的UDF 可以载入。这样可以防止从未包含合法UDF的共享文件装载函数。参见27.2.3.6节，“用户定义函数安全注意事项”。

- --ansi

使用标准(ANSI)SQL语法代替MySQL语法。参见1.8.3节，“在ANSI模式下运行MySQL”。使用--sql-mode选项可以更精确控制服务器SQL模式。

- --basedir=*path*, -b *path*

MySQL安装目录的路径。通常所有路径根据该路径来解析。

- --bind-address=*IP*

待绑定的IP地址。

- --bootstrap

mysql_install_db脚本使用该选项来创建MySQL授权表，不需要启动MySQL服务器。

- --console

将错误日志消息写入stderr和stdout，即使指定了--log-error。在Windows中，如果使用该选项，**mysqld**不关闭控制台窗口。

- --character-sets-dir=*path*

字符集安装的目录。参见5.10.1节，“数据和排序用字符集”。

- --chroot=path

通过chroot()系统调用在启动过程中将**mysqld**服务器放入一个封闭环境中。这是推荐的一个安全措施。请注意使用该选项可以 限制LOAD DATA INFILE和SELECT ... INTO OUTFILE。

- --character-set-server=*charset*

使用*charset*作为 默认服务器字符集。参见5.10.1节，“数据和排序用字符集”。

- --core-file

如果**mysqld**终止，写内核文件。在某些系统中，你还必须为**mysqld_safe**指定--core-file-size 选项。参见5.1.3节，“mysqld_safe：MySQL服务器启动脚本”。请注意对于一些系统，例如Solaris，如果你使用--user选项不会获得内核文件。

- --collation-server=*collation*

使用*collation*作为 默认服务器校对规则。参见5.10.1节，“数据和排序用字符集”。

- --datadir=*path*, -h *path*

数据目录的路径。

- --debug[=*debug_options*], -# [*debug_options*]

如果MySQL配置了--with-debug，你可以使用该选项来获得一个跟踪文件，跟踪**mysqld**正进行的操作。*debug_options*字符串通常为'd:t:o，*file_name*'。参见E.1.2节，“创建跟踪文件”。

- (*DEPRECATED*) --default-character-set=*charset*

使用*char*设置作为 默认字符集。由于--character-set-server，反对使用该选项。参见5.10.1节，“数据和排序用字符集”。

- --default-collation=*collation*

使用*collation* 作为默认校对规则。由于--collation-server，反对使用该选项。参见5.10.1节，“数据和排序用字符集”。

- --default-storage-engine=*type*

该选项为--default-table-type的同义词。

- --default-table-type=*type*

设置表的默认类型。参见第15章：存储引擎和表类型。

- --default-time-zone=*type*

设置默认服务器时区。该选项设置全局time_zone系统变量。如果未给出该选项， 默认时区与系统时区相同(用system_time_zone系统变量值给定)。

- --delay-key-write[= OFF | ON | ALL]

如何使用DELAYED KEYS选项。键写入延迟会造成再次写MyISAM表时键缓冲区不能被清空。OFF禁用延迟的键写入。ON启用用DELAYED KEYS选项创建的表的延迟的键写入。ALL延迟所有MyISAM表的键写入。参见7.5.2节，“调节服务器参数”。参见15.1.1节，“MyISAM启动选项”。

注释：如果你将该变量设置为ALL，你不应从另一个正使用MyISAM表的程序中使用MyISAM表(例如从另一个MySQL服务器或用**myisamchk**)。这样操作会导致索引破坏。

- --des-key-file=*file_name*

从该文件读DES_ENCRYPT()和DES_DECRYPT()使用的 默认键。

- --enable-named-pipe

启用命名管道支持。该选项只适用Windows NT、2000、XP和2003系统，并且只适用支持命名管道连接的**mysqld-nt**和**mysqld-max-nt**服务器。

- --exit-info[=*flags*], -T [*flags*]

这是不同标志的一个位掩码，你可以用来调试**mysqld**服务器。不要使用该选项，除非你确切知道它在做什么！

· --external-locking

启用系统锁定。请注意如果你在lockd不能完全工作的系统上使用该选项(例如在Linux中)，**mysqld**容易死锁。该选项以前叫--enable-locking。

注释：如果你在许多MySQL进程中使用该选项来更新MyISAM表，你必须确保满足下述条件：

o 使用正被另一个进程更新的表的查询的缓存不可使用。

o 不应在共享表中使用--delay-key-write=ALL或DELAY_KEY_WRITE=1。

最简单的方法是结合使用--external-locking和--delay-key-write=OFF --query-cache-size=0。

(默认不能实现，因为在许多设置中，结合使用上述选项很有用）。

· --flush

执行SQL语句后向硬盘上清空更改。一般情况执行SQL语句后 MySQL向硬盘写入所有更改，让操作系统处理与硬盘的同步。参见A.4.2节，“如果MySQL依然崩溃，应作些什么”。

· --init-file=*file*

启动时从该文件读SQL语句。每个语句必须在同一行中并且不应包括注释。

· --language=*lang_name*, -L *lang_name*

用给定语言给出客户端错误消息。*lang_name*可以为语言名或语言文件安装目录的全路径名。参见5.10.2节，“设置错误消息语言”。

· --large-pages

一些硬件/操作系统架构支持大于 默认值(通常4 KB)的内存页。实际支持取决于使用的硬件和OS。大量访问内存的应用程序通过使用较大的页，降低了Translation Lookaside Buffer (TLB)损失，可以改善性能。

目前，MySQL只在Linux中支持大页面(在Linux中被称作HugeTLB)。我们已经计划将该支持扩展到FreeBSD、Solaris和其它可能的平台。

在Linux中可以使用大页面前，需要配置HugeTLB内存池。参考Linux内核源码中的hugetlbpage.txt文件。

默认情况下该选项被禁用。

· ---log[=*file*], -l [*file*]

日志连接和对文件的查询。参见5.11.2节，“通用查询日志”。如果你不指定文件名，MySQL使用*host_name*.log作为文件名。

· --log-bin=[*file*]

二进制日志文件。将更改数据的所有查询记入该文件。用于备份和复制。参见5.11.3节，“二进制日志”。建议指定一个文件名(原因参见A.8.1节，“MySQL中的打开事宜”)，否则MySQL使用*host_name*-bin作为日志文件基本名。

· --log-bin-index[=*file*]

二进制日志文件名的索引文件。参见5.11.3节，“二进制日志”。如果你不指定文件名，并且如果你没有在--log-bin中指定，MySQL使用*host_name*-bin.index作为文件名。

· --log-bin-trust-routine-creators[={0|1}]

没有参数或参数为1,该选项将系统变量log_bin_trust_routine_creators设置为1。为参数 0时，该选项将系统变量设置为0。log_bin_trust_routine_creators影响MySQL如何对保存的程序的创建强加限制。参见20.4节，“存储子程序和触发程序的二进制日志功能”。

· --log-error[=*file*]

该文件的日志错误和启动消息。参见5.11.1节，“错误日志”。如果你不指定文件名，MySQL使用*host_name.err*作为文件名。如果文件名没有扩展名，则加上.err扩展名。

· --log-isam[=*file*]

将所有MyISAM更改记入该文件(只有调试MyISAM时才使用)。

· (*DEPRECATED*) --log-long-format

记录激活的更新日志、二进制更新日志、和慢查询日志的大量信息。例如，所有查询的用户名和时间戳将记录下来。不赞成选用该选项，因为它现在代表 默认记录行为。(参见--log-short-format描述）。--log-queries-not-using-indexes选项适合将未使用索引的查询记录到慢查询日志中。

· --log-queries-not-using-indexes

如果你结合--log-slow-queries使用该选项，未使用索引的查询也被记录到慢查询日志中。参见5.11.4节，“慢速查询日志”。

· --log-short-format

记录激活的更新日志、二进制更新日志、和慢查询日志的少量信息。例如，用户名和时间戳不记录下来。

- ---log-slow-admin-statements

将慢管理语句例如OPTIMIZE TABLE、ANALYZE TABLE和ALTER TABLE记入慢查询日志。

- --log-slow-queries[=*file*]

将所有执行时间超过long_query_time 秒的查询记入该文件。参见5.11.4节，“慢速查询日志”。详细信息参见--log-long-format和--log-short-format选项描述。

- --log-warnings, -W

将警告例如Aborted connection...打印到错误日志。建议启用该选项，例如，如果你使用复制 (你可以得到关于所发生事情的详细信息，例如关于网络故障和重新连接的消息)。默认情况下启用该选项；要想禁用它，使用--skip-log-warnings。中断的连接不会记入错误日志，除非值大于1。参见A.2.10节，“通信错误和失效连接”。

- --low-priority-updates

表修改(INSERT, REPLACE, DELETE, UPDATE)比选择的优先级要低。也可以通过{INSERT | REPLACE | DELETE | UPDATE} LOW_PRIORITY ... 来降低某个查询的优先级来实现，或通过SET LOW_PRIORITY_UPDATES=1来更改一个线程的优先级。参见7.3.2节，“表锁定事宜”。

- --memlock

将**mysqld** 进程锁定在内存中。在支持mlockall()系统调用的系统上有效，例如Solaris。如果操作系统使**mysqld**在硬盘上交换时出现问题，可以为你提供帮助。请注意使用该选项时需要以root运行服务器，但从安全考虑并不是一个好注意。

- --myisam-recover [=*option*[,*option*...]]]

将存储引擎MyISAM设置为恢复模式。该选项值是DEFAULT、BACKUP、FORCE或QUICK值的任何组合。如果你指定多个值，用逗号间隔开。你还可以使用""值来禁用该选项。使用如果该选项，当**mysqld**打开MyISAM表时，检查是否表标记为崩溃或没有正确关闭。(只有用--skip-external-lockingare运行时，最后的选项才工作）。 如果是这种情况，**mysqld**则检查 表。如果表被破坏，**mysqld**试图维护它。

下面的选项影响维护工作:

选项	描述
DEFAULT	与没有使用--myisam-recover选项相同。
BACKUP	如果在恢复过程中，数据文件被更改了，将*tbl_name*.MYD文件备份为*tbl_name-datetime*.BAK。
FORCE	即使.MYD文件将丢掉多个行也进行恢复。
QUICK	如果没有删除块，不要检查表中的行。

在表自动修复前，MySQL错误日志添加一条注解。如果你不想用户干涉干涉大多数问题，你应使用BACKUP,FORCE选项。该选项强制维护表，即使一些行将会被删除也不例外，但它保持旧的数据文件做为备份，以便你可以在后来进行检查。

- --ndb-connectstring=*connect_string*

当使用NDB存储引擎时，可以指出通过设置连接字符串选项来分发群集配置的管理服务器。相关语法参见17.4.4.2节，“MySQL簇连接字符串”。

- --ndbcluster

如果二进制支持NDB CLUSTER存储引擎，使用该选项可以代替禁用MySQL Cluster支持的 默认设置。参见第17章：*MySQL*簇。

- --old-passwords

强制服务器为新密码生成短(4.1前)密码哈希。如果服务器必须支持旧客户端程序，为保证兼容性这很有用。参见5.7.9节，“MySQL 4.1中的密码哈希处理”。

- --one-thread

只使用一个线程(用于在Linux中调试)。只有服务器启用了调试，该选项才可用。参见E.1节，“调试MySQL服务器”。

- --open-files-limit=*count*

用来更改**mysqld**文件描述符的数量。如果没有设置或设置为0，则**mysqld**通过setrlimit()使用该值来保存文件描述符。如果该值为0，则**mysqld** 保存max_connections*5或max_connections + table_cache*2(取较大者)个文件。如果**mysqld**给出你错误"打开的文件太多。"，你应试试增加该值。

- --pid-file=path

mysqld_safe使用的进程ID文件的路径。

- --port=*port_num*, -P *port_num*

帧听TCP/IP连接时使用的端口号。

- --safe-mode

跳过一些优化阶段。

- (*DEPRECATED*) --safe-show-database

参见5.7.3节，“MySQL提供的权限”。

- --safe-user-create

启用后如果用户没有mysql.user表或表中列的INSERT权限，则用户不能用GRANT语句创建新用户。

- --secure-auth

不允许使用旧(4.1之前)密码的账户进行鉴定。

- --shared-memory

启用本地客户端的共享内存连接。该选项只用于Windows。

- --shared-memory-base-name=*name*

共享内存连接名。该选项只用于Windows。

- --skip-bdb

禁用BDB存储引擎。这样可以节省内存，并可能加速某些操作。如果你需要BDB表则不要使用该选项。

- --skip-concurrent-insert

关闭在同一时间在MyISAM表中选择和插入的能力。(只有你发现缺陷时才使用该选项）。

- --skip-external-locking

不要使用系统锁定。要想使用**myisamchk**，你必须关闭服务器。(参见1.4.3节，“MySQL稳定性”）。 要避免该需求，使用MySQL Monitor中的CHECK TABLE和REPAIR TABLE来检查并维护MyISAM表。

- --skip-grant-tables

该选项使服务器不使用权限系统。该权限允许访问服务器的用户不受限制地访问所有数据库。你可以从系统外壳命令行执行**mysqladmin flush-privileges**或**mysqladmin reload**命令，或执行MySQL FLUSH PRIVILEGES语句让运行的服务器重新开始使用 授权表。

- --skip-host-cache

为了更快地在名称-IP之间进行解析，不要使用内部主机名缓存。相反，每次客户端连接时查询DNS服务器。参见7.5.6节，“MySQL如何使用DNS”。

- --skip-innodb

禁用InnoDB存储引擎。这样可以节省内存，并可能加速某些操作。如果你需要BDB表则不要使用该选项。

- --skip-name-resolve

不要解析正检查客户端连接的主机名。只使用IP号。如果你使用该项， 授权表中的所有Host列值必须为IP号或localhost。参见7.5.6节，“MySQL如何使用DNS”。

- --skip-ndbcluster

禁用NDB CLUSTER存储引擎。这是支持NDB CLUSTER存储引擎的二进制的默认设置，说明只有用--ndbcluster选项显式覆盖--skip-ndbcluster选项时，系统才为该存储引擎分配内存和其它资源。使用示例参见17.4.3节，“MySQL簇的快速测试设置”。

- --skip-networking

不帧听TCP/IP连接。必须通过命名管道或共享内存(在Windows中)或Unix套接字文件(在Unix中)完成**mysqld**的相互操作。对于只允许本地客户端的系统，大力推荐该选项。参见7.5.6节，“MySQL如何使用DNS”。

- --standalone

只适合基于Windows-NT的系统；指导MySQL服务器不做为服务来运行。

- --symbolic-links, --skip-symbolic-links

启用或禁用符号链接支持。在Windows和Unix中，该选项的作用是不同的：

o 在Windows中，启用符号链接，你可以通过创建包含实际目录路径的directory.sym文件来建立数据库目录的符号链接。参见7.6.1.3节，“在Windows平台上使用关于数据库的符号链接”。

在Unix中，启用符号链接表示你可以用CREATE TABLE语句的INDEX DIRECTORY或DATA DIRECTORY选项将MyISAM索引文件或数据文件链接到另一个目录。如果你删除或重新命名表，符号链接所指的文件也被删除或重新命名。参见13.1.5节，“CREATE TABLE语法”。

- --skip-safemalloc

如果MySQL配置了--with-debug=full，所有MySQL程序在内存分配和释放时检查内存是否溢出。检查很慢，因此如果你不需要你可以用--skip-safemalloc选项来避免。

- --skip-show-database

使用该选项，只允许具有SHOW DATABASES权限的用户执行SHOW DATABASES语句，该语句显示所有数据库名。不使用该选项，允许所有用户执行SHOW

DATABASES，但只向具有SHOW DATABASES权限或部分数据库权限的用户显示每个数据库名。请注意全局权限为数据库的一种权限。

· --skip-stack-trace

不跟踪写堆栈。当调试运行**mysqld**时该选项有用。在一些系统中，你还必须使用该选项来获得内核文件。参见E.1节，“调试MySQL服务器”。

· --skip-thread-priority

在快速响应中禁用线程优先级。

· --socket=path

在Unix中，该选项指定用于本地连接的Unix套接字文件。 默认值是/tmp/mysql.sock。在Windows中，该选项指定本地连接所使用的管道名。 默认值是MySQL。

· --sql-mode=*value*[,*value*[,*value*...]]

将MySQL设置为SQL模式。参见5.3.2节，“SQL服务器模式”。

· --temp-pool

该选项使服务器创建的大多数临时文件使用一系列文件名，而不是每个新文件使用唯一的文件名。这样解决了在Linux内核中用 不同的名创建许多新文件的问题。在以前，Linux似乎“泄漏”内存，因为它被直接分配到directory entry缓存而不是硬盘缓存。

· --transaction-isolation=*level*

设置默认事务隔离级别，可以READ-UNCOMMITTED、READ-COMMITTEE、REPEATABLE-READ或SERIALIZABLE。参见13.4.6节，“SET TRANSACTION语法”。

· --tmpdir=*path*, -t *path*

创建临时文件的目录路径。默认/tmp目录在太小不能容纳临时表的分区时该选项很有用。该选项接受round-robin模式的几个路径。在Unix中路径应用冒号(':') 间隔开，在Windows、NetWare和OS/2中用分号(`; ') 间隔开。如果MySQL服务器为复制从机，你不应让--tmpdir指向基于内存的文件系统中的目录或服务器主机重启时会清除的目录。复制从机需要临时文件，机器重启时可以复制临时表或执行LOAD DATA INFILE操作。如果服务器重启时临时文件目录中的文件丢失，复制失败。

· --user={*user_name* | *user_id*}, -u {*user_name* | *user_id*}

用*user_name*或数字用户ID *user_id*运行**mysqld**服务器。(“用户”指系统登录账户，而不是 授权表中所列的MySQL用户）。

用root启动**mysqld**时强制使用该选项。服务器在启动序列中更改用户ID，让它做为具体用户而不是root运行。参见5.6.1节，“通用安全指南”。

要避免用户在my.cnf文件中添加--user=root选项(使服务器用root运行)时可能出现的安全漏洞，**mysqld**只使用指定的第1个--user选项，如果有多个--user选项则会出现警告。在命令行选项前处理/etc/my.cnf和$MYSQL_HOME/my.cnf中的选项，因此建议你在/etc/my.cnf中放一个--user选项，并指定root之外的其它值。在其它--user选项前先找到/etc/my.cnf中的选项，确保服务器用其它用户运行，如果找到其它--user选项则会出现警告。

· --version, -V

显示版本信息并退出。

你可以使用--*var_name*=*value*形式的选项为服务器系统变量。例如，--key_buffer_size=32M将变量key_buffer_size设为32MB。

请注意设置变量时，MySQL可以自动将它纠正到某个给定范围内，或如果只允许某个值，则将设置值调节到最接近允许的值。

还可以通过--set-variable=*var_name*=*value*或-O *var_name*=*value*语法来设置变量。但是，现在不赞成使用该语法。

5.3.3节，“服务器系统变量”完全描述了全部系统变量。调节服务器参数部分包括如何对他们进行优化。参见7.5.2节，“调节服务器参数”。

你可以用SET语句更改大多数服务器系统变量的值。参见13.5.3节，“SET语法”。

如果你想用SET 限制启动项可设的最大值，你可以使用--maximum-var_name命令行选项来定义。

5.3.2. SQL服务器模式

MySQL服务器可以以不同的SQL模式来操作，并且可以为不同客户端应用不同模式。这样每个应用程序可以根据自己的需求来定制服务器的操作模式。

模式定义MySQL应支持哪些SQL语法，以及应执行哪种数据验证检查。这样可以更容易地在不同的环境中使用MySQL，并结合其它数据库服务器使用MySQL。

你可以用--sql-mode="*modes*"选项启动**mysqld**来设置默认SQL模式。如果你想要重设，该值还可以为空(--sql-mode =“”)。

你还可以在启动后用SET [SESSION|GLOBAL] sql_mode='*modes*'语句设置sql_mode变量来更改SQL模式。设置 GLOBAL变量时需要拥有SUPER权限，并且会影响从那时起连接的所有客户端的操作。设置SESSION变量只影响当前的客户端。任何客户端可以随时更改自己的会话 sql_mode值。

*Modes*is是用逗号(`, ')间隔开的一系列不同的模式。你可以用SELECT @@sql_mode语句查询当前的模式。默认值是空(没有设置任何模式)。

主要重要sql_mode值为:

· ANSI

更改语法和行为，使其更符合标准SQL。

· STRICT_TRANS_TABLES

如果不能将给定的值插入到事务表中，则放弃该语句。对于非事务表，如果值出现在单行语句或多行语句的第1行，则放弃该语句。本节后面给出了更详细的描述。

· TRADITIONAL

Make MySQL的行为象"传统"SQL数据库系统。该模式的简单描述是当在列中插入不正确的值时"给出错误而不是警告"。注释：一旦发现错误立即放弃INSERT/UPDATE。如果你使用非事务存储引擎，这种方式不是你想要的，因为出现错误前进行的数据更改不会"滚动"，结果是更新"只进行了一部分"。

本手册指"严格模式"，表示至少STRICT _TRANS_TABLES或STRICT _ALL_TABLES被启用的模式。

下面描述了支持的所有模式：

· ALLOW_INVALID_DATES

在严格模式下不要检查全部日期。只检查1到12之间的月份和1到31之间的日。这在Web应用程序中，当你从三个不同的字段获取年、月、日，并且想要确切保存用户插入的内容(不进行日期验证)时很重要。该模式适用于DATE和DATETIME列。不适合TIMESTAMP列，TIMESTAMP列需要验证日期。

启用严格模式后，服务器需要合法的月和日，不仅仅是分别在1到12和1到31范围内。例如，禁用严格模式时'2004-04-31'是合法的，但启用严格模式后是非法的。要想在严格模式允许遮掩固定日期，还应启用ALLOW_INVALID_DATES。

· ANSI_QUOTES

将""视为识别符引号(``引号字符)，不要视为字符串的引号字符。在ANSI模式，你可以仍然使用``来引用识别符。启用ANSI_QUOTES后，你不能用双引号来引用字符串，因为它被解释为识别符。

· ERROR_FOR_DIVISION_BY_ZERO

在严格模式，在INSERT或UPDATE过程中，如果被零除(或MOD(X，0))，则产生错误(否则为警告)。如果未给出该模式，被零除时MySQL返回NULL。如果用到INSERT IGNORE或UPDATE IGNORE中，MySQL生成被零除警告，但操作结果为NULL。

· HIGH_NOT_PRECEDENCE

NOT操作符的优先顺序是表达式例如NOT a BETWEEN b AND c被解释为NOT (a BETWEEN b AND c)。在一些旧版本MySQL中， 表达式被解释为(NOT a) BETWEEN b AND c。启用HIGH_NOT_PRECEDENCESQL模式，可以获得以前的更高优先级的结果。

```
mysql> SET sql_mode = '';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
        -> 0
mysql> SET sql_mode = 'broken_not';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
        -> 1
```

· IGNORE_SPACE

允许函数名和'('之间有空格。强制将所有函数名视为保存的字。结果是，如果你想要访问保存为字的数据库、表或列名，你必须引用它。例如，因为有USER()函数，mysql数据库中的user表名和该表内的User列被保存下来，因此你必须引用它们：

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

· NO_AUTO_CREATE_USER

防止GRANT自动创建新用户，除非还指定了密码。

· NO_AUTO_VALUE_ON_ZERO

NO_AUTO_VALUE_ON_ZERO影响AUTO_INCREMENT列的处理。一般情况，你可以向该列插入NULL或0生成下一个序列号。NO_AUTO_VALUE_ON_ZERO禁用0，因此只有NULL可以生成下一个序列号。

如果将0保存到表的AUTO_INCREMENT列，该模式会很有用。(不推荐采用该惯例)。例如，如果你用**mysqldump**转储表并重载，MySQL遇到0值一般会生成新的序列号，生成的表的内容与转储的表不同。重载转储文件前启用NO_AUTO_VALUE_ON_ZERO可以解决该问题。**mysqldump**在输出中自动包括启用NO_AUTO_VALUE_ON_ZERO的语句。

· NO_BACKSLASH_ESCAPES

禁用反斜线字符('\')做为字符串内的退出字符。启用该模式，反斜线则成为普通字符。

· NO_DIR_IN_CREATE

创建表时，忽视所有INDEX DIRECTORY和DATA DIRECTORY指令。该选项对从复制服务器有用。

· NO_ENGINE_SUBSTITUTION

如果需要的存储引擎被禁用或未编译，可以防止自动替换存储引擎。

· NO_FIELD_OPTIONS

不要在SHOW CREATE TABLE的输出中打印MySQL专用列选项。该模式在可移植模式（portability mode）下用于**mysqldump**。

· NO_KEY_OPTIONS

不要在SHOW CREATE TABLE的输出中打印MySQL专用索引选项。该模式在可移植模式（portability mode）下用于**mysqldump**。

· NO_TABLE_OPTIONS

不要在SHOW CREATE TABLE的输出中打印MySQL专用表选项（例如ENGINE）。该模式在可移植模式（portability mode）下用于**mysqldump**。

· NO_UNSIGNED_SUBTRACTION

在减运算中，如果某个操作数没有符号，不要将结果标记为UNSIGNED。请注意这样使UNSIGNED BIGINT不能100%用于上下文中。参见12.8节，“Cast函数和操作符”。

· NO_ZERO_DATE

在严格模式，不要将 '0000-00-00'做为合法日期。你仍然可以用IGNORE选项插入零日期。在非严格模式，可以接受该日期，但会生成警告。

· NO_ZERO_IN_DATE

在严格模式，不接受月或日部分为0的日期。如果使用IGNORE选项，我们为类似的日期插入'0000-00-00'。在非严格模式，可以接受该日期，但会生成警告。

· ONLY_FULL_GROUP_BY

不要让GROUP BY部分中的查询指向未选择的列。

· PIPES_AS_CONCAT

将||视为字符串连接操作符（+）(同CONCAT())，而不视为OR。

· REAL_AS_FLOAT

将REAL视为FLOAT的同义词，而不是DOUBLE的同义词。

· STRICT_TRANS_TABLES

为所有存储引擎启用严格模式。非法数据值被拒绝。后面有详细说明。

· STRICT_TRANS_TABLES

为事务存储引擎启用严格模式，也可能为非事务存储引擎启用严格模式。后面有详细说明。

严格模式控制MySQL如何处理非法或丢失的输入值。有几种原因可以使一个值为非法。例如，数据类型错误，不适合列，或超出范围。当新插入的行不包含某列的没有显示定义DEFAULT子句的值，则该值被丢失。

对于事务表，当启用STRICT_ALL_TABLES或STRICT_TRANS_TABLES模式时，如果语句中有非法或丢失值，则会出现错误。语句被放弃并滚动。

对于非事务表，如果插入或更新的第1行出现坏值，两种模式的行为相同。语句被放弃，表保持不变。如果语句插入或修改多行，并且坏值出现在第2或后面的行，结果取决于启用了哪个严格选项：

· 对于STRICT_ALL_TABLES，MySQL返回错误并忽视剩余的行。但是，在这种情况下，前面的行已经被插入或更新。这说明你可以部分更新，这可能不是你想要的。要避免这点，最好使用单行语句，因为这样可以不更改表即可以放弃。

· 对于STRICT_TRANS_TABLES，MySQL将非法值转换为最接近该列的合法值并插入调整后的值。如果值丢失，MySQL在列中插入隐式 默认值。在任何情况下，MySQL都会生成警告而不是给出错误并继续执行语句。13.1.5节，“CREATE TABLE语法”描述了隐式默认值。

严格模式不允许非法日期，例如'2004-04-31'。它不允许禁止日期使用“零”部分，例如'2004-04-00'或“零”日期。要想禁止，应在严格模式基础上，启用NO_ZERO_IN_DATE和NO_ZERO_DATE SQL模式。

如果你不使用严格模式(即不启用STRICT_TRANS_TABLES或STRICT_ALL_TABLES模式)，对于非法或丢失的值，MySQL将插入调整后的值并给出警告。在严格模式，你可以通过INSERT IGNORE或UPDATE IGNORE来实现。参见13.5.4.22节，“SHOW WARNINGS语法”。

下面的特殊模式快速组合了前面所列的模式。

其中包括大多数最新版本MySQL中的所有模式值。旧版本中，组合模式不包括新版本中没有的不适用的具体模式值。

· ANSI

等同REAL_AS_FLOAT、PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE。参见1.8.3节，“在ANSI模式下运行MySQL”。

· DB2

等同PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、NO_KEY_OPTIONS、NO_TABLE_OPTIONS、NO_FIELD_OPTIONS。

· MAXDB

等同PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、NO_KEY_OPTIONS、NO_TABLE_OPTIONS、NO_FIELD_OPTIONS、NO_AUTO_CREATE_USER。

· MSSQL

等同PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、NO_KEY_OPTIONS、NO_TABLE_OPTIONS、 NO_FIELD_OPTIONS。

· MYSQL323

等同NO_FIELD_OPTIONS、HIGH_NOT_PRECEDENCE。

· MYSQL40

等同NO_FIELD_OPTIONS、HIGH_NOT_PRECEDENCE。

· ORACLE

等同PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、NO_KEY_OPTIONS、NO_TABLE_OPTIONS、NO_FIELD_OPTIONS、NO_AUTO_CREATE_USER。

· POSTGRESQL

等同PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、NO_KEY_OPTIONS、NO_TABLE_OPTIONS、NO_FIELD_OPTIONS。

· TRADITIONAL

等同STRICT_TRANS_TABLES、STRICT_ALL_TABLES、NO_ZERO_IN_DATE、NO_ZERO_DATE、ERROR_FOR_DIVISION_BY_ZERO、NO_AUTO_CREATE_USER。

5.3.3. 服务器系统变量

5.3.3.1. 动态系统变量

服务器将维护许多表示其配置的系统变量。所有变量均有默认值。可以在命令行中或选项文件设置选项在服务器启动时对它们进行设置。大多数可以在运行时使用SET语句来设置。

mysqld服务器维护两种变量。全局变量影响服务器的全局操作。会话变量影响具体客户端连接相关操作。

服务器启动时，将所有全局变量初始化为默认值。可以在选项文件或命令行中指定的选项来更改这些默认值。服务器启动后，通过连接服务器并执行SET GLOBAL *var_name*语句可以更改动态全局变量。要想更改全局变量，必须具有SUPER权限。

服务器还为每个客户端连接维护会话变量。连接时使用相应全局变量的当前值对客户端会话变量进行初始化。客户可以通过SET SESSION *var_name*语句来更改动态会话变量。设置会话变量不需要特殊权限，但客户可以只更改自己的会话变量，而不更改其它客户的会话变量。

任何访问全局变量的客户端都可以看见对全局变量的更改。但是，它只影响在更改后连接的从该全局变量初始化相应会话变量的客户端。它不会影响已经连接上的客户端的会话变量(甚至是执行SET GLOBAL语句的客户端)。

当使用启动选项设置变量时，变量值可以使用后缀K、M或G分别表示千字节、兆字节或gigabytes。例如，下面的命令启动服务器时的键值缓冲区大小为16 megabytes：

```
mysqld --key_buffer_size=16M
```

后缀的大小写美关系；16M和16m是同样的。

运行时，使用SET语句来设置系统变量。此时，不能使用后缀，但值可以采取下列表达式：

```
mysql> SET sort_buffer_size = 10 * 1024 * 1024;
```

要想显式指定是否设置全局或会话变量，使用GLOBAL或SESSION选项：

```
mysql> SET GLOBAL sort_buffer_size = 10 * 1024 * 1024;
mysql> SET SESSION sort_buffer_size = 10 * 1024 * 1024;
```

两个选项均没有，则语句设置会话变量。

5.3.3.1节，“动态系统变量”中列出了可以在运行时设置的变量。

如果你想用SET语句限制系统变量可设的最大值，可以在服务器启动时通过--maximum-*var_name*形式的选项来指定。例如，要想防止query_cache_size的值运行时超过32MB，使用选项--maximum-query_cache_size=32M。

你可以通过SHOW VARIABLES语句查看系统变量及其值。详细信息参见9.4节，“系统变量”。

```
mysql> SHOW VARIABLES;
+---------------------------------+-------------------------------------------+
| Variable_name                   | Value                                     |
+---------------------------------+-------------------------------------------+
| auto_increment_increment        | 1                                         |
| auto_increment_offset           | 1                                         |
| automatic_sp_privileges         | ON                                        |
```

```
| back_log                        | 50                                          |
| basedir                         | /home/jon/bin/mysql/                        |
| binlog_cache_size               | 32768                                       |
| bulk_insert_buffer_size         | 8388608                                     |
| character_set_client            | latin1                                      |
| character_set_connection        | latin1                                      |
| character_set_database          | latin1                                      |
| character_set_results           | latin1                                      |
| character_set_server            | latin1                                      |
| character_set_system            | utf8                                        |
| character_sets_dir              | /home/jon/bin/mysql/share/mysql/charsets/ |
| collation_connection            | latin1_swedish_ci                           |
| collation_database              | latin1_swedish_ci                           |
| collation_server                | latin1_swedish_ci                           |
| completion_type                 | 0                                           |
| concurrent_insert               | 1                                           |
| connect_timeout                 | 5                                           |
| datadir                         | /home/jon/bin/mysql/var/                    |
| date_format                     | %Y-%m-%d                                    |
| datetime_format                 | %Y-%m-%d %H:%i:%s                           |
| default_week_format             | 0                                           |
| delay_key_write                 | ON                                          |
| delayed_insert_limit            | 100                                         |
| delayed_insert_timeout          | 300                                         |
| delayed_queue_size              | 1000                                        |
| div_precision_increment         | 4                                           |
| engine_condition_pushdown       | OFF                                         |
| expire_logs_days                | 0                                           |
| flush                           | OFF                                         |
| flush_time                      | 0                                           |
| ft_boolean_syntax               | + -><()~*:""&|                              |
| ft_max_word_len                 | 84                                          |
| ft_min_word_len                 | 4                                           |
| ft_query_expansion_limit        | 20                                          |
| ft_stopword_file                | (built-in)                                  |
| group_concat_max_len            | 1024                                        |
| have_archive                    | YES                                         |
| have_bdb                        | NO                                          |
| have_blackhole_engine           | YES                                         |
| have_compress                   | YES                                         |
| have_crypt                      | YES                                         |
| have_csv                        | YES                                         |
| have_example_engine             | NO                                          |
| have_federated_engine           | NO                                          |
| have_geometry                   | YES                                         |
| have_innodb                     | YES                                         |
| have_isam                       | NO                                          |
| have_ndbcluster                 | DISABLED                                    |
| have_openssl                    | NO                                          |
| have_partition_engine           | YES                                         |
| have_query_cache                | YES                                         |
| have_raid                       | NO                                          |
| have_rtree_keys                 | YES                                         |
| have_symlink                    | YES                                         |
| init_connect                    |                                             |
| init_file                       |                                             |
| init_slave                      |                                             |
| innodb_additional_mem_pool_size | 1048576                                     |
| innodb_autoextend_increment     | 8                                           |
| innodb_buffer_pool_awe_mem_mb   | 0                                           |
| innodb_buffer_pool_size         | 8388608                                     |
| innodb_checksums                | ON                                          |
| innodb_commit_concurrency       | 0                                           |
| innodb_concurrency_tickets      | 500                                         |
| innodb_data_file_path           | ibdata1:10M:autoextend                      |
| innodb_data_home_dir            |                                             |
| innodb_doublewrite              | ON                                          |
| innodb_fast_shutdown            | 1                                           |
| innodb_file_io_threads          | 4                                           |
| innodb_file_per_table           | OFF                                         |
| innodb_flush_log_at_trx_commit  | 1                                           |
| innodb_flush_method             |                                             |
| innodb_force_recovery           | 0                                           |
| innodb_lock_wait_timeout        | 50                                          |
| innodb_locks_unsafe_for_binlog  | OFF                                         |
| 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_max_dirty_pages_pct      | 90                                          |
| innodb_max_purge_lag            | 0                                           |
| innodb_mirrored_log_groups      | 1                                           |
| innodb_open_files               | 300                                         |
| innodb_support_xa               | ON                                          |
| innodb_sync_spin_loops          | 20                                          |
| innodb_table_locks              | ON                                          |
| innodb_thread_concurrency       | 20                                          |
| innodb_thread_sleep_delay       | 10000                                       |
| interactive_timeout             | 28800                                       |
| join_buffer_size                | 131072                                      |
| key_buffer_size                 | 8388600                                     |
| key_cache_age_threshold         | 300                                         |
| key_cache_block_size            | 1024                                        |
| key_cache_division_limit        | 100                                         |
| language                        | /home/jon/bin/mysql/share/mysql/english/    |
| large_files_support             | ON                                          |
| large_page_size                 | 0                                           |
| large_pages                     | OFF                                         |
| license                         | GPL                                         |
| local_infile                    | ON                                          |
| locked_in_memory                | OFF                                         |
| log                             | ON                                          |
| log_bin                         | ON                                          |
| log_bin_trust_routine_creators  | OFF                                         |
| log_error                       | /home/jon/bin/mysql/var/master1.err         |
| log_slave_updates               | OFF                                         |
| log_slow_queries                | OFF                                         |
| log_warnings                    | 1                                           |
| long_query_time                 | 10                                          |
| low_priority_updates            | OFF                                         |
| lower_case_file_system          | OFF                                         |
| lower_case_table_names          | 0                                           |
```

```
| max_allowed_packet                | 1048576                                   |
| max_binlog_cache_size             | 4294967295                                |
| max_binlog_size                   | 1073741824                                |
| max_connect_errors                | 10                                        |
| max_connections                   | 100                                       |
| max_delayed_threads               | 20                                        |
| max_error_count                   | 64                                        |
| max_heap_table_size               | 16777216                                  |
| max_insert_delayed_threads        | 20                                        |
| max_join_size                     | 4294967295                                |
| max_length_for_sort_data          | 1024                                      |
| max_relay_log_size                | 0                                         |
| max_seeks_for_key                 | 4294967295                                |
| max_sort_length                   | 1024                                      |
| max_tmp_tables                    | 32                                        |
| max_user_connections              | 0                                         |
| max_write_lock_count              | 4294967295                                |
| multi_range_count                 | 256                                       |
| myisam_data_pointer_size          | 6                                         |
| myisam_max_sort_file_size         | 2147483647                                |
| myisam_recover_options            | OFF                                       |
| myisam_repair_threads             | 1                                         |
| myisam_sort_buffer_size           | 8388608                                   |
| ndb_autoincrement_prefetch_sz     | 32                                        |
| ndb_cache_check_time              | 0                                         |
| ndb_force_send                    | ON                                        |
| ndb_index_stat_cache_entries      | 32                                        |
| ndb_index_stat_enable             | ON                                        |
| ndb_index_stat_update_freq        | 20                                        |
| ndb_use_exact_count               | ON                                        |
| ndb_use_transactions              | ON                                        |
| net_buffer_length                 | 16384                                     |
| net_read_timeout                  | 30                                        |
| net_retry_count                   | 10                                        |
| net_write_timeout                 | 60                                        |
| new                               | OFF                                       |
| old_alter_table                   | OFF                                       |
| old_passwords                     | OFF                                       |
| open_files_limit                  | 1024                                      |
| optimizer_prune_level             | 1                                         |
| optimizer_search_depth            | 62                                        |
| pid_file                          | /home/jon/bin/mysql/var/hostname.pid1     |
| port                              | 3306                                      |
| preload_buffer_size               | 32768                                     |
| protocol_version                  | 10                                        |
| query_alloc_block_size            | 8192                                      |
| query_cache_limit                 | 1048576                                   |
| query_cache_min_res_unit          | 4096                                      |
| query_cache_size                  | 0                                         |
| query_cache_type                  | ON                                        |
| query_cache_wlock_invalidate      | OFF                                       |
| query_prealloc_size               | 8192                                      |
| range_alloc_block_size            | 2048                                      |
| read_buffer_size                  | 131072                                    |
| read_only                         | OFF                                       |
| read_rnd_buffer_size              | 262144                                    |
| relay_log_purge                   | ON                                        |
| relay_log_space_limit             | 0                                         |
| rpl_recovery_rank                 | 0                                         |
| secure_auth                       | OFF                                       |
| server_id                         | 1                                         |
| skip_external_locking             | ON                                        |
| skip_networking                   | OFF                                       |
| skip_show_database                | OFF                                       |
| slave_compressed_protocol         | OFF                                       |
| slave_load_tmpdir                 | /tmp/                                     |
| slave_net_timeout                 | 3600                                      |
| slave_skip_errors                 | OFF                                       |
| slave_transaction_retries         | 10                                        |
| slow_launch_time                  | 2                                         |
| socket                            | /tmp/mysql.sock                           |
| sort_buffer_size                  | 2097144                                   |
| sql_mode                          |                                           |
| sql_notes                         | ON                                        |
| sql_warnings                      | ON                                        |
| storage_engine                    | MyISAM                                    |
| sync_binlog                       | 0                                         |
| sync_frm                          | ON                                        |
| sync_replication                  | 0                                         |
| sync_replication_slave_id         | 0                                         |
| sync_replication_timeout          | 10                                        |
| system_time_zone                  | EST                                       |
| table_cache                       | 64                                        |
| table_lock_wait_timeout           | 50                                        |
| table_type                        | MyISAM                                    |
| thread_cache_size                 | 0                                         |
| thread_stack                      | 196608                                    |
| time_format                       | %H:%i:%s                                  |
| time_zone                         | SYSTEM                                    |
| timed_mutexes                     | OFF                                       |
| tmp_table_size                    | 33554432                                  |
| tmpdir                            |                                           |
| transaction_alloc_block_size      | 8192                                      |
| transaction_prealloc_size         | 4096                                      |
| tx_isolation                      | REPEATABLE-READ                           |
| updatable_views_with_limit        | YES                                       |
| version                           | 5.1.2-alpha-log                           |
| version_comment                   | Source distribution                       |
| version_compile_machine           | i686                                      |
| version_compile_os                | suse-linux                                |
| wait_timeout                      | 28800                                     |
+-----------------------------------+-------------------------------------------+
218 rows in set (0.03 sec)
```

此处描述了大多数系统变量。没有版本的变量在所有MySQL 5.1 发布中适用。关于其使用历史信息，请参见*MySQL 5.0*参考指南和*MySQL 4.1*参考指南。InnoDB系统变量列于 15.2.4节，“InnoDB启动选项”。

若没有另行规定，缓冲区大小、长度和堆栈大小的单位均为字节。

关于这些变量的调节信息参见7.5.2节，“调节服务器参数”。

- auto_increment_increment

auto_increment_increment和auto_increment_offset用于主服务器－主服务器（master-to-master）复制，并可以用来控制AUTO_INCREMENT列的操作。两个变量均可以设置为全局或局部变量，并且假定每个值都可以为1到65,535之间的整数值。将其中一个变量设置为0会使该变量为1。如果试图将这些变量设置为大于65,535或小于0的值，则会将该值设置为65,535。如果向将auto_increment_increment或auto_increment_offset设置为非整数值，则会给出错误，并且变量的实际值在这种情况下保持不变。

这两个变量影响AUTO_INCREMENT列的方式：

o auto_increment_increment控制列中的值的增量值。例如：

```
o                       mysql> SHOW VARIABLES LIKE 'auto_inc%';
o                       +--------------------------+-------+
o                       | Variable_name            | Value |
o                       +--------------------------+-------+
o                       | auto_increment_increment | 1     |
o                       | auto_increment_offset    | 1     |
o                       +--------------------------+-------+
o                       2 rows in set (0.00 sec)
o
o                       mysql> CREATE TABLE autoinc1 (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
o                       Query OK, 0 rows affected (0.04 sec)
o
o                       mysql> SET @auto_increment_increment=10;
o                       Query OK, 0 rows affected (0.00 sec)
o
o                       mysql> SHOW VARIABLES LIKE 'auto_inc%';
o                       +--------------------------+-------+
o                       | Variable_name            | Value |
o                       +--------------------------+-------+
o                       | auto_increment_increment | 10    |
o                       | auto_increment_offset    | 1     |
o                       +--------------------------+-------+
o                       2 rows in set (0.01 sec)
o
o                       mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
o                       Query OK, 4 rows affected (0.00 sec)
o                       Records: 4  Duplicates: 0  Warnings: 0
o
o                       mysql> SELECT col FROM autoinc1;
o                       +-----+
o                       | col |
o                       +-----+
o                       |   1 |
o                       |  11 |
o                       |  21 |
o                       |  31 |
o                       +-----+
o                       4 rows in set (0.00 sec)
```

(注明如何使用SHOW VARIABLES来获取这些变量的当前值）。

o auto_increment_offset确定AUTO_INCREMENT列值的起点。假定在与前面的例子的相同的会话中执行下面的命令：

```
o                       mysql> SET @auto_increment_offset=5;
o                       Query OK, 0 rows affected (0.00 sec)
o
o                       mysql> SHOW VARIABLES LIKE 'auto_inc%';
o                       +--------------------------+-------+
o                       | Variable_name            | Value |
```

```
o                       +--------------------------+-------+
o                       | auto_increment_increment | 10    |
o                       | auto_increment_offset    | 5     |
o                       +--------------------------+-------+
o                       2 rows in set (0.00 sec)
o
o                       mysql> CREATE TABLE autoinc2 (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
o                       Query OK, 0 rows affected (0.06 sec)
o
o                       mysql> INSERT INTO autoinc2 VALUES (NULL), (NULL), (NULL), (NULL);
o                       Query OK, 4 rows affected (0.00 sec)
o                       Records: 4  Duplicates: 0  Warnings: 0
o
o                       mysql> SELECT col FROM autoinc2;
o                       +-----+
o                       | col |
o                       +-----+
o                       |   5 |
o                       |  15 |
o                       |  25 |
o                       |  35 |
o                       +-----+
o                       4 rows in set (0.02 sec)
o
```

如果auto_increment_offset的值大于auto_increment_increment的值，则auto_increment_offset的值被忽略。

如果其中一个或两个变量被更改了，然后更改插入到包含AUTO_INCREMENT列的表中的新行，结果可能看上去有问题，由于计算AUTO_INCREMENT系列值时没有考虑列内已经存在的值，并且插入的下一个值是列内最小的值，大于AUTO_INCREMENT列内已有的最大值。换句话说，数值的计算方法为：

auto_increment_offset+ *N* * auto_increment_increment

其中*N*为系列内的正整数值[1,2,3,...]。例如：

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 5     |
+--------------------------+-------+
2 rows in set (0.00 sec)

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
|   1 |
|  11 |
|  21 |
|  31 |
+-----+
4 rows in set (0.00 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT col FROM autoinc1;
```

```
+-----+
| col |
+-----+
|   1 |
|  11 |
|  21 |
|  31 |
|  35 |
|  45 |
|  55 |
|  65 |
+-----+
8 rows in set (0.00 sec)
```

auto_increment_increment和auto_increment_offset所示的值可以生成系列5 + *N* * 10,即，[5,15,25,35,45,...]。在INSERT前col 列内最大的值为31,AUTO_INCREMENT数列的下一个值为35,因此col中插入的值从该点开始，结果如SELECT查询所示。

一定要记住不可能将这两个变量的结果限制到一个表中，因此不会替代其它数据库管理系统提供的序列；这些变量控制MySQL服务器上**all**表AUTO_INCREMENT列的所有行为。如果某个变量设为全局变量，则只有通过局部设置将全局值更改和覆盖后或**mysqld**重启后其作用方可改变；如果局部设置，则新值影响所有表的AUTO_INCREMENT列，在这个会话期间当前用户在这些表中插入了新行，除非在会话期间更改了这些值。

auto_increment_increment的 默认值为1。参见6.12节，“多服务器复制中的Auto-Increment”。

- auto_increment_offset

该变量的默认值为1。详见auto_increment_increment的描述。

- back_log

MySQL有的主要连接请求的数量。当主MySQL线程在短时间内得到许多连接请求时发挥作用。主线程需要花一些时间(尽管很少)来检查连接并启动一个新线程。back_log值说明MySQL临时停止响应新请求前在短时间内可以堆起多少请求。如果你需要在短时间内允许大量连接，可以增加该数值。

换句话说，该值为“进”TCP/IP连接帧听队列的大小。操作系统有该队列自己的限制值。本手册中Unix listen()系统调用页应有更详细的信息。该变量最大值请查阅OS文档。企图将back_log设置为高于你的操作系统限值是徒劳无益的。

- basedir

MySQL安装基准目录。可以用--basedir选项设置该变量。

- bdb_cache_size

为BDB表缓存索引和行分配的缓冲区的大小。如果你不使用BDB表，你应用--skip-bdb启动**mysqld**以便不浪费该缓存。

- bdb_home

BDB表基准目录。应与datadir变量的值相同。

- bdb_log_buffer_size

为BDB表缓存索引和行分配的缓冲区的大小。如果你不使用BDB表，你应将该值设置为0或用--skip-bdb启动**mysqld**以便不浪费该缓存。

- bdb_logdir

BDB存储引擎写它日志文件的目录。可以用--bdb-logdir选项设置该变量。

- bdb_max_lock

在BDB表下可以激活的最大锁数(默认为10,000)。如果当你执行长事务或当**mysqld**必须检查许多行来计算查询时出现下面的错误，你应增加该值：

```
bdb: Lock table is out of available locks
Got error 12 from ...
```

- bdb_shared_data

如果你正使用--bdb-shared-data应为ON。

- bdb_tmpdir

--bdb-tmpdir选项的值。

- binlog_cache_size

在事务过程中容纳二进制日志SQL语句的缓存大小。二进制日志缓存是服务器支持事务存储引擎并且服务器启用了二进制日志(--log-bin选项)的前提下为每个

客户端分配的内存。如果你经常使用大的，多语句事务，你可以增加该值以获得更有的性能。Binlog_cache_use和Binlog_cache_disk_use状态变量可以用来调整该变量的大小。参见5.11.3节，“二进制日志”。

· bulk_insert_buffer_size

MyISAM 使用专用树状缓存来使INSERT ... SELECT、INSERT ... VALUES (...)、(...)、 ...和LOAD DATA INFILE的大块插入更快。该变量用每线程的字节数限制缓存树的大小。将它设置为0禁用优化。注释：只有向非空表添加数据时才使用该缓存。 默认值是8MB。

· character_set_client

来自客户端的语句的字符集。

· character_set_connection

用于没有字符集导入符的文字和数字－字符串转换。

· character_set_database

默认数据库使用的字符集。当默认数据库更改时，服务器则设置该变量。如果没有默认数据库，变量的值同character_set_server。

· character_set_results

用于向客户端返回查询结果的字符集。

· character_set_ server

服务器的默认字符集。

· character_set_system

服务器用来保存识别符的字符集。该值一定是utf8。

· character_sets_dir

字符集安装目录。

· collation_connection

连接字符集的校对规则。

· collation_database

默认数据库使用的校对规则。当默认数据库改变时服务器则设置该变量。如果没有默认数据库，变量的值同collation_server。

· collation_server

服务器的默认校对规则。

· completion_type

事务结束类型：

o 如果该值为0(默认)，COMMIT和ROLLBACK不受影响。

o 如果该值为1，COMMIT和ROLLBACK分别等同于COMMIT AND CHAIN和ROLLBACK AND CHAIN。(新事务用刚刚结束的事务相同的间隔等级立即启动）。

o 如果该值为2，COMMIT和ROLLBACK分别等同于COMM它RELEASE和ROLLBACK RELEASE。(事务终止后，服务器断开）。

· concurrent_insert

如果为ON(默认值)，MySQL允许INSERT和SELECT语句在中间没有空数据块的MyISAM表中并行运行。你可以用--safe或--skip-new启动**mysqld**关闭该选项。

该变量为整数，有3个值：

值	描述
0	关
1	(默认)在没有空数据块的MyISAM表中启用并行插入
2	为所有MyISAM表启用并行插入。如果表有空记录或正被另一线程使用，新行将插入到表的最后。如果表未使用，MySQL将进行普通读锁定并将新行插入空记录。

· connect_timeout

mysqld服务器用Bad handshake响应前等待连接包的秒数。

· datadir

MySQL数据目录。可以用--datadir选项设置该变量。

- date_format

该变量未使用。

- datetime_format

该变量未使用。

- default_week_format

WEEK() 函数使用的默认模式。

- delay_key_write

该选项只适用MyISAM表。它具有下述值可以影响CREATE TABLE语句使用的DELAY_KEY_WRITE表选项的处理。

选项	描述
OFF	DELAY_KEY_WRITE被忽略。
ON	MySQL在CREATE TABLE中用DELAY_KEY_WRITE选项。这是 默认值。
ALL	用启用DELAY_KEY_WRITE选项创建表的相同方法对所有新打开表的进行处理。

如果启用了DELAY_KEY_WRITE，说明使用该项的表的键缓冲区在每次更新索引时不被清空，只有关闭表时才清空。遮掩盖可以大大加快键的写操作，但如果你使用该特性，你应用--myisam-recover选项启动服务器，为所有MyISAM表添加自动检查(例如，--myisam-recover=BACKUP,FORCE)。参见5.3.1节，“**mysqld**命令行选项”和15.1.1节，“MyISAM启动选项”。

请注意--external-locking不为使用延迟键写入的表提供索引破坏保护。

- delayed_insert_limit

插入delayed_insert_limit 延迟行后，INSERT DELAYED 处理器线程检查是否有挂起的SELECT语句。如果有，在继续插入延迟的行之前，允许它们先执行。

- delayed_insert_timeout

INSERT DELAYED处理器线程终止前应等待INSERT语句的时间。

- delayed_queue_size

这是各个表中处理INSERT DELAYED语句时队列中行的数量限制。如果队列满了，执行INSERT DELAYED语句的客户端应等待直到队列内再有空间。

- div_precision_increment

该变量说明用/操作符执行除操作的结果可增加的精确度的位数。 默认值是4。最小和最大值分别为0和30。下面的示例说明了增加 默认值的结果。

```
mysql> SELECT 1/7;
+--------+
| 1/7    |
+--------+
| 0.1429 |
+--------+
mysql> SET div_precision_increment = 12;
mysql> SELECT 1/7;
+----------------+
| 1/7            |
+----------------+
| 0.142857142857 |
+----------------+
```

- engine_condition_pushdown

该变量适用于NDB。默认值为0(OFF)：如果你执行类似查询SELECT * FROM t WHERE mycol = 42，其中mycol为没有索引的列，当满了的表扫描每个NDB节点时，执行该查询。每个节点使用WHERE条件将每一行发送给MySQL服务器。如果engine_condition_pushdown被设置为1(ON)，该条件“pushed down”给存储引擎并发送给NDB节点。每个节点都执行扫描，并只向MySQL服务器发送回匹配条件的行。

- expire_logs_days

二进制日志自动删除的天数。默认值为0,表示“没有自动删除”。启动时和二进制日志循环时可能删除。

- flush

如果用--flush选项启动**mysqld**该值为ON。

- flush_time

如果设为非零值，每隔flush_time秒则关闭所有表以释放硬盘资源并同步未清空的数据。我们建议只在Windows 9x或Me，或有最小资源的系统中使用该选

项。

· ft_boolean_syntax

使用IN BOOLEAN MODE执行的布尔全文搜索支持的操作符系列。参见12.7.1节，“布尔全文搜索”。

默认变量值为 '+ -><()~*:""&|'。更改这些值的规则是：

o 操作符函数由其在字符串内的位置决定。

o 替换值必须是14个字符。

o 每个字符必须为ASCII码非文字数字字符。

o 第1个或第2个字符必须为空格。

o 除非语句在第11个字符和第12个字符处引用了操作符，否则不允许复制。这两个字符可以不相同，但这是唯一可能的两个。

o 位置10、13和14(默认设置为':'、'&'和'|')保留用于将来扩展。

· ft_max_word_len

FULLTEXT索引中所包含的字的最大长度。

注释：更改该变量后必须重建FULLTEXT索引。应使用REPAIR TABLE *tbl_name* QUICK。

· ft_min_word_len

FULLTEXT索引中所包含的字的最小长度。

注释：更改该变量后必须重建FULLTEXT索引。应使用REPAIR TABLE *tbl_name* QUICK。

· ft_query_expansion_limit

使用WITH QUERY EXPANSION进行全文搜索的最大匹配数。

· ft_stopword_file

用于读取全文搜索的停止字清单的文件。该文件中的所有字都会用到；注释不重要。默认情况下，使用内嵌式停止字清单(如myisam/ft_static.c文件中所定义)。将该变量设置为空字符串("")则禁用停止字过滤。

注释：更改该变量或停止字文件的内容后必须重建FULLTEXT索引。应使用REPAIR TABLE *tbl_name* QUICK。

· group_concat_max_len

允许的GROUP_CONCAT()函数结果的最大长度。

· have_archive

如果**mysqld**支持ARCHIVE表则为YES，否则为NO。

· have_bdb

如果**mysqld**支持BDB表则为YES。如果使用--skip-bdb则为DISABLED。

· have_blackhole_engine

如果**mysqld**支持BLACKHOLE表则为YES，否则为NO。

· have_compress

是否zlib压缩库适合该服务器。如果不适合，不能使用COMPRESS()和UNCOMPRESS()函数。

· have_crypt

是否crypt()系统调用适合该服务器。如果不适合，不能使用CRYPT()函数。

· have_csv

如果**mysqld**支持ARCHIVE表则为YES，否则为NO。

· have_example_engine

如果**mysqld**支持EXAMPLE表则为YES，否则为NO。

have_federated_engine

如果**mysqld**支持FEDERATED表则为YES，否则为NO。

· have_geometry

是否服务器支持空间数据类型。

- have_innodb

如果**mysqld**支持InnoDB表则为YES。如果使用--skip-innodb则为DISABLED。

- have_isam

在MySQL 5.1中,只是为了向后兼容显示该值，并且总是NO，因为不再支持ISAM表。

- have_ndbcluster

如果**mysqld**支持NDB CLUSTER表则为YES。如果使用了--skip-ndbcluster则为DISABLED。

- have_partition_engine

如果**mysqld**支持分区则为YES。在MySQL 5.1.1中加入。

- have_openssl

如果**mysqld**支持客户端/服务器协议的SSL(加密)则为YES。

- have_query_cache

如果**mysqld**支持查询缓存则为YES。

- have_raid

如果**mysqld**支持RAID选项则为YES。

- have_rtree_keys

RTREE索引是否可用。(用于MyISAM表的空间索引）。

- have_symlink

是否启用符号链接支持。在Unix中需要用于支持DATA DIRECTORY和INDEX DIRECTORY表选项。

- init_connect

服务器为每个连接的客户端执行的字符串。字符串由一个或多个SQL语句组成。要想指定多个语句，用分号间隔开。例如，每个客户端开始时默认启用autocommit模式。没有全局服务器变量可以规定autocommit默认情况下应禁用，但可以用init_connect来获得相同的效果：

```
SET GLOBAL init_connect='SET AUTOCOMMIT=0';
```

还可以在命令行或选项文件中设置该变量。要想使用选项文件设置变量，应包括下述行：

```
[mysqld]
init_connect='SET AUTOCOMMIT=0'
```

请注意init_connect的内容并不为拥有SUPER权限的用户执行；实际是内容设置错误(包含错误查询，例如语法错误)，这样使所有连接失败。不为SUPER用户执行，使SUPER用户可以打开连接并固定init_connect。

- init_file

启动服务器时用--init-file选项指定的文件名。文件中包含服务器启动时要执行的SQL语句。每个语句必须在同一行中并且不能包括注释。

- init_slave

该变量类似init_connect，但是每次SQL线程启动时从服务器应执行该字符串。该字符串的格式与init_connect变量相同。

- innodb_*xxx*

InnoDB系统变量列入15.2.4节，“InnoDB启动选项”。

- interactive_timeout

服务器关闭交互式连接前等待活动的秒数。交互式客户端定义为在mysql_real_connect()中使用CLIENT_INTERACTIVE选项的客户端。又见wait_timeout。

- join_buffer_size

用于完全联接的缓冲区的大小(当不使用索引的时候使用联接操作)。一般情况获得快速联接的最好方法是添加索引。当增加索引时不可能通过增加join_buffer_size值来获得快速完全联接。将为两个表之间的每个完全联接分配联接缓冲区。对于多个表之间不使用索引的复杂联接，需要多联接缓冲区。

- key_buffer_size

MyISAM表的索引块分配了缓冲区，由所有线程共享。key_buffer_size是索引块缓冲区的大小。键值缓冲区即为键值缓存。

key_buffer_size的最大允许设定值为4GB。有效最大值可以更小，取决于可用物理RAM和操作系统或硬件平台强加的每个进程的RAM限制。

增加该值，达到你可以提供的更好的索引处理(所有读和多个写操作)。通常为主要运行MySQL的机器内存的25%。但是，如果你将该值设得过大(例如，大于

总内存的50%)，系统将转换为页并变得极慢。MySQL依赖操作系统来执行数据读取时的文件系统缓存，因此你必须为文件系统缓存留一些空间。

同时写多行时要想速度更快，应使用LOCK TABLES。参见13.4.5节，“LOCK TABLES和UNLOCK TABLES语法”。

你可以通过执行SHOW STATUS语句并检查Key_read_requests、Key_reads、Key_write_requests和Key_writes状态变量来检查键值缓冲区的性能。参见13.5.4节，“SHOW语法”。

Key_reads/Key_read_requests比例一般应小于0.01。如果你使用更新和删除，Key_writes/Key_write_requests比例通常接近1，但如果你更新时会同时影响到多行或如果你正使用DELAY_KEY_WRITE表选项，可能小得多。

用key_buffer_size结合Key_blocks_unused状态变量和缓冲区块大小，可以确定使用的键值缓冲区的比例。从key_cache_block_size服务器变量可以获得缓冲区块大小。使用的缓冲区的比例为:

```
1 - ((Key_blocks_unused * key_cache_block_size) / key_buffer_size)
```

该值为约数，因为键值缓冲区的部分空间被分配用作内部管理结构。

可以创建多个MyISAM键值缓存。4GB限制可以适合每个缓存，而不是一个组。参见7.4.6节，“MyISAM键高速缓冲”。

- key_cache_age_threshold

该值控制将缓冲区从键值缓存热子链(sub-chain)降级到温子链(sub-chain)。如果值更低，则降级更快。最小值为100。 默认值是300。参见7.4.6节，“MyISAM键高速缓冲”。

- key_cache_block_size

键值缓存内块的字节大小。默认值是1024。参见7.4.6节，“MyISAM键高速缓冲”。

- key_cache_division_limit

键值缓存缓冲区链热子链和温子链的划分点。该值为缓冲区链用于温子链的百分比。允许的值的范围为1到100。 默认值是100。参见7.4.6节，“MyISAM键高速缓冲”。

- language

错误消息所用语言。

- large_file_support

mysqld编译时是否使用了大文件支持选项。

- large_pages

说明是否启用了大页面支持。

- license

服务器的许可类型。

- local_infile

是否LOCAL支持LOAD DATA INFILE语句。

- locked_in_memory

是否用–memlock将**mysqld**锁在内存中。

- log

是否启用将所有查询记录到常规查询日志中。参见5.11.2节，“通用查询日志”。

- log_bin

是否启用二进制日志。参见5.11.3节，“二进制日志”。

- log_bin_trust_routine_creators

若启用了二进制记录，则该变量适用。它控制是否可以信任保存的程序的作者不会创建向二进制日志写入不安全事件的程序。如果设置为0(默认情况)，不允许用户创建或修改保存的程序，除非他们不仅拥有CREATE ROUTINE或ALTER ROUTINE权限还拥有SUPER权限。

设置为0还强制限制，程序必须用DETERMINISTIC 特征或用READS SQL DATA或NO SQL特征声明。如果变量设置为1,MySQL不对保存程序的创建强加限制。

参见20.4节，“存储子程序和触发程序的二进制日志功能”。

- log_error

错误日志的位置。

- log_slave_updates

是否从服务器从主服务器收到的更新应记入从服务器自己的二进制日志。要想生效，必须启用从服务器的二进制记录。参见6.8节，“复制启动选项”。

- log_slow_queries

是否记录慢查询。用long_query_time变量的值来确定"慢查询"。参见5.11.4节，"慢速查询日志"。

- log_warnings

是否产生其它警告消息。默认情况下启用。放弃的连接不记入错误日志，除非值大于1。

- long_query_time

如果查询时间超过该值，则增加Slow_queries状态变量。如果你正使用--log-slow-queries选项，则查询记入慢查询日志文件。用实际时间测量该值，而不是CPU时间，因此低于轻负载系统阈值的查询可能超过重负载系统的阈值。参见5.11.4节，"慢速查询日志"。

- low_priority_updates

如果设置为1，所有INSERT、UPDATE、DELETE和LOCK TABLE WRITE语句将等待直到受影响的表没有挂起的SELECT或LOCK TABLE READ。该变量以前叫做sql_low_priority_updates。

- lower_case_file_system

该变量说明是否数据目录所在的文件系统对文件名的大小写敏感。ON说明对文件名的大小写不敏感，OFF表示敏感。

- lower_case_table_names

如果设置为1,表名用小写保存到硬盘上，并且表名比较时不对大小写敏感。如果设置为2，按照指定的保存表名，但按照小写来比较。该选项还适合数据库名和表的别名。参见9.2.2节，"识别符大小写敏感性"。

如果你正使用InnoDB表，你应在所有平台上将该变量设置为1，强制将名字转换为小写。

如果运行MySQL的系统对文件名的大小写不敏感(例如Windows或Mac OS X)，你不应将该变量设置为0。如果启动时没有设置该变量，并且数据目录所在文件系统对文件名的大小写不敏感，MySQL自动将lower_case_table_names设置为2。

- max_allowed_packet

包或任何生成的/中间字符串的最大大小。

包消息缓冲区初始化为net_buffer_length字节，但需要时可以增长到max_allowed_packet字节。该值默认很小，以捕获大的(可能是错误的)数据包。

如果你使用大的BLOB 列或长字符串，你必须增加该值。应同你想要使用的最大的BLOB一样大。max_allowed_packet的协议限制为1GB。

- max_binlog_cache_size

如果多语句事务需要更大的内存，你会得到错误Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage。

- max_binlog_size

如果二进制日志写入的内容超出给定值，日志就会发生滚动。你不能将该变量设置为大于1GB或小于4096字节。 默认值是1GB。

请注意如果你正使用事务：事务以一个块写入二进制日志，因此不不能被几个二进制日志拆分。因此，如果你有大的事务，二进制日志可能会大于max_binlog_size。

如果max_relay_log_size为0, max_binlog_size的值也适用于中继日志。

- max_connect_errors

如果中断的与主机的连接超过该数目，该主机则阻塞后面的连接。你可以用 FLUSH HOSTS语句解锁锁定的主机。

- max_connections

允许的并行客户端连接数目。增大该值则增加**mysqld** 需要的文件描述符的数量。关于文件描述符限制的注释参见7.4.9节，"MySQL如何打开和关闭表"。还可参见A.2.6节，"连接数过多"。

- max_delayed_threads

不要启动大于该数目的线程来处理INSERT DELAYED语句。如果所有INSERT DELAYED线程已经在使用，你想在新表中插入数据，行 插入时好像未指定DELAYED属性。如果你将该值设置为0,MySQL不会创建线程来处理DELAYED行；其结果是完全禁用了DELAYED。

- max_error_count

保存由SHOW ERRORS或SHOW WARNINGS显示的错误、警告和注解的最大数目。

- max_heap_table_size

该变量设置MEMORY (HEAP)表可以增长到的最大空间大小。该变量用来计算MEMORY表的MAX_ROWS值。在已有的MEMORY表上设置该变量没有效果，除非用CREATE TABLE或TRUNCATE TABLE等语句重新创建表。

- max_insert_delayed_threads

该变量为max_delayed_threads的同义词。

- max_join_size

不允许可能需要检查多于max_join_size行(为单个表语句)或行组合(为多个表语句)或可能执行大于max_join_size次硬盘查询的SELECT语句。通过设置该值，你可以捕获键使用不正确并可能花很长时间的SELECT语句。如果用户想要执行没有WHERE子句的花较长时间或返回数百万行的联接，则设置它。

将该变量设置为DEFAULT之外的值，将SQL_BIG_SELECTS的值重设为0。如果你重新设置SQL_BIG_SELECTS值，max_join_size变量被忽略。

如果查询结果位于查询缓存中，则不检查结果大小，因为前面已经计算了结果，不会要求服务器将它发送给客户端。

该变量以前叫做sql_max_join_size。

· max_length_for_sort_data

确定使用的filesort算法的索引值大小的限值。参见7.2.12节，“MySQL如何优化ORDER BY”。

· max_relay_log_size

如果复制从服务器写入中继日志时超出给定值，则滚动中继日志。通过该变量你可以对中继日志和二进制日志设置不同的限制。但是，将该变量设置为0，MySQL可以对二进制日志和中继日志使用max_binlog_size。max_relay_log_size必须设置在4096字节和1GB(包括)之间，或为0。 默认值是0。参见6.3节，“复制实施细节”。

· max_seeks_for_key

限制根据键值寻找行时的最大搜索数。MySQL优化器假定当用扫描键在表内搜索匹配的行时，不需要超过该数量的键值搜索，而不管键的实际基数是什么(参见13.5.4.11节，“SHOW INDEX语法”)。将该值设置为较低的值(100?)，你可以强制MySQL选择键值而不选择表扫描。

· max_sort_length

当排序BLOB或TEXT值时使用的字节数。只使用每个值的前max_sort_length字节；其它的被忽略。

· max_tmp_tables

客户端可以同时打开的临时表的最大数。(但该选项还未生效）。

· max_user_connections

任何给定的MySQL账户允许的最大同时连接数。0值表示“没有限制”。

该变量具有全局范围和(只读)会话范围。会话变量的的值与全局变量的值相同，除非当前账户具有非零MAX_USER_CONNECTIONS资源限制。在这种情况下，会话值反应了账户限制。

· max_write_lock_count

超过写锁定限制后，允许部分读锁定。

· myisam_data_pointer_size

默认指针大小，单位是字节，当未指定MAX_ROWS选项时，CREATE TABLE使用该变量创建MyISAM表。该变量不能小于2或大于7。 默认值是6。参见A.2.11节，“表已满”。

· (*DEPRECATED*) myisam_max_extra_sort_file_size

注释：MySQL 5.1不支持该变量。详细信息参见*MySQL 5.0* 参考手册。

· myisam_max_sort_file_size

重建MyISAM索引(在REPAIR TABLE、ALTER TABLE或LOAD DATA INFILE过程中)时，允许MySQL使用的临时文件的最大空间大小。如果文件的大小超过该值，则使用键值缓存创建索引，要慢得多。该值的单位为字节。

· myisam_recover_options

--myisam-recover选项的值。

· myisam_repair_threads

如果该值大于1，在Repair by sorting过程中并行创建MyISAM表索引(每个索引在自己的线程内)。 默认值是1。注释：多线程维护仍然是*alpha* 编码。

· myisam_sort_buffer_size

当在REPAIR TABLE或用CREATE INDEX创建索引或ALTER TABLE过程中排序 MyISAM索引分配的缓冲区。

· myisam_stats_method

当为MyISAM表搜集关于索引值分发的统计信息时服务器如何处理NULL值。该变量有两个可能的值，nulls_equal和nulls_unequal。对于nulls_equal，认为所有NULL索引值时相等的，并形成一个数值组，其空间大小等于NULL值的数。对于nulls_unequal，NULL值认为是不相等的，每个NULL形成一个数值组，大小为1。

方法用于生成表统计信息，影响优化器如何选择索引来执行查询，详细描述见7.4.7节，“MyISAM索引统计集合”。

· multi_read_range

指定范围选择过程中发送到存储引擎的范围的最大值。默认值是256。向引擎发送多个范围可以大大改进某些选择的性能，特别是对NDBCLUSTER。该引擎需

要向所有节点发送范围请求，同时发送许多请求可以大大降低通信成本。

· named_pipe

(只适用Windows)说明服务器是否支持命名管道连接。

· net_buffer_length

在查询之间将通信缓冲区重设为该值。一般情况不应改变，但如果内存很小，可以将它设置为期望的客户端发送的SQL语句的长度。如果语句超出该长度，缓冲区自动扩大，直到max_allowed_packet字节。

· net_read_timeout

中断读前等待连接的其它数据的秒数。当服务器从客户端读数时，net_read_timeout指控制何时中断的超时值。当服务器向客户端写时，net_write_timeout指控制何时中断的超时值。又见slave_net_timeout。

· net_retry_count

如果某个通信端口的读操作中断了，在放弃前重试多次。在FreeBSD中该值应设得很高，因为内部中断将发送至所有线程。

· net_write_timeout

中断写之前等待块写入连接的秒数。又见net_read_timeout。

· new

在MySQL 4.0中使用该变量来打开4.1中的一些行为，并用于向后兼容性。在MySQL 5.1中,它的值一直是OFF.

· old_passwords

是否服务器应为MySQL用户账户使用pre-4.1-style密码。参见A.2.3节，“客户端不支持鉴定协议”。

· one_shot

这不是一个变量，但当设置变量是可以使用它。其描述见13.5.3节，“SET语法”。

· one_shot

这不是一个变量，但当设置变量是可以使用它。其描述见13.5.3节，“SET语法”。

· open_files_limit

操作系统允许**mysqld**打开的文件的数量。这是系统允许的实际值，可能与你在启动选项中赋给**mysqld**的值不同。若在系统中MySQL不能更改打开的文件的数量，则该值为0。

· optimizer_prune_level

在查询优化从优化器搜索空间裁减低希望局部计划中使用的控制方法。0值禁用该方法，以便优化器进行穷举搜索。值为1使优化器根据中间方案中得出的行数来裁减方案。

· optimizer_search_depth

查询优化器进行的搜索的最大深度。如果值大于查询中的关系数则查询方案比较佳，但生成查询执行方案需要的时间更长。值大于查询中的关系数则返回的执行方案更快，但方案远没有优化。如果设置为0, 系统自动选择合理的值。如果设置为查询中使用的表的最大数加2,优化器转换为MySQL 5.0.0(和以前的版本)中使用的算法并搜索。

· pid_file

进程ID (PID)文件的路径名。可以用--pid-file选项设置该变量。

· plugin_dir

插件目录的路径。在MySQL 5.1.2中加入了该变量。

· port

服务器帧听TCP/IP连接所用端口。可以用--port选项设置该变量。

· preload_buffer_size

重载索引时分配的缓冲区大小。

· protocol_version

MySQL服务器使用的客户端/服务器协议的版本。

· query_alloc_block_size

为查询分析和执行过程中创建的对象分配的内存块大小。如果内存分段过程中遇到问题，将该变量增加一位会有帮助。

· query_cache_limit

不要缓存大于该值的结果。默认值是1048576(1MB)。

· query_cache_min_res_unit

查询缓存分配的最小块的大小(字节)。 默认值是4096(4KB)。关于该变量的调节信息参见5.13.3节，“查询高速缓冲配置”。

· query_cache_size

为缓存查询结果分配的内存的数量。默认值是0，即禁用查询缓存。请注意即使query_cache_type设置为0也将分配此数量的内存。详细信息参见5.13.3节，“查询高速缓冲配置”。

· query_cache_type

设置查询缓存类型。设置GLOBAL值可以设置后面的所有客户端连接的类型。客户端可以设置SESSION值以影响他们自己对查询缓存的使用。下面的表显示了可能的值:

选项	描述
0或OFF	不要缓存或查询结果。请注意这样不会取消分配的查询缓存区。要想取消，你应将query_cache_size设置为0。
1或ON	缓存除了以SELECT SQL_NO_CACHE开头的所有查询结果。
2或DEMAND	只缓存以SELECT SQL_NO_CACHE开头的查询结果。

该变量默认设为ON。

· query_cache_wlock_invalidate

一般情况，当客户端对MyISAM表进行WRITE锁定时，如果查询结果位于查询缓存中，则其它客户端未被锁定，可以对该表进行查询。将该变量设置为1，则可以对表进行WRITE锁定，使查询缓存内所有对该表进行的查询变得非法。这样当锁定生效时，可以强制其它试图访问表的客户端来等待。

· query_prealloc_size

用于查询分析和执行的固定缓冲区的大小。在查询之间该缓冲区不释放。如果你执行复杂查询，分配更大的query_prealloc_size值可以帮助提高性能，因为它可以降低查询过程中服务器分配内存的需求。

· range_alloc_block_size

范围优化时分配的块的大小。

· read_buffer_size

每个线程连续扫描时为扫描的每个表分配的缓冲区的大小(字节)。如果进行多次连续扫描，可能需要增加该值， 默认值为131072。

· read_only

当变量对复制从服务器设置为ON时，从服务器不允许更新，除非通过从服务器的线程或用户拥有SUPER权限。可以确保从服务器不接受客户端的更新命令。

· relay_log_purge

当不再需要中继日志时禁用或启用自动清空中继日志。默认值是1(启用)。

· read_rnd_buffer_size

当排序后按排序后的顺序读取行时，则通过该缓冲区读取行，避免搜索硬盘。将该变量设置为较大的值可以大大改进ORDER BY的性能。但是，这是为每个客户端分配的缓冲区，因此你不应将全局变量设置为较大的值。相反，只为需要运行大查询的客户端更改会话变量。

· secure_auth

如果用--secure-auth选项启动了MySQL服务器，它将阻塞有旧格式(4.1之前)密码的所有账户所发起的连接。在这种情况下，该变量的值为ON，否则为OFF。

如果你想要防止使用旧格式的密码(致使网络通信不安全)，你应启用该选项。

如果启用该选项并且授权表为pre-4.1格式，服务器启动失败并且会出现错误。参见A.2.3节，“客户端不支持鉴定协议”。

当用于客户端选项时，如果服务器需要该客户端账户的旧格式的密码，则客户端拒绝连接该服务器。

· server_id

--server-id选项的值。用于主复制服务器和从复制服务器。

· shared_memory

(只用于Windows)服务器是否允许共享内存连接。

· shared_memory_base_name

(只用于Windows)说明服务器是否允许共享内存连接，并为共享内存设置识别符。当在单台机器上运行多个MySQL实例时很有用。

· skip_external_locking

如果**mysqld**使用外部锁定，该值为OFF。

- skip_networking

如果服务器只允许本地(非TCP/IP)连接，该值为ON。在Unix中，本地连接使用Unix套接字文件。在Windows中，本地连接使用命名管道或共享内存。在NetWare中，只支持TCP/IP连接，因此不要将该变量设置为ON。

- skip_show_database

防止不具有SHOW DATABASES权限的人们使用SHOW DATABASES语句。如果你担心用户能够看见属于其它用户的数据库，这样设置可以提高安全性。其效果取决于SHOW DATABASES权限：如果变量值为ON，只允许具有SHOW DATABASES权限的人们使用SHOW DATABASES 语句，并且该语句将显示所有数据库名。如果值为OFF，允许所有用户执行SHOW DATABASES，但只显示用户具有SHOW DATABASES或其它权限的数据库的名称。

- slave_compressed_protocol

如果主、从服务器均支持，确定是否使用从/主压缩协议。

- slave_load_tmpdir

从服务器为复制LOAD DATA INFILE语句创建临时文件的目录名。

- slave_net_timeout

放弃读操作前等待主/从连接的更多数据的等待秒数。

- slave_skip_errors

从服务器应跳过(忽视)的复制错误。

- slave_transaction_retries

如果由于ofInnoDB死锁或超过InnoDB的innodb_lock_wait_timeout或NDBCLUSTER的TransactionDeadlockDetectionTimeout或TransactionInactiveTimeout，复制从服务器SQL线程未能执行事务，在提示错误并停止前它自动重复slave_transaction_retries次。 默认值是10。

- slow_launch_time

如果创建线程的时间超过该秒数，服务器增加Slow_launch_threads状态变量。

- socket

*Unix*平台：用于本地客户端连接的套接字文件。默认为/var/lib/mysql/mysql.sock。

Windows：用于本地客户端连接的命名管道名。默认为mysql。

- sort_buffer_size

每个排序线程分配的缓冲区的大小。增加该值可以加快ORDER BY或GROUP BY操作。参见A.4.4节，“MySQL将临时文件储存在哪里”。

- sql_mode

当前的服务器SQL模式，可以动态设置。参见5.3.2节，“SQL服务器模式”。

- sql_slave_skip_counter

从服务器应跳过的来自主服务器的事件数。

- storage_engine

该变量是table_typeis的同义词。在MySQL 5.1中,首选storage_engine。

- sync_binlog

如果为正，当每个sync_binlog'th写入该二进制日志后，MySQL服务器将它的二进制日志同步到硬盘上(fdatasync())。请注意如果在autocommit模式，每执行一个语句向二进制日志写入一次，否则每个事务写入一次。 默认值是0，不与硬盘同步。值为1是最安全的选择，因为崩溃时，你最多丢掉二进制日志中的一个语句/事务；但是，这是最慢的选择(除非硬盘有电池备份缓存，从而使同步工作较快)。

- sync_frm

如果该变量设为1,当创建非临时表时它的.frm文件被同步到硬盘上(fdatasync())；这样较慢但出现崩溃时较安全。 默认值为1。

- system_time_zone

服务器系统时区。当 服务器开始执行时，它继承机器默认时区设置值，可以由运行服务器的账户或在启动脚本中进行修改。该值用来设置system_time_zone。典型情况用TZ环境变量来指定时区。还可以用**mysqld_safe**脚本的--timez选项来指定。

- table_cache

所有线程打开的表的数目。增大该值可以增加**mysqld**需要的文件描述符的数量。你可以检查Opened_tables状态变量来检查你是否需要增加表缓存。参见5.3.4节，“服务器状态变量”。如果Opened_tables值较大，并且多次执行FLUSH TABLES(只是强制关闭所有表并重新)，则应增加table_cache变量的值。

关于表缓存的详细信息，参见7.4.9节，“MySQL如何打开和关闭表”。

- table_type

默认表类型(存储引擎)。要想在服务器启动时设置表类型，使用--default-table-type选项。参见5.3.1节，“**mysqld**命令行选项”。

- thread_cache_size

服务器应缓存多少线程以便重新使用。当客户端断开连接时，如果线程少于thread_cache_size，则客户端的线程被放入缓存。当请求线程时如果允许可以从缓存中重新利用线程，并且只有当缓存空了时才会创建新线程。如果新连接很多，可以增加该变量以提高性能。(一般情况，如果线程执行得很好，性能提高不明显)。检查Connections和Threads_created状态变量的差(详见5.3.4节，“服务器状态变量”)，你可以看见线程缓存的效率。

- thread_concurrency

在Solaris中，**mysqld**用该值调用thr_setconcurrency()。该函数使应用程序向线程系统提供需要同时运行的期望的线程数目的提示。

- thread_stack

每个线程的堆栈大小。用crash-me测试检测出的许多限制取决于该值。 默认值足够大，可以满足普通操作。参见7.1.4节，“MySQL基准套件”。

- time_format

该变量为使用。

- time_zone

当前的时区。初使值是'SYSTEM'(使用system_time_zone的值)，但可以用--default-time-zone选项在服务器启动时显式指定。

- tmp_table_size

如果内存内的临时表超过该值，MySQL自动将它转换为硬盘上的MyISAM表。如果你执行许多高级GROUP BY查询并且有大量内存，则可以增加tmp_table_size的值。

- tmpdir

保存临时文件和临时表的目录。该变量可以设置为几个路径，按round-robin模式使用。在Unix中应该用冒号(‘:’)间隔开路径，在Windows、NetWare和OS/2中用分号(‘; ’)。

用来将负荷分散到几个物理硬盘上。如果MySQL服务器为复制从服务器，你不应将tmpdir设置为指向基于内存的文件系统上的目录或当服务器主机重启时声明的目录。复制从服务器需要部分临时文件来在机器重启后仍可用，以便它可以复制临时表或执行LOAD DATA INFILE操作。如果服务器重启时临时文件夹中的文件丢失了，则复制失败。但是，如果你使用MySQL 4.0.0或更新版本，你可以使用 slave_load_tmpdir变量设置从服务器的临时目录。在这种情况下，从服务器不再使用常规tmpdir，说明你可以将tmpdir设置到一个非固定位置。

- transaction_alloc_block_size

为保存将保存到二进制日志中的事务的查询而分配的内存块的大小(字节)。

- transaction_prealloc_size

为transaction_alloc_blocks分配的固定缓冲区的大小（字节），在两次查询之间不会释放。使该值足够大，将所有查询固定到一个事务中，可以避免多次malloc()调用。

- tx_isolation

默认事务隔离级别。默认值为REPEATABLE-READ。

- updatable_views_with_limit

该变量控制如果更新包含LIMIT子句，是否可以在当前表中使用不包含主关键字的视图进行更新。(通常用GUI工具生成这类更新)。更新指UPDATE或DELETE语句。这儿主关键字指PRIMARY KEY，或一个UNIQUE索引，其中任何列不可以包含NULL。

该变量有两个值:

- 1或YES：只发出警告(没有错误消息)。这是 默认值。
- 0或NO：禁止更新。

- version

服务器版本号。

- version_bdb

BDB存储引擎版本。

- version_comment

configure脚本有一个--with-comment选项，当构建MySQL时可以进行注释。该变量包含注释值。

- version_compile_machine

MySQL构建的机器或架构的类型。

- version_compile_os

MySQL构建的操作系统的类型。

- wait_timeout

服务器关闭非交互连接之前等待活动的秒数。

在线程启动时，根据全局wait_timeout值或全局interactive_timeout值初始化会话wait_timeout值，取决于客户端类型(由mysql_real_connect()的连接选项CLIENT_INTERACTIVE定义)。又见interactive_timeout。

5.3.3.1. 动态系统变量

许多服务器系统变量是动态的，可以使用SET GLOBAL或SET SESSION在运行时设置。你还可以使用SELECT获得它们的值。参见9.4节，“系统变量”。

下面的表列出了所有动态系统变量。最后1列说明每个变量是否适用GLOBAL或SESSION(或二者)。

变量名	值类型	类型
autocommit	boolean	SESSION
big_tables	boolean	SESSION
binlog_cache_size	numeric	GLOBAL
bulk_insert_buffer_size	numeric	GLOBAL | SESSION
character_set_client	string	GLOBAL | SESSION
character_set_connection	string	GLOBAL | SESSION
character_set_results	string	GLOBAL | SESSION
character_set_server	string	GLOBAL | SESSION
collation_connection	string	GLOBAL | SESSION
collation_server	string	GLOBAL | SESSION
completion_type	numeric	GLOBAL | SESSION
concurrent_insert	boolean	GLOBAL
connect_timeout	numeric	GLOBAL
convert_character_set	string	GLOBAL | SESSION
default_week_format	numeric	GLOBAL | SESSION
delay_key_write	OFF | ON | ALL	GLOBAL
delayed_insert_limit	numeric	GLOBAL
delayed_insert_timeout	numeric	GLOBAL
delayed_queue_size	numeric	GLOBAL
div_precision_increment	numeric	GLOBAL | SESSION
engine_condition_pushdown	boolean	GLOBAL | SESSION
error_count	numeric	SESSION
expire_logs_days	numeric	GLOBAL
flush	boolean	GLOBAL
flush_time	numeric	GLOBAL
foreign_key_checks	boolean	SESSION
ft_boolean_syntax	numeric	GLOBAL
group_concat_max_len	numeric	GLOBAL | SESSION
identity	numeric	SESSION
innodb_autoextend_increment	numeric	GLOBAL
innodb_concurrency_tickets	numeric	GLOBAL
innodb_max_dirty_pages_pct	numeric	GLOBAL
innodb_max_purge_lag	numeric	GLOBAL
innodb_support_xa	boolean	GLOBAL | SESSION
innodb_sync_spin_loops	numeric	GLOBAL
innodb_table_locks	boolean	GLOBAL | SESSION
innodb_thread_concurrency	numeric GLOBAL	
innodb_thread_sleep_delay	numeric GLOBAL	
insert_id	boolean	SESSION
interactive_timeout	numeric	GLOBAL | SESSION
join_buffer_size	numeric	GLOBAL | SESSION
key_buffer_size	numeric	GLOBAL
last_insert_id	numeric	SESSION
local_infile	boolean	GLOBAL
log_warnings	numeric	GLOBAL
long_query_time	numeric	GLOBAL | SESSION
low_priority_updates	boolean	GLOBAL | SESSION
max_allowed_packet	numeric	GLOBAL | SESSION
max_binlog_cache_size	numeric	GLOBAL
max_binlog_size	numeric	GLOBAL

max_connect_errors	numeric	GLOBAL
max_connections	numeric	GLOBAL
max_delayed_threads	numeric	GLOBAL
max_error_count	numeric	GLOBAL | SESSION
max_heap_table_size	numeric	GLOBAL | SESSION
max_insert_delayed_threads	numeric	GLOBAL
max_join_size	numeric	GLOBAL | SESSION
max_relay_log_size	numeric	GLOBAL
max_seeks_for_key	numeric	GLOBAL | SESSION
max_sort_length	numeric	GLOBAL | SESSION
max_tmp_tables	numeric	GLOBAL | SESSION
max_user_connections	numeric	GLOBAL
max_write_lock_count	numeric	GLOBAL
myisam_stats_method	enum	GLOBAL | SESSION
multi_read_range	numeric	GLOBAL | SESSION
myisam_data_pointer_size	numeric	GLOBAL
log_bin_trust_routine_creators	boolean	GLOBAL
myisam_max_sort_file_size	numeric	GLOBAL | SESSION
myisam_repair_threads	numeric	GLOBAL | SESSION
myisam_sort_buffer_size	numeric	GLOBAL | SESSION
net_buffer_length	numeric	GLOBAL | SESSION
net_read_timeout	numeric	GLOBAL | SESSION
net_retry_count	numeric	GLOBAL | SESSION
net_write_timeout	numeric	GLOBAL | SESSION
old_passwords	numeric	GLOBAL | SESSION
optimizer_prune_level	numeric	GLOBAL | SESSION
optimizer_search_depth	numeric	GLOBAL | SESSION
preload_buffer_size	numeric	GLOBAL | SESSION
query_alloc_block_size	numeric	GLOBAL | SESSION
query_cache_limit	numeric	GLOBAL
query_cache_size	numeric	GLOBAL
query_cache_type	enumeration	GLOBAL | SESSION
query_cache_wlock_invalidate	boolean	GLOBAL | SESSION
query_prealloc_size	numeric	GLOBAL | SESSION
range_alloc_block_size	numeric	GLOBAL | SESSION
read_buffer_size	numeric	GLOBAL | SESSION
read_only	numeric	GLOBAL
read_rnd_buffer_size	numeric	GLOBAL | SESSION
rpl_recovery_rank	numeric	GLOBAL
safe_show_database	boolean	GLOBAL
secure_auth	boolean	GLOBAL
server_id	numeric	GLOBAL
slave_compressed_protocol	boolean	GLOBAL
slave_net_timeout	numeric	GLOBAL
slave_transaction_retries	numeric	GLOBAL
slow_launch_time	numeric	GLOBAL
sort_buffer_size	numeric	GLOBAL | SESSION
sql_auto_is_null	boolean	SESSION
sql_big_selects	boolean	SESSION
sql_big_tables	boolean	SESSION
sql_buffer_result	boolean	SESSION
sql_log_bin	boolean	SESSION
sql_log_off	boolean	SESSION
sql_log_update	boolean	SESSION
sql_low_priority_updates	boolean	GLOBAL | SESSION
sql_max_join_size	numeric	GLOBAL | SESSION
sql_mode	enumeration	GLOBAL | SESSION
sql_notes	boolean	SESSION
sql_quote_show_create	boolean	SESSION
sql_safe_updates	boolean	SESSION
sql_select_limit	numeric	SESSION
sql_slave_skip_counter	numeric	GLOBAL
updatable_views_with_limit	enumeration	GLOBAL | SESSION
sql_warnings	boolean	SESSION
sync_binlog	numeric	GLOBAL

sync_frm	boolean	GLOBAL
storage_engine	enumeration	GLOBAL | SESSION
table_cache	numeric	GLOBAL
table_type	enumeration	GLOBAL | SESSION
thread_cache_size	numeric	GLOBAL
time_zone	string	GLOBAL | SESSION
timestamp	boolean	SESSION
tmp_table_size	enumeration	GLOBAL | SESSION
transaction_alloc_block_size	numeric	GLOBAL | SESSION
transaction_prealloc_size	numeric	GLOBAL | SESSION
tx_isolation	enumeration	GLOBAL | SESSION
unique_checks	boolean	SESSION
wait_timeout	numeric	GLOBAL | SESSION
warning_count	numeric	SESSION

标记为**string**的变量采用字符串值。标记为**numeric**的变量采用数字值。标记为**boolean**的变量可以设置为0、1、ON或OFF。标记为**enumeration**的变量一般情况应设置为该变量的某个可用值，但还可以设置为对应期望的枚举值的数字。对于枚举系统变量，第1个枚举值应对应0。这不同于ENUM列，第1个枚举值对应1。

5.3.4. 服务器状态变量

服务器维护许多提供操作相关信息的状态变量。你可以通过SHOW STATUS语句查看这些变量和它们的值：

```
mysql> SHOW STATUS;
+-----------------------------------+------------+
| Variable_name                     | Value      |
+-----------------------------------+------------+
| Aborted_clients                   | 0          |
| Aborted_connects                  | 0          |
| Bytes_received                    | 155372598  |
| Bytes_sent                        | 1176560426 |

…

| Connections                       | 30023      |
| Created_tmp_disk_tables           | 0          |
| Created_tmp_files                 | 3          |
| Created_tmp_tables                | 2          |

…

| Threads_created                   | 217        |
| Threads_running                   | 88         |
| Uptime                            | 1389872    |
+-----------------------------------+------------+
```

用FLUSH STATUS语句可以将许多状态变量重设为0。

状态变量有以下含义。没有指示版本的变量在MySQL 5.1之前已经出现。关于它们的使用历史，参见*MySQL 5.0*参考手册。

- Aborted_clients

由于客户端没有正确关闭连接导致客户端终止而中断的连接数。参见A.2.10节，“通信错误和失效连接”。

- Aborted_connects

试图连接到MySQL服务器而失败的连接数。参见A.2.10节，“通信错误和失效连接”。

- Binlog_cache_disk_use

使用临时二进制日志缓存但超过binlog_cache_size值并使用临时文件来保存事务中的语句的事务数量。

- Binlog_cache_use

使用临时二进制日志缓存的事务数量。

- Bytes_received

从所有客户端接收到的字节数。

- Bytes_sent

发送给所有客户端的字节数。

- Com_*xxx*

Com_*xxx* 语句计数变量表示每个*xxx* 语句执行的次数。每类语句有一个状态变量。例如，Com_delete和Com_insert分别统计DELETE 和INSERT语句执行的次数。

Com_stmt_*xxx*状态变量为：

- Com_stmt_prepare
- Com_stmt_execute
- Com_stmt_fetch
- Com_stmt_send_long_data
- Com_stmt_reset
- Com_stmt_close

这些变量代表准备好的语句命令。它们的名字对应网络层使用的COM_*xxx* 命令系列；换句话说：当准备好的语句API调用如**mysql_stmt_prepare()**、**mysql_stmt_**执行**()**并执行时，它们的值增加。但是，当执行下面的SQL语句时，Com_stmt_prepare, Com_stmt_execute和Com_stmt_close也增加：PREPARE、EXECUTE或DEALLOCATE PREPARE。此外，旧(从MySQL 4.1.3起可用)语句计数变量Com_prepare_sql、Com_execute_sql和Com_dealloc_sql的值也随PREPARE、EXECUTE和DEALLOCATE PREPARE语句增加。Com_stmt_fetch代表通过光标获取的网络round-trips的总数量。

所有Com_stmt_*xxx*变量将增加，即使语句参数未知或执行过程中出现错误。换句话说，它们的值对应发出的请求数，而不是成功完成的请求数。

- Connections

试图连接到(不管是否成功)MySQL服务器的连接数。

- Created_tmp_disk_tables

服务器执行语句时在硬盘上自动创建的临时表的数量。

- Created_tmp_files

mysqld已经创建的临时文件的数量。

- Created_tmp_files

服务器执行语句时自动创建的内存中的临时表的数量。如果Created_tmp_disk_tables较大，你可能要增加tmp_table_size值使临时 表基于内存而不基于硬盘。

- Delayed_errors

用INSERT DELAYED写的出现错误的行数(可能为duplicate key)。

- Delayed_insert_threads

使用的INSERT DELAYED处理器线程数。

- Delayed_writes

写入的INSERT DELAYED行数。

- Flush_commands

执行的FLUSH语句数。

- Handler_commit

内部提交语句数。

- Handler_discover

MySQL服务器可以问NDB CLUSTER存储引擎是否知道某一名字的表。这被称作发现。Handler_discover说明通过该方法发现的次数。

- Handler_delete

行从表中删除的次数。

- Handler_read_first

索引中第一条被读的次数。如果较高，它建议服务器正执行大量全索引扫描；例如，SELECT col1 FROM foo，假定col1有索引。

- Handler_read_key

根据键读一行的请求数。如果较高，说明查询和表的索引正确。

- Handler_read_next

按照键顺序读下一行的请求数。如果你用范围约束或如果执行索引扫描来查询索引列，该值增加。

- Handler_read_prev

按照键顺序读前一行的请求数。该读方法主要用于优化ORDER BY ... DESC。

- Handler_read_rnd

根据固定位置读一行的请求数。如果你正执行大量查询并需要对结果进行排序该值较高。你可能使用了大量需要MySQL扫描整个表的查询或你的连接没有正确使用键。

- Handler_read_rnd_next

在数据文件中读下一行的请求数。如果你正进行大量的表扫描，该值较高。通常说明你的表索引不正确或写入的查询没有利用索引。

- Handler_rollback

内部ROLLBACK语句的数量。

- Handler_update

在表内更新一行的请求数。

- Handler_write

在表内插入一行的请求数。

- Innodb_buffer_pool_pages_data

包含数据的页数(脏或干净)。

- Innodb_buffer_pool_pages_dirty

当前的脏页数。

- Innodb_buffer_pool_pages_flushed

要求清空的缓冲池页数。

- Innodb_buffer_pool_pages_free

空页数。

- Innodb_buffer_pool_pages_latched

在InnoDB缓冲池中锁定的页数。这是当前正读或写或由于其它原因不能清空或删除的页数。

- Innodb_buffer_pool_pages_misc

忙的页数，因为它们已经被分配优先用作管理，例如行锁定或适用的哈希索引。该值还可以计算为Innodb_buffer_pool_pages_total - Innodb_buffer_pool_pages_free - Innodb_buffer_pool_pages_data。

- Innodb_buffer_pool_pages_total

缓冲池总大小（页数）。

- Innodb_buffer_pool_read_ahead_rnd

InnoDB初始化的“随机”read-aheads数。当查询以随机顺序扫描表的一大部分时发生。

- Innodb_buffer_pool_read_ahead_seq

InnoDB初始化的顺序read-aheads数。当InnoDB执行顺序全表扫描时发生。

- Innodb_buffer_pool_read_requests

InnoDB已经完成的逻辑读请求数。

- Innodb_buffer_pool_reads

不能满足InnoDB必须单页读取的缓冲池中的逻辑读数量。

- Innodb_buffer_pool_wait_free

一般情况，通过后台向InnoDB缓冲池写。但是，如果需要读或创建页，并且没有干净的页可用，则它还需要先等待页面清空。该计数器对等待实例进行记数。如果已经适当设置缓冲池大小，该值应小。

- Innodb_buffer_pool_write_requests

向InnoDB缓冲池的写数量。

- Innodb_data_fsyncs

fsync()操作数。

- Innodb_data_pending_fsyncs

当前挂起的fsync()操作数。

- Innodb_data_pending_reads

当前挂起的读数。

- Innodb_data_pending_writes

当前挂起的写数。

- Innodb_data_read

至此已经读取的数据数量（字节）。

- Innodb_data_reads

数据读总数量。

- Innodb_data_writes

数据写总数量。

- Innodb_data_written

至此已经写入的数据量（字节）。

- Innodb_dblwr_writes, Innodb_dblwr_pages_written

已经执行的双写操作数量和为此目的已经写好的页数。参见15.2.14.1节，“磁盘I/O”。

- Innodb_log_waits

我们必须等待的时间，因为日志缓冲区太小，我们在继续前必须先等待对它清空。

- Innodb_log_write_requests

日志写请求数。

- Innodb_log_writes

向日志文件的物理写数量。

- Innodb_os_log_fsyncs

向日志文件完成的fsync()写数量。

- Innodb_os_log_pending_fsyncs

挂起的日志文件fsync()操作数量。

- Innodb_os_log_pending_writes

挂起的日志文件写操作。

- Innodb_os_log_written

写入日志文件的字节数。

- Innodb_page_size

编译的InnoDB页大小(默认16KB)。许多值用页来记数；页的大小很容易转换为字节。

- Innodb_pages_created

创建的页数。

- Innodb_pages_read

读取的页数。

- Innodb_pages_written

写入的页数。

- Innodb_row_lock_current_waits

当前等待的待锁定的行数。

- Innodb_row_lock_time

行锁定花费的总时间，单位毫秒。

- Innodb_row_lock_time_avg

行锁定的平均时间，单位毫秒。

- Innodb_row_lock_time_max

行锁定的最长时间，单位毫秒。

- Innodb_row_lock_waits

一行锁定必须等待的时间数。

- Innodb_rows_deleted

从InnoDB表删除的行数。

- Innodb_rows_inserted

插入到InnoDB表的行数。

- Innodb_rows_read

从InnoDB表读取的行数。

- Innodb_rows_updated

InnoDB表内更新的行数。

- Key_blocks_not_flushed

键缓存内已经更改但还没有清空到硬盘上的键的数据块数量。

- Key_blocks_unused

键缓存内未使用的块数量。你可以使用该值来确定使用了多少键缓存；参见5.3.3节，“服务器系统变量”中Key_buffer_size的讨论。

- Key_blocks_used

键缓存内使用的块数量。该值为高水平线标记，说明已经同时最多使用了多少块。

- Key_read_requests

从缓存读键的数据块的请求数。

- Key_reads

从硬盘读取键的数据块的次数。如果Key_reads较大，则Key_buffer_size值可能太小。可以用Key_reads/Key_read_requests计算缓存损失率。

- Key_write_requests

将键的数据块写入缓存的请求数。

- Key_writes

向硬盘写入将键的数据块的物理写操作的次数。

- Last_query_cost

用查询优化器计算的最后编译的查询的总成本。用于对比同一查询的不同查询方案的成本。默认值0表示还没有编译查询。 默认值是0。Last_query_cost具有会话范围。

- Max_used_connections

服务器启动后已经同时使用的连接的最大数量。

- Not_flushed_delayed_rows

等待写入INSERT DELAY队列的行数。

- Open_files

打开的文件的数目。

- Open_streams

打开的流的数量(主要用于记录)。

- Open_tables

当前打开的表的数量。

- Opened_tables

已经打开的表的数量。如果Opened_tables较大，table_cache 值可能太小。

- QCACHE_free_blocks

查询缓存内自由内存块的数量。

- QCACHE_free_memory

用于查询缓存的自由内存的数量。

- QCACHE_hits

查询缓存被访问的次数。

- QCACHE_inserts

加入到缓存的查询数量。

- QCACHE_lowmem_prunes

由于内存较少从缓存删除的查询数量。

- QCACHE_not_cached

非缓存查询数(不可缓存，或由于query_cache_type设定值未缓存)。

- Qcache_queries_in_cache

登记到缓存内的查询的数量。

- Qcache_total_blocks

查询缓存内的总块数。

- Questions

已经发送给服务器的查询的个数。

- Rpl_status

失败安全复制状态(还未使用)。

- Select_full_join

没有使用索引的联接的数量。如果该值不为0,你应仔细检查表的索引。

- Select_full_range_join

在引用的表中使用范围搜索的联接的数量。

- Select_range

在第一个表中使用范围的联接的数量。一般情况不是关键问题，即使该值相当大。

- Select_range_check

在每一行数据后对键值进行检查的不带键值的联接的数量。如果不为0，你应仔细检查表的索引。

- Select_scan

对第一个表进行完全扫描的联接的数量。

- Slave_open_temp_tables

当前由从SQL线程打开的临时表的数量。

- Slave_running

如果该服务器是连接到主服务器的从服务器，则该值为ON。

- Slave_retried_transactions

启动后复制从服务器SQL线程尝试事务的总次数。

- Slow_launch_threads

创建时间超过slow_launch_time秒的线程数。

- Slow_queries

查询时间超过long_query_time秒的查询的个数。参见5.11.4节，“慢速查询日志”。

- Sort_merge_passes

排序算法已经执行的合并的数量。如果这个变量值较大，应考虑增加sort_buffer_size系统变量的值。

- Sort_range

在范围内执行的排序的数量。

- Sort_rows

已经排序的行数。

- Sort_scan

通过扫描表完成的排序的数量。

- Ssl_*xxx*

用于SSL连接的变量。

- Table_locks_immediate

立即获得的表的锁的次数。

- Table_locks_waited

不能立即获得的表的锁的次数。如果该值较高，并且有性能问题，你应首先优化查询，然后拆分表或使用复制。

- Threads_cached

线程缓存内的线程的数量。

- Threads_connected

当前打开的连接的数量。

- Threads_created

创建用来处理连接的线程数。如果Threads_created较大，你可能要增加thread_cache_size值。缓存访问率的计算方法Threads_created/Connections。

- Threads_running

激活的（非睡眠状态）线程数。

- Uptime

服务器已经运行的时间（以秒为单位）。

5.4. mysql_fix_privilege_tables：升级MySQL系统表

一些MySQL发布对mysql数据库中的系统表的结构进行了更改，添加了新权限或特性。当你更新到新版本MySQL，你应同时更新系统表，以确保它们的结构最新。首先备份mysql数据库，然后按照下面的程序操作。

在Unix或Unix类系统中，运行**mysql_fix_privilege_tables**脚本来更新系统表：

```
shell> mysql_fix_privilege_tables
```

你必须在服务器运行时执行该脚本。它试图连接本机上用root运行的服务器。如果root账户需要密码，在命令行中按下述方法给出密码：

```
shell> mysql_fix_privilege_tables--password=root_password
```

mysql_fix_privilege_tables脚本可以执行将系统表转换为当前格式的任何动作。运行时你可能会看见一些Duplicate column name警告；你可以忽略它们。

运行完脚本后，停止服务器并重启。

在Windows系统中，MySQL分发包括mysql_fix_privilege_tables.sql SQL脚本，你可以用**mysql**客户端来运行。例如，如果MySQL安装到C:\Program Files\MySQL\MySQL Server 5.1，命令应为：

```
C:\> C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql -u root -p mysql
mysql> SOURCE C:/Program Files/MySQL/MySQL Server 5.1/scripts/mysql_fix_privilege_tables.sql
```

如果安装到其它目录，相应地更改路径名。

mysql命令将提示输入root密码；按照提示输入密码。

在Unix中，当**mysql**处理mysql_fix_privilege_tables.sql script脚本中的语句时，你可能会看见一些Duplicate column name警告；你可以忽略它们。

运行完脚本后，停止服务器并重启。

5.5. MySQL服务器关机进程

1. 服务器关闭进程可以概括为：
   1. 启动关闭进程
   2. 服务器根据需要创建关闭线程
   3. 服务器停止接收新连接
   4. 服务器终止当前的活动
   5. 存储引擎被停掉或关闭
   6. 服务器退出

   下面是更详细的描述：

   1. 启动关闭进程。

   可以用多种方法启动服务器的关闭。例如，拥有SHUTDOWN权限的用户可以执行**mysqladmin shutdown**命令。**mysqladmin**可以用于所有支持MySQL的平台上。其它操作系统相关的关闭开始方法还可能有：在Unix中，当接收到SIGTERM信号后，服务器关闭。对于在Windows中作为服务运行的服务器，当服务管理器让它关闭时，则关闭。

   2. 服务器根据需要创建关闭线程。

   根据开始关闭的方式，服务器可以创建线程来处理关闭进程。如果客户端需要关闭，则创建关闭线程。如果收到SIGTERM信号后关闭，信号线程可以自己关闭，或者创建单独的线程来完成。如果服务器尝试创建关闭线程而不能创建(例如，如果内存被耗尽)，它在错误日志中给出诊断消息：

   ```
   Error: Can't create thread to kill server
   ```

   3. 服务器停止接收新连接。

   在关闭过程中要想防止启动新活动，服务器停止接收新的客户端连接。它将关闭它帧听的网络连接：TCP/IP端口、Unix套接字文件、Windows命名管道和在Windows中的共享内存。

   4. 服务器终止当前的活动。

   对于每个与客户端连接相关的线程，与客户端的连接被中断，线程被标记为"杀掉的"。当线程注意到此类标记后则线程终止。空闲连接的线程很快终止。当前正处理查询的线程定期检查它们的状态，终止的时间较长。关于线程终止的详细信息，参见13.5.5.3节，"KILL语法"，特别是关于对MyISAM表的杀掉的REPAIR TABLE或OPTIMIZE TABLE操作。

   对于有打开事务的线程，事务被回滚。请注意如果某个线程正在更新非事务表，多行UPDATE或INSERT等操作会使表部分更新，因为操作在完成前会终止。

   如果服务器是主复制服务器，与当前连接的从服务器相关的线程的处理方式同其它客户端线程。即每个线程被标记为杀掉的，在下次检查他的状态后会退出。

   如果服务器是从复制服务器，在客户端线程标记为杀掉的之前，激活的I/O和SQL线程被停止。SQL线程允许先结束它当前的语句(以避免造成复制问题)然后停止。如果此时SQL线程正位于事务中部，事务则 回滚。

   5. 存储引擎被停掉或关闭。

   在该阶段，表缓存被清空，所有打开的表被关闭。

   每个存储引擎执行它管理的表需要的任何动作。例如，MyISAM清空任何挂起的表索引写操作。InnoDB将它的缓冲池清空到硬盘上（除非innodb_fast_shutdown为2），将当前的LSN写入表内，并终止自己的内部线程。

   6. 服务器退出。

5.6. 一般安全问题

5.6.1. 通用安全指南

5.6.2. 使MySQL在攻击者面前保持安全
5.6.3. Mysqld安全相关启动选项
5.6.4. LOAD DATA LOCAL安全问题

本节描述一些常见的需要注意的安全问题，以及一些可以使你的MySQL安装更加安全以防止黑客和误用的措施。关于MySQL用于设置用户账户并检查数据库访问的访问控制系统的具体信息，参见5.7节，“MySQL访问权限系统”。

5.6.1. 通用安全指南

任何在与Internet联网的计算机上使用MySQL的用户都应仔细阅读本节，以避免最常见的安全问题。

讨论安全时，我们强调必须完全保护整个服务器主机的安全(而不只是MySQL服务器)防范各种类型的可能的攻击：偷听、修改、重放和拒绝服务。我们在这里不能覆盖各方面的内容和措施。

MySQL根据访问控制列表(ACL)对所有连接、查询和其它用户尝试执行的操作进行安全管理。MySQL客户端和服务器之间还支持SSL-加密连接。这儿讨论的许多概念并不是MySQL专有的；该思想几乎同样适合所有应用程序。

运行MySQL时，应尽量遵从下面的指导：

- 不要让任何人**(**除了**MySQL root**账户**)**访问**mysql**数据库中的**user**表！这很关键。加密的密码才是**MySQL**中的真正的密码。知道user表中所列的密码并且能访问该账户客访问的主机的人可以很容易地用该用户登录。

- 学习MySQL访问权限系统。用GRANT和REVOKE语句来控制对MySQL的访问。不要授予超过需求的权限。决不能为所有主机授权。

检查清单：

o 试试mysql -u root。如果你能够成功连接服务器而没有要任何密码，则说明有问题。任何人可以作为MySQLroot用户用它的全部权限来连接MySQL服务器！查阅MySQL安装说明，应特别注意关于设置root密码的信息。参见2.9.3节，“使初始MySQL账户安全”。

o 通过SHOW GRANTS语句检查查看谁已经访问了什么。然后使用REVOKE语句删除不再需要的权限。

- 不要将纯文本密码保存到数据库中。如果你的计算机有安全危险，入侵者可以获得所有的密码并使用它们。相反，应使用MD5()、SHA1()或单向哈希函数。

- 不要从词典中选择密码。有专门的程序可以破解它们。即使象“xfish98”这样的密码也很差。而“duag98”要好得多，虽然包含了相同的字“fish”，但从标准QWERTY键盘向左输入。另一种方法是使用“Mhall”，来自句子“Mary had a little lamb.”中每个字的第一个字符。这样很容易记住并输入，但是不知道的人很难猜出来。

- 购买防火墙。这样可以保护你防范各种软件中至少50%的各种类型的攻击。把MySQL放到防火墙后或隔离区(DMZ)。

检查清单：

o 试试从Internet使用nmap工具扫描端口。MySQL默认使用端口3306。不应从不可信任主机访问该端口。另一种检查是否MySQL端口打开的简单方式是从远程机器试试下面的命令，其中server_host是MySQL服务器运行的主机：

o
```
shell> telnet server_host 3306
```

如果得到连接并得到一些垃圾字符，则端口打开着，则应从防火墙或路由器上关闭，除非你有合理的理由让它开着。如果telnet挂起或连接被拒绝，则端口被阻塞，这是你所希望的。

不要信任应用程序的用户输入的任何数据。它们可以用Web形式、URL或构建的应用程序输入特殊或逃溢字符序列来尝试欺骗你的代码。如果某个用户输入“; DROP DATABASE mysql;”等内容，应确保你的应用程序保持安全。这是特例，但当黑客使用类似技术时，如果你没有做好准备，结果可能会出现大的安全漏洞和数据丢失。

一个常见的错误是只保护字符串数据值。一定要记住还应检查数字数据。如果当用户输入值234时，应用程序生成查询SELECT * FROM table WHERE ID=234，用户可以输入值234 OR 1=1使应用程序生成查询SELECT * FROM table WHERE ID=234 OR 1=1。结果是服务器查找表内的每个记录。这样会暴露每个记录并造成过多的服务器负载。保护防范这类攻击的最简单的方法是使用单引号将数字常量引起来：SELECT * FROM table WHERE ID='234'。如果用户输入其它信息，均变为字符串的一部分。在数字部分，MySQL自动将字符串转换为数字并剥离字符串包含的附加的非数字字符。

有时候人们会认为如果数据库只包含供公共使用的数据，则不需要保护。这是不正确的。即使允许显示数据库中的任何记录，你仍然应保护防范拒绝服务攻击(例如，基于前面段落中所述的技术的攻击，会使服务器浪费资源)。否则，你的服务器不再响应合法用户。

检查清单：

o 试试用Web形式输入单引号和双引号(‘’和‘”’)。如果得到任何形式的MySQL错误，立即分析原因。

o 试试修改动态URL，可以在其中添加%22(‘”’)、%23(‘#’)和%27(‘’)。

o 试试在动态URL中修改数据类型，使用前面示例中的字符，包括数字和字符类型。你的应用程序应足够安全，可以防范此类修改和类似攻击。

o 试试输入字符、空格和特殊符号，不要输入数值字段的数字。你的应用程序应在将它们传递到MySQL之前将它们删除或生成错误。将未经过检查的值传递给MySQL是很危险的！

o 将数据传给MySQL之前先检查其大小。

o 用管理账户之外的用户名将应用程序连接到数据库。不要给应用程序任何不需要的访问权限。

- 许多应用程序编程接口提供了措施逃逸数据值中的特殊字符。如果使用正确，可以防止应用程序用户输入使应用程序生成不期望的效果的语句的数

值：

o MySQL C API：使用mysql_real_escape_string() API调用。

o MySQL++：查询流使用escape和quote修订符。

o PHP：使用mysql_escape_string()函数基于MySQL C API中的同名函数。(在PHP 4.0.3之前,使用addslashes()）。在PHP 5中,可以使用mysqli扩展名，它支持改进的MySQL鉴定协议和密码，以及用占位符编写的语句。

o Perl DBI：使用quote()方法或使用占位符。

o Java JDBC：使用一个PreparedStatement对象和占位符。

其它编程接口有类似的功能。

· 不要通过Internet传送明文(未加密的)数据。该信息可以被有足够时间和能力来截取它并用于个人目的的任何人访问。相反，应使用加密协议，例如SSL或SSH。MySQL支持内部SSL连接，例如版本 4.0.0。可以使用SSH端口映射为通信创建加密(并压缩)的隧道。

· 学会使用tcpdump和strings工具。在大多数情况下，你可以使用下面的命令检查是否MySQL数据流未加密：

```
shell> tcpdump -l -i eth0 -w - src or dst port 3306 | strings
```

(该命令在Linux中可以工作，在其它系统中经过小小的修改后应可以工作）。 警告：如果你没有看见明文数据，并不一定说明信息实际上被加密了。如果你需要较高级别的安全，你应咨询安全专家。

5.6.2. 使MySQL在攻击者面前保持安全

当你连接到MySQL服务器时，你应使用一个密码。密码不以明文在上传输。客户端连接序列中的密码处理在MySQL 4.1.1中已经升级，很安全。如果你仍然使用pre-4.1.1-风格的密码，加密算法不如新算法强；通过一些工作，可以窃取客户端和服务器之间的通信的聪明的攻击者可以破解密码。(关于不同的密码处理方法的讨论参见5.7.9节，“MySQL 4.1中的密码哈希处理”）。 如果客户端和服务器之间的连接通过不可信任网络，你应使用SSH隧道来加密通信。

所有其它信息以文本传送，可以被可以看到连接的任何人读取。如果你担心这个，你可以使用压缩协议来使通信更难以解密。要想使连接更加安全，你应使用SSH来获得加密的MySQL服务器和MySQL客户端之间的TCP/IP连接。你可以从http://www.openssh.org/找到开放源码SSH 客户端，并可以从http://www.ssh.com/获得商业SSH客户端。

你还可以使用MySQL内部OpenSSL支持。参见5.8.7节，“使用安全连接”。

为了使MySQL系统安全，强烈要求你考虑下列建议：

· 对所有MySQL用户使用密码。客户端程序不需要知道运行它的人员的身份。对于客户端/服务器应用程序，用户可以指定客户端程序的用户名。例如，如果*other_user*没有密码，任何人可以简单地用mysql -u *other_user db_name*冒充他人调用**mysql**程序进行连接。如果所有用户有密码，使用其它用户的账户进行连接要困难得多。

要想更改用户的密码，应使用SET PASSWORD语句。还可以直接更新mysql数据库中的user表。例如，要更改所有root用户的MySQL账户的密码，应：

```
shell> mysql -u root
mysql> UPDATE mysql.user SET Password=PASSWORD('newpwd')
    -> WHERE User='root';
mysql> FLUSH PRIVILEGES;
```

· 绝对不要作为Unix的root用户运行MySQL服务器。这样做非常危险，因为任何具有FILE权限的用户能够用root创建文件(例如，~root/.bashrc)。为了防止，**mysqld**拒绝用root运行，除非使用--user=root选项明显指定。

应可以(并且应该)用普通非特权用户运行**mysqld**。你可以创建独立的Unix中的mysql账户来以便使所有内容更加安全。该账户只用于管理MySQL。要想用其它Unix用户启动**mysqld**，增加user选项指定/etc/my.cnf选项文件或服务器数据目录的my.cnf选项文件中的[mysqld]组的用户名。例如：

```
[mysqld]
user=mysql
```

该命令使服务器用指定的用户来启动，无论你手动启动或通过**mysqld_safe**或**mysql.server**启动。详细信息参见A.3.2节，“如何以普通用户身份运行MySQL”。

作为其它Unix用户而不用root运行mysqld，你不需要更改user表中的root用户名，因为MySQL账户的用户名与Unix账户的用户名无关。

· 不要允许使用表的符号链接。(可以用--skip-symbolic-links选项禁用）。如果你用root运行**mysqld**则特别重要，因为任何对服务器的数据目录有写访问权限的人则能够删除系统中的任何文件！参见7.6.1.2节，“在Unix平台上使用表的符号链接”。

· 确保**mysqld**运行时，只使用对数据库目录具有读或写权限的Unix用户来运行。

· 不要将PROCESS或SUPER权限授给非管理用户。**mysqladmin processlist**的输出显示出当前执行的查询正文，如果另外的用户发出一个UPDATE user SET password=PASSWORD('not_secure')查询，被允许执行那个命令的任何用户可能看得到。

mysqld为有SUPER权限的用户专门保留一个额外的连接，因此即使所有普通连接被占用，MySQL root用户仍可以登录并检查服务器的活动。

可以使用SUPER权限来终止客户端连接，通过更改系统变量的值更改服务的器操作，并控制复制服务器。

· 不要向非管理用户授予FILE权限。有这权限的任何用户能在拥有**mysqld**守护进程权限的文件系统那里写一个文件！为了更加安全，由SELECT ...

INTO OUTFILE生成的所有文件对每个人是可写的，并且你不能覆盖已经存在的文件。

file权限也可以被用来读取任何作为运行服务器的Unix用户可读取或访问的文件。使用该权限，你可以将任何文件读入数据库表。这可能被滥用，例如，通过使用LOAD DATA装载"/etc/passwd"进一个数据库表，然后能用SELECT显示它。

· 如果你不信任你的DNS，你应该在授权表中使用IP数字而不是主机名。在任何情况下，你应该非常小心地使用包含通配符的主机名来创建 授权表条目！

· 如果你想要限制单个账户允许的连接数量，你可以设置**mysqld**中的max_user_connections变量来完成。GRANT语句也可以支持 资源控制选项来限制服务器对一个账户允许的使用范围。参见13.5.1.3节，"GRANT和REVOKE语法"。

5.6.3. Mysqld安全相关启动选项

下列**mysqld**选项影响安全：

· --allow-suspicious-udfs

该选项控制是否可以载入主函数只有xxx符的用户定义函数。默认情况下，该选项被关闭，并且只能载入至少有辅助符的UDF。这样可以防止从未包含合法UDF的共享对象文件载入函数。参见27.2.3.6节，"用户定义函数安全注意事项"。

· --local-infile[={0|1}]

如果用--local-infile=0启动服务器，则客户端不能使用LOCAL in LOAD DATA语句。参见5.6.4节，"LOAD DATA LOCAL安全问题"。

· --old-passwords

强制服务器为新密码生成短(pre-4.1)密码哈希。当服务器必须支持旧版本客户端程序时，为了保证兼容性这很有用。参见5.7.9节，"MySQL 4.1中的密码哈希处理"。

· (*OBSOLETE*) --safe-show-database

在以前版本的MySQL中，该选项使SHOW DATABASES语句只显示用户具有部分权限的数据库名。在MySQL 5.1中，该选项不再作为现在的 默认行为使用，有一个SHOW DATABASES权限可以用来控制每个账户对数据库名的访问。参见13.5.1.3节，"GRANT和REVOKE语法"。

· --safe-user-create

如果启用，用户不能用GRANT语句创建新用户，除非用户有mysql.user表的INSERT权限。如果你想让用户具有授权权限来创建新用户，你应给用户授予下面的权限：

```
mysql> GRANT INSERT(user) ON mysql.user TO 'user_name'@'host_name';
```

这样确保用户不能直接更改权限列，必须使用GRANT语句给其它用户授予该权限。

· --secure-auth

不允许鉴定有旧(pre-4.1)密码的账户。

· --skip-grant-tables

这个选项导致服务器根本不使用权限系统。这给每个人以完全访问所有的数据库的权力！（通过执行**mysqladmin flush-privileges**或**mysqladmin reload**命令，或执行FLUSH PRIVILEGES语句，你能告诉一个正在运行的服务器再次开始使用授权表。）

· --skip-name-resolve

主机名不被解析。所有在授权表的Host的列值必须是IP号或localhost。

· --skip-networking

在网络上不允许TCP/IP连接。所有到mysqld的连接必须经由Unix套接字进行。

· --skip-show-database

使用该选项，只允许有SHOW DATABASES权限的用户执行SHOW DATABASES语句，该语句显示所有数据库名。不使用该选项，允许所有用户执行SHOW DATABASES，但只显示用户有SHOW DATABASES权限或部分数据库权限的数据库名。请注意全局权限指数据库的权限。

5.6.4. LOAD DATA LOCAL安全问题

LOAD DATA语句可以装载服务器主机上的文件，若指定LOCAL关键字，可以装载客户端文件。

支持LOCAL版本的LOAD DATA语句有两个可能的安全问题：

· 由MySQL服务器启动文件从客户端向服务器主机的传输。理论上，打过补丁的服务器可以告诉客户端程序传输服务器选择的文件，而不是客户用LOAD DATA语句指定的文件。这样服务器可以访问客户端上客户有读访问权限的任何文件。

· 在Web环境中，客户从Web服务器连接，用户可以使用LOAD DATA LOCAL来读取Web服务器进程有读访问权限的任何文件(假定用户可以运行SQL服务器的任何命令)。在这种环境中，MySQL服务器的客户实际上是Web服务器，而不是连接Web服务器的用户运行的程序。

要处理这些问题，我们更改了MySQL 3.23.49和MySQL 4.0.2(Windows中的4.0.13)中的LOAD DATA LOCAL的处理方法：

- 默认情况下，现在所有二进制分中的发MySQL客户端和库是用--enable-local-infile选项编译，以便与MySQL 3.23.48和以前的版本兼容。
- 如果你从源码构建MySQL但没有使用--enable-local-infile选项来进行**configure**，则客户不能使用LOAD DATA LOCAL，除非显式调用mysql_options(...MYSQL_OPT_本地_INFILE，0)。参见25.2.3.48节，“mysql_options()”。
- 你可以用--local-infile=0选项启动**mysqld**从服务器端禁用所有LOAD DATA LOCAL命令。
- 对于**mysql**命令行客户端，可以通过指定--local-infile[=1]选项启用LOAD DATA LOCAL，或通过--local-infile=0选项禁用。类似地，对于**mysqlimport**，--local or -L选项启用本地数据文件装载。在任何情况下，成功进行本地装载需要服务器启用相关选项。
- 如果你使用LOAD DATA LOCAL Perl脚本或其它读选项文件中的[client]组的程序，你可以在组内添加local-infile=1选项。但是，为了便面不理解local-infile的程序产生问题，则规定使用loose- prefix：

```
·               [client]
·               loose-local-infile=1
```

- 如果LOAD DATA LOCAL INFILE在服务器或客户端被禁用，试图执行该语句的客户端将收到下面的错误消息：

```
ERROR 1148: The used command is not allowed with this MySQL version
```

5.7. MySQL访问权限系统

5.7.1. 权限系统的作用
5.7.2. 权限系统工作原理
5.7.3. MySQL提供的权限
5.7.4. 与MySQL服务器连接
5.7.5. 访问控制, 阶段1：连接核实
5.7.6. 访问控制, 阶段2：请求核实
5.7.7. 权限更改何时生效
5.7.8. 拒绝访问错误的原因
5.7.9. MySQL 4.1中的密码哈希处理

MySQL有先进但非标准的安全/权限系统。本节描述它的工作原理。

5.7.1. 权限系统的作用

MySQL权限系统的主要功能是证实连接到一台给定主机的用户，并且赋予该用户在数据库上的SELECT、INSERT、UPDATE和DELETE权限。

附加的功能包括有匿名的用户并对于**MySQL**特定的功能例如LOAD DATA INFILE进行授权及管理操作的能力。

5.7.2. 权限系统工作原理

MySQL权限系统保证所有的用户只执行允许做的事情。当你连接**MySQL**服务器时，你的身份由你从那儿连接的主机和你指定的用户名来决定。连接后发出请求后，系统根据你的身份和你想做什么来授予权限。

MySQL在认定身份中考虑你的主机名和用户名字，是因为几乎没有原因假定一个给定的用户在因特网上属于同一个人。例如，从office.com连接的用户joe不一定和从elsewhere.com连接的joe是同一个人。**MySQL**通过允许你区分在不同的主机上碰巧有同样名字的用户来处理它：你可以对joe从office.com进行的连接授与一个权限集，而为joe从elsewhere.com的连接授予一个不同的权限集。

MySQL存取控制包含2个阶段：

- 阶段1：服务器检查是否允许你连接。
- 阶段2：假定你能连接，服务器检查你发出的每个请求。看你是否有足够的权限实施它。例如，如果你从数据库表中选择(select)行或从数据库删除表，服务器确定你对表有SELECT权限或对数据库有DROP权限。

如果连接时你的权限被更改了(通过你和其它人)，这些更改不一定立即对你发出的下一个语句生效。详情参见5.7.7节，“权限更改何时生效”。

服务器在mysql数据库的 授权表中保存权限信息(即在mysql数据库中)。当MySQL服务器启动时将这些表的内容读入内存，在5.7.7节，“权限更改何时生效”的环境下重新读取它们。访问控制决策取决于内存中的 授权表的份数。

一般情况，你通过GRANT和REVOKE语句间接来操作 授权表的内容，设置账户并控制个人的权限。参见13.5.1.3节，“GRANT和REVOKE语法”。下面讨论了授权表的结构以及服务器与客户端交互操作时如何使用其内容。

服务器在存取控制的两个阶段使用mysql数据库中的user、db和host表，这些授权表中的列如下：

表名	**user**	**db**	**host**
列范围	Host	Host	Host
	User	Db	Db
	Password	User	
权限列	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
	Create_view_priv	Create_view_priv	Create_view_priv
	Show_view_priv	Show_view_priv	Show_view_priv
	Create_routine_priv	Create_routine_priv	
	Alter_routine_priv	Alter_routine_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		
安全列	ssl_type		
	ssl_cipher		
	x509_issuer		
	x509_subject		
资源控制列	max_questions		
	max_updates		
	max_connections		
	max_user_connections		

对存取控制的第二阶段(请求证实)，服务器执行请求验证以确保每个客户端有充分的权限满足各需求。除了user、db和host授权表，如果请求涉及表，服务器可以另外参考tables_priv和columns_priv表。tables_priv和columns_priv表可以对表和列提供更精确的权限控制。这些表的列如下:

表名	**tables_priv**	**columns_priv**
范围列	Host	Host
	Db	Db
	User	User
	Table_name	Table_name
		Column_name
权限列	Table_priv	Column_priv
	Column_priv	
其它列	Timestamp	Timestamp
	Grantor	

Timestamp和Grantor列当前还未使用，这儿不再进一步讨论。

为了对涉及保存程序的请求进行验证，服务器将查阅procs_priv表。该表具有以下列:

表名	**procs_priv**
范围列	Host
	Db
	User
	Routine_name
	Routine_type
权限列	Proc_priv
其它列	Timestamp
	Grantor

Routine_type列为ENUM列，值为'FUNCTION'或'PROCEDURE'，表示行所指的程序类型。该列允许为同名函数和程序单独授权。

Timestamp和Grantor列当前还未使用，这儿不再进一步讨论。

每个授权表包含范围列和权限列：

- 范围列决定表中每个条目（行）的范围，即，行适用的上下文。例如，一个user表行的Host和User值为'thomas.loc.gov'和'bob'，将被用于证实来自主机thomas.loc.gov的bob对服务器的连接。同样，一个db表行的Host、User和Db列的值是'thomas.loc.gov'、'bob'和'reports'将用在bob从主机thomas.loc.gov联接访问reports数据库的时候。tables_priv和columns_priv表包含范围列，指出每个行适用的表或表/列的组合。procs_priv范围列指出每个行适用的保存程序。

  对于检查存取的用途，比较Host值是忽略大小写的。User、Password、Db和Table_name值是区分大小写的。Column_name值在**MySQL**3.22.12或以后版本是忽略大小写的。

- 权限列指出由一个表行授予的权限，即，可实施什么操作。服务器组合各种的授权表的信息形成一个用户权限的完整描述。为此使用的规则在5.7.6节，“访问控制，阶段2：请求核实”描述。

范围列包含字符串，如下所述；每个列的默认值是空字符串：

列名	类型
Host	CHAR(60)
User	CHAR(16)
Password	CHAR(16)
Db	CHAR(64)
Table_name	CHAR(64)
Column_name	CHAR(64)
Routine_name	CHAR(64)

为了访问检查目的，Host值的比较对大小写不敏感。User、Password、Db和Table_name值对大小写敏感。Column_name值对大小写不敏感。

在user、db和host表中，所有权限列于单独的列内，被声明为ENUM('N','Y') DEFAULT 'N'。换句话说，每一个权限都可以被禁用和启用，并且 默认是禁用。

在tables_priv、columns_priv和procs_priv表中，权限列被声明为SET列。这些列的值可以包含该表控制的权限的组合：

表名	列名	可能的设置元素
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'
procs_priv	Proc_priv	'Execute', 'Alter Routine', 'Grant'

简单地说，服务器使用这样的授权表：

- user表范围列决定是否允许或拒绝到来的连接。对于允许的连接，user表授予的权限指出用户的全局(超级用户)权限。这些权限适用于服务器上的*all*数据库。

- db表范围列决定用户能从哪个主机存取哪个数据库。权限列决定允许哪个操作。授予的数据库级别的权限适用于数据库和它的表。

- 当你想要一个给定的db表行应用于若干主机时，db和host表一起使用。例如，如果你想要一个用户能在你的网络从若干主机使用一个数据库，在用户的db表行的Host值设为空值，然后将那些主机的每一个移入host表。这个机制详细描述在5.7.6节，“访问控制，阶段2：请求核实”。

注释：host表不受GRANT和REVOKE语句的影响。大多数MySQL安装根本不需要使用该表。

- tables_priv和columns_priv表类似于db表，但是更精致：它们在表和列级应用而非在数据库级。授予表级别的权限适用于表和所有它的列。授予列级别的权限只适用于专用列。

- procs_priv表适用于保存的程序。授予程序级别的权限只适用于单个程序。

管理权限(例如RELOAD或SHUTDOWN等等)仅在user表中被指定。这是因为管理性操作是服务器本身的操作并且不是特定数据库，因此没有理由在其他授权表中列出这样的权限。事实上，只需要查询user表来决定你是否执行一个管理操作。

FILE权限也仅在user表中指定。它不是管理性权限，但你在服务器主机上读或写文件的能力与你正在存取的数据库无关。

当**mysqld**服务器启动时，将授权表的内容读入到内存中。你可以通过FLUSH PRIVILEGES语句或执行**mysqladmin flush-privileges**或**mysqladmin reload**命令让它重新读取表。对授权表的更改生效在5.7.7节，“权限更改何时生效”描述。

当你修改授权表的内容时，确保你按你想要的方式更改权限设置是一个好主意。要检查给定账户的权限，使用SHOW GRANTS语句。例如，要检查Host和User值分别为pc84.example.com和bob的账户所授予的权限，应通过语句：

```
mysql> SHOW GRANTS FOR 'bob'@'pc84.example.com';
```

一个有用的诊断工具是**mysqlaccess**脚本，由Carlier Yves 提供给**MySQL**分发。使用--help选项调用**mysqlaccess**查明它怎样工作。注意：**mysqlaccess**仅用user、db和host表检查存取。它不检查tables_priv、columns_priv或procs_priv表中指定的表、列和程序级权限。

对于诊断权限相关的问题的其它帮助，参见5.7.8节，“拒绝访问错误的原因”。对于安全问题常规建议，参见5.6节，“一般安全问题”。

5.7.3. MySQL提供的权限

账户权限信息被存储在mysql数据库的user、db、host、tables_priv、columns_priv和procs_priv表中。在**MySQL**启动时并在5.7.7节，“权限更改何时生效”所说的情况时，服务器将这些数据库表内容读入内存。

GRANT和REVOKE语句所用的涉及权限的名称显示在下表，还有在授权表中每个权限的表列名称和每个权限有关的上下文。关于每个权限的含义相关的详细信息参见13.5.1.3节，“GRANT和REVOKE语法”。

权限	列	上下文
CREATE	Create_priv	数据库、表或索引
DROP	Drop_priv	数据库或表
GRANT OPTION	Grant_priv	数据库、表或保存的程序
REFERENCES	References_priv	数据库或表
ALTER	Alter_priv	表
DELETE	Delete_priv	表
INDEX	Index_priv	表
INSERT	Insert_priv	表
SELECT	Select_priv	表
UPDATE	Update_priv	表
CREATE VIEW	Create_view_priv	视图
SHOW VIEW	Show_view_priv	视图
ALTER ROUTINE	Alter_routine_priv	保存的程序
CREATE ROUTINE	Create_routine_priv	保存的程序
EXECUTE	Execute_priv	保存的程序
FILE	File_priv	服务器主机上的文件访问
CREATE TEMPORARY TABLES	Create_tmp_table_priv	服务器管理
LOCK TABLES	Lock_tables_priv	服务器管理
CREATE USER	Create_user_priv	服务器管理
PROCESS	Process_priv	服务器管理
RELOAD	Reload_priv	服务器管理
REPLICATION CLIENT	Repl_client_priv	服务器管理
REPLICATION SLAVE	Repl_slave_priv	服务器管理
SHOW DATABASES	Show_db_priv	服务器管理
SHUTDOWN	Shutdown_priv	服务器管理
SUPER	Super_priv	服务器管理

当从早期的没有CREATE VIEW、SHOW VIEW、CREATE ROUTINE、ALTER ROUTINE和EXECUTE权限的版本的MySQL中升级时，要想使用这些权限，你必须使用MySQL分发提供的**mysql_fix_privilege_tables**脚本升级 授权表。参见2.10.2节，“升级授权表”。

如果启用了二进制记录，要想创建或修改保存的程序，你还需要SUPER权限，详细描述见20.4节，“存储子程序和触发程序的二进制日志功能”。

通过CREATE和DROP权限，你可以创建新数据库和表，或删除(移掉)已有数据库和表。如果你将*mysql*数据库中的*DROP*权限授予某用户，用户可以删掉*MySQL*访问权限保存的数据库。

SELECT、INSERT、UPDATE和DELETE权限允许你在一个数据库现有的表上实施操作。

SELECT语句只有在他们真正从一个表中检索行时才需要SELECT权限。一些SELECT语句不访问表，甚至没有任何到服务器上的数据库里的存取任何东西的许可。例如，你可使用**mysql**客户端作为一个简单的计算器来评估未引用表的表达式：

```
mysql> SELECT 1+1;

mysql> SELECT PI()*2;
```

INDEX权限允许你创建或删除索引。INDEX适用已有表。如果你具有某个表的CREATE权限，你可以在CREATE TABLE语句中包括索引定义。

通过ALTER权限，你可以使用ALTER TABLE来更改表的结构和重新命名表。

需要CREATE ROUTINE权限来创建保存的程序（函数和程序），ALTER ROUTINE权限来更改和删除保存的程序，EXECUTE来执行保存的程序。

GRANT权限允许你把你自己拥有的那些权限授给其他的用户。可以用于数据库、表和保存的程序。

FILE权限给予你用LOAD DATA INFILE和SELECT ... INTO OUTFILE语句读和写服务器上的文件，任何被授予FILE权限的用户都能读或写MySQL服务器能读或写的任何文件。(说明用户可以读任何数据库目录下的文件，因为服务器可以访问这些文件）。 FILE权限允许用户在MySQL服务器具有写权限的目录下创建新文件。不能覆盖已有文件。

其余的权限用于管理性操作，它使用**mysqladmin**程序或SQL语句实施。下表显示每个管理性权限允许你执行的**mysqladmin**命令：

权限	权限拥有者允许执行的命令
RELOAD	flush-hosts, flush-logs, flush-privileges, flush-status, flush-tables, flush-threads, refresh, reload
SHUTDOWN	shutdown
PROCESS	processlist
SUPER	kill

reload命令告诉服务器将授权表重新读入内存。flush-privileges是reload的同义词，refresh命令清空所有表并打开并关闭记录文件，其它flush-*xxx*命令执行类似refresh的功能，但是范围更有限，并且在某些情况下可能更好用。例如，如果你只是想清空记录文件，flush-logs比refresh是更好的选择。

shutdown命令关掉服务器。只能从**mysqladmin**发出命令。没有相应的SQL语句。

processlist命令显示在服务器内执行的线程的信息（即其它账户相关的客户端执行的语句）。kill命令杀死服务器线程。你总是能显示或杀死你自己的线程，但是你需要PROCESS权限来显示或杀死其他用户和SUPER权限启动的线程。参见13.5.5.3节，“KILL语法”。

拥有CREATE TEMPORARY TABLES权限便可以使用CREATE TABLE语句中的关键字TEMPORARY。

拥有LOCK TABLES权限便可以直接使用LOCK TABLES语句来锁定你拥有SELECT权限的表。包括使用写锁定，可以防止他人读锁定的表。

拥有REPLICATION CLIENT权限便可以使用SHOW MASTER STATUS和SHOW SLAVE STATUS。

REPLICATION SLAVE权限应授予从服务器所使用的将当前服务器连接为主服务器的账户。没有这个权限，从服务器不能发出对主服务器上的数据库所发出的更新请求。

拥有SHOW DATABASES权限便允许账户使用SHOW DATABASE语句来查看数据库名。没有该权限的账户只能看到他们具有部分权限的数据库， 如果数据库用--skip-show-database选项启动，则根本不能使用这些语句。请注意全局权限指数据库的权限。

总的说来，只授予权限给需要他们的那些用户是好主意，但是你应该在授予FILE和管理权限时试验特定的警告：

- FILE权限可以被滥用于将服务器主机上MySQL能读取的任何文件读入到数据库表中。包括任何人可读的文件和服务器数据目录中的文件。可以使用SELECT访问数据库表，然后将其内容传输到客户端上。
- GRANT权限允许用户将他们的权限给其他用户。有不同的权限并有GRANT权限的2个用户可以合并权限。
- ALTER权限可以用于通过重新命名表来推翻权限系统。
- SHUTDOWN权限通过终止服务器可以被滥用完全拒绝为其他用户服务。
- PROCESS权限能被用来察看当前执行的查询的明文文本，包括设定或改变密码的查询。
- SUPER权限能用来终止其它用户或更改服务器的操作方式。
- 授给mysql数据库本身的权限能用来改变密码和其他访问权限信息。密码被加密存储，所以恶意的用户不能简单地读取他们以知道明文密码。然而，具有user表Password列写访问权限的用户可以更改账户的密码，并可以用该账户连接MySQL服务器。

有一些事情你不能用**MySQL**权限系统做到：

- 你不能明显地指定某个给定的用户应该被拒绝访问。即，你不能明显地匹配用户然后拒绝连接。
- 你不能指定用户有权创建立或删除数据库中的表，但不能创建或删除数据库本身。

5.7.4. 与MySQL服务器连接

当你想要访问MySQL服务器时，MySQL客户端程序一般要求你指定参数：

- MySQL服务器运行的主机名
- 姓名
- 密码

例如，可以从命令行按照下述提示启动**MySQL**客户端(用shell>表示)：

```
shell> MySQL -h host_name -u user_name -pyour_pass
```

-h, -u和-p选项还可以采用形式--host=*host_name*、--user=*user_name*和--password=*your_pass*。请注意在-p或--password=和后面的密码之间没有空格。

如果你使用-p或--password选项但没有指定密码值，客户端程序提示你输入密码。当你输入密码时并不显示密码。这比在在命令行输入密码要安全得多。系统上的任何用户可以通过命令**ps auxww**在命令行中指定密码。参见5.8.6节，“使你的密码安全”。

如果没有指定连接参数，MySQL客户端程序使用默认值：

- 默认主机名是localhost。
- 默认用户名在Windows中是ODBC，在Unix中是你的Unix登录名。
- 如果没有-p，则不提供密码。

这样, 对Unix用户joe，下列命令是等价的：

```
shell> MySQL -h localhost -u joe
shell> MySQL -h localhost
shell> MySQL -u joe
shell> MySQL
```

其它**MySQL**客户端程序类似。

当进行连接时，你可以指定要使用的不同的默认值，这样不必每次在你调用客户端程序是在命令行上输入它们。这可以有很多方法做到：

- 你可以在选项文件的[client]小节里指定连接参数。文件的相关小节看上去可能像这样：

```
[client]
```

- host=*host_name*
- user=*user_name*
- password=*your_pass*

在4.3.2节，“使用选项文件”中详细讨论了选项文件。

- 你可以用环境变量指定连接参数。主机可用MYSQL_HOST指定，**MySQL**用户名可用USER指定(仅对Windows和NetWare)，密码可用MYSQL_PWD指定，尽管这不安全；参见5.8.6节，“使你的密码安全”。变量参见附录F：环境变量。

5.7.5. 访问控制, 阶段1：连接核实

当你试图连接**MySQL**服务器时，服务器基于你的身份以及你是否能通过供应正确的密码验证身份来接受或拒绝连接。如果不是，服务器完全拒绝你的访问，否则，服务器接受连接，然后进入阶段2并且等待请求。

你的身份基于2个信息：

- 你从那个主机连接
- 你的**MySQL**用户名

身份检查使用3个user表(Host, User和Password)范围列执行。服务器只有在user表记录的Host和User列匹配客户端主机名和用户名并且提供了正确的密码时才接受连接。

在user表Host值的指定方法：

- Host值可以是主机名或IP号，或'localhost'指出本地主机。
- 你可以在Host列值使用通配符字符“%”和“_”。
- Host值'%'匹配任何主机名，空Host值等价于'%'。它们的含义与LIKE操作符的模式匹配操作相同。例如，'%'的Host值与所有主机名匹配，而'%.mysql.com'匹配mysql.com域的所有主机。

- 对于指定为IP号的Host值，你可以指定一个网络掩码，说明使用多少位地址位来评比网络号。例如：

```
mysql> GRANT ALL PRIVILEGES ON db.*
    -> -> TO david@'192.58.197.0/255.255.255.0';
```

允许david从任何客户端用IP号client_ip来连接，下面的条件为真：

```
client_ip & netmask = host_ip
```

That is, for the GRANT statement just shown:

```
client_ip & 255.255.255.0 = 192.58.197.0
```

满足该条件并可以连接MySQL服务器的IP号的范围为192.58.197.0到192.58.197.255。

- 注释：网络掩码只用来告诉服务器使用8、16、24或32位地址，例如：

```
192.0.0.0/255.0.0.0(192 A类网络的任何地址)
192.168.0.0/255.255.0.0(192.168 A类网络的任何地址)
192.168.1.0/255.255.255.0(192.168.1 C类网络的任何地址)
192.168.1.1(只有该IP)
```

下面的网络掩码(28 位)无效：

```
192.168.0.1/255.255.255.240
```

- db表记录中的空Host值表示它的权限应结合匹配客户端名的host表中的行使用。通过AND(相与)而不是或(联合)操作将权限组合到一起。你可以从5.7.6节，“访问控制, 阶段2：请求核实”找到关于host表的详细信息。

其它grant表的空Host值与'%'相同。

既然你能在Host字段使用IP通配符值(例如，'144.155.166.%'匹配在一个子网上的每台主机)，有可能某人可能企图探究这种能力，通过命名一台主机为144.155.166.somewhere.com。为了阻止这样的企图，MySQL不允许匹配以数字和一个点起始的主机名，这样，如果你用一个命名为类似1.2.foo.com的主机，它的名字决不会匹配授权表中的Host列。只有一个IP数字能匹配IP通配符值。

通配符字符在User列中不允许，但是你能指定空的值，它匹配任何名字。如果user表匹配的连接有一个空用户名，用户被认为是匿名用户(没有名字的用户)，而非客户端实际指定的名字。这意味着一个空的用户名被用于在连接期间的进一步的访问检查(即，在阶段2期间)。

Password列可以是空的。这不是通配符，也不意味着匹配任何密码，它意味着用户必须不指定一个密码进行连接。

user表中的非空Password值代表加密的密码。**MySQL**不以任何人可以看的明文文本格式存储密码，相反，正在试图联接的用户提供的密码被加密(使用PASSWORD( )函数)，在连接过程中使用加密的密码检查密码是否正确。(加密后的密码未通过连接即可实现）。从MySQL角度，加密的密码是实际密码，因此你不应让其它人访问它！特别是，绝对不要让非管理用户读mysql数据库中的表！

MySQL 5.1使用强鉴定方法(最先在MySQL 4.1中适用)在前面的版本中在连接进程中的密码保护较好。即使TCP/IP包被截取或mysql数据库 被捕获也很安全。5.7.9节，“MySQL 4.1中的密码哈希处理”中详细讨论了密码加密。

下面的例子显示出各种user表中Host和User值的组合如何应用于到来的连接：

Host值	User值	被条目匹配的连接
'thomas.loc.gov'	'fred'	fred, 从thomas.loc.gov 连接
'thomas.loc.gov'	''	任何用户, 从thomas.loc.gov连接
'%'	'fred'	fred, 从任何主机连接
'%'	''	任何用户, 从任何主机连接
'%.loc.gov'	'fred'	fred, 从在loc.gov域的任何主机连接
'x.y.%'	'fred'	fred, 从x.y.net、x.y.com,x.y.edu等联接。（这或许无用）
'144.155.166.177'	'fred'	fred, 从有144.155.166.177 IP地址的主机连接
'144.155.166.%'	'fred'	fred, 从144.155.166 C类子网的任何主机连接

到来的连接中的客户端名和用户名可能与user表中的多行匹配。例如，由fred从thomas.loc.gov的连接匹配多个条目如上所述。

如果有多个匹配，服务器必须选择使用哪个条目。按照下述方法解决问题：

- 服务器在启动时读入user表后进行排序。
- 然后当用户试图连接时，以排序的顺序浏览条目
- 服务器使用与客户端和用户名匹配的第一行。

user表排序工作如下，假定user表看起来像这样：

```
+-----------+----------+-
| Host      | User     | …
+-----------+----------+-
| %         | root     | …
| %         | jeffrey  | …
| localhost | root     | …
| localhost |          | …
+-----------+----------+-
```

当服务器读取表时，它首先以最具体的Host值排序。主机名和IP号是最具体的。'%'意味着"任何主机"并且是最不特定的。有相同Host值的条目首先以最具体的User值排序(空User值意味着"任何用户"并且是最不特定的)。最终排序的user表看起来像这样：

```
+-----------+----------+-
| Host      | User     | …
+-----------+----------+-
| localhost | root     | … ...
| localhost |          | … ...
| %         | jeffrey  | … ...
| %         | root     | … ...
+-----------+----------+-
```

当客户端试图连接时，服务器浏览排序的条目并使用找到的第一匹配。对于由jeffrey从localhost的连接，表内有两个条目匹配：Host和User值为'localhost'和''的条目，和值为'%'和'jeffrey'的条目。'localhost'条目首先匹配，服务器可以使用。

还有一个例子。假定user表看起来像这样：

```
+----------------+----------+-
| Host           | User     | …
+----------------+----------+-
| %              | jeffrey  | …
| thomas.loc.gov |          | …
+----------------+----------+-
```

排序后的表看起来像这样：

```
+----------------+----------+-
| Host           | User     | …
+----------------+----------+-
| thomas.loc.gov |          | …
| %              | jeffrey  | …
+----------------+----------+-
```

由jeffrey从thomas.loc.gov的连接与第一行匹配，而由jeffrey从whitehouse.gov的连接被第二个匹配。

普遍的误解是认为，对给定的用户名，当服务器试图对连接寻找匹配时，明确命名那个用户的所有条目将首先被使用。这明显不符合事实。先前的例子说明了这点，在那里由jeffrey从thomas.loc.gov的连接没被包含'jeffrey'作为User列值的行匹配，但是由没有用户名的题目匹配！结果是，jeffrey被鉴定为匿名用

户，即使他连接时指定了用户名。

如果你能够连接服务器，但你的权限不是你期望的，你可能被鉴定为其它账户。要想找出服务器用来鉴定你的账户，使用CURRENT_USER()函数。它返回*user_name*@*host_name*格式的值，说明User和Host 值匹配user表记录。假定jeffrey连接并发出下面的查询：

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

这儿显示的结果说明user表行有空的User列值。换句话说，服务器将jeffrey视为匿名用户。

诊断鉴定问题的另一个方法是打印出user表并且手动排序它看看第一个匹配在哪儿进行。又见12.9.3节，“信息函数”。

5.7.6. 访问控制, 阶段2：请求核实

一旦你建立了连接，服务器进入访问控制的阶段2。对在此连接上进来的每个请求，服务器检查你想执行什么操作，然后检查是否有足够的权限来执行它。这正是在授权表中的权限列发挥作用的地方。这些权限可以来自user、db、host、tables_priv或columns_priv表。（你会发现参考5.7.2节，“权限系统工作原理”很有帮助，它列出了每个 授权表中呈现的列。）

user表在全局基础上授予赋予你的权限，该权限不管当前的数据库是什么均适用。例如，如果user表授予你DELETE权限， 你可以删除在服务器主机上从任何数据库删除行！换句话说，user表权限是超级用户权限。只把user表的权限授予超级用户如服务器或数据库主管是明智的。对其他用户，你应该把在user表中的权限设成'N'并且仅在特定数据库的基础上授权。你可以为特定的数据库、表或列授权。

db和host表授予数据库特定的权限。在这些表中的范围列的值可以采用以下方式：

- 通配符字符“%”并“_”可用于两个表的Host和Db列。它们与用LIKE操作符执行的模式匹配操作具有相同的含义。如果授权时你想使用某个字符，必须使用反斜现引用。例如，要想在数据库名中包括下划线('_')，在GRANT语句中用'_'来指定。
- 在db表的'%'Host值意味着“任何主机”，在db表中空Host值意味着“对进一步的信息咨询host表”（本节后面将描述的一个过程）。
- 在host表的'%'或空Host值意味着“任何主机”。
- 在两个表中的'%'或空Db值意味着“任何数据库”。
- 在两个表中的空User值匹配匿名用户。

db和host表在服务器启动时被读取并排序(同时它读user表)。db表在Host、Db和User范围列上排序，并且host表在Host和Db范围列上排序。对于user表，首先根据最具体的值最后根据最不具体的值排序，并且当服务器寻找匹配条目时，它使用它找到的第一匹配。

tables_priv和columns_priv表授予表和列特定的权限。这些表的范围列的值可以如下被指定：

- 通配符“%”并“_”可用在使用在两个表的Host列。
- 在两个表中的'%'或空Host意味着“任何主机”。
- 在两个表中的Db、Table_name和Column_name列不能包含通配符或空。

tables_priv和columns_priv表根据Host、Db和User列被排序。这类似于db表的排序，因为只有Host列可以包含通配符，排序更简单。

请求证实进程在下面描述。（如果你熟悉访问检查的源码，你会注意到这里的描述与在代码使用的算法略有不同。描述等价于代码实际做的东西；不同处只是使解释更简单。）

对需要管理权限的请求(SHUTDOWN、RELOAD等等)，服务器仅检查user表条目，因为那是唯一指定管理权限的表。如果行许可请求的操作，访问被授权，否则拒绝。例如，如果你想要执行**mysqladmin shutdown**，但是由于user表行没有为你授予HUTDOWN权限，甚至不用检查db或host表就拒绝你的访问。（因为它们不包含hutdown_priv行列，没有这样做的必要。）

对数据库有关的请求(INSERT、UPDATE等等)，服务器首先通过查找user表行来检查用户的全局(超级用户)权限。如果行允许请求的操作，访问被授权。如果在user表中全局权限不够，服务器通过检查db和host表确定特定的用户数据库权限：

1. 服务器在db表的Host、Db和User列上查找匹配。Host和User对应连接用户的主机名和**MySQL**用户名。Db列对应用户想要访问的数据库。如果没有Host和User的行，访问被拒绝。
2. 如果db表中有匹配的行而且它的Host列不是空的，该行定义用户的数据库特定的权限。
3. 如果匹配的db表的行的Host列是空的，它表示host表列举被允许访问数据库的主机。在这种情况下，在host表中作进一步查找以发现Host和Db列上的匹配。如果没有host表行匹配，访问被拒绝。如果有匹配，用户数据库特定的权限以在db和host表的行的权限，即在两个行都是'Y'的权限的交集(而不是并集！)计算。（这样你可以授予在db表行中的一般权限，然后用host表行按主机主机为基础有选择地限制它们。）

在确定了由db和host表行授予的数据库特定的权限后，服务器把他们加到由user表授予的全局权限中。如果结果允许请求的操作，访问被授权。否则，服务器检查在tables_priv和columns_priv表中的用户的表和列权限并把它们加到用户权限中。基于此结果允许或拒绝访问。

用布尔术语表示，前面关于用户权限如何计算的描述可以这样总结：

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
```

它可能不明显，为什么呢，如果全局user行的权限最初发现对请求的操作不够，服务器以后把这些权限加到数据库、表并列的特定权限。原因是请求可能要求超过一种类型的权限。例如，如果你执行INSERT INTO ... SELECT语句，你就需要INSERT和SELECT权限。你的权限必须是user表行授予一个权限而db表

行授予另一个权限。在这种情况下，你有必要的权限执行请求，但是服务器不能自己把两个表区别开来；两个行授予的权限必须组合起来。

host表不受GRANT或REVOKE语句的影响，因此在大多数MySQL安装中没有使用。如果你直接修改它，你可以用于某种专门目的，例如用来维护安全服务器列表。例如，在TcX，host表包含在本地网络上所有的机器的表。这些表被授予所有的权限。

你也可以使用host表指定不安全的主机。假定你有一台机器public.your.domain，它位于你认为不安全的公共区域，你可以用下列的host表条目允许除了那台机器外的网络上所有主机的访问：

```
+--------------------+----+-
| Host               | Db | ...
+--------------------+----+-
| public.your.domain | %  | ... (all privileges set to 'N')
| %.your.domain      | %  | ... (all privileges set to 'Y')
+--------------------+----+-
```

当然，一定要测试授权表中的行(例如，使用SHOW GRANTS或**mysqlaccess**)，确保你的访问权限实际按你期望的方式被设置。

5.7.7. 权限更改何时生效

当mysqld启动时，所有授权表的内容被读进内存并且从此时生效。

当服务器注意到授权表被改变了时，现存的客户端连接有如下影响：

- 表和列权限在客户端的下一次请求时生效。
- 数据库权限改变在下一个USE *db_name*命令生效。
- 全局权限的改变和密码改变在下一次客户端连接时生效。

如果用GRANT、REVOKE或SET PASSWORD对授权表进行修改，服务器会注意到并立即重新将授权表载入内存。

如果你手动地修改授权表(使用INSERT、UPDATE或DELETE等等)，你应该执行**mysqladmin flush-privileges**或**mysqladmin reload**告诉服务器再装载授权表，否则你的更改将不会生效，除非你重启服务器。

如果你直接更改了授权表但忘记重载，重启服务器后你的更改方生效。这样可能让你迷惑为什么你的更改没有什么变化！

5.7.8. 拒绝访问错误的原因

当你试着联接**MySQL**服务器时，如果碰到问题，下面各项可以帮助你纠正问题：

- 确保服务器在运行。如果服务器没有运行，则你不能连接服务器。如果你视图连接服务器并看到下述消息，可能是服务器没有运行：

```
shell> mysql
ERROR 2003: Can't connect to MySQL server on 'host_name' (111)
shell> mysql
ERROR 2002: Can't connect to local MySQL server through socket
'/tmp/mysql.sock' (111)
```

也可能服务器正在运行，但你可能使用与服务器上侦听的不一样的TCP/IP端口、命名管道或Unix套接字文件。你可以调用客户端程序，指定端口选项来指示正确的端口或套接字选项来指示正确的命名管道或Unix套接字文件。要找出套接字文件的地点，应：

```
shell> netstat -ln | grep mysql
```

- 必须正确设置授权表，以便服务器可以使用它们进行访问控制。对于某些分发版类型(例如Windows中的二进制分发版或Linux中的RPM分发版),安装过程初始化包含 授权表的mysql数据库。如果分发版没有这样做，你必须运行**mysql_install_db**脚本来手动初始化授权表。详细内容参见2.9.2节，“Unix下安装后的过程”。

  确定是否要初始化授权表的一个方法是寻找数据目录下的mysql目录（数据目录名通常为data或var，位于MySQL安装目录下）。应保证**MySQL**数据库目录有文件“user.MYD”。否则，执行**mysql_install_db**脚本。运行并重启服务器后，执行该命令来测试初始权限：

  shell> **mysql -u root test**

  服务器应该让你无误地连接。

- 在新的安装以后，你应该连接服务器并且设置你的用户及其访问许可：

```
shell> mysql -u root mysql
```

服务器应该让你连接，因为**MySQL** root用户初始时没有密码。那也是安全风险，当你正在设置其他**MySQL**用户时，也应设定root密码是一件重要的事请。关于设置初始密码的说明，参见2.9.3节，“使初始MySQL账户安全”。

- 如果你将一个现存的MySQL安装升级到较新的版本，运行了**mysql_fix_privilege_tables**脚本吗？如果没有，运行它。增加新功能后，授权表的结构可能会改变，因此更新后应确保表的结构随之更新。相关说明参见2.10.2节，“升级授权表”。

- 如果客户端程序试图连接时收到以下错误信息，说明服务器需要新格式的密码，而客户端不能生成：

```
shell> mysql
```

```
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

关于如何处理的详细信息，参见5.7.9节，“MySQL 4.1中的密码哈希处理”和A.2.3节，“客户端不支持鉴定协议”。

- 如果你作为root试试连接并且得到这个错误，这意味着，你没有行在user表中的User列值为'root'并且mysqld不能为你的客户端解析主机名：
- Access denied for user ''@'unknown' to database mysql

在这种情况下，你必须用--skip-grant-tables选项重启服务器并且编辑“/etc/hosts”或“\windows\hosts”文件为你的主机增加行。

- 如果你从3.22.11以前的版本更新现存的**MySQL**安装到3.22.11版或以后版本，你运行了mysql_fix_privilege_tables脚本吗？如果没有，运行它。在GRANT语句变得能工作时，授权表的结构用**MySQL** 3.22.11修改 。

· 记住客户端程序使用选项文件或环境变量中指定的连接参数。如果客户端程序发送不正确的默认连接参数，而你没有在命令行中指定，检查环境变量和适用的选项文件。例如，当你不用任何选项运行客户端程序，得到Access denied错误，确保你没有在选项文件中指定旧密码！

你可以通过使用--no-defaults选项调用客户端程序来禁用选项文件。例如：

```
shell> mysqladmin --no-defaults -u root version
```

客户端使用的选项文件见4.3.2节，“使用选项文件”。环境变量列于附录F：环境变量。

· 如果遇到下述错误，说明root密码错误：

```
shell> mysqladmin -u root -pxxxx ver
Access denied for user 'root'@'localhost' (using password: YES)
```

如果你未指定密码时出现前面的错误，说明某个选项文件中的密码不正确。试试前面所说的--no-defaults选项。

关于密码更改的信息参见5.8.5节，“设置账户密码”。

如果你丢失或忘记root密码，你可以用--skip-grant-tables重启 **mysqld**来更改密码。参见A.4.1节，“如何复位根用户密码”.

· 如果你使用SET PASSWORD、INSERT或UPDATE更改密码，你必须使用 PASSWORD()函数加密密码。如果你不使用PASSWORD()函数，密码不工作。例如，下面的语句设置密码，但没能加密，因此用户后面不能连接：

```
mysql> SET PASSWORD FOR 'abe'@'host_name' = 'eagle';
```

相反，应这样设置密码：

```
mysql> SET PASSWORD FOR 'abe'@'host_name' = PASSWORD('eagle');
```

当你使用GRANT或CREATE USER语句或**mysqladmin password**命令指定密码时，不需要PASSWORD()函数，它们会自动使用PASSWORD()来加密密码。参见5.8.5节，“设置账户密码”和13.5.1.1节，“CREATE USER语法”。

· localhost是你本地主机名的一个同义词，并且也是如果你不明确地指定主机而客户端尝试连接的默认主机。

要想在这种系统上避免该问题，你可以使用--host=127.0.0.1选项来明确命名服务器主机。这样将通过TCP/IP协议来连接本地**mysqld**服务器。你还可以指定--host选项使用TCP/IP，使用实际的本机主机名。在这种情况下，主机名必须指定为服务器主机上的user表行，即使你在服务器上运行客户端程序。

· 当尝试用mysql -u user_name与数据库连接时，如果你得到一个Access denied错误，可能会遇到与user表有关的问题，通过执行mysql -u root mysql并且执行下面的SQL语句进行检查：

```
mysql> SELECT * FROM user;
```

结果应该包含一个有Host和User列的行匹配你的计算机主机名和你的**MySQL**用户名。

- Access denied错误消息将告诉你，你正在用哪个用户尝试登录，你正在试图连接哪个主机，是否使用了密码。通常，你应该在user表中有一行，正确地匹配在错误消息给出的主机名和用户名。例如，如果遇到包含using password: NO的错误信息，说明你登录时没有密码。

· 如果当你试着从一个不是**MySQL**服务器正在运行的主机上连接时，遇到下列错误，那么在user表中没有匹配那台主机的行：

```
Host ... is not allowed to connect to this MySQL server
```

可以通过组合你正在试图连接的用户/主机名设置一个账户来修正它。如果你不知道正连接的机器的IP号或主机名，应该把一个'%'行作为Host列值放在user表中。在试图从客户端器连接以后，通过SELECT USER()查询显示你如何真正进行连接。（然后用在日志文件上面显示出的实际的主机名代替user表中的'%'行。否则，你将得到一个不安全的系统，因为它允许从任何主机上以任何用户名连接。）

在Linux中，发生该错误的另一个原因可能是你正使用于你所使用版本的glibc库不同版本的库编译的二进制MySQL版本。在这种情况下，你应升级操作系统或glibc，或下载MySQL版本的源码分发版并自己编译。源码RPM一般很容易编译并安装，因此不是大问题。

· 如果你连接时指定主机名，但得到错误消息主机名未显示或为IP号，表示当MySQL服务器将IP号解析为客户端来名时遇到错误：

```
shell> mysqladmin -u root -pxxxx -h some-hostname ver
Access denied for user 'root'@'' (using password: YES)
```

这表示DNS问题。要想修复，执行**mysqladmin flush-hosts**来重设内部 DNS主机名缓存。参见7.5.6节，“MySQL如何使用DNS”。

一些常用的解决方案包括：

o 试试找出DNS服务器的错误并修复。

- 在MySQL授权表中指定IP号而不是主机名。
- 在/etc/hosts中放入客户端名。
- 用--skip-name-resolve选项启动**mysqld**。
- 用--skip-host-cache选项启动**mysqld**。
- 在Unix中，如果你在同一台机器上运行服务器和客户端，连接到localhost。连接到的localhost的Unix连接使用Unix套接字文件而不是TCP/IP。
- 在Windows中，你在同一台机器上运行服务器和客户端并且服务器支持命名管道连接，连接主机名(周期)。连接使用命名管道而不是TCP/IP。

- 如果mysql -u root test工作但是mysql -h your_hostname -u root test导致Access denied（*your_hostname*是本地机的实际主机名），那么在user表中可能没有你的主机的正确名字。这里的一个普遍的问题是在user表行中的Host值指定一个唯一的主机名，但是你系统的名字解析例程返回一个完全正规的域名(或相反)。例如，如果你在user表中有一个主机是'tcx'的行，但是你的DNS告诉**MySQL**你的主机名是'tcx.subnet.se'，行将不工作。尝试把一个行加到user表中，它包含你主机的IP号作为Host列的值。（另外，你可以把一个行加到user表中，它有包含一个通配符如'tcx.%'的Host值。然而，使用以“%”结尾的主机名是不安全的并且不推荐！）
- 如果mysql -u *user_name* test工作但是mysql -u *user_name other_db_name*不工作，你没有为给定的用户授予*other_db_name*数据库的访问权限。
- 当在服务器上执行mysql -u *user_name*时，它工作，但是在其它远程客户端上执mysql -h *host_name* -u *user_name*时，它却不工作，你没有为给定的用户授予从远程主机访问服务器的权限。
- 如果你不能弄明白你为什么得到Access denied，从user表中删除所有Host包含通配符值的行(包含“%”或“_”的条目)。一个很普遍的错误是用Host='%'和User='*some_user*'插入一个新行，认为这将允许你指定localhost从同一台机器进行连接。它不工作的原因是 默认权限包括一个有Host='localhost'和User=''的行，因为那个行的Host值'localhost'比'%'更具体，当从localhost连接时，它用于指向新行！正确的步骤是插入Host='localhost'和User='*some_user*'的第2个行，或删除Host='localhost'和User=''行。删除条目后，记住用FLUSH PRIVILEGES语句重载授权表。

- 如果你得到下列错误，可以与db或host表有关：

```
Access to database denied
```

如果从db表中选择了在Host列有空值的条目，保证在host表中有一个或多个相应的条目，指定db表中的条目适用哪些主机。

- 如果你能够连接MySQL服务器，但如果在使用命令SELECT ... INTO OUTFILE或LOAD DATA INFILE语句时，你得到Access denied错误，在user表中的条目可能没有启用FILE权限。

- 如果你直接更改授权表(例如，使用INSERT、UPDATE或DELETE语句)并且你的更改好像被忽略了，记住你必须执行FLUSH PRIVILEGES语句或**mysqladmin flush-privileges**命令让服务器来重读授权表。否则，直到服务器下次重启，你的更改方有效。记住用UPDATE命令更改root密码后，在清空权限前，你不需要指定新密码，因为服务器还不知道你已经更改了密码！

- 如果你的权限似乎在一个会话过程中改变了，可能是一个超级用户改变了他们。再次装入授权表会影响新客户端连接，但是它也影响现存的连接，如5.7.7节，“权限更改何时生效”小节所述。

- 如果你有Perl、Python或ODBC程序的存取问题，试着用mysql -u *user_name db_name*或mysql -u *user_name* -p*your_pass db_name*与服务器连接。如果你能用**mysql**客户端进行连接，这是程序的一个问题而不是访问权限的问题。（注意在-p和密码之间没有空格；也可以使用--password=*your_pass*语法指定密码。如果使用-p选项，MySQL提示你输入密码。）

- 为了测试，用--skip-grant-tables选项启动**mysqld**守护进程，然后你可以改变**MySQL**授权表并且使用**mysqlaccess**脚本检查你的修改是否有如期的效果。当你对你的改变满意时，执行**mysqladmin flush-privileges**告诉**mysqld**服务器开始使用新的 授权表。（再次装入授权表覆盖了--skip-grant-tables选项。这允许你告诉服务器开始使用授权表，而不用停掉并重启它）。

- 如果任何其它事情失败，用调试选项(例如，--debug=d,general,query)启动**mysqld**服务器。这将打印有关尝试连接的主机和用户信息，和发出的每个命令的信息。请参见E.1.2节，“创建跟踪文件”。

- 如果你有任何与**MySQL**授权表的其它问题，而且觉得你必须将这个问题发送到邮件表，一定要提供一个**MySQL**授权表的倾倒副本(dump)。你可用**mysqldump mysql**命令复制数据库表。象平时一样，用**mysqlbug**脚本邮寄你的问题。参见1.7.1.3节，“如何通报缺陷和问题”。在一些情况下可以用--skip-grant-tables重启**mysqld**以便能运行**mysqldump**。

5.7.9. MySQL 4.1中的密码哈希处理

5.7.9.1. 更改应用程序密码哈希值的含义

MySQL用户账户列于mysql数据库中的user表内。每个MySQL账户指定一个密码，尽管保存在user表Password列的密码不是明文，但哈希值是从表中的记录计算的。用PASSWORD()函数来计算密码的哈希值。

MySQL在客户端/服务器通信的两个阶段使用密码：

- 如果客户端试图连接服务器，有一个初始鉴定步骤，客户必须提供一个密码，并且必须与客户想要使用的账户在user表保存的哈希值匹配。

- 客户端连接后，它可以(如果有充分的权限) 设置或更改user表内所列的账户的密码哈希值值。客户端可以通过PASSWORD()函数来生成密码哈希值，或使用GRANT或SET PASSWORD语句。

换句话说，当客户端首次试图连接时，服务器使用哈希值进行鉴定。如果连接的客户端调用PASSWORD()函数或使用GRANT或SET语句来设置或更改密码，则服务器产生哈希值。

在MySQL 4.1中密码哈希算法已经更新，提供了更好的安全性并降低了密码被截取的风险。但是，该新机制只能在MySQL 4.1(和更新版本的)服务器和客户端中使用，会产生一些兼容性问题。4.1或新客户端可以连接4.1之前的服务器，因为客户端可以同时理解旧的和新的密码哈希机制。但是，4.1之前的客户端试

图连接4.1版或更新版的服务器时会遇到困难。例如，3.23版**mysql**客户端试图连接5.1服务器时会失败并出现下面的错误消息：

```
shell> mysql -h localhost -u root
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

出现该问题的另一个普通例子是在升级到MySQL 4.1或更新版后，试图使用旧版本的PHP **mysql**扩展名。(参见25.3.1节，“使用MySQL和PHP的常见问题”）。

下面讨论了新、旧密码机制之间的差别，以及如果你升级了服务器但需要为4.1版以前的客户端保持向后兼容性该怎样做。A.2.3节，“客户端不支持鉴定协议”中有更详细的信息。该信息将MySQL数据库从4.0版本或更低版升级到4.1版或更高版的PHP编程人员特别重要。

注释：该讨论对比了4.1版的行为和4.1前的行为，这儿描述的4.1中的行为实际上从4.1.1开始。MySQL 4.1.0是一个“旧”的发布，因为它的实施机制与4.1.1版和更新版中的稍有不同。在MySQL 5.0 参考手册中详细描述了4.1.0和最新版之间的差别。

在MySQL 4.1之前,用PASSWORD()函数计算的密码哈希值有16个字节长。应为：

```
mysql> SELECT PASSWORD('mypass');
+--------------------+
| PASSWORD('mypass') |
+--------------------+
| 6f8c114b58f2ce9e   |
+--------------------+
```

在MySQL 4.1之前，user表的Password列(保存了哈希值)也是16字节长。

在MySQL 4.1中,已经对PASSWORD()函数进行了修改，可以生成41字节的哈希值：

```
mysql> SELECT PASSWORD('mypass');
+-------------------------------------------+
| PASSWORD('mypass')                        |
+-------------------------------------------+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
+-------------------------------------------+
```

同样，user表的Password列必须有41字节长来保存这些值：

- 如果你新安装MySQL 5.1, Password列自动为41字节长。
- 从MySQL 4.1(4.1.1或4.1系列的更新版)升级到MySQL 5.1，应不会出现相关问题，因为两个版本使用相同的密码哈希机制。如果你想要将更早版本的MySQL升级到MySQL5.1,你应先升级到4.1，然后将4.1升级到5.1。

加宽的Password列可以同时保存新、旧格式的密码哈希值。可以有两种方式确定任何给定格式的密码哈希值：

- 明显的不同之处是长度(16字节和41字节)。
- 第2个不同之处是新格式的密码哈希值都以‘*’字符开头，而旧格式的密码绝对不是。

长密码哈希值具有更好的加密属性，并且客户端根据长哈希值进行鉴定比旧的短哈希值更加安全。

短密码哈希值和长密码哈希值之间的不同之处与服务器如何使用密码进行鉴定以及如何为执行密码更改操作的连接的客户端生成密码哈希值都有关。

服务器使用密码哈希值进行鉴定的方式受Password列的宽度影响：

- 如果列较短，只用短哈希鉴定。
- 如果列较长，可以有短或长哈希值，并且服务器可以使用任何一种格式：
  - 4.1之前的客户端可以连接，它们只可以使用旧的哈希机制，它们可以只鉴定有短哈希的账户。
  - 4.1及以后版本的客户端可以鉴定有短哈希或长哈希的账户。

对于短哈希账户的鉴定过程，4.1和以后版本的客户端比为旧版本的客户端实际要安全得多。从安全性角度，从最低安全到最安全的梯度为：

- 4.1之前的客户端用短密码哈希值进行鉴定
- 4.1或以后版本的客户端用短密码哈希值进行鉴定
- 4.1或以后版本的客户端用长密码哈希值进行鉴定

服务器为连接的客户端生成密码哈希值的方式受Password列宽度和--old-passwords选项的影响。4.1或更新版本的服务器只有满足某个条件才生成长哈希：Password列必须足够宽以容纳长哈希值并且未给定--old-passwords选项。这些条件适合：

- Password列必须足够宽以容纳长哈希(41字节)值。如果列没有更新，仍然为4.1之前的16字节宽，当客户端使用PASSWORD()、GRANT或SET PASSWORD执行密码更改操作时，服务器注意到长哈希不适合，只生成短哈希。如果你已经升级到4.1但还没有运行 **mysql_fix_privilege_tables**脚本来扩

宽Password列时会出现这种行为。

· 如果Password列足够宽，则可以保存短或长密码哈希值。在这种情况下，PASSWORD()、GRANT或SET PASSWORD生成长哈希，除非 用--old-passwords选项启动服务器。该选项强制服务器生成短密码哈希值。

--old-passwords选项的目的是当服务器生成长密码哈希值时，允许你维持同4.1之前的客户端的向后兼容性。该选项不影响鉴定(4.1和以后版本的客户端仍然可以使用有长密码哈希值的账户)，但它防止在密码更改操作中在user表中创建长密码哈希值。在这种情况下，该账户不能再用于4.1之前的客户端。没有--old-passwords选项，可能会出现下面的不期望的情况：

· 旧客户端连接有短密码哈希值的账户。

· 客户更改自己的密码。没有--old-passwords，可以为该账户生成长密码哈希值。

· 下次旧客户试图连接账户时不能连接上，因为账户有长密码哈希值，需要新的哈希机制进行鉴定。(一旦账户user表中为长密码哈希值，只有4.1和以后版本的客户端可以鉴定它，因为4.1之前的客户端不理解长哈希）。

该场景说明，如果你必须支持旧的4.1之前的客户端，不使用--old-passwords选项运行4.1或更新版本的服务器很危险。用--old-passwords运行服务器，密码更改操作不会生成长密码哈希值，这样旧客户端也可以访问账户。(这些客户端不能意外地因更改了密码将自己锁出去，并留下长密码哈希值）。

--old-passwords选项的不利之处是你创建或更改的密码使用短哈希，甚至对于4.1客户端也如此。这样，你丢失了长密码哈希值提供的安全性。如果你想要创建有长哈希的账户(例如，为4.1客户端)，你必须不使用--old-passwords来运行服务器。

下面的场景可用于运行4.1或以后的服务器，包括MySQL 5.1：

场景**1**：user表中的短Password列：

· 只有短哈希可以保存到Password列。

· 服务器只使用短哈希进行客户端鉴定。

· 对于连接的客户端，调用PASSWORD()、GRANT或SET PASSWORD的密码哈希生成操作专使用短哈希。对账户的任何更改均会生成短密码哈希值。

· --old-passwords选项可以使用但是多余，因为Password列较短，服务器只生成短密码哈希值。

场景**2**：长Password列；没有用--old-passwords选项启动服务器：

· 短或长哈希可以保存到Password列。

· 4.1和以后版本的客户端(包括5.1客户端)可以鉴定有短或长哈希的账户。

· 4.1之前的客户端只能鉴定有短哈希的账户。

· 对于连接的客户端，调用PASSWORD()、GRANT或SET PASSWORD的密码哈希生成操作专使用短哈希。对账户的任何更改均会生成短密码哈希值。

如前面所示，该场景的危险性在于4.1之前的客户端可能不能访问有短密码哈希值的账户。通过PASSWORD()、GRANT或SET PASSWORDA对这些账户密码的更改会产生长的密码哈希值。从该点看，4.1之前的客户端升级到4.1之前不能鉴定该账户。

要处理该问题，可以用特殊方法更改密码。例如，一般情况你可以使用SET PASSWORD按照下面的方法更改账户密码：

```
mysql> SET PASSWORD FOR 'some_user'@'some_host' = PASSWORD('mypass');
```

要想更改密码但创建短哈希，使用OLD_PASSWORD()函数：

```
mysql> SET PASSWORD FOR 'some_user'@'some_host' = OLD_PASSWORD('mypass');
```

当你想明显生成短哈希时，OLD_PASSWORD()很有用。

场景**3**：长Password列；用--old-passwords选项启动4.1或新版本的服务器：

· 短或长哈希可以保存到Password列。

· 4.1和以后版本的客户端可以鉴定有短或长哈希的账户(请注意只有不使用--old-passwords选项启动服务器，方可以创建长哈希)。

· 4.1之前的客户端只可以鉴定短哈希账户。

· 对于连接的客户端，调用PASSWORD()、GRANT或SET PASSWORD的密码哈希生成操作专使用短哈希。对账户的任何更改均会生成短密码哈希值。

在该场景中，你不能创建长密码哈希值的账户，因为--old-passwords选项防止生成长哈希。并且，如果你在使用--old-passwords选项前创建长哈希账户，当--old-passwords有时更改账户密码，结果会使账户的密码为短密码，安全性较长哈希要降低。

这些场景的不利之处可以概括为：

在场景1中,你不能利用长哈希提供更安全的鉴定。

在场景2中, 如果你没有显式使用OLD_PASSWORD()来更改密码，则4.1之前的客户端不能再访问短哈希账户。

在场景3中,--old-passwords防止短哈希账户不可访问，但密码更改操作使账户的长哈希转换为短哈希，当--old-passwords有效时不能将它改回长哈希。

5.7.9.1. 更改应用程序密码哈希值的含义

升级到MySQL4.1或更新版本后，使用PASSWORD()为自己的目的生成密码的应用程序会出现兼容性问题。应用程序实际不应这样做，因为PASSWORD()只应用来管理MySQL账户的密码。但一些应用程序使用PASSWORD()用于自己的目的。

如果你从MySQL 4.1之前的版本升级到4.1或以后版本，并在生成长密码哈希值的条件下运行服务器，应用程序使用PASSWORD()破解自己的密码。这种情况下推荐的方法是修改应用程序，使用其它函数，例如SHA1()或MD5()，来产生哈希值。如果不行，你可以使用OLD_PASSWORD()函数，该函数用来提供旧格式的短哈希。但是，请注意OLD_PASSWORD()可能有一天不再被支持。

如果服务器运行在生成短哈希的条件下，可以使用 OLD_PASSWORD()但与PASSWORD()等同。

将MySQL数据库从4.0或更低版本移植到4.1或更高版本的PHP编程人员应参阅旧客户端。

5.8. MySQL用户账户管理

5.8.1. MySQL用户名和密码
5.8.2. 向MySQL增加新用户账户
5.8.3. 从MySQL删除用户账户
5.8.4. 限制账户资源
5.8.5. 设置账户密码
5.8.6. 使你的密码安全
5.8.7. 使用安全连接

本节描述如何为MySQL服务器的客户端设置账户。讨论了下面的主题:

- MySQL使用的账户名和密码的含义，以及如何比较你的操作系统所使用的账户名和密码
- 如何设置新账户并移除已有账户
- 如何更改密码
- 安全使用密码指导
- 如何使用安全SSL连接

5.8.1. MySQL用户名和密码

用用户名和客户端或主机定义MySQL账户，用户可以根据这些名称来连接服务器。账户也有密码。MySQL和操作系统使用用户名和密码的方式有几处区别:

- MySQL用于鉴定目的用户名与Windows或Unix使用的用户名(登录名)没有关系。在Unix中，大多数MySQL客户端默认试图使用当前Unix的用户名作为MySQL用户名来登录，但这样只是为了方便。 默认值可以很容易被覆盖，因为客户端程序允许用-u或--user选项来指定用户名。因为这表示任何人可以试图使用任何用户名来连接服务器，除非所有MySQL账户有密码，否则你不能使数据库保持安全。通过为没有密码的账户指定用户名，任何人能够成功连接服务器。

- MySQL用户名最大客达16字符长。这样可以限制MySQL服务器和客户端之间的硬编码，并且防止通过修改mysql数据库中表的定义来偷窃密码。

注:你应绝对不要以任何方式修改*mysql*数据库中的任何表，只能运行*MySQL*分发中专为此目的提供的脚本。将*MySQL*系统表重新定义为其它方式会导致未定义的*(*和不支持的*!)*行为。

操作系统用户名与MySQL用户名完全不相关，甚至最大长度可能不同。例如， Unix用户名限制为8个字符。

- MySQL密码与登录到你的操作系统的密码没有关系。不需要将你用来登录Windows或Unix机器的密码和你用来访问该机器上的MySQL服务器的密码关联起来。

- MySQL的加密密码使用自己的算法。该加密算法不同于Unix登录过程使用的算法。MySQL密码加密与PASSWORD()SQL函数的方法相同。Unix密码加密与ENCRYPT()SQL函数的方法相同。PASSWORD()和ENCRYPT()函数的描述参见12.9.2节，“加密函数”。从版本4.1 起，MySQL使用更强的鉴定方法，同以前的版本相比可以在连接过程中提供更好的密码保护。即使TCP/IP包被截取或mysql数据库被捕获也很安全。(在前面的版本中，即使密码以加密形式保存到user表中，仍可以通过加密密码值来连接MySQL服务器）。

当安装MySQL时，授权表装载时有一系列初使账户。这些账户的名称和访问权限见2.9.3节，“使初始MySQL账户安全”，其中还讨论了如何未这些账户赋予密码。因此，你一般应使用GRANT和REVOKE语句来设置、修改和移除MySQL账户。参见13.5.1.3节，“GRANT和REVOKE语法”。

当用命令行客户端连接MySQL服务器时，你应为想要使用的账户指定用户名和密码:

```
shell> mysql --user=monty --password=guess db_name
```

如果你想用较短的选项，命令应为:

```
shell> mysql -u monty -pguess db_name
```

-p选项和后面的密码值之间绝对不能有空格。参见5.7.4节，“与MySQL服务器连接”。

前面的命令包括命令行中的密码值，会很危险。参见5.8.6节，“使你的密码安全”。要想避免，指定--password或-p选项后面不跟任何密码值:

```
shell> mysql --user=monty --password db_name
shell> mysql -u monty -p db_name
```

然后客户端程序输出提示符并等待你输入密码。(在这些示例中，*db_name*并不为密码，因为用空格将它同前面的密码项隔离开了）。

在一些系统中，MySQL用来提示输入密码的库调用自动将密码限制到8个字符。这是系统库的问题，而不是MySQL的问题。MySQL本身并不限制密码的长度。要解决该问题，将MySQL密码改为8个字符和更少字符的值，或将密码放入选项文件中。

5.8.2. 向MySQL增加新用户账户

可以用两种方式创建MySQL账户：

- 使用GRANT语句
- 直接操作MySQL授权表

最好的方法是使用GRANT语句，因为这样更精确，错误少。从MySQL 3.22.11起提供了GRANT；其语法见13.5.1.3节，“GRANT和REVOKE语法”。

创建账户的其它方法是使用MySQL账户管理功能的第三方程序。phpMyAdmin即是一个程序。

下面的示例说明如何使用**MySQL**客户端程序来设置新用户。假定按照2.9.3节，“使初始MySQL账户安全”描述的 默认值来设置权限。这说明为了更改，你必须以MySQL root用户连接MySQL服务器，并且root账户必须有mysql数据库的INSERT权限和RELOAD管理权限。

首先，使用**MySQL**程序以MySQL root用户来连接服务器：

```
shell> MySQL --user=root MySQL
```

如果你为root账户指定了密码，还需要为该**MySQL**命令和本节中的其它命令提供--password或-p选项。

以root连接到服务器上后，可以添加新账户。下面的语句使用GRANT来设置四个新账户：

```
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';
```

用GRANT语句创建的账户有下面的属性：

- 其中两个账户有相同的用户名monty和密码some_pass。两个账户均为超级用户账户，具有完全的权限可以做任何事情。一个账户('monty'@'localhost')只用于从本机连接时。另一个账户('monty'@'%')可用于从其它主机连接。请注意monty的两个账户必须能从任何主机以monty连接。没有localhost账户，当monty从本机连接时，**mysql_install_db**创建的localhost的匿名用户账户将占先。结果是，monty将被视为匿名用户。原因是匿名用户账户的Host列值比'monty'@'%'账户更具体，这样在user表排序顺序中排在前面。(user表排序的讨论参见5.7.5节，“访问控制, 阶段1：连接核实”）。
- 一个账户有用户名admin，没有密码。该账户只用于从本机连接。授予了RELOAD和PROCESS管理权限。这些权限允许admin用户执行**mysqladmin reload**、**mysqladmin refresh**和**mysqladmin flush-***xxx*命令，以及**mysqladmin processlist**。未授予访问数据库的权限。你可以通过GRANT语句添加此类权限。
- 一个账户有用户名dummy，没有密码。该账户只用于从本机连接。未授予权限。通过GRANT语句中的USAGE权限，你可以创建账户而不授予任何权限。它可以将所有全局权限设为'N'。假定你将在以后将具体权限授予该账户。

除了GRANT，你可以直接用INSERT语句创建相同的账户，然后使用FLUSH PRIVILEGES告诉服务器重载授权表：

```
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;
```

当你用INSERT创建账户时使用FLUSH PRIVILEGES的原因是告诉服务器重读授权表。否则，只有重启服务器后更改方会被注意到。使用 GRANT，则不需要使用FLUSH PRIVILEGES。

用INSERT使用PASSWORD()函数是为了加密密码。GRANT语句为你加密密码，因此不需要PASSWORD()。

'Y'值启用账户权限。对于admin账户，还可以使用更加可读的INSERT扩充的语法（使用SET）。

在为dummy账户的INSERT语句中，只有user表中的Host、User和Password列记录为指定的值。没有一个权限列为显式设置，因此MySQL将它们均指定为 默认值'N'。这样等同于GRANT USAGE的操作。

请注意要设置超级用户账户，只需要创建一个权限列设置为'Y'的user表条目。user表权限为全局权限，因此其它 授权表不再需要条目。

下面的例子创建3个账户，允许它们访问专用数据库。每个账户的用户名为custom，密码为obscure。

要想用GRANT创建账户，使用下面的语句：

```
shell> MySQL --user=root MySQL
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';
```

这3个账户可以用于：

- 第1个账户可以访问bankaccount数据库，但只能从本机访问。
- 第2个账户可以访问expenses数据库，但只能从主机whitehouse.gov访问。
- 第3个账户可以访问customer数据库，但只能从主机server.domain访问。

要想不用GRANT设置custom账户，使用INSERT语句直接修改 授权表：

```
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;
```

前3个INSERT语句在user表中加入条目，允许用户custom从各种主机用给定的密码进行连接，但不授予全局权限(所有权限设置为 默认值'N')。后面3个INSERT语句在user表中加入条目，为custom授予bankaccount、expenses和customer数据库权限，但只能从合适的主机访问。通常若直接修改 授权表，则应告诉服务器用FLUSH PRIVILEGES重载授权表，使权限更改生效。

如果你想要让某个用户从给定域的所有机器访问(例如，mydomain.com)，你可以在账户名的主机部分使用含'%'通配符的GRANT语句：

```
mysql> GRANT ...
    ->     ON *.*
    ->     TO 'myname'@'%.mydomain.com'
    ->     IDENTIFIED BY 'mypass';
```

要想通过直接修改授权表来实现:

```
mysql> INSERT INTO user (Host,User,Password,...)
    ->     VALUES('%.mydomain.com','myname',PASSWORD('mypass'),...);
```

mysql> **FLUSH PRIVILEGES;**

5.8.3. 从MySQL删除用户账户

要想移除账户，应使用DROP USER语句，请参见13.5.1.2节，“DROP USER语法”。

5.8.4. 限制账户资源

限制MySQL服务器资源使用的一个方法是将max_user_connections系统变量设置为非零值。但是，该方法严格限于全局，不允许管理具体账户。并且，它只限制使用单一账户同时连接的数量，而不是客户端连接后的操作。许多MySQL管理员对两种类型的控制均感兴趣，特别是Internet服务提供者。

在MySQL 5.1中,你可以为具体账户限制下面的服务器资源:

- 账户每小时可以发出的查询数
- 账户每小时可以发出的更新数
- 账户每小时可以连接服务器的次数

客户端可以执行的语句根据查询限制来记数。只有修改数据库或表的语句根据更新限制来记数。

还可以限制每个账户的同时连接服务器的连接数。

本文中的账户为user表中的单个记录。根据User和Host列值唯一识别每个账户。

做为使用该特性的先决条件，mysql数据库的user表必须包含资源相关的列。资源限制保存在max_questions、max_updates、max_connections和max_user_connections列内。如果user表没有这些列，必须对它进行升级；参见2.10.2节，“升级授权表”。

要想用GRANT语句设置资源限制，使WITH子句来命名每个要限制的资源和根据每小时记数的限制值。例如，要想只以限制方式创建可以访问customer数据库的新账户，执行该语句:

```
mysql> GRANT ALL ON customer.* TO 'francis'@'localhost'
    ->     IDENTIFIED BY 'frank'
    ->     WITH MAX_QUERIES_PER_HOUR 20
    ->          MAX_UPDATES_PER_HOUR 10
    ->          MAX_CONNECTIONS_PER_HOUR 5
    ->          MAX_USER_CONNECTIONS 2;
```

限制类型不需要全部在WITH子句中命名，但已经命名的可以按任何顺序。每个每小时限制值均应为整数，代表每小时的记数。如果GRANT语句没有WITH子句，则每个限制值设置为 默认值零(即没有限制)。对于MAX_USER_CONNECTIONS，限制为整数，表示账户一次可以同时连接的最大连接数。如果限制设置为 默认值零，则根据MAX_USER_CONNECTIONS系统变量确定该账户可以同时连接的数量。

要想设置或更改已有账户的限制，在全局级别使用GRANT USAGE语句(在*.*)。下面的语句可以将francis的查询限制更改为100:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
    ->     WITH MAX_QUERIES_PER_HOUR 100;
```

该语句没有改变账户的已有权限，只修改了指定的限制值。

要想取消已有限制，将该值设置为零。例如，要想取消francis每小时可以连接的次数的限制，使用该语句:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
    ->     WITH MAX_CONNECTIONS_PER_HOUR 0;
```

当账户使用资源时如果有非零限制，则对资源使用进行记数。

服务器运行时，它统计每个账户使用资源的次数。如果账户在最后一个小时的连接次数达到限制，该账户的进一步的连接被拒绝。类似地，如果账户达到查询或更新次数的限制，进一步的查询或更新被拒绝。在这种情况下，会给出相关错误消息。

根据每个账户进行资源计算，而不是根据每个客户端。例如，如果你的账户的查询限制为50,你不能通过两个客户端同时连接服务器将限制增加到100。两个连接的查询被计算到一起。

可以为所有账户从全局重设当前的每小时资源使用记数，或单独重设给定的账户：

- 要想将所有账户当前的记数重设为零，可以执行FLUSH USER_RESOURCES语句。还可以通过重载授权表来重设记数(例如，使用FLUSH PRIVILEGES语句或**mysqladmin reload**命令)。
- 将具体账户的限制重新授予任何值，可以将它设置为零。要想实现，按照前面所述使用GRANT USAGE，并将限制值指定为该账户当前的限制值。

计数器重设不影响MAX_USER_CONNECTIONS限制。

当服务器启动时所有记数从零开始。

5.8.5. 设置账户密码

- 可以用**mysqladmin**命令在命令行指定密码：

  ```
  shell> mysqladmin -u user_name -h host_name password "newpwd"
  ```

  该命令重设密码的账户为user表内匹配User列的*user_name*和Host列你发起连接的客户端的记录。

  为账户赋予密码的另一种方法是执行SET PASSWORD语句：

  ```
  mysql> SET PASSWORD FOR 'jeffrey'@'%' = PASSWORD('biscuit');
  ```

  只有root等可以更新mysql数据库的用户可以更改其它用户的密码。如果你没有以匿名用户连接，省略FOR子句便可以更改自己的密码：

  ```
  mysql> SET PASSWORD = PASSWORD('biscuit');
  ```

  你还可以在全局级别使用GRANT USAGE语句(在*.*)来指定某个账户的密码而不影响账户当前的权限：

  ```
  mysql> GRANT USAGE ON *.* TO 'jeffrey'@'%' IDENTIFIED BY 'biscuit';
  ```

  一般情况下最好使用上述方法来指定密码，你还可以直接修改user表：

  - 要想在创建新账户时建立密码，在Password列提供一个值：

    ```
    shell> mysql -u root mysql
    mysql> INSERT INTO user (Host,User,Password)
        -> VALUES('%','jeffrey',PASSWORD('biscuit'));
    mysql> FLUSH PRIVILEGES;
    ```

  - 要想更改已有账户的密码，使用UPDATE来设置Password列值：

    ```
    shell> mysql -u root mysql
     mysql> UPDATE user SET Password = PASSWORD('bagel')
           -> WHERE Host = '%' AND User = 'francis';
    mysql> FLUSH PRIVILEGES;
    ```

  当你使用SET PASSWORD、INSERT或UPDATE指定账户的密码时，必须用PASSWORD()函数对它进行加密。(唯一的特例是如果密码为空，你不需要使用PASSWORD())。需要使用PASSWORD()是因为user表以加密方式保存密码，而不是明文。如果你忘记了，你可能会象这样设置密码：

  ```
  shell> mysql -u root mysql
  mysql> INSERT INTO user (Host,User,Password)
      -> VALUES('%','jeffrey','biscuit');
  mysql> FLUSH PRIVILEGES;
  ```

  结果是密码'biscuit'保存到user表后没有加密。当jeffrey使用该密码连接服务器时，值被加密并同保存在user表中的进行比较。但是，保存的值为字符串'biscuit'，因此比较将失败，服务器拒绝连接：

  ```
  shell> mysql -u jeffrey -pbiscuit test
  Access denied
  ```

  如果你使用GRANT ... IDENTIFIED BY语句或**mysqladmin password**命令设置密码，它们均会加密密码。在这种情况下，不需要使用 PASSWORD()函数。

  注释：PASSWORD()加密不同于Unix密码加密。参见5.8.1节，“MySQL用户名和密码”。

5.8.6. 使你的密码安全

在管理级别，你决不能将mysql.user表的访问权限授予任何非管理账户。

当你运行客户端程序连接MySQL服务器时，以一种暴露的可被其他用户发现的方式指定你的密码是不妥当的。当你运行客户端程序时，你可以使用下列方法指定你的密码，还有每个方法的风险评估：

- 使用一个在命令行上-p*your_pass*或--password=*your_pass*的选项。例如：

```
shell> mysql -u francis -pfrank db_name
```

这很方便但是不安全，因为你的密码对系统状态程序(例如ps)变得可见，它可以被其他的用户调用来显示命令行。一般**MySQL**客户在他们的初始化顺序期间用零覆盖命令行参数，但是仍然有一个短暂间隔时间内参数值可见的。

- 使用一个-p或--password选项(没有指定密码)。在这种情况下，客户端程序请求来自终端的密码：

```
shell> mysql -u francis -p db_name
Enter password: ********
```

"*"字符指示输入密码的地方。输入密码时密码看不见。

因为它对其他用户不可见，与在命令行上指定它相比，这样进入你的密码更安全。然而，这个输入一个密码的方法仅仅为你交互式运行程序是合适的。如果你想要从非交互式运行的一个脚本调用一个客户端，就没有从终端输入入密码的机会。在某些系统中，你甚至会发现脚本的第一行被（错误地）读并解释为你的密码！

- 在一个配置文件中存储你的密码。例如，在Unix中，你可在主目录的".my.cnf"文件中的[client]节列出你的密码：

```
[client]
password=your_pass
```

如果你在".my.cnf"里面存储密码，除了你本人其它人不能访问该文件。保证文件的访问模式是400或600。例如：

```
shell> chmod 600 .my.cnf
```

关于选项文件的详细讨论参见4.3.2节，"使用选项文件"。

- 你可在MYSQL_PWD环境变量中存储密码。但是这种指定MySQL密码的方法是极不安全的，不应该使用。ps的某些版本包括显示运行进程的环境的选项；如果你设定MYSQL_PWD，你的密码将被运行**ps**的所有人看见，甚至在没有这样一个版本的ps的系统上，没有其它方法观察到进程环境的假设是不明智的。参见附录F：环境变量。

总之，最安全的方法是让客户端程序提示输入密码或在适当保护的选项文件中指定密码。

5.8.7. 使用安全连接

5.8.7.1. SSL基本概念
5.8.7.2. 需求(OpenSSL)
5.8.7.3. 使用yaSSL
5.8.7.4. 为MySQL设置SSL证书
5.8.7.5. SSL GRANT 选项
5.8.7.6. SSL命令行选项
5.8.7.7. 用SSH以远程方式从Windows连接到MySQL

MySQL支持MySQL客户端和服务器之间的安全(加密的)连接所使用的安全套接字层(SSL)协议。本节讨论如何使用SSL连接。还描述了在Windows中设置SSH的方法。

MySQL的标准配置倾向于尽可能快，因此默认情况不使用加密连接。使用该协议会使客户端/服务器协议慢得多。对数据进行加密非常耗CPU，需要计算机多做许多工作，会延迟MySQL的其它任务。对于需要通过加密连接提供安全的应用程序，可以保证额外的计算。

MySQL允许在连接前启用加密。你可以根据具体应用程序的需求选择普通未加密连接或安全加密SSL连接。

5.8.7.1. SSL基本概念

要想理解MySQL如何使用SSL，需要解释一些基本SSL和X509概念。熟悉的人们可以跳过该部分。

默认情况下，MySQL在客户端和服务器之间使用未加密的连接。这说明可以访问网络的部分人可以看到你的通信，并看到发送和接收的数据。他们甚至可以更改在客户端和服务器之间传递的数据。要想提高安全性，当调用客户端程序时，你可以通过--compress选项压缩客户端/服务器之间的通信。但是，这样并不能阻挡住顽固的攻击者。

当你需要以安全方式在网络中传递信息时，未加密的连接是不可接受的。加密是使任何数据不可读的方法。事实上，今天的许多惯例需要加密算法提供更加安全的要素。它们应能抵抗各种已知的攻击，例如更改加密消息的顺序或两次重放数据。

SSL是一种使用不同的加密算法确保从公用网接收到的数据是可信的协议。它具有检测数据更改、丢失或重放的机制。SSL还包括使用 X509标准提供身份认证的算法。

使用X509，可以识别Internet上的某些人。通常用于电子商务应用程序中。按照基本概念，应有某种称之为"认证机构"(或CA)的机构，可以向请求者分发电子证书。证书依赖非对称加密算法，有两个加密密钥(公共密钥和私人密钥)。认证持有者可以向其它方出示证书来证明身份。证书包括持有者的公共密钥。只能使用对应的私人密钥对含该公共密钥的加密数据进行解密，私人密钥由证书持有者拥有。

如果你需要关于SSL、X509、或加密的详细信息，使用Internet搜索引擎来搜索你感兴趣的关键字。

5.8.7.2. 需求(OpenSSL)

要想在MySQL服务器和客户端程序之间使用SSL连接，系统必须能够支持OpenSSL。如果用支持内嵌式yaSSL的MySQL版本，不要读该节，但应阅读5.8.7.3节，"使用yaSSL"。

要想获得安全的MySQL连接，必须：

1. 安装OpenSSL库。我们已经测试了带OpenSSL 0.9.6的MySQL。如果你需要OpenSSL，请访问http://www.openssl.org。
2. 配置MySQL，用--with-vio和--with-openssl选项运行**configure**脚本。
3. 确保升级了授权表，使mysql.user表内包含SSL相关列。如果 授权表是从MySQL 4.0.0之前的版本升级，这很重要。升级过程见2.10.2节，“升级授权表”。
4. 要想检查是否运行的**mysqld**服务器支持OpenSSL，应检查have_openssl系统变量的值：

```
5.          mysql> SHOW VARIABLES LIKE 'have_openssl';
6.          +---------------+-------+
7.          | Variable_name | Value |
8.          +---------------+-------+
9.          | have_openssl  | YES   |
10.       +---------------+-------+
```

如果值为YES，服务器支持OpenSSL连接。

5.8.7.3. 使用**yaSSL**

使用MySQL的内嵌式yaSSL支持，可以很容易地使用安全连接。不需要安装OpenSSL和执行5.8.7.2节，“需求(OpenSSL)”中的步骤。并且，MySQL和yaSSL使用相同的许可模型。

当前，在以下平台上支持yaSSL：

- Linux/x86-64 Red Hat Enterprise 3.0
- Linux RHAS21 Itanium-2，带gcc，静态链接
- Linux Itanium-2，带gcc
- Windows

当从源码构建MySQL时如果你想要启用yaSSL，应这样配置MySQL：

```
./configure --with-yassl=yes
```

要想启动MySQL服务器支持yaSSL，使用支持OpenSSL的相同的选项，并识别建立安全连接需要的证书：

```
shell> mysqld --ssl-ca=cacert.pem \
       --ssl-cert=server-cert.pem \
       --ssl-key=server-key.pem
```

- --ssl-ca识别认证机构证书。
- --ssl-cert识别服务器证书。
- --ssl-key识别客户证书。

要想用yaSSL支持建立与MySQL服务器的安全连接，应这样启动客户端：

```
shell> mysql --ssl-ca=cacert.pem \
       --ssl-cert=server-cert.pem \
       --ssl-key=server-key.pem
```

换句话说，选项与服务器的相同，并且认证机构证书必须相同。

要想从应用程序建立安全连接，调用mysql_real_connect()之前，应使用mysql_ssl_set()API函数来设置相应认证选项。参见25.2.3.64节，“mysql_ssl_set()”。

5.8.7.4. 为**MySQL**设置**SSL**证书

下面是一个为MySQ设置SSL证书的例子：

```
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
```

要想测试SSL连接，按下面方法启动服务器，其中$DIR是示例my.cnf选项文件安装的路径名：

```
shell> MySQLd --defaults-file=$DIR/my.cnf &
```

然后使用相同的选项文件调用客户端程序：

```
shell> MySQL --defaults-file=$DIR/my.cnf
```

如果你有MySQL源码分发版，还可以修改前面的my.cnf文件来指向SSL目录中的示范证书和密钥文件来测试你的设置。

5.8.7.5. SSL GRANT 选项

MySQL可以检查X509证书的属性和基于用户名和密码的通用鉴定方法。要想为MySQL账户指定SSL相关选项，使用GRANT语句的REQUIRE子句。参见13.5.1.3节，“GRANT和REVOKE语法”。

有多种可能来限制一个账户的连接类型：

- 如果账户没有SSL或X509需求，如果用户名和密码合法，允许未加密的连接。但是，如果客户有正确的证书和密钥文件，在客户选项中可以使用加密连接。
- REQUIRE SSL选项限制服务器只允许该账户的SSL加密连接。请注意如果有ACL记录允许非SSL连接，该选项会被忽略。

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    -> IDENTIFIED BY 'goodsecret' REQUIRE SSL;
```

- REQUIRE X509表示客户必须有合法证书但确切的证书、分发者和主体不重要。唯一的需求是应可以被某个CA认证机构验证它的签名。

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    -> IDENTIFIED BY 'goodsecret' REQUIRE X509;
```

- REQUIRE ISSUER 'issuer'限制连接企图，即客户必须出示CA 'issuer'签发的合法X509证书。如果客户出示了一个合法证书，但是是由不同的分发者签发，服务器拒绝连接。使用X509证书表示要加密，因此不需要SSL选项。

```
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@example.com';
```

请注意ISSUER值应做为单一字符串输入。

- REQUIRE SUBJECT 'subject' 限制连接企图，即客户必须出示主题为'subject'的合法X509证书。如果客户出示了一个合法证书，但是有不同的主题，服务器拒绝连接。

```
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@example.com';
```

请注意SUBJECT值应做为单一字符串输入。

· REQUIRE CIPHER 'cipher'用来确保使用足够强的密码和密钥长度。如果使用旧的短加密密钥算法，SSL本身可能很弱。使用该选项，我们可以索取确切的加密方法来连接。

```
·                mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
·                    -> IDENTIFIED BY 'goodsecret'
    -> REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

在REQUIRE子句中，可以结合使用SUBJECT、ISSUER和CIPHER选项：

```
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@example.com'
    -> AND ISSUER '/C=FI/ST=Some-State/L=Helsinki/
       O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com'
    -> AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

请注意SUBJECT和ISSUER值应做为单一字符串输入。

在MySQL 5.1中,在REQUIRE选项之间可以选用AND关键字。

选项的顺序不重要，但任何选项不能用两次。

5.8.7.6. SSL命令行选项

下面列出了规定SSL、证书文件和密钥文件使用的选项。它们可以位于命令行中或选项文件中。

· --ssl

对于服务器，该选项规定该服务器允许SSL连接。对于客户端程序，它允许客户使用SSL连接服务器。单单该选项不足以使用SSL连接。还必须指定--ssl-ca、--ssl-cert和--ssl-key选项。

通常从反向使用该选项表示不应使用SSL。要想实现，将选项指定为--skip-ssl或--ssl=0。

请注意使用--ssl不需要SSL连接。例如，如果编译的服务器或客户不支持SSL，则使用普通的未加密的连接。

确保使用SSL连接的安全方式是使用含REQUIRE SSL子句的GRANT语句在服务器上创建一个账户。然后使用该账户来连接服务器，服务器和客户端均应启用SSL支持。

· --ssl-ca=*file_name*

含可信SSL CA的清单的文件的路径。

· --ssl-capath=*directory_name*

包含pem格式的可信SSL CA证书的目录的路径。

· --ssl-cert=*file_name*

SSL证书文件名，用于建立安全连接。

· --ssl-cipher=*cipher_list*

允许的用于SSL加密的密码的清单。*cipher_list*的格式与OpenSSL ciphers命令相同。

示例：--ssl-cipher=ALL:-AES:-EXP

· --ssl-key=*file_name*

SSL密钥文件名，用于建立安全连接。

5.8.7.7. 用SSH以远程方式从Windows连接到MySQL

本节说明如何用SSH安全连接到远程MySQL服务器(David Carlson <dcarlson@mplcomm。com>)：

1. 在Windows主机上安装SSH客户端。作为用户，我所发现的最好的非免费客户端来自http://www.vandyke.com/的SecureCRT。另一个选则是http://www.f-secure.com/的f-secure。你还可以从http://directory.google.com/Top/Computers/Security/Products_and_Tools/Cryptography/SSH/Clients/Windows/ 的Google找到一些免费客户端。

2. 启动Windows SSH客户端。设置Host_Name = yourmysqlserver_URL_or_IP。设置userid=your_userid以便登录服务器。此userid值可以与你的MySQL账户的用户名不相同。

3. 设置端口映射。可以进行远程映射(设置local_port: 3306, remote_host: yourmysqlservername_or_ip, remote_port: 3306)或本地映射(设置port: 3306, host: localhost, remote port: 3306)。

4. 进行保存，否则下次需要重设。

5. 刚创建的SSH会话登录服务器。

6. 在Windows机器上启动相应ODBC应用程序(例如Access)。

7. 在Windows中创建一个新的文件按照常用方法通过ODBC驱动程序链接MySQL，不同的是要为MySQL服务器输入localhost，而不是yourmysqlservername。

你应有使用SSH加密的ODBC连接到MySQL。

5.9. 备份与恢复

5.9.1. 数据库备份
5.9.2. 示例用备份与恢复策略
5.9.3. 自动恢复
5.9.4. 表维护和崩溃恢复
5.9.5. myisamchk：MyISAM表维护实用工具
5.9.6. 建立表维护计划
5.9.7. 获取关于表的信息

本节讨论如何进行数据库备份(完全备份和增量备份)，以及如何执行表的维护。本节描述的SQL语句语法参见第5章：数据库管理。此处提供的大多数信息主要适合MyISAM表。InnoDB备份程序参见15.2.8节，“InnoDB数据库的备份和恢复”。

5.9.1. 数据库备份

因为MySQL表保存为文件方式，很容易备份。要想保持备份的一致性，对相关表执行LOCK TABLES操作，然后对表执行FLUSH TABLES。参见13.4.5节，“LOCK TABLES和UNLOCK TABLES语法”和13.5.5.2节，“FLUSH语法”。你只需要读锁定；这样当你复制数据库目录中的文件时，允许其它客户继续查询表。需要FLUSH TABLES语句来确保开始备份前将所有激活的索引页写入硬盘。

如果你想要进行SQL级别的表备份，你可以使用SELECT INTO ...OUTFILE或BACKUP TABLE。对于SELECT INTO ...OUTFILE， 输出的文件不能先存在。对于BACKUP TABLE也如此，因为覆盖完整的文件会有安全风险。参见13.2.7节，“SELECT语法”和13.5.2.2节，“BACKUP TABLE语法”。

备份数据库的另一个技术是使用**mysqldump**程序或**mysqlhotcopy**脚本。参见8.8节，“mysqldump：数据库备份程序”和8.9节，“mysqlhotcopy：数据库备份程序”。

1. 完全备份数据库：

2.
```
shell> mysqldump --tab=/path/to/some/dir --opt db_name
```

或：

```
shell> mysqlhotcopy db_name /path/to/some/dir
```

只要服务器不再进行更新，还可以只复制所有表文件(*.frm、*.MYD和*.MYI文件)。**mysqlhotcopy**脚本使用该方法。(但请注意如果数据库包含InnoDB表，这些方法不工作。InnoDB不将表的内容保存到数据库目录中，**mysqlhotcopy**只适合MyISAM表）。

3. 如果**mysqld**在运行则停止，然后用--log-bin[=file_name]选项来启动。参见5.11.3节，“二进制日志”。二进制日志文件中提供了 执行**mysqldump**之后对数据库的更改进行复制所需要的信息。

对于InnoDB表，可以进行在线备份，不需要对表进行锁定；参见8.8节，“mysqldump：数据库备份程序”。

MySQL支持增量备份:需要用--log-bin选项来启动服务器以便启用二进制日志；参见5.11.3节，“二进制日志”。当想要进行增量备份时(包含上一次完全备份或增量备份之后的所有更改)，应使用FLUSH LOGS回滚二进制日志。然后，你需要将从最后的完全或增量备份的某个时刻到最后某个点的所有二进制日志复制到备份位置。这些二进制日志为增量备份；恢复时，按照下面的解释应用。下次进行完全备份时，还应使用FLUSH LOGS或mysqlhotcopy --flushlogs回滚二进制日志。参见8.8节，“mysqldump：数据库备份程序”和8.9节，“mysqlhotcopy：数据库备份程序”。

如果MySQL服务器为从复制服务器，则无论选择什么备份方法，当备份从机数据时，还应备份master.info和relay-log.info文件。恢复了从机数据后，需要这些文件来继续复制。如果从机执行复制LOAD DATA INFILE命令，你应还备份用--slave-load-tmpdir选项指定的目录中的SQL_LOAD-*文件。(如果未指定，该位置默认为tmpdir变量值）。从机需要这些文件来继续复制中断的LOAD DATA INFILE操作。

如果必须恢复MyISAM表，先使用REPAIR TABLE或**myisamchk -r**来恢复。99.9%的情况下该方法可以工作。如果**myisamchk**失败，试试下面的方法。请注意只有用--log-bin选项启动了MySQL从而启用二进制日志它才工作；参见5.11.3节，“二进制日志”。

1. 恢复原**mysqldump**备份，或二进制备份。

2. 执行下面的命令重新更新二进制日志：

3.
```
shell> mysqlbinlog hostname-bin.[0-9]* | mysql
```

在某些情况下，你可能只想要从某个位置重新运行某些二进制日志。(通常你想要从恢复备份的日期重新运行所有二进制日志，查询不正确时例外)。关

于**mysqlbinlog**工具和如何使用它的详细信息参见8.6节，“mysqlbinlog：用于处理二进制日志文件的实用工具”。

还可以对具体文件进行选择备份：

- 要想复制表，使用SELECT * INTO OUTFILE '*file_name*' FROM *tbl_name*。
- 要想重载表，使用LOAD DATA INFILE 'file_name' REPLACE ...并恢复。要避免复制记录，表必须有PRIMARY KEY或一个UNIQUE索引。当新记录复制唯一键值的旧记录时，REPLACE关键字可以将旧记录替换为新记录。

如果备份时遇到服务器性能问题，可以有帮助的一个策略是在从服务器而不是主服务器上建立复制并执行备份。参见6.1节，“复制介绍”。

如果使用Veritas文件系统，可以这样备份：

1. 从客户端程序执行FLUSH TABLES WITH READ LOCK。
2. 从另一个shell执行mount vxfs snapshot。
3. 从第一个客户端执行UNLOCK TABLES。
4. 从快照复制文件。
5. 卸载快照。

5.9.2. 示例用备份与恢复策略

5.9.2.1. 备份策略
5.9.2.2. 为恢复进行备份
5.9.2.3. 备份策略摘要

本节讨论进行备份的程序，在出现崩溃后，可以恢复数据：

- 操作系统崩溃
- 电源故障
- 文件系统崩溃
- 硬件问题(硬盘、母板等等)

该命令不包括**mysqldump**和**mysql**程序的--user和—password等选项。应包括必要的选项让MySQL服务器允许你连接它。

我们假定数据保存在MySQL的InnoDB存储引擎中，支持事务和自动崩溃恢复。我们假定崩溃时MySQL服务器带负载。如果不带负载，则不需要恢复。

出现操作系统崩溃或电源故障时，我们可以假定重启后硬盘上的MySQLs数据仍可用。由于崩溃，InnoDB数据文件中的数据可能不再保持一致性，但InnoDB读取它的日志并会查到挂起的提交的和未提交的事务清单，它们没有清空到数据文件中。InnoDB自动卷回未提交的事务，并清空到它的数据文件中。通过MySQL错误日志将该恢复过程相关信息传达给用户。下面的例子为日志摘录：

```
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
```

如果文件系统崩溃或出现硬件问题，我们可以假定重启后硬盘上的MySQLs数据不可用。这说明MySQL未能成功启动，因为一些硬盘数据块不再可读。在这种

情况下，需要重新格式化硬盘，安装一个新的，或纠正问题。然后需要从备份中恢复MySQL数据，这说明我们必须先做好备份。要想确保，应及时返回并设计备份策略。

5.9.2.1. 备份策略

我们都知道必须按计划定期进行备份。可以用几个工具完全备份(在某个时间点的数据快照)MySQL。例如，InnoDB Hot Backup为InnoDB数据文件提供在线非数据块物理备份，**mysqldump**提供在线逻辑备份。这里使用**mysqldump**。

假定我们在星期日下午1点进行了备份，此时负荷较低。下面的命令可以完全备份所有数据库中的所有InnoDB表：

```
shell> mysqldump --single-transaction --all-databases > backup_sunday_1_PM.sql
```

这是在线非块备份，不会干扰对表的读写。我们以前假定我们的表为InnoDB表，因此--single-transaction使用一致性地读，并且保证**mysqldump**所看见的数据不会更改。(其它客户端对InnoDB表进行的更改不会被**mysqldump**进程看见）。如果我们还有其它类型的表，我们必须假定在备份过程中它们不会更改。例如，对于mysql数据库中的MyISAM表，我们必须假定在备份过程中没有对MySQL账户进行管理更改。

mysqldump命令产生的.sql文件包含一系列SQL INSERT语句，可以用来重载转储的表。

需要进行完全备份，但有时不方便。会产生大的备份文件并需要花时间来生成。从某个角度，完全备份并不理想，因为每个成功的完全备份包括所有数据，甚至自从上一次完全备份以来没有更改的部分。完成了初使完全备份后，进行增量备份会更有效。这样备份文件要小得多，备份时间也较短。不利之处是，恢复时不能只重载完全备份来恢复数据。还必须要用增量备份来恢复增量更改。

要想进行增量备份，我们需要保存增量更改。应使用--log-bin选项启动MySQL服务器，以便更新数据时将这些更改保存到文件中。该选项启用二进制日志，因此服务器写将每个更新数据的SQL语句写入MySQL二进制日志。让我们看看用--log-bin选项启动的已经运行多日的MySQL服务器的数据目录。我们找到以下MySQL二进制日志文件：

```
-rw-rw---- 1 guilhem  guilhem   1277324 Nov 10 23:59 gbichot2-bin.000001
-rw-rw---- 1 guilhem  guilhem         4 Nov 10 23:59 gbichot2-bin.000002
-rw-rw---- 1 guilhem  guilhem        79 Nov 11 11:06 gbichot2-bin.000003
-rw-rw---- 1 guilhem  guilhem       508 Nov 11 11:08 gbichot2-bin.000004
-rw-rw---- 1 guilhem  guilhem 220047446 Nov 12 16:47 gbichot2-bin.000005
-rw-rw---- 1 guilhem  guilhem    998412 Nov 14 10:08 gbichot2-bin.000006
-rw-rw---- 1 guilhem  guilhem       361 Nov 14 10:07 gbichot2-bin.index
```

每次重启，MySQL服务器用序列中的下一个编号创建一个新的二进制日志文件。当服务器运行时，你还可以通过执行FLUSH LOGS SQL语句或**mysqladmin flush-logs**命令，告诉服务器关闭当前的二进制日志文件并创建一个新文件。**mysqldump**也有一个选项来清空日志。数据目录中的.index文件包含该目录下所有MySQL二进制日志的清单。该文件用于复制。

恢复时MySQL二进制日志很重要，因为它们是增量备份。如果进行完全备份时确保清空了日志，则后面创建的二进制日志文件包含了备份后的所有数据更改。让我们稍稍修改前面的**mysqldump**命令，让它在完全备份时能够清空 MySQL二进制日志，以便转储文件包含包含新的当前的二进制日志：

```
shell> mysqldump --single-transaction --flush-logs --master-data=2
           --all-databases > backup_sunday_1_PM.sql
```

执行该命令后，数据目录则包含新的二进制日志文件，gbichot2-bin.000007。结果.sql文件包含下列行：

```
-- Position to start replication or point-in-time 恢复时y from
-- CHANGE MASTER TO MASTER_LOG_FILE='gbichot2-bin.000007',MASTER_LOG_POS=4;
```

因为**mysqldump**命令可以执行完全备份，这些行表示两件事情：

- .sql文件包含所有写入gbichot2-bin.000007二进制日志文件或最新的文件之前的更改。
- 备份后所记录的所有数据更改不出现在.sql中，但出现在gbichot2-bin.000007二进制日志文件或最新的文件中。

在星期一下午1点，我们可以清空日志开始新的二进制日志文件来创建增量备份。例如，执行**mysqladmin flush-logs**命令创建gbichot2-bin.000008。星期日下午1点的完全备份和星期一下午1点之间的所有更改为文件gbichot2-bin.000007。该增量备份很重要，因此最好将它复制到安全的地方。(例如，备份到磁带或DVD上，或复制到另一台机器上）。在星期二下午1点，执行另一个**mysqladmin flush-logs**命令。星期一下午1点和星期二下午1点之间的所有所有更改为文件gbichot2-bin.000008(也应复制到某个安全的地方)。

MySQL二进制日志占据硬盘空间。要想释放空间，应随时清空。操作方法是删掉不再使用的二进制日志，例如进行完全备份时：

```
shell> mysqldump --single-transaction --flush-logs --master-data=2
           --all-databases --delete-master-logs > backup_sunday_1_PM.sql
```

注释：如果你的服务器为复制主服务器，用**mysqldump --delete-master-logs**删掉MySQL二进制日志很危险，因为从服务器可能还没有完全处理该二进制日志的内容。

PURGE MASTER LOGS语句的描述中解释了为什么在删掉MySQL二进制日志之前应进行确认。参见13.6.1.1节，“PURGE MASTER LOGS语法”。

5.9.2.2. 为恢复进行备份

现在假设在星期三上午8点出现了灾难性崩溃，需要使用备份文件进行恢复。恢复时，我们首先恢复最后的完全备份(从星期日下午1点开始)。完全备份文件是一系列SQL语句，因此恢复它很容易：

```
shell> mysql < backup_sunday_1_PM.sql
```

在该点，数据恢复到星期日下午1点的状态。要想恢复从那时起的更改，我们必须使用增量备份，也就是，gbichot2-bin.000007和gbichot2-bin.000008二进制日志文件。根据需要从备份处取过这些文件，然后按下述方式处理：

```
shell> mysqlbinlog gbichot2-bin.000007 gbichot2-bin.000008 | mysql
```

我们现在将数据恢复到星期二下午1点的状态，但是从该时刻到崩溃之间的数据仍然有丢失。要想恢复，我们需要MySQL服务器将MySQL二进制日志保存到安全的位置(RAID disks, SAN, ...)，应为与数据文件的保存位置不同的地方，保证这些日志不在毁坏的硬盘上。(也就是，我们可以用--log-bin选项启动服务器，指定一个其它物理设备上的与数据目录不同的位置。这样，即使包含该目录的设备丢失，日志也不会丢失）。如果我们执行了这些操作，我们手头上会有gbichot2-bin.000009文件，我们可以用它来恢复大部分最新的数据更改，而不会丢失到崩溃时的数据。

5.9.2.3. 备份策略摘要

出现操作系统崩溃或电源故障时，InnoDB自己可以完成所有数据恢复工作。但为了确保你可以睡好觉，应遵从下面的指导：

- 一定用--log-bin或甚至--log-bin=*log_name*选项运行MySQL服务器，其中日志文件名位于某个安全媒介上，不同于数据目录所在驱动器。如果你有这样的安全媒介，最好进行硬盘负载均衡(这样能够提高性能)。
- 定期进行完全备份，使用**mysqldump**命令进行在线非块备份。
- 用FLUSH LOGS或**mysqladmin flush-logs**清空日志进行定期增量备份。

5.9.3. 自动恢复

5.9.3.1. 指定恢复时间
5.9.3.2. 指定恢复位置

如果MySQL服务器启用了二进制日志，你可以使用**mysqlbinlog**工具来恢复从指定的时间点开始 (例如，从你最后一次备份)直到现在或另一个指定的时间点的数据。关于启用二进制日志的信息，参见5.11.3节，“二进制日志”。对于**mysqlbinlog**的详细信息，参见8.6节，“mysqlbinlog：用于处理二进制日志文件的实用工具”。

要想从二进制日志恢复数据，你需要知道当前二进制日志文件的路径和文件名。一般可以从选项文件(即my.cnf or my.ini，取决于你的系统)中找到路径。如果未包含在选项文件中，当服务器启动时，可以在命令行中以选项的形式给出。启用二进制日志的选项为--log-bin。要想确定当前的二进制日志文件的文件名，输入下面的MySQL语句：

```
SHOW BINLOG EVENTS \G
```

你还可以从命令行输入下面的内容：

```
mysql --user=root -pmy_pwd -e 'SHOW BINLOG EVENTS \G'
```

将密码*my_pwd*替换为服务器的*root*密码。

5.9.3.1. 指定恢复时间

对于MySQL 4.1.4，可以在**mysqlbinlog**语句中通过--start-date和--stop-date选项指定DATETIME格式的起止时间。举例说明，假设在今天上午10:00(今天是2005年4月20日)，执行SQL语句来删除一个大表。要想恢复表和数据，你可以恢复前晚上的备份，并输入：

```
mysqlbinlog --stop-date="2005-04-20 9:59:59" /var/log/mysql/bin.123456 \
     | mysql -u root -pmypwd
```

该命令将恢复截止到在--stop-date选项中以DATETIME格式给出的日期和时间的所有数据。如果你没有检测到几个小时后输入的错误的SQL语句，可能你想要恢复后面发生的活动。根据这些，你可以用起使日期和时间再次运行**mysqlbinlog**：

```
mysqlbinlog --start-date="2005-04-20 10:01:00" /var/log/mysql/bin.123456 \
     | mysql -u root -pmypwd \
```

在该行中，从上午10:01登录的SQL语句将运行。组合执行前夜的转储文件和**mysqlbinlog**的两行可以将所有数据恢复到上午10:00前一秒钟。你应检查日志以确保时间确切。下一节介绍如何实现。

5.9.3.2. 指定恢复位置

也可以不指定日期和时间，而使用**mysqlbinlog**的选项--start-position和--stop-position来指定日志位置。它们的作用与起止日选项相同，不同的是给出了从日志起的位置号。使用日志位置是更准确的恢复方法，特别是当由于破坏性SQL语句同时发生许多事务的时候。要想确定位置号，可以运行**mysqlbinlog**寻找执行了不期望的事务的时间范围，但应将结果重新指向文本文件以便进行检查。操作方法为：

```
mysqlbinlog --start-date="2005-04-20 9:55:00" --stop-date="2005-04-20 10:05:00" \
      /var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

该命令将在/tmp目录创建小的文本文件，将显示执行了错误的SQL语句时的SQL语句。你可以用文本编辑器打开该文件，寻找你不要想重复的语句。如果二进制日志中的位置号用于停止和继续恢复操作，应进行注释。用log_pos加一个数字来标记位置。使用位置号恢复了以前的备份文件后，你应从命令行输入下面内容：

```
mysqlbinlog --stop-position="368312" /var/log/mysql/bin.123456 \
    | mysql -u root -pmypwd
```

```
mysqlbinlog --start-position="368315" /var/log/mysql/bin.123456 \
    | mysql -u root -pmypwd \
```

上面的第1行将恢复到停止位置为止的所有事务。下一行将恢复从给定的起始位置直到二进制日志结束的所有事务。因为**mysqlbinlog**的输出包括每个SQL语句记录之前的SET TIMESTAMP语句，恢复的数据和相关MySQL日志将反应事务执行的原时间。

5.9.4. 表维护和崩溃恢复

后面几节讨论如何使用**myisamchk**来检查或维护MyISAM表(对应.MYI和.MYD文件的表)。

你可以使用**myisamchk**实用程序来获得有关你的数据库表的信息或检查、修复、优化他们。下列小节描述如何调用**myisamchk**（包括它的选项的描述），如何建立表的维护计划，以及如何使用**myisamchk**执行各种功能。

尽管用**myisamchk**修复表很安全，在修复(或任何可以大量更改表的维护操作)之前先进行备份也是很好的习惯

影响索引的**myisamchk**操作会使ULLTEXT索引用full-text参数重建，不再与MySQL服务器使用的值兼容。要想避免，请阅读5.9.5.1节，“用于myisamchk的一般选项”的说明。

在许多情况下，你会发现使用SQL语句实现MyISAM表的维护比执行**myisamchk**操作要容易地多：

- 要想检查或维护MyISAM表，使用CHECK TABLE或REPAIR TABLE。
- 要想优化MyISAM表，使用OPTIMIZE TABLE。
- 要想分析MyISAM表，使用ANALYZE TABLE。

可以直接这些语句，或使用**mysqlcheck**客户端程序，可以提供命令行接口。

这些语句比**myisamchk**有利的地方是服务器可以做任何工作。使用**myisamchk**，你必须确保服务器在同一时间不使用表。否则，**myisamchk**和服务器之间会出现不期望的相互干涉。

5.9.5. myisamchk：MyISAM表维护实用工具

5.9.5.1. 用于myisamchk的一般选项
5.9.5.2. 用于myisamchk的检查选项
5.9.5.3. myisamchk的修复选项
5.9.5.4. 用于myisamchk的其它选项
5.9.5.5. **myisamchk**内存使用
5.9.5.6. 将myisa**mchk**用于崩溃恢复
5.9.5.7. 如何检查MyISAM表的错误
5.9.5.8. 如何修复表
5.9.5.9. 表优化

可以使用**myisamchk**实用程序来获得有关数据库表的信息或检查、修复、优化他们。**myisamchk**适用MyISAM表(对应.MYI和.MYD文件的表)。

调用**myisamchk**的方法：

```
shell> myisamchk [options] tbl_name ...
```

*options*指定你想让**myisamchk**做什么。在后面描述它们。还可以通过调用**myisamchk --help**得到选项列表。

*tbl_name*是你想要检查或修复的数据库表。如果你不在数据库目录的某处运行**myisamchk**，你必须指定数据库目录的路径，因为**myisamchk**不知道你的数据库位于哪儿。实际上，**myisamchk**不在乎你正在操作的文件是否位于一个数据库目录；你可以将对应于数据库表的文件拷贝到别处并且在那里执行恢复操作。

如果你愿意，可以用**myisamchk**命令行命名几个表。还可以通过命名索引文件(用“ .MYI”后缀)来指定一个表。它允许你通过使用模式“*.MYI”指定在一个目录所有的表。例如，如果你在数据库目录，可以这样在目录下检查所有的MyISAM表：

```
shell> myisamchk *.MYI
```

如果你不在数据库目录下，可通过指定到目录的路径检查所有在那里的表：

```
shell> myisamchk /path/to/database_dir/*.MYI
```

你甚至可以通过为**MySQL**数据目录的路径指定一个通配符来检查所有的数据库中的所有表：

```
shell> myisamchk /path/to/datadir/*/*.MYI
```

推荐的快速检查所有MyISAM表的方式是：

```
shell> myisamchk --silent --fast /path/to/datadir/*/*.MYI
```

如果你想要检查所有MyISAM表并修复任何破坏的表，可以使用下面的命令：

```
shell> 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
```

该命令假定你有大于64MB的自由内存。关于用**myisamchk**分配内存的详细信息，参见5.9.5.5节，“**myisamchk**内存使用”。

当你运行**myisamchk**时，必须确保其它程序不使用表。否则，当你运行**myisamchk**时，会显示下面的错误消息：

```
warning: clients are using or haven't closed the table properly
```

这说明你正尝试检查正被另一个还没有关闭文件或已经终止而没有正确地关闭文件的程序(例如**mysqld**服务器)更新的表。

如果**mysqld**正在运行，你必须通过FLUSH TABLES强制清空仍然在内存中的任何表修改。当你运行**myisamchk**时，必须确保其它程序不使用表。避免该问题的最容易的方法是使用CHECK TABLE而不用**myisamchk**来检查表。

5.9.5.1. 用于**myisamchk**的一般选项

本节描述的选项可以用于用**myisamchk**执行的任何类型的表维护操作。本节后面的章节中描述的选项只适合具体操作，例如检查或修复表。

- --help，-?

显示帮助消息并退出。

- --debug=*debug_options*, -# *debug_options*

输出调试记录文件。*debug_options*字符串经常是'd:t:o,*filename*'。

- --silent，-s

沉默模式。仅当发生错误时写输出。你能使用-s两次(-ss)使**myisamchk**沉默。

- --verbose，-v

冗长模式。打印更多的信息。这能与-d和-e一起使用。为了更冗长，使用-v多次(-vv, -vvv)！

- --version, -V

显示版本信息并退出。

- --wait, -w

如果表被锁定，不是提示错误终止，而是在继续前等待到表被解锁。请注意如果用--skip-external-locking选项运行**mysqld**，只能用另一个**myisamchk**命令锁定表。

还可以通过--*var_name*=*value*选项设置下面的变量：

变量	默认值
decode_bits	9
ft_max_word_len	取决于版本
ft_min_word_len	4
ft_stopword_file	内建列表
key_buffer_size	523264
myisam_block_size	1024
read_buffer_size	262136
sort_buffer_size	2097144
sort_key_blocks	16
stats_method	nulls_unequal
write_buffer_size	262136

可以用**myisamchk --help**检查**myisamchk**变量及其 默认值：

当用排序键值修复键值时使用sort_buffer_size，使用--recover时这是很普通的情况。

当用--extend-check检查表或通过一行一行地将键值插入表中(如同普通插入)来修改键值时使用Key_buffer_size。在以下情况通过键值缓冲区进行修复：

- 使用--safe-recover。
- 当直接创建键值文件时，需要对键值排序的临时文件有两倍大。通常是当CHAR、VARCHAR、或TEXT列的键值较大的情况，因为排序操作在处理过程中需要保存全部键值。如果你有大量临时空间，可以通过排序强制使用**myisamchk**来修复，可以使用--sort-recover选项。

通过键值缓冲区的修复占用的硬盘空间比使用排序么少，但是要慢。

如果想要快速修复，将key_buffer_size和sort_buffer_size变量设置到大约可用内存的25%。可以将两个变量设置为较大的值，因为一个时间只使用一个变量。

myisam_block_size是用于索引块的内存大小。

stats_method影响当给定--analyze选项时，如何为索引统计搜集处理NULL值。它如同myisam_stats_method系统变量。详细信息参见5.3.3节，“服务器系统

变量”和7.4.7节，“MyISAM索引统计集合”的myisam_stats_method的描述。

ft_min_word_len和ft_max_word_len表示FULLTEXT索引的最小和最大字长。ft_stopword_file为停止字文件的文件名。需要在以下环境中对其进行设置。

如果你使用**myisamchk**来修改表索引(例如修复或分析)，使用最小和最大字长和停止字文件的 默认全文参数值（除非你另外指定）重建FULLTEXT索引。这样会导致查询失败。

出现这些问题是因为只有服务器知道这些参数。它们没有保存在MyISAM索引文件中。如果你修改了服务器中的最小或最大字长或停止字文件，要避免该问题，为用于**mysqld**的**myisamchk**指定相同的ft_min_word_len，ft_max_word_len和ft_stopword_file值。例如，如果你将最小字长设置为3,可以这样使用**myisamchk**来修复表：

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

要想确保**myisamchk**和服务器使用相同的全文参数值，可以将它们放入选项文件的[mysqld]和[myisamchk]小节：

```
[mysqld]
ft_min_word_len=3


[myisamchk]
ft_min_word_len=3
```

除了**myisamchk**，还可以使用REPAIR TABLE、ANALYZE TABLE、OPTIMIZE TABLE或ALTER TABLE。这些语句由服务器执行，知道要使用的正确的全文参数值。

5.9.5.2. 用于**myisamchk**的检查选项

myisamchk支持下面的表检查操作选项：

- --check, -c

检查表的错误。如果你不明确指定操作类型选项，这就是默认操作。

- --check-only-changed, -C

只检查上次检查后有变更的表。

- --extend-check, -e

非常仔细地检查表。如果表有许多索引将会相当慢。该选项只能用于极端情况。一般情况下，可以使用**myisamchk**或**myisamchk --medium-check**来确定表内是否有错误。

如果你使用了--extend-check并且有充分的内存，将key_buffer_size变量设置为较大的值可以使修复操作运行得更快。

- --fast，-F

只检查没有正确关闭的表。

- --force, -f

如果**myisamchk**发现表内有任何错误，则自动进行修复。维护类型与--repair或-r选项指定的相同。

- --information, -i

打印所检查表的统计信息。

- --medium-check, -m

比--extend-check更快速地进行检查。只能发现99.99%的错误，在大多数情况下就足够了。

- --read-only, -T

不要将表标记为已经检查。如果你使用**myisamchk**来检查正被其它应用程序使用而没有锁定的表很有用，例如当用--skip-external-locking选项运行时运行**mysqld**。

- --update-state, -U

将信息保存在.MYI文件中，来表示表检查的时间以及是否表崩溃了。该选项用来充分利用--check-only-changed选项，但如果**mysqld**服务器正使用表并且正用--skip-external-locking选项运行时不应使用该选项。

5.9.5.3. **myisamchk**的修复选项

myisamchk支持下面的表修复操作的选项：

- --backup, -B

将.MYD文件备份为file_name-time.BAK

- --character-sets-dir=*path*

字符集安装目录。参见5.10.1节，“数据和排序用字符集”。

- --correct-checksum

纠正表的校验和信息。

- --data-file-length=*len*, -D *len*

数据文件的最大长度(当重建数据文件且为“满”时)。

- --extend-check，-e

进行修复，试图从数据文件恢复每一行。一般情况会发现大量的垃圾行。不要使用该选项,除非你不顾后果。

- --force, -f

覆盖旧的中间文件(文件名类似*tbl_name*.TMD)，而不是中断。

- --keys-used=*val*, -k *val*

对于**myisamchk**，该选项值为位值，说明要更新的索引。选项值的每一个二进制位对应表的一个索引，其中第一个索引对应位0。选项值0禁用对所有索引的更新，可以保证快速插入。通过**myisamchk -r**可以重新激活被禁用的索引。

- --no-symlinks, -l

不跟随符号连接。通常**myisamchk**修复一个符号连接所指的表。在MySQL 4.0中该选项不存在，因为从4.0开始的版本在修复过程中不移除符号链接。

- --parallel-recover, -p

与-r和-n的用法相同，但使用不同的线程并行创建所有键。这是*alpha*代码。自己承担风险！

- --quick，-q

不修改数据文件，快速进行修复。出现复制键时，你可以两次指定该项以强制**myisamchk**修改原数据文件。

- --recover, -r

可以修复几乎所有一切问题，除非唯一的键不唯一时(对于MyISAM表，这是非常不可能的情况)。如果你想要恢复表，这是首先要尝试的选项。如果**myisamchk**报告表不能用-r恢复，则只能尝试-o。在不太可能的情况下-r失败，数据文件保持完好）。

如果你有大量内存，你应增加sort_buffer_size的值。

- --safe-recover, -o

使用一个老的恢复方法读取，按顺序读取所有行，并根据找到的行更新所有索引树。这比-r慢些，但是能处理-r不能处理的情况。该恢复方法使用的硬盘空间比-r少。一般情况，你应首先用-r维修，如果-r失败则用-o。

如果你有大量内存，你应增加sort_buffer_size的值。

- (*OBSOLETE*) --set-character-set=*name*

在MySQL 5.1中不使用。参见--set-collation。

- --set-collation=*name*

更改用来排序表索引的校对规则。校对规则名的第一部分包含字符集名。

- --sort-recover, -n

强制**myisamchk**通过排序来解析键值，即使临时文件将可能很大。

- --tmpdir=*path*, -t *path*

用于保存临时文件的目录的路径。如果未设置，**myisamchk**使用TMPDIR环境变量的值。tmpdir可以设置为一系列目录路径，用于成功地以round-robin模式创建临时文件。在Unix中，目录名之间的间隔字符为冒号(‘:’)，在Windows、NetWare和OS/2中为分号 (‘；’)。

- --unpack，-u

将用**myisampack**打包的表解包。

5.9.5.4. 用于**myisamchk**的其它选项

myisamchk支持以下表检查和修复之外的其它操作的选项：

- --analyze，-a

分析键值的分布。这通过让联结优化器更好地选择表应该以什么次序联结和应该使用哪个键来改进联结性能。要想获取分布相关信息，使用**myisamchk --description --verbose** *tbl_name*命令或SHOW KEYS FROM *tbl_name*语句。

- --description, -d

打印出关于表的描述性信息。

- --set-auto-increment[=*value*], -A[*value*]

强制从给定值开始的新记录使用AUTO_INCREMENT编号(或如果已经有AUTO_INCREMENT值大小的记录，应使用更高值)。如果未指定*value*，新记录的AUTO_INCREMENT编号应使用当前表的最大值加上1。

- --sort-index, -S

以从高到低的顺序排序索引树块。这将优化搜寻并且将使按键值的表扫描更快。

- --sort-records=*N*, -R *N*

根据一个具体索引排序记录。这使你的数据更局部化并且可以加快在该键上的SELECT和ORDER BY的范围搜索。（第一次做排序可能很慢！）为了找出一张表的索引编号，使用SHOW INDEX，它以**myisamchk**看见他们的相同顺序显示一张表的索引。索引从1开始编号。

如果键没有打包(PACK_KEYS=0)，它们的长度相同，因此当**myisamchk** 排序并移动记录时，只覆盖索引中的记录偏移量。如果键已经打包(PACK_KEYS=1)，**myisamchk**必须先解开打包的键块，然后重新创建索引并再次将键块打包。(在这种情况下，重新创建索引比更新每个索引的偏移量要快）。

5.9.5.5. **myisamchk**内存使用

当你运行**myisamchk**时内存分配重要.MYI**samchk**使用的内存大小不能超过用-O选项指定的。如果你想对每一个大表使用**myisamchk**，你必须首先确定你想使用多少内存。修复时可以使用的 默认值只有3MB。使用更大的内存，可以让**myisamchk**工作得更快一些。例如，如果有大于32MB的RAM，可以使用如下所示选项(除了你可以指定的其它选项)：

```
shell> myisamchk -O sort=16M -O key=16M -O read=1M -O write=1M ...
```

对于大多数情况，使用-O sort=16M应该足够了。

应记住**myisamchk**使用TMPDIR中的临时文件。如果TMPDIR指向内存文件系统，你可能很容易得到内存溢出的错误。如果发生，设定TMPDIR指向有更多空间的文件系统目录并且重启**myisamchk**。

修复时**myisamchk**也需要大量硬盘空间：

- 将数据文件大小扩大一倍(原文件和复制文件)。如果你用--quick修复则不需要该空间；在这种情况下，只重新创建了索引文件。在文件系统上需要的空间与原数据文件相同！(创建的复制文件位于原文件所在目录）。
- 代替旧索引文件的新索引文件所占空间。修复工作一开始，就对旧索引文件进行了删减，因此你通常会忽略该空间。在文件系统上需要的该空间与原数据文件相同！
- 当使用--recover或---sort-recover(但不使用--safe-recover)时，需要排序缓冲区空间。需要的空间为：
- `(largest_key + row_pointer_length) * number_of_rows * 2`

可以用**myisamchk -dv** *tbl_name*检查键值和row_pointer_length的长度。在临时目录分配该空间(用TMPDIR或--tmpdir=*path*指定)。

如果在修复过程中出现硬盘空间问题，可以试试用--safe-recover代替--recover。

5.9.5.6. 将**myisamchk**用于崩溃恢复

如果用--skip-external-locking运行**mysqld**(在某些系统上为 默认设置，例如Linux)，当**mysqld**使用某个表时，你不能可靠地使用**myisamchk**来检查相同的表。当你运行**myisamchk**时如果可以确保没有人在通过**mysqld**访问表，在开始检查表前，你只需要运行**mysqladmin flush-tables**。如果你不能保证，则你检查表时你必须停止**mysqld**。如果**mysqld**更新表时运行**myisamchk**，你可能会获得表被破坏的警告，即使事实并非如此。

如果不使用--skip-external-locking，可以随时使用**myisamchk**来检查表。当检查表时，所有尝试更新表的客户端将等待，直到**myisamchk**准备好可以继续。

如果使用**myisamchk**来修复或优化表，必须确保**mysqld**服务器没有在使用该表(如果你正使用--skip-external-locking选项也适用)。如果不关闭**mysqld**，在运行**myisamchk**之前至少应执行**mysqladmin flush-tables**。如果服务器和**myisamchk**同时访问表，表可能会被破坏。

本节描述如何检查和处理MySQL数据库中的数据破坏。如果表经常被破坏，你应尽力找到原因。参见A.4.2节，“如果MySQL依然崩溃，应作些什么”。

关于MyISAM表怎样会被破坏的解释，参见15.1.4节，“MyISAM表方面的问题”。

在执行崩溃恢复时，理解在一个数据库中的每一个*MyISAM*表tbl_name对应的在数据库目录中的3个文件是很重要的：

文件	目的
tbl_name.frm	定义(格式)文件
tbl_name.MYD	数据文件
tbl_name.MYI	索引文件

这3类文件的每一类都可能遭受不同形式的损坏，但是问题最常发生在数据文件和索引文件。

myisamchk通过一行一行地创建一个".MYD"数据文件的副本来工作，它通过删除旧的".MYD 文件并且重命名新文件到原来的文件名结束修复阶段。如果你使用--quick，**myisamchk**不创建一个临时".MYD"文件，只是假定".MYD"文件是正确的并且仅创建一个新的索引文件，不接触".MYD"文件，这是安全的，因为**myisamchk**自动检测".MYD"文件是否损坏并且在这种情况下，放弃修复。你也可以给**myisamchk**两个--quick选项。在这种情况下，**myisamchk**不会在一些错误上(象重复键)放弃，相反试图通过修改".MYD"文件解决它们。通常，只有在太少的空闲磁盘空间上实施正常修复，使用两个--quick选项时才有用。在这种情况下，你至少应该在运行**myisamchk**前做进行备份。

5.9.5.7. 如何检查MyISAM表的错误

要想检查MyISAM表，应使用下面的命令：

· myisamchk *tbl_name*

这样能找出99.99%的错误。它不能找出的是仅仅涉及数据文件的损坏(这很不常见)。如果想要检查一张表，通常应该没有选项地运行myisamchk或用-s或--silent选项的任何一个。

· myisamchk -m *tbl_name*

这样能找出99.99%的错误。它首先检查所有索引条目的错误并通读所有行。它还计算行内所有键值的校验和，并确认校验和与索引树内键的校验和相匹配。

· myisamchk -e *tbl_name*

可以完全彻底地检查数据(-e意思是"扩展检查")。它对每一行做每个键的读检查以证实它们确实指向正确的行。这在一个有很多键的大表上可能花很长时间。**myisamchk**通常将在它发现第一个错误以后停止。如果你想要获得更多的信息，可以增加--verbose(-v)选项。这使得**myisamchk**继续一直到最多20个错误。

· myisamchk -e -i *tbl_name*

象前面的命令一样，但是-i选项告诉**myisamchk**还打印出一些统计信息。

在一般使用中，一个简单的**myisamchk**（没有除表名以外的参数）就足够检查表了。

5.9.5.8. 如何修复表

本节描述如何对MyISAM表使用**myisamchk**(扩展名.MYI和.MYD)。

你还可以(并且应该，如果可能)使用CHECK TABLE和REPAIR TABLE语句来检查和修复MyISAM表。参见13.5.2.3节，"CHECK TABLE语法"和13.5.2.6节，"REPAIR TABLE语法"。

一张损坏的表的症状通常是查询意外中断并且能看到下述错误：

- "*tbl_name*.frm"被锁定不能更改。
- 不能找到文件"*tbl_name*.MYI"（Errcode：*nnn*）。
- 文件意外结束。
- 记录文件被毁坏。
- 从表处理器得到错误*nnn*。

要想得到错误相关的详细信息，你可以运行**perror** *nnn*，其中*nnn*为错误编号。下面的示例显示了如何使用**perror**来找到最常用错误编号（用表的方式指出问题）的含义：

```
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
```

请注意错误135(记录文件中没有更多的空间)和错误136(索引文件中没有更多的空间)不是可以通过简单修复可以修复的错误。在这种情况下，必须使用ALTER TABLE来增加MAX_ROWS和AVG_ROW_LENGTH表选项值：

```
ALTER TABLE tbl_name MAX_ROWS=xxx AVG_ROW_LENGTH=yyy;
```

如果你不知道当前的表的选项值，使用SHOW CREATE TABLE或DESCRIBE来查询。

对于其它的错误，你必须修复表。**myisamchk**通常可以检测和修复大多数问题。

修复过程包括四个阶段，此处将进行描述。开始修复前，应进入数据库目录并检查表文件的许可。在Unix中，确保它们对于运行**mysqld**的用户可读(你也应可读，因为你需要访问检查的文件)。如果你需要修改文件，你还必须拥有写访问权限。

用**myisamchk**修复表的选项的描述参见5.9.5节，"myisamchk：MyISAM表维护实用工具"的前几节。

下面几节列出了上述命令失败或你想要使用**myisamchk**提供的扩展特性等情况的例子。

如果你要通过命令行来修复表，必须首先停止**mysqld**服务器。请注意当你在远程服务器上运行**mysqladmin shutdown**时，**mysqladmin**返回后，**mysqld**服务器将仍然运行一会儿，直到停止所有查询并将所有键清空到硬盘上。

阶段**1**：检查你的表

如果你有很多时间，运行**myisamchk *.MYI**或**myisamchk -e *.MYI**。使用-s（沉默）选项禁止不必要的信息。

如果**mysqld**服务器处于宕机状态，应使用--update-state选项来告诉**myisamchk**将表标记为'检查过的'。

你必须只修复那些**myisamchk**报告有错误的表。对这样的表，继续到阶段2。

如果在检查时，你得到奇怪的错误(例如out of memory错误)，或如果**myisamchk**崩溃，到阶段3。

阶段**2**：简单安全的修复

注释：如果想更快地进行修复，当运行**myisamchk**时，你应将sort_buffer_size和Key_buffer_size变量的值设置为可用内存的大约25%。

首先，试试**myisamchk -r -q** *tbl_name*(-r -q意味着"快速恢复模式")。这将试图不接触数据文件来修复索引文件。如果数据文件包含它应有的一切内容和指向数据文件内正确地点的删除连接，这应该管用并且表可被修复。开始修复下一张表。否则，执行下列过程：

1. 在继续前对数据文件进行备份。
2. 使用**myisamchk -r** *tbl_name*(-r意味着"恢复模式")。这将从数据文件中删除不正确的记录和已被删除的记录并重建索引文件。
3. 如果前面的步骤失败，使用**myisamchk --safe-recover** *tbl_name*。安全恢复模式使用一个老的恢复方法，处理常规恢复模式不行的少数情况(但是更慢)。

如果在修复时，你得到奇怪的错误(例如out of memory错误)，或如果**myisamchk**崩溃，到阶段3。

阶段**3**：困难的修复

只有在索引文件的第一个16K块被破坏，或包含不正确的信息，或如果索引文件丢失，你才应该到这个阶段。在这种情况下，需要创建一个新的索引文件。按如下步骤操做：

1. 把数据文件移到安全的地方。
2. 使用表描述文件创建新的(空)数据文件和索引文件：
3. shell> **mysql** *db_name*
4. mysql> **SET AUTOCOMMIT=1;**
5. mysql> **TRUNCATE TABLE** *tbl_name***;**
6. mysql> **quit**

如果你的MySQL版本没有TRUNCATE TABLE，则使用DELETE FROM *tbl_name*。

7. 将老的数据文件拷贝到新创建的数据文件之中。（不要只是将老文件移回新文件之中；你要保留一个副本以防某些东西出错。）

回到阶段2。现在**myisamchk -r -q**应该工作了。（这不应该是一个无限循环）。

你还可以使用REPAIR TABLE *tbl_name* USE_FRM，将自动执行整个程序。

阶段**4**：非常困难的修复

只有.frm描述文件也破坏了，你才应该到达这个阶段。这应该从未发生过，因为在表被创建以后，描述文件就不再改变了。

1. 从一个备份恢复描述文件然后回到阶段3。你也可以恢复索引文件然后回到阶段2。对后者，你应该用**myisamchk -r**启动。
2. 如果你没有进行备份但是确切地知道表是怎样创建的，在另一个数据库中创建表的一个拷贝。删除新的数据文件，然后从其他数据库将描述文件和索引文件移到破坏的数据库中。这样提供了新的描述和索引文件，但是让.MYD数据文件独自留下来了。回到阶段2并且尝试重建索引文件。

5.9.5.9. 表优化

为了组合碎片记录并且消除由于删除或更新记录而浪费的空间，以恢复模式运行**myisamchk**：

```
shell> myisamchk -r tbl_name
```

你可以用SQL的OPTIMIZE TABLE语句使用的相同方式来优化表，OPTIMIZE TABLE可以修复表并对键值进行分析，并且可以对索引树进行排序以便更快地查找键值。实用程序和服务器之间不可能交互操作，因为当你使用OPTIMIZE TABLE时，服务器做所有的工作。参见13.5.2.5节，"OPTIMIZE TABLE语法"。

myisamchk还有很多其它可用来提高表的性能的选项：

- -S, --sort-index
- -R *index_num*, --sort-records=*index_num*
- -a, --analyze

关于这些选项的完整的描述，参见5.9.5节，"myisamchk：MyISAM表维护实用工具"。

5.9.6. 建立表维护计划

定期对表进行检查而非等到问题出现后再检查数据库表是一个好主意。检查和修复MyISAM表的一个方式是使用CHECK TABLE和REPAIR TABLE语句。参见13.5.2.3节，“CHECK TABLE语法”和13.5.2.6节，“REPAIR TABLE语法”。

检查表的另一个方法是使用**myisamchk**。为维护目的，可以使用**myisamchk -s**检查表。-s选项（简称--silent）使**myisamchk**以沉默模式运行，只有当错误出现时才打印消息。

在服务器启动时检查表是一个好主意。例如，无论何时机器在更新当中重新启动了，你通常需要检查所有可能受影响的表。（即“预期的破坏了的表”）。要想自动检查MyISAM表，用--myisam-recover选项启动服务器。

一个更好的测试将是检查最后修改时间比“.pid”文件新的表。

你还应该在正常系统操作期间定期检查表。在MySQL AB，我们运行一个**cron**任务，每周一次检查所有重要的表，使用“crontab”文件中这样的行：

```
35 0 * * 0 /path/to/myisamchk --fast --silent /path/to/datadir/*/*.MYI
```

可以打印损坏的表的信息，以便我们在需要时能够检验并且修复它们。

多年了我们还没有发现(的确是真的)都没有任何意外损坏的表时(由于除硬件故障外的其它原因造成损坏的表)，每周一次对我们是足够了。

我们建议现在开始，你对所有最后24小时内被更新了的表每晚都执行**myisamchk -s**，直到你变得象我们那样信任**MySQL**。

一般情况，MySQL表很少需要维护。如果你用动态大小的行更改MyISAM表(含VARCHAR、BLOB或TEXT列的表)或有删除了许多行的表，你可能想要不时地（每月一次）整理/组合表的空间。

可以对有问题的表执行OPTIMIZE TABLE来优化。或者是，如果可以停一会**mysqld**服务器，进入数据目录，当服务器停止时使用该命令：

shell> **myisamchk -r -s --sort-index -O sort_buffer_size=16M */*.MYI**

5.9.7. 获取关于表的信息

为了获得关于一个表的描述或统计，使用下面的命令：

- **myisamchk -d** *tbl_name*

以“描述模式”运行**myisamchk**，生成表的描述。如果用--skip-external-locking选项启动**MySQL**服务器，**myisamchk**可以报告运行的表被更新的错误。然而，既然在描述模式中**myisamchk**不更改表，没有任何破坏数据的风险。

- **myisamchk -d -v** *tbl_name*

为了生成更多关于**myisamchk**正在做什么的信息，加上-v告诉它以冗长模式运行。

- **myisamchk -eis** *tbl_name*

仅显示表的最重要的信息。因为必须读取整个表，该操作很慢。

- **myisamchk -eiv** *tbl_name*

这类似 -eis，只是告诉你正在做什么。

下面为这些命令的输出示例。它们基于含这些数据和索引文件大小的表：

```
-rw-rw-r--   1 monty    tcx     317235748 Jan 12 17:30 company.MYD
-rw-rw-r--   1 davida   tcx      96482304 Jan 12 18:35 company.MYM
```

myisamchk -d输出示例：

```
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 输出示例:

```
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
Datafile pointer (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 输出示例:

```
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 输出示例:

```
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
```

下面解释**myisamchk**产生的信息的类型。“keyfile”是索引文件。“记录”和“行”是同义词。

- MyISAM file

ISAM(索引)文件名。

- File-version

ISAM格式的版本。当前总是2。

· Creation time

数据文件创建的时间。

· Recover time

索引/数据文件上次被重建的时间。

· Data records

在表中有多少记录。

· Deleted blocks

有多少删除的块仍然保留着空间。你可以优化表以使这个空间减到最小。参见第7章：优化。

· Datafile parts

对动态记录格式，这指出有多少数据块。对于一个没有碎片的优化过的表，这与Data records相同。

· Deleted data

不能回收的删除数据有多少字节。你可以优化表以使这个空间减到最小。参见第7章：优化。

· Datafile pointer

数据文件指针的大小，以字节计。它通常是2、3、4或5个字节。大多数表用2个字节管理，但是目前这还不能从**MySQL**控制。对固定表，这是一个记录地址。对动态表，这是一个字节地址。

· Keyfile pointer

索引文件指针的大小，以字节计。它通常是1、2或3个字节。大多数表用 2 个字节管理，但是它自动由**MySQL**计算。它总是一个块地址。

· Max datafile length

表的数据文件(.MYD文件)能够有多长，以字节计。

· Max keyfile length

表的键值文件(.MYI文件)能够有多长，以字节计。

· Recordlength

每个记录占多少空间，以字节计。

· Record format

用于存储表行的格式。上面的例子使用Fixed length。其他可能的值是Compressed和Packed。

· table description

在表中所有键值的列表。对每个键，给出一些底层的信息：

o Key

该键的编号。

o Start

该索引部分从记录的哪里开始。

o Len

该索引部分是多长。对于紧凑的数字，这应该总是列的全长。对字符串，它可以比索引的列的全长短些，因为你可能会索引到字符串列的前缀。

o Index

unique或multip（multiple)。表明一个值是否能在该索引中存在多次。

o Type

该索引部分有什么数据类型。这是一个packed、stripped或empty选项的ISAM数据类型。

o Root

根索引块的地址。

o Blocksize

1024 MySQL

每个索引块的大小。默认是 ，但是从源码构建 时，该值可以在编译时改变。

o Rec/key

这是由优化器使用的统计值。它告诉对该键的每个值有多少条记录。唯一键总是有一个1值。在一个表被装载后(或变更很大)，可以用**myisamchk -a**更新。如果根本没被更新，给定一个30的默认值。

在上面例子的表中，第9个键有两个table description行。者说明它是有2个部分的多部键。

· Keyblocks used

键块使用的百分比是什么。当在例子中使用的表刚刚用**myisamchk**重新组织时，该值非常高(很接近理论上的最大值)。

· Packed

MySQL试图用一个通用后缀压缩键。这只能被用于CHAR/VARCHAR/DECIMAL列的键。对于左部分类似的长字符串，能显著地减少使用空间。在上面的第3个例子中，第4个键是10个字符长，可以减少60%的空间。

· Max levels

对于该键的B树有多深。有长键的大表有较高的值。

· Records

表中有多少行。

· M.recordlength

平均记录长度。对于有定长记录的表，这是准确的记录长度，因为所有记录的长度相同。

· Packed

MySQL从字符串的结尾去掉空格。Packed值表明这样做达到的节约的百分比。

· Recordspace used

数据文件被使用的百分比。

· Empty space

数据文件未被使用的百分比。

· Blocks/Record

每个记录的平均块数(即，一个碎片记录由多少个连接组成)。对固定格式表，这总是1。该值应该尽可能保持接近1.0。如果它变得太大，你可以重新组织表。参见第7章：优化。

· Recordblocks

多少块(链接)被使用。对固定格式，它与记录的个数相同。

· Deleteblocks

多少块(链接)被删除。

· Recorddata

在数据文件中使用了多少字节。

· Deleted data

在数据文件中多少字节被删除(未使用)。

· Lost space

如果一个记录被更新为更短的长度，就损失了一些空间。这是所有这样的损失之和，以字节计。

· Linkdata

当使用动态表格式，记录碎片用指针连接(每个4 ～ 7字节)。 Linkdata指这样的指针使用的内存量之和。

如果一张表已经用**myisampack**压缩了，**myisamchk -d**打印每个表列的附加信息。对于它的一个例子及其含义的描述，参见8.2节，“myisampack：生成压缩、只读MyISAM表”。

5.10. MySQL本地化和国际应用

5.10.1. 数据和排序用字符集
5.10.2. 设置错误消息语言
5.10.3. 添加新的字符集
5.10.4. 字符定义数组

5.10.5. 字符串比较支持
5.10.6. 多字节字符支持
5.10.7. 字符集问题
5.10.8. MySQL服务器时区支持

本节描述如何配置服务器来使用不同的字符集。还讨论如何设置服务器的时区并启用各个连接的时区支持。

5.10.1. 数据和排序用字符集

5.10.1.1. 使用德国字符集

默认情况下，MySQL使用cp1252(Latin1)字符集根据Swedish/Finnish规则进行排序。这些 默认值适合美国和西欧大部分国家。

所有MySQL二进制分发版用--with-extra-charsets=complex编译而成。可以在所有标准程序中添加代码，使它们可以处理latin1和所有多字节二进制字符集。其它字符集根据需要从字符集定义文件载入。

字符集确定在名称中使用什么字符。它还确定如何用SELECT语句的ORDER BY和GROUP BY子句对字符串进行排序。

还可以在启动服务器时用--default-character-set选项更改字符集。字符集可以用--with-charset=*charset*和--with-extra-charsets=*list-of-charsets* | complex | all | none选项来**configure**，字符集配置文件列于SHAREDIR/charsets/Index。参见2.8.2节，“典型配置选项”。

还可以在你启动服务器时用--default-collation选项更改字符集 校对规则。校对规则必须是默认字符集的合法校对规则。(使用SHOW COLLATION语句来确定每个字符集使用哪个校对规则）。 参见2.8.2节，“典型配置选项”。

如果在运行MySQL时更改字符集，还会更改排序顺序。结果是你必须对所有表运行**myisamchk -r -q --set-character-set=***charset* ，或正确地对索引进行排序。

当客户端连接MySQL服务器时，服务器告诉客户端服务器的默认字符集是什么。客户端切换到该字符集进行连接。

当转义SQL查询的字符串时，你应使用mysql_real_escape_string()。mysql_real_escape_string()等价于旧的mysql_escape_string()函数，不同的是它使用MYSQL连接句柄作为第一个参数，以便转义字符时可以使用相应的字符集。

如果客户端的编译路径不是服务器的安装目录，并且配置MySQL时没有包括MySQL二进制中的所有字符集，如果服务器运行时需要使用客户端设置的字符集之外的其它字符集，你必须告诉客户端从哪里找到更多的字符集。

可以指定--character-sets-dir选项来表示动态MySQL字符集所保存目录的路径。例如，可以将下面的行放入选项文件中：

```
[client]
character-sets-dir=/usr/local/mysql/share/mysql/charsets
```

你可以强制客户端使用专用字符集：

```
[client]
default-character-set=charset
```

但是一般情况不需要。

5.10.1.1. 使用德国字符集

在MySQL 5.1中,分别指定字符集和 校对规则。这说明如果你想使用German排序顺序，你应选择latin1 字符集和latin1_german1_ci或latin1_german2_ci校对规则。例如，要用latin1_german1_ci校对规则启动服务器，应使用--character-set-server=latin1和--collation-server=latin1_german1_ci选项。

关于这两种校对规则的不同之处，参见10.10.2节，“西欧字符集”。

5.10.2. 设置错误消息语言

默认情况下，**mysqld**用英语给出错误消息，但也可以用以下语言显示：Czech、Danish、Dutch、Estonian、French、German、Greek、Hungarian、Italian、Japanese、Korean、Norwegian、Norwegian-ny、Polish、Portuguese、Romanian、Russian、Slovak、Spanish或 Swedish。

要想在启动**mysqld**后用具体语言显示错误消息，使用--language或-L选项。选项值可以为语言名称或错误消息文件的全路径。例如：

```
shell> mysqld --language=swedish
```

或：

```
shell> mysqld --language=/usr/local/share/swedish
```

语言名应为小写。

语言文件位于(默认情况下)MySQL基本目录的share/*LANGUAGE*目录下。

要想更改错误消息文件，应编辑errmsg.txt文件，然后执行下面的命令以生成errmsg.sys文件：

```
shell> comp_err errmsg.txt errmsg.sys
```

如果你升级到新版本的MySQL，记住使用新的errmsg.txt文件来重新更改。

5.10.3. 添加新的字符集

本节讨论在MySQL中添加新字符集的程序。你必须有一个MySQL源码分发版。

要选择正确的程序，先确定字符集是简单字符集还是复杂字符集：

- 如果字符集不需要使用特殊字符串校对规则程序进行排序，并且不需要多字节字符支持，则为简单字符集。
- 如果需要上述某个特性，则为复杂字符集。

例如，latin1和danish为简单字符集，而big5和czech为复杂字符集。

在下面的程序中，字符集名用*MYSET*表示。

对于简单字符集，应：

1. 在sql/share/charsets/Index文件最后添加*MYSET*。并指定唯一的编号。
2. 创建文件sql/share/charsets/*MYSET*.conf。(你可以使用sql/share/charsets/latin1.conf的备份文件作为该文件的基础）。

该文件的语法很简单：

- 注释从'#'字符开始，一直到该行末尾。
- 各字之间用任意数量的空格间隔开。
- 定义字符集时，每个字必须为十六进制格式的数字。
- ctype数组占据前257个字。to_lower[]、to_upper[]和sort_order[]数组依次占据256个字。

参见5.10.4节，"字符定义数组"。

3. 将字符集名添加到configurE.in的CHARSETS_AVAILABLE和COMPILED_CHARSETS列。
4. 重新配置、编译并测试。

对于复杂字符集，应：

1. 在MySQL源码分发版中创建文件strings/ctype-*MYSET*.c。
2. 在sql/share/charsets/Index文件最后添加*MYSET*。并指定唯一的编号。
3. 看看已有的ctype-*.c文件(例如strings/ctype-big5.c)，看看需要定义什么。请注意文件中的数组名必须为ctype_*MYSET*、to_lower_*MYSET*等等。对应简单字符集的数组。参见5.10.4节，"字符定义数组"。
4. 在文件顶部，添加注释：

```
5.          /*
6.          * This comment is parsed by configure to create ctype.c,
7.          * so don't change it unless you know what you are doing.
8.          *
9.          * .configure. number_MYSET=MYNUMBER
10.       * .configure. strxfrm_multiply_MYSET=N
11.       * .configure. mbmaxlen_MYSET=N
12.       */
```

configure程序使用该注释自动将字符集包括进MySQL库中。

在下面章节中解释strxfrm_multiply和mbmaxlen 行。只有需要字符串比较函数或多字节字符集函数时，才需要单独将它们包括进来。

13.然后你应创建下面的函数：

- my_strncoll_*MYSET*()
- my_strcoll_*MYSET*()
- my_strxfrm_*MYSET*()
- my_like_range_*MYSET*()

参见5.10.5节，"字符串比较支持"

14.将字符集名添加到configurE.in的CHARSETS_AVAILABLE和COMPILED_CHARSETS列。

15.重新配置、编译并测试。

sql/share/charsets/README文件中包括详细的说明。

如果你想要MySQL分发中的字符集，请向MySQL内部邮件系统发邮件。参见1.7.1.1节，"The MySQL邮件列表"。

5.10.4. 字符定义数组

to_lower[]和to_upper[]是简单数组，含有小写和大写字符，对应字符集的每个成员。例如：

```
to_lower['A'] should contain 'a'
to_upper['a'] should contain 'A'
```

sort_order[]是一个映射，表示如何排列字符的顺序，以便进行比较和排序。通常(但非对于所有字符集)与to_upper[]相同，说明排序对大小写敏感。MySQL排序字符基于sort_order[]元素的值。对于更加复杂的排序规则，参见5.10.5节，“字符串比较支持”的字符串 校对规则讨论。

ctype[]是一个位数组，每个字符为一个元素。(请注意字符值索引to_lower[]、to_upper[]和sort_order[]，但用字符值+ 1索引ctype[]。这是传统的转换方法，能够处理EOF）。

m_ctype.h中有下面的位掩码定义：

```
#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 */
```

每个字符的ctype[]条目应为相应的描述字符的位掩码值的联合。例如，'A'是大写字符(_U)以及十六进制整数(_X)，因此ctype['A'+1]应包含 值：

_U + _X = 01 + 0200 = 0201

5.10.5. 字符串比较支持

如果语言的排序规则比较复杂，不能用简单sort_order[]表来处理，需要使用字符串比较函数。

最好的文档是已有字符集。以big5、czech、gbk、sjis和tis160字符集作为例子。

你必须在文件顶部的特殊注释处指定strxfrm_multiply_*MYSET*=*N*值。*N*应设置为字符串在my_strxfrm_*MYSET*过程中可能增长的最大比例(必须为正整数)。

5.10.6. 多字节字符支持

如果你想要添加包括多字节字符的新字符集支持，需要使用多字节字符函数。

最好的文档是已有字符集。以euc_kr、gb2312,gbk、sjis和ujis字符集作为例子。这些字符集位于strings目录的ctype-*charset*.c文件中。

必须在文件顶部的特殊注释处指定mbmaxlen_*MYSET*=*N*值。*N*应设置为字符集内最长字符的字节数。

5.10.7. 字符集问题

如果你想要使用没有编译为二进制的字符集，可能会遇到下面的问题：

- 你的程序的字符集保存路径不正确。(默认为/usr/local/mysql/share/mysql/charsets)。可以在运行有问题的程序时通过--character-sets-dir选项来修复。
- 字符集为多字节字符集，不能动态载入。在这种情况下，你必须重新编译程序，以支持字符集。
- 字符集为动态字符集，但你没有对应的配置文件。在这种情况下，你应从新MySQL分发安装该字符集的配置文件。
- 如果Index文件没有包含字符集名，程序将显示下面的错误消息：

```
ERROR 1105: File '/usr/local/share/mysql/charsets/?.conf'
not found (Errcode: 2)
```

在这种情况下，你应获得新Index文件或在当前文件中手动添加字符集。

对于MyISAM表，可以用**myisamchk -dvv** *tbl_name*检查表的字符集名和编号。

5.10.8. MySQL服务器时区支持

MySQL服务器有几个时区设置：

- 系统时区。服务器启动时便试图确定主机的时区，用它来设置system_time_zone系统变量。
- 服务器当前的时区。全局系统变量time_zone表示服务器当前使用的时区。初使值为'SYSTEM'，说明服务器时区与系统时区相同。可以用--default-time-zone=*timez*选项显式指定初使值。如果你有SUPER 权限，可以用下面的语句在运行时设置全局变量值：

```
mysql> SET GLOBAL time_zone = timezone;
```

- 每个连接的时区。每个客户端连接有自己的时区设置，用会话time_zone变量给出。其初使值与全局变量time_zone相同，但可以用下面的语句重设：

```
mysql> SET time_zone = timezone;
```

可以用下面的方法查询当前的全局变量值和每个连接的时区：

```
mysql> SELECT @@global.time_zone, @@session.time_zone;
```

*timezone*值为字符串，表示UTC的偏移量，例如'+10:00'或'-6:00'。如果已经创建并装入mysql数据库中的时区相关表，你还可以使用命名的时区，例如'Europe/Helsinki'、'US/Eastern'或'MET'。值'SYSTEM'说明该时区应与系统时区相同。时区名对大小写不敏感。

MySQL安装程序在mysql数据库中创建时区表，但不装载。你必须手动装载。(如果你正从以前的版本升级到MySQL 4.1.3或更新版本，你应通过升级mysql数据库来创建表。参见2.10.2节，“升级授权表”中的说明）。

如果你的系统有自己的时区信息数据库(描述时区的一系列文件)，应使用**mysql_tzinfo_to_sql**程序来填充时区表。示例系统如Linux、FreeBSD、Sun Solaris和Mac OS X。这些文件的可能位置为/usr/share/zoneinfo目录。如果你的系统没有时区信息数据库，可以使用本节后面描述的下载的软件包。

mysql_tzinfo_to_sql程序用来装载时区表。在命令行中，将时区信息目录路径名传递到**mysql_tzinfo_to_sql**并输出发送到**mysql**程序。例如：

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

mysql_tzinfo_to_sql读取系统时区文件并生成SQL语句。**mysql**处理这些语句并装载时区表。

mysql_tzinfo_to_sql还可以用来装载单个时区文件，并生成闰秒信息。

要想装载对应时区*tz_name*的单个时区文件*tz_file*，应这样调用**mysql_tzinfo_to_sql**：

```
shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
```

如果你的时区需要计算闰秒，按下面方法初使化闰秒信息，其中*tz_file*是时区文件名：

```
shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
```

如果你的系统没有时区信息数据库 (例如，Windows或HP-UX)，你可以从http://dev.mysql.com/downloads/timezones.html下载预构建时区表软件包。该软件包包含MyISAM时区表所用的.frm、.MYD和.MYI文件。这些表应属于mysql数据库，因此应将这些文件放到MySQL服务器数据目录的mysql子目录。操作时应关闭服务器。

警告！如果你的系统有时区信息数据库，请不要使用下载的软件包。而应使用**mysql_tzinfo_to_sql**实用工具！否则，MySQL和系统上其它应用程序处理日期时间的方法会有所不同。

关于在复制时时区设置相关请查阅6.7节，“复制特性和已知问题”。

5.11. MySQL日志文件

5.11.1. 错误日志
5.11.2. 通用查询日志
5.11.3. 二进制日志
5.11.4. 慢速查询日志
5.11.5. 日志文件维护

MySQL有几个不同的日志文件，可以帮助你找出**mysqld**内部发生的事情：

日志文件	记入文件中的信息类型
错误日志	记录启动、运行或停止**mysqld**时出现的问题。
查询日志	记录建立的客户端连接和执行的语句。
更新日志	记录更改数据的语句。不赞成使用该日志。
二进制日志	记录所有更改数据的语句。还用于复制。
慢日志	记录所有执行时间超过long_query_time秒的所有查询或不使用索引的查询。

默认情况下，所有日志创建于**mysqld**数据目录中。通过刷新日志，你可以强制 **mysqld**来关闭和重新打开日志文件（或者在某些情况下切换到一个新的日志）。当你执行一个FLUSH LOGS语句或执行**mysqladmin flush-logs**或**mysqladmin refresh**时，出现日志刷新。参见13.5.5.2节，“FLUSH语法”。

如果你正使用MySQL复制功能，从复制服务器将维护更多日志文件，被称为接替日志。相关讨论参见第6章：*MySQL*中的复制。

5.11.1. 错误日志

错误日志文件包含了当**mysqld**启动和停止时，以及服务器在运行过程中发生任何严重错误时的相关信息。

如果**mysqld**莫名其妙地死掉并且**mysqld_safe**需要重新启动它，**mysqld_safe**在错误日志中写入一条restarted mysqld消息。如果**mysqld**注意到需要自动检查或着修复一个表，则错误日志中写入一条消息。

在一些操作系统中，如果**mysqld**死掉，错误日志包含堆栈跟踪信息。跟踪信息可以用来确定**mysqld**死掉的地方。参见E.1.4节，“使用堆栈跟踪”。

可以用--log-error[=*file_name*]选项来指定**mysqld**保存错误日志文件的位置。如果没有给定*file_name*值，**mysqld**使用错误日志名*host_name.err* 并在数据目录中写入日志文件。如果你执行FLUSH LOGS，错误日志用-old重新命名后缀并且**mysqld**创建一个新的空日志文件。(如果未给出--log-error选项，则不会重新命名）。

如果不指定--log-error，或者(在Windows中)如果你使用--console选项，错误被写入标准错误输出stderr。通常标准输出为你的终端。

在Windows中，如果未给出--console选项，错误输出总是写入.err文件。

5.11.2. 通用查询日志

如果你想要知道**mysqld**内部发生了什么，你应该用--log[=*file_name*]或-l [*file_name*]选项启动它。如果没有给定*file_name*的值， 默认名是*host_name*.log。所有连接和语句被记录到日志文件。当你怀疑在客户端发生了错误并想确切地知道该客户端发送给**mysqld**的语句时，该日志可能非常有用。

mysqld按照它接收的顺序记录语句到查询日志。这可能与执行的顺序不同。这与更新日志和二进制日志不同，它们在查询执行后，但是任何一个锁释放之前记录日志。(查询日志还包含所有语句，而二进制日志不包含只查询数据的语句）。

服务器重新启动和日志刷新不会产生新的一般查询日志文件(尽管刷新关闭并重新打开一般查询日志文件)。在Unix中，你可以通过下面的命令重新命名文件并创建一个新文件：

```
shell> mv hostname.log hostname-old.log
shell> mysqladmin flush-logs
shell> cp hostname-old.log to-backup-directory
shell> rm hostname-old.log
```

在Windows中，服务器打开日志文件期间你不能重新命名日志文件。你必须先停止服务器然后重新命名日志文件。然后，重启服务器来创建新的日志文件。

5.11.3. 二进制日志

二进制日志以一种更有效的格式，并且是事务安全的方式包含更新日志中可用的所有信息。

二进制日志包含了所有更新了数据或者已经潜在更新了数据（例如，没有匹配任何行的一个DELETE）的所有语句。语句以"事件"的形式保存，它描述数据更改。

注释：二进制日志已经代替了老的更新日志，更新日志在MySQL 5.1中不再使用。

二进制日志还包含关于每个更新数据库的语句的执行时间信息。它不包含没有修改任何数据的语句。如果你想要记录所有语句（例如，为了识别有问题的查询），你应使用一般查询日志。参见5.11.2节，"通用查询日志"。

二进制日志的主要目的是在恢复使能够最大可能地更新数据库，因为二进制日志包含备份后进行的所有更新。

二进制日志还用于在主复制服务器上记录所有将发送给从服务器的语句。参见第6章：*MySQL*中的复制。

运行服务器时若启用二进制日志则性能大约慢1%。但是，二进制日志的好处，即用于恢复并允许设置复制超过了这个小小的性能损失。

当用--log-bin[=*file_name*]选项启动时，**mysqld**写入包含所有更新数据的SQL命令的日志文件。如果未给出*file_name*值， 默认名为-bin后面所跟的主机名。如果给出了文件名，但没有包含路径，则文件被写入数据目录。建议指定一个文件名，原因参见A.8.1节，"MySQL中的打开事宜"。

如果你在日志名中提供了扩展名(例如，--log-bin=*file_name.extension*)，则扩展名被悄悄除掉并忽略。

mysqld在每个二进制日志名后面添加一个数字扩展名。每次你启动服务器或刷新日志时该数字则增加。如果当前的日志大小达到max_binlog_size，还会自动创建新的二进制日志。如果你正使用大的事务，二进制日志还会超过max_binlog_size：事务全写入一个二进制日志中，绝对不要写入不同的二进制日志中。

为了能够知道还使用了哪个不同的二进制日志文件，**mysqld**还创建一个二进制日志索引文件，包含所有使用的二进制日志文件的文件名。默认情况下与二进制日志文件的文件名相同，扩展名为'.index'。你可以用--log-bin-index[=*file_name*]选项更改二进制日志索引文件的文件名。当**mysqld**在运行时，不应手动编辑该文件；如果这样做将会使**mysqld**变得混乱。

可以用RESET MASTER语句删除所有二进制日志文件，或用PURGE MASTER LOGS只删除部分二进制文件。参见13.5.5.5节，"RESET语法"和13.6.1节，"用于控制主服务器的SQL语句"。

二进制日志格式有一些已知限制，会影响从备份恢复。参见6.7节，"复制特性和已知问题"。

保存程序和触发器的二进制日志的描述参见20.4节，"存储子程序和触发程序的二进制日志功能"。

可以使用下面的**mysqld**选项来影响记录到二进制日志知的内容。又见选项后面的讨论。

- --binlog-do-db=*db_name*

告诉主服务器，如果当前的数据库(即USE选定的数据库)是*db_name*，应将更新记录到二进制日志中。其它所有没有明显指定的数据库 被忽略。如果使用该选项，你应确保只对当前的数据库进行更新。

对于CREATE DATABASE、ALTER DATABASE和DROP DATABASE语句，有一个例外，即通过操作的数据库来决定是否应记录语句，而不是用当前的数据库。

一个不能按照期望执行的例子：如果用binlog-do-db=sales启动服务器，并且执行USE prices; UPDATE sales.january SET amount=amount+1000；，该语句不写入二进制日志。

- --binlog-ignore-db=*db_name*

告诉主服务器，如果当前的数据库(即USE选定的数据库)是*db_name*，不应将更新保存到二进制日志中。如果你使用该选项，你应确保只对当前的数据库进

行更新。

一个不能按照你期望的执行的例子：如果服务器用binlog-ignore-db=sales启动，并且执行USE prices; UPDATE sales.january SET amount=amount+1000；，该语句不写入二进制日志。

类似于--binlog-do-db，对于CREATE DATABASE、ALTER DATABASE和DROP DATABASE语句，有一个例外，即通过操作的数据库来决定是否应记录语句，而不是用当前的数据库。

要想记录或忽视多个数据库，使用多个选项，为每个数据库指定相应的选项。

服务器根据下面的规则对选项进行评估，以便将更新记录到二进制日志中或忽视。请注意对于CREATE/ALTER/DROP DATABASE语句有一个例外。在这些情况下，根据以下规则，所创建、修改或删除的数据库将代替当前的数据库。

1. 是否有binlog-do-db或binlog-ignore-db规则?

- 没有：将语句写入二进制日志并退出。
- 有：执行下一步。

2. 有一些规则(binlog-do-db或binlog-ignore-db或二者都有)。当前有一个数据库(USE是否选择了数据库?)?

- 没有：不要写入语句，并退出。
- 有：执行下一步。

3. 有当前的数据库。是否有binlog-do-db规则?

- 有：当前的数据库是否匹配binlog-do-db规则?
  - 有：写入语句并退出。
  - 没有：不要写入语句，退出。
- No：执行下一步。

4. 有一些binlog-ignore-db规则。当前的数据库是否匹配binlog-ignore-db规则?

- 有：不要写入语句，并退出。
- 没有：写入查询并退出。

例如，只用binlog-do-db=sales运行的服务器不将当前数据库不为sales的语句写入二进制日志(换句话说，binlog-do-db有时可以表示"忽视其它数据库")。

如果你正进行复制，应确保没有从服务器在使用旧的二进制日志文件，方可删除它们。一种方法是每天一次执行**mysqladmin flush-logs**并删除三天前的所有日志。可以手动删除，或最好使用PURGE MASTER LOGS(参见13.6.1节，"用于控制主服务器的SQL语句")，该语句还会安全地更新二进制日志索引文件(可以采用日期参数)。

具有SUPER权限的客户端可以通过SET SQL_LOG_BIN=0语句禁止将自己的语句记入二进制记录。参见13.5.3节，"SET语法"。

你可以用**mysqlbinlog**实用工具检查二进制日志文件。如果你想要重新处理日志止的语句，这很有用。例如，可以从二进制日志更新MySQL服务器，方法如下：

```
shell> mysqlbinlog log-file | mysql -h server_name
```

关于**mysqlbinlog**实用工具的详细信息以及如何使用它，参见8.6节，"mysqlbinlog：用于处理二进制日志文件的实用工具"。

如果你正使用事务，必须使用MySQL二进制日志进行备份，而不能使用旧的更新日志。

查询结束后、锁定被释放前或提交完成后则立即记入二进制日志。这样可以确保按执行顺序记入日志。

对非事务表的更新执行完毕后立即保存到二进制日志中。对于事务表，例如BDB或InnoDB表，所有更改表的更新(UPDATE、DELETE或INSERT) 被缓存起来，直到服务器接收到COMMIT语句。在该点，执行完COMMIT之前，**mysqld**将整个事务写入二进制日志。当处理事务的线程启动时，它为缓冲查询分配binlog_cache_size大小的内存。如果语句大于该值，线程则打开临时文件来保存事务。线程结束后临时文件被删除。

Binlog_cache_use状态变量显示了使用该缓冲区(也可能是临时文件)保存语句的事务的数量。Binlog_cache_disk_use状态变量显示了这些事务中实际上有多少必须使用临时文件。这两个变量可以用于将binlog_cache_size调节到足够大的值，以避免使用临时文件。

max_binlog_cache_size(默认4GB)可以用来限制用来缓存多语句事务的缓冲区总大小。如果某个事务大于该值，将会失败并 回滚。

如果你正使用更新日志或二进制日志，当使用CREATE ... SELECT or INSERT ... SELECT时，并行插入被转换为普通插入。这样通过在备份时使用日志可以确保重新创建表的备份。

请注意MySQL 5.1值的二进制日志格式与以前版本的MySQL不同，因为复制改进了。参见6.5节，"不同MySQL版本之间的复制兼容性"。

默认情况下，并不是每次写入时都将二进制日志与硬盘同步。因此如果操作系统或机器(不仅仅是MySQL服务器)崩溃，有可能二进制日志中最后的语句丢失了。要想防止这种情况，你可以使用sync_binlog全局变量(1是最安全的值，但也是最慢的)，使二进制日志在每*N*次二进制日志写入后与硬盘同步。参见5.3.3节，"服务器系统变量"。即使sync_binlog设置为1,出现崩溃时，也有可能表内容和二进制日志内容之间存在不一致性。例如，如果使用InnoDB表，MySQL服务器处理COMMIT语句，它将整个事务写入二进制日志并将事务提交到InnoDB中。如果在两次操作之间出现崩溃，重启时，事务被InnoDB回滚，但仍然存在二进制日志中。可以用--innodb-safe-binlog选项解决该问题，可以增加InnoDB表内容和二进制日志之间的一致性。(注释：在MySQL 5.1中不需要--innodb-safe-binlog；由于引入了XA事务支持，该选项作废了）。

该选项可以提供更大程度的安全，还应对MySQL服务器进行配置，使每个事务的二进制日志(sync_binlog =1)和(默认情况为真)InnoDB日志与硬盘同步。该选项的效果是崩溃后重启时，在滚回事务后，MySQL服务器从二进制日志剪切 回滚的InnoDB事务。这样可以确保二进制日志反馈InnoDB表的确切数据等，并使从服务器保持与主服务器保持同步(不接收 回滚的语句)。

请注意即使MySQL服务器更新其它存储引擎而不是InnoDB，也可以使用--innodb-safe-binlog。在InnoDB崩溃恢复时，只从二进制日志中删除影响InnoDB表的语句/事务。如果崩溃恢复时MySQL服务器发现二进制日志变短了(即至少缺少一个成功提交的InnoDB事务)，如果sync_binlog =1并且硬盘/文件系统的确能根据需要进行同步(有些不需要)则不会发生，则输出错误消息 ("二进制日志<名>比期望的要小")。在这种情况下，二进制日志不准确，复制应从主服务器的数据快照开始。

写入二进制日志文件和二进制日志索引文件的方法与写入MyISAM表相同。参见A.4.3节，“MySQL处理磁盘满的方式”。

5.11.4. 慢速查询日志

用--log-slow-queries[=*file_name*]选项启动时，**mysqld**写一个包含所有执行时间超过long_query_time秒的SQL语句的日志文件。获得初使表锁定的时间不算作执行时间。

如果没有给出*file_name*值， 默认未主机名，后缀为-slow.log。如果给出了文件名，但不是绝对路径名，文件则写入数据目录。

语句执行完并且所有锁释放后记入慢查询日志。记录顺序可以与执行顺序不相同。

慢查询日志可以用来找到执行时间长的查询，可以用于优化。但是，检查又长又慢的查询日志会很困难。要想容易些，你可以使用**mysqldumpslow**命令获得日志中显示的查询摘要来处理慢查询日志。

在MySQL 5.1的慢查询日志中，不使用索引的慢查询同使用索引的查询一样记录。要想防止不使用索引的慢查询记入慢查询日志，使用--log-short-format选项。参见5.3.1节，“**mysqld**命令行选项”。

在MySQL 5.1中,通过--log-slow-admin-statements服务器选项，你可以请求将慢管理语句，例如OPTIMIZE TABLE、ANALYZE TABLE和 ALTER TABLE写入慢查询日志。

用查询缓存处理的查询不加到慢查询日志中，因为表有零行或一行而不能从索引中受益的查询也不写入慢查询日志。

5.11.5. 日志文件维护

MySQL服务器可以创建各种不同的日志文件，从而可以很容易地看见所进行的操作。参见5.11节，“MySQL日志文件”。但是，你必须定期清理这些文件，确保日志不会占用太多的硬盘空间。

当启用日志使用MySQL时，你可能想要不时地备份并删除旧的日志文件，并告诉MySQL开始记入新文件。参见5.9.1节，“数据库备份”。

在 Linux (Redhat)的安装上，你可为此使用mysql-log-rotate脚本。如果你从RPM分发安装**MySQL**，脚本应该自动被安装了。

在其它系统上，你必须自己安装短脚本，你可从**cron**等入手处理日志文件。

你可以通过**mysqladmin flush-logs**或SQL语句FLUSH LOGS来强制**MySQL**开始使用新的日志文件。

日志清空操作做下列事情：

- 如果使用标准日志(--log)或慢查询日志(--log-slow-queries)，关闭并重新打开日志文件。(默认为mysql.log和`hostname`-slow.log)。
- 如果使用更新日志(--log-update)或二进制日志(--log-bin)，关闭日志并且打开有更高序列号的新日志文件。

如果你只使用更新日志，你只需要重新命名日志文件，然后在备份前清空日志。例如，你可以这样做：

```
shell> cd mysql-data-directory
shell> mv mysql.log mysql.old
shell> mysqladmin flush-logs
```

然后做备份并删除“mysql.old”。

5.12. 在同一台机器上运行多个**MySQL**服务器

5.12.1. 在Windows下运行多个服务器
5.12.2. 在Unix中运行多个服务器
5.12.3. 在多服务器环境中使用客户端程序

在一些情况下，你可能想要在同一台机器上运行多个**mysqld**服务器。你可能想要测试一个新的MySQL发布，同时不影响现有产品的设置。或者，你可能想使不同的用户访问来访问不同的**mysqld**服务器以便他们自己来管理。(例如，你可能是一个Internet服务提供商，希望为不同的客户来提供独立的MySQL安装）。

要想在一个单独的机器上运行多个服务器，每个服务器必须有唯一的各运行参数值。这些值可以在命令行中设置或在选项文件中设置。参见4.3节，“指定程序选项”。

至少下面的选项对每个服务器必须是不同的：

- --port=*port_num*

--port控制着TCP/IP连接的端口号。

· --socket=*path*

--socket控制Unix中的Unix套接字文件路径和在Windows中的命名管道名称。在Windows中，只有支持命名管道连接的服务器才需要明确指定管道名称。

· --shared-memory-base-name=*name*

该选项当前只在Windows中使用。它指定Windows服务器使用的、允许客户端通过共享内存来连接的共享内存名。

· --pid-file=*path*

该选项只在Unix中使用。它指出服务器在其中写入进程ID的文件名。

如果你使用下面的日志文件选项，对于每个服务器来说，它们必须是不同的：

· --log=*path*

· --log-bin=*path*

· --log-update=*path*

· --log-error=*path*

· --bdb-logdir=*path*

日志文件选项的描述参见5.11.5节，“日志文件维护”。

为了提高性能，你可以为每个服务器指定下面选项的不同的值，以便在物理磁盘之间平均分配负荷：

· --tmpdir=*path*

· --bdb-tmpdir=*path*

还推荐使用不同的临时目录，以便容易地确定哪个MySQL服务器创建了给定的临时文件。

一般情况，每个服务器应还使用不同的数据目录，可以通过--datadir=path选项来指定。

警告：一般情况，你决不要让两个服务器更新相同数据库中的数据。否则，如果你的操作系统不支持故障排除系统锁定，该可能会导致非常奇怪的结果。如果（不理会该警告）运行的多个服务器使用相同的数据目录并且启用了日志记录，你必须使用适当的选项来为每个服务器指定唯一的日志文件名。否则，服务器尝试用相同的文件来记录日志。请注意这种类型的设置只能在MyISAM和MERGE表上工作，对其它任何存储引擎不起作用。

多个服务器共享一个数据目录的警告也适用于NFS环境。允许多个MySQL服务器通过NFS访问一个共同的数据目录是一个非常不好的主义。

· 主要问题是NFS存在速度瓶颈。它不是用于这种用途。

· 用NFS的另一个冒险是你必须提出一个方法来确保两个或多个服务器不会相互干扰。NFS文件的锁定通常由lockd后台程序处理，但是目前，没有一个运行平台能够在每种情况下100%可靠地进行锁定。

使你更加容易：忘记在服务器之间通过NFS共享数据目录。一个较好的解决方案是使用包含几个CPU并且和使用有效处理多线程的操作系统的机器。

如果在不同的位置有多个MySQL的安装，一般情况可以用--basedir=path选项为每个服务器指定基本安装目录，使每个服务器使用不同的数据目录、日志文件和PID文件。（所有这些值的 默认值相对于根目录来确定）。那样的话， 你唯一需要指定的其它选项是--socket和--port选项。例如，假如使用tar文件二进制分发版安装不同的MySQL版本。这些安装在不同的位置，因此可以使用各个安装服务器相应的根目录中的**bin/mysqld_safe**命令来启动服务器。**mysqld_safe**确定正确的--basedir选项传递给**mysqld**，你仅需要为**mysqld_safe**指定--socket和--port选项。

正如下面几节所讨论的那样，可以通过设置环境变量或用指定的命令行选项来启动更多的服务器。但是，如果你需要在一个更稳定的基础上运行多个服务器，一个更方便的方法是使用选项文件来为每个服务器指定那些选项值，它对每个服务器必须是唯一的。

5.12.1. 在**Windows**下运行多个服务器

5.12.1.1. 在命令行中启动多个Windows服务器
5.12.1.2. 做为服务启动多个Windows服务器

在Windows中，可以从命令行手动启动来运行多个服务器，每个服务器使用合适的操作参数。在基于Windows NT的系统中，安装几个服务器时，你还有将多个服务器安装为Windows服务并运行的选项。关于从命令行运行MySQL服务器或作为服务运行的一般说明在 2.3节，“在Windows上安装MySQL”中给出。本节描述怎样确保你用不同的启动选项值（对于每个服务器必须是唯一的，例如数据目录）启动各个服务器。这些选项的描述见5.12节，“在同一台机器上运行多个MySQL服务器”。

5.12.1.1. 在命令行中启动多个**Windows**服务器

为了从命令行手动启动多个服务器，可以在命令行中或在选项文件中指定适当的选项。把选项放在选项文件中比较方便，但是需要确保每个服务器可以获得自己的选项。为了实现，为每个创建一个选项文件，并且运行服务时通过--defaults-file选项告诉服务器选项文件名。

假设你想要在端口3307使用数据目录C:\mydata1运行**mysqld**，并且想在端口3308使用数据目录C:\mydata1运行**mysqld-max**。(要想这样做，启动服务器之前要确保，每个数据目录存在并且有自己的mysql数据库拷贝，它包含 授权表）。

然后创建两个选项文件。例如，创建一个文件名为C:\my-opts1.cnf的配置文件，它看起来象这个样子：

```
[mysqld]
datadir = C:/mydata1
port = 3307
```

创建第二个文件名为C:\my-opts1.cnf的配置文件，它看起来象这个样子：

```
mysqld]
datadir = C:/mydata2
port = 3308
```

然后，用它们自己的选项文件启动每个服务器：

```
C:\> C:\mysql\bin\mysqld --defaults-file=C:\my-opts1.cnf
C:\> C:\mysql\bin\mysqld-max --defaults-file=C:\my-opts2.cnf
```

在NT中，每个服务器在前台启动（服务器退出前，不会显示新的提示符）；需要在两个控制台窗口中执行这两个命令。

要想关闭服务器，必须连接到相应的端口号：

C:\> **C:\mysql\bin\mysqladmin --port=3307 shutdown**

C:\> **C:\mysql\bin\mysqladmin --port=3308 shutdown**

如刚才所讨论的，服务器配置允许客户端通过TCP/IP来连接。如果你的Windows版本支持命名管道并且你想允许命名管道连接，使用**mysqld-nt**或**mysqld-max-nt**服务器并指定启用命名管道并且指定管道名的选项。支持命名管道连接的每个服务器必须使用一个唯一的管道名。例如，C:\my-opts1.cnf文件可能象这样来书写：

```
[mysqld]
datadir = C:/mydata1
port = 3307
enable-named-pipe
socket = mypipe1
```

然后，这样启动服务器：

```
C:\> C:\mysql\bin\mysqld-nt --defaults-file=C:\my-opts1.cnf
```

同样修改第2个服务器使用的C:\my-opts2.cnf文件。

5.12.1.2. 做为服务启动多个**Windows**服务器

在基于NT的系统中，MySQL服务器可以以Windows服务的方式来运行。安装、控制和删除单个MySQL服务的过程描述见2.3.12节，“以Windows服务方式启动MySQL”。

你还可以以服务的方式安装多个MySQL服务器。此时，除了所有参数对每个服务器必须是唯一的，你还必须确保每个服务器使用不同的服务名。

在下面的说明中，假设你想要运行**mysqld-nt**服务器的两个不同的版本，它们分别安装在C:\mysql-4.1.8和C:\mysql-5.1.2-alpha目录中。(可能存在这种情况，如果你正在运行版本4.1.8作为你的产品服务器，还想使用5.1.2-alpha版本来进行测试）。

当用--install或--install-manual选项安装一个MySQL服务时，应遵从以下原则：

- 如果你没有指定服务名，服务器使用默认的MySQL服务名，从标准选项文件的[mysqld]组中读取选项。
- 如果你在--install选项后指定了服务名，服务器忽略[mysqld]选项组，从具有相同名的组中读取选项作为服务名。服务器从标准选项文件中读取选项。
- 如果你在服务名后面指定一个--defaults-file选项，服务器忽略标准选项文件，只从命名的文件的[mysqld]组读取选项。

注释：MySQL 4.0.17之前，只有使用默认服务名（MySQL）安装的一个服务器或使用服务名**mysqld**显式安装的一个服务器从标准选项文件读[mysqld]组。到4.0.17时，如果服务器读标准选项文件，则它们均读[mysqld]组，即使它们安装时使用了另一个服务名。这样允许你为选项使用[mysqld]组，用于所有MySQL服务器，并将根据每个服务器命名的选项组用于该服务器，即使用那个服务名安装的服务器。

根据前面叙述，你可以通过几个方法来设置多个服务器。下面的说明描述了一些示例。在尝试之前，应确保你首先关闭并且卸载了所有已有的MySQL服务器。

- 方法**1**：在一个标准选项文件中指定所有服务器选项。要想这样做，为每个服务器使用不同的服务名。假设你想使用服务名mysqld1运行4.1.8版的**mysqld-nt**并使用服务名mysqld2运行5.1.2-alpha版的**mysqld-nt**。在这种情况下，你可以为4.1.8使用[mysqld1]组，为5.1.2-alpha使用[mysqld2]组。例如，你可以象这样建立 C:\my.cnf文件：

```
# options for mysqld1 service
[mysqld1]
basedir = C:/mysql-4.1.8
port = 3307
enable-named-pipe
socket = mypipe1
```

```
·
·                # options for mysqld2 service
·                [mysqld2]
·                basedir = C:/mysql-5.1.2-alpha
·                port = 3308
·                enable-named-pipe
·                socket = mypipe2
```

如下面所示安装服务器，使用服务器的全路径名来确保Windows为每个服务注册正确的可执行程序：

```
C:\> C:\mysql-4.1.8\bin\mysqld-nt --install mysqld1
C:\> C:\mysql-5.1.2-alpha\bin\mysqld-nt --install mysqld2
```

为了启动服务器，使用服务管理器，或用带有适当的服务名的**NET START**：

C:\> **NET START mysqld1**

C:\> **NET START mysqld2**

要想停止服务，使用服务管理器，或用带有适当的服务名的**NET STOP**：

```
C:\> NET STOP mysqld1
C:\> NET STOP mysqld2
```

· 方法**2**：为每个服务器用不同的文件指定选项，当你安装服务时使用--defaults-file告诉每个服务器使用什么文件。此时，每个文件应用一个[mysqld]组列出选项。

使用这种方法为4.1.8版本的**mysqld-nt**指定选项，应象这样创建一个C:\my-opts1.cnf文件：

```
[mysqld]
basedir = C:/mysql-4.1.8
port = 3307
enable-named-pipe
socket = mypipe1
```

对于5.1.2-alpha版的**mysqld-nt**，象这样创建一个C:\my-opts2.cnf文件：

```
[mysqld]
basedir = C:/mysql-5.1.2-alpha port = 3308
enable-named-pipe
socket = mypipe2
```

安装服务如下（在一个单一行中输入每个命令）：

```
C:\> C:\mysql-4.1.8\bin\mysqld-nt -- installmysqld1
           --defaults-file=C:\my-opts1.cnf
C:\> C:\mysql-5.1.2-alpha\bin\mysqld-nt -- installmysqld2
           --defaults-file=C:\my-opts2.cnf
```

当你作为服务安装一个MySQL服务器时，要想使用--defaults-file选项，你必须在此选项之前加服务名。

安装服务后，按照与前面的示例相同的方法启动和停止。

要想卸载多个服务，对每个服务使用**mysqld --remove**，在--remove选项后指定服务名。如果服务名是 默认的(MySQL)，你可以不指定。

5.12.2. 在**Unix**中运行多个服务器

在Unix中运行多个服务器最容易的方法是使用不同的TCP/IP端口s和Unix套接字文件编译，因此每个实例在不同的网络接口侦听。另外，每个安装应在不同的基础目录中编译，那将自动为你的每个服务器产生使用不同的编译进来的数据目录、日志文件和PID文件位置。

假设一个现有的4.1.8版本服务器配置为默认TCP/IP端口号(3306)和Unix套接字文件(/tmp/mysql.sock)。要想配置一个新的5.1.2-alpha版的服务器来使用不同的操作参数，使用一个**configure**命令，大概象这样使用：

```
shell> ./configure --with-tcp-port=port_number \
             --with-unix-socket-path=file_name \
             --prefix=/usr/local/mysql-5.1.2-alpha
```

这里，*port_number*和*file_name*必须不同于默认TCP/IP端口号和Unix套接字文件路径名，并且--prefix值应指定一个不同于现有MySQL安装目录的安装目录。

如果你有一个MySQL服务器正侦听一个给定的端口号，你可以使用下面的命令来找出针对一些重要配置变量它使用了那些操作参数，包括基础目录和Unix套接字文件名：

```
shell>
```

```
mysqladmin --host=host_name --port=port_number variables
```

通过该命令显示的信息，当配置其它服务器时，你可以告诉服务器该选项没有使用的值。

请注意，如果你指定localhost作为一个主机名，**mysqladmin**默认使用Unix套接字文件连接，而不是TCP/IP。从 MySQL 4.1开始，通过--protocol= TCP | SOCKET | PIPE | MEMORY}选项，你可以显示地指定连接协议。

如果只是用一个不同的Unix套接字文件和TCP/IP端口号启动，不必编译新的MySQL服务器。还可以在运行时指定这些值。这样做的一个方法是使用命令行选项：

```
shell> mysqld_safe --socket=file_name --port=port_number
```

要启动第二个服务器，提供不同的--socket和--port选项值，并且传递一个--datadir=path选项给**mysqld_safe**，以便服务器使用一个不同的数据目录。

达到相似效果的另一个方法是使用环境变量来设置 Unix套接字文件名和TCP/IP端口号：

```
shell> MYSQL_UNIX_PORT=/tmp/mysqld-new.sock
shell> MYSQL_TCP_PORT=3307
shell> export MYSQL_UNIX_PORT MYSQL_TCP_PORT
shell> mysql_install_db --user=mysql
shell> mysqld_safe --datadir=/path/to/datadir &
```

这是一个快速启动第二个服务器以用于测试的方法。该方法的最大好处是环境变量设定值适用于你从相同的shell调用的任何客户端程序。因而，那些客户端连接自动指向第二个服务器！

附录F：环境变量包括你可以使用的影响**mysqld**的其它环境变量列表。

对于自动服务器启动，对于每个服务器，机器引导时执行的启动脚本应执行下面的命令，每个命令用一个相应的选项文件路径：

```
mysqld_safe --defaults-file=path
```

每个选项文件应包含一个给定服务器特定的选项值。

在Unix中，**mysqld_multi**脚本是启动多个服务器的另一个方法。参见5.1.5节，“mysqld_multi：管理多个MySQL服务器的程序”。

5.12.3. 在多服务器环境中使用客户端程序

当你想要用一个客户端程序连接一个MySQL服务器时，该服务器侦听不同的网络接口，而不是编译到你的客户端的网络接口，你可以使用下面的方法：

- 启动客户端，用--host=*host_name* --port=*port_number*通过TCP/IP来连接一个远程服务器，用--host=127.0.0.1 --port=*port_number*通过TCP/IP来连接一个本地服务器，或者用--host=localhost --socket=*file_name*通过一个Unix套接字文件或一个Windows命名管道来连接一个本地服务器。
- 从MySQL 4.1起，启动客户端时用--protocol=tcp通过TCP/IP来连接，用--protocol=socket通过一个Unix套接字文件来连接，用--protocol=pipe通过一个命名管道来连接，或用--protocol=memory通过共享内存来连接。对于TCP/IP连接，你可能还需要指定--host和--port选项。对于其它类型的连接，你可能需要指定一个--socket选项来指定一个Unix套接字文件或命名管道名，或者一个--shared-memory-base-name选项来指定共享内存名。共享内存连接仅在Windows中支持。
- 在Unix中，在你启动客户端之前，设置MYSQL_UNIX_PORT和MYSQL_TCP_PORT环境变量来指定Unix套接字文件和TCP/IP端口号。如果你经常使用具体的套接字文件或端口号，你可以在.login文件中放置命令来设置环境变量以便你每次登录时该命令起作用。参见附录F：环境变量。
- 在一个选项文件的[client]组中指定默认Unix套接字文件和TCP/IP端口号。例如，你可以在Windows中使用C:\my.cnf文件，或在Unix中主目录内使用.my.cnf文件。参见4.3.2节，“使用选项文件”。
- 在C程序中，在mysql_real_connect()调用时，你可以指定套接字文件或端口号参数。通过调用mysql_options()，你还可以有程序读选项文件。参见25.2.3节，“C API函数描述”。
- 如果你正在使用Perl DBD::mysql模块，你可以从MySQL选项文件中读取选项。例如：

```
$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);
```

参见25.4节，“MySQL Perl API”。

其它程序接口可以为读选项文件提供相似的功能。

5.13. MySQL查询高速缓冲

5.13.1. 查询高速缓冲如何工作
5.13.2. 查询高速缓冲SELECT选项
5.13.3. 查询高速缓冲配置
5.13.4. 查询高速缓冲状态和维护

查询缓存存储SELECT查询的文本以及发送给客户端的相应结果。如果随后收到一个相同的查询，服务器从查询缓存中重新得到查询结果，而不再需要解析和执行查询。

如果你有一个不经常改变的表并且服务器收到该表的大量相同查询，查询缓存在这样的应用环境中十分有用。对于许多Web服务器来说存在这种典型情况，它根据数据库内容生成大量的动态页面。

注释：查询缓存不返回旧的数据。当表更改后，查询缓存值的相关条目被清空。

注释：如果你有许多**mysqld**服务器更新相同的MyISAM表，在这种情况下查询缓存不起作用。

注释：查询缓存不适用于服务器方编写的语句。如果正在使用服务器方编写的语句，要考虑到这些语句将不会应用查询缓存。参见 25.2.4节，“C API预处理语句”。

下面是查询缓存的一些性能数据。这些结果是在Linux Alpha 2 x 500MHz系统（2GB RAM，64MB查询缓存）上运行MySQL基准组件产生的。

- 如果执行的所有查询是简单的(如从只有一行数据的表中选取一行)，但查询是不同的，查询不能被缓存，查询缓存激活率是13%。这可以看作是最坏的情形。在实际应用中，查询要复杂得多，因此，查询缓存使用率一般会很低。
- 从只有一行的表中查找一行数据时，使用查询缓存比不使用速度快238%。这可以看作查询使用缓存时速度提高最小的情况。

服务器启动时要禁用查询缓存，设置query_cache_size系统变量为0。禁用查询缓存代码后，没有明显的速度提高。编译MySQL时，通过在**configure**中使用--without-query-cache选项，可以从服务器中彻底去除查询缓存能力。

5.13.1. 查询高速缓冲如何工作

本节描述查询缓存的工作原理。5.13.3节，“查询高速缓冲配置”描述了怎样控制是否使用查询缓存。

查询解析之前进行比较，因此下面的两个查询被查询缓存认为是不相同的：

```
SELECT * FROM tbl_name
Select * from tbl_name
```

查询必须是完全相同的(逐字节相同)才能够被认为是相同的。另外，同样的查询字符串由于其它原因可能认为是不同的。使用不同的数据库、不同的协议版本或者不同 默认字符集的查询被认为是不同的查询并且分别进行缓存。

从查询缓存中提取一个查询之前，MySQL检查用户对所有相关数据库和表的SELECT权限。如果没有权限，不使用缓存结果。

如果从查询缓存中返回一个查询结果，服务器把Qcache_hits状态变量的值加一，而不是Com_select变量。参见5.13.4节，“查询高速缓冲状态和维护”。

如果一个表被更改了，那么使用那个表的所有缓冲查询将不再有效，并且从缓冲区中移出。这包括那些映射到改变了的表的使用MERGE表的查询。一个表可以被许多类型的语句更改，例如INSERT、UPDATE、DELETE、TRUNCATE、ALTER TABLE、DROP TABLE或DROP DATABASE。

COMMIT执行完后，被更改的事务InnoDB表不再有效。

使用InnoDB表时，查询缓存也在事务中工作，使用该表的版本号来检测其内容是否仍旧是当前的。

在MySQL 5.1中，视图产生的查询被缓存。

SELECT SQL_CALC_FOUND_ROWS ...和SELECT FOUND_ROWS() type类型的查询使用查询缓存。即使因创建的行数也被保存在缓冲区内，前面的查询从缓存中提取，FOUND_ROWS()也返回正确的值。

如果一个查询包含下面函数中的任何一个，它不会被缓存：

BENCHMARK()	CONNECTION_ID()	CURDATE()
CURRENT_DATE()	CURRENT_TIME()	CURRENT_TIMESTAMP()
CURTIME()	DATABASE()	带一个参数的ENCRYPT()
FOUND_ROWS()	GET_LOCK()	LAST_INSERT_ID()
LOAD_FILE()	MASTER_POS_WAIT()	NOW()
RAND()	RELEASE_LOCK()	SYSDATE()
不带参数的UNIX_TIMESTAMP()	USER()	

在下面的这些条件下，查询也不会被缓存：

- 引用自定义函数(UDFs)。
- 引用自定义变量。
- 引用mysql系统数据库中的表。
- 下面方式中的任何一种：

```
SELECT ...IN SHARE MODE
SELECT ...FOR UPDATE
SELECT ...INTO OUTFILE ...
SELECT ...INTO DUMPFILE ...
SELECT * FROM ...WHERE autoincrement_col IS NULL
```

最后一种方式不能被缓存是因为它被用作为ODBC工作区来获取最近插入的ID值。参见26.1.14.1节，“如何在ODBC中获取AUTO_INCREMENT列的值”。

- 被作为编写好的语句，即使没有使用占位符。例如，下面使用的查询：

```
char *my_sql_stmt = "SELECT a, b FROM table_c";
   /* ...*/
mysql_stmt_prepare(stmt, my_sql_stmt, strlen(my_sql_stmt));
```

不被缓存。参见25.2.4节，“C API预处理语句”。

- 使用TEMPORARY表。
- 不使用任何表。
- 用户有某个表的列级权限。

5.13.2. 查询高速缓冲SELECT选项

可以在SELECT语句中指定查询缓存相关选项：

- SQL_CACHE

如果query_cache_type系统变量的值是ON或DEMAND，查询结果被缓存。

- SQL_NO_CACHE

查询结果不被缓存。

示例：

```
SELECT SQL_CACHE id, name FROM customer;
SELECT SQL_NO_CACHE id, name FROM customer;
```

5.13.3. 查询高速缓冲配置

通过have_query_cache服务器系统变量指示查询缓存是否可用：

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
```

即使禁用查询缓存，当使用标准 MySQL二进制时，这个值总是YES。

其它几个系统变量控制查询缓存操作。当启动**mysqld**时，这些变量可以在选项文件或者命令行中设置。所有查询缓存系统变量名以query_cache_ 开头。它们的详细描述见5.3.3节，“服务器系统变量”，还给出了额外的配置信息。

为了设置查询缓存大小，设置query_cache_size系统变量。设置为0表示禁用查询缓存。 默认缓存大小设置为0；也就是禁用查询缓存。

当设置query_cache_size变量为非零值时，应记住查询缓存至少大约需要40KB来分配其数据结构。(具体大小取决于系统结构）。如果你把该值设置的太小，将会得到一个警告，如本例所示：

mysql> **SET GLOBAL query_cache_size = 40000;**

Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> **SHOW WARNINGS\G**

********** 1. row **********

Level: Warning

Code: 1282

Message: Query cache failed to set size 39936; new query cache size is 0

mysql> **SET GLOBAL query_cache_size = 41984;**

Query OK, 0 rows affected (0.00 sec)

mysql> **SHOW VARIABLES LIKE 'query_cache_size';**

```
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| query_cache_size | 41984 |
+------------------+-------+
```

如果查询缓存大小设置为大于0，query_cache_type变量影响其工作方式。这个变量可以设置为下面的值：

- 0或OFF将阻止缓存或查询缓存结果。
- 1或ON将允许缓存，以SELECT SQL_NO_CACHE开始的查询语句除外。
- 2或DEMAND，仅对以SELECT SQL_CACHE开始的那些查询语句启用缓存。

设置query_cache_type变量的GLOBAL值将决定更改后所有连接客户端的缓存行为。具体客户端可以通过设置query_cache_type变量的会话值控制它们本身连接的缓存行为。例如，一个客户可以禁用自己的查询缓存，方法如下：

mysql> **SET SESSION query_cache_type = OFF;**

要控制可以被缓存的具体查询结果的最大值，应设置query_cache_limit变量。 默认值是1MB。

当一个查询结果（返回给客户端的数据）从查询缓冲中提取期间，它在查询缓存中排序。因此，数据通常不在大的数据块中处理。查询缓存根据数据排序要求分配数据块，因此，当一个数据块用完后分配一个新的数据块。因为内存分配操作是昂贵的(费时的)，所以通过query_cache_min_res_unit系统变量给查询缓存分配最小值。当查询执行时，最新的结果数据块根据实际数据大小来确定，因此可以释放不使用的内存。根据你的服务器执行查询的类型，你会发现调整query_cache_min_res_unit变量的值是有用的：

- query_cache_min_res_unit默认值是4KB。这应该适合大部分情况。
- 如果你有大量返回小结果数据的查询，默认数据块大小可能会导致内存碎片，显示为大量空闲内存块。由于缺少内存，内存碎片会强制查询缓存从缓存内存中修整（删除）查询。这时，你应该减少query_cache_min_res_unit变量的值。空闲块和由于修整而移出的查询的数量通过Qcache_free_blocks和Qcache_lowmem_prunes变量的值给出。
- 如果大量查询返回大结果（检查 Qcache_total_blocks和Qcache_queries_in_cache状态变量），你可以通过增加query_cache_min_res_unit变量的值来提高性能。但是，注意不要使它变得太大（参见前面的条目）。

5.13.4. 查询高速缓冲状态和维护

可以使用下面的语句检查MySQL服务器是否提供查询缓存功能：

mysql> **SHOW VARIABLES LIKE 'have_query_cache';**

```
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
```

可以使用FLUSH QUERY CACHE语句来清理查询缓存碎片以提高内存使用性能。该语句不从缓存中移出任何查询。

RESET QUERY CACHE语句从查询缓存中移出所有查询。FLUSH TABLES语句也执行同样的工作。

为了监视查询缓存性能，使用SHOW STATUS查看缓存状态变量：

```
mysql> SHOW STATUS LIKE 'Qcache%';
+--------------------------+---------+
|变量名                    |值 |
+--------------------------+---------+
| Qcache_free_blocks       | 36      |
| Qcache_free_memory       | 138488 |
| Qcache_hits              | 79570   |
| Qcache_inserts           | 27087   |
| Qcache_lowmem_prunes     | 3114    |
| Qcache_not_cached        | 22989   |
| Qcache_queries_in_cache | 415     |
| Qcache_total_blocks      | 912     |
+--------------------------+---------+
```

这些变量的描述见5.3.4节，“服务器状态变量”。这里描述它们的一些应用。

SELECT查询的总数量等价于：

```
Com_select
+ Qcache_hits
+ queries with errors found by parser
```

Com_select的值等价于：

```
Qcache_inserts
+ Qcache_not_cached
+ queries with errors found during columns/rights check
```

查询缓存使用长度可变块，因此Qcache_total_blocks和Qcache_free_blocks可以显示查询缓存内存碎片。执行FLUSH QUERY CACHE后，只保留一个空闲块。

每个缓存查询至少需要两个块（一个块用于查询文本，一个或多个块用于查询结果）。并且，每一个查询使用的每个表需要一个块。但是，如果两个或多个查询使用相同的表，仅需要分配一个块。

Qcache_lowmem_prunes状态变量提供的信息能够帮助你你调整查询缓存的大小。它计算为了缓存新的查询而从查询缓冲区中移出到自由内存中的查询的数目。查询缓冲区使用最近最少使用(LRU)策略来确定哪些查询从缓冲区中移出。调整信息在5.13.3节，“查询高速缓冲配置”中给出。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第6章：MySQL中的复制

目录

6.1. 复制介绍
6.2. 复制实施概述
6.3. 复制实施细节

6.3.1. 复制主线程状态
6.3.2. 复制从I/O线程状态
6.3.3. 复制从SQL线程状态
6.3.4. 复制传递和状态文件

6.4. 如何设置复制
6.5. 不同MySQL版本之间的复制兼容性
6.6. 升级复制设置

6.6.1. 将复制升级到5.0版

6.7. 复制特性和已知问题
6.8. 复制启动选项
6.9. 复制FAQ
6.10. 复制故障诊断与排除
6.11. 通报复制缺陷
6.12. 多服务器复制中的Auto-Increment

本章描述了MySQL提供的各种复制特性。引入了复制概念，显示如何设置复制服务器和服务以指导相应的复制选项。还提供了FAQ(以及答案) 列表，以及解决复制问题的排错建议。

关于复制相关的SQL语句的语法描述，参见13.6节，“复制语句”。

我们建议你经常访问我们的网址http://www.mysql.com，并检查对本章的修改。复制在不断地得到改进，我们用最新的信息定期更新本手册。

6.1. 复制介绍

MySQL支持单向、异步复制，复制过程中一个服务器充当主服务器，而一个或多个其它服务器充当从服务器。(这与同步复制可以进行对比，同步复制是MySQL簇的一个特征—参见第17章：*MySQL*簇）。主服务器将更新写入二进制日志文件，并维护文件的一个索引以跟踪日志循环。这些日志可以记录发送到从服务器的更新。当一个从服务器连接主服务器时，它通知主服务器从服务器在日志中读取的最后一次成功更新的位置。从服务器接收从那时起发生的任何更新，然后封锁并等待主服务器通知新的更新。

如果你想要设置链式复制服务器，从服务器本身也可以充当主服务器。

请注意当你进行复制时，所有对复制中的表的更新必须在主服务器上进行。否则，你必须要小心，以避免用户对主服务器上的表进行的更新与对从服务器上的表所进行的更新之间的冲突。

单向复制有利于健壮性、速度和系统管理：

- 主服务器/从服务器设置增加了健壮性。主服务器出现问题时，你可以切换到从服务器作为备份。
- 通过在主服务器和从服务器之间切分处理客户查询的负荷，可以得到更好的客户响应时间。SELECT查

询可以发送到从服务器以降低主服务器的查询处理负荷。但修改数据的语句仍然应发送到主服务器，以便主服务器和从服务器保持同步。如果非更新查询为主，该负载均衡策略很有效，但一般是更新查询。

· 使用复制的另一个好处是可以使用一个从服务器执行备份，而不会干扰主服务器。在备份过程中主服务器可以继续处理更新。参见5.9.1节，“数据库备份”。

6.2. 复制实施概述

MySQL复制基于主服务器在二进制日志中跟踪所有对数据库的更改(更新、删除等等)。因此，要进行复制，必须在主服务器上启用二进制日志。参见5.11.3节，“二进制日志”。

每个从服务器从主服务器接收主服务器已经记录到其二进制日志的保存的更新，以便从服务器可以对其数据拷贝执行相同的更新。

认识到二进制日志只是一个从启用二进制日志的固定时间点开始的记录非常重要。任何设置的从服务器需要主服务器上的在主服务器上启用二进制日志时的数据库拷贝。如果启动从服务器时，其数据库与主服务器上的启动二进制日志时的状态不相同，从服务器很可能失败。

将主服务器的数据拷贝到从服务器的一个途径是使用LOAD DATA FROM MASTER语句。请注意LOAD DATA FROM MASTER目前只在所有表使用MyISAM存储引擎的主服务器上工作。并且，该语句将获得全局读锁定，因此当表正复制到从服务器上时，不可能在主服务器上进行更新。当我们执行表的无锁热备份时，则不再需要全局读锁定。

由于这些限制，我们建议只有主服务器上的数据集相对较小，或者主服务器上延迟读锁定已经被接受，才可以使用LOAD DATA FROM MASTER。而LOAD DATA FROM MASTER的实际速度随系统的不同而不同，对于执行时间，最好的规则是每1MB的数据用1秒钟。这是一个粗略的估计，但你会发现如果主服务器和从服务器的性能上等价于700MHz Pentium CPU，通过100Mbps的网络进行连接，则该估计相当准确。

从服务器设置为复制主服务器的数据后，它连接主服务器并等待更新过程。如果主服务器失败，或者从服务器失去与主服务器之间的连接，从服务器保持定期尝试连接，直到它能够继续帧听更新。由--master-connect-retry选项控制重试间隔。 默认为60秒。

每个从服务器跟踪复制时间。主服务器不知道有多少个从服务器或在某一时刻有哪些被更新了。

6.3. 复制实施细节

6.3.1. 复制主线程状态
6.3.2. 复制从I/O线程状态
6.3.3. 复制从SQL线程状态
6.3.4. 复制传递和状态文件

MySQL使用3个线程来执行复制功能(其中1个在主服务器上，另两个在从服务器上。当发出START SLAVE时，从服务器创建一个I/O线程，以连接主服务器并让它发送记录在其二进制日志中的语句。主服务器创建一个线程将二进制日志中的内容发送到从服务器。该线程可以识别为主服务器上SHOW PROCESSLIST的输出中的Binlog Dump线程。从服务器I/O线程读取主服务器Binlog Dump线程发送的内容并将该数据拷贝到从服务器数据目录中的本地文件中，即中继日志。第3个线程是SQL线程，是从服务器创建用于读取中继日志并执行日志中包含的更新。

在前面的描述中，每个从服务器有3个线程。有多个从服务器的主服务器创建为每个当前连接的从服务器创建一个线程；每个从服务器有自己的I/O和SQL线程。

这样读取和执行语句被分成两个独立的任务。如果语句执行较慢则语句读取任务没有慢下来。例如，如果从服

I/O

务器有一段时间没有运行了，当从服务器启动时，其　线程可以很快地从主服务器索取所有二进制日志内容，即使SQL线程远远滞后。如果从服务器在SQL线程执行完所有索取的语句前停止，I/O 线程至少已经索取了所有内容，以便语句的安全拷贝保存到本地从服务器的中继日志中，供从服务器下次启动时执行。这样允许清空主服务器上的二进制日志，因为不再需要等候从服务器来索取其内容。

SHOW PROCESSLIST语句可以提供在主服务器上和从服务器上发生的关于复制的信息。

下面的例子说明了这3个线程在SHOW PROCESSLIST中的显示。

在主服务器上，SHOW PROCESSLIST的输出看上去应为:

```
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
```

这儿，线程2是一个连接从服务器的复制线程。该信息表示所有主要更新已经被发送到从服务器，主服务器正等待更多的更新出现。

在从服务器上，SHOW PROCESSLIST的输出看上去应为:

```
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
```

该信息表示线程10是同主服务器通信的I/O线程，线程11是处理保存在中继日志中的更新的SQL线程。SHOW PROCESSLIST运行时，两个线程均空闲，等待其它更新。

请注意Time列的值可以显示从服务器比主服务器滞后多长时间。参见6.9节，“复制FAQ”。

6.3.1. 复制主线程状态

下面列出了主服务器的Binlog Dump线程的State列的最常见的状态。如果你没有在主服务器上看见任何Binlog Dump线程，这说明复制没有在运行—即，目前没有连接任何从服务器。

- Sending binlog event to slave

二进制日志由各种事件组成，一个事件通常为一个更新加一些其它信息。线程已经从二进制日志读取了一个事件并且正将它发送到从服务器。

- Finished reading one binlog; switching to next binlog

线程已经读完二进制日志文件并且正打开下一个要发送到从服务器的日志文件。

- Has sent all binlog to slave; waiting for binlog to be updated

线程已经从二进制日志读取所有主要的更新并已经发送到了从服务器。线程现在正空闲，等待由主服务器上新的更新导致的出现在二进制日志中的新事件。

- Waiting to finalize termination

线程停止时发生的一个很简单的状态。

6.3.2. 复制从**I/O**线程状态

下面列出了从服务器的I/O线程的State列的最常见的状态。该状态也出现在Slave_IO_State列，由SHOW SLAVE STATUS显示。这说明你可以只通过该语句仔细浏览所发生的事情。

- Connecting to master

线程正试图连接主服务器。

- Checking master version

建立同主服务器之间的连接后立即临时出现的状态。

- Registering slave on master

建立同主服务器之间的连接后立即临时出现的状态。

- Requesting binlog dump

建立同主服务器之间的连接后立即临时出现的状态。线程向主服务器发送一条请求，索取从请求的二进制日志文件名和位置开始的二进制日志的内容。

- Waiting to reconnect after a failed binlog dump request

如果二进制日志转储请求失败(由于没有连接)，线程进入睡眠状态，然后定期尝试重新连接。可以使用--master-connect-retry选项指定重试之间的间隔。

· Reconnecting after a failed binlog dump request

线程正尝试重新连接主服务器。

· Waiting for master to send event

线程已经连接上主服务器，正等待二进制日志事件到达。如果主服务器正空闲，会持续较长的时间。如果等待持续slave_read_timeout秒，则发生超时。此时，线程认为连接被中断并企图重新连接。

· Queueing master event to the relay log

线程已经读取一个事件，正将它复制到中继日志供SQL线程来处理。

· Waiting to reconnect after a failed master event read

读取时(由于没有连接)出现错误。线程企图重新连接前将睡眠master-connect-retry秒。

· Reconnecting after a failed master event read

线程正尝试重新连接主服务器。当连接重新建立后，状态变为Waiting for master to send event。

· Waiting for the slave SQL thread to free enough relay log space

正使用一个非零relay_log_space_limit值，中继日志已经增长到其组合大小超过该值。I/O线程正等待直到SQL线程处理中继日志内容并删除部分中继日志文件来释放足够的空间。

· Waiting for slave mutex on exit

线程停止时发生的一个很简单的状态。

6.3.3. 复制从SQL线程状态

下面列出了从服务器的SQL线程的State列的最常见的状态。

· Reading event from the relay log

线程已经从中继日志读取一个事件，可以对事件进行处理了。

· Has read all relay log; waiting for the slave I/O thread to update it

线程已经处理了中继日志文件中的所有事件，现在正等待I/O线程将新事件写入中继日志。

· Waiting for slave mutex on exit

线程停止时发生的一个很简单的状态。

I/O线程的State列也可以显示语句的文本。这说明线程已经从中继日志读取了一个事件，从中提取了语句，并且正在执行语句。

6.3.4. 复制传递和状态文件

默认情况，中继日志使用*host_name-relay-bin.nnnnnn*形式的文件名，其中*host_name*是从服务器主机

名，*nnnnnn*是序列号。用连续序列号来创建连续中继日志文件，从000001开始。从服务器跟踪索引文件中目前正使用的中继日志。 默认中继日志索引文件名为*host_name-relay-bin.index*。默认情况，在从服务器的数据目录中创建这些文件。可以用--relay-log和--relay-log-index服务器选项覆盖 默认文件名。参见6.8节，“复制启动选项”。

中继日志与二进制日志的格式相同，并且可以用**mysqlbinlog**读取。SQL线程执行完中继日志中的所有事件并且不再需要之后，立即自动删除它。没有直接的删除中继日志的机制，因为SQL线程可以负责完成。然而，FLUSH LOGS可以循环中继日志，当SQL线程删除日志时会有影响。

在下面的条件下创建新的中继日志：

- 每次I/O线程启动时创建一个新的中继日志。
- 当日志被刷新时；例如，用FLUSH LOGS或**mysqladmin flush-logs**。
- 当当前的中继日志文件变得太大时。“太大”含义的确定方法：
  - max_relay_log_size，如果max_relay_log_size > 0
  - max_binlog_size，如果max_relay_log_size = 0

从属复制服务器在数据目录中另外创建两个小文件。这些状态文件默认名为主master.info和relay-log.info。它们包含SHOW SLAVE STATUS语句的输出所显示的信息(关于该语句的描述参见13.6.2节，“用于控制从服务器的SQL语句”)。状态文件保存在硬盘上，从服务器关闭时不会丢失。下次从服务器启动时，读取这些文件以确定它已经从主服务器读取了多少二进制日志，以及处理自己的中继日志的程度。

由I/O线程更新master.info文件。文件中的行和SHOW SLAVE STATUS显示的列的对应关系为：

行	描述
1	文件中的行号
2	Master_Log_File
3	Read_Master_Log_Pos
4	Master_Host
5	Master_User
6	密码(不由SHOW SLAVE STATUS显示)
7	Master_Port
8	Connect_Retry
9	Master_SSL_Allowed
10	Master_SSL_CA_File
11	Master_SSL_CA_Path
12	Master_SSL_Cert
13	Master_SSL_Cipher
14	Master_SSL_Key

由SQL线程更新relay-log.info文件。文件中的行和SHOW SLAVE STATUS显示的列的对应关系为：

行	描述
1	Relay_Log_File
2	Relay_Log_Pos
3	Relay_Master_Log_File
4	Exec_Master_Log_Pos

当备份从服务器的数据时，你还应备份这两个小文件以及中继日志文件。它们用来在恢复从服务器的数据后继续进行复制。如果丢失了中继日志但仍然有relay-log.info文件，你可以通过检查该文件来确定SQL线程已经执行的主服务器中二进制日志的程度。然后可以用Master_Log_File和Master_LOG_POS选项执行CHANGE MASTER TO来告诉从服务器重新从该点读取二进制日志。当然，要求二进制日志仍然在主服务器上。

如果从服务器正复制LOAD DATA INFILE语句，你应也备份该目录内从服务器用于该目的的任何SQL_LOAD-*文件。从服务器需要这些文件继续复制任何中断的LOAD DATA INFILE操作。用--slave-load-tmpdir选项来指定目录的位置。如果未指定， 默认值为tmpdir变量的值。

6.4. 如何设置复制

这里简单描述了如何为你当前的MySQL服务器设置完整的复制。假设你想要复制主服务器上的所有数据库，并且还没有配置的复制。你需要关闭主服务器来完成下面所列的步骤。

下面的程序针对设置一个从服务器，你可以用来设置多个从服务器。

虽然该方法是设置从服务器的最直接的途径，它并不是唯一的一个。例如，如果你有一个主服务器的数据快照，并且主服务器已经设置了服务器ID，启用了二进制日志，不需要关闭主服务器或停止对它的更新也可以设置从服务器。详情请参见6.9节，“复制FAQ”。

如果想要管理MySQL复制设置，我们建议你通读本章，并尝试13.6.1节，“用于控制主服务器的SQL语句”和13.6.2节，“用于控制从服务器的SQL语句”中的所有语句。还应熟悉6.8节，“复制启动选项”中描述的复制启动选项。

注释：该程序和后面章节所示的复制SQL语句需要SUPER权限。

1. 确保在服务器和从服务器上安装的MySQL版本与6.5节，“不同MySQL版本之间的复制兼容性”所示的表兼容。理想情况，应在主服务器和从服务器上使用最近版本的MySQL。

请先证实问题不是出现在最新的MySQL版本中再通报bug。

2. 在主服务器上为服务器设置一个连接账户。该账户必须授予REPLICATION SLAVE权限。如果账户仅用于复制(推荐这样做)，则不需要再授予任何其它权限。(关于设置用户 账户和权限的信息，参见5.8节，“MySQL用户账户管理”）。

假定你的域为mydomain.com,想要创建用户名为repl的一个账户，从服务器可以使用该账户从你的域内的任何主机使用密码slavepass来访问主服务器。要创建该 账户，可使用GRANT语句：

```
mysql> GRANT REPLICATION SLAVE ON *.*
    -> TO 'repl'@'%.mydomain.com' IDENTIFIED BY 'slavepass';
```

如果你计划从从属服务器主机使用LOAD TABLE FROM MASTER或LOAD DATA FROM MASTER语句，你需要授予该账户其它权限：

- 授予账户SUPER和RELOAD全局权限。
- 为所有想要装载的表授予SELECT权限。任何该 账户不能SELECT的主服务器上的表被LOAD DATA FROM MASTER忽略掉。

3. 执行FLUSH TABLES WITH READ LOCK语句清空所有表和块写入语句：

4. `mysql> FLUSH TABLES WITH READ LOCK;`

对于InnoDB表，请注意：FLUSH TABLES WITH READ LOCK还锁定COMMIT操作。当获得全局读锁定后，可以

开始InnoDB表的文件系统快照。快照不能保证内部(在InnoDB存储引擎内部)一致性(因为InnoDB缓存没有刷新)，但并不需要关心该问题，因为InnoDB可以在启动时解决该问题并给出一致的结果。这说明InnoDB在启动快照时可以进行崩溃恢复，而不会破坏。然而，当保证一致的InnoDB表快照时，还没有途径来停止MySQL服务器。

让客户程序保持运行，发出FLUSH TABLES语句让读锁定保持有效。(如果退出客户程序，锁被释放）。然后对主服务器上的数据进行快照。

创建快照最简单的途径是使用归档程序对主服务器上的数据目录中的数据库进行二进制备份。例如，在Unix中使用**tar**，或者在Windows中使用**PowerArchiver**、**WinRAR**、**WinZip**或者类似的软件。要使用**tar**来创建包括所有数据库的归档文件，进入主服务器的数据目录，然后执行命令：

```
shell> tar -cvf /tmp/mysql-snapshot.tar .
```

如果你想让归档只包括this_db数据库，应使用命令：

```
shell> tar -cvf /tmp/mysql-snapshot.tar ./this_db
```

然后将归档文件复制到从服务器主机的/tmp目录。在该机器上，进入从服务器的数据目录，并使用下述命令解压缩归档文件：

```
shell> tar -xvf /tmp/mysql-snapshot.tar
```

如果从服务器的用户账户与主服务器的不同，你可能不想复制mysql数据库。在这种情况下，应从归档中排除该数据库。你也不需要在归档中包括任何日志文件或者master.info或relay-log.info文件。

当FLUSH TABLES WITH READ LOCK所置读锁定有效时，读取主服务器上当前的二进制日志名和偏移量值：

```
mysql > SHOW MASTER STATUS;
+---------------+----------+--------------+------------------+
| File          | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+---------------+----------+--------------+------------------+
| mysql-bin.003 | 73       | test         | manual,mysql     |
+---------------+----------+--------------+------------------+
```

File列显示日志名，而Position显示偏移量。在该例子中，二进制日志值为mysql-bin.003，偏移量为73。记录该值。以后设置从服务器时需要使用这些值。它们表示复制坐标，从服务器应从该点开始从主服务器上进行新的更新。

取得快照并记录日志名和偏移量后，可以在主服务器上重新启用写活动：

```
mysql> UNLOCK TABLES;
```

如果你正使用InnoDB表，理想情况应使用**InnoDB Hot Backup**工具，使用该工具可以获得一致的快照而不需要在主服务器上进行锁定，并且可以对应从服务器上使用的快照来记录日志名和偏移量。**Hot Backup**是一个附加的非免费(商业)工具，没有包含在标准 MySQL分发中。详细信息参见http://www.innodb.com/manual.php的**InnoDB Hot Backup**主页。

不使用**Hot Backup**工具，最快捷的途径是使用InnoDB表的二进制快照来关闭主服务器并复制InnoDB数据文件、日志文件和表定义文件(.frm文件)。要记录当前的日志文件名和偏移量，关闭服务器之前应发出下面的语句：

```
mysql> FLUSH TABLES WITH READ LOCK;

mysql> SHOW MASTER STATUS;
```

然后记录前面所示的SHOW MASTER STATUS的输出中显示的日志名和偏移量。记录日志名和偏移量后，不解锁表关闭服务器以确保 服务器关闭时的快照与当前的日志文件和偏移量相对应：

```
shell> mysqladmin -u root shutdown
```

适合MyISAM和InnoDB表的另一个方法是对主服务器上的SQL进行转储而不是对前面讨论的二进制复制进行转储。为了实现，可以在主服务器上使用**mysqldump --master-data**，以后将SQL转储文件装入从服务器。但是，这样比二进制复制要慢一些。

如果主服务器运行时没有启用--logs-bin，SHOW MASTER STATUS或**mysqldump --master-data**显示的日志名和位置值为空。在这种情况下，当以后指定从服务器的日志文件和位置时需要使用的值为空字符串(")和4.

5. 确保主服务器主机上my.cnf文件的[mysqld]部分包括一个log-bin选项。该部分还应有一个server-id=Master_id选项，其中master_id必须为1到2^{32}–1之间的一个正整数值。例如：

```
6.          [mysqld]
7.          log-bin=mysql-bin
8.          server-id=1
```

如果没有提供那些选项，应添加它们并重启服务器。

9. 停止用于从服务器的服务器并在其my.cnf文件中添加下面的行：

```
10.       [mysqld]
11.       server-id=slave_id
```

slave_id值同Master_id值一样，必须为1到2^{32}–1之间的一个正整数值。并且，从服务器的ID必须与主服务器的ID不相同。例如：

```
[mysqld]
server-id=2
```

如果设置多个从服务器，每个从服务器必须有一个唯一的server-id值，必须与主服务器的以及其它从服务器的不相同。可以认为server-id值类似于IP地址：这些ID值能唯一识别复制服务器群集中的每个服务器实例。

如果不指定一个server-id值，如果没有定义master-host，则将它设置为1；否则设置为2。请注意如果server-id太长，主服务器 拒绝所有来自从服务器的连接，并且从服务器拒绝连接到主服务器。这样，省略server-id只适合用二进制日志备份。

12.如果对主服务器的数据进行二进制备份，启动从服务器之前将它复制到从服务器的数据目录中。确保对这些文件和目录的权限正确。服务器 MySQL运行的用户必须能够读写文件，如同在主服务器上一样。

如果使用**mysqldum**备份，先启动从服务器(看下一步)。

13.启动从服务器。如果前面已经复制了，用--skip-slave-start选项启动从服务器，以便它不立即尝试连接主服务器。你也可能想要用--logs-warnings选项启动从服务器(默认设置启用)，以便在错误日志中显示更多的问题相关的信息(例如，网络或连接问题)。放弃的连接将记入错误日志，除非其值大于1。

14.如果使用**mysqldump**备份主服务器的数据，将转储文件装载到从服务器：

```
15.       shell> mysql -u root -p < dump_file.sql
16.       在从服务器上执行下面的语句，用你的系统的实际值替换选项值:
17.       mysql> CHANGE MASTER TO
18.           ->     MASTER_HOST='master_host_name',
```

```
19.             ->     MASTER_USER='replication_user_name',
20.             ->     MASTER_PASSWORD='replication_password',
21.             ->     MASTER_LOG_FILE='recorded_log_file_name',
22.             ->     MASTER_LOG_POS=recorded_log_position;
```

下面的表显示了字符串选项的最大长度:

Master_Host	60
Master_USER	16
Master_PASSWORD	32
Master_Log_File	255

23.启动从服务器线程:

```
24.         mysql> START SLAVE;
```

执行这些程序后，从服务器应连接主服务器，并补充自从快照以来发生的任何更新。

如果你忘记设置主服务器的server-id值，从服务器不能连接主服务器。

如果你忘记设置从服务器的server-id值，在从服务器的错误日志中会出现下面的错误:

```
Warning: You should set server-id to a non-0 value if master_host is set;
we will force server id to 2, but this MySQL server will not act as a slave.
```

如果由于其它原因不能复制，从服务器的错误日志中也会出现错误消息。

从服务器复制时，会在其数据目录中发现文件dmaster.info和relay-log.info。从服务器使用这两个文件跟踪已经处理了多少主服务器的二进制日志。不要移除或编辑这些文件，除非你确切知你正在做什么并完全理解其意义。即使这样，最好是使用CHANGE MASTER TO语句。

注释：master.info的内容会覆盖命令行或in my.cnf中指定的部分选项。详情参见6.8节，“复制启动选项”。

有了一个快照，你可以用它根据刚刚描述的从服务器部分来设置其它从服务器。你不需要主服务器的另一个快照；每个从服务器可以使用相同的快照。

注释：为了保证事务InnoDB复制设置的最大可能的耐受性和一致性，应在主服务器的my.cnf文件中使用innodb_flush_log_at_trx_commit=1和sync-binlog=1。

6.5. 不同MySQL版本之间的复制兼容性

MySQL 5.1中使用的二进制日志格式与以前的版本中所使用的大大不同，特别是在字符集处理、LOAD DATA INFILE以及时区方面。

注释：你不能从使用新二进制日志格式的主服务器向使用旧二进制日志格式的从服务器复制(例如，从MySQL 5.0到MySQL 4.1）。。这样操作在复制设置升级服务器时后果严重，参见6.6节，“升级复制设置”。

我们推荐使用最近的MySQL版本，因为复制功能在不断地改进中。我们还推荐主服务器和从服务器使用相同的版本。我们建议升级主服务器和从服务器，运行alpha或beta版本到新的(产品)版本。在许多情况下，从新的主服务器向旧的从服务器复制将会失败。一般原则，运行MySQL 5.1.x的从服务器可以与旧的主服务器(可以运行MySQL 3.23、4.0或者4.1)一起使用，但不能反过来。

前面的信息适合协议级复制兼容性。然而，还会有一个约束条件，例如SQL级兼容性问题。例如， 5.1版本的主服务器不能复制到5.0版本的从服务器，如果复制语句使用5.1版本的SQL特性而不是5.0版本。这些问题和其它问题均在6.7节，“复制特性和已知问题”中讨论。

6.6. 升级复制设置

6.6.1. 将复制升级到5.0版

当在复制设置中升级服务器时，升级过程取决于当前的服务器版本和要升级的服务器版本。

6.6.1. 将复制升级到5.0版

该节适用于将复制从MySQL 3.23、4.0或者4.1升级到5.1。4.0服务器应为4.0.3或更新版。

当将早期MySQL版本系列主服务器升级到5.1时，应先确保该主服务器的所有从服务器使用了相同的5.1.x版本。如果不是这样，你应先升级从服务器。升级从服务器时，应先关闭从服务器，升级到相应5.1.x版本，然后重启从服务器并重新开始复制。5.1版本的从服务器能够读取升级前写入的旧的中继日志并执行日志中包含的语句。升级后从服务器创建的中继日志为5.1格式。

从服务器升级后，关闭主服务器，将它升级到与从服务器相同的5.1.x版本并重启它。5.1主服务器能够读取升级前写入的旧的二进制日志并将它们发送到5.1从服务器。从服务器可以识别旧的格式并正确处理它。升级后主服务器创建的二进制日志采用5.1格式。这样也可以由5.1从服务器识别。

换句话说，当升级到5.1时没有什么措施，只有将主服务器升级到5.1之前先将从服务器升级到5.1。请注意从5.1降级到旧版本不会如此简单：必须确保已经完全处理所有5.1版本的二进制日志或中继日志，以便在降级前可以移除它们。

6.7. 复制特性和已知问题

一般原则，SQL级复制兼容性要求主服务器和从服务器均支持使用的特性。例如，在MySQL 5.0.0中开始使用TIMESTAMPADD()函数。如果在主服务器上使用该函数，不能复制到MySQL 5.0.0之前的从服务器。如果你计划在5.1和以前版本的MySQL之间进行复制，你应查阅对应以前版本系列的MySQL参考手册，查询该系列复制特征相关信息。

下面列出了关于支持什么和不支持什么的详细信息。关于复制的其它InnoDB具体信息参见15.2.6.5节，“InnoDB和MySQL复制”。

关于保存的程序和触发器的复制问题在20.4节，“存储子程序和触发程序的二进制日志功能”中讨论。

- 用AUTO_INCREMENT、LAST_INSERT_ID()和TIMESTAMP值正确实现复制。
- USER()、UUID()和LOAD_FILE()函数毫无改变地被，这样不能可靠地在从服务器上工作。
- 下面的限制只适合基于语句的复制，而不是基于行的复制。处理用户级锁定的函数GET_LOCK()、RELEASE_LOCK()、IS_FREE_LOCK()、IS_USED_LOCK()复制时从服务器不知道在主服务器上同时进行的相关文本；因此如果从服务器上的内容不同，这些函数不用来插入到主服务器的表中(例如不执行INSERT INTO mytable VALUES(GET_LOCK(...)))。
- 在MySQL 5.1中FOREIGN_KEY_CHECKS、SQL_MODE、UNIQUE_CHECKS和SQL_AUTO_IS_NULL变量均复制。但TABLE_TYPE，即STORAGE_ENGINE变量 不复制，有利于在不同的存储引擎之间进行复制。
- 即使主服务器和从服务器有不同的全局字符集变量，以及即使有不同的全局时区变量仍可以复制。

- 下面适合使用不同字符集的MySQL服务器之间的复制：

1. 必须在主服务器和从服务器上总是使用相同的全局字符集和校对规则(--default-character-set、--default-collation)。否则，会在从服务器上遇到复制键值错误，因为在主服务器的字符集中被认为是唯一的键值在从服务器的字符集中可能不是唯一的。

2. 如果主服务器早于MySQL 4.1.3，则会话中的字符集不应与其全局值不同(换句话说，不要使用SET NAMES、SET CHARACTER SET等等)，因为从服务器不知道该字符集的更改。如果主服务器和从服务器均为4.1.3或更新版，可以随便将会话的字符集变量设置为本地值(例如NAMES、CHARACTER SET、COLLATION_CLIENT和COLLATION_SERVER)，因为这些设定值被写入二进制日志，因此从服务器知道。然而，禁止更改会话中这些变量的全局值；如前面所述，主服务器和从服务器必须具有唯一的全局字符集值。

3. 如果在主服务器上的数据库的字符集与全局collation_server值不同，则应设计CREATE TABLE语句，以便它们不隐含依赖数据库的默认字符集(Bug #2326)；一个好的解决办法是在CREATE TABLE中明显说明字符集和校对规则。

- 应在主服务器和从服务器上设置相同的系统时区。否则一些语句，例如使用NOW()或FROM_UNIXTIME()函数的语句，将不会正确复制。可以使用脚本mysqld_safe的--timezone=*timezone_name*选项或通过设置TZ环境变量设置MySQL服务器运行的系统的时区。主服务器和从服务器还应有相同的默认连接时区设置；即主服务器和从服务器应有相同的--default-time-zone参数值。

- CONVERT_TZ(...,...,@global.time_zone)不能正确复制。只有主服务器和从服务器均为5.0.4或更新版才能正确复制CONVERT_TZ(...,...,@session.time_zone)。

- 会话变量只有在更新表的语句中使用时才能正确复制；例如：SET MAX_JOIN_SIZE=1000；INSERT INTO mytable VALUES(@MAX_JOIN_SIZE)不能将相同的数据插入到主服务器上和从服务器上。不适用于通用的SET TIME_ZONE=...；INSERT INTO mytable VALUES(CONVERT_TZ(...,...,@time_zone))。

- 可以将从服务器上的非事务表复为主服务器上的事务表。例如，可以将主服务器上的InnoDB表复制为从服务器上的MyISAM表。然而，复制过程中，如果从服务器在BEGIN/COMMIT块过程中停止则会产生问题，因为从服务器在BEGIN块开始时会重启。该问题出现在TODO中，不久将会得到修复。

- 在MySQL 5.1中可以正确复制引用用户变量(即@*var_name*形式的变量)的更新语句；但在4.1以前的版本中却不可能。请注意从MySQL 5.1开始对用户变量名的大小写不再敏感；当在5.1和旧版本之间设置复制时应考虑该问题。

- 从服务器可以使用SSL连接到主服务器。

- 有一个全局系统变量slave_transaction_retries：如果因为某个InnoDB死锁或超过InnoDB的innodb_lock_wait_timeout或NDB簇的TransactionDeadlockDetectionTimeout或TransactionInactiveTimeout，REPLICATION SLAVESQL线程未能执行某个事务，在给出错误停止前自动重试slave_transaction_retries次。 默认值是10。从MySQL 5.0.4开始，可以从SHOW STATUS的输出中看到重试总次数；参见5.3.4节，“服务器状态变量”。

- 如果在主服务器上的CREATE TABLE语句中使用了DATA DIRECTORY或INDEX DIRECTORY子句，子句也可以在从服务器上使用。如果在从服务器主机文件系统中不存在一致的目录或虽然存在但不能被从服务器访问，则会带来问题。MySQL 5.1支持一个称为NO_DIR_IN_CREATE的sql_mode选项。如果从服务器运行时将SQL模式设置为包括该选项，复制CREATE TABLE语句时将忽略这些子句。结果是在表的数据库目录中创建了MyISAM数据和索引文件。

- 下面的限制只适合基于语句的复制，而不是基于行的复制：如果在查询中数据修改不确定，主服务器和从服务器上的数据可以不同；也就是由查询优化器确定。(这是常用的但不是很好的习惯，即使不是在复制中也不好）。关于该问题的详细解释，参见A.8.1节，“MySQL中的打开事宜”。

· 带READ LOCK的FLUSH LOGS、FLUSH MASTER、FLUSH SLAVE和FLUSH TABLES不记入日志，因为如果复制到从服务器会造成问题。关于语法示例，参见13.5.5.2节，“FLUSH语法”。FLUSH TABLES、ANALYZE TABLE、OPTIMIZE TABLE和REPAIR TABLE语句被写入二进制日志并会复制到从服务器。一般情况不会造成问题，因为这些语句不修改表的数据。但是在某些情况下会带来问题。如果你复制mysql数据库中的授权表并且不使用GRANT直接更新那些表，必须在从服务器上执行FLUSH PRIVILEGES使新的权限生效。并且，如果使用FLUSH TABLES重新命名MERGE表的MyISAM表，必须手动在从服务器上执行FLUSH TABLES。如果不指定NO_WRITE_TO_BINLOG或其别名LOCAL，则这些语句被写入二进制日志。

· MySQL只支持一个主服务器和多个从服务器。我们计划将来添加一个投票算法，当前的主服务器出现问题时自动切换。我们还计划引入代理过程通过向不同的从服务器发送SELECT查询以帮助进行负载均衡。

· 当服务器关闭、重启时，其MEMORY表将变为空。主服务器按下述方法复制该结果：启动后第1次主服务器使用每个MEMORY表，它通知从服务器需要向表写入DELETE FROM语句来清空二进制日志的表。详细信息参见15.4节，“MEMORY (HEAP)存储引擎”。

· 除了关闭从服务器(而不仅仅是从服务器线程) 临时表都被复制，并且还没有在从服务器上执行的更新所使用的临时表也已经复制。如果关闭从服务器，从服务器重启后更新需要的那些临时表不可再用。为了避免该问题，临时表打开时不要关闭从服务器。而应遵照下面的程序：

1. 执行STOP SLAVE语句。
2. 使用SHOW STATUS检查slave_open_temp_tables变量的值。
3. 如果值为0，使用**mysqladmin shutdown**命令关闭从服务器。
4. 如果值不为0，用START SLAVE重启从服务器线程。
5. 后面再重复该程序看下次的运气是否好一些。

我们计划在不久的将来修复该问题。

· 可以很安全地连接用--logs-slave-updates选项指定的循环主服务器/从服务器关系中的服务器。但请注意许多语句在这种设置中不能正确工作，除非你的客户代码关注了潜在的在不同的服务器不同顺序的更新中可能发生的这类问题。

这说明你可以象这样创建设置：

```
A -> B -> C -> A
```

服务器ID被编码在二进制日志事件中，因此服务器A知道何时自己首次创建它读取的事件并且不执行事件(除非用--replicate-same-server-id选项启动了服务器A，只在很少情况下有意义)。这样，没有无限循环。只有对表执行没有冲突的更新时该类循环设置才能工作。换句话说，如果在A和C中插入数据，绝对不应在A中插入键值可能与插入到C中的行相冲突的一行。如果更新的顺序很重要，还不应更新两个服务器上相同的行。

· 如果从服务器上的某个语句产生错误，则从服务器上的SQL线程终止，并且从服务器向错误日志写入一条消息。此时应手动连接从服务器，修复该问题(例如，一个不存在的表)，然后运行START SLAVE。

· 可以很安全地关闭主服务器并在以后重启。如果某个从服务器丢失与主服务器的连接，从服务器尝试立即重新连接。如果失败，从服务器定期重试。(默认设置是每60秒重试一次。可以通过--master-connect-retry选项更改）。从服务器也能够处理网络连接中断。但是，只有从服务器超过slave_net_timeout秒没有从主服务器收到数据才通知网络中断。如果中断时间短，可以降低slave_net_timeout。参见5.3.3节，“服务器系统变量”。

· 关闭从服务器（净关闭）也很安全，因为它可以跟踪它离开的地点。不纯净的关闭操作会产生问题，特别是系统关闭前硬盘缓存未刷新到硬盘上时。如果有不间断电源，可以大大提高系统容错能力。不纯净的关闭主服务器会造成主服务器上的表和二进制日志内容之间的不一致性；在主服务器上使用InnoDB表和--innodb-

safe-binlog选项可以避免该问题。参见5.11.3节，“二进制日志”。(注释：MySQL 5.1中不需要--innodb-safe-binlog，由于引入了XA事务支持已经作废了）。

· 由于MyISAM表的非事务属性，可以有一个语句只是更新一个表并返回错误代码。例如，多行插入时有一个行超过键值约束，或者如果长的更新语句更新部分行后被杀掉了。如果发生在主服务器上，除非错误代码合法并且语句执行产生相同的错误代码，从服务器线程将退出并等待数据库管理员决定如何做。如果该错误代码验证行为不理想，可以用--slave-skip-errors选项掩盖(忽视)部分或全部错误。

· 如果从BEGIN/COMMIT系列的非事务表更新事务表，如果提交事务前更新非事务表，对二进制日志的更新可能会不同步。这是因为事务提交后才被写入二进制日志。

· 事务混合更新事务表和非事务表时，二进制日志中语句的顺序是正确的，即使在ROLLBACK时，所有需要的语句也会写入二进制日志。但是如果在第1个连接的事务完成前，第2个连接更新非事务表，语句记入日志时会出现顺序错误，因为第2个连接的更新执行完后立即写入日志，而不管第1个连接执行的事务的状态如何。

6.8. 复制启动选项

在主服务器和从服务器上，均必须使用server-id选项为每个服务器建立唯一的复制ID。你应为每个主服务器和从服务器从1到$2^{32}-1$的范围挑一个唯一的正整数。例如：server-id=3

用于主服务器上控制二进制日志的选项的相关描述见5.11.3节，“二进制日志”。

下表描述了可以用于MySQL 5.1从属复制服务器的选项。你可以在命令行中或在选项文件中指定这些选项。

某些从服务器复制选项按特殊方式处理，当从服务器启动时如果master.info文件存在并且包含选项值，它们将被忽略掉。下面的选项按这种方式处理：

· --master-host

· --master-user

· --master-password

· --master-port

· --master-connect-retry

· --master-ssl

· --master-ssl-ca

· --master-ssl-capath

· --master-ssl-cert

· --master-ssl-cipher

· --master-ssl-key

5.1中的master.info文件格式包括对应SSL选项的值。并且，文件格式包括文件中的行号，如同第1行。如果你将旧的服务器升级到新的版本，新服务器启动时自动将smaster.info文件升级到新的格式。然而，如果将新服务器降级到旧的版本，首次启动旧版本的服务器之前应删除第1行。

如果从服务器启动时master.info文件不存在，选项采用选项文件或命令行中指定的值。首次将服务器作为从服务器启动时，或者已经运行RESET SLAVE然后已经关闭并重启从服务器时会发生。

如果从服务器启动时master.info文件存在，服务器忽略那些选项。使用master.info文件中发现的值。

如果你使用与master.info文件中相对应的启动选项的不同的值重启从服务器，启动选项的不同的值不会生效，因为服务器继续使用master.info文件。要想使用启动选项的不同的值，必须删除master.info文件并重启从服务器，或(最好是)在从服务器运行时使用CHANGE MASTER TO语句重新设置值。

假定在my.cnf文件中指定该选项：

```
[mysqld]

master-host=some_host
```

第1次作为复制从服务器启动服务器时，从my.cnf文件读取并使用选项。服务器然后记录master.info文件中的值。下次启动服务器时，它只从服务器的master.info文件读取主服务器主机值并忽略选项文件中的值。如果你修改my.cnf文件为some_other_host指定其它主服务器主机，更改仍然不会生效。你应使用CHANGE MASTER TO。

因为服务器给已有master.info文件的优先权高于刚刚描述的启动选项，可以选择不使用这些值的启动选项，而是使用CHANGE MASTER TO语句来指定。参见13.6.2.1节，“CHANGE MASTER TO语法”。

下面的例子显示了如何更广泛地使用启动选项来配置从服务器：

```
[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
```

下面列出了控制复制的启动选项：许多选项可以在服务器运行时通过CHANGE MASTER TO语句重新进行设置。其它选项，例如--replicate-*选项，只能在从服务器启动时进行设置。我们计划将修复该问题。

- --logs-slave-updates

通常情况，从服务器从主服务器接收到的更新不记入它的二进制日志。该选项告诉从服务器将其SQL线程执行的更新记入到从服务器自己的二进制日志。为了使该选项生效，还必须用--logs-bin选项启动从服务器以启用二进制日志。如果想要应用链式复制服务器，应使用--logs-slave-updates。例如，可能你想要这样设置：

```
A -> B -> C
```

也就是说，A为从服务器B的主服务器，B为从服务器C的主服务器。为了能工作，B必须既为主服务器又为从服务器。你必须用--logs-bin启动A和B以启用二进制日志，并且用--logs-slave-updates选项启动B。

- --logs-warnings

让从服务器向错误日志输出更详细的关于其执行操作的消息。例如，通知你网络/连接失败后已经成功重新连接，并通知你每个从服务器线程如何启动。该选项默认启用；要想禁用它，使用--skip-logs-warnings。放弃的连接不记入错误日志，除非该值大于1。

请注意该选项的效果不限于复制。可以对服务器的部分动作产生警告。

- --master-connect-retry=*seconds*

在主服务器宕机或连接丢失的情况下，从服务器线程重新尝试连接主服务器之前睡眠的秒数。如果主服务器.info文件中的值可以读取则优先使用。如果未设置， 默认值为60。

· --master-host=*host*

主复制服务器的主机名或IP地址。如果没有给出该选项，从服务器线程不启动。如果主服务器.info文件中的值可以读取则优先使用。

· --master-info-file=*file_name*

从服务器用于记录主服务器的相关信息使用的文件名。默认名为数据目录中的mysql.info。

· --master-password=*password*

连接主服务器时从服务器线程用于鉴定的账户的密码。如果主服务器.info文件中的值可以读取则优先使用。如果未设置，假定 密码为空。

· --master-port=*port_number*

主服务器正帧听的TCP/IP端口号。如果主服务器.info文件中的值可以读取则优先使用。如果未设置，假定使用编译进来的设定值。如果你未曾用**configure**选项进行修改，该值应为3306。

· --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*

这些选项用于使用SSL设置与主服务器的安全复制连接。它们的含义与5.8.7.6节，“SSL命令行选项”中描述的相应—ssl、--ssl-ca、--ssl-capath、--ssl-cert、--ssl-cipher、--ssl-key选项相同。如果主服务器.info文件中的值可以读取则优先使用。

· --master-user=*username*

连接主服务器时从服务器线程用于鉴定的账户的用户名。该账户必须具有REPLICATION SLAVE权限。如果主服务器.info文件中的值可以读取则优先使用。如果未设置主服务器用户，假定使用用户test。

· --max-relay-logs-size=*size*

自动循环中继日志。参见5.3.3节，“服务器系统变量”。

· --read-only

该选项让从服务器只允许来自从服务器线程或具有SUPER权限的用户的更新。可以确保从服务器不接受来自客户的更新。

· --relay-log=*file_name*

中继日志名。默认名为*host_name-relay-bin.nnnnnn*，其中*host_name*是从服务器主机的名，*nnnnnn*表示中继日志在编号序列中创建。如果中继日志太大(并且你不想降低max_relay_log_size)，需要将它们放到数据目录之外的其它地方，或者如果想要通过硬盘之间的负载均衡提高速度，可以指定选项创建与主机名无关的中继日志名。

· --relay-log-index=*file_name*

中继日志索引文件使用的位置和名称。默认名为*host_name-relay-bin.index*，其中*host_name*为从服务器名。

· --relay-log-info-file=*file_name*

从服务器用于记录中继日志相关信息的文件名。默认名为数据目录中的relay-log.info。

- --relay-log-purge={0|1}

禁用或启用不再需要中继日志时是否自动清空它们。默认值为1(启用)。这是一个全局变量，可以用SET GLOBAL Relay_log_purge动态更改。

- --relay-log-space-limit=*size*

限制所有中继日志在从服务器上所占用空间的上限(0值表示"无限制")。从服务器主机硬盘空间有限时很有用。达到限制后，I/O线程停止从主服务器读取二进制日志中的事件，直到SQL线程被闭锁并且删除了部分未使用的中继日志。请注意该限制并不是绝对的：有可能SQL线程删除中继日志前需要更多的事件。在这种情况下，I/O线程将超过限制，直到SQL线程可以删除部分中继日志。(不这样做将会造成死锁）。--relay-log-space-limit的值不能小于--max-relay-logs-size(或如果--max-relay-logs-size为0，选--max-binlog-size)的值的两倍。在这种情况下，有可能I/O线程等待释放空间，因为超过了--relay-log-space-limit，但SQL线程没有要清空的中继日志，不能满足I/O线程的需求。强制I/O线程临时忽视--relay-log-space-limit。

- --replicate-do-db=*db_name*

告诉从服务器限制默认数据库(由USE所选择)为*db_name*的语句的复制。要指定多个数据库，应多次使用该选项，每个数据库使用一次。请注意不复制跨数据库的语句，例如当已经选择了其它数据库或没有数据库时执行UPDATE *some_db.some_table* SET foo='bar'。如果需要跨数据库进行更新，使用--replicate-wild-do-table=*db_name*.%。请读取该选项列表后面的注意事项。

一个不能按照期望工作的例子：如果用--replicate-do-db=sales启动从服务器，并且在主服务器上执行下面的语句，UPDATE语句不会复制：

```
USE prices;

UPDATE sales.january SET amount=amount+1000;
```

如果需要跨数据库进行更新，应使用--replicate-wild-do-table=*db_name*.%。

"只检查默认数据库"行为的主要原因是语句自己很难知道它是否应被复制(例如，如果你正使用跨数据库的多表DELETE语句或多表UPDATE语句)。如果不需要，只检查默认数据库比检查所有数据库要快得多。

- --replicate-do-table=*db_name.tbl_name*

告诉从服务器线程限制对指定表的复制。要指定多个表，应多次使用该选项，每个表使用一次。同--replicate-do-db对比，允许跨数据库更新。请读取该选项列表后面的注意事项。

- --replicate-ignore-db=*db_name*

告诉从服务器不要复制默认数据库(由USE所选择)为*db_name*的语句。要想忽略多个数据库，应多次使用该选项，每个数据库使用一次。如果正进行跨数据库更新并且不想复制这些更新，不应使用该选项。请读取该选项后面的注意事项。

一个不能按照期望工作的例如：如果用--replicate-ignore-db=sales启动从服务器，并且在主服务器上执行下面的语句，UPDATE语句不会复制：

```
·               USE prices;
·               UPDATE sales.january SET amount=amount+1000;
```

如果需要跨数据库更新，应使用--replicate-wild-ignore-table=*db_name*.%。

- --replicate-ignore-table=*db_name.tbl_name*

告诉从服务器线程不要复制更新指定表的任何语句(即使该语句可能更新其它的表)。要想忽略多个表，应多次

使用该选项，每个表使用一次。同--replicate-ignore-db对比，该选项可以跨数据库进行更新。请读取该选项后面的注意事项。

- --replicate-wild-do-table=*db_name.tbl_name*

告诉从服务器线程限制复制更新的表匹配指定的数据库和表名模式的语句。模式可以包含'%'和'_'通配符，与LIKE模式匹配操作符具有相同的含义。要指定多个表，应多次使用该选项，每个表使用一次。该选项可以跨数据库进行更新。请读取该选项后面的注意事项。

例如：--replicate-wild-do-table=foo%.bar%只复制数据库名以foo开始和表名以bar开始的表的更新。

如果表名模式为%，可匹配任何表名，选项也适合数据库级语句(CREATE DATABASE、DROP DATABASE和ALTER DATABASE)。例如，如果使用--replicate-wild-do-table=foo%.%，如果数据库名匹配模式foo%，则复制数据库级语句。

要想在数据库或表名模式中包括通配符，用反斜线对它们进行转义。例如，要复制名为my_own%db的数据库的所有表，但不复制my1ownAABCdb数据库的表，应这样转义'_'和'%'字符：--replicate-wild-do-table=my_own\%db。如果在命令行中使用选项，可能需要双反斜线或将选项值引起来，取决于命令解释符。例如，用**bash**外壳则需要输入--replicate-wild-do-table=my_own\\%db。

- --replicate-wild-ignore-table=*db_name.tbl_name*

告诉从服务器线程不要复制表匹配给出的通配符模式的语句。要想忽略多个表，应多次使用该选项，每个表使用一次。该选项可以跨数据库进行更新。请读取该选项后面的注意事项。

例如：--replicate-wild-ignore-table=foo%.bar%不复制数据库名以foo开始和表名以bar开始的表的更新。

关于匹配如何工作的信息，参见--replicate-wild-do-table选项的描述。在选项值中包括通配符的规则与--replicate-wild-ignore-table相同。

- --replicate-rewrite-db=*from_name->to_name*

告诉从服务器如果默认数据库(由USE所选择)为主服务器上的*from_name*，则翻译为*to_name*。只影响含有表的语句(不是类似CREATE DATABASE、DROP DATABASE和ALTER DATABASE的语句)，并且只有*from_name*为主服务器上的默认数据库时。该选项不可以跨数据库进行更新。请注意在测试--replicate-*规则之前翻译数据库名。

如果在命令行中使用该选项，'>'字符专用于命令解释符，应将选项值引起来。例如：

```
shell> mysqld --replicate-rewrite-db="olddb->newdb"
```

- --replicate-same-server-id

将用于从服务器上。通常可以默认设置为0以防止循环复制中的无限循环。如果设置为1，该从服务器不跳过有自己的服务器id的事件；通常只在有很少配置的情况下有用。如果使用--logs-slave-updates不能设置为1。请注意默认情况下如果有从服务器的id，服务器I/O线程不将二进制日志事件写入中继日志(该优化可以帮助节省硬盘的使用)。因此如果想要使用--replicate-same-server-id，让从服务器读取自己的SQL线程执行的事件前，一定要用该选项启动。

- --report-host=*slave_name*

从服务器注册过程中报告给主服务器的主机名或IP地址。该值出现在主服务器上SHOW SLAVE HOSTS的输出中。如果不想让从服务器自己在主服务器上注册，则不设置该值。请注意从服务器连接后，主服务器仅仅从TCP/IP套接字读取从服务器的IP号是不够的。由于 NAT和其它路由问题，IP可能不合法，不能从主服务器或其它主机连接从服务器。

- --report-port=*slave_port*

连接从服务器的TCP/IP端口号，从服务器注册过程中报告给主服务器。只有从服务器帧听非默认端口或如果有一个特殊隧道供主服务器或其它客户连接从服务器时才设置它。如果你不确定，不设置该选项。

- --skip-slave-start

告诉从服务器当服务器启动时不启动从服务器线程。使用START SLAVE语句在以后启动线程。

- --slave_compressed_protocol={0|1}

如果该选项设置为 1，如果从服务器和主服务器均支持，使用压缩从服务器/主服务器协议。

- --slave-load-tmpdir=*file_name*

从服务器创建临时文件的目录名。该选项默认等于tmpdir系统变量的值。当从服务器SQL线程复制LOAD DATA INFILE语句时，从中继日志将待装载的文件提取到临时文件，然后将这些文件装入到表中。如果装载到主服务器上的文件很大，从服务器上的临时文件也很大。因此，建议使用该选项告诉从服务器将临时文件放到文件系统中有大量可用空间的目录下。在这种情况下，也可以使用--relay-log选项将中继日志放到该文件系统中，因为中继日志也很大。--slave-load-tmpdir应指向基于硬盘的文件系统，而非基于内存的文件系统：从服务器需要用临时文件在机器重启时用于复制LOAD DATA INFILE。系统启动过程中操作系统也不能清除该目录。

- --slave-net-timeout=*seconds*

放弃读之前从主服务器等候更多数据的秒数，考虑到连接中断和尝试重新连接。超时后立即开始第1次重试。由--master-connect-retry选项控制重试之间的间隔。

- --slave-skip-errors=[*err_code1*,*err_code2*,... | all]

通常情况，当出现错误时复制停止，这样给你一个机会手动解决数据中的不一致性问题。该选项告诉从服务器SQL线程当语句返回任何选项值中所列的错误时继续复制。

如果你不能完全理解为什么发生错误，则不要使用该选项。如果复制设置和客户程序中没有bug，并且MySQL自身也没有bug，应不会发生停止复制的错误。滥用该选项会使从服务器与主服务器不能保存同步，并且你找不到原因。

对于错误代码，你应使用从服务器错误日志中错误消息提供的编号和SHOW SLAVE STATUS的输出。服务器错误代码列于附录B：错误代码和消息。

你也可以(但不应)使用不推荐的all值忽略所有错误消息，不考虑所发生的错误。无需而言，如果使用该值，我们不能保证数据的完整性。在这种情况下，如果从服务器的数据与主服务器上的不相近请不要抱怨(或编写bug报告)。已经警告你了。

例如：

```
--slave-skip-errors=1062,1053
--slave-skip-errors=all
```

从服务器按下面评估--replicate-*规则，确定是否执行或忽视语句：

1. 是否有--replicate-do-db或--replicate-ignore-db规则？

- 有：测试--binlog-do-db和--binlog-ignore-db(参见5.11.3节，“二进制日志”)。测试结果是什么？

o 忽视语句：忽视并退出。

- o 许可语句：不立即执行语句。推迟决策；继续下一步。
- 没有：继续下一步。

2. 我们目前正执行保存的程序或函数吗？
   - 是：执行查询并退出。
   - 否：继续下一步。
3. 是否有--replicate-*-table规则？
   - 没有：执行查询并退出。
   - 有：继续下一步并开始按所示顺序评估表规则(首先是非通配规则，然后是通配规则)。只有待更新的表根据这些规则进行比较(INSERT INTO sales SELECT * FROM prices:只有sales根据这些规则进行比较)。如果要更新几个表(多表语句)，第1个匹配的表(匹配"do"或"ignore")获赢。也就是说，根据这些规则比较第1个表。然后，如果不能进行决策，根据这些规则比较第2个表等等。
4. 是否有--replicate-do-table规则？
   - 有：表匹配吗？
     - o 是：执行查询并退出。
     - o 否：继续下一步。
   - 没有：继续下一步。
5. 是否有--replicate-ignore-table规则？
   - 有：表匹配吗？
     - o 是：忽视查询并退出。
     - o 否：继续下一步。
   - 没有：继续下一步。
6. 是否有--replicate-wild-do-table规则？
   - 有：表匹配吗？
     - o 是：执行查询并退出。
     - o 否：继续下一步。
   - 没有：继续下一步。
7. 是否有--replicate-wild-ignore-table规则？
   - 有：表匹配吗？
     - o 是：忽视查询并退出。
     - o 否：继续下一步。

- 没有：继续下一步。

8. 没有匹配的--replicate-*-table规则。要根据这些规则测试其它表吗？

- 是：执行循环。

- 否：我们现在已经测试了所有待更新的表，结果不能匹配任何规则。是否有--replicate-do-table或--replicate-wild-do-table规则？

o 有：有"do"规则但不匹配。忽视查询并退出。

o 没有：执行查询并退出。

6.9. 复制FAQ

Q：如果主服务器正在运行并且不想停止主服务器，怎样配置一个从服务器？

A：有多种方法。如果你在某时间点做过主服务器备份并且记录了相应快照的二进制日志名和偏移量(通过SHOW MASTER STATUS命令的输出)，采用下面的步骤：

1. 确保从服务器分配了一个唯一的服务器ID号。

2. 在从服务器上执行下面的语句，为每个选项填入适当的值：

```
mysql> CHANGE MASTER TO
    ->     MASTER_HOST='master_host_name',
    ->     MASTER_USER='master_user_name',
    ->     MASTER_PASSWORD='master_pass',
    ->     MASTER_LOG_FILE='recorded_log_file_name',
    ->     MASTER_LOG_POS=recorded_log_position;
```

3. 在从服务器上执行START SLAVE语句。

如果你没有备份主服务器，这里是一个创建备份的快速程序。所有步骤都应该在主服务器主机上执行。

1. 发出该语句：

```
mysql> FLUSH TABLES WITH READ LOCK;
```

2. 仍然加锁时，执行该命令（或它的变体）：

```
shell> tar zcf /tmp/backup.tar.gz /var/lib/mysql
```

3. 发出该语句并且确保记录了以后用到的输出：

```
mysql>SHOW MASTER STATUS;
```

4. 释放锁：

```
mysql> UNLOCK TABLES;
```

一个可选择的方法是，转储主服务器的SQL来代替前面步骤中的二进制复制。要这样做，你可以在主服务器上使用**mysqldump --master-data**，以后装载SQL转储到到你的从服务器。然而，这比进行二进制复制速度慢。

不管你使用这两种方法中的那一个，当你有一个快照和记录了日志名与偏移量时，后来根据说明操作。你可以使用相同的快照建立多个从服务器。一旦你拥有主服务器的一个快照，可以等待创建一个从服务器，只要主服务器的二进制日志完整。两个能够等待的时间实际的限制是指在主服务器上保存二进制日志的可用硬盘空间和从服务器同步所用的时间。

你也可以使用LOAD DATA FROM MASTER。这是一个方便的语句，它传输一个快照到从服务器并且立即调整日志名和偏移量。将来，LOAD DATA FROM MASTER将成为创建从服务器的推荐方法。然而需要注意，它只工作在MyISAM 表上并且可能长时间持有读锁定。它并不象我们希望的那样高效率地执行。如果你有大表，执行FLUSH TABLES WITH READ LOCK语句后，这时首选方法仍然是在主服务器上制作二进制快照。

Q：从服务器需要始终连接到主服务器吗？

A：不，不需要。从服务器可以宕机或断开连接几个小时甚至几天，重新连接后获得更新信息。例如，你可以在通过拨号的链接上设置主服务器/从服务器关系，其中只是偶尔短时间内进行连接。这意味着，在任何给定时间，从服务器不能保证与主服务器同步除非你执行某些特殊的方法。将来，我们将使用选项来阻塞主服务器直到有一个从服务器同步。

Q：我怎样知道从服务器与主服务器的最新比较? 换句话说，我怎样知道从服务器复制的最后一个查询的日期？

A：你可以查看SHOW SLAVE STATUS语句的Seconds_Behind_Master列的结果。参见6.3节，“复制实施细节”。

当从服务器SQL线程执行从主服务器读取的事件时，它根据事件时间戳修改自己的时间（这是TIMESTAMP能够很好复制的原因）。在SHOW PROCESSLIST语句输出的Time列内，为从服务器SQL线程显示的秒数是最后一个复制事件的时间戳和从服务器主机的实际时间之间相差的秒数。你可以使用它来确定最后一个复制事件的日期。注意，如果你的从服务器与主服务器连接断开一个小时，然后重新连接，在SHOW PROCESSLIST结果中，你可以立即看到从服务器SQL线程的Time值为3600。这可能是因为从服务器执行的语句是一个一小时之前的。

Q：我怎样强制主服务器阻塞更新直到从服务器同步？

A：使用下面的步骤：

1. 在主服务器上，执行这些语句：

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SHOW MASTER STATUS;
```

记录SHOW语句的输出的日志名和偏移量。这些是复制坐标。

2. 在从服务器上，发出下面的语句，其中Master_POS_WAIT()函数的参量是前面步骤中的得到的复制坐标值：

```
mysql> SELECT MASTER_POS_WAIT('log_name', log_offset);
```

SELECT语句阻塞直到从服务器达到指定的日志文件和偏移量。此时，从服务器与主服务器同步，语句返回。

3. 在主服务器上，发出下面的语句允许主服务器重新开始处理更新：

```
mysql> UNLOCK TABLES;
```

Q：当设置双向复制时我应该知道发出那些语句？

A：MySQL复制目前不支持主服务器和从服务器之间的任何锁定协议来保证分布式(跨服务器)更新的原子性。换句话说，这样做是可能的：客户A根据协作-主服务器1更新，同时，在它传给协作-主服务器2之前，客户B能够根据协作-主服务器2更新，这样客户A的更新与它在协作-主服务器1的更新不同。这样，当客户A根据协作-主服务器2更新时，它产生的表与在协作-主服务器1上的不同，即使所有根据协作-主服务器2的更新已经传过来。这意味着，在双向复制关系中，你不应该把两个服务器串连在一起，除非你确信任何顺序的更新是安全的，或者除非你在客户端代码中注意怎样避免更新顺序错误。

你还必须认识到从更新角度，双向复制实际上并不能显著地提高性能（或者根本不能提高性能）。两个服务器都需要做相同数量的更新，如同在一个服务器做的那样。唯一的差别是锁竞争要少，这因为源于另一个服务器的更新在一个从线程中序列化。即使这个益处可能被网络延迟抵消。

Q：怎样通过复制来提高系统的性能？

A：你应将一个服务器设置为主服务器并且将所有写指向该服务器。然后根据预算配置尽可能多的从服务器以及栈空间，并且在主服务器和从服务器之间分发读取操作。你也可以用--skip-innodb、--skip-bdb、--low-priority-updates以及--delay-key-write=ALL选项启动从服务器，以便在从服务器端提高速度。在这种情况下，为了提高速度，从服务器使用非事务MyISAM表来代替InnoDB和BDB表。

Q：为了使用高性能的复制，我应该在自己的应用程序中怎样准备客户端代码？

A：如果你的代码中数据库访问部分已经正确地模块化，应该能够平滑和容易地转换为在复制步骤中运行的代码。仅需要更改数据库访问执行部分，以便发送所有的写操作到主服务器，以及发送读操作到主服务器或某个从服务器。如果你的代码没有这个级别，设置一个复制系统以便清除。应先通过下面的函数创建一个包装库或模块：

- safe_writer_connect()
- safe_reader_connect()
- safe_reader_statement()
- safe_writer_statement()

每个函数名的safe_意味着函数比较小心地处理所有错误。你可以使用不同名的函数。重要是对于读连接、写连接、读和写有一个统一的接口。

然后，你应该转换客户端代码使用包装库。刚开始这可能是痛苦和恐慌的过程，但从长远来看是值得的。使用刚才讨论的方法的所有应用程序都能够利用主服务器/从服务器配置的优越性，即使是含有多个从服务器的配置。代码非常容易维护，并且添加排错选项也很容易。你仅需要修改一两个函数；例如，记录每个语句执行的时间，或者你的上千个语句中哪个语句发生了错误。

如果你已经编写了许多代码，你可能想使用**replace**工具自动进行转换，该工具随标准MySQL一起发布，或可以自己编写转换脚本。理想情况，你的代码使用一致的程序转换风格。否则，可能最好重新编写代码，或者至少手工对其进行规则化以使用一致的风格。

Q：MySQL复制能够何时和多大程度提高系统性能？

A：MySQL复制对于频繁读和频繁写的系统具有最大好处。理论上，通过使用单个主服务器/多从服务器设置，可以通过添加更多的从服务器来扩充系统，直到用完网络带宽，或者你的更新负载已经增长到主服务器不能处理的点。

在获得的收益开始吃平之前，为了确定可以有多少从服务器，以及可以将你的站点的性能提高多少，需要知道查询模式，并且要通过基准测试并根据经验确定一个典型的主服务器和从服务器中的读取（每秒钟读取量，或者max_reads）吞吐量和写（max_writes）吞吐量的关系。通过一个假设的带有复制的系统，本例给出了一个非常简单的计算结果。

假设系统负载包括10%的写和90%的读取，并且我们通过基准测试确定max_reads是1200 –2 × max_writes。换句话说，如果没有写操作，系统每秒可以进行1,200次读取操作，平均写操作是平均读操作所用时间的两倍，并且关系是线性的。我们假定主服务器和每个从服务器具有相同的性能，并且我们有一个主服务器和*N*个从服务器。那么，对于每个服务器（主服务器或从服务器），我们有：

reads = 1200 – 2 × writes

reads = 9 × writes / (*N* + 1) (读取是分离的, 但是写入所有服务器)

9 × writes / (*N* + 1) + 2 × writes = 1200

writes = 1200 / (2 + 9/(*N*+1))

最后的等式表明了*N*个从服务器的最大写操作数，假设最大可能的读取速率是每分钟1,200次，读操作与写操作的比率是9。

如上分析可以得到下面的结论：

- 如果*N* = 0（这表明没有复制），系统每秒可以处理大约1200/11 = 109个写操作。
- 如果*N* = 1，每秒得到184个写操作。
- 如果*N* = 8，每秒得到400个写操作。
- 如果*N* = 17，每秒得到480个写操作。
- 最后，当 *N* 趋于无穷大（以及我们预算的负无穷大）时，可以得到非常接近每秒600个写操作，系统吞吐量增加将近5.5倍。然而，如果只用8个服务器，增加接近4倍。

请注意，这些计算假设网络带宽无穷大并忽略掉了其它一些因素，那些因素可能对系统产生重要的影响。在许多情况下，不能执行与刚才类似的计算，即如果添加*N*台复制从服务器，应该准确预报系统将发生哪些影响。回答下面的问题应能够帮助你确定复制是否和在多大程度上能够提高系统的性能：

- 系统上的读取/写比例是什么?
- 如果减少读取操作，一个服务器可以多处理多少写负载？
- 网络带宽可满足多少从服务器的需求?

Q：如何使用复制来提供冗余/高可用性?

A：利用目前的可用特性，必须设置一个主服务器和一个从服务器（或多个从服务器），以及写一个脚本来监视主服务器是否启动。如果主服务器失败，通知应用程序和从服务器切换主服务器。下面是一些建议：

- 告知从服务器更改其主服务器，使用CHANGE MASTER TO语句。
- 通知应用程序主服务器位置的一个很好的方法是对主服务器提供动态DNS入口。用bind可以使用nsupdate动态更新DNS。
- 应该用--logs-bin选项而不用 --logs-slave-updates选项运行从服务器。这样，一旦你在其它从服务器上发出STOP SLAVE; RESET MASTER, 以及CHANGE MASTER TO语句，该从服务器可以切换为主服务器。例如，假设有下面的设置：

```
·                    WC
·                     \
```

```
·                          v
·                  WC----> M
·                        / | \
·                       /  |  \
·                      v   v   v
·                     S1   S2  S3
```

M代表主服务器，**S**代表从服务器，**WC**代表发出数据库写和读取操作的客户；只发出数据库读取操作的客户没有给出，因为它们不需要切换。**S1**、**S2**以及**S3**是从服务器，用--logs-bin选项而没有用--logs-slave-updates运行。因为从服务器收到的主服务器的更新没有记录在二进制日志中，除非指定 --logs-slave-updates选项，每个从服务器上的二进制日志是空的。如果因为某些原因**M** 变得不可用，你可以选取一个从服务器变为新的主服务器。例如，如果你选取了**S1**，所有**WC**应该重新指向**S1**和**S2**，并且**S3**然后应从**S1**复制。

确保所有从服务器已经处理了中继日志中的所有语句。在每个从服务器上，发出STOP SLAVE IO_THREAD语句，然后检查SHOW PROCESSLIST语句的输出，直到你看到Has read all relay log。当所有从服务器都执行完这些，它们可以被重新配置为一个新的设置。在被提升为主服务器的从服务器**S1**上，发出STOP SLAVE和RESET MASTER语句。

在其它从服务器**S2**和**S3**上，使用STOP SLAVE和CHANGE MASTER TO MASTER_HOST='S1'（其中'S1'表示**S1**实际的主机名）。为CHANGE MASTER添加关于从**S2**或**S3**如何连接到**S1**的所有信息（*user*、*password*、*port*）。在CHANGE MASTER命令中，不需要指定从其读取的**S1**的二进制日志名或二进制日志位置：我们知道它是第1个二进制日志，位置是4，这是CHANGE MASTER命令的默认值。最后，在**S2**和**S3**上使用START SLAVE 命令。

然后，指示所有**WC** 把它们的语句指向**S1**。此后，**WC**发出的所有发送到**S1**的更新语句被写入**S1**的二进制日志，**S1**则包含**M**死掉之后的发送到 **S1**的每一个更新语句。

结果是下面的配置：

```
       WC
      /
      |
 WC   |  M(unavailable)
  \   |
   \  |
    v v
     S1<--S2  S3
      ^       |
      +-------+
```

当 **M**重新启动后，你必须在**M**上发出相同的CHANGE MASTER语句，与在**S2**和**S3**上发出的语句一样，以便**M**变为**S1**的从服务器并且恢复在它宕机后丢失的所有**WC**写操作。要把 **M** 再次作为主服务器（例如，因为它是功能最强的机器），使用前面的步骤，好像**S1**不可用并且**M**变为一个新的主服务器一样。在这个过程中，在**S1**、**S2**以及**S3**作为**M**的从服务器之前，不要忘记在**M**上运行RESET MASTER。否则，它们可能拾取**M**变得不可用之前的旧**WC**写操作。

我们目前正在MySQL集成自动主服务器选择系统，但在准备好之前，你必须创建自己的监控工具。

6.10. 复制故障诊断与排除

如果你遵从了上述说明，复制设置仍然不工作，首先检查下面各项：

- 检查错误日志的消息。许多用户遇到问题后没有及时地这样做而浪费了时间。

- 主服务器记录到了二进制日志？用SHOW MASTER STATUS检查。如果已经记录，Position应为非零。如果没有记录，确认正用log-bin和server-id选项运行主服务器。

- 是否从服务器在运行？使用SHOWSHOW SLAVE STATUS检查是否slave_IO_Running和slave_SQL_Running的值均为Yes。如果不是，验证当启动从服务器时使用的选项。

- 如果从服务器正在运行，建立了与主服务器的连接吗？使用SHOW PROCESSLIST，找出I/O和SQL线程并检查它们的State列看它们如何显示。参见6.3节，“复制实施细节”。如果I/O线程状态为Connecting to master，验证主服务器上复制用户的权限、主服务器主机名、DNS设置，是否主服务器真正在运行，以及是否可以从从属服务器访问。

- 如果从服务器以前在运行但是现在已经停止，原因通常是在主服务器上成功的部分语句在从服务器上失败了。如果你正确快照了主服务器，并且从来没有不通过服务器线程修改从服务器上的数据，这种现象不应发生。如果发生，应为一个bug或你遇到了一个6.7节，“复制特性和已知问题” 描述的已知的复制限制。如果是一个bug，参见6.11节，“通报复制缺陷”查阅如何通报的说明。

- 如果某个在主服务器上成功的语句拒绝在从服务器上运行，并且不能执行完全的数据库重新同步(即删除从服务器的数据库并从主服务器复制新的快照)，尝试：

1. 确定是否从服务器的表与主服务器的不同。尽力了解发生的原因。然后让从服务器的表与主服务器的一样并运行START SLAVE。

2. 如果前面的步骤不工作或不适合，尽力了解手动更新是否安全(如果需要)，然后忽视来自主服务器的下一个语句。

3. 如果你确定可以跳过来自主服务器的下一个语句，执行下面的语句：

4. `mysql> SET GLOBAL SQL_slave_SKIP_COUNTER = n;`

5. `mysql> START SLAVE;`

如果来自主服务器的下一个语句不使用AUTO_INCREMENT或LAST_INSERT_ID()，*n* 值应为1。否则，值应为2。使用AUTO_INCREMENT或LAST_INSERT_ID()的语句使用值2的原因是它们从主服务器的二进制日志中取两个事件。

6. 如果你确保从服务器启动时完好地与主服务器同步，并且没有更新从服务器线程之外的表，则大概诧异是由于bug。如果你正运行最近的版本，请通报该问题。如果你正运行旧版本MySQL，尽力升级到最新的产品版本。

6.11. 通报复制缺陷

如果你确定没有用户错误，但复制仍然不工作或不稳定，则是向我们发送bug通报的时候了。我们需要尽可能从你那儿获得更多的信息已跟踪bug。请花一些时间和努力编写一份好的bug通报。

如果你有一个重复的测试案例来说明bug，请把它输入我们的bug数据库，位置为http://bugs.mysql.com/。如果你有一个“phantom”问题(不能按照期望进行复制)，则使用下面的程序：

1. 确认未包括用户错误。例如，如果你不用从服务器线程来更新从服务器，数据将不同步，并且会遇到唯一的键值违背更新。在这种情况下，从服务器线程停止并等待你手动清理表使它们同步。这不是复制问题。这是一个外部接口问题造成复制失败。

2. 用--logs-slave-updates和--logs-bin选项运行从服务器。这些选项使从服务器将从主服务器接收的更新记入自己的二进制日志。

3. 重新设置复制状态之前保存所有的证据。如果我们没有信息或只有粗略的信息，则难以或不可能跟踪问题。应搜集的证据为：

- 所有主服务器的二进制日志
- 所有从服务器的二进制日志
- 你发现问题时主服务器的SHOW MASTER STATUS的输出
- 你发现问题时主服务器的SHOW SLAVE STATUS的输出
- 主服务器和从服务器的错误日志

4. 使用**mysqlbinlog**检查二进制日志。下面命令应有助于发现有问题的查询，例如：

```
shell> mysqlbinlog -j pos_from_slave_status \
            /path/to/log_from_slave_status | head
```

搜集了问题的证据后，首先作为一个测试案例隔离开。然后将问题输入我们的bug数据库，位置为http://bugs.mysql.com/，应提供尽可能多的信息。

6.12. 多服务器复制中的Auto-Increment

当将多个服务器配置为复制主服务器时，使用auto_increment时应采取特殊步骤以防止键值冲突，否则插入行时多个主服务器会试图使用相同的auto_increment值。

服务器变量auto_increment_increment和auto_increment_offset可以帮助协调多主服务器复制和AUTO_INCREMENT列。每个变量有一个默认的(并且是最小的)值1，最大值为65,535。

将这些变量设置为非冲突的值，当在同一个表主插入新行时，多主服务器配置主的服务器将不会与AUTO_INCREMENT值冲突。

这两个变量这样影响AUTO_INCREMENT列：

- auto_increment_increment控制列值增加的间隔。例如：

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 1     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoinc1 (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> SET @auto_increment_increment=10;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set (0.01 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
|   1 |
|  11 |
|  21 |
|  31 |
+-----+
4 rows in set (0.00 sec)
```

(这里注明如何使用SHOW VARIABLES以获得这些变量的当前值）。

- auto_increment_offset确定AUTO_INCREMENT列值的起点。影响到在复制设置主可以有多少主服务器(例如将该值设置为10表示设置可以支持10个服务器)。

考虑下面的命令，假定在前面所示示例中的相同的会话中执行这些命令:

```
mysql> SET @auto_increment_offset=5;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
```

```
| auto_increment_increment | 10    |
| auto_increment_offset    | 5     |
+--------------------------+-------+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoinc2 (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO autoinc2 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT col FROM autoinc2;
+-----+
| col |
+-----+
|   5 |
|  15 |
|  25 |
|  35 |
+-----+
4 rows in set (0.02 sec)
```

详细信息参见5.3.3节，“服务器系统变量”。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第7章：优化

目录

7.1. 优化概述

7.1.1. MySQL设计局限与折衷
7.1.2. 为可移植性设计应用程序
7.1.3. 我们已将MySQL用在何处？
7.1.4. MySQL基准套件
7.1.5. 使用自己的基准

7.2. 优化SELECT语句和其它查询

7.2.1. EXPLAIN语法（获取SELECT相关信息）
7.2.2. 估计查询性能
7.2.3. SELECT查询的速度
7.2.4. MySQL怎样优化WHERE子句
7.2.5. 范围优化
7.2.6. 索引合并优化
7.2.7. MySQL如何优化IS NULL
7.2.8. MySQL如何优化DISTINCT
7.2.9. MySQL如何优化LEFT JOIN和RIGHT JOIN
7.2.10. MySQL如何优化嵌套Join
7.2.11. MySQL如何简化外部联合
7.2.12. MySQL如何优化ORDER BY
7.2.13. MySQL如何优化GROUP BY
7.2.14. MySQL如何优化LIMIT
7.2.15. 如何避免表扫描
7.2.16. INSERT语句的速度
7.2.17. UPDATE语句的速度
7.2.18. DELETE语句的速度
7.2.19. 其它优化技巧

7.3. 锁定事宜

7.3.1. 锁定方法
7.3.2. 表锁定事宜

7.4. 优化数据库结构

7.4.1. 设计选择
7.4.2. 使你的数据尽可能小
7.4.3. 列索引
7.4.4. 多列索引
7.4.5. MySQL如何使用索引
7.4.6. MyISAM键高速缓冲
7.4.7. MyISAM索引统计集合
7.4.8. MySQL如何计算打开的表
7.4.9. MySQL如何打开和关闭表
7.4.10. 在同一个数据库中创建多个表的缺陷

7.5. 优化MySQL服务器

7.5.1. 系统因素和启动参数的调节
7.5.2. 调节服务器参数
7.5.3. 控制查询优化器的性能
7.5.4. 编译和链接怎样影响MySQL的速度
7.5.5. MySQL如何使用内存
7.5.6. MySQL如何使用DNS

7.6. 磁盘事宜

7.6.1. 使用符号链接

优化是一个复杂的任务，因为最终要求了解整个待优化的系统。尽管可以进行局部优化而不需要了解系统或应用程序，为了优化得更好，你必须知道更多的信息。

本章解释并给出不同的优化MySQL的方法示例。但要记住总有一些其它方法使系统更快，尽管需要更多的工作。

7.1. 优化概述

7.1.1. MySQL设计局限与折衷
7.1.2. 为可移植性设计应用程序
7.1.3. 我们已将MySQL用在何处？
7.1.4. MySQL基准套件
7.1.5. 使用自己的基准

使一个系统更快的最重要因素当然是基本设计。此外，还需要知道系统正做什么样的事情，以及瓶颈是什么。

最常见的系统瓶颈是:

- 磁盘搜索。需要花时间从磁盘上找到一个数据，用在现代磁盘的平均时间通常小于10ms，因此理论上我们能够每秒大约搜索1000次。这个时间在新磁盘上提高不大并且很难为一个表进行优化。优化它的方法是将数据分布在多个磁盘上。
- 磁盘读/写。当磁盘放入正确位置后，我们需要从中读取数据。对于现代的磁盘，一个磁盘至少传输10-20Mb/s的吞吐。这比搜索要容易优化，因为你能从多个磁盘并行地读。
- CPU周期。我们将数据读入内存后，需要对它进行处理以获得我们需要的结果。表相对于内存较小是最常见的限制因素。但是对于小表，速度通常不成问题。

· 内存带宽。当CPU需要的数据超出CPU缓存时，主缓存带宽就成为内存的一个瓶颈。这在大多数系统正是一个不常见的瓶颈但是你应该知道它。

7.1.1. MySQL设计局限与折衷

当使用MyISAM存储引擎时，MySQL使用极快速的表锁定，以便允许多次读或一次写。使用该存储引擎的最大问题出现在同一个表中进行混合稳定数据流更新与慢速选择。如果这只是某些表的问题，你可以使用另一个存储引擎。参见第15章：存储引擎和表类型。

MySQL可以使用事务表和非事务表。为了更容易地让非事务表顺利工作(如果出现问题不能回滚)，MySQL采用下述规则。请注意这些规则只适用于不运行在严格模式下或为INSERT或UPDATE使用IGNORE规定程序时。

· 所有列有默认值。请注意当运行在严格SQL模式(包括TRADITIONAL SQL模式)时，必须为NOT NULL列指定默认值。

· 如果向列内插入不合适的或超出范围的值，MySQL将该列设定为"最好的可能的值"，而不是报告错误。

对于数字值，为0、可能的最小值或最大值。对于字符串，为空字符串或列内可以保存的字符串。请注意当运行在严格模式或TRADITIONAL SQL模式时该行为不 适用。

· 所有表达式的计算结果返回一个表示错误状况的信号。例如，1/0返回NULL。(使用ERROR_FOR_DIVISION_BY_ZERO SQL模式可以更改该行为)。

如果正使用非事务表，不应该使用MySQL来检查列的内容。一般情况，最安全的(通常是最快的)方法径是让应用程序确保只向数据库传递合法值。

相关详细信息参见1.8.6节，“MySQL处理约束的方式”和13.2.4节，“INSERT语法”或5.3.2节，“SQL服务器模式”。

7.1.2. 为可移植性设计应用程序

因为不同SQL服务器实现了标准SQL的不同部分，需要花功夫来编写可移植的SQL应用程序。对很简单的选择/插入，很容易实现移植，但是需要的功能越多则越困难。如果想要应用程序对很多数据库系统都快，它变得更难！

为了使一个复杂应用程序可移植，你需要选择它应该工作的SQL服务器，并确定这些服务器支持什么特性。

所有数据库都有一些弱点。这就是它们不同的设计折衷导致的不同行为。

可以使用**MySQL**的**crash-me**程序来找出能用于数据库服务器选择的函数、类型和限制。**crash-me**并不能找出所有的特性，但是其广度仍然很合理，可以进行大约450个测试。

crash-me可以提供的一种类型的信息的例子：如果想要使用Informix或DB2，不应该使用超过18个字符的列名。

crash-me程序和**MySQL**基准程序是独立于数据库的。通过观察它们是如何编写的，编可以知道必须为编写独立于数据库的应用程序做什么。基准本身可在**MySQL**源码分发的“`sql-bench`”目录下找到。它们用DBI数据库接口以Perl写成。使用DBI本身即可以解决部分移植性问题，因为它提供与数据库无关的的存取方法。

如果你为数据库的独立性而努力，需要很好地了解每个SQL服务器的瓶颈。例如，**MySQL**在检索和更新MyISAM表记录方面很快，但是在同一个表上混合慢速读者和写者方面有一个问题。另一方面，当你试图访问最近更新了(直到它们被刷新到磁盘上)的行时，在Oracle中有一个很大的问题。事务数据库总的来说在从记录文件表中生成总结表方面不是很好，因为在这种情况下，行锁定几乎没有用。

为了使应用程序“确实”独立于数据库，需要定义一个容易扩展的接口，用它可操纵数据。因为C++在大多数系统上可以适用，使用数据库的一个C++ 类接口是有意义的。

如果你使用某个数据库特定的功能(例如**MySQL**专用的REPLACE语句)，应该为SQL服务器编码一个方法以实现同样的功能。尽管慢些，但确允许其它服务器执行同样的任务。

用**MySQL**，可以使用/*! */语法把**MySQL**特定的关键词加到查询中。在/**/中的代码将被其它大多数SQL服务器视为注释(并被忽略)。

如果高性能真的比准确性更重要，就像在一些web应用程序那样，一种可行的方法是创建一个应用层，缓存所有的结果以便得到更高的性能。通过只是让旧的结果在短时间后‘过期’，能保持缓存合理地刷新。这在极高负载的情况下是相当不错的，在此情况下，能动态地增加缓存并且设定较高的过期时限直到一切恢复正常。

在这种情况下，表创建信息应该包含缓存初始大小和表刷新频率等信息。

实施应用程序缓存的一种方法是使用MySQL查询缓存。启用查询缓存后，服务器可以确定是否可以重新使用查询结果。这样简化了你的应用程序。参见5.13节，“MySQL查询高速缓冲”。

7.1.3. 我们已将MySQL用在何处？

该节描述了Mysql的早期应用程序。

在MySQL最初开发期间，MySQL的功能适合大多数客户。MySQL为瑞典的一些最大的零售商处理数据仓库。

我们从所有商店得到所有红利卡交易的每周总结，并且我们期望为所有店主提供有用的信息以帮助他们得出他们的广告战如何影响他们的顾客。

数据是相当巨量的(大约每月7百万宗交易总结)并且我们保存4-10年来的数据需要呈现给用户。我们每周从顾客那里得到请求，他们想要"立刻"访问来自该数据的新报告。

我们通过每月将所有信息存储在压缩的"交易"表中来解决它。我们有一套简单的宏/脚本用来生成来自交易表的不同条件(产品组、顾客id，商店...)的总结表。报告是由一个进行语法分析网页的小perl脚本动态生成的网页，在脚本中执行SQL语句并且插入结果。我们很想使用PHP或**mod_perl**，但是那时它们还不可用。

对图形数据，我们用C语言编写了一个简单的工具，它能基于那些结果处理SQL查询结果并生成GIF图形。该工具也从分析Web网页的perl脚本中动态地执行。

在大多数情况下，一个新的报告通过简单地复制一个现有脚本并且修改其中的SQL查询来完成。在一些情况下，我们将需要把更多的列加到一个现有的总结表中或产生一个新的，但是这也相当简单，因为我们在磁盘上保存所有交易表。（目前我们大约有50G的交易表和200G的其它顾客数据)。

我们也让我们的顾客直接用ODBC访问总结表以便高级用户能自己用这些数据进行试验。

该系统工作得很好，我们可以毫无问题地用很适度的Sun Ultra SPARC工作站硬件(2x200MHz)来处理数据。该系统被逐步移植到了Linux中。

7.1.4. MySQL基准套件

本节应该包含**MySQL**基准套件(和**crash-me**)的技术描述，但是该描述还没写成。目前，你可以通过在MySQL源码分发中的`"sql-bench"`目录下的代码和结果了解基准套件是如何工作的。

通过基准用户可以了解一个给定的SQL实现在哪方面执行得很好或很糟糕。

注意，这个基准是单线程的，它可以测量操作执行的最小时间。我们计划将来在基准套件中添加多线程测试。

要使用基准套件，必须满足下面的要求：

- 基准套件随MySQL源码分发提供。可以从http://dev.mysql.com/downloads/下载分发，或者使用当前的开发源码树(参见2.8.3节，"从开发源码树安装")。

- 基准脚本用Perl编写而成，使用Perl DBI模块访问数据库服务器，因此必须安装DBI。还需要为每个待测试的服务器提供服务器专用DBD驱动程序。例如，要测试MySQL、PostgreSQL和DB2，必须安装DBD::mysql、DBD::Pg和DBD::DB2模块。参见2.13节，"Perl安装注意事项"。

获得MySQL源码分发后，可以在sql-bench目录找到基准套件。要运行基准测试，应构建MySQL，然后进入sql-bench目录并执行run-all-tests脚本：

shell> **cd sql-bench**

shell> **perl run-all-tests --server=***server_name*

*server_name*是一个支持的服务器。要获得所有选项和支持的服务器，调用命令：

```
shell> perl run-all-tests --help
```

crash-me脚本也位于sql-bench目录。**crash-me**尝试通过实际运行查询确定数据库支持的特性以及其功能和限制。例如，它确定：

- 支持什么列类型
- 支持多少索引
- 支持什么函数
- 查询可以多大
- VARCHAR列可以多大

关于一些可移植的基准程序的例子，参见MySQL基准套件。请参见7.1.4节，“MySQL基准套件”。可以利用这个套件的任何程序并且根据你的需要修改它。通过这样做，可以尝试不同的问题的解决方案并测试哪一个是最好的解决方案。

另一个免费基准套件是开放源码数据库基准套件，参见http://osdb.sourceforge.net/。

在系统负载繁重时出现一些问题是很普遍的，并且很多客户已经与我们联系了，他们在生产系统中有一个(测试)系统并且有负载问题。大多数情况下，性能问题经证明是与基本数据库设计有关的问题(例如，表扫描在高负载时表现不好)或操作系统或库问题。如果系统已经不在生产系统中，它们大多数将很容易修正。

为了避免这样的问题，应该把工作重点放在在可能最坏的负载下测试你的整个应用程序。你可以使用Super Smack。该工具可以从http://jeremy.zawodny.com/mysql/super-smack/获得。正如它的名字所建议，它可以根据你的需要提供合理的系统，因此确保只用于你的开发系统。

7.2. 优化SELECT语句和其它查询

7.2.1. EXPLAIN语法（获取SELECT相关信息）
7.2.2. 估计查询性能
7.2.3. SELECT查询的速度
7.2.4. MySQL怎样优化WHERE子句
7.2.5. 范围优化
7.2.6. 索引合并优化
7.2.7. MySQL如何优化IS NULL
7.2.8. MySQL如何优化DISTINCT
7.2.9. MySQL如何优化LEFT JOIN和RIGHT JOIN
7.2.10. MySQL如何优化嵌套Join
7.2.11. MySQL如何简化外部联合
7.2.12. MySQL如何优化ORDER BY
7.2.13. MySQL如何优化GROUP BY
7.2.14. MySQL如何优化LIMIT
7.2.15. 如何避免表扫描
7.2.16. INSERT语句的速度
7.2.17. UPDATE语句的速度
7.2.18. DELETE语句的速度
7.2.19. 其它优化技巧

首先，影响所有语句的一个因素是：你的许可设置得越复杂，所需要的开销越多。

执行GRANT语句时使用简单的许可，当客户执行语句时，可以使MySQL降低许可检查开销。例如，如果未授予任何表级或列级权限，服务器不需要检查tables_priv和columns_priv表的内容。同样地，如果不对任何 账户进行限制，服务器不需要对资源进行统计。如果查询量很高，可以花一些时间使用简化的授权结构来降低许可检查开销。

如果你的问题是与具体MySQL表达式或函数有关，可以使用**mysql**客户程序所带的BENCHMARK()函数执行定时测试。其语法为BENCHMARK(*loop_count*,*expression*)。例如：

```
mysql> SELECT BENCHMARK(1000000,1+1);
+------------------------+
| BENCHMARK(1000000,1+1) |
+------------------------+
|                      0 |
+------------------------+
1 row in set (0.32 sec)
```

上面结果在PentiumII 400MHz系统上获得。它显示**MySQL**在该系统上在0.32秒内可以执行1,000,000个简单的+表达式运算。

所有**MySQL**函数应该被高度优化，但是总有可能有一些例外。BENCHMARK()是一个找出是否查询有问题的优秀的工具。

7.2.1. EXPLAIN语法（获取SELECT相关信息）

```
EXPLAIN tbl_name
```

或：

```
EXPLAIN [EXTENDED] SELECT select_options
```

EXPLAIN语句可以用作DESCRIBE的一个同义词，或获得关于MySQL如何执行SELECT语句的信息：

- EXPLAIN *tbl_name*是DESCRIBE *tbl_name*或SHOW COLUMNS FROM *tbl_name*的一个同义词。
- 如果在SELECT语句前放上关键词EXPLAIN，**MySQL**将解释它如何处理SELECT，提供有关表如何联接和联接的次序。

该节解释EXPLAIN的第2个用法。

借助于EXPLAIN，可以知道什么时候必须为表加入索引以得到一个使用索引来寻找记录的更快的SELECT。

如果由于使用不正确的索引出现了问题，应运行ANALYZE TABLE更新表的统计（例如关键字集的势），这样会影响优化器进行的选择。参见13.5.2.1节，“ANALYZE TABLE语法”。

还可以知道优化器是否以一个最佳次序联接表。为了强制优化器让一个SELECT语句按照表命名顺序的联接次序，语句应以STRAIGHT_JOIN而不只是SELECT开头。

EXPLAIN为用于SELECT语句中的每个表返回一行信息。表以它们在处理查询过程中将被MySQL读入的顺序被列出。**MySQL**用一遍扫描多次联接（*single-sweep multi-join*）的方式解决所有联接。这意味着**MySQL**从第一个表中读一行，然后找到在第二个表中的一个匹配行，然后在第3个表中等等。当所有的表处理完后，它输出选中的列并且返回表清单直到找到一个有更多的匹配行的表。从该表读入下一行并继续处理下一个表。

当使用EXTENDED关键字时，EXPLAIN产生附加信息，可以用SHOW WARNINGS浏览。该信息显示优化器限定SELECT语句中的表和列名，重写并且执行优化规则后SELECT语句是什么样子，并且还可能包括优化过程的其它注解。

EXPLAIN的每个输出行提供一个表的相关信息，并且每个行包括下面的列：

· id

SELECT识别符。这是SELECT的查询序列号。

· select_type

SELECT类型，可以为以下任何一种：

o SIMPLE

简单SELECT(不使用UNION或子查询)

o PRIMARY

最外面的SELECT

o UNION

UNION中的第二个或后面的SELECT语句

o DEPENDENT UNION

UNION中的第二个或后面的SELECT语句，取决于外面的查询

o UNION RESULT

UNION的结果。

o SUBQUERY

子查询中的第一个SELECT

o DEPENDENT SUBQUERY

子查询中的第一个SELECT，取决于外面的查询

o DERIVED

导出表的SELECT(FROM子句的子查询)

· table

输出的行所引用的表。

· type

联接类型。下面给出各种联接类型，按照从最佳类型到最坏类型进行排序：

o system

表仅有一行(=系统表)。这是const联接类型的一个特例。

- const

表最多有一个匹配行，它将在查询开始时被读取。因为仅有一行，在这行的列值可被优化器剩余部分认为是常数。const表很快，因为它们只读取一次！

const用于用常数值比较PRIMARY KEY或UNIQUE索引的所有部分时。在下面的查询中，*tbl_name*可以用于const表：

```
SELECT * from tbl_name WHERE primary_key=1;


SELECT * from tbl_name
WHERE primary_key_part1=1和 primary_key_part2=2;
```

- eq_ref

对于每个来自于前面的表的行组合，从该表中读取一行。这可能是最好的联接类型，除了const类型。它用在一个索引的所有部分被联接使用并且索引是UNIQUE或PRIMARY KEY。

eq_ref可以用于使用= 操作符比较的带索引的列。比较值可以为常量或一个使用在该表前面所读取的表的列的表达式。

在下面的例子中，MySQL可以使用eq_ref联接来处理*ref_tables*：

```
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

对于每个来自于前面的表的行组合，所有有匹配索引值的行将从这张表中读取。如果联接只使用键的最左边的前缀，或如果键不是UNIQUE或PRIMARY KEY（换句话说，如果联接不能基于关键字选择单个行的话），则使用ref。如果使用的键仅仅匹配少量行，该联接类型是不错的。

ref可以用于使用=或<=>操作符的带索引的列。

在下面的例子中，MySQL可以使用ref联接来处理*ref_tables*：

```
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

该联接类型如同ref，但是添加了MySQL可以专门搜索包含NULL值的行。在解决子查询中经常使用该联接类型的优化。

在下面的例子中，MySQL可以使用ref_or_null联接来处理*ref_tables*：

```
SELECT * FROM ref_table
WHERE key_column=expr OR key_column IS NULL;
```

参见7.2.7节，“MySQL如何优化IS NULL”。

- index_merge

该联接类型表示使用了索引合并优化方法。在这种情况下，key列包含了使用的索引的清单，key_len包含了使用的索引的最长的关键元素。详细信息参见7.2.6节，“索引合并优化”。

- unique_subquery

该类型替换了下面形式的IN子查询的ref：

```
value IN (SELECT primary_key FROM single_table WHERE some_expr)
```

unique_subquery是一个索引查找函数，可以完全替换子查询，效率更高。

- index_subquery

该联接类型类似于unique_subquery。可以替换IN子查询，但只适合下列形式的子查询中的非唯一索引：

```
value IN (SELECT key_column FROM single_table WHERE some_expr)
```

- range

只检索给定范围的行，使用一个索引来选择行。key列显示使用了哪个索引。key_len包含所使用索引的最长关键元素。在该类型中ref列为NULL。

当使用=、<>、>、>=、<、<=、IS NULL、<=>、BETWEEN或者IN操作符，用常量比较关键字列时，可以使用range：

```
SELECT * FROM tbl_name
WHERE key_column = 10;

SELECT * FROM tbl_name
WHERE key_column BETWEEN 10 and 20;

SELECT * FROM tbl_name
WHERE key_column IN (10,20,30);

SELECT * FROM tbl_name
WHERE key_part1= 10 AND key_part2 IN (10,20,30);
```

- index

该联接类型与ALL相同，除了只有索引树被扫描。这通常比ALL快，因为索引文件通常比数据文件小。

当查询只使用作为单索引一部分的列时，MySQL可以使用该联接类型。

o ALL

对于每个来自于先前的表的行组合，进行完整的表扫描。如果表是第一个没标记const的表，这通常不好，并且通常在它情况下很差。通常可以增加更多的索引而不要使用ALL，使得行能基于前面的表中的常数值或列值被检索出。

· possible_keys

possible_keys列指出**MySQL**能使用哪个索引在该表中找到行。注意，该列完全独立于EXPLAIN输出所示的表的次序。这意味着在possible_keys中的某些键实际上不能按生成的表次序使用。

如果该列是NULL，则没有相关的索引。在这种情况下，可以通过检查WHERE子句看是否它引用某些列或适合索引的列来提高你的查询性能。如果是这样，创造一个适当的索引并且再次用EXPLAIN检查查询。参见13.1.2节，“ALTER TABLE语法”。

为了看清一张表有什么索引，使用SHOW INDEX FROM *tbl_name*。

· key

key列显示**MySQL**实际决定使用的键（索引）。如果没有选择索引，键是NULL。要想强制MySQL使用或忽视possible_keys列中的索引，在查询中使用FORCE INDEX、USE INDEX或者IGNORE INDEX。参见13.2.7节，“SELECT语法”。

对于MyISAM和BDB表，运行ANALYZE TABLE可以帮助优化器选择更好的索引。对于MyISAM表，可以使用**myisamchk --analyze**。参见13.5.2.1节，“ANALYZE TABLE语法”和5.9.4节，“表维护和崩溃恢复”。

· key_len

key_len列显示**MySQL**决定使用的键长度。如果键是NULL，则长度为NULL。注意通过key_len值我们可以确定MySQL将实际使用一个多部关键字的几个部分。

· ref

ref列显示使用哪个列或常数与key一起从表中选择行。

· rows

rows列显示**MySQL**认为它执行查询时必须检查的行数。

· Extra

该列包含MySQL解决查询的详细信息。下面解释了该列可以显示的不同的文本字符串：

o Distinct

MySQL发现第1个匹配行后，停止为当前的行组合搜索更多的行。

o Not exists

MySQL能够对查询进行LEFT JOIN优化，发现1个匹配LEFT JOIN标准的行后，不再为前面的的行组合在该表内检查更多的行。

下面是一个可以这样优化的查询类型的例子：

```
SELECT * 从t1 LEFT JOIN t2 ON t1.id=t2.id
```

```
  WHERE t2.id IS NULL;
```

假定t2.id定义为NOT NULL。在这种情况下，MySQL使用t1.id的值扫描t1并查找t2中的行。如果MySQL在t2中发现一个匹配的行，它知道t2.id绝不会为NULL，并且不再扫描t2内有相同的id值的行。换句话说，对于t1的每个行，MySQL只需要在t2中查找一次，无论t2内实际有多少匹配的行。

- range checked for each record (index map: #)

MySQL没有发现好的可以使用的索引，但发现如果来自前面的表的列值已知，可能部分索引可以使用。对前面的表的每个行组合，MySQL检查是否可以使用range或index_merge访问方法来索取行。关于适用性标准的描述参见7.2.5节，“范围优化”和7.2.6节，“索引合并优化”，不同的是前面表的所有列值已知并且认为是常量。

这并不很快，但比执行没有索引的联接要快得多。

- Using filesort

MySQL需要额外的一次传递，以找出如何按排序顺序检索行。通过根据联接类型浏览所有行并为所有匹配WHERE子句的行保存排序关键字和行的指针来完成排序。然后关键字被排序，并按排序顺序检索行。参见7.2.12节，“MySQL如何优化ORDER BY”。

- Using index

从只使用索引树中的信息而不需要进一步搜索读取实际的行来检索表中的列信息。当查询只使用作为单一索引一部分的列时，可以使用该策略。

- Using temporary

为了解决查询，MySQL需要创建一个临时表来容纳结果。典型情况如查询包含可以按不同情况列出列的GROUP BY和ORDER BY子句时。

- Using where

WHERE子句用于限制哪一个行匹配下一个表或发送到客户。除非你专门从表中索取或检查所有行，如果Extra值不为Using where并且表联接类型为ALL或index，查询可能会有一些错误。

如果想要使查询尽可能快，应找出Using filesort 和Using temporary的Extra值。

- Using sort_union(...), Using union(...), Using intersect(...)

这些函数说明如何为index_merge联接类型合并索引扫描。详细信息参见7.2.6节，“索引合并优化”。

- Using index for group-by

类似于访问表的Using index方式，Using index for group-by表示MySQL发现了一个索引，可以用来查询GROUP BY或DISTINCT查询的所有列，而不要额外搜索硬盘访问实际的表。并且，按最有效的方式使用索引，以便对于每个组，只读取少量索引条目。详情参见7.2.13节，“MySQL如何优化GROUP BY”。

通过相乘EXPLAIN输出的rows列的所有值，你能得到一个关于一个联接如何的提示。这应该粗略地告诉你**MySQL**必须检查多少行以执行查询。当你使用max_join_size变量限制查询时，也用这个乘积来确定执行哪个多表SELECT语句。参见7.5.2节，“调节服务器参数”。

下列例子显示出一个多表JOIN如何能使用EXPLAIN提供的信息逐步被优化。

假定你有下面所示的SELECT语句，计划使用EXPLAIN来检查它：

```
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;
```

对于这个例子，假定:

- 被比较的列声明如下:

表	列	列类型
tt	ActualPC	CHAR(10)
tt	AssignedPC	CHAR(10)
tt	ClientID	CHAR(10)
et	EMPLOYID	CHAR(15)
do	CUSTNMBR	CHAR(15)

- 表有下面的索引:

表	索引
tt	ActualPC
tt	AssignedPC
tt	ClientID
et	EMPLOYID(主键)
do	CUSTNMBR(主键)

- tt.ActualPC值不是均匀分布的。

开始，在进行优化前，EXPLAIN语句产生下列信息:

```
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,   NULL NULL    NULL 3872
           ClientID,
           ActualPC
      range checked for each record (key map: 35)
```

因为type对每张表是ALL，这个输出显示**MySQL**正在对所有表产生一个笛卡尔乘积；即每一个行的组合！这将花相当长的时间，因为必须检查每张表的行数的乘积！对于一个实例，这是74 * 2135 * 74 * 3872 = 45,268,558,720行。如果表更大，你只能想象它将花多长时间……

这里的一个问题是**MySQL**能更高效地在声明具有相同类型和尺寸的列上使用索引。在本文中，VARCHAR和CHAR是相同的，除非它们声明为不同的长度。因为tt.ActualPC被声明为CHAR(10)并且et.EMPLOYID被声明为CHAR(15)，长度不匹配。

为了修正在列长度上的不同，使用ALTER TABLE将ActualPC的长度从10个字符变为15个字符：

```
mysql> ALTER TABLE tt MODIFY ActualPC VARCHAR(15);
```

现在tt.ActualPC和et.EMPLOYID都是VARCHAR(15)，再执行EXPLAIN语句产生这个结果：

```
table type   possible_keys key     key_len ref         rows    Extra
tt    ALL    AssignedPC,   NULL    NULL    NULL        3872    Using
             ClientID,                                         where
             ActualPC
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
```

这不是完美的，但是好一些了：rows值的乘积少了一个因子74。这个版本在几秒内执行完。

第2种方法能消除tt.AssignedPC = et_1.EMPLOYID和tt.ClientID = do.CUSTNMBR比较的列的长度失配问题：

```
mysql> ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),
    ->                MODIFY ClientID   VARCHAR(15);
```

EXPLAIN产生的输出显示在下面：

```
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
             ClientID,                                          where
             ActualPC
et_1  eq_ref PRIMARY       PRIMARY  15      tt.AssignedPC 1
do    eq_ref PRIMARY       PRIMARY  15      tt.ClientID   1
```

这几乎很好了。

剩下的问题是，默认情况，**MySQL**假设在tt.ActualPC列的值是均匀分布的，并且对tt表不是这样。幸好，很容

易告诉**MySQL**来分析关键字分布：

```
mysql> ANALYZE TABLE tt;
```

现在联接是"完美"的了，而且EXPLAIN产生这个结果：

```
table type   possible_keys key     key_len ref           rows Extra
tt    ALL    AssignedPC    NULL    NULL    NULL          3872 Using
             ClientID,                                        where
             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
```

注意在从EXPLAIN输出的rows列是一个来自**MySQL**联接优化器的"教育猜测"。你应该检查数字是否接近事实。如果不是，可以通过在SELECT语句里面使用STRAIGHT_JOIN并且试着在FROM子句以不同的次序列出表，可能得到更好的性能。

7.2.2. 估计查询性能

在大多数情况下，可以通过计算磁盘搜索来估计性能。对小的表，通常能在1次磁盘搜索中找到行(因为索引可能被缓存)。对更大的表，可以使用B－树索引进行估计，将需要log(*row_count*)/log(*index_block_length*/3 * 2/(*index_length* + *data_pointer_length*))+1次搜索才能找到行。

在**MySQL**中，索引块通常是1024个字节，数据指针通常是4个字节，这对于有一个长度为3(中等整数)的索引的500,000行的表，通过公式可以计算出log(500,000)/log(1024/3*2/(3+4))+1= 4次搜索。

上面的索引需要大约500,000 * 7 * 3/2 = 5.2MB，(假设典型情况下索引缓存区填充率为2/3)，可以将大部分索引保存在内存中，仅需要1-2调用从OS读数据来找出行。

然而对于写，将需要4次搜索请求(如上)来找到在哪儿存放新索引，并且通常需要2次搜索来更新这个索引并且写入行。

注意，上述讨论并不意味着应用程序的性能将缓慢地以log*N* 退化！当表格变得更大时，所有内容缓存到OS或SQL服务器后，将仅仅或多或少地更慢。在数据变得太大不能缓存后，将逐渐变得更慢，直到应用程序只能进行磁盘搜索(以log*N*增加)。为了避免这个问题，随数据增加而增加 键高速缓冲区大小。对于MyISAM表，由key_buffer_size系统变量控制 键高速缓冲区大小。参见7.5.2节，"调节服务器参数"。

7.2.3. SELECT查询的速度

总的来说，要想使一个较慢速SELECT ... WHERE更快，应首先检查是否能增加一个索引。不同表之间的引用通常通过索引来完成。你可以使用EXPLAIN语句来确定SELECT语句使用哪些索引。参见7.4.5节，"MySQL如何使用索引"和7.2.1节，"EXPLAIN语法（获取关于SELECT的信息）"。

下面是一些加速对MyISAM表的查询的一般建议：

- 为了帮助**MySQL**更好地优化查询，在一个装载数据后的表上运行ANALYZE TABLE或**myisamchk --analyze**。这样为每一个索引更新指出有相同值的行的平均行数的值（当然，如果只有一个索引，这总是1。）MySQL使用该方法来决定当你联接两个基于非常量表达式的表时选择哪个索引。你可以使用SHOW INDEX FROM *tbl_name*并检查Cardinality值来检查表分析结果。**myisamchk --description --verbose**可以

显示索引分布信息。

- 要想根据一个索引排序一个索引和数据，使用**myisamchk --sort-index --sort-records=1**（如果你想要在索引1上排序）。如果只有一个索引，想要根据该索引的次序读取所有的记录，这是使查询更快的一个好方法。但是请注意，第一次对一个大表按照这种方法排序时将花很长时间！

7.2.4. MySQL怎样优化WHERE子句

该节讨论为处理WHERE子句而进行的优化。例子中使用了SELECT语句，但相同的优化也适用DELETE和UPDATE语句中的WHERE子句。

请注意对MySQL优化器的工作在不断进行中，因此该节并不完善。MySQL执行了大量的优化，本文中所列的并不详尽。

下面列出了MySQL执行的部分优化：

- 去除不必要的括号：

```
((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)
```

- 常量重叠：

```
(a<b AND b=c) AND a=5
-> b>5 AND b=c AND a=5
```

- 去除常量条件(由于常量重叠需要)：

```
(B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)
-> B=5 OR B=6
```

- 索引使用的常数表达式仅计算一次。

- 对于MyISAM和HEAP表，在一个单个表上的没有一个WHERE的COUNT(*)直接从表中检索信息。当仅使用一个表时，对NOT NULL表达式也这样做。
- 无效常数表达式的早期检测。**MySQL**快速检测某些SELECT语句是不可能的并且不返回行。
- 如果不使用GROUP BY或分组函数(COUNT()、MIN()……)，HAVING与WHERE合并。
- 对于联接内的每个表，构造一个更简单的WHERE以便更快地对表进行WHERE计算并且也尽快跳过记录。
- 所有常数的表在查询中比其它表先读出。常数表为：
  - 空表或只有1行的表。
  - 与在一个PRIMARY KEY或UNIQUE索引的WHERE子句一起使用的表，这里所有的索引部分使用常数表达式并且索引部分被定义为NOT NULL。

下列的所有表用作常数表：

```
mysql> SELECT * FROM t WHERE primary_key=1;

mysql> SELECT * FROM t1,t2

           WHERE t1.primary_key=1 AND t2.primary_key=t1.id;
```

- 尝试所有可能性便可以找到表联接的最好联接组合。如果所有在ORDER BY和GROUP BY的列来自同一个表，那么当联接时，该表首先被选中。
- 如果有一个ORDER BY子句和不同的GROUP BY子句，或如果ORDER BY或GROUP BY包含联接队列中的第一个表之外的其它表的列，则创建一个临时表。

如果使用SQL_SMALL_RESULT，**MySQL**使用内存中的一个临时表。
- 每个表的索引被查询，并且使用最好的索引，除非优化器认为使用表扫描更有效。是否使用扫描取决于是否最好的索引跨越超过30%的表。优化器更加复杂，其估计基于其它因素，例如表大小、行数和I/O块大小，因此固定比例不再决定选择使用索引还是扫描。
- 在一些情况下，MySQL能从索引中读出行，甚至不查询数据文件。如果索引使用的所有列是数值类，那么只使用索引树来进行查询。
- 输出每个记录前，跳过不匹配HAVING子句的行。

下面是一些快速查询的例子:

```
SELECT COUNT(*) FROM tbl_name;


SELECT MIN(key_part1),MAX(key_part1) FROM tbl_name;


SELECT MAX(key_part2) FROM tbl_name
    WHERE key_part1=constant;


SELECT ... FROM tbl_name
    ORDER BY key_part1,key_part2,... LIMIT 10;


SELECT ... FROM tbl_name
    ORDER BY key_part1 DESC, key_part2 DESC, ... LIMIT 10;
```

下列查询仅使用索引树就可以解决(假设索引的列为数值型):

```
SELECT key_part1,key_part2 FROM tbl_name WHERE key_part1=val;


SELECT COUNT(*) FROM tbl_name
    WHERE key_part1=val1 AND key_part2=val2;


SELECT key_part2 FROM tbl_name GROUP BY key_part1;
```

下列查询使用索引按排序顺序检索行，不用另外的排序:

```
SELECT ... FROM tbl_name
    ORDER BY key_part1,key_part2,... ;


SELECT ... FROM tbl_name
    ORDER BY key_part1 DESC, key_part2 DESC, ... ;
```

7.2.5. 范围优化

7.2.5.1. 单元素索引的范围访问方法
7.2.5.2. 多元素索引的范围访问方法

range访问方法使用单一索引来搜索包含在一个或几个索引值距离内的表记录的子集。可以用于单部分或多元素索引。后面的章节将详细描述如何从WHERE子句提取区间。

7.2.5.1. 单元素索引的范围访问方法

对于单元素索引，可以用WHERE子句中的相应条件很方便地表示索引值区间，因此我们称为范围条件而不是"区间"。

单元素索引范围条件的定义如下：

- 对于BTREE和HASH索引，当使用=、<=>、IN、IS NULL或者IS NOT NULL操作符时，关键元素与常量值的比较关系对应一个范围条件。
- 对于BTREE索引，当使用>、<、>=、<=、BETWEEN、!=或者<>，或者LIKE '*pattern*'(其中'*pattern*'不以通配符开始)操作符时，关键元素与常量值的比较关系对应一个范围条件。
- 对于所有类型的索引，多个范围条件结合OR或AND则产生一个范围条件。

前面描述的"常量值"系指：

- 查询字符串中的常量
- 同一联接中的const或system表中的列
- 无关联子查询的结果
- 完全从前面类型的子表达式组成的表达式

下面是一些WHERE子句中有范围条件的查询的例子：

```
SELECT * FROM t1
    WHERE key_col > 1
    AND key_col < 10;

SELECT * FROM t1
    WHERE key_col = 1
    OR key_col IN (15,18,20);

SELECT * FROM t1
    WHERE key_col LIKE 'ab%'
    OR key_col BETWEEN 'bar' AND 'foo';
```

请注意在常量传播阶段部分非常量值可以转换为常数。

MySQL尝试为每个可能的索引从WHERE子句提取范围条件。在提取过程中，不能用于构成范围条件的条件被放弃，产生重叠范围的条件组合到一起，并且产生空范围的条件被删除。

例如，考虑下面的语句，其中key1是有索引的列，nonkey没有索引：

```
SELECT * FROM t1 WHERE
   (key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
   (key1 < 'bar' AND nonkey = 4) OR
```

```
(key1 < 'uux' AND key1 > 'z');
```

key1的提取过程如下:

1. 用原始WHERE子句开始:

2. (key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR

3. (key1 < 'bar' AND nonkey = 4) OR

4. (key1 < 'uux' AND key1 > 'z')

5. 删除nonkey = 4和key1 LIKE '%b', 因为它们不能用于范围扫描。删除它们的正确途径是用TRUE替换它们, 以便进行范围扫描时不会丢失匹配的记录。用TRUE替换它们后, 可以得到:

```
6.            (key1 < 'abc' AND (key1 LIKE 'abcde%' OR TRUE)) OR
7.            (key1 < 'bar' AND TRUE) OR
8.            (key1 < 'uux' AND key1 > 'z')
```

9. 取消总是为true或false的条件:

- (key1 LIKE 'abcde%' OR TRUE)总是true
- (key1 < 'uux' AND key1 > 'z')总是false

用常量替换这些条件, 我们得到:

```
(key1 < 'abc' AND TRUE) OR (key1 < 'bar' AND TRUE) OR (FALSE)
```

删除不必要的TRUE和FALSE常量, 我们得到

```
(key1 < 'abc') OR (key1 < 'bar')
```

10.将重叠区间组合成一个产生用于范围扫描的最终条件:

```
11.        (key1 < 'bar')
```

总的来说(如前面的例子所述), 用于范围扫描的条件比WHERE子句限制少。MySQL再执行检查以过滤掉满足范围条件但不完全满足WHERE子句的行。

范围条件提取算法可以处理嵌套的任意深度的AND/OR结构, 并且其输出不依赖条件在WHERE子句中出现的顺序。

7.2.5.2. 多元素索引的范围访问方法

多元素索引的范围条件是单元素索引的范围条件的扩展。多元素索引的范围条件将索引记录限制到一个或几个关键元组内。使用索引的顺序, 通过一系列关键元组来定义关键元组区间。

例如, 考虑定义为key1(*key_part1*, *key_part2*, *key_part3*)的多元素索引, 以及下面的按关键字顺序所列的关键元组:

```
key_part1  key_part2  key_part3
  NULL        1          'abc'
  NULL        1          'xyz'
  NULL        2          'foo'
```

```
1          1          'abc'
1          1          'xyz'
1          2          'abc'
2          1          'aaa'
```

条件*key_part1* = 1定义了下面的范围:

```
(1, -inf, -inf) <= (key_part1, key_part2, key_part3) < (1, +inf, +inf)
```

范围包括前面数据集中的第4、5和6个元组，可以用于范围访问方法。

通过对比，条件*key_part3* = 'abc'不定义单一的区间，不能用于范围访问方法。

下面更加详细地描述了范围条件如何用于多元素索引中。

- 对于HASH索引，可以使用包含相同值的每个区间。这说明区间只能由下面形式的条件产生:

```
key_part1 cmp const1
AND key_part2 cmp const2
AND ...
AND key_partN cmp constN;
```

这里，*const1*，*const2*，...为常量，*cmp*是=、<=>或者IS NULL比较操作符之一，条件包括所有索引部分。(也就是说，有*N* 个条件，每一个对应*N*-元素索引的每个部分）。

关于常量的定义，参见7.2.5.1节，“单元素索引的范围访问方法”。

例如，下面为三元素HASH索引的范围条件:

```
key_part1 = 1 AND key_part2 IS NULL AND key_part3 = 'foo'
```

- 对于BTREE索引，区间可以对结合AND的条件有用，其中每个条件用一个常量值通过=、<=>、IS NULL、>、<、>=、<=、!=、<>、BETWEEN或者LIKE '*pattern*' (其中'*pattern*'不以通配符开头)比较一个关键元素。区间可以足够长以确定一个包含所有匹配条件(或如果使用<>或!=，为两个区间)的记录的单一的关键元组。例如，对于条件:

```
key_part1 = 'foo' AND key_part2 >= 10 AND key_part3 > 10
```

单一区间为:

```
('foo', 10, 10)
  < (key_part1, key_part2, key_part3)
    < ('foo', +inf, +inf)
```

创建的区间可以比原条件包含更多的记录。例如，前面的区间包括值('foo'，11，0)，不满足原条件。

- 如果包含区间内的一系列记录的条件结合使用OR，则形成包括一系列包含在区间并集的记录的一个条件。如果条件结合使用了AND，则形成包括一系列包含在区间交集内的记录的一个条件。例如，对于两部分索引的条件:

```
(key_part1 = 1 AND key_part2 < 2)
OR (key_part1 > 5)
```

区间为:

(1, -inf) < (*key_part1*, *key_part2*) < (1, 2)

(5, -inf) < (*key_part1*, *key_part2*)

在该例子中，第1行的区间左侧的约束使用了一个关键元素，右侧约束使用了两个关键元素。第2行的区间只使用了一个关键元素。EXPLAIN输出的key_len列表示所使用关键字前缀的最大长度。

在某些情况中，key_len可以表示使用的关键元素，但可能不是你所期望的。假定*key_part1*和*key_part2*可以为NULL。则key_len列显示下面条件的两个关键元素的长度:

```
key_part1 >= 1 AND key_part2 < 2
```

但实际上，该条件可以变换为:

```
key_part1 >= 1 AND key_part2 IS NOT NULL
```

7.2.5.1节，“单元素索引的范围访问方法”描述了如何进行优化以结合或删除单元素索引范围条件的区间。多元素索引范围条件的区间的步骤类似。

7.2.6. 索引合并优化

7.2.6.1. 索引合并交集访问算法
7.2.6.2. 索引合并并集访问算法
7.2.6.3. 索引合并排序并集访问算法

索引合并方法用于通过range扫描搜索行并将结果合成一个。合并会产生并集、交集或者正在进行的扫描的交集的并集。

在EXPLAIN输出中，该方法表现为type列内的index_merge。在这种情况下，key列包含一列使用的索引，key_len包含这些索引的最长的关键元素。

例如:

SELECT * FROM *tbl_name* WHERE *key_part1* = 10 OR *key_part2* = 20;

SELECT * FROM *tbl_name*

WHERE (*key_part1* = 10 OR *key_part2* = 20) AND *non_key_part*=30;

SELECT * FROM t1, t2

WHERE (t1.*key1* IN (1,2) OR t1.*key2* LIKE '*value*%')

AND t2.*key1*=t1.*some_col*;

SELECT * FROM t1, t2

WHERE t1.*key1*=1

AND (t2.*key1*=t1.*some_col* OR t2.*key2*=t1.*some_col2*);

索引合并方法有几种访问算法 (参见EXPLAIN输出的Extra字段):

- 交集
- 联合
- 排序并集

后面几节更加详细地描述了这些方法。

注释: 索引合并优化算法具有以下几个已知缺陷:

- 如果可以对某些关键字进行范围扫描，则不考虑索引合并。例如，下面的查询:

```
SELECT * FROM t1 WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;
```

对于该查询，可以有两个方案:

1. 使用(goodkey1 < 10 OR goodkey2 < 20)条件进行索引合并扫描。
2. 使用badkey < 30条件进行范围扫描。

然而，优化器只考虑第2个方案。如果这不是你想要的，你可以通过使用IGNORE INDEX或FORCE INDEX让优化器考虑index_merge。下面的查询使用索引合并执行:

SELECT * FROM t1 FORCE INDEX(goodkey1,goodkey2)

WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;

SELECT * FROM t1 IGNORE INDEX(badkey)

WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;

- 如果查询有一个复杂的WHERE子句，有较深的AND/OR嵌套关系，MySQL不选择该优选方案，通过下面的识别法则尝试分布各条件:

```
(x AND y) OR z = (x OR z) AND (y OR z)
(x OR y) AND z = (x AND z) OR (y AND z)
```

index_merge访问方法的不同变量之间的选择和其它访问方法基于各适用选项的成本估计。

7.2.6.1. 索引合并交集访问算法

该访问算法可以用于当WHERE子句结合AND被转换为不同的关键字的几个范围条件，每个条件为下面之一:

- 以这种形式，即索引有确切的*N*部分(即包括了所有索引部分):

```
key_part1=const1 AND key_part2=const2 ... AND key_partN=constN
```

- 任何InnoDB或BDB表的主键的范围条件。

下面是一些例子:

SELECT * FROM *innodb_table* WHERE *primary_key* < 10 AND *key_col1*=20;

SELECT * FROM *tbl_name*

WHERE (*key1_part1*=1 AND *key1_part2*=2) AND *key2*=2;

索引合并交集算法同时对所有使用的索引进行扫描，并产生从合并的索引扫描接收的行序列的交集。

如果使用的索引包括查询中使用的所有列，所有表记录均不搜索，并且在这种情况下EXPLAIN的输出包含Extra字段中的Using index。下面是一个此类查询的例子:

```
SELECT COUNT(*) FROM t1 WHERE key1=1 AND key2=1;
```

如果使用的索引未包括查询中使用的所有列，只有满足所有使用的关键字的范围条件才搜索所有记录。

如果某个合并条件是InnoDB或BDB表的主键的一个条件，不用于记录查询，但用于过滤使用其它条件搜索的记录。

7.2.6.2. 索引合并并集访问算法

该算法的适用标准类似于索引合并方法交集算法的标准。算法可以用于当WHERE子句结合OR被转换为不同的关键字的几个范围条件的时候，每个条件为下面之一:

- 以这种形式，即索引有确切的*N*部分(即包括了所有索引部分):
- *key_part1=const1* AND *key_part2=const2* ... AND *key_partN=constN*
- 任何InnoDB或BDB表的主键的范围条件。
- 索引合并方法交集算法适用的一个条件。

下面是一些例子:

```
SELECT * FROM t1 WHERE key1=1 OR key2=2 OR key3=3;
```

```
SELECT * FROM innodb_table WHERE (key1=1 AND key2=2) OR
  (key3='foo' AND key4='bar') AND key5=5;
```

7.2.6.3. 索引合并排序并集访问算法

该访问算法可以用于当WHERE子句结合OR被转换为不同的关键字的几个范围条件，但索引合并方法联合算法并不适用的时候。

下面是一些例子:

```
SELECT * FROM tbl_name WHERE key_col1 < 10 OR key_col2 < 20;
```

```
SELECT * FROM tbl_name
     WHERE (key_col1 > 10 OR key_col2 = 20) AND nonkey_col=30;
```

排序联合算法和联合算法的区别是排序联合算法必须先索取所有记录的行ID，然后在返回记录前对它们进行排序。

7.2.7. MySQL如何优化IS NULL

MySQL可以对可以结合*col_name* = *constant_value*使用的*col_name* IS NULL进行相同的优化。例如，MySQL可以使用索引和范围用IS NULL搜索NULL。

```
SELECT * FROM tbl_name WHERE key_col IS NULL;


SELECT * FROM tbl_name WHERE key_col <=> NULL;


SELECT * FROM tbl_name
    WHERE key_col=const1 OR key_col=const2 OR key_col IS NULL;
```

如果WHERE子句包括声明为NOT NULL的列的*col_name* IS NULL条件，表达式则优化。当列会产生NULL时，不会进行优化；例如，如果来自LEFT JOIN右侧的表。

MySQL也可以优化组合*col_name* = *expr* AND *col_name* IS NULL，这是解决子查询的一种常用形式。当使用优化时EXPLAIN显示ref_or_null。

该优化可以为任何关键元素处理IS NULL。

下面是一些优化的查询例子，假定表t2的列a和b有一个索引：

```
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 ...);
```

ref_or_null首先读取参考关键字，然后单独搜索NULL关键字的行。

请注意该优化只可以处理一个IS NULL。在后面的查询中，MySQL只对表达式(t1.a=t2.a AND t2.a IS NULL)使用关键字查询，不能使用b的关键元素：

```
SELECT * FROM t1, t2
     WHERE (t1.a=t2.a AND t2.a IS NULL)
     OR (t1.b=t2.b AND t2.b IS NULL);
```

7.2.8. MySQL如何优化DISTINCT

在许多情况下结合ORDER BY的DISTINCT需要一个临时表。

请注意因为DISTINCT可能使用GROUP BY，必须清楚MySQL如何使用所选定列的一部分的ORDER BY或HAVING子句中的列。参见12.10.3节，“具有隐含字段的GROUP BY”。

在大多数情况下，DISTINCT子句可以视为GROUP BY的特殊情况。例如，下面的两个查询是等效的：

```
SELECT DISTINCT c1, c2, c3 FROM t1 WHERE c1 > const;


SELECT c1, c2, c3 FROM t1 WHERE c1 > const GROUP BY c1, c2, c3;
```

由于这个等效性，适用于GROUP BY查询的优化也适用于有DISTINCT子句的查询。这样，关于DISTINCT查询的优化的更详细的情况，参见7.2.13节，“MySQL如何优化GROUP BY”。

结合LIMIT *row_count*和DISTINCT后，MySQL发现唯一的*row_count*行后立即停止。

如果不使用查询中命名的所有表的列，MySQL发现第1个匹配后立即停止扫描未使用的表。在下面的情况中，假定t1在t2之前使用(可以用EXPLAIN检查)，发现t2中的第1行后，MySQL不再(为t1中的任何行)读t2：

```
SELECT DISTINCT t1.a FROM t1, t2 where t1.a=t2.a;
```

7.2.9. MySQL如何优化LEFT JOIN和RIGHT JOIN

在MySQL中，*A* LEFT JOIN *B* join_condition执行过程如下：

- 根据表*A*和*A*依赖的所有表设置表B。
- 根据LEFT JOIN条件中使用的所有表(除了*B*)设置表*A*。
- LEFT JOIN条件用于确定如何从表*B*搜索行。(换句话说，不使用WHERE子句中的任何条件）。
- 可以对所有标准联接进行优化，只是只有从它所依赖的所有表读取的表例外。如果出现循环依赖关系，MySQL提示出现一个错误。
- 进行所有标准WHERE优化。
- 如果*A*中有一行匹配WHERE子句，但*B*中没有一行匹配ON条件，则生成另一个*B*行，其中所有列设置为NULL。
- 如果使用LEFT JOIN找出在某些表中不存在的行，并且进行了下面的测试：WHERE部分的*col_name* IS NULL，其中*col_name*是一个声明为 NOT NULL的列，MySQL找到匹配LEFT JOIN条件的一个行后停止(为具体的关键字组合)搜索其它行。

RIGHT JOIN的执行类似LEFT JOIN，只是表的角色反过来。

联接优化器计算表应联接的顺序。LEFT JOIN和STRAIGHT_JOIN强制的表读顺序可以帮助联接优化器更快地工作，因为检查的表交换更少。请注意这说明如果执行下面类型的查询，MySQL进行全扫描b，因为LEFT JOIN强制它在d之前读取：

```
SELECT *
```

```
    FROM a,b LEFT JOIN c ON (c.key=a.key) LEFT JOIN d ON (d.key=a.key)
    WHERE b.key=d.key;
```

在这种情况下修复时用a的相反顺序，b列于FROM子句中：

```
SELECT *
    FROM b,a LEFT JOIN c ON (c.key=a.key) LEFT JOIN d ON (d.key=a.key)
    WHERE b.key=d.key;
```

MySQL可以进行下面的LEFT JOIN优化：如果对于产生的NULL行，WHERE条件总为假，LEFT JOIN变为普通联接。

例如，在下面的查询中如果t2.column1为NULL，WHERE 子句将为false：

```
SELECT * FROM t1 LEFT JOIN t2 ON (column1) WHERE t2.column2=5;
```

因此，可以安全地将查询转换为普通联接：

```
SELECT * FROM t1, t2 WHERE t2.column2=5 AND t1.column1=t2.column1;
```

这样可以更快，因为如果可以使查询更佳，MySQL可以在表t1之前使用表t2。为了强制使用表顺序，使用STRAIGHT_JOIN。

7.2.10. MySQL如何优化嵌套Join

表示联接的语法允许嵌套联接。下面的讨论引用了13.2.7.1节，“JOIN语法”中描述的联接语法。

同SQL标准比较，*table_factor*语法已经扩展了。后者只接受*table_reference*，而不是括号内所列的。

*table_reference*项列表内的每个逗号等价于内部联接，这是一个保留扩展名。例如：

```
SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

等价于：

```
SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

在MySQL中，CROSS JOIN语法上等价于INNER JOIN (它们可以彼此代替。在标准SQL中，它们不等价。INNER JOIN结合ON子句使用；CROSS JOIN 用于其它地方。

总的来说，在只包含内部联接操作的联接表达式中可以忽略括号。删除括号并将操作组合到左侧后，联接表达式：

```
t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
   ON t1.a=t2.a
```

转换为表达式：

```
(t1 LEFT JOIN t2 ON t1.a=t2.a) LEFT JOIN t3
    ON t2.b=t3.b OR t2.b IS NULL
```

但是这两个表达式不等效。要说明这点，假定表t1、t2和t3有下面的状态：

- 表t1包含行{1}、{2}
- 表t2包含行{1,101}
- 表t3包含行{101}

在这种情况下，第1个表达式返回包括行{1,1,101,101}、{2,NULL,NULL,NULL}的结果，第2个表达式返回行{1,1,101,101}、{2,NULL,NULL,101}：

```
mysql> SELECT *
    -> FROM t1
    ->      LEFT JOIN
    ->      (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
    ->      ON t1.a=t2.a;
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL | NULL |
+------+------+------+------+


mysql> SELECT *
    -> FROM (t1 LEFT JOIN t2 ON t1.a=t2.a)
    ->      LEFT JOIN t3
    ->      ON t2.b=t3.b OR t2.b IS NULL;
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL |  101 |
+------+------+------+------+
```

在下面的例子中，外面的联接操作结合内部联接操作使用：

```
t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
```

该表达式不能转换为下面的表达式：

```
t1 LEFT JOIN t2 ON t1.a=t2.a, t3.
```

对于给定的表状态，第1个表达式返回行{1,1,101,101}、{2,NULL,NULL,NULL}，第2个表达式返回行{1,1,101,101}、{2,NULL,NULL,101}：

```
mysql> SELECT *
    -> FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a;
+------+------+------+------+
```

```
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL | NULL |
+------+------+------+------+


mysql> SELECT *
    -> FROM t1 LEFT JOIN t2 ON t1.a=t2.a, t3;
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL |  101 |
+------+------+------+------+
```

因此，如果我们忽略联接表达式中的括号连同外面的联接操作符，我们会改变原表达式的结果。

更确切地说，我们不能忽视左外联接操作的右操作数和右联接操作的左操作数中的括号。换句话说，我们不能忽视外联接操作中的内表达式中的括号。可以忽视其它操作数中的括号(外部表的操作数)。

对于任何表t1、t2、t3和属性t2.b和t3.b的任何条件P，下面的表达式：

```
(t1,t2) LEFT JOIN t3 ON P(t2.b,t3.b)
```

等价于表达式

```
t1, t2 LEFT JOIN t3 ON P(t2.b,t3.b)
```

如果联接表达式(*join_table*)中的联接操作的执行顺序不是从左到右，我们则应讨论嵌套的联接。这样，下面的查询：

```
SELECT * FROM t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b) ON t1.a=t2.a
  WHERE t1.a > 1


SELECT * FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
  WHERE (t2.b=t3.b OR t2.b IS NULL) AND t1.a > 1
```

联接表：

```
t2 LEFT JOIN t3 ON t2.b=t3.b
t2, t3
```

认为是嵌套的。第1个查询结合左联接操作则形成嵌套的联接，而在第二个查询中结合内联接操作形成嵌套联接。

在第1个查询中，括号可以忽略：联接表达式的语法结构与联接操作的执行顺序相同。但对于第2个查询，括号不能省略，尽管如果没有括号，这里的联接表达式解释不清楚。(在外部扩展语法中，需要第2个查询的(t2，t3)的括号，尽管从理论上对查询分析时不需要括号：这些查询的语法结构将仍然不清楚，因为LEFT JOIN和ON将充当表达式(t2,t3)的左、右界定符的角色)。

前面的例子说明了这些点:

- 对于只包含内联接(而非外联接)的联接表达式，可以删除括号。你可以移除括号并从左到右评估(或实际上，你可以按任何顺序评估表)。
- 总的来说，对外联接却不是这样。去除括号可能会更改结果。
- 总的来说，对外联接和内联接的结合，也不是这样。去除括号可能会更改结果。

含嵌套外联接的查询按含内联接的查询的相同的管道方式执行。更确切地说，利用了嵌套环联接算法。让我们回忆嵌套环联接执行查询时采用什么算法。

假定我们有一个如下形式的表T1、T2、T3的联接查询:

```
SELECT * FROM T1 INNER JOIN T2 ON P1(T1,T2)
                 INNER JOIN T3 ON P2(T2,T3)
  WHERE P(T1,T2,T3).
```

这里，P1(T1,T2)和P2(T3,T3)是一些联接条件(表达式)，其中P(t1,t2,t3)是表T1、T2、T3的列的一个条件。

嵌套环联接算法将按下面的方式执行该查询:

```
FOR each row t1 in T1 {
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1,t2,t3) {
         t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

符号t1||t2||t3表示"连接行t1、t2和t3的列组成的行"。在下面的一些例子中，出现行名的NULL表示NULL用于行的每个列。例如，t1||t2||NULL表示"连接行t1和t2的列以及t3的每个列的NULL组成的行"。

现在让我们考虑带嵌套的外联接的查询:

```
SELECT * FROM T1 LEFT JOIN
              (T2 LEFT JOIN T3 ON P2(T2,T3))
              ON P1(T1,T2)
  WHERE P(T1,T2,T3)。
```

对于该查询我们修改嵌套环模式可以得到:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    BOOL f2:=FALSE;
```

```
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f2=TRUE;
      f1=TRUE;
    }
    IF (!f2) {
      IF P(t1,t2,NULL) {
        t:=t1||t2||NULL; OUTPUT t;
      }
      f1=TRUE;
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

总的来说，对于外联接操作中的第一个内表的嵌套环，引入了一个标志，在环之前关闭并且在环之后打开。如果对于外部表的当前行，如果匹配表示内操作数的表，则标志打开。如果在循环结尾处标志仍然关闭，则对于外部表的当前行，没有发现匹配。在这种情况下，对于内表的列，应使用NULL值补充行。结果行被传递到输出进行最终检查或传递到下一个嵌套环，但只能在行满足所有嵌入式外联接的联接条件时。

在我们的例子中，嵌入了下面表达式表示的外联接表：

```
(T2 LEFT JOIN T3 ON P2(T2,T3))
```

请注意对于有内联接的查询，优化器可以选择不同的嵌套环顺序，例如：

```
FOR each row t3 in T3 {
  FOR each row t2 in T2 such that P2(t2,t3) {
    FOR each row t1 in T1 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
         t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

对于有外联接的查询，优化器可以只选择这样的顺序：外表的环优先于内表的环。这样，对于有外联接的查

询，只可能有一种嵌套顺序。在下面的查询中，优化器将评估两个不同的嵌套：

```
SELECT * T1 LEFT JOIN (T2,T3) ON P1(T1,T2) AND P2(T1,T3)
  WHERE P(T1,T2,T3)
```

嵌套为：

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t1,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f1:=TRUE
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

和

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t3 in T3 such that P2(t1,t3) {
    FOR each row t2 in T2 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f1:=TRUE
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

在两个嵌套中，必须在外环中处理T1，因为它用于外联接中。T2和T3用于内联接中，因此联接必须在内环中处理。但是，因为该联接是一个内联接，T2和T3可以以任何顺序处理。

当讨论内联接嵌套环的算法时，我们忽略了部分详情，可能对查询执行的性能的影响会很大。我们没有提及所谓的"下推"条件。假定可以用连接公式表示我们的WHERE条件P(T1,T2,T3)：

```
P(T1,T2,T2) = C1(T1) AND C2(T2) AND C3(T3)。
```

在这种情况下，MySQL实际使用了下面的嵌套环方案来执行带内联接得到查询：

```
FOR each row t1 in T1 such that C1(t1) {
  FOR each row t2 in T2 such that P1(t1,t2) AND C2(t2)  {
    FOR each row t3 in T3 such that P2(t2,t3) AND C3(t3) {
      IF P(t1,t2,t3) {
         t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

你会看见每个连接 C1(T1)，C2(T2)，C3(T3)被从最内部的环内推出到可以对它进行评估的最外的环中。如果C1(T1)是一个限制性很强的条件，下推条件可以大大降低从表T1传递到内环的行数。结果是查询大大加速。

对于有外联接的查询，只有查出外表的当前的行可以匹配内表后，才可以检查WHERE条件。这样，对内嵌套环下推的条件不能直接用于带外联接的查询。这里我们必须引入有条件下推前提，由遇到匹配后打开的标志保护。

对于带下面的外联接的例子

```
P(T1,T2,T3)=C1(T1) AND C(T2) AND C3(T3)
```

使用受保护的下推条件的嵌套环方案看起来应为：

```
FOR each row t1 in T1 such that C1(t1) {
  BOOL f1:=FALSE;
  FOR each row t2 in T2
      such that P1(t1,t2) AND (f1?C2(t2):TRUE) {
    BOOL f2:=FALSE;
    FOR each row t3 in T3
        such that P2(t2,t3) AND (f1&&f2?C3(t3):TRUE) {
      IF (f1&&f2?TRUE:(C2(t2) AND C3(t3))) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f2=TRUE;
      f1=TRUE;
    }
    IF (!f2) {
```

```
      IF (f1?TRUE:C2(t2) && P(t1,t2,NULL)) {
        t:=t1||t2||NULL; OUTPUT t;
      }
      f1=TRUE;
    }
  }
  IF (!f1 && P(t1,NULL,NULL)) {
      t:=t1||NULL||NULL; OUTPUT t;
  }
}
```

总的来说，可以从联接条件（例如P1(T1,T2)和P(T2,T3)）提取下推前提。在这种情况下，下推前提也受一个标志保护，防止检查由相应外联接操作所产生的NULL-补充的行的断言。

请注意如果从判断式的WHERE条件推导出，根据从一个内表到相同嵌套联接的另一个表的关键字进行的访问被禁止。(在这种情况下，我们可以使用有条件关键字访问，但是该技术还未用于MySQL 5.1中）。

7.2.11. MySQL如何简化外部联合

在许多情况下，一个查询的FROM子句的表的表达式可以简化。

在分析阶段，带右外联接操作的查询被转换为只包含左联接操作的等效查询。总的来说，根据以下原则进行转换：

```
(T1, ...) RIGHT JOIN (T2,...) ON P(T1,...,T2,...) =
(T2, ...) LEFT JOIN (T1,...) ON P(T1,...,T2,...)
```

所有T1 INNER JOIN T2 ON P(T1,T2)形式的内联接表达式被替换为T1,T2、P(T1,T2)，并根据WHERE条件(或嵌入连接的联接条件，如果有)联接为一个连接。

当优化器为用外联接操作的联接查询评估方案时，它只考虑在访问内表之前访问外表的操作的方案。优化器选项受到限制，因为只有这样的方案允许我们用嵌套环机制执行带外联接操作的查询。

假定我们有一个下列形式的查询：

SELECT * T1 LEFT JOIN T2 ON P1(T1,T2)

  WHERE P(T1,T2) AND R(T2)

R(T2)大大减少了表T2中匹配的行数。如果我们这样执行查询，优化器将不会有其它选择，只能在访问表T2之前访问表T1，从而导致执行方案非常低。

幸运的是，如果WHERE条件拒绝null，MySQL可以将此类查询转换为没有外联接操作的查询。如果为该操作构建的NULL补充的行评估为FALSE或UNKNOWN，则该条件称为对于某个外联接操作拒绝null。

因此，对于该外联接：

```
T1 LEFT JOIN T2 ON T1.A=T2.A
```

类似下面的条件为拒绝null：

```
T2.B IS NOT NULL,
```

```
T2.B > 3,
T2.C <= T1.C,
T2.B < 2 OR T2.C > 1
```

类似下面的条件不为拒绝null:

```
T2.B IS NULL,
T1.B < 3 OR T2.B IS NOT NULL,
T1.B < 3 OR T2.B > 3
```

检查一个外联接操作的条件是否拒绝null的总原则很简单。以下情况下为拒绝null的条件:

- 形式为A IS NOT NULL，其中A是任何内表的一个属性
- 包含内表引用的判断式，当某个参量为NULL时评估为UNKNOWN
- 包含用于连接的拒绝null的条件的联合
- 拒绝null的条件的逻辑和

一个条件可以对于一个查询中的一个外联接操作为拒绝null的而对于另一个不为拒绝null的。在下面的查询中:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
            LEFT JOIN T3 ON T3.B=T1.B
  WHERE T3.C > 0
```

WHERE条件对于第2个外联接操作为拒绝null的但对于第1个不为拒绝null的。

如果WHERE条件对于一个查询中的一个外联接操作为拒绝null的，外联接操作被一个内联接操作代替。

例如，前面的查询被下面的查询代替:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
            INNER JOIN T3 ON T3.B=T1.B
  WHERE T3.C > 0
```

对于原来的查询，优化器将评估只与一个访问顺序T1、T2、T3兼容的方案。在替换的查询中，还考虑了访问顺序T3、T1、T2。

一个外联接操作的转化可以触发另一个的转化。这样，查询:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
            LEFT JOIN T3 ON T3.B=T2.B
  WHERE T3.C > 0
```

将首先转换为查询:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
            INNER JOIN T3 ON T3.B=T2.B
```

WHERE T3.C > 0

该查询等效于查询:

SELECT * FROM (T1 LEFT JOIN T2 ON T2.A=T1.A), T3

WHERE T3.C > 0 AND T3.B=T2.B

现在剩余的外联接操作也可以被一个内联接替换，因为条件T3.B=T2.B为拒绝null的，我们可以得到一个根本没有外联接的查询:

SELECT * FROM (T1 INNER JOIN T2 ON T2.A=T1.A), T3

WHERE T3.C > 0 AND T3.B=T2.B

有时我们可以成功替换嵌入的外联接操作，但不能转换嵌入的外联接。下面的查询:

SELECT * FROM T1 LEFT JOIN

(T2 LEFT JOIN T3 ON T3.B=T2.B)

ON T2.A=T1.A

WHERE T3.C > 0

被转换为:

SELECT * FROM T1 LEFT JOIN

(T2 INNER JOIN T3 ON T3.B=T2.B)

ON T2.A=T1.A

WHERE T3.C > 0,

只能重新写为仍然包含嵌入式外联接操作的形式:

SELECT * FROM T1 LEFT JOIN

(T2,T3)

ON (T2.A=T1.A AND T3.B=T2.B)

WHERE T3.C > 0。

如果试图转换一个查询中的嵌入式外联接操作，我们必须考虑嵌入式外联接的联接条件和WHERE条件。在下面的查询中:

```
SELECT * FROM T1 LEFT JOIN

              (T2 LEFT JOIN T3 ON T3.B=T2.B)

              ON T2.A=T1.A AND T3.C=T1.C

  WHERE T3.D > 0 OR T1.D > 0
```

WHERE条件对于嵌入式外联接不为拒绝null的，但嵌入式外联接T2.A=T1.A AND T3.C=T1.C的联接条件为拒绝null的。因此该查询可以转换为:

SELECT * FROM T1 LEFT JOIN

(T2, T3)

ON T2.A=T1.A AND T3.C=T1.C AND T3.B=T2.B

WHERE T3.D > 0 OR T1.D > 0

7.2.12. MySQL如何优化ORDER BY

在某些情况中，MySQL可以使用一个索引来满足ORDER BY子句，而不需要额外的排序。

即使ORDER BY不确切匹配索引，只要WHERE子句中的所有未使用的索引部分和所有额外的ORDER BY 列为常数，就可以使用索引。下面的查询使用索引来解决ORDER BY部分：

SELECT * FROM t1

ORDER BY *key_part1*,*key_part2*,... ;

SELECT * FROM t1

WHERE *key_part1=constant*

ORDER 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;

在某些情况下，MySQL不能使用索引来解决ORDER BY，尽管它仍然使用索引来找到匹配WHERE子句的行。这些情况包括：

- 对不同的关键字使用ORDER BY：
- `SELECT * FROM t1 ORDER BY key1, key2;`
- 对关键字的非连续元素使用ORDER BY：
- `SELECT * FROM t1 WHERE key2=constant ORDER BY key_part2;`
- 混合ASC和DESC：
- `SELECT * FROM t1 ORDER BY key_part1 DESC, key_part2 ASC;`
- 用于查询行的关键字与ORDER BY中所使用的不相同：
- `SELECT * FROM t1 WHERE key2=constant ORDER BY key1;`

- 你正联接许多表，并且ORDER BY中的列并不是全部来自第1个用于搜索行的非常量表。(这是EXPLAIN输出中的没有const联接类型的第1个表）。
- 有不同的ORDER BY和GROUP BY表达式。
- 使用的表索引的类型不能按顺序保存行。例如，对于HEAP表的HASH索引情况即如此。

通过EXPLAIN SELECT ...ORDER BY，可以检查MySQL是否可以使用索引来解决查询。如果Extra列内有Using filesort，则不能解决查询。参见7.2.1节，“EXPLAIN语法（获取关于SELECT的信息）”。

文件排序优化不仅用于记录排序关键字和行的位置，并且还记录查询需要的列。这样可以避免两次读取行。文件排序算法的工作象这样：

1. 读行匹配WHERE子句的行，如前面所示。
2. 对于每个行，记录构成排序关键字和行位置的一系列值，并且记录查询需要的列。
3. 根据排序关键字排序元组
4. 按排序的顺序检索行，但直接从排序的元组读取需要的列，而不是再一次访问表。

该算法比以前版本的Mysql有很大的改进。

为了避免速度变慢，该优化只用于排序元组中的extra列的总大小不超过max_length_for_sort_data系统变量值的时候。(将该变量设置得太高的的迹象是将看到硬盘活动太频繁而CPU活动较低）。

如果想要增加ORDER BY的速度，首先看是否可以让MySQL使用索引而不是额外的排序阶段。如果不能，可以尝试下面的策略：

- 增加sort_buffer_size变量的大小。
- 增加read_rnd_buffer_size变量的大小。
- 更改tmpdir指向具有大量空闲空间的专用文件系统。该选项接受几个使用round-robin(循环)模式的路径。在Unix中路径应用冒号(':')区间开，在Windows、NetWare和OS/2中用分号(';')。可以使用该特性将负载均分到几个目录中。注释：路径应为位于不同物理硬盘上的文件系统的目录，而不是同一硬盘的不同的分区。

默认情况下，MySQL排序所有GROUP BY *col1*，*col2*，...查询的方法如同在查询中指定ORDER BY *col1*，*col2*，...。如果显式包括一个包含相同的列的ORDER BY子句，MySQL可以毫不减速地对它进行优化，尽管仍然进行排序。如果查询包括GROUP BY但你想要避免排序结果的消耗，你可以指定ORDER BY NULL禁止排序。例如：

```
INSERT INTO foo

SELECT a, COUNT(*) FROM bar GROUP BY a ORDER BY NULL;
```

7.2.13. MySQL如何优化GROUP BY

7.2.13.1. 松散索引扫描
7.2.13.2. 紧凑索引扫描

满足GROUP BY子句的最一般的方法是扫描整个表并创建一个新的临时表，表中每个组的所有行应为连续的，然后使用该临时表来找到组并应用累积函数(如果有)。在某些情况中，MySQL能够做得更好，通过索引访问而不用创建临时表。

为GROUP BY使用索引的最重要的前提条件是 所有GROUP BY列引用同一索引的属性，并且索引按顺序保存其关键字(例如，这是B-树索引，而不是HASH索引)。是否用索引访问来代替临时表的使用还取决于在查询中使用了哪部分索引、为该部分指定的条件，以及选择的累积函数。

有两种方法通过索引访问执行GROUP BY查询，如下面的章节所描述。在第1个方法中，组合操作结合所有范围判断式使用(如果有)。第2个方法首先执行范围扫描，然后组合结果元组。

7.2.13.1. 松散索引扫描

使用索引时最有效的途径是直接搜索组域。通过该访问方法，MySQL使用某些关键字排序的索引类型(例如，B-树)的属性。该属性允许使用 索引中的查找组而不需要考虑满足所有WHERE条件的索引中的所有关键字。既然该访问方法只考虑索引中的关键字的一小部分，它被称为松散索引扫描。如果没有WHERE子句， 松散索引扫描读取的关键字数量与组数量一样多，可以比所有关键字数小得多。如果WHERE子句包含范围判断式(关于range联接类型的讨论参见7.2.1节，“EXPLAIN语法（获取关于SELECT的信息）”)， 松散索引扫描查找满足范围条件的每个组的第1个关键字，并且再次读取尽可能最少数量的关键字。在下面的条件下是可以的：

- 查询针对一个单表。
- GROUP BY包括索引的第1个连续部分(如果对于GROUP BY，查询有一个DISTINCT子句，则所有显式属性指向索引开头)。
- 只使用累积函数(如果有)MIN()和MAX()，并且它们均指向相同的列。
- 索引的任何其它部分（除了那些来自查询中引用的GROUP BY）必须为常数(也就是说，必须按常量数量来引用它们)，但MIN()或MAX() 函数的参数例外。

此类查询的EXPLAIN输出显示Extra列的Using indexforgroup-by。

下面的查询提供该类的几个例子，假定表t1(c1,c2,c3,c4)有一个索引idx(c1，c2，c3)：

SELECT c1, c2 FROM t1 GROUP BY c1, c2;

SELECT DISTINCT c1, c2 FROM t1;

SELECT c1, MIN(c2) FROM t1 GROUP BY c1;

SELECT c1, c2 FROM t1 WHERE c1 < *const* GROUP BY c1, c2;

SELECT MAX(c3), MIN(c3), c1, c2 FROM t1 WHERE c2 > *const* GROUP BY c1, c2;

SELECT c2 FROM t1 WHERE c1 < *const* GROUP BY c1, c2;

SELECT c1, c2 FROM t1 WHERE c3 = *const* GROUP BY c1, c2；

由于上述原因，不能用该快速选择方法执行下面的查询：

1. 除了MIN()或MAX()，还有其它累积函数，例如：

```
SELECT c1, SUM(c2) FROM t1 GROUP BY c1;
```

2. GROUP BY子句中的域不引用索引开头，如下所示：

```
SELECT c1,c2 FROM t1 GROUP BY c2, c3;
```

3. 查询引用了GROUP BY部分后面的关键字的一部分，并且没有等于常量的等式，例如：

```
SELECT c1,c3 FROM t1 GROUP BY c1, c2;
```

7.2.13.2. 紧凑索引扫描

紧凑式索引扫描可以为索引扫描或一个范围索引扫描，取决于查询条件。

如果不满足松散索引扫描条件，GROUP BY查询仍然可以不用创建临时表。如果WHERE子句中有范围条件，该方法只读取满足这些条件的关键字。否则，进行索引扫描。该方法读取由WHERE子句定义的每个范围的所有关键字，或没有范围条件式扫描整个索引，我们将它定义为紧凑式索引扫描。请注意对于紧凑式索引扫描，只有找到了满足范围条件的所有关键字后才进行组合操作。

要想让该方法工作，对于引用GROUP BY关键字元素的前面、中间关键字元素的查询中的所有列，有一个常量等式条件即足够了。等式条件中的常量填充了搜索关键字中的"差距"，可以形成完整的索引前缀。这些索引前缀可以用于索引查找。如果需要排序GROUP BY结果，并且能够形成索引前缀的搜索关键字，MySQL还可以避免额外的排序操作，因为使用有顺序的索引的前缀进行搜索已经按顺序检索到了所有关键字。

上述的第一种方法不适合下面的查询，但第2种索引访问方法可以工作(假定我们已经提及了表t1的索引idx)：

- GROUP BY中有一个差距，但已经由条件c2 = 'a'覆盖。

```
SELECT c1, c2, c3 FROM t1 WHERE c2 = 'a' GROUP BY c1, c3;
```

- GROUP BY不以关键字的第1个元素开始，但是有一个条件提供该元素的常量：

```
SELECT c1, c2, c3 FROM t1 WHERE c1 = 'a' GROUP BY c2, c3;
```

7.2.14. MySQL如何优化LIMIT

在一些情况中，当你使用LIMIT *row_count*而不使用HAVING时，**MySQL**将以不同方式处理查询。

- 如果你用LIMIT只选择一些行，当**MySQL**选择做完整的表扫描时，它将在一些情况下使用索引。

- 如果你使用LIMIT *row_count*与ORDER BY，**MySQL**一旦找到了排序结果的第一个*row_count*行，将结束排序而不是排序整个表。如果使用索引，将很快。如果必须进行文件排序（filesort），必须选择所有匹配查询没有LIMIT子句的行，并且在确定已经找到第1个*row_count*行前，必须对它们的大部分进行排序。在任何一种情况下，一旦找到了行，则不需要再排序结果的其它部分，并且MySQL不再进行排序。

- 当结合LIMIT *row_count*和DISTINCT时，**MySQL**一旦找到*row_count*个唯一的行，它将停止。

- 在一些情况下，GROUP BY能通过顺序读取键(或在键上做排序)来解决，然后计算摘要直到关键字的值改变。在这种情况下，LIMIT *row_count*将不计算任何不必要的GROUP BY值。

- 只要MySQL已经发送了需要的行数到客户，它将放弃查询，除非你正使用SQL_CALC_FOUND_ROWS。

- LIMIT 0将总是快速返回一个空集合。这对检查查询的有效性是有用的。当使用MySQL API时，它也可以用来得到结果列的列类型。(该技巧在MySQL Monitor中不工作，只显示Empty set；应使用SHOW COLUMNS或DESCRIBE）。

- 当服务器使用临时表来进行查询时，使用LIMIT *row_count*子句来计算需要多少空间。

7.2.15. 如何避免表扫描

EXPLAIN的输出显示了当MySQL使用表扫描来解决查询时使用的所有类型列。这通常在如下条件下发生：

- 表很小，扫描表比查找关键字速度快。这对于少于10行并且行较短的表比较普遍。

- 在ON或WHERE子句中没有适用的索引列的约束。

- 正用常量值比较索引列，并且MySQL已经计算到(基于索引树)常数覆盖了表的很大部分并且表扫描将会比较快。参见7.2.4节，“MySQL怎样优化WHERE子句”。

- 你正通过另一个列使用一个低的集的势的关键字(许多行匹配关键字)。在这种情况下，MySQL假设通过使用关键字它可能会进行许多关键字查找，表扫描将会更快。

对于小表，表扫描通常合适。对于大表，尝试下面的技巧以避免优化器错选了表扫描：

- 使用ANALYZE TABLE *tbl_name*为扫描的表更新关键字分布。参见13.5.2.1节，“ANALYZE TABLE语法”。

- 对扫描的表使用FORCE INDEX告知MySQL，相对于使用给定的索引表扫描将非常耗时。参见13.2.7节，“SELECT语法”。

```
SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
    WHERE t1.col_name=t2.col_name;
```

- 用--max-seeks-for-key=1000选项启动**mysqld**或使用SET max_seeks_for_key=1000告知优化器假设关键字扫描不会超过1,000次关键字搜索。参见5.3.3节，“服务器系统变量”。

7.2.16. INSERT语句的速度

插入一个记录需要的时间由下列因素组成，其中的数字表示大约比例：

- 连接：(3)
- 发送查询给服务器：(2)
- 分析查询：(2)
- 插入记录：（1x记录大小）
- 插入索引：（1x索引）
- 关闭：(1)

这不考虑打开表的初始开销，每个并发运行的查询打开。

表的大小以log*N* (B树)的速度减慢索引的插入。

加快插入的一些方法：

- 如果同时从同一个客户端插入很多行，使用含多个VALUE的INSERT语句同时插入几行。这比使用单行INSERT语句快(在某些情况下快几倍)。如果你正向一个非空表添加数据，可以调节bulk_insert_buffer_size变量，使数据插入更快。参见5.3.3节，“服务器系统变量”。

- 如果你从不同的客户端插入很多行，能通过INSERT DELAYED语句加快速度。参见13.2.4节，“INSERT语法”。

- 用MyISAM，如果在表中没有删除的行，能在SELECT语句正在运行的同时插入行。

- 当从一个文本文件装载一个表时，使用LOAD DATA INFILE。这通常比使用很多INSERT语句快20倍。参见13.2.5节，“LOAD DATA INFILE语法”。

- 当表有很多索引时，有可能要多做些工作使得LOAD DATA INFILE更快些。使用下列过程：

1. 有选择地用CREATE TABLE创建表。
2. 执行FLUSH TABLES语句或命令**mysqladmin flush-tables**。
3. 使用**myisamchk --keys-used=0 -rq** */path/to/db/tbl_name*。这将从表中取消所有索引的使用。
4. 用LOAD DATA INFILE把数据插入到表中，因为不更新任何索引，因此很快。
5. 如果只想在以后读取表，使用**myisampack**压缩它。参见15.1.3.3节，“压缩表特性”。
6. 用**myisamchk -r -q** */path/to/db/tbl_name*重新创建索引。这将在写入磁盘前在内存中创建索引树，并且它更快，因为避免了大量磁盘搜索。结果索引树也被完美地平衡。
7. 执行FLUSH TABLES语句或**mysqladmin flush-tables**命令。

请注意如果插入一个空MyISAM表，LOAD DATA INFILE也可以执行前面的优化；主要不同处是可以让**myisamchk**为创建索引分配更多的临时内存，比执行LOAD DATA INFILE语句时为服务器重新创建索引分配得要多。

也可以使用ALTER TABLE *tbl_name* DISABLE KEYS代替**myisamchk --keys-used=0 -rq** */path/to/db/tbl_name*，使用ALTER TABLE *tbl_name* ENABLE KEYS代替**myisamchk -r -q** */path/to/db/tbl_name*。使用这种方式，还可以跳过FLUSH TABLES。

· 锁定表可以加速用多个语句执行的INSERT操作：

- LOCK TABLES a WRITE;
- INSERT INTO a VALUES (1,23),(2,34),(4,33);
- INSERT INTO a VALUES (8,26),(6,29);
- UNLOCK TABLES;

这样性能会提高，因为索引缓存区仅在所有INSERT语句完成后刷新到磁盘上一次。一般有多少INSERT语句即有多少索引缓存区刷新。如果能用一个语句插入所有的行，就不需要锁定。

对于事务表，应使用BEGIN和COMMIT代替LOCK TABLES来加快插入。

锁定也将降低多连接测试的整体时间，尽管因为它们等候锁定最大等待时间将上升。例如：

Connection 1 does 1000 inserts

Connections 2, 3, and 4 do 1 insert

Connection 5 does 1000 inserts

如果不使用锁定，2、3和4将在1和5前完成。如果使用锁定，2、3和4将可能不在1或5前完成，但是整体时间应该快大约40%。

INSERT、UPDATE和DELETE操作在**MySQL**中是很快的，通过为在一行中多于大约5次连续不断地插入或更新的操作加锁，可以获得更好的整体性能。如果在一行中进行多次插入，可以执行LOCK TABLES，随后立即执行UNLOCK TABLES(大约每1000行)以允许其它的线程访问表。这也会获得好的性能。

INSERT装载数据比LOAD DATA INFILE要慢得多，即使是使用上述的策略。

· 为了对LOAD DATA INFILE和INSERT在MyISAM表得到更快的速度，通过增加key_buffer_size系统变量来扩大 键高速缓冲区。参见7.5.2节，“调节服务器参数”。

7.2.17. UPDATE语句的速度

更新查询的优化同SELECT查询一样，需要额外的写开销。写速度依赖于更新的数据大小和更新的索引的数量。没有更改的索引不被更新。

使更改更快的另一个方法是推迟更改然后在一行内进行多次更新。如果锁定表，同时做多个更新比一次做一个快得多。

请注意对使用动态记录格式的MyISAM表，更新一个较长总长的记录可能会切分记录。如果经常这样该，偶尔使用OPTIMIZE TABLE很重要。参见13.5.2.5节，“OPTIMIZE TABLE语法”。

7.2.18. DELETE语句的速度

删除一个记录的时间与索引数量确切成正比。为了更快速地删除记录，可以增加键高速缓冲的大小。参见7.5.2节，“调节服务器参数”。

如果想要删除一个表的所有行，使用TRUNCATE TABLE *tbl_name* 而不要用DELETE FROM *tbl_name*。参见13.2.9节，“TRUNCATE语法”。

7.2.19. 其它优化技巧

该节列出了提高查询速度的各种技巧：

- 使用持久的连接数据库以避免连接开销。如果不能使用持久的连接并且你正启动许多新的与数据库的连接，可能要更改thread_cache_size变量的值。参见7.5.2节，“调节服务器参数”。

- 总是检查所有查询确实使用已经在表中创建了的索引。在**MySQL**中，可以用EXPLAIN命令做到。参见7.2.1节，“EXPLAIN语法（获取关于SELECT的信息）”。

- 尝试避免在频繁更新的表上执行复杂的SELECT查询，以避免与锁定表有关的由于读、写冲突发生的问题。

- 对于没有删除的行的MyISAM表，可以在另一个查询正从表中读取的同时在末尾插入行。如果这很重要，应考虑按照避免删除行的方式使用表。另一个可能性是在删除大量行后运行OPTIMIZE TABLE。参见15.1节，“MyISAM存储引擎”。

- 要修复任何ARCHIVE表可以发生的压缩问题，可以执行OPTIMIZE TABLE。参见15.8节，“ARCHIVE存储引擎”。

- 如果你主要按*expr1*，*expr2*，...顺序检索行，使用ALTER TABLE ... ORDER BY *expr1*, *expr2*, ...。对表大量更改后使用该选项，可以获得更好的性能。

- 在一些情况下，使得基于来自其它表的列的信息引入一个“哈希”的列有意义。如果该列较短并且有合理的唯一值，它可以比在许多列上的一个大索引快些。在**MySQL**中，很容易使用这个额外列：

```
SELECT * FROM tbl_name
     WHERE hash_col=MD5(CONCAT(col1,col2))
     AND col1='constant' AND col2='constant';
```

- 对于频繁更改的MyISAM表，应试图避免所有变长列(VARCHAR、BLOB和TEXT)。如果表包括单一的变长列则使用动态记录格式。参见第15章：存储引擎和表类型。

- 只是因为行太大，将一张表分割为不同的表一般没有什么用处。为了访问行，最大的性能冲击是磁盘搜索以找到行的第一个字节。在找到数据后，大多数新型磁盘对大多数应用程序来说足够快，能读入整个行。确实有必要分割的唯一情形是如果它是使用动态记录格式使之变为固定的记录大小的MyISAM表(见上述)，或如果你需要很频繁地扫描表而不需要大多数列。参见第15章：存储引擎和表类型。

- 如果你需要很经常地计算结果，例如基于来自很多行的信息的计数，引入一个新表并实时更新计数器可

能更好一些。下面形式的更新会更快一些：

```
UPDATE tbl_name SET count_col=count_col+1 WHERE key_col=constant;
```

当你使用象MyISAM那样的只有表级锁定的**MySQL**存储引擎(多重读/单个写)时，这确实很重要。这也给大多数数据库较好的性能，因为行锁定管理器在这种情况下有较少的事情要做。

· 如果你需要从大的记录文件表中收集统计信息，使用总结性的表而不是扫描整个表。维护总结应该比尝试做"实时"统计要快些。当有变化时从日志重新生成新的总结表比改变运行的应用(取决于业务决策)要快得多。

- 如果可能，应该将报告分类为"实时"或"统计"，这里统计报告所需的数据仅仅基于从实际数据定期产生的总结表中产生。
- 充分利用列有默认值的事实。只有当插入的值不同于默认值时，才明确地插入值。这减少MySQL需要做的语法分析从而提高插入速度。
- 在一些情况下，包装并存储数据到一个BLOB列中是很方便的。在这种情况下，必须在你的应用中增加额外的代码来打包/解包信息，但是这种方法可以在某些阶段节省很多访问。当有不符合行和列表结构的数据时，这很实用。
- 在一般情况下，应该尝试以非冗余方式(查看数据库理论中的第三正则形式)保存数据，但是为了获得更快的速度，可以复制信息或创建总结表。
- 存储过程或UDF(用户定义函数)可能是获得更好性能的一个好方法，详细信息参见第20章：存储程序和函数和27.2节，"为MySQL添加新函数"。
- 总是能通过在应用程序中缓存查询/答案并尝试同时执行很多插入/更新来获得一些好处。如果数据库支持锁定表(象**MySQL**和Oracle)，这应该有助于确保索引缓存只在所有更新后刷新一次。还可以利用MySQL的查询缓存来获得类似的结果；参见5.13节，"MySQL查询高速缓冲"。
- 当不需要知道何时写入数据时，使用INSERT DELAYED。这样可以加快处理，因为很多记录可以通过一次磁盘写入被写入。
- 当你想要让选择显得更重要时，使用INSERT /*! LOW_PRIORITY */。
- 使用INSERT LOW_PRIORITY来取得插入队列的检索，也就是即使有另一个客户等待写入也要执行SELECT。
- 使用多行INSERT语句通过一个SQL命令来存储很多行(许多SQL服务器支持它，包括MySQL)。
- 使用LOAD DATA INFILE装载较大数量的数据。这比使用INSERT要快得多。
- 使用AUTO_INCREMENT列构成唯一值。
- 当MyISAM使用动态表格式时，偶尔使用OPTIMIZE TABLE可以避免碎片。参见15.1.3节，"MyISAM表的存储格式"。
- 可能时使用MEMORY表以得到更快的速度。参见15.4节，"MEMORY (HEAP)存储引擎"。
- 在Web服务器中，图象和其它二进制资产应该作为文件存储。也就是仅在数据库中存储的本文件的引用而不是文件本身。大多数Web服务器在缓存文件方面比数据库内容要好得多，因此使用文件一般要快得多。
- 对经常访问的不重要数据(如为没有在Web 浏览器中启用cookie的用户最后显示的标语的相关信息)使用内存表。在许多Web应用程序环境中也可以使用用户会话来处理可变状态数据。
- 在不同表中具有相同信息的列应该被声明为相同的并有相同的名字。尝试使名字简单化。例如，在customer表中使用name而不是customer_name。为了使名字能移植到其它SQL服务器，应该使名字短于18个字符。
- 如果确实需要很高的速度，应该研究一下不同SQL服务器支持的数据存储的低层接口！例如直接访问**MySQL** MyISAM存储引擎，比起使用SQL接口，速度可以提高2-5倍。为了能实现，数据必须与应用程序在同一台服务器上，并且通常只应该被一个进程访问(因为外部文件锁定确实很慢)。通过在**MySQL**服务器中引进低层MyISAM命令能消除以上问题(如果需要，这可能是获得更好性能的一个简单的方法)。通过精心设计数据库接口，应该能相当容易地支持这类优化。
- 如果正使用数字数据，在许多情况下，从一个数据库访问信息(使用实时连接)比访问一个文本文件快些。这是因为数据库中的信息比文本文件更紧凑，因此这将涉及更少的磁盘访问。还可以在应用程序中节省代码，因为不须分析文本文件来找出行和列的边界。

· 通过复制可以提高某些操作的性能。可以在复制服务器中分布客户的检索以均分负载。为了防止备份时

主服务器变慢，可以使用一个从服务器来备份。参见第 章： 中的复制。

· 用DELAY_KEY_WRITE=1选项声明MyISAM表可以使索引更新更快，因为在表关闭之前它们不刷新到硬盘上。不利之处是当表打开时如果杀掉服务器，应确保用--myisam-recover选项运行服务器保证没有问题，或者在重启服务器之前运行**myisamchk**。(然而，即使在这种情况下，应通过使用DELAY_KEY_WRITE保证不丢失数据，因为关键字信息总是可以从数据行产生）。

7.3. 锁定事宜

7.3.1. 锁定方法
7.3.2. 表锁定事宜

7.3.1. 锁定方法

MySQL 5.1支持对MyISAM和MEMORY表进行表级锁定，对BDB表进行页级锁定，对InnoDB表进行行级锁定。

在许多情况下，可以根据培训猜测应用程序使用哪类锁定类型最好，但一般很难说出某个给出的锁类型就比另一个好。一切取决于应用程序，应用程序的不同部分可能需要不同的锁类型。

为了确定是否想要使用行级锁定的存储引擎，应看看应用程序做什么并且混合使用什么样的选择和更新语句。例如，大多数Web应用程序执行许多选择，而很少进行删除，只对关键字的值进行更新，并且只插入少量具体的表。基本MySQL MyISAM设置已经调节得很好。

在MySQL中对于使用表级锁定的存储引擎，表锁定时不会死锁的。这通过总是在一个查询开始时立即请求所有必要的锁定并且总是以同样的顺序锁定表来管理。

对WRITE，**MySQL**使用的表锁定方法原理如下：

- 如果在表上没有锁，在它上面放一个写锁。
- 否则，把锁定请求放在写锁定队列中。

对READ，**MySQL**使用的锁定方法原理如下：

- 如果在表上没有写锁定，把一个读锁定放在它上面。
- 否则，把锁请求放在读锁定队列中。

当一个锁定被释放时，锁定可被写锁定队列中的线程得到，然后是读锁定队列中的线程。

这意味着，如果你在一个表上有许多更新，SELECT语句将等待直到没有更多的更新。

可以通过检查table_locks_waited和table_locks_immediate状态变量来分析系统上的表锁定争夺：

mysql> **SHOW STATUS LIKE 'Table%';**

```
+-----------------------+---------+
| Variable_name         | Value   |
+-----------------------+---------+
| Table_locks_immediate | 1151552 |
```

| Table_locks_waited | 15324 |

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

如果INSERT语句不冲突，可以自由为MyISAM表混合并行的INSERT和SELECT语句而不需要锁定。也就是说，你可以在其它客户正读取MyISAM表的时候插入行。如果数据文件中间不包含空闲块，不会发生冲突，因为在这种情况下，记录总是插入在数据文件的尾部。(从表的中部删除或更新的行可能导致空洞）。如果有空洞，当所有空洞填入新的数据时，并行的插入能够重新自动启用。

如果不能同时插入，为了在一个表中进行多次INSERT和SELECT操作，可以在临时表中插入行并且立即用临时表中的记录更新真正的表。

这可用下列代码做到:

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;**

InnoDB使用行锁定，BDB使用页锁定。对于这两种存储引擎，都可能存在死锁。这是因为，在SQL语句处理期间，InnoDB自动获得行锁定和BDB获得页锁定，而不是在事务启动时获得。

行级锁定的优点:

- 当在许多线程中访问不同的行时只存在少量锁定冲突。
- 回滚时只有少量的更改。
- 可以长时间锁定单一的行。

行级锁定的缺点:

- 比页级或表级锁定占用更多的内存。
- 当在表的大部分中使用时，比页级或表级锁定速度慢，因为你必须获取更多的锁。
- 如果你在大部分数据上经常进行GROUP BY操作或者必须经常扫描整个表，比其它锁定明显慢很多。
- 用高级别锁定，通过支持不同的类型锁定，你也可以很容易地调节应用程序，因为其锁成本小于行级锁定。

在以下情况下，表锁定优先于页级或行级锁定:

- 表的大部分语句用于读取。
- 对严格的关键字进行读取和更新，你可以更新或删除可以用单一的读取的关键字来提取的一行:
- `UPDATE tbl_name SET column=value WHERE unique_key_col=key_value;`
- `DELETE FROM tbl_name WHERE unique_key_col=key_value;`
- SELECT 结合并行的INSERT语句，并且只有很少的UPDATE或DELETE语句。

- 在整个表上有许多扫描或GROUP BY操作，没有任何写操作。

不同于行级或页级锁定的选项：

- 版本(例如，为并行的插入在MySQL中使用的技术)，其中可以一个写操作，同时有许多读取操作。这说明数据库或表支持数据依赖的不同视图，取决于访问何时开始。其它共同的术语是"时间跟踪"、"写复制"或者"按需复制"。

- 按需复制在许多情况下优先于页级或行级锁定。然而，在最坏的情况下，它可能比使用常规锁定使用更多的内存。

- 除了行级锁定外，你可以使用应用程序级锁定，例如在MySQL中使用GET_LOCK()和RELEASE_LOCK()。这些是建议性锁定，它们只能在运行良好的应用程序中工作。

7.3.2. 表锁定事宜

为达到最高锁定速度，除InnoDB和BDB之外，对所有存储引擎，MySQL使用表锁定(而不是页、行或者列锁定)。

对于InnoDB和BDB表，如果你用LOCK TABLES显式锁定表，MySQL只使用表锁定。对于这些表类型，我们建议你根本不要使用LOCK TABLES，因为InnoDB使用自动行级锁定而BDB使用页级锁定来保证事务隔离。

对于大表，对于大多数应用程序，表锁定比行锁定更好，但存在部分缺陷。

表锁定使许多线程同时从一个表中进行读取操作，但如果一个线程想要对表进行写操作，它必须首先获得独占访问。更新期间，所有其它想要访问该表的线程必须等待直到更新完成。

表更新通常情况认为比表检索更重要，因此给予它们更高的优先级。这应确保更新一个表的活动不能"饿死"，即使该表上有很繁重的SELECT活动。

表锁定在这种情况下会造成问题，例如当线程正等待，因为硬盘已满并且在线程可以处理之前必须有空闲空间。在这种情况下，所有想要访问出现问题的表的线程也被设置成等待状态，直到有更多的硬盘空间可用。

表锁定在下面的情况下也存在问题：

- 一个客户发出长时间运行的查询。

- 然后，另一个客户对同一个表进行更新。该客户必须等待直到SELECT完成。

- 另一个客户对同一个表上发出了另一个SELECT语句。因为UPDATE比SELECT优先级高，该SELECT语句等待UPDATE完成，并且等待第1个SELECT完成。

下面描述了一些方法来避免或减少表锁定造成的竞争：

- 试图使SELECT语句运行得更快。你可能必须创建一些摘要(summary)表做到这点。

- 用--low-priority-updates启动**mysqld**。这将给所有更新(修改)一个表的语句以比SELECT语句低的优先级。在这种情况下，在先前情形的第2个SELECT语句将在UPDATE语句前执行，而不需要等候第1个SELECT完成。

- 可以使用SET LOW_PRIORITY_UPDATES=1语句指定具体连接中的所有更新应使用低优先级。参见13.5.3节，"SET语法"

- 可以用LOW_PRIORITY属性给与一个特定的INSERT、UPDATE或DELETE语句较低优先级。

- 可以用HIGH_PRIORITY属性给与一个特定的SELECT语句较高优先级。参见13.2.7节，“SELECT语法”。

- 为**max_write_lock_count**系统变量指定一个低值来启动**mysqld**来强制MySQL在具体数量的插入完成后临时提高所有等待一个表的SELECT语句的优先级。这样允许在一定数量的WRITE锁定后给出READ锁定。

- 如果你有关于INSERT结合SELECT的问题，切换到使用新的MyISAM表，因为它们支持并发的SELECT和INSERT。

- 如果你对同一个表混合插入和删除，INSERT DELAYED将会有很大的帮助。参见13.2.4.2节，“INSERT DELAYED语法”。

- 如果你对同一个表混合使用SELECT和DELETE语句出现问题，DELETE的LIMIT选项可以有所帮助。参见13.2.1节，“DELETE语法”。

- 对SELECT语句使用SQL_BUFFER_RESULT可以帮助使表锁定时间变短。参见13.2.7节，“SELECT语法”。

- 可以更改mysys/thr_lock.c中的锁代码以使用单一的队列。在这种情况下，写锁定和读锁定将具有相同的优先级，对一些应用程序会有帮助。

这里是一些MySQL中表锁定相关的技巧：

- 如果不混合更新与需要在同一个表中检查许多行的选择，可以进行并行操作。

- 可以使用LOCK TABLES来提高速度，因为在一个锁定中进行许多更新比没有锁定的更新要快得多。将表中的内容切分为几个表也可以有所帮助。

- 如果在MySQL中表锁定时遇到速度问题，可以将你的表转换为InnoDB或BDB表来提高性能。参见15.2节，“InnoDB存储引擎”和15.5节，“BDB (BerkeleyDB)存储引擎”。

7.4. 优化数据库结构

7.4.1. 设计选择
7.4.2. 使你的数据尽可能小
7.4.3. 列索引
7.4.4. 多列索引
7.4.5. MySQL如何使用索引
7.4.6. MyISAM键高速缓冲
7.4.7. MyISAM索引统计集合
7.4.8. MySQL如何计算打开的表
7.4.9. MySQL如何打开和关闭表
7.4.10. 在同一个数据库中创建多个表的缺陷

7.4.1. 设计选择

MySQL将行数据和索引数据保存在不同的文件中。许多(几乎所有)其它数据库将行数据和索引数据混合保存在用一个文件中。我们认为MySQL 选择对广范围的现代系统更好一些。

保存行数据的另一种方式是将每个列的信息保存在单独的区域(例如SDBM和Focus)。这样会对每个访问多个列的查询造成性能问题。因为当访问多个列时退化得很快，我们认为该模型对一般数据库不合适。

更常见的情形是索引和数据保存在一起(例如Oracle/Sybase)。在这种情况下，你可以在索引的叶级页找到行的信息。该布局比较好的事情是在许多情况下，根据索引缓存得怎样，可以保存一个硬盘读取。该布局的不利之

处表现在:

- 表扫描要慢得多，因为你必须通读索引以获得数据。
- 你不能只使用表来检索查询的数据。
- 你需要使用更多的空间，因为你必须从节点复制索引(你不能保存节点上的行)。
- 删除要经过一段时间后才退化表(因为删除时通常不会更新节点上的索引)。
- 只缓存索引数据会更加困难。

7.4.2. 使你的数据尽可能小

最基本的优化之一是使表在磁盘上占据的空间尽可能小。这能给出巨大的改进，因为磁盘读入较快，并且在查询执行过程中小表的内容被处理时占用较少的主存储器。如果在更小的列上做索引，索引也占据较少的资源。

MySQL支持许多不同的存储引擎(表类型)和行格式。对于每个表，可以确定使用哪个存储引擎和索引方法。为应用程序选择合适的表格式可以大大提高性能。参见第15章：存储引擎和表类型。

可以使用下面的技术可以使表的性能更好并且使存储空间最小:

- 尽可能地使用最有效(最小)的数据类型。**MySQL**有很多节省磁盘空间和内存的专业化类型。
- 尽可能使用较小的整数类型使表更小。例如，MEDIUMINT经常比INT好一些，因为MEDIUMINT列使用的空间要少25%。
- 如果可能，声明列为NOT NULL。它使任何事情更快而且每列可以节省一位。注意如果在应用程序中确实需要NULL，应该毫无疑问使用它，只是避免 默认地在所有列上有它。
- 对于MyISAM表，如果没有任何变长列(VARCHAR、TEXT或BLOB列)，使用固定尺寸的记录格式。这比较快但是不幸地可能会浪费一些空间。参见15.1.3节，“MyISAM表的存储格式”。即使你已经用CREATE选项让VARCHAR列ROW_FORMAT=fixed，也可以提示想使用固定长度的行。
- 在MySQL/InnoDB中，InnoDB表使用更紧凑的存储格式。在以前版本的MySQL中，InnoDB记录包含一些冗余信息，例如列数目和每个列的长度，即使对于固定大小的列。默认情况，创建的表为紧凑格式(ROW_FORMAT=COMPACT)。如果想要降级旧版本的MySQL/InnoDB，可以用ROW_FORMAT=REDUNDANT要求旧的格式。
- 紧凑的InnoDB格式也改变了包含UTF-8数据的CHAR列的保存方式。在ROW_FORMAT=REDUNDANT格式中，UTF-8 CHAR(*n*)占用3*_n_字节，假定UTF-8编码的字符的最大长度是3字节。许多语言可以主要用单字节UTF-8字符来编写，固定的存储长度通常会浪费空间。通过根据需要剥离尾部的空格，ROW_FORMAT=COMPACT格式为这些列分配可变数量的n..3*n字节。最小存储长度按顺序保存为n字节，以在典型情况下帮助更新。
- 每张表的主索引应该尽可能短。这使一行的识别容易而有效。
- 只创建你确实需要的索引。索引对检索有好处，但是当你需要快速存储东西时就变得糟糕。如果主要通过搜索列的组合来存取一个表，对它们做一个索引。第一个索引部分应该是最常用的列。如果从表中选择时总是使用许多列，应该首先以更多的副本使用列以获得更好的索引压缩。
- 如果很可能一个索引在头几个字符上有唯一的前缀，仅仅索引该前缀比较好。**MySQL**支持对一个字符列的最左边部分创建一个索引(参见13.1.4节，“CREATE INDEX语法”)。更短的索引会更快，不仅因为它们占较少的磁盘空间，而且因为它们将在索引缓存中提供更多的访问，因此磁盘搜索更少。参见7.5.2节，“调节服务器参数”。

- 在一些情形下，将一个经常被扫描的表分割为2个表是有益的。特别是如果它是一个动态格式的表，并且可能使用一个扫描表时能用来找出相关行的较小静态格式的表。

7.4.3. 列索引

所有MySQL列类型可以被索引。对相关列使用索引是提高SELECT操作性能的最佳途径。

根据存储引擎定义每个表的最大索引数和最大索引长度。参见第15章：存储引擎和表类型。所有存储引擎支持每个表至少16个索引，总索引长度至少为256字节。大多数存储引擎有更高的限制。

在索引定义中用*col_name*(*length*)语法，你可以创建一个只使用CHAR或VARCHAR列的第1个*length*字符的索引。按这种方式只索引列值的前缀可以使索引文件小得多。

MyISAM和InnoDB存储引擎还支持对BLOB和TEXT列的索引。当索引一个BLOB或TEXT列时，你必须为索引指定前缀长度。例如：

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

在MySQL 5.1中，对于MyISAM和InnoDB表，前缀可以达到1000字节长。请注意前缀的限制应以字节为单位进行测量，而CREATE TABLE语句中的前缀长度解释为字符数。当为使用多字节字符集的列指定前缀长度时一定要加以考虑。

还可以创建FULLTEXT索引。该索引可以用于全文搜索。只有MyISAM存储引擎支持FULLTEXT索引，并且只为CHAR、VARCHAR和TEXT列。索引总是对整个列进行，不支持局部(前缀)索引。详情参见12.7节，“全文搜索功能”。

也可以为空间列类型创建索引。只有MyISAM存储引擎支持空间类型。空间索引使用R-树。

默认情况MEMORY(HEAP)存储引擎使用hash索引，但也支持B-树索引。

7.4.4. 多列索引

MySQL可以为多个列创建索引。一个索引可以包括15个列。对于某些列类型，可以索引列的前缀(参见7.4.3节，“列索引”)。

多列索引可以视为包含通过连接索引列的值而创建的值的排序的数组。

MySQL按这样的方式使用多列索引：当你在WHERE子句中为索引的第1个列指定已知的数量时，查询很快，即使你没有指定其它列的值。

假定表具有下面的结构：

```
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)
);
```

name索引是一个对last_name和first_name的索引。索引可以用于为last_name，或者为last_name和first_name在已知范围内指定值的查询。因此，name索引用于下面的查询：

```
SELECT * FROM test WHERE last_name='Widenius';


SELECT * FROM test
```

```
    WHERE last_name='Widenius' AND first_name='Michael';


SELECT * FROM test
    WHERE last_name='Widenius'
    AND (first_name='Michael' OR first_name='Monty');


SELECT * FROM test
    WHERE last_name='Widenius'
    AND first_name >='M' AND first_name < 'N';
```

然而，name索引不用于下面的查询：

```
SELECT * FROM test WHERE first_name='Michael';


SELECT * FROM test
    WHERE last_name='Widenius' OR first_name='Michael';
```

MySQL使用索引提高查询性能的方式将在7.4.5节，“MySQL如何使用索引”中讨论。

7.4.5. MySQL如何使用索引

索引用于快速找出在某个列中有一特定值的行。不使用索引，MySQL必须从第1条记录开始然后读完整个表直到找出相关的行。表越大，花费的时间越多。如果表中查询的列有一个索引，MySQL能快速到达一个位置去搜寻到数据文件的中间，没有必要看所有数据。如果一个表有1000行，这比顺序读取至少快100倍。注意如果你需要访问大部分行，顺序读取要快得多，因为此时我们避免磁盘搜索。

大多数MySQL索引(PRIMARY KEY、UNIQUE、INDEX和FULLTEXT)在B树中存储。只是空间列类型的索引使用R-树，并且MEMORY表还支持hash索引。

字符串自动地压缩前缀和结尾空格。参见13.1.4节，“CREATE INDEX语法”。

总的来说，按后面的讨论使用索引。本节最后描述hash索引(用于MEMORY表)的特征。

索引用于下面的操作：

- 快速找出匹配一个WHERE子句的行。
- 删除行。如果可以在多个索引中进行选择，MySQL通常使用找到最少行的索引。
- 当执行联接时，从其它表检索行。
- 对具体有索引的列*key_col*找出MAX()或MIN()值。由预处理器进行优化，检查是否对索引中在key_*col*之前发生所有关键字元素使用了WHERE *key_part_# = constant*。在这种情况下，MySQL为每个MIN()或MAX()表达式执行一次关键字查找，并用常数替换它。如果所有表达式替换为常量，查询立即返回。例如：
- SELECT MIN(*key_part2*),MAX(*key_part2*)
- FROM *tbl_name* WHERE *key_part1*=10;
- 如果对一个可用关键字的最左面的前缀进行了排序或分组(例如，ORDER BY key_part_1,key_part_2)，

排序或分组一个表。如果所有关键字元素后面有DESC，关键字以倒序被读取。参见7.2.12节，“MySQL如何优化ORDER BY”。

- 在一些情况中，可以对一个查询进行优化以便不用查询数据行即可以检索值。如果查询只使用来自某个表的数字型并且构成某些关键字的最左面前缀的列，为了更快，可以从索引树检索出值。

```
SELECT key_part3 FROM tbl_name
    WHERE key_part1=1
```

假定你执行下面的SELECT语句：

```
mysql> SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;
```

如果col1和col2上存在一个多列索引，可以直接取出相应行。如果col1和col2上存在单列索引，优化器试图通过决定哪个索引将找到更少的行来找出更具限制性的索引并且使用该索引取行。

如果表有一个多列索引，优化器可以使用最左面的索引前缀来找出行。例如，如果有一个3列索引(col1,col2,col3)，则已经对(col1)、(col1,col2)和(col1,col2,col3)上的搜索进行了索引。

如果列不构成索引最左面的前缀，**MySQL**不能使用局部索引。假定有下面显示的SELECT语句。

```
SELECT * FROM tbl_name WHERE col1=val1;
SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;

SELECT * FROM tbl_name WHERE col2=val2;
SELECT * FROM tbl_name WHERE col2=val2 AND col3=val3;
```

如果 (col1，col2，col3)有一个索引，只有前2个查询使用索引。第3个和第4个查询确实包括索引的列，但(col2)和(col2，col3)不是 (col1，col2，col3)的最左边的前缀。

也可以在表达式通过=、>、>=、<、<=或者BETWEEN操作符使用B-树索引进行列比较。如果LIKE的参数是一个不以通配符开头的常量字符串，索引也可以用于LIKE比较。例如，下面的SELECT语句使用索引：

```
SELECT * FROM tbl_name WHERE key_col LIKE 'Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE 'Pat%_ck%';
```

在第1个语句中，只考虑带'Patrick' <=key_*col* < 'Patricl'的行。在第2个语句中，只考虑带'Pat' <=key_*col* < 'Pau'的行。

下面的SELECT语句不使用索引：

```
SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE other_col;
```

在第一条语句中，LIKE值以一个通配符字符开始。在第二条语句中，LIKE值不是一个常数。

如果使用... LIKE '%*string*%'并且*string*超过3个字符，MySQL使用*Turbo Boyer-Moore*算法初始化字符串的模式然后使用该模式来更快地进行搜索。

如果*col_name*被索引，使用*col_name* IS NULL的搜索将使用索引。

任何不跨越WHERE子句中的所有AND级的索引不用于优化查询。换句话说，为了能够使用索引，必须在每

个AND组中使用索引前缀。

下面的WHERE子句使用索引:

```
... WHERE index_part1=1 AND index_part2=2 AND other_column=3
    /* index = 1 OR index = 2 */
... WHERE index=1 OR A=10 AND index=2
    /* optimized like "index_part1='hello'" */
... WHERE index_part1='hello' AND index_part3=5
    /* Can use index on index1 but not on index2 or index3 */
... WHERE index1=1 AND index2=2 OR index1=3 AND index3=3;
```

下面的WHERE子句不使用索引:

```
    /* index_part1 is not used */
... WHERE index_part2=1 AND index_part3=2


    /*  Index is not used in both parts of the WHERE clause  */
... WHERE index=1 OR A=10


    /* No index spans all rows  */
... WHERE index_part1=1 OR index_part2=10
```

有时MySQL不使用索引，即使有可用的索引。一种情形是当优化器估计到使用索引将需要MySQL访问表中的大部分行时。(在这种情况下，表扫描可能会更快些，因为需要的搜索要少）。然而，如果此类查询使用LIMIT只搜索部分行，MySQL则使用索引，因为它可以更快地找到几行并在结果中返回。

Hash索引还有一些其它特征:

- 它们只用于使用=或<=>操作符的等式比较(但很快)。它们用于比较 操作符，例如发现范围值的<。
- 优化器不能使用hash索引来加速ORDER BY操作。(该类索引不能用来按顺序搜索下一个条目）。
- MySQL不能确定在两个值之间大约有多少行(这被范围优化器用来确定使用哪个索引)。如果你将一个MyISAM表改为hash-索引的MEMORY表，会影响一些查询。
- 只能使用整个关键字来搜索一行。(用B-树索引，任何关键字的最左面的前缀可用来找到行）。

7.4.6. MyISAM键高速缓冲

7.4.6.1. 共享键高速缓冲访问
7.4.6.2. 多键高速缓冲
7.4.6.3. 中点插入策略
7.4.6.4. 索引预加载
7.4.6.5. 键高速缓冲块大小
7.4.6.6. 重构键高速缓冲

为了使硬盘I/O最小化，MyISAM存储引擎使用一个被许多数据库管理系统使用的策略。它使用一个缓存机制将经常访问的表锁在内存中:

- 对于索引块，维护一个称之为键高速缓冲(或键高速缓冲区)的特殊结构。该结构包含大量块缓存区，其中放置了最常用的索引块。

- 对于数据块，MySQL不使用特殊缓存。而使用原生的操作系统文件系统的缓存。

本节首先描述了MyISAM键高速缓冲的基本操作。然后讨论了提高 键高速缓冲性能并使你更好地控制缓存操作的最新的更改：

- 多个线程可以并行访问缓存。

- 可以设置多个键高速缓冲，并将表索引指定给具体缓存。

可以使用key_buffer_size系统变量控制 键高速缓冲的大小。如果该变量设置为零，不使用键高速缓冲。如果key_buffer_size值太小不能分配最小数量的块缓存区(8)，也不使用 键高速缓冲。

如果键高速缓冲不工作，只使用操作系统提供的原生文件系统缓存区访问索引文件。(换句话说，使用与表数据块相同的策略表来访问索引块）。

索引块是一个连续的访问MyISAM索引文件的单位。通常一个索引块的大小等于索引B-树节点的大小。(在硬盘上使用B-树数据结构表示索引。树底部的节点为叶子节点。叶子节点上面的节点为非叶子节点）。

键高速缓冲结构中的所有块缓存区大小相同。该大小可以等于、大于或小于表索引块的大小。通常这两个值中的一个是另一个的几倍。

当必须访问表索引块中的数据时，服务器首先检查是否它可以用于键高速缓冲中的某些块缓存区。如果适用，服务器访问键高速缓冲中的数据而不是硬盘上的数据。也就是说，从缓存读取或写入缓存，而不是从硬盘读写。否则，服务器选择一个包含一个不同的表索引块的缓存块缓存区，并用需要的表索引块的拷贝替换那里的数据。一旦新的索引块位于缓存中，可以访问索引数据。

如果用于替换的块已经被修改了，块被视为"脏了"。在这种情况下，在替换前，其内容被刷新到它来自的表索引。

通常服务器遵从*LRU(*最近最少使用*)*策略：当选择一个块用于替换时，它选择最近最少使用的索引块。为了使该选择更容易， 键高速缓冲模块维护所有使用的块的专门队列(*LRU*链)。当访问块时，它被放到队列最后。当块需要替换时，队列开头的块是最近最少使用的块，并成为第1个候选者。

7.4.6.1. 共享键高速缓冲访问

在以下条件下，线程可以同时访问键高速缓冲缓存区：

- 没有被更新的缓存区可以被多个线程访问。

- 正被更新的缓存区让需要使用它的线程等待直到更新完成。

- 多个线程可以发起请求替换缓存块，只要它们不彼此干扰(也就是说，只要它们需要不同的索引块，并且使不同的缓存块被替换)。

对键高速缓冲的共享访问允许服务器大大提高吞吐量。

7.4.6.2. 多键高速缓冲

对键高速缓冲的共享访问可以提高性能但不能完全消除线程之间的竞争。它们仍然竞争对键高速缓冲缓存区的访问进行管理的控制结构。为了进一步降低 键高速缓冲访问竞争，MySQL 5.1还提供了多个键高速缓冲，允许你为不同的键高速缓冲分配不同的表索引。

有多个键高速缓冲时，当为给定的MyISAM表处理查询时，服务器必须知道使用哪个缓存。默认情况，所有MyISAM表索引被缓存到默认 键高速缓冲中。要想为具体键高速缓冲分配表索引，应使用CACHE INDEX语句(参见13.5.5.1节，“CACHE INDEX语法”)。

例如，下面的语句将表t1、t2和t3的索引分配给名为hot_cache的 键高速缓冲：

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+---------+--------------------+----------+----------+
| Table   | Op                 | Msg_type | Msg_text |
+---------+--------------------+----------+----------+
| test.t1 | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
| test.t3 | assign_to_keycache | status   | OK       |
+---------+--------------------+----------+----------+
```

可以用SET GLOBAL参数设置语句或使用服务器启动选项设置在CACHE INDEX语句中引用的键高速缓冲的大小来创建键高速缓冲。例如：

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

要想删除键高速缓冲，将其大小设置为零：

```
mysql> SET GLOBAL keycache1.key_buffer_size=0;
```

请注意不能删除默认键高速缓冲。删除默认键高速缓冲的尝试将被忽略：

```
mysql> set global key_buffer_size = 0;

mysql> show variables like 'key_buffer_size';
+-----------------+---------+
| Variable_name   | Value   |
+-----------------+---------+
| key_buffer_size | 8384512 |
+-----------------+---------+
```

键高速缓冲变量是结构式系统变量，有一个名和组件。对于keycache1.key_buffer_size，keycache1是缓存变量名，key_buffer_size是缓存组件。关于引用结构式 键高速缓冲系统变量所使用的语法的描述，参见9.4.1节，“结构式系统变量”

默认情况下，表索引被分配给服务器启动时创建的主要(默认)键高速缓冲。当 键高速缓冲被删除后，所有分配给它的索引被重新分配给默认键高速缓冲。

对于一个忙的服务器，我们建议采用使用三个键高速缓冲的策略：

- 占用为所有键高速缓冲分配的空间的20%的“热”键高速缓冲。该缓存用于频繁用于搜索但没有更新的表。
- 占用为所有键高速缓冲分配的空间的20%的“冷”键高速缓冲。该缓存用于中等大小、大量修改的表，例

如临时表。

· 占用键高速缓冲空间的20%的"温"键高速缓冲。使用它作为默认 键高速缓冲，默认情况被所有其它表使用。

使用3个键高速缓冲有好处的一个原因是对一个键高速缓冲结构的访问不会阻挡对其它的访问。访问分配给一个缓存的表的查询不会与访问分配给其它缓存的表的查询竞争。由于其它原因也会提高性能：

· 热缓存只用于检索查询，因此其内容决不会被修改。结果是，无论何时需要从硬盘上拉入索引块，选择用于替换的缓存块的内容不需要先刷新。

· 对于分配给热缓存的索引，如果没有查询需要索引扫描，很有可能对应索引B-树的非叶子节点的索引块仍然在缓存中。

· 当更新的节点位于缓存中并且不需要先从硬盘读入时，为临时表频繁执行的更新操作会执行得更快。如果临时表的索引的大小可以与冷键高速缓冲相比较，很可能更新的节点位于缓存中。

CACHE INDEX在一个表和 键高速缓冲之间建立一种联系，但每次服务器重启时该联系被丢失。如果你想要每次服务器重启时该联系生效，一个发办法是使用选项文件：包括配置 键高速缓冲的变量设定值，和一个init-file选项用来命名包含待执行的CACHE INDEX语句的一个文件。例如：

```
key_buffer_size = 4G

hot_cache.key_buffer_size = 2G

cold_cache.key_buffer_size = 2G

init_file=/path/to/data-directory/mysqld_init.sql
```

每次服务器启动时执行mysqld_init.sql中的语句。该文件每行应包含一个SQL语句。下面的例子分配几个表，分别对应hot_cache和cold_cache：

```
CACHE INDEX a.t1, a.t2, b.t3 IN hot_cache

CACHE INDEX a.t4, b.t5, b.t6 IN cold_cache
```

7.4.6.3. 中点插入策略

默认情况，键高速缓冲管理系统采用LRU策略选择要收回的键高速缓冲块，但它也支持更复杂的方法，称之为"中点插入策略"。

当使用中点插入策略时，LRU链被分为两个部分：一条热子链和一条温子链。两部分之间的划分点不固定，但键高速缓冲管理系统关注温部分不"太短"，总是包含至少key_cache_division_limit比例的 键高速缓冲块。key_cache_division_limit是结构式 键高速缓冲变量的一个组件，因此其值是一个可以根据每个缓存进行设置的参数。

当一个索引块从表中读入键高速缓冲，它被放入温子链的末端。经过一定量的访问后(访问块)，它被提升给热子链。目前，需要用来提升一个块(3)的访问次数与所有索引块的相同。

提升到热子链的块被放到子链的末端。块然后在该子链中循环。如果块在子链的开头停留足够长的时间，它被降到温链。该时间由键高速缓冲key_cache_age_threshold组件的值确定。

对于包含*N*个块的 键高速缓冲，阈值表示，热子链开头的没有在最后*N* *key_cache_age_threshold/100次访问中被访问的块将被移动到温子链开头。该块然后变为第1个挤出的候选者，因为替换的块总是来自温子链的开头。

中点插入策略允许你将更有价值的块总是在缓存中。如果你想使用简单的LRU策略，使key_cache_division_limit值保持其默认值100。

若执行的查询要求索引扫描有效推出所有索引块对应有数值的高级B-树节点的缓存，中点插入策略可以帮助提高性能。要想避免，必须使用中点插入策略，而key_cache_division_limit设置为远小于100。然后在索引扫描操作过程中，有数值的经常访问的节点被保留在热子链中。

7.4.6.4. 索引预加载

如果键高速缓冲内有足够的块以容纳整个索引的块，或者至少容纳对应其非叶节点的块，则在使用前，预装含索引块的键高速缓冲很有意义。预装可以以更有效的方式将表索引块放入 键高速缓冲缓存区中：通过顺序地从硬盘读取索引块。

不进行预装，块仍然根据查询需要放入键高速缓冲中。尽管块将仍然在缓存中（因为有足够的缓存区保存它们），它们以随机方式从硬盘上索取，而不是以顺序方式。

要想将索引预装到缓存中，使用LOAD INDEX INTO CACHE语句。例如，下面的语句可以预装表t1和t2索引的节点(索引块)：

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+---------+--------------+----------+----------+
| Table   | Op           | Msg_type | Msg_text |
+---------+--------------+----------+----------+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+---------+--------------+----------+----------+
```

IGNORE LEAVES修改器只允许预装索引非叶节点所用的块。这样，上述的语句预装t1中的所有索引块，但只预装t2中的非叶节点对应的块。

如果已经使用CACHE INDEX语句为一个索引分配了一个键高速缓冲，预装可以将索引块放入该缓存。否则，索引被装入默认键高速缓冲。

7.4.6.5. 键高速缓冲块大小

可以使用key_cache_block_size变量为具体的 键高速缓冲指定块缓存区的大小。这样允许为索引文件调节I/O操作的性能。

当读缓存区的大小等于原生操作系统I/O缓存区的大小时，可以获得I/O操作的最佳性能。但是将关键字节点的大小设置为等于I/O缓存区的大小并不总是能保证最佳整体性能。当读取大的叶节点时，服务器读入大量的不需要的数据，结果防止读入其它叶子的节点。

目前，你不能控制表内索引块的大小。该大小由服务器在创建.MYI索引文件时设置，取决于表定义中索引的关键字大小。在大多数情况下，它被设置为与I/O缓存区大小相等。

7.4.6.6. 重构键高速缓冲

键高速缓冲可以通过更新其参数值随时重新构建。例如：

```
mysql> SET GLOBAL cold_cache。key_buffer_size=4*1024*1024;
```

如果你为key_buffer_size或key_cache_block_size键高速缓冲组件分配的值与组件当前的值不同，服务器将毁掉缓存的旧结构并根据新值创建一个新的。如果缓存包含任何脏的块，服务器在销毁前将它们保存到硬盘上并重新创建缓存。如果你设置其它 键高速缓冲参数，则不会发生重新构建。

当重新构建键高速缓冲时，服务器首先将任何脏缓存区的内容刷新到硬盘上。之后，缓存内容不再需要。然而，重新构建并不阻塞需要使用分配给缓存的索引的查询。相反，服务器使用原生文件系统缓存直接访问表索引。文件系统缓存不如使用 键高速缓冲有效，因此尽管查询可以执行，但速度会减慢。缓存被重新构建后，它又可以缓存分配给它的索引了，并且索引不再使用文件系统缓存。

7.4.7. MyISAM索引统计集合

存储引擎搜集优化器使用的表的统计信息。表统计基于数数值组，其中数数值组是一系列有相同的关键字前缀值的记录。对于优化器，重要的统计即为数数值组的平均大小。

MySQL用下述方式使用平均数数值组:

- 估计必须为每个ref访问读取多少行
- 估计部分联接将产生多少行；也就是说，下述形式的操作将产生的行数:
- `(...) JOIN tbl_name ON tbl_name.key = expr`

随着索引的平均数数值组大小的增加，索引将更没有用，因为每个查找的平均行数增加：为了让索引有利于优化目的，最好是每个索引值对应表内的少量行数。当某个给定的索引值产生较多行时，索引更加没有用，MySQL更不可能使用它。

平均数数值组大小与表的集的势相关，即数数值组的数目。SHOW INDEX语句显示集的势值（基于*N*/*S*），其中*N*是表内的记录数，*S*是平均数数值组大小。该比例产生表内数数值组的大约数。

对于基于<=>比较 操作符的联接，NULL并不视为与任何其它值不同：NULL <=> NULL，正如对于其它*N*，*N* <=> *N*。

然而，对于基于=操作符的联接，NULL与非NULL值不同：当*expr1*或*expr2*(或两者)为NULL时，*expr1* = *expr2*不为真。这样影响比较形式*tbl_name.key* = *expr*的ref访问：如果*expr*当前的值为NULL，MySQL不会访问表，因为比较不能为真。

对于=比较，表内有多少NULL值并不重要。为了优化目的，相关值为非NULL数值组的平均大小。然而，MySQL目前不允许搜集或使用该平均大小。

对于MyISAM表，你可以使用myisam_stats_method系统变量部分控制表统计信息的搜集。该变量有两个可能的不同值，如下所示:

- 当myisam_stats_method为nulls_equal时，所有NULL值被视为相等的(也就是说，它们都形成一个数值组)。

如果NULL数值组大小远大于平均非NULL数值组大小，该方法向上倾斜平均数数值组大小。这样使索引对于优化器来说比它实际为查找非NULL值的联接更加没有用。结果是，nulls_equal方法会使优化器进行ref访问时本应使用索引而没有使用。

- 当myisam_stats_method为nulls_unequal时，NULL值不视为相同。相反，每个NULL值形成一个单独的数值组，大小为1。

如果你有许多NULL值，该方法向下倾斜平均数数值组大小。如果平均非NULL数值组较大，统计大小为1的每个组的NULL值会使优化器过高估计查找非NULL值的联接的索引值。结果是，当其它方法会更好时，nulls_unequal方法会使优化器为ref查找使用该索引。

如果你要使用许多使用<=>而不是=的联接，在比较过程中NULL值并不特殊，一个NULL等于另一个NULL。在这种情况下，nulls_equal是合适的统计方法。

myisam_stats_method系统变量有全局和会话值。设置全局值会影响MyISAM 为所有MyISAM表的统计的搜集。设置会话值只影响当前客户连接的统计的搜集。这说明你可以强制用给定的方法重新生成表的统计的搜集，而不需要因为设置myisam_stats_method的会话值而影响其它客户。

可以使用下面任一方法来重新生成表的统计信息：

- 设置myisam_stats_method，然后执行CHECK TABLE语句
- 执行**myisamchk --stats_method=method_name --analyze**
- 更改表，使其统计信息不为最新(例如，插入一行然后删除它)，然后设置myisam_stats_method并执行ANALYZE TABLE语句

使用myisam_stats_method的一些警告：

你可以强制显式搜集表的统计信息，如上所述。然而，MySQL也可以自动搜集统计信息。例如，如果在为表执行语句的过程中，一些语句修改了表，MySQL可以搜集统计信息。(例如，大批插入或删除，或者执行ALTER TABLE语句时可能发生）。如果发生，使用myisam_stats_method此时所有的值搜集统计信息。这样，如果你使用一个方法搜集统计信息，但当后面自动搜集一个表的统计信息时myisam_stats_method被设置为另一个方法，将使用其它方法。

对于给定的MyISAM表，还不能说出使用哪个方法来产生统计信息。

myisam_stats_method只适合MyISAM表。其它存储引擎只有一个方法来搜集表的统计信息。通常它接近于nulls_equal方法。

7.4.8. MySQL如何计算打开的表

当运行**mysqladmin status**时，将看见象这样的一些东西：

```
Uptime: 426 Running threads: 1 Questions: 11082

Reloads: 1 Open tables: 12
```

如果你仅有6个表，Open tables值为12可能有点令人困惑。

MySQL是多线程的，因此许多客户可以同时在同一个表上进行查询。为了使多个客户线程在同一个表上有不同状态的问题减到最小，表被每个并发进程独立地打开。这样需要额外的内存但一般会提高性能。对于MyISAM表，数据文件需要为每个打开表的客户提供一个额外的文件描述符。(索引文件描述符在所有线程之间共享)。

下一节中提供了该主题的更多的信息。参见7.4.9节，“MySQL如何打开和关闭表”。

7.4.9. MySQL如何打开和关闭表

table_cache、max_connections和max_tmp_tables系统变量影响服务器保持打开的文件的最大数量。如果你增加这些值其中的一个或两个，会遇到操作系统为每个进程打开文件描述符的数量强加的限制。许多操作系统允许你增加打开的文件的限制，尽管该方法随系统的不同而不同。查阅操作系统文档以确定是否可以增加限制以及如何操作。

table_cache与max_connections有关。例如，对于200个并行运行的连接，应该让表的缓存至少有200 * *N*，这里*N*是可以执行的查询的一个联接中表的最大数量。还需要为临时表和文件保留一些额外的文件描述符。

确保操作系统可以处理table_cache设置所指的打开的文件描述符的数目。如果table_cacheis设得太高，MySQL可能为文件描述符耗尽资源并拒绝连接，不能执行查询，并且很不可靠。还必须考虑到MyISAM存储引擎需要为每个打开的表提供两个文件描述符。可以在**mysqld_safe**中使用--open-files-limit启动选项来增加MySQL适用的文件描述符的数量。参见A.2.17节，“文件未找到”。

打开表的缓存可以保持在table_cache条。 默认为64；可以用**mysqld**的--table_cache选项来更改。请注意MySQL可以临时打开更多的 表以执行查询。

在下面的条件下，未使用的表将被关闭并从表缓存中移出：

- 当缓存满了并且一个线程试图打开一个不在缓存中的表时。
- 当缓存包含超过table_cache个条目，并且缓存中的表不再被任何线程使用。
- 当表刷新操作发生。当执行FLUSH TABLES语句或执行**mysqladmin flush-tables**或**mysqladmin refresh**命令时会发生。

当表缓存满时，服务器使用下列过程找到一个缓存入口来使用：

- 当前未使用的表被释放，以最近最少使用顺序。
- 如果缓存满了并且没有表可以释放，但是一个新表需要打开，缓存必须临时被扩大。

如果缓存处于一个临时扩大状态并且一个表从在用变为不在用状态，它被关闭并从缓存中释放。

对每个并发访问打开一个表。这意味着，如果2个线程访问同一个表或在同一个查询中访问表两次(例如，将表连接为自身时)，表需要被打开两次。每个并行的打开要求在表缓存中有一个条目。任何表的第一次打开占2个文件描述符：一个用于数据文件另一个用于索引文件。表的每一次额外使用仅占一个数据文件的文件描述符。索引文件描述符在所有线程之间共享。

如果你正用HANDLER *tbl_name* OPEN语句打开一个表，将为该线程专门分配一个表。该表不被其它线程共享，只有线程调用HANDLER *tbl_name* CLOSE或线程终止后才被关闭。表关闭后，被拉回表缓存中(如果缓存不满)。参见13.2.3节，“HANDLER语法”。

可以通过检查**mysqld**的状态变量Opened_tables确定表缓存是否太小：

```
mysql> SHOW STATUS LIKE 'Opened_tables';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Opened_tables | 2741  |
+---------------+-------+
```

如果值很大，即使你没有发出许多FLUSH TABLES语句，也应增加表缓存的大小。参见5.3.3节，“服务器系统变量”和5.3.4节，“服务器状态变量”。

7.4.10. 在同一个数据库中创建多个表的缺陷

如果在同一个数据库目录中有许多MyISAM表，打开、关闭和创建操作将会很慢。如果对许多不同的表执行SELECT语句，当表缓存满时，将有一点开销，因为对每个必须打开的表，另外一个必须被关闭。可以通过使表缓存更大些来减少这个开销。

7.5. 优化MySQL服务器

7.5.1. 系统因素和启动参数的调节
7.5.2. 调节服务器参数
7.5.3. 控制查询优化器的性能
7.5.4. 编译和链接怎样影响MySQL的速度
7.5.5. MySQL如何使用内存
7.5.6. MySQL如何使用DNS

7.5.1. 系统因素和启动参数的调节

我们从系统级因素开始，因为必须尽早地进行部分决策以获得较大性能。在其它情况下，快速浏览该节就足够了。但是，了解一下更改该层次的参数能够获得多少性能提高是很有意义的。

使用的操作系统很重要。为了更好地使用多CPU机器，应使用Solaris(因为其线程工作得很好)或Linux(因为2.4和以后的内核有很好的SMP支持)。请注意默认情况旧的Linux内核有一个2GB的文件大小限制。如果有这样的一个内核并且需要文件大于2GB，应得到ext2文件系统的大文件支持(LFS)补丁。其它文件系统例如ReiserFS和XFS没有此2GB限制。

将MySQL用于生产前，我们建议你在想用的平台上对它进行测试。

其它技巧:

- 如果有足够的RAM，可以移除所有的交换设备。有些操作系统即使有自由内存也使用交换设备。
- 使用--skip-external-locking MySQL选项以避免外部锁定。该选项默认开启。

请注意只要你只运行一个服务器，--skip-external-locking选项不影响MySQL的功能。只要记住运行**myisamchk**之前关闭服务器(或锁定并刷新相关表)。在一些系统上该选项是强制的，因为在任何情况下外部锁定均不工作。

不能使用--skip-external-locking的唯一情况是对相同的数据运行多个MySQL服务器(非客户)的情况，或者如果没有事先告诉服务器刷新并锁定一个表即运行**myisamchk**来检查(非修复)该表。请注意一般不建议使用多个MySQL服务器来并行访问相同的数据，除了使用MySQL Cluster时。

即使使用--skip-external-locking，仍然可以使用LOCK TABLES和UNLOCK TABLES。

7.5.2. 调节服务器参数

可以用这个命令得到**mysqld**服务器 默认缓存区的大小:

```
shell> mysqld --verbose --help
```

这个命令生成所有**mysqld**选项和可配置变量的列表。输出包括 默认值并且看上去象这样:

```
help                            TRUE
abort-slave-event-count         0
allow-suspicious-udfs           FALSE
auto-increment-increment        1
auto-increment-offset           1
```

```
automatic-sp-privileges          TRUE
basedir                          /home/jon/bin/mysql/
bdb                              FALSE
bind-address                     (No default value)
character-set-client-handshake   TRUE
character-set-server             latin1
character-sets-dir               /home/jon/bin/mysql/share/mysql/charsets/
chroot                           (No default value)
collation-server                 latin1_swedish_ci
completion-type                  0
concurrent-insert                1
console                          FALSE
datadir                          /home/jon/bin/mysql/var/
default-character-set            latin1
default-collation                latin1_swedish_ci
default-time-zone                (No default value)
disconnect-slave-event-count     0
enable-locking                   FALSE
enable-pstack                    FALSE
engine-condition-pushdown        FALSE
external-locking                 FALSE
gdb                              FALSE
large-pages                      FALSE
init-connect                     (No default value)
init-file                        (No default value)
init-slave                       (No default value)
innodb                           TRUE
innodb_checksums                 TRUE
innodb_data_home_dir             (No default value)
innodb_doublewrite               TRUE
innodb_fast_shutdown             1
innodb_file_per_table            FALSE
innodb_flush_log_at_trx_commit   1
innodb_flush_method              (No default value)
innodb_locks_unsafe_for_binlog   FALSE
innodb_log_arch_dir              (No default value)
innodb_log_group_home_dir        (No default value)
innodb_max_dirty_pages_pct       90
innodb_max_purge_lag             0
```

```
innodb_status_file                FALSE
innodb_table_locks                TRUE
innodb_support_xa                 TRUE
isam                              FALSE
language                          /home/jon/bin/mysql/share/mysql/english
local-infile                      TRUE
log                               /home/jon/bin/mysql/var/master1.log
log-bin                           /home/jon/bin/mysql/var/master1
log-bin-index                     (No default value)
log-bin-trust-routine-creators    FALSE
log-error                         /home/jon/bin/mysql/var/master1.err
log-isam                          myisam.log
log-queries-not-using-indexes     FALSE
log-short-format                  FALSE
log-slave-updates                 FALSE
log-slow-admin-statements         FALSE
log-slow-queries                  (No default value)
log-tc                            tc.log
log-tc-size                       24576
log-update                        (No default value)
log-warnings                      1
low-priority-updates              FALSE
master-connect-retry              60
master-host                       (No default value)
master-info-file                  master.info
master-password                   (No default value)
master-port                       3306
master-retry-count                86400
master-ssl                        FALSE
master-ssl-ca                     (No default value)
master-ssl-capath                 (No default value)
master-ssl-cert                   (No default value)
master-ssl-cipher                 (No default value)
master-ssl-key                    (No default value)
master-user                       test
max-binlog-dump-events            0
memlock                           FALSE
myisam-recover                    OFF
ndbcluster                        FALSE
```

```
ndb-connectstring                  (No default value)
ndb-mgmd-host                      (No default value)
ndb-nodeid                         0
ndb-autoincrement-prefetch-sz      32
ndb-distibution                    KEYHASH
ndb-force-send                     TRUE
ndb_force_send                     TRUE
ndb-use-exact-count                TRUE
ndb_use_exact_count                TRUE
ndb-shm                            FALSE
ndb-optimized-node-selection       TRUE
ndb-cache-check-time               0
ndb-index-stat-enable              TRUE
ndb-index-stat-cache-entries       32
ndb-index-stat-update-freq         20
new                                FALSE
old-alter-table                    FALSE
old-passwords                      FALSE
old-style-user-limits              FALSE
pid-file                           /home/jon/bin/mysql/var/hostname.pid1
port                               3306
relay-log                          (No default value)
relay-log-index                    (No default value)
relay-log-info-file                relay-log.info
replicate-same-server-id           FALSE
report-host                        (No default value)
report-password                    (No default value)
report-port                        3306
report-user                        (No default value)
rpl-recovery-rank                  0
safe-user-create                   FALSE
secure-auth                        FALSE
server-id                          1
show-slave-auth-info               FALSE
skip-grant-tables                  FALSE
skip-slave-start                   FALSE
slave-load-tmpdir                  /tmp/
socket                             /tmp/mysql.sock
sporadic-binlog-dump-fail          FALSE
```

```
sql-mode                          OFF
symbolic-links                    TRUE
tc-heuristic-recover              (No default value)
temp-pool                         TRUE
timed_mutexes                     FALSE
tmpdir                            (No default value)
use-symbolic-links                TRUE
verbose                           TRUE
warnings                          1
back_log                          50
binlog_cache_size                 32768
bulk_insert_buffer_size           8388608
connect_timeout                   5
date_format                       (No default value)
datetime_format                   (No default value)
default_week_format               0
delayed_insert_limit              100
delayed_insert_timeout            300
delayed_queue_size                1000
expire_logs_days                  0
flush_time                        0
ft_max_word_len                   84
ft_min_word_len                   4
ft_query_expansion_limit          20
ft_stopword_file                  (No default value)
group_concat_max_len              1024
innodb_additional_mem_pool_size   1048576
innodb_autoextend_increment       8
innodb_buffer_pool_awe_mem_mb     0
innodb_buffer_pool_size           8388608
innodb_concurrency_tickets        500
innodb_file_io_threads            4
innodb_force_recovery             0
innodb_lock_wait_timeout          50
innodb_log_buffer_size            1048576
innodb_log_file_size              5242880
innodb_log_files_in_group         2
innodb_mirrored_log_groups        1
innodb_open_files                 300
```

```
innodb_sync_spin_loops            20
innodb_thread_concurrency         20
innodb_commit_concurrency         0
innodb_thread_sleep_delay         10000
interactive_timeout               28800
join_buffer_size                  131072
key_buffer_size                   8388600
key_cache_age_threshold           300
key_cache_block_size              1024
key_cache_division_limit          100
long_query_time                   10
lower_case_table_names            0
max_allowed_packet                1048576
max_binlog_cache_size             4294967295
max_binlog_size                   1073741824
max_connect_errors                10
max_connections                   100
max_delayed_threads               20
max_error_count                   64
max_heap_table_size               16777216
max_join_size                     4294967295
max_length_for_sort_data          1024
max_relay_log_size                0
max_seeks_for_key                 4294967295
max_sort_length                   1024
max_tmp_tables                    32
max_user_connections              0
max_write_lock_count              4294967295
multi_range_count                 256
myisam_block_size                 1024
myisam_data_pointer_size          6
myisam_max_extra_sort_file_size   2147483648
myisam_max_sort_file_size         2147483647
myisam_repair_threads             1
myisam_sort_buffer_size           8388608
myisam_stats_method               nulls_unequal
net_buffer_length                 16384
net_read_timeout                  30
net_retry_count                   10
```

```
net_write_timeout                 60
open_files_limit                  0
optimizer_prune_level             1
optimizer_search_depth            62
preload_buffer_size               32768
query_alloc_block_size            8192
query_cache_limit                 1048576
query_cache_min_res_unit          4096
query_cache_size                  0
query_cache_type                  1
query_cache_wlock_invalidate      FALSE
query_prealloc_size               8192
range_alloc_block_size            2048
read_buffer_size                  131072
read_only                         FALSE
read_rnd_buffer_size              262144
div_precision_increment           4
record_buffer                     131072
relay_log_purge                   TRUE
relay_log_space_limit             0
slave_compressed_protocol         FALSE
slave_net_timeout                 3600
slave_transaction_retries         10
slow_launch_time                  2
sort_buffer_size                  2097144
sync-binlog                       0
sync-frm                          TRUE
sync-replication                  0
sync-replication-slave-id         0
sync-replication-timeout          10
table_cache                       64
table_lock_wait_timeout           50
thread_cache_size                 0
thread_concurrency                10
thread_stack                      196608
time_format                       (No default value)
tmp_table_size                    33554432
transaction_alloc_block_size      8192
transaction_prealloc_size         4096
```

```
updatable_views_with_limit        1
wait_timeout                      28800
```

如果有一个**mysqld**服务器正在运行，通过连接它并执行这个命令，可以看到实际上使用的变量的值：

```
mysql> SHOW VARIABLES;
```

还可以通过下面的语句看到运行服务器的统计和状态指标：

```
mysql>SHOW STATUS;
```

使用**mysqladmin**还可以获得系统变量和状态信息：

```
shell> mysqladmin variables
shell> mysqladmin extended-status
```

关于所有系统和状态变量的完全描述参见5.3.3节，“服务器系统变量”和5.3.4节，“服务器状态变量”。

MySQL使用完全可以升级的算法，因此通常运行时可以用很少的内存。然而，通常情况若给MySQL更多的内存性能会更好。

当调节MySQL服务器时，要配置的两个最重要的变量是key_buffer_size和table_cache。在试图更改其它变量前你应先确信已经适当地配置了这些变量。

下面的例子显示了部分典型的不同的运行时配置的变量值。

- 如果至少有256MB内存和许多表，想要在中等数量的客户时获得最大性能，应使用：

```
shell> mysqld_safe --key_buffer_size=64M --table_cache=256 \
          --sort_buffer_size=4M --read_buffer_size=1M &
```

- 如果只有128MB内存和少量表，但仍然要进行大量的排序，可以使用：

```
shell> mysqld_safe --key_buffer_size=16M --sort_buffer_size=1M
```

如果有许多并行连接，交换问题会发生，除非**mysqld**已经配置成为每个连接分配很少的内存。如果有足够的内存用于所有连接，**mysqld**会执行得更好。

- 对于少量内存和大量连接，应使用：

```
shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=100K \
          --read_buffer_size=100K &
```

或甚至为：

```
shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=16K \
          --table_cache=32 --read_buffer_size=8K \
          --net_buffer_length=1K &
```

如果正对远远大于可用内存的表执行GROUP BY或ORDER BY操作，应增加read_rnd_buffer_size的值以加速排序操作后面的行读取。

如果已经安装了MySQL，support-files目录包含一些不同的my.cnf示例文件：my-huge.cnf、my-大.cnf、my-medium.cnf和my-small.cnf。可以使用这些文件来优化系统。

请注意如果在命令行中为**mysqld**或**mysqld_safe**指定一个选项，它只在该次服务器调用中保持有效。要想每次服务器运行时使用该选项，将它放在选项文件中。

要想看参数更改的效果，应执行：

```
shell> mysqld --key_buffer_size=32M --verbose ---help
```

变量值列于输出的最后。确保--verbose和---help选项在最后。否则，在命令行中列于它们后面的选项的效果不会反映到输出中。

关于调节InnoDB存储引擎的信息，参见15.2.11节，“InnoDB性能调节提示”。

7.5.3. 控制查询优化器的性能

查询优化器的任务是发现执行SQL查询的最佳方案。因为“好”方案和“坏”方案之间的性能差别会巨大(也就是说，秒相对于小时或甚至天)，大多数查询优化器，包括MySQL的查询优化器，总或多或少地在所有可能的查询评估方案中搜索最佳方案。对于联接查询，MySQL优化器所调查的可能的方案数随查询中所引用的表的数目呈指数增长。对于小数量的表(典型小于7-10)，这不是一个问题。然而，当提交的查询更大时，查询优化所花的时间会很容易地成为服务器性能的主要瓶颈。

查询优化的一个更加灵活的方法是允许用户控制优化器详尽地搜索最佳查询评估方案。一般思想是优化器调查的方案越少，它编译一个查询所花费的时间越少。另一方面，因为优化器跳过了一些方案，它可能错过一个最佳方案。

优化器关于方案数量评估的行为可以通过两个系统变量来控制：

- optimizer_prune_level变量告诉优化器根据对每个表访问的行数的估计跳过某些方案。我们的试验显示该类“有根据的猜测”很少错过最佳方案，并且可以大大降低查询编辑次数。这就是为什么默认情况该选项为on(optimizer_prune_level=1)。然而，如果你认为优化器错过了一个更好的查询方案，则该选项可以关闭(optimizer_prune_level=0)，风险是查询编辑花费的时间更长。请注意即使使用该启发，优化器仍然可以探测呈指数数目的方案。

- optimizer_search_depth变量告诉优化器对于每个未完成的“未来的”方案，应查看多深，以评估是否应对它进一步扩大。optimizer_search_depth值较小会使查询编辑次数大大减小。例如，如果optimizer_search_depth接近于查询中表的数量，对12、13或更多表的查询很可能需要几小时甚至几天的时间来编译。同时，如果用optimizer_search_depth等于3或4编辑，对于同一个查询，编译器编译时间可以少于1分钟。如果不能确定合理的optimizer_search_depth值，该变量可以设置为0，告诉优化器自动确定该值。

7.5.4. 编译和链接怎样影响**MySQL**的速度

下列大多数测试是在Linux上并用**MySQL**基准进行的，但是它们能对其它操作系统和工作负载给出一些指示。

当你用-static链接时，可以得到最快的可执行文件。

在Linux上，最好用**pgcc**和-O3编译服务器。为了用这些选项编译“`sql_yacc.cc`”，需要大约200M内存，因为**gcc**或**pgcc**需要大量的内存使所有函数嵌入（inline）。在配置**MySQL**时，也应该设定CXX=gcc以避免包括libstdc++库(它不需要)。请注意对于某些版本的**pgcc**，生成的二进制只能运行在真Pentium处理器上，即使你使用编译器选项说明你想让最终的代码在所有x586-类处理器上工作(例如AMD)。

只通过使用一个较好的编译器或较好的编译器选项，在应用中能得到10-30%的加速。如果你自己编译SQL服务器，这特别重要！

当我们测试Cygnus CodeFusion或Fujitsu编译器时，二者均还没足够不出错来让MySQL启用优化进行编译。

标准MySQL二进制分发编译为支持所有字符集。当你自己编译MySQL时，应只包括将使用的字符集的支持。通过**configure**的--with-charset选项来控制。

这里是我们做过的一些测量表:

· 如果你使用**pgcc**并用-O6编译，**mysqld**服务器比用**gcc** 2.95.2快11%。

· 如果你动态地链接(没有-static)，在Linux中结果慢了13%。注意你仍能在客户应用程序中使用动态链接MySQL库。只有服务器对性能是关键的。

· 如果你用strip mysqld剥离**mysqld**二进制，生成的二进制可以快4%。

· 对于在同一主机上运行的客户与服务器之间的连接，如果你使用TCP/IP而非Unix套接字文件进行连接，结果慢7.5%。(在Unix中，如果你连接localhost主机，MySQL默认使用一个套接字文件）。

· 对于从客户到服务器的TCP/IP连接，从另一台主机连接一台远程服务器要比连接同一主机上的服务器慢8-11%，即使通过100Mb/s以太网进行连接。

· 当使用安全连接运行我们的基准测试时(所有数据用内部SSL支持进行加密)，性能比未加密连接慢55%。

· 如果你用--with-debug=full编译，大多数查询慢20%。部分查询时间会很长；例如，MySQL基准的运行要慢35%。如果你使用--with-debug(没有=full)，速度只下降15%。对于用--with-debug=full编译的**mysqld**版本，可以用--skip-safemalloc选项启动以便在运行时禁用内存检查。执行速度则接近用--with-debug配置的时候。

· 在Sun UltraSPARC-Iie上，用Forte 5.0编译的服务器比用**gcc** 3.2编译的要快4%。

· 在Sun UltraSPARC-Iie上，用Forte 5.0编译的32位模式服务器比64位模式服务器要快4%。

· 用**gcc** 2.95.2编译带-mcpu=v8 -Wa的UltraSPARC，使用-xarch=v8plusa选项性能会提高4%。

· 在Solaris 2.5.1上，在单个处理器上MIT-pthreads比带原生线程的Solaris慢8-12%。如果有更大的负载/cpus，差别应该更大。

· 在Linux-x86上使用**gcc**编译而不用帧指针(-fomit-frame-pointer or -fomit-frame-pointer -ffixed-ebp)，可以使**mysqld**快1-4%。

MySQL AB提供的Linux上的二进制MySQL分发一般用**pgcc**编译。我们必须返回到常规**gcc**，因为**pgcc**中有一个bug，使生成的二进制不能在AMD上运行。我们将继续使用**gcc**直到该bug被解决。同时，如果你有一个非AMD机，你可以用**pgcc**编译构建一个更快的二进制。标准MySQL Linux二进制是通过静态链接，以使它更快并且更加易于移植。

7.5.5. MySQL如何使用内存

下面的列表中列出了**mysqld**服务器使用内存的一些方法。在适用的地方，给出了内存相关的系统变量名:

· 键缓存(变量key_buffer_size)被所有线程共享；服务器使用的其它缓存则根据需要分配。参见7.5.2节，“调节服务器参数”。

· 每个连接使用具体线程的空间:

o 堆栈(默认64KB，变量thread_stack)

- 连接缓存区(变量net_buffer_length)
- 结果缓存区(变量net_buffer_length)

连接缓存区和结果缓存区可以根据需要动态扩充到max_allowed_packet。当某个查询运行时，也为当前查询字符串分配内存。

- 所有线程共享相同的基本内存。
- 只有压缩MyISAM表映射到内存。这是因为4GB的32位内存空间不足以容纳大多数大表。当64位地址空间的系统变得越来越普遍后，我们可以增加常规的内存映射支持。
- 对表进行顺序扫描的请求将分配一个缓存区(变量read_buffer_size)。
- 当按任意顺序读取行时(例如，按照排序顺序)，将分配一个随机读 缓存区(变量read_rnd_buffer_size)以避免硬盘搜索。
- 所有联合在一个令牌内完成，并且大多数联合甚至可以不用临时表即可以完成。大多数临时表是基于内存的(HEAP)表。具有大的记录长度的临时表 (所有列的长度的和)或包含BLOB列的表存储在硬盘上。

如果某个内部heap（堆积）表大小超过tmp_table_size，MySQL可以根据需要自动将内存中的heap表改为基于硬盘的MyISAM表。还可以通过设置**mysqld**的tmp_table_size选项来增加临时表的大小，或设置客户程序中的SQL选项SQL_BIG_TABLE。参见13.5.3节，“SET语法”。

- 进行排序的大多数请求将分配一个排序缓存区，并根据结果集的大小为两个临时文件分配零。参见A.4.4节，“MySQL将临时文件储存在哪里”。
- 几乎所有解析和计算在局部内存中完成。小项目不需要内存，因此避免了普通的慢内存分配和释放。只为不期望的大字符串分配内存；使用函数malloc()和free()来完成。
- 对于每个打开的MyISAM表，索引文件打开一次；数据文件为每个并行运行的线程打开一次。对于每个并行线程，将分配一个表结构、一个每个列的列结构和大小为3 * *N*的缓存区(其中*N*是最大行的长度，而不是计算BLOB列)。一个BLOB列需要5至8个字节加上BLOB数据的长度。MyISAM 存储引擎维护一个额外的行缓存区供内部应用。
- 对于每个具有BLOB列的表，将对缓存区进行动态扩大以读入大的BLOB 值。如果你扫描一个表，则分配一个与最大的BLOB值一样大的缓存区。
- 所有使用的表的句柄结构保存在高速缓存中并以FIFO管理。默认情况，高速缓存有64个入口。如果某个表同时被两个运行的线程使用，高速缓存则为该提供两个入口。参见7.4.9节，“MySQL如何打开和关闭表”。
- 当并行执行的线程结束时，FLUSH TABLE语句或**mysqladmin flush-table**命令可以立即关闭所有不使用的表并将所有使用中的表标记为已经关闭。这样可以有效释放大多数使用中的内存。FLUSH TABLE在关闭所有表之前不返回结果。

ps和其它系统状态程序可以报导mysqld使用很多内存。这可以是在不同的内存地址上的线程栈造成的。例如，Solaris版本的ps将栈间未用的内存算作已用的内存。你可以通过用swap -s检查可用交换区来验证它。我们用商业内存漏洞探查器测试了mysqld，因此应该有没有内存漏洞。

7.5.6. MySQL如何使用DNS

当新的客户连接**mysqld**时，**mysqld**创建一个新的线程来处理请求。该线程先检查是否主机名在主机名缓存中。如果不在，线程试图解析主机名：

- 如果操作系统支持线程安全gethostbyaddr_r ()和gethostbyname_r()调用，线程使用它们来执行主机名解析。

- 如果操作系统不支持线程安全调用，线程锁定一个互斥体并调用gethostbyaddr()和gethostbyname()。在这种情况下，在第1个线程解锁互斥体前，没有其它线程可以解析不在主机名缓存中的主机名。

你可以用--skip-name-resolve选项启动**mysqld**来禁用DNS主机名查找。然而，在这种情况下，你只可以使用MySQL中的授权表中的IP号。

如果你有一个很慢的DNS和许多主机，你可以通过用--skip-name-resolve禁用DNS查找或增加HOST_CACHE_SIZE定义(默认值：128)并重新编译**mysqld**来提高性能。

你可以用--skip-host-cache选项启动服务器来禁用主机名缓存。要想清除主机名缓存，执行FLUSH HOSTS语句或执行**mysqladmin flush-hosts**命令。

如果你想要完全禁止TCP/IP连接，用--skip-networking选项启动**mysqld**。

7.6. 磁盘事宜

7.6.1. 使用符号链接

- 磁盘搜索是巨大的性能瓶颈。当数据量变得非常大以致于缓存性能变得不可能有效时，该问题变得更加明显。对于大数据库，其中你或多或少地随机访问数据，你可以确信对读取操作需要至少一次硬盘搜索，写操作需要多次硬盘搜索。要想使该问题最小化，应使用搜索次数较少的磁盘。

- 通过链接文件到不同的磁盘或对硬盘分段来增加可用磁盘锭数量（因此降低搜索成本）：

  - 使用符号链接

  这说明，对于MyISAM表，你符号链接索引文件和/或数据文件，从它们数据目录内的通常位置到另一个硬盘（也可以被条纹化）。这将使搜索和读次数达到最好，假定硬盘不再为其它目的使用。参见7.6.1节，“使用符号链接”。

  - 分条

  分条意味着你有许多磁盘，将第1个块放到第1个硬盘，第2个块放到第2个磁盘，并且第*N*块在（*N* mod number_of_disks）磁盘上等等。这意味着如果正常数据大小小于分条大小（或完全匹配），能够得到最佳性能。分条完全取决于操作系统和分条大小，因此用不同的条纹大小对应用程序进行基准测试。参见7.1.5节，“使用自己的基准”。

  分条的不同速度完全依赖于参数。依赖于怎样设置条纹参数和硬盘数量，可以根据不同数量级别得到不同的标准。你必须进行选择以便优化随机或顺序存取。

- 为了高可靠性你可能想使用RAID 0+1（条纹加镜像），但在这种情况下，需要2*N块磁盘来保持N个磁盘的数据。如果你肯为它花钱，这可能是最好的选项。然而，你可能还必须投资一部分资金到卷管理软件中以便有效地管理它。

- 一个较好的选择是根据数据类型的重要性程度改变RAID级别。例如，保存可以在RAID 0硬盘上重新生成的不太重要的数据，但保存 真正重要数据（例如主机信息和日志）到RAID 0+1或RAID N硬盘。如果你有许多写操作，RAID N可能会存在问题，因为需要时间来更新校验位。

- 在Linux上，通过配置磁盘接口时使用hdparm，你可以获得更多的性能。（在一般负载下达到100%并不困难。）下面的hdparm选项 应该非常适用于MySQL，并且可能适用于许多其它应用程序：

```
hdparm -m 16 -d 1
```

请注意，当使用该命令时，性能和可靠性依赖于硬件，因此我们强烈建议，使用hdparm后，完全测试你的系统。为获取更多详细信息，请查阅hdparm手册帮助页。如果hdparm使用的不好，会导致文件系统破坏，因此试验之前请做好每个备份!

- 也可以设置数据库使用的文件系统参数:

如果不需要知道文件最后一次访问的时间（这对数据库服务器并没有实际的用途），可以用-o noatime选项安装文件系统。这将跳过对文件系统中的节点的最后一次访问时间的更新，从而能够避免一些硬盘搜索。

在许多操作系统中，用-o async选项安装，可以将文件系统设置为异步更新。如果电脑相当稳定，这应该给予你更高的性能而不需要牺牲太多的可靠性。（该标志是Linux中默认开启。）

7.6.1. 使用符号链接

7.6.1.1. 在Unix上使用针对数据库的符号链接
7.6.1.2. 在Unix平台上使用表的符号链接
7.6.1.3. 在Windows平台上使用关于数据库的符号链接

你可以将表和数据库从数据库目录移动到其它的位置并且用指向新位置的符号链接进行替换。你可能需要这样作，例如，移动数据库到一个有更多空闲空间的文件系统中或通过将表分散到不同的磁盘上以增加系统的访问速度。

推荐的方法值需要将数据库通过符号链接指到不同的磁盘。符号链接表仅作为是最后的办法。

7.6.1.1. 在Unix上使用针对数据库的符号链接

在Unix中，符号链接一个数据库的方法是，首先在一些有空闲空间的硬盘上创建一个目录，然后从MySQL数据目录中创建它的一个符号链接。

```
shell> mkdir /dr1/databases/test

shell> ln -s /dr1/databases/test /path/to/datadir
```

MySQL不支持链接一个目录到多个数据库。用一个符号链接替换一个数据库目录，只要你不在数据库之间制作符号链接。假定你有一个数据库db1在MySQL数据目录下，然后生成一个符号链接db2指向 db1:

```
shell> cd /path/to/datadir

shell> ln -s db1 db2
```

对于db1中的任何表tbl_a，在db2也出现并且还是表tbl_a。如果一个客户更新db1.tbl_a并且另一个客户更新db2.tbl_a，可能会出现问题。

然而，如果你真的需要这样做，可以通过改变源文件mysys/my_symlink.c来实现，在该文件中你应查找下面的语句:

```
if (!(MyFlags & MY_RESOLVE_LINK) ||

   (!lstat(filename,&stat_buff) && S_ISLNK(stat_buff.st_mode)))
```

将该语句修改为:

```
if (1)
```

请注意，对于所有的Windows服务器，默认启用符号链接支持。

7.6.1.2. 在Unix平台上使用表的符号链接

你不应在没有完全可操作的realpath()调用的系统中对表进行符号链接。（Linux和Solaris支持realpath()）。可以通过发出一个SHOW VARIABLES LIKE 'have_symlink'语句，检查系统是否支持符号链接。

只有MyISAM表完全支持符号链接。对于其它表类型，如果试图在操作系统中的文件上用前面的任何语句使用符号链接，可能会出现奇怪的问题。

对于MyISAM表的符号链接的处理如下：

- 在数据目录指，一定会有表定义文件、数据文件和索引文件。数据文件和索引文件可以移到别处和在数据目录中符号链接替代。表定义文件不能进行符号链接替换。

- 可以分别通过符号链接将数据文件和索引文件指到不同的目录。

- 如果**mysqld**没有运行，符号链接可以从服务器命令行使用ln -s手动完成。同样，通过使用DATA DIRECTORY和INDEX DIRECTORY选项创建表，你可以指示运行的MySQL服务器执行符号链接。参见13.1.5节，“CREATE TABLE语法”。

- **myisamchk**不用数据文件或索引文件替换符号链接。它直接工作在符号链接指向的文件。任何临时文件创建在数据文件或索引文件所处的目录中。

- 注释：当你删掉一个表时，如果该表使用了符号链接，符号链接和该符号链接指向的文件都被删除掉。这就是你不应以系统root用户运行**mysqld**或允许系统用户对MySQL数据库目录有写访问权限的原因。

- 如果你用ALTER TABLE ... RENAME重命名一个表并且不将表移到另一个数据库，数据库目录中的符号链接被重新命名为一个新名字并且数据文件和索引文件也相应地重新命名。

- 如果你用ALTER TABLE ... RENAME移动一个表到另一个数据库，表移动到另一个数据库目录。旧的符号链接和其所指向的文件被删除。换句话说，新表不再被链接。

- 如果不使用符号链接，你应对**mysqld**使用--skip-symbolic-links选项以确保没有人能够使用**mysqld**来删除或重新命名数据目录之外的文件。

表符号链接还不支持以下操作：

- ALTER TABLE忽略DATA DIRECTORY 和INDEX DIRECTORY表选项。

- BACKUP TABLE 和RESTORE TABLE不考虑符号链接。

- .frm文件必须绝不能是一个符号链接（如前面所述，只有数据和索引文件可以是符号链接）。如果试图这样做（例如，生成符号链接）会产生不正确的结果。假定你在MySQL数据目录下有一个数据库db1，该数据库有一个表tbl1，并且在db1目录中你制作了一个符号链接tbl2指向tbl1：

```
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
```

如果一个线程读取db1.tbl1同时另一个线程更新 db1.tbl2会发生问题：

- 查询缓存将变为"傻瓜" (它没有办法知道tbl1是否被更新，因此它返回过时的结果)。
- tbl2上的ALTER语句也会失败。

7.6.1.3. 在**Windows**平台上使用关于数据库的符号链接

Windows的**mysqld-max**和mysql-max-nt服务器使用-DUSE_SYMDIR选项编译成。允许你放置数据库目录到一个不同的硬盘，通过设置一个符号链接指向它。这类似于Unix中的符号链接，尽管设置链接的过程不同。

符号链接默认为启用。如果你不需要，使用skip-symbolic-links选项来禁用它：

```
[mysqld]
skip-symbolic-links
```

在Windows中，通过在数据目录中创建一个文件，该文件包含目标目录的路径，你可以为MySQL数据库创建一个符号链接。该文件应该被命名为db_name.sym，其中*db_name*是数据库名。

假定MySQ数据目录是C:\mysql\data并且你想要数据库foo放置在D:\data\foo。设置一个符号链接如下所示：

1. 确保D:\data\foo目录存在，如果必要创建它。如果你在数据目录内有一个命名为foo的数据库目录，你应移动它到D:\data目录。否则，符号链接无效。为避免出现问题，当你移动数据库目录时服务器不应该运行。
2. 创建一个文本文件C:\mysql\data\foo.sym，该本文文件包含路径名D:\data\foo\。

此后，数据库foo创建的所有表将创建在D:\data\foo文件中。请注意，如果在*MySQL*数据目录中存在同名的数据库目录，不能使用符号链接。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第8章：客户端和实用工具程序

目录

8.1. 客户端脚本和实用工具概述
8.2. myisampack：生成压缩、只读MyISAM表
8.3. mysql：MySQL命令行工具

- 8.3.1. 选项
- 8.3.2. **mysql**命令
- 8.3.3. 怎样从文本文件执行SQL语句
- 8.3.4. **mysql**技巧

8.4. mysqlaccess：用于检查访问权限的客户端
8.5. mysqladmin：用于管理MySQL服务器的客户端
8.6. mysqlbinlog：用于处理二进制日志文件的实用工具
8.7. mysqlcheck：表维护和维修程序
8.8. mysqldump：数据库备份程序
8.9. mysqlhotcopy：数据库备份程序
8.10. mysqlimport：数据导入程序
8.11. mysqlshow－显示数据库、表和列信息
8.12. myisamlog：显示MyISAM日志文件内容
8.13. perror：解释错误代码
8.14. replace：字符串替换实用工具
8.15. mysql_zap：杀死符合某一模式的进程

有许多不同的MySQL客户端程序可以连接服务器以访问数据库或执行管理任务。也可以使用其它工具。这些程序不与服务器进行通讯但可以执行MySQL相关的操作。

本章简述了这些程序然后详细描述了每个程序。描述了如何调用这些程序和它们理解的选项。调用程序和指定程序选项的总信息参见第4章：*MySQL*程序概述。

8.1. 客户端脚本和实用工具概述

下面简单列出了MySQL客户端程序和实用工具：

· myisampack

压缩MyISAM表以产生更小的只读表的一个工具。参见8.2节，“myisampack：生成压缩、只读MyISAM表”。

· mysql

交互式输入SQL语句或从文件以批处理模式执行它们的命令行工具。参见8.3节，“mysql：MySQL命令行工具”。

· mysqlaccess

检查访问主机名、用户名和数据库组合的权限的脚本。

· mysqladmin

执行管理操作的客户程序，例如创建或删除数据库，重载授权表，将表刷新到硬盘上，以及重新打开日志文件。**mysqladmin**还可以用来检索版本、进程，以及服务器的状态信息。参见8.5节，“mysqladmin：用于管理MySQL服务器的客户端”。

- mysqlbinlog

从二进制日志读取语句的工具。在二进制日志文件中包含的执行过的语句的日志可用来帮助从崩溃中恢复。参见8.6节，“mysqlbinlog：用于处理二进制日志文件的实用工具”。

- mysqlcheck

检查、修复、分析以及优化表的表维护客户程序。参见8.7节，“mysqlcheck：表维护和维修程序”。

- mysqldump

将MySQL数据库转储到一个文件（例如SQL语句或tab分隔符文本文件）的客户程序。增强版免费软件首先由Igor Romanenko提供。参见8.8节，“mysqldump：数据库备份程序”。

- mysqlhotcopy

当服务器在运行时，快速备份MyISAM或ISAM表的工具。参见8.9节，“mysqlhotcopy：数据库备份程序”。

- mysql import

使用LOAD DATA INFILE将文本文件导入相关表的客户程序。参见8.10节，“mysqlimport：数据导入程序”。

- mysqlshow

显示数据库、表、列以及索引相关信息的客户程序。参见8.11节，“mysqlshow：显示数据库、表和列信息”。

- perror

显示系统或MySQL错误代码含义的工具。参见8.13节，“perror：解释错误代码”。

- replace

更改文件中或标准输入中的字符串的实用工具。参见8.14节，“replace：字符串替换实用工具”。

MySQL AB还提供了大量GUI工具用于管理和MySQL服务器的其它工作。相关基本信息参见第4章：*MySQL*程序概述。

每个MySQL程序有许多不同的选项。但每个MySQL程序均提供一个---help选项，可以用来全面描述程序不同的选项。例如，可以试试**mysql---help**。

使用mysqlclient库同服务器进行通讯的MySQL客户使用下面的环境变量：

MYSQL_UNIX_PORT	默认Unix套接字文件；用于连接localhost
MYSQL_TCP_PORT	默认端口号；用于TCP/IP连接
MYSQL_PWD	默认密码
MYSQL_DEBUG	调试过程中的调试跟踪选项
TMPDIR	创建临时表和文件的目录

使用MYSQL_PWD不安全。参见5.8.6节，“使你的密码安全”。

可以在选项文件中或在命令行中指定选项来替换所有标准程序的默认选项值或指定的环境变量的值。参见4.3节，“指定程序选项”。

8.2. myisampack：生成压缩、只读MyISAM表

myisampack工具可以压缩MyISAM表.MYI**sampack**分别压缩表中的每一列。通常，**myisampack**可以将数据文件压缩到40%-70%。

当以后使用表时，解压缩列需要的信息被读入内存。当访问具体的记录时性能会更好，因为你只需要解压缩一个记录。

MySQL使用mmap()对压缩的表进行内存映射。如果mmap()不工作，MySQL返回到普通读/写文件操作。

请注意:

- 如果用--skip-external-locking选项调用**mysqld**服务器，如果在压缩过程中表可能被更新，调用**myisampack**不是一个好注意。

- 表压缩后，它变为只读。这是故意的(例如当访问CD上的压缩的表时)。允许写入到压缩的表位于我们的TODO列表中，但优先级较低。

- **myisampack**可以压缩BLOB或TEXT列。旧版本ISAM表的**pack_isam**程序不可以。

调用**myisampack**的方法:

```
shell> myisampack [options] filename ...
```

文件名应为索引(.MYI)文件的文件名。如果不在数据库目录，应指定文件的路径名。允许忽略.MYI扩展名。

myisampack支持下面的选项:

- --help，-?

显示帮助消息并退出。

- --backup，-b

使用*tbl_name*.OLD名备份表数据文件。

- ---debug[=*debug_options*]，-# [*debug_options*]

写调试日志。*debug_options*字符串通常为'd:t:o,*file_name*'。

- --force，-f

产生一个压缩的表，即使它比原始表大，或如果以前调用**myisampack**的中间文件存在。(**myisampack**压缩表时在数据库目录中创建一个名为*tbl_name*.TMD的中间文件。如果杀掉**myisampack**，.TMD文件会被删除）。通常情况，如果**myisampack**发现*tbl_name*.TMD存在则退出并提示错误。用--force，**myisampack**则一定压缩表。

- -join=*big_tbl_name*，-j *big_tbl_name*

将命令行中的所有表联接为一个表*big_tbl_name*。将要连接的所有表必须有相等的结构(相同的列名和类型，相同的索引等等)。

- --pack length=*len*，-p *len*

指定记录长度存储大小，以字节计。值应为1、2或者3。**myisampack**保存所有长度指针为1、2或者3字节的行。在大多数正常情况下，**myisampack**在开始压缩文件前可以确定准确的长度值，但在压缩过程中它可以提示它可能已经使用了一个短的长度。在这种情况下，**myisampack**输出一条提示，下次你压缩同一文件时，你可以使用更短的记录长度。

- --silent，-s

沉默模式。只有发生错误时才写输出。

- --test，-t

没有实际地压缩表，只是测试压缩。

- --tmpdir=*path*，-T *path*

使用**myisamchk**创建临时文件的目录。

- --verbose，-v

冗长模式。写压缩操作过程相关信息和其结果。

- --version，-V

显示版本信息并退出。

- --wait，-w

如果表正使用则等待并重试。如果用--skip-external-locking选项调用了**mysqld**服务器，如果在压缩过程中表可能被更新，调用**myisampack**不是一个好注意。

下面的顺序命令说明了典型的表压缩会话:

```
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%
Remember to run myisamchk -rq on compressed tables

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
Datafile pointer (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
```

myisampack显示下面的各种信息:

- normal

不需要进行额外压缩的列的数量。

· empty-space

只包含空格的列的数量；占一个比特。

· empty-zero

只包含二进制零的列的数量；占一个比特。

· empty-fill

不占该类全字节范围的整数列的数量；这些列被改为较小的类型。例如，如果所有值的范围为从-128到127，BIGINT列(8个字节)可以保存为TINYINT列(1个字节)。

· pre-space

用引导空格保存的十进制列的数量。在这种情况下，每个值包含一个引导空格的数量计数。

· end-space

有大量结尾空格的列的数量。在这种情况下，每个值包含一个结尾空格的数量计数。

· table-lookup

该列只有少量的不同的值，在进行哈夫曼压缩前被转换为一个ENUM。

· zero

所有值为零的列的数量。

· Original trees

哈夫曼树的最初数量。

· After join

联接树以节省一些头空间之后留下的哈夫曼树的数量。

表被压缩后，**myisamchk -dvv**为每列输出详细信息：

· Type

列的类型。该值可以包含下面的任何描述符：

o constant

所有行具有相同的值。

o no endspace

不保存结尾空格。

o no endspace，not_always

不保存结尾空格并且对于所有的值不压缩结尾空格。

o no endspace，no empty

不保存结尾空格。不保存空值。

o table-lookup

列被转换为一个ENUM。

o zerofill(*n*)

值中最有意义的*n*字节总为0，并且不保存。

o no zeros

不保存零。

o always zeros

用一个位保存零值。

· Huff tree

列相关的哈夫曼树的数量。

· Bits

哈夫曼树使用的位数。

运行**myisampack**后，必须运行**myisamchk**以重新创建索引。此时，你也可以排序索引块并创建MySQL优化器需要的统计信息以更有效地工作：

```
shell> myisamchk -rq --sort-index --analyze tbl_name.MYI
```

将压缩的表安装到MySQL数据库目录中后，应执行**mysqladmin flush-tables**以强制**mysqld**使用新的表。

要想解压缩一个压缩的表，使用**myisamchk**或**isamchk**的--unpack选项。

8.3. mysql：MySQL命令行工具

8.3.1. 选项
8.3.2. **mysql**命令
8.3.3. 怎样从文本文件执行SQL语句
8.3.4. **mysql**技巧

mysql是一个简单的SQL外壳(有GNU readline功能)。它支持交互式和非交互式使用。当交互使用时，查询结果采用ASCII表格式。当采用非交互式(例如，用作过滤器)模式时，结果为tab分割符格式。可以使用命令行选项更改输出格式。

如果由于结果较大而内存不足遇到问题，使用--quick选项。这样可以强制**mysql**从服务器每次一行搜索结果，而不是检索整个结果集并在显示之前不得不将它保存到内存中。使用mysql_use_result()而不是mysql_store_result()来搜索结果集。

使用**mysql**很简单。从命令解释符提示来调用它：

```
shell> mysql db_name
```

或：

```
shell> mysql --user=user_name --password=your_password db_name
```

这样输入一个SQL语句，用'；'、\g或者\G结尾并按回车键。

你可以这样运行一个脚本：

```
shell> mysql db_name < script.sql > output.tab
```

8.3.1. 选项

mysql支持下面的选项：

- ---help，-?

显示帮助消息并退出。

- --batch，-B

打印结果，使用tab作为列间隔符，每个行占用新的一行。使用该选项，则**mysql**不使用历史文件。

- --character-sets -dir=*path*

字符集的安装目录。参见5.10.1节，“数据和排序用字符集”。

- --compress，-C

压缩在客户端和服务器之间发送的所有信息（如果二者均支持压缩）。

- ---database=*db_name*，-D *db_name*

要使用的数据库。主要在选项文件中有用。

- ---debug[=*debug_options*]，-# [*debug_options*]

写调试日志。*debug_options*字符串通常为'd:t:o,*file_name*'。 默认为'd:t:o,/tmp/mysql.trace'。

- ---debug-info，-T

当程序退出时输出部分调试信息。

- --default-character-set=*charset*

使用*charset*as作为默认字符集。参见5.10.1节，“数据和排序用字符集”。

- --execute=*statement*, -e *statement*

执行语句并退出。默认输出格式与用--batch产生的相同。4.3.1节，“在命令行上使用选项”中提供了一些例子。

- --force，-f

即使出现一个SQL错误仍继续。

- --host=*host_name*，-h *host_name*

连接给定主机上的MySQL服务器。

- --html，-H

产生HTML输出。

- --ignore-space，-i

忽视函数名后面的空格。其结果描述参见5.3.2节，“SQL服务器模式”中的IGNORE_SPACE的讨论。

- --local-infile[={0|1}]

为LOAD DATA INFILE启用或禁用LOCAL功能。没有值，该选项启用LOCAL。还可以采用--local-infile=0或--local-infile=1以显式禁用或启用LOCAL。如果服务器不支持，启用LOCAL不会生效。

- --named-commands，-G

命名的命令被启用。允许长格式命令和短格式*命令。例如，quit和\q均被识别。

- --no-auto-rehash，-A

不自动重新进行哈希运算。该选项使**mysql**启动得更快，但果你想要完成表和列名，你必须发出rehash命令。

- --no-beep，-b

当发生错误时不要保持。

- --no-named-commands，-g

命名的命令被禁用。只使用*形式，或者只使用行开头的命名用分号(‘；’)结束的的命令。对于MySQL 3.23.22，默认情况**mysql**启动时启用该选项。然而，即使使用该选项，长格式命令仍然从第1行工作。

- --no-pager

不使用分页器来显示查询输出。在8.3.2节，“**mysql**命令”中详细讨论了输出分页。

- --no-tee

不将输出复制到文件中。在8.3.2节，“**mysql**命令”中详细讨论了Tee文件。

- --one--database，-O

忽视除了为命令行中命名的默认数据库的语句。可以帮助跳过对二进制日志中的其它数据库的更新。

- --pager[=*command*]

使用给出的命令来分页查询输出。如果该命令被删除，默认分页器为PAGER环境变量的值。合法pagers是**less**、**more**、**cat [>filename]**等等。该选项只在Unix中工作。不能以批处理模式工作。在8.3.2节，“**mysql**命令”中详细讨论了输出分页。

- --password[=*password*]，-p[*password*]

当连接服务器时使用的密码。如果使用短选项形式(-p)，选项和 密码之间不能有空格。如果在命令行中--password或-p选项后面没有 密码值，则提示输入一个密码。在SysV-based UNIX系统中应省略密码，因为密码可以显示在ps的输出中。

- --port=*port_num*，-P *port_num*

用于连接的TCP/IP端口号。

- --prompt=*format_str*

将提示设置为指定的格式。默认为mysql>。在8.3.2节，“**mysql**命令”中描述了提示中可以包含的具体顺序。

- --protocol={TCP | SOCKET | PIPE | MEMORY}

使用的连接协议。

- --quick，-q

不缓存每个查询的结果，按照接收顺序打印每一行。如果输出被挂起，服务器会慢下来。使用该选项，**mysql**不使用历史文件。

- --raw，-r

写列的值而不转义转换。通常结合--batch选项使用。

- --reconnect

如果与服务器之间的连接断开，自动尝试重新连接。每次连接断开后则尝试一次重新连接。要想禁止重新连接，使用--skip-reconnect。

- --safe-updates，--i-am-a-dummy，-U

只允许那些使用键值指定行生效的UPDATE和DELETE语句。如果已经在选项文件中设置了该选项，可以用命令行中的--safe-updates覆盖它。关于该选项的详细信息参见8.3.4节，“**mysql**技巧”。

- --secure-auth

不向旧(pre-4.1.1)格式的服务器发送密码。这样可以防止不使用新密码格式的服务器的连接。

- --show-warnings

如果每个语句后有警告则显示。该选项适用于交互式和批处理模式。

- --sigint-ignore

忽视SIGINT符号(一般为Control-C的结果)。

- --silent，-s

沉默模式。产生少的输出。可以多次使用该选项以产生更少的输出。

- --skip-column-names，-N

在结果中不写列名。

- --skip-line-numbers，-L

在错误信息中不写行号。当你想要比较包括错误消息的结果文件时有用。

- --socket=*path*，-S *path*

用于连接的套接字文件。

- --tables，-t

用表格式显示输出。这是交互式应用的默认设置，但可用来以批处理模式产生表输出。

· --tee=*file_name*

将输出拷贝添加到给定的文件中。该选项在批处理模式不工作。在8.3.2节，“**mysql**命令”中详细讨论了Tee文件。

· --unbuffered，-n

每次查询后刷新缓存区。

· --user=*user_name*，-u *user_name*

当连接服务器时MySQL使用的用户名。

· --verbose，-v

冗长模式。产生更多的输出。可以多次使用该选项以产生更多的输出。(例如，-v -v -v甚至可以在批处理模式产生表输出格式）。

· --version，-V

显示版本信息并退出。

· --vertical，-E

垂直输出查询输出的行。没有该选项，可以用\G结尾来指定单个语句的垂直输出。

· --wait，-w

如果不能建立连接，等待并重试而不是放弃。

· --xml，-X

产生XML输出。

你还可以使用--*var_name*=*value*选项设置下面的变量：

· connect_timeout

连接超时前的秒数。(默认值是0）。

· max_allowed_packet

从服务器发送或接收的最大包长度。(默认值是16MB）。

· max_join_size

当使用--safe-updates时联接中的行的自动限制。(默认值是1,000,000）。

· net_buffer_length

TCP/IP和套接字通信缓冲区大小。(默认值是16KB）。

· select_limit

当使用--safe-updates时SELECT语句的自动限制。(默认值是1,000）。

也可以使用--set-variable=*var_name*=*value* or -O *var_name*=*value*语法来设置变量。不赞成使用该语法。

在Unix中，**mysql**客户程序向历史文件中写入已执行语句的一条记录。默认情况，历史文件名为.mysql_history并在根目录中重建。要想指定不同的文件，应设置MYSQL_HISTFILE环境变量的值。

如果不想要历史文件，首先删除.mysql_history（如果有），然后使用下面的任何一种方法：

- 将MYSQL_HISTFILE变量设到/dev/null。要想在每次登录时让该设置生效，将该设置放入外壳的一个启动文件中。

- 创建.mysql_history，作为一个符号链接指向/dev/null：

```
shell> ln -s /dev/null $HOME/.mysql_history
```

只需要执行一次。

8.3.2. mysql命令

mysql将发出的SQL语句发送到待执行的服务器。还有一系列命令**mysql**可以自己解释。要查看这些命令，在mysql>提示下输入help或\h：

```
mysql> help

List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?         (\?) Synonym for `help'.
clear     (\c) Clear command.
connect   (\r) Reconnect to the server. Optional arguments are db and host.
delimiter (\d) Set statement delimiter. NOTE: Takes the rest of the line as new 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.
help      (\h) Display this help.
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 a 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.
warnings  (\W) Show warnings after every statement.
nowarning (\w) Don't show warnings after every statement.
```

每个命令有长形式和短形式。长形式对大小写不敏感；短形式敏感。长形式后面可以加一个分号结束符，但短形式不可以。

在delimiter命令中，应避免使用反斜线('\')，因为这是MySQL的转义符。

Edit、**nopager**、**pager**和**system**命令只在Unix中工作。

status命令提供连接和使用的服务器相关的部分信息。如果用--safe-updates模式运行，status也打印影响查询的MySQL变量的值。

要想记录查询及其输出，应使用**tee**命令。屏幕上显示的所有数据被追加到给定的文件后面。这对于调试也很有用。可以用--tee选项在命令行中启用该特性，或者用**tee**命令交互式启用。可以用**notee**命令交互式禁用**tee**文件。再次执行**tee**可以重新启用日志。前面的文件使用时不带任何参数。请注意执行完每个语句后**tee**可以将查询结果刷新到一个文件中，并且在**mysql**打印下一个提示前刷新。

用--pager选项，可以用交互式模式使用Unix程序，例如**less**、**more**或者其它类似的程序，来浏览或搜索查询结果。如果未指定该选项的值，**mysql**检查PAGER环境变量的值并将PAGER设为该值。可以用**pager**命令交互式启用输出分页，并用**nopager**禁用。该命令采用可选参量；如果给出，分页程序设置为该值。如果没有给出参量，分页器被设置为在命令行中设置的分页器的值，如果未指定分页器，则设置为stdout。

输出分页只适合Unix，因为它使用popen()函数，该函数在Windows中不存在。在Windows中，可以使用**tee**选项来保存查询输出，尽管在某些情况下浏览输出时不如**pager**方便。

关于**pager**命令的一些技巧：

- 可以使用它写入一个文件，将结果只输出到该文件中：

```
mysql> pager cat > /tmp/log.txt
```

也可以为将用作分页器的程序传递选项：

```
mysql> pager less -n -i -S
```

- 注意前面例子中的-S选项。它可以帮助浏览广范围的查询结果。有时太广的结果很难在屏幕上读出来。**less**的-S选项可以使结果更易于读，因为可以用左、右箭头水平滚动它。还可以在**less**中交互式使用-S，以关闭或打开水平浏览模式。详细信息请阅读手册中的**less**页：

```
shell> man less
```

- 可以指定很复杂的pager命令来处理查询输出：

```
mysql> PAGER cat | tee /dr1/tmp/res.txt \
                 | tee /dr2/tmp/res2.txt | less -n -i -S
```

在该例子中，该命令将查询结果发送到位于/dr1和/dr2上安装的两个不同的文件系统中的两个不同目录中的两个文件中，但仍然可以通过**less**将结果显示在屏幕上。

还可以结合使用**tee**和**pager**函数。启用一个**tee**文件并将**pager**设置为**less**，能够使用**less** 程序浏览结果，并且仍然可以同时将内容添加到一个文件中。结合**pager**命令使用的Unix **tee**和 **mysql**嵌入式**tee**命令的差别是即使没有可用的Unix **tee**，嵌入式**tee**仍然可以工作。嵌入式**tee**还可以记录在屏幕上输出的内容，而结合**pager**命令使用的Unix **tee**不能记录那么多的内容。并且，可以从MySQL中交互式打开或关闭**tee**文件日志。当你想要将部分查询记录到一个文件中时很有用，但其它不适合。

默认mysql>提示符可以重新配置。定义提示符的字符串可以包含下面的特殊序列：

选项	描述
\v	服务器版本
\d	当前的数据库
\h	服务器主机
\p	当前的TCP/IP端口或套接字文件
\u	你的用户名
\U	你的全*user_name*@*host_name*账户名
\\	'\'反斜线字符
\n	新行字符
\t	Tab字符
\	空格(反斜线后面的空格)
_	空格
\R	当前的时间，24-小时军用时间(0-23)

\r	当前的时间，标准12-小时(1-12)
\m	当前时间的分钟
\y	当前的年，两位
\Y	当前的年，四位
\D	当前的日期
\s	当前时间的秒
\w	当前周的天，3字符格式(Mon，Tue，...)
\P	am/pm
\o	当前的月，数字格式
\O	当前的月，3字符格式(Jan，Feb，...)
\c	随发出的每个语句递增的计数
\S	分号
\'	单引号
\"	双引号

'\'后面跟随的其它字母则变为该字母。

如果不用任何参量指定提示命令，**mysql**将提示重新设置位默认mysql>。

可以用几种方式设置提示：

- 使用环境变量

可以用MYSQL_PS1环境变量来设置提示字符串。例如：

```
shell> export MYSQL_PS1="(\u@\h) [\d]> "
```

- 使用选项文件

可以在MySQL选项文件中的[mysql]组设置提示，例如根目录中的/etc/my.cnf或.my.cnf文件。例如：

```
[mysql]
prompt=(\\u@\\h) [\\d]>\\_
```

在该例子中，请注意反斜线是双线。如果使用选项文件中的prompt选项来设置提示，当使用特殊提示选项时，建议使用双反斜线。在允许的提示选项和选项文件中可识别的特殊转义序列中有部分重叠。(这些序列列于4.3.2节，“使用选项文件”）。如果使用单反斜线，会遇到问题。例如，\s被解释为空格而不是当前的秒值。下面的例子显示了如何在选项文件中定义提示以包括当前的时间，格式为HH:MM:SS>：

```
[mysql]
prompt="\\r:\\m:\\s> "
```

- 使用命令行选项

可以在**mysql**的命令行中设置--prompt选项。例如：

```
shell> mysql --prompt="(\u@\h) [\d]> "
(user@host) [database]>
```

- 交互式

你可以使用prompt(或\R)命令交互地更改提示。例如：

```
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>
```

8.3.3. 怎样从文本文件执行SQL语句

mysql客户程序一般交互使用：

```
shell> mysql db_name
```

还可以将SQL语句放到一个文件中然后告诉**mysql**从该文件读取输入。要想实现，创建一个文本文件text_file，并包含你想要执行的语句。然后按如下所示调用**mysql**：

```
shell> mysql db_name < text_file
```

还可以用一个USE *db_name*语句启动文本文件。在这种情况下，不需要在命令行中指定数据库名：

```
shell> mysql < text_file
```

如果正运行**mysql**，可以使用source或\.命令执行SQL脚本文件：

```
mysql> source filename

mysql> \. filename
```

有时想要使用脚本来向用户显示进度信息；为此可以插入下述行：

```
SELECT '<info_to_display>' AS ' ';
```

将输出<info_to_display>。

关于批处理模式的详细信息，参见3.5节，“在批处理模式下使用**mysql**”。

8.3.4. mysql技巧

8.3.4.1. 垂直显示查询结果
8.3.4.2. 使用--safe-updates选项
8.3.4.3. 禁用mysql自动连接

该节描述了可以帮助你更有效使用**mysql**的一些技术。

8.3.4.1. 垂直显示查询结果

一些查询结果如果垂直显示而不用通常的水平表格式显示，则更容易读取。用\G而不用分号结束查询可以垂直显示查询。例如，包括新行的更长的文本值垂直输出时通常更容易读取：

```
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
Thimble> with UTF-8 or Unicode? Otherwise, I'll put this on my
Thimble> TODO list and see what happens.

Yes, please do that.

Regards,
Monty
     file: inbox-jani-1
     hash: 190402944
1 row in set (0.09 sec)
```

8.3.4.2. 使用**--safe-updates**选项

对于新手，有一个有用的启动选项--safe-updates(或--i-am-a-dummy，具有相同的效果)。当你已经发出一个DELETE FROM *tbl_name*语句但忘记了WHERE子句时很有用。通常情况，这样的语句从表中删除所有行。用--safe-updates，可以通过指定可以识别它们的键值只删除某些行。这样可以帮助防止事故。

若使用--safe-updates选项，**mysql**连接MySQL服务器时发出下面的语句：

```
SET SQL_SAFE_UPDATES=1,SQL_SELECT_LIMIT=1000, SQL_MAX_JOIN_SIZE=1000000;
```

参见13.5.3节，“SET语法”。

SET语句有下面的效果：

- 不允许你执行UPDATE或DELETE语句，除非在WHERE子句中指定一个键值约束或提供一个LIMIT子句(或二者皆使用)。例如：

- UPDATE *tbl_name* SET *not_key_column=val* WHERE *key_column=val*;
-
- UPDATE *tbl_name* SET *not_key_column=val* LIMIT 1;

- 所有大的SELECT结果自动限制到1,000行，除非语句包括一个LIMIT子句。

- 放弃可能需要检查1,000,000多行组合的多表sELECT语句。

要将限制指定为1,000和1,000,000之外的值，可以使用--select_limit和--max_join_size选项覆盖默认值：

```
shell> mysql --safe-updates --select_limit=500 --max_join_size=10000
```

8.3.4.3. 禁用**mysql**自动连接

如果**mysql**客户程序发送查询时断开与服务器的连接，它立即并自动尝试重新连接服务器并再次发送查询。然而，即使**mysql**重新连接成功，你的第1个连接也已经结束，并且以前的会话对象和设定值被丢失：包括临时表、自动提交模式，以及用户和会话变量。该行为很危险，如下面的例子所示，服务器将在你不知道的情况下关闭并重启：

```
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)
```

@a用户变量已经随连接丢失，并且重新连接后它也没有定义。如果有必要在连接断开时终止**mysql**并提示错误，你可以用--skip-reconnect选项启动**mysql**客户程序。

8.4. mysqlaccess：用于检查访问权限的客户端

mysqlaccess是Yves Carlier为MySQL分发提供的一个诊断工具。它检查访问权限的主机名、用户名和数据库组合。请注意 **mysqlaccess**检查只使用user、db和host表的访问。它不检查在tables_priv、columns_priv或procs_priv表中指定的表、列或者程序的权限。

调用**mysqlaccess**的方法：

```
shell> mysqlaccess [host_name [user_name [db_name]]] [options]
```

mysqlaccess理解下面的选项：

- ---help，-?

显示帮助消息并退出。

- --brief，-b

生成单行表格式的报告。

- --commit

从临时表将新访问权限复制到原授权表。必须为新的权限刷新授权表以使其生效。(例如，执行**mysqladmin RELOAD**命令）。

- --copy

从原授权表重载临时授权表。

- --db=*db_name*，-d *db_name*

指定数据库名。

- ---debug=*N*

指定调试级别。*N*可以为从0到3的一个整数。

- --host=*host_name*，-h *host_name*

在访问权限中使用的主机名。

- --howto

显示一些例子显示如何使用**mysqlaccess**。

- --old_server

假定服务器是一个旧版本的MySQL服务器(MySQL 3.21之前)，还不知道如何处理全WHERE子句。

- --password[=*password*]，-p[*password*]

当连接服务器时使用的密码。如果你在命令行中在--password或-p选项后面省略 密码值，将提示你输入密码。

- --plan

显示将来发布的建议和思想。

- --preview

更改临时授权表后显示权限的不同。

- --relnotes

显示发布注解。

- --rhost=*host_name*，-H *host_name*

连接给定主机的MySQL服务器。

- --rollback

取消对临时授权表的最新的更改。

- --spassword[=*password*]，-P[*password*]

以super用户连接服务器时使用的密码。如果在命令行中在--password或-p选项后面省略了 密码值，将提示你输入密码。

- --superuser=*user_name*，-U *user_name*

指定以super用户连接时的用户名。

- --tables，-t

生成表格式的报告。

- --user=*user_name*，-u *user_name*

在访问权限中使用的主机名。

- --version，-v

显示版本信息并退出。

如果你的MySQL分发安装在某个非标准位置，必须进入**mysqlaccess**期望找到**mysql**客户的目录。编辑大约在18行处的mysqlaccess脚本。搜索类似下面的一行：

```
$MYSQL     = '/usr/local/bin/mysql';    # path to mysql executable
```

将路径更改为**mysql**实际在系统中保存的位置。如果不这样做，当运行**mysqlaccess**时会发生Broken pipe错误。

8.5. mysqladmin：用于管理MySQL服务器的客户端

mysqladmin是一个执行管理操作的客户程序。可以用它来检查服务器的配置和当前的状态，创建并删除数据库等等。

这样调用**mysqladmin**：

```
shell> mysqladmin [options] command [command-options] [command [command-options]] ...
```

mysqladmin支持下面的命令：

- create *db_name*

创建一个名为*db_name*的新数据库。

- debug

告诉服务器向错误日志写入调试信息。

- drop *db_name*

删除名为*db_nam*的数据库和所有表。

- extended-status

显示服务器状态变量及其值。

- flush-hosts

刷新主机缓存中的所有信息。

- flush-logs

刷新所有日志。

- flush-privileges

重载授权表(类似reload)。

- flush-status

清除状态变量。

- flush-tables

刷新所有表。

- flush-threads

刷新线程缓存。

- kill id,id,...

杀掉服务器线程。

- old-password *new-password*

类似password但使用旧的(pre-4.1)密码哈希格式保存 密码。(参见5.7.9节，“MySQL 4.1中的密码哈希处理”）。

- password *new-password*

设置一个新密码。将用**mysqladmin**连接服务器使用的 账户的密码更改为new-password。

如果*new-password*包含空格或其它命令解释符的特殊字符，需要用引号将它引起来。在Windows中，一定要使用双引号而不要用单引号；单引号不会从 密码中剥离出来，而是解释为密码的一部分。例如：

shell> mysqladmin password "my new password"

- ping

检查服务器是否仍活动。如果服务器在运行**mysqladmin**返回状态0，如果不运行返回1。即使出现错误例如Access denied也为0，因为这说明服务器在运行但拒绝了连接，与服务器不在运行不同。

- processlist

显示活动服务器线程的列表。类似SHOW PROCESSLIST语句的输出。如果给出了--verbose选项，输出类似SHOW FULL PROCESSLIST。(参见13.5.4.16节，“SHOW PROCESSLIST语法”）。

- reload

重载授权表。

- refresh

刷新所有表并关闭和打开日志文件。

- shutdown

停止服务器。

- start-slave

开始从服务器上的复制。

- status

显示短服务器状态消息。

- stop-slave

停止从服务器上的复制。

- variables

显示服务器系统变量及其值。

- version

显示服务器的版本信息。

所有命令可以简化为任何唯一的前缀。例如:

```
shell> mysqladmin proc stat
+----+-------+-----------+----+---------+------+-------+------------------+
| Id | User  | Host      | db | Command | Time | State | Info             |
+----+-------+-----------+----+---------+------+-------+------------------+
| 51 | monty | localhost |    | Query   | 0    |       | show processlist |
+----+-------+-----------+----+---------+------+-------+------------------+
Uptime: 1473624  Threads: 1  Questions: 39487
Slow queries: 0  Opens: 541  Flush tables: 1
Open tables: 19  Queries per second avg: 0.0268
```

mysqladmin status命令的结果显示下面的值:

- Uptime

MySQL服务器已经运行的秒数。

- Threads

活动线程(客户)的数目。

- Questions

服务器启动以来客户的问题(查询)数目。

- Slow queries

执行时间超过long_query_time秒的查询的数量。参见5.11.4节，“慢速查询日志”。

- Opens

服务器已经打开的数据库表的数量。

- Flush tables

服务器已经执行的flush ...、refresh和reload命令的数量。

- Open tables

目前打开的表的数量。

- Memory in use

mysqld代码直接分配的内存数量。只有用--with--debug=full编译了MySQL该值才显示。

- Maximum memory used

mysqld代码直接分配的最大内存数量。只有用--with--debug=full编译了MySQL该值才显示。

如果当使用Unix套接字文件连接本地服务器时执行**mysqladmin shutdown**，**mysqladmin**将等待直到服务器的进程ID文件被删除，以确保服务器正确停止。

mysqladmin支持下面的选项：

· ---help，-?

显示帮助消息并退出。

· --character-sets-dir=*path*

字符集的安装目录。参见5.10.1节，“数据和排序用字符集”。

· --compress，-C

压缩客户和服务器之间发送的所有信息（如果二者均支持压缩）。

· --count=*num*，-c *num*

迭代数目。该选项只有结合--sleep (-i)才能工作。

· ---debug[=*debug_options*]，-# [*debug_options*]

写调试日志。*debug_options*字符串通常为'd:t:o,*file_name*'。 默认为'd:t:o,/tmp/mysqladmin.trace'。

· --default-character-set=*charset*

使用*charset*as作为默认字符集。参见5.10.1节，“数据和排序用字符集”。

· --force，-f

不再为drop database命令进行确认。对于多个命令，即使出现错误也继续。

· --host=*host_name*，-h *host_name*

连接给定主机上的MySQL服务器。

· --password[=*password*]，-p[*password*]

连接服务器使用的密码。如果使用短选项形式(-p)，该选项和 密码之间不能有空格。如果你在命令行中在--password或-p选项后面省略 密码值，将提示你输入密码。

· --port=*port_num*，-P *port_num*

用于连接的TCP/IP端口号。

· --protocol={TCP | SOCKET | PIPE | MEMORY}

使用的连接协议。

· --relative，-r

当带-I使用时显示当前和前面值的差别。目前，该选项只用于extended-status命令。

· --silent，-s

如果不能建立与服务器的连接则以沉默方式退出。

- --sleep=*delay*，-i *delay*

每睡眠*delay*秒后执行一次命令。

- --socket=*path*，-S *path*

用于连接的套接字文件。

- --user=*user_name*，-u *user_name*

当连接服务器时使用的MySQL用户名。

- --verbose，-v

冗长模式。打印出程序操作的详细信息。

- --version，-V

显示版本信息并退出。

- --vertical，-E

垂直打印输出。类似于--relative，但垂直打印输出。

- --wait[=*count*]，-w[*count*]

如果连接不能建立，等待并重试而不是放弃。如果给出一个选项值，则指示重试的次数。默认是一次。

也可以使用--*var_name*=*value*选项设置下面的变量：

- connect_timeout

连接超时之前的最大秒数。默认值为43200(12小时)。

- shutdown_timeout

等候关闭的最大秒数。默认值为3600(1小时)。

也可以使用--set-variable=*var_name*=*value*或-O *var_name*=*value*语法来设置变量。然而，现在不赞成该语法，并且不再使用。

8.6. mysqlbinlog：用于处理二进制日志文件的实用工具

服务器生成的二进制日志文件写成二进制格式。要想检查这些文本格式的文件，应使用**mysqlbinlog**实用工具。

应这样调用**mysqlbinlog**：

```
shell> mysqlbinlog [options] log-files...
```

例如，要想显示二进制日志binlog.000003的内容，使用下面的命令：

```
shell> mysqlbinlog binlog.0000003
```

输出包括在binlog.000003中包含的所有语句，以及其它信息例如每个语句花费的时间、客户发出的线程ID、发出线程时的时间戳等等。

通常情况，可以使用**mysqlbinlog**直接读取二进制日志文件并将它们用于本地MySQL服务器。也可以使用--read-from-remote-server选项从远程服务器读取二进制日志。

当读取远程二进制日志时，可以通过连接参数选项来指示如何连接服务器，但它们经常被忽略掉，除非你还指定了--read-from-remote-server选项。这些选项是--host、--password、--port、--protocol、--socket和--user。

还可以使用**mysqlbinlog**来读取在复制过程中从服务器所写的中继日志文件。中继日志格式与二进制日志文件相同。

在5.11.3节，"二进制日志"中详细讨论了二进制日志。

mysqlbinlog支持下面的选项：

· ---help，-?

显示帮助消息并退出。

· ---database=*db_name*，-d *db_name*

只列出该数据库的条目(只用本地日志)。

· --force-read，-f

使用该选项，如果**mysqlbinlog**读它不能识别的二进制日志事件，它会打印警告，忽略该事件并继续。没有该选项，如果**mysqlbinlog**读到此类事件则停止。

· --hexdump，-H

在注释中显示日志的十六进制转储。该输出可以帮助复制过程中的调试。在MySQL 5.1.2中添加了该选项。

· --host=*host_name*，-h *host_name*

获取给定主机上的MySQL服务器的二进制日志。

· --local-load=*path*，-l *pat*

为指定目录中的LOAD DATA INFILE预处理本地临时文件。

· --offset=*N*，-o *N*

跳过前*N*个条目。

· --password[=*password*]，-p[*password*]

当连接服务器时使用的密码。如果使用短选项形式(-p)，选项和 密码之间不能有空格。如果在命令行中--password或-p选项后面没有 密码值，则提示输入一个密码。

· --port=*port_num*，-P port_*num*

用于连接远程服务器的TCP/IP端口号。

· --position=*N*，-j *N*

不赞成使用，应使用--start-position。

· --protocol={TCP | SOCKET | PIPE | -position

使用的连接协议。

- --read-from-remote-server，-R

从MySQL服务器读二进制日志。如果未给出该选项，任何连接参数选项将被忽略。这些选项是--host、--password、--port、--protocol、--socket和--user。

- --result-file=*name*, -r *name*

将输出指向给定的文件。

- --short-form，-s

只显示日志中包含的语句，不显示其它信息。

- --socket=*path*，-S *path*

用于连接的套接字文件。

- --start-datetime=*datetime*

从二进制日志中第1个日期时间等于或晚于*datetime*参量的事件开始读取。*datetime*值相对于运行**mysqlbinlog**的机器上的本地时区。该值格式应符合DATETIME或TIMESTAMP数据类型。例如：

```
shell> mysqlbinlog --start-datetime="2004-12-25 11:25:56" binlog.000003
```

该选项可以帮助点对点恢复。

- --stop-datetime=*datetime*

从二进制日志中第1个日期时间等于或晚于*datetime*参量的事件起停止读。关于*datetime*值的描述参见--start-datetime选项。该选项可以帮助及时恢复。

- --start-position=*N*

从二进制日志中第1个位置等于*N*参量时的事件开始读。

- --stop-position=*N*

从二进制日志中第1个位置等于和大于*N*参量时的事件起停止读。

- --to-last-logs，-t

在MySQL服务器中请求的二进制日志的结尾处不停止，而是继续打印直到最后一个二进制日志的结尾。如果将输出发送给同一台MySQL服务器，会导致无限循环。该选项要求--read-from-remote-server。

- --disable-logs-bin，-D

禁用二进制日志。如果使用--to-last-logs选项将输出发送给同一台MySQL服务器，可以避免无限循环。该选项在崩溃恢复时也很有用，可以避免复制已经记录的语句。注释：该选项要求有SUPER权限。

- --user=*user_name*，-u *user_name*

连接远程服务器时使用的MySQL用户名。

- --version，-V

显示版本信息并退出。

还可以使用--var_name=value选项设置下面的变量:

- open_files_limit

指定要保留的打开的文件描述符的数量。

可以将**mysqlbinlog**的输出传到**mysql**客户端以执行包含在二进制日志中的语句。如果你有一个旧的备份，该选项在崩溃恢复时也很有用(参见5.9.1节，“数据库备份”):

```
shell> mysqlbinlog hostname-bin.000001 | mysql
```

或:

```
shell> mysqlbinlog hostname-bin.[0-9]* | mysql
```

如果你需要先修改含语句的日志，还可以将**mysqlbinlog**的输出重新指向一个文本文件。(例如，想删除由于某种原因而不想执行的语句)。编辑好文件后，将它输入到**mysql**程序并执行它包含的语句。

mysqlbinlog有一个--position选项，只打印那些在二进制日志中的偏移量大于或等于某个给定位置的语句(给出的位置必须匹配一个事件的开始)。它还有在看见给定日期和时间的事件后停止或启动的选项。这样可以使用--stop-datetime选项进行点对点恢复(例如，能够说“将数据库前滚动到今天10:30 AM的位置”)。

如果MySQL服务器上有多个要执行的二进制日志，安全的方法是在一个连接中处理它们。下面是一个说明什么是不安全的例子:

```
shell> mysqlbinlog hostname-bin.000001 | mysql # DANGER!!
shell> mysqlbinlog hostname-bin.000002 | mysql # DANGER!!
```

使用与服务器的不同连接来处理二进制日志时，如果第1个日志文件包含一个CREATE TEMPORARY TABLE语句，第2个日志包含一个使用该临时表的语句，则会造成问题。当第1个**mysql**进程结束时，服务器撤销临时表。当第2个**mysql**进程想使用该表时，服务器报告“不知道该表”。

要想避免此类问题，使用一个连接来执行想要处理的所有二进制日志中的内容。下面提供了一种方法:

```
shell> mysqlbinlog hostname-bin.000001 hostname-bin.000002 | mysql
```

另一个方法是:

```
shell> mysqlbinlog hostname-bin.000001 >  /tmp/statements.sql
shell> mysqlbinlog hostname-bin.000002 >> /tmp/statements.sql
shell> mysql -e "source /tmp/statements.sql"
```

mysqlbinlog产生的输出可以不需要原数据文件即可重新生成一个LOAD DATA INFILE操作。**mysqlbinlog**将数据复制到一个临时文件并写一个引用该文件的LOAD DATA LOCAL INFILE语句。由系统确定写入这些文件的目录的默认位置。要想显式指定一个目录，使用--local-load选项。

因为**mysqlbinlog**可以将LOAD DATA INFILE语句转换为LOAD DATA LOCAL INFILE语句(也就是说，它添加了LOCAL)，用于处理语句的客户端和服务器必须配置为允许LOCAL操作。参见5.6.4节，“LOAD DATA LOCAL安全问题”。

警告：为LOAD DATA LOCAL语句创建的临时文件不会自动删除，因为在实际执行完那些语句前需要它们。不再需要语句日志后应自己删除临时文件。文件位于临时文件目录中，文件名类似original_file_name-#-#。

--hexdump选项可以在注释中产生日志内容的十六进制转储:

```
shell> mysqlbinlog --hexdump master-bin.000001
```

上述命令的输出应类似:

```
/*!40019 SET @@session.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
# at 4
#051024 17:24:13 server id 1  end_log_pos 98
# Position  Timestamp   Type   Master ID        Size      Master Pos    Flags
# 00000004 9d fc 5c 43   0f   01 00 00 00   5e 00 00 00   62 00 00 00   00 00
# 00000017 04 00 35 2e 30 2e 31 35  2d 64 65 62 75 67 2d 6c |..5.0.15.debug.l|
# 00000027 6f 67 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |og..............|
# 00000037 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |................|
# 00000047 00 00 00 00 9d fc 5c 43  13 38 0d 00 08 00 12 00 |.......C.8......|
# 00000057 04 04 04 04 12 00 00 4b  00 04 1a                 |.......K...|
#       Start: binlog v 4, server v 5.0.15-debug-log created 051024 17:24:13
#       at startup
ROLLBACK;
```

十六进制转储的输出包含下面的元素:

- Position: The byte position within the log file.
- Timestamp: The event timestamp. In the example just shown, '9d fc 5c 43' is the representation of '051024 17:24:13' in hexadecimal.
- Type: The type of the log event. '0f' means that the example event is a FORMAT_DESCRIPTION_EVENT. The types are:

```
·                00  UNKNOWN_EVENT
·                    This event should never be present in the log.
·                01  START_EVENT_V3
·                    This indicates the start of a log file written by MySQL 4 or earlier.
·                02  QUERY_EVENT
·                    The most common type of events.  These contain queries executed
·                    on the master.
·                03  STOP_EVENT
·                    Indicates that master has stopped.
·                04  ROTATE_EVENT
·                    Written when the master switches to a new log file.
·                05  INTVAR_EVENT
·                    Used mainly for AUTO_INCREMENT values and if the LAST_INSERT_ID()
·                    function is used in the statement.
·                06  LOAD_EVENT
·                    Used for LOAD DATA INFILE in MySQL 3.23.
```

```
07  SLAVE_EVENT
    Reserved for future use.
08  CREATE_FILE_EVENT
    Used for LOAD DATA INFILE statements.  This indicates the start
    of execution of such a statement.  A temporary file is created
    on the slave.  Used in MySQL 4 only.
09  APPEND_BLOCK_EVENT
    Contains data for use in a LOAD DATA INFILE statement.  The
    data is stored in the temporary file on the slave.
0a  EXEC_LOAD_EVENT
    Used for LOAD DATA INFILE statements.  The contents of the
    temporary file is stored in the table on the slave.
    Used in MySQL 4 only.
0b  DELETE_FILE_EVENT
    Rollback of LOAD DATA INFILE statement.  The temporary file
    should be deleted on slave.
0c  NEW_LOAD_EVENT
    Used for LOAD DATA INFILE in MySQL 4 and earlier.
0d  RAND_EVENT
    Used to send information about random values if the RAND()
    function is used in the query.
0e  USER_VAR_EVENT
    Used to replicate user variables.
0f  FORMAT_DESCRIPTION_EVENT
    This indicates the start of a log file written by MySQL 5 or later.
10  XID_EVENT
    Event indicating commit of XA transaction
11  BEGIN_LOAD_QUERY_EVENT
    Used for LOAD DATA statements in MySQL 5 and later.
12  EXECUTE_LOAD_QUERY_EVENT
    Used for LOAD DATA statements in MySQL 5 and later.
13  TABLE_MAP_EVENT
    Reserved for future use
14  WRITE_ROWS_EVENT
    Reserved for future use
15  UPDATE_ROWS_EVENT
    Reserved for future use
16  DELETE_ROWS_EVENT
    Reserved for future use
```

- Master ID: The server id of the master that created the event.
- Size: The size in bytes of the event.
- Master Pos: The position of the event in the original master log file.
- Flags: 16 flags.

```
01  LOG_EVENT_BINLOG_IN_USE_F
    Log file correctly closed (Used only in FORMAT_DESCRIPTION_EVENT)
    If this flag is set (if the flags are e.g. '01 00') in an
    FORMAT_DESCRIPTION_EVENT, then the log file has not been
    properly closed.  Most probably because of a master crash (for
    example, due to power failure).
02  Reserved for future use.
04  LOG_EVENT_THREAD_SPECIFIC_F
    Set if the event is dependent on the connection it was
    executed in (example '04 00'), e.g. if the event uses
    temporary tables.
08  LOG_EVENT_SUPPRESS_USE_F
    Set in some circumstances when the event is not dependent on
    the current database
```

其它标志保留用于将来使用。

在以后的版本中十六进制转储输出的格式可能会改变。

8.7. mysqlcheck：表维护和维修程序

mysqlcheck客户端可以检查和修复MyISAM表。它还可以优化和分析表。

mysqlcheck的功能类似**myisamchk**，但其工作不同。主要差别是当**mysqld**服务器在运行时必须使用**mysqlcheck**，而**myisamchk**应用于服务器没有运行时。使用**mysqlcheck**的好处是不需要停止服务器来检查或修复表。

Mysqlcheck为用户提供了一种方便的使用SQL语句CHECK TABLE、REPAIR TABLE、ANALYZE TABLE和OPTIMIZE TABLE的方式。它确定在要执行的操作中使用使用哪个语句，然后将语句发送到要执行的服务器上。

有3种方式来调用**mysqlcheck**:

```
shell> mysqlcheck[options] db_name [tables]
shell> mysqlcheck[options] ---database DB1 [DB2 DB3...]
shell> mysqlcheck[options] --all--database
```

如果没有指定任何表或使用---database或--all--database选项，则检查整个数据库。

同其它客户端比较，**mysqlcheck**有一个特殊特性。重新命名二进制可以更改检查表的默认行为(--check)。如果你想要一个工具默认可以修复表的工具，只需要将**mysqlcheck**重新复制为**mysqlrepair**，或者使用一个符号链接**mysqlrepair**链接**mysqlcheck**。如果调用**mysqlrepair**，可按照命令修复表。

下面的名可用来更改**mysqlcheck**的默认行为:

mysqlrepair	默认选项为--repair
mysqlanalyze	默认选项为--analyze
mysqloptimize	默认选项为--optimize

mysqlcheck支持下面的选项:

· ---help，-?

显示帮助消息并退出。

· --all--database，-A

检查所有数据库中的所有表。与使用---database选项相同，在命令行中命名所有数据库。

· --all-in-1，-1

不是为每个表发出一个语句，而是为命名数据库中待处理的所有表的每个数据库执行一个语句。

· --analyze，-a

分析表。

· --auto-repair

如果某个被检查的表破坏了，自动修复它。检查完所有表后自动进行所有需要的修复。

· --character-sets-dir=*path*

字符集的安装目录。参见5.10.1节，“数据和排序用字符集”。

· --check，-c

检查表的错误。

· --check-only-changed，-C

只检查上次检查以来已经更改的或没有正确关闭的表。

· --compress

压缩在客户端和服务器之间发送的所有信息（如果二者均支持压缩）。

· ---database，-B

处理数据库中命名的所有表。使用该选项，所有字名参量被看作数据库名，而不是表名。

· ---debug[=*debug_options*]，-# [*debug_options*]

写调试日志。*debug_options*字符串通常为'd:t:o,*file_name*'。

· --default-character-set=*charset*

使用*charset*as默认字符集。参见5.10.1节，“数据和排序用字符集”。

- --extended，-e

如果你正使用该选项来检查表，可以确保它们100%地一致，但需要很长的时间。

如果你正使用该选项来修复表，则运行扩展修复，不但执行的时间很长，而且还会产生大量的垃圾行!

- --fast，-F

只检查没有正确关闭的表。

- --force，-f

即使出现SQL错误也继续。

- --host=*host_name*，-h *host_name*

连接给定主机上的MySQL服务器。

- --medium-check，-m

执行比--extended操作更快的检查。只能发现99.99%的错误，在大多数情况下这已经足够了。

- --optimize，-o

优化表。

- --password[=*password*]，-p[*password*]

当连接服务器时使用的密码。如果使用短选项形式(-p)，选项和 密码之间不能有空格。如果在命令行中--password或-p选项后面没有 密码值，则提示输入一个密码。

- --port=*port_num*，-P port_*num*

用于连接的TCP/IP端口号。

- --protocol={TCP | SOCKET | PIPE | MEMORY}

使用的连接协议。

- --quick，-q

如果你正使用该选项在检查表，它防止扫描行以检查错误链接的检查。这是最快的检查方法。

如果你正使用该选项在修复表，它尝试只修复索引树。这是最快的修复方法。

- --repair，-r

执行可以修复大部分问题的修复，只是唯一值不唯一时不能修复。

- --silent，-s

沉默模式。只打印错误消息。

- --socket=*path*，-S *path*

用于连接的套接字文件。

- --tables

覆盖---database或-B选项。选项后面的所有参量被视为表名。

- --user=*user_name*，-u *user_name*

当连接服务器时使用的MySQL用户名。

- --verbose，-v

冗长模式。打印关于各阶段程序操作的信息。

- --version，-V

显示版本信息并退出。

8.8. mysqldump：数据库备份程序

mysqldump客户端可用来转储数据库或搜集数据库进行备份或将数据转移到另一个SQL服务器(不一定是一个MySQL服务器)。转储包含创建表和/或装载表的SQL语句。

如果你在服务器上进行备份，并且表均为MyISAM表，应考虑使用**mysqlhotcopy**，因为可以更快地进行备份和恢复。参见8.9节，“mysqlhotcopy：数据库备份程序”。

有3种方式来调用**mysqldump**：

```
shell> mysqldump [options] db_name [tables]
shell> mysqldump [options] ---database DB1 [DB2 DB3...]
shell> mysqldump [options] --all--database
```

如果没有指定任何表或使用了---database或--all--database选项，则转储整个数据库。

要想获得你的版本的**mysqldump**支持的选项，执行**mysqldump ---help**。

如果运行**mysqldump**没有--quick或--opt选项，**mysqldump**在转储结果前将整个结果集装入内存。如果转储大数据库可能会出现问题。该选项默认启用，但可以用--skip-opt禁用。

如果使用最新版本的**mysqldump**程序生成一个转储重装到很旧版本的MySQL服务器中，不应使用--opt或-e选项。

mysqldump支持下面的选项：

- ---help，-?

显示帮助消息并退出。

- --add-drop--database

在每个CREATE DATABASE语句前添加DROP DATABASE语句。

- --add-drop-tables

在每个CREATE TABLE语句前添加DROP TABLE语句。

- --add-locking

用LOCK TABLES和UNLOCK TABLES语句引用每个表转储。重载转储文件时插入得更快。参见7.2.16节，“INSERT语句的速度”。

· --all--database，-A

转储所有数据库中的所有表。与使用---database选项相同，在命令行中命名所有数据库。

· --allow-keywords

允许创建关键字列名。应在每个列名前面加上表名前缀。

· ---comments[={0|1}]

如果设置为 0，禁止转储文件中的其它信息，例如程序版本、服务器版本和主机。--skip—comments与---comments=0的结果相同。 默认值为1，即包括额外信息。

· --compact

产生少量输出。该选项禁用注释并启用--skip-add-drop-tables、--no-set-names、--skip-disable-keys和--skip-add-locking选项。

· --compatible=*name*

产生与其它数据库系统或旧的MySQL服务器更兼容的输出。值可以为ansi、mysql323、mysql40、postgresql、oracle、mssql、db2、maxdb、no_key_options、no_tables_options或者no_field_options。要使用几个值，用逗号将它们隔开。这些值与设置服务器SQL模式的相应选项有相同的含义。参见5.3.2节，“SQL服务器模式”。

该选项不能保证同其它服务器之间的兼容性。它只启用那些目前能够使转储输出更兼容的SQL模式值。例如，--compatible=oracle 不映射Oracle类型或使用Oracle注释语法的数据类型。

· --complete-insert，-c

使用包括列名的完整的INSERT语句。

· --compress，-C

压缩在客户端和服务器之间发送的所有信息（如果二者均支持压缩）。

· --create-option

在CREATE TABLE语句中包括所有MySQL表选项。

· ---database，-B

转储几个数据库。通常情况，**mysqldump**将命令行中的第1个名字参量看作数据库名，后面的名看作表名。使用该选项，它将所有名字参量看作数据库名。CREATE DATABASE IF NOT EXISTS *db_name*和USE *db_name*语句包含在每个新数据库前的输出中。

· ---debug[=*debug_options*]，-# [*debug_options*]

写调试日志。*debug_options*字符串通常为'd:t:o,*file_name*'。

· --default-character-set=*charset*

使用*charset*as默认字符集。参见5.10.1节，“数据和排序用字符集”。如果没有指定，**mysqldump**使用utf8。

- --delayed-insert

使用INSERT DELAYED语句插入行。

- --delete-master-logs

在主复制服务器上，完成转储操作后删除二进制日志。该选项自动启用--master-data。

- --disable-keys，-K

对于每个表，用/*!40000 ALTER TABLE *tbl_name* DISABLE KEYS */;和/*!40000 ALTER TABLE *tbl_name* ENABLE KEYS */;语句引用INSERT语句。这样可以更快地装载转储文件，因为在插入所有行后创建索引。该选项只适合MyISAM表。

- --extended-insert，-e

使用包括几个VALUES列表的多行INSERT语法。这样使转储文件更小，重载文件时可以加速插入。

- --fields-terminated-by=...，--fields-enclosed-by=...，--fields-optionally-enclosed-by=...，--fields-escaped-by=...，--行-terminated-by=...

这些选项结合-T选项使用，与LOAD DATA INFILE的相应子句有相同的含义。参见13.2.5节，“LOAD DATA INFILE语法”。

- --first-slave，-x

不赞成使用，现在重新命名为--lock-all-tables。

- --flush-logs，-F

开始转储前刷新MySQL服务器日志文件。该选项要求RELOAD权限。请注意如果结合--all--database(或-A)选项使用该选项，根据每个转储的数据库刷新日志。例外情况是当使用--lock-all-tables或--master-data的时候：在这种情况下，日志只刷新一次，在所有 表被锁定后刷新。如果你想要同时转储和刷新日志，应使用--flush-logs连同--lock-all-tables或--master-data。

- --force，-f

在表转储过程中，即使出现SQL错误也继续。

- --host=*host_name*，-h *host_name*

从给定主机的MySQL服务器转储数据。默认主机是localhost。

- --hex-blob

使用十六进制符号转储二进制字符串列(例如，'abc' 变为0x616263)。影响到的列有BINARY、VARBINARY、BLOB。

- --lock-all-tables，-x

所有数据库中的所有表加锁。在整体转储过程中通过全局读锁定来实现。该选项自动关闭--single-transaction和--lock-tables。

- --lock-tables，-l

开始转储前锁定所有表。用READ LOCAL锁定表以允许并行插入MyISAM表。对于事务表例如InnoDB和BDB，--single-transaction是一个更好的选项，因为它不根本需要锁定表。

请注意当转储多个数据库时，--lock-tables分别为每个数据库锁定表。因此，该选项不能保证转储文件中的表在数据库之间的逻辑一致性。不同数据库表的转储状态可以完全不同。

· --master-data[=*value*]

该选项将二进制日志的位置和文件名写入到输出中。该选项要求有RELOAD权限，并且必须启用二进制日志。如果该选项值等于1，位置和文件名被写入CHANGE MASTER语句形式的转储输出，如果你使用该SQL转储主服务器以设置从服务器，从服务器从主服务器二进制日志的正确位置开始。如果选项值等于2，CHANGE MASTER语句被写成SQL注释。如果*value*被省略，这是默认动作。

--master-data选项启用--lock-all-tables，除非还指定--single-transaction(在这种情况下，只在刚开始转储时短时间获得全局读锁定。又见--single-transaction。在任何一种情况下，日志相关动作发生在转储时。该选项自动关闭--lock-tables。

· --no-create-db，-n

该选项禁用CREATE DATABASE /*!32312 IF NOT EXISTS*/ db_name语句，如果给出---database或--all--database选项，则包含到输出中。

· --no-create-info，-t

不写重新创建每个转储表的CREATE TABLE语句。

· --no-data，-d

不写表的任何行信息。如果你只想转储表的结构这很有用。

· --opt

该选项是速记；等同于指定 --add-drop-tables--add-locking --create-option --disable-keys--extended-insert --lock-tables --quick --set-charset。它可以给出很快的转储操作并产生一个可以很快装入MySQL服务器的转储文件。该选项默认开启，但可以用*--skip-opt*禁用。要想只禁用确信用-opt启用的选项，使用--skip形式；例如，--skip-add-drop-tables或--skip-quick。

· --password[=*password*]，-p[*password*]

连接服务器时使用的密码。如果你使用短选项形式(-p)，不能在选项和密码之间有一个空格。如果在命令行中，忽略了--password或-p选项后面的 密码值，将提示你输入一个。

· --port=*port_num*，-P port_*num*

用于连接的TCP/IP端口号。

· --protocol={TCP | SOCKET | PIPE | MEMORY}

使用的连接协议。

· --quick，-q

该选项用于转储大的表。它强制**mysqldump**从服务器一次一行地检索表中的行而不是检索所有行并在输出前将它缓存到内存中。

· --quote-names，-Q

用`` ’字符引用数据库、表和列名。如果服务器SQL模式包括ANSI_QUOTES选项，用`"’字符引用名。默认启用该选项。可以用--skip-quote-names禁用，但该选项应跟在其它选项后面，例如可以启用--quote-names的--

compatible。

· --result-file=*file*，-r *file*

将输出转向给定的文件。该选项应用在Windows中，因为它禁止将新行'\n'字符转换为'\r\n'回车、返回/新行序列。

· --routines，-R

在转储的数据库中转储存储程序(函数和程序)。使用---routines产生的输出包含CREATE PROCEDURE和CREATE FUNCTION语句以重新创建子程序。但是，这些语句不包括属性，例如子程序定义者或创建和修改时间戳。这说明当重载子程序时，对它们进行创建时定义者应设置为重载用户，时间戳等于重载时间。

如果你需要创建的子程序使用原来的定义者和时间戳属性，不使用--routines。相反，使用一个具有mysql数据库相应权限的MySQL账户直接转储和重载mysql.proc表的内容。

该选项在MySQL 5.1.2中添加进来。在此之前，存储程序不转储。

· --set-charset

将SET NAMES *default_character_set*加到输出中。该选项默认启用。要想禁用SET NAMES语句，使用--skip-set-charset。

· --single-transaction

该选项从服务器转储数据之前发出一个BEGIN SQL语句。它只适用于事务表，例如InnoDB和BDB，因为然后它将在发出BEGIN而没有阻塞任何应用程序时转储一致的数据库状态。

当使用该选项时，应记住只有InnoDB表能以一致的状态被转储。例如，使用该选项时任何转储的MyISAM或HEAP表仍然可以更改状态。

--single-transaction选项和--lock-tables选项是互斥的，因为LOCK TABLES会使任何挂起的事务隐含提交。

要想转储大的表，应结合--quick使用该选项。

· --socket=*path*，-S *path*

当连接localhost(为默认主机)时使用的套接字文件。

· --skip--comments

参见---comments选项的描述。

· --tab=*path*，-T *path*

产生tab分割的数据文件。对于每个转储的表，**mysqldump**创建一个包含创建表的CREATE TABLE语句的*tbl_name*.sql文件，和一个包含其数据的*tbl_name*.txt文件。选项值为写入文件的目录。

默认情况，.txt数据文件的格式是在列值和每行后面的新行之间使用tab字符。可以使用--fields-*xxx*和--行--*xxx*选项明显指定格式。

注释：该选项只适用于**mysqldump**与**mysqld**服务器在同一台机器上运行时。你必须具有FILE权限，并且服务器必须有在你指定的目录中有写文件的许可。

· --tables

覆盖---database或-B选项。选项后面的所有参量被看作表名。

- --triggers

为每个转储的表转储触发器。该选项默认启用；用--skip-triggers禁用它。

- --tz-utc

在转储文件中加入SET TIME_ZONE='+00:00'以便TIMESTAMP列可以在具有不同时区的服务器之间转储和重载。(不使用该选项，TIMESTAMP列在具有本地时区的源服务器和目的服务器之间转储和重载）。--tz-utc也可以保护由于夏令时带来的更改。--tz-utc默认启用。要想禁用它，使用--skip-tz-utc。该选项在MySQL 5.1.2中加入。

- --user=*user_name*，-u *user_name*

连接服务器时使用的MySQL用户名。

- --verbose，-v

冗长模式。打印出程序操作的详细信息。

- --version，-V

显示版本信息并退出。

- --where='*where-condition*', -w '*where-condition*'

只转储给定的WHERE条件选择的记录。请注意如果条件包含命令解释符专用空格或字符，一定要将条件引用起来。

例如：

```
"--where=user='jimf'"
"-wuserid>1"
"-wuserid<1"
```

- --xml，-X

将转储输出写成XML。

还可以使用--var_name=value选项设置下面的变量：

- max_allowed_packet

客户端/服务器之间通信的缓存区的最大大小。最大为1GB。

- net_buffer_length

客户端/服务器之间通信的缓存区的初始大小。当创建多行插入语句时(如同使用选项--extended-insert或--opt)，**mysqldump**创建长度达net_buffer_length的行。如果增加该变量，还应确保在MySQL服务器中的net_buffer_length变量至少这么大。

还可以使用--set-variable=*var_name*=*value*或-O *var_name*=*value*语法设置变量。然而，现在不赞成使用该语法。

mysqldump最常用于备份一个整个的数据库：

```
shell> mysqldump --opt db_name > backup-file.sql
```

你可以这样将转储文件读回到服务器:

```
shell> mysql db_name < backup-file.sql
```

或者为:

```
shell> mysql -e "source /path-to--backup/backup-file.sql" db_name
```

mysqldump也可用于从一个MySQL服务器向另一个服务器复制数据时装载数据库:

```
shell> mysqldump --opt db_name | mysql --host=remote_host -C db_name
```

可以用一个命令转储几个数据库:

```
shell> mysqldump ---database db_name1 [db_name2 ...] > my_databases.sql
```

如果你想要转储所有数据库，使用--all--database选项:

```
shell> mysqldump --all-databases > all_databases.sql
```

如果表保存在InnoDB存储引擎中，mysqldump提供了一种联机备份的途径(参见下面的命令)。该备份只需要在开始转储时对所有表进行全局读锁定(使用FLUSH TABLES WITH READ LOCK)。获得锁定后，读取二进制日志的相应内容并将锁释放。因此如果并且只有当发出FLUSH...时正执行一个长的更新语句，MySQL服务器才停止直到长语句结束，然后转储则释放锁。因此如果MySQL服务器只接收到短("短执行时间")的更新语句，即使有大量的语句，也不会注意到锁期间。

```
shell> mysqldump --all-databases --single-transaction > all_databases.sql
```

对于点对点恢复(也称为"前滚"，当你需要恢复旧的备份并重放该备份以后的更改时)，循环二进制日志(参见5.11.3节，"二进制日志")或至少知道转储对应的二进制日志内容很有用:

```
shell> mysqldump --all-databases --master-data=2 > all_databases.sql
```

或

```
shell> mysqldump --all-databases --flush-logs --master-data=2 > all_databases.sql
```

如果表保存在InnoDB存储引擎中，同时使用--master-data和--single-transaction提供了一个很方便的方式来进行适合点对点恢复的联机备份。

关于备份的详细信息，参见5.9.1节，"数据库备份"。

8.9. mysqlhotcopy：数据库备份程序

mysqlhotcopy是一个Perl脚本，最初由Tim Bunce编写并提供。它使用LOCK TABLES、FLUSH TABLES和cp或scp来快速备份数据库。它是备份数据库或单个表的最快的途径，但它只能运行在数据库目录所在的机器上。**mysqlhotcopy**只用于备份MyISAM。它运行在Unix和NetWare中。

```
shell> mysqlhotcopy db_name [/path/to/new_directory]
shell> mysqlhotcopy db_name_1 ... db_name_n /path/to/new_directory
```

备份给定数据库中的匹配正则表达式的表:

```
shell> mysqlhotcopy db_name./regex/
```

加上发音符('~')前缀，表名的正则表达式可以被否定:

```
shell> mysqlhotcopy db_name./~regex/
```

mysqlhotcopy支持下面的选项:

· ---help，-?

显示帮助消息并退出。

· --allowold

如果目标存在不放弃(加上一个_old后缀重新命名它)。

· --checkpoint=*db_name.tbl_name*

在指定的*db_name.tbl_name*插入检查点条目。

· ---debug

启用调试输出。

· --dryrun，-n

报告动作而不执行它们。

· --flushlog

所有表锁定后刷新日志。

· --keepold

完成后不删除以前(重新命名的)的目标。

· -- method=*command*

复制方法(cp或scp)。

· --noindices

备份中不包括全部索引文件。这样使备份更小、更快。可以在以后用**myisamchk -rq**重新构建索引。

· --password=*password*，-p *password*

当连接服务器时使用的密码。请注意该选项的密码值是不可选的，不象其它MySQL程序。

· --port=*port_num*，-P port_*num*

当连接本地服务器时使用的TCP/IP端口号。

· --quiet，-q

除了出现错误时保持沉默。

· --regexp=*expr*

复制所有数据库名匹配给出的正则表达式的数据库。

· --socket=*path*，-S *path*

用于连接的Unix套接字文件。

- --suffix=*str*

所复制的数据库名的后缀。

- --tmpdir=*path*

临时目录(代替/tmp)。

- --user=*user_name*，-u *user_name*

当连接服务器时使用的MySQL用户名。

mysqlhotcopy从选项文件读取[client]和[mysqlhotcopy]选项组。

要想执行**mysqlhotcopy**，你必须可以访问备份的表文件，具有那些表的SELECT权限和RELOAD权限(以便能够执行FLUSH TABLES)。

使用perldoc调用其它**mysqlhotcopy**文档：

```
shell> perldoc mysqlhotcopy
```

8.10. mysqlimport：数据导入程序

mysqlimport客户端提供了LOAD DATA INFILEQL语句的一个命令行接口。**mysqlimport**的大多数选项直接对应LOAD DATA INFILE子句。参见13.2.5节，“LOAD DATA INFILE语法”。

这样调用**mysqlimport**：

```
shell> mysqlimport [options] db_name textfile1 [textfile2 ...]
```

对于在命令行中命名的每个文本文件，**mysqlimport**去掉文件名的扩展名并使用结果来确定将导入文件内容的表名。例如，文件 patient.txt、patient.text和patient均将导入表patient。

mysqlimport支持下面的选项：

- ---help，-?

显示帮助消息并退出。

- --columns=*column_list*, -c *column_list*

该选项采用用逗号分隔的列名作为其值。列名的顺序指示如何匹配数据文件列和表列。

- --compress，-C

压缩在客户端和服务器之间发送的所有信息（如果二者均支持压缩）。

- ---debug[=*debug_options*]，-# [*debug_options*]

写调试日志。*debug_options*字符串通常是'd:t:o,*file_name*'。

- --delete，-D

导入文本文件前清空表。

- --fields-terminated-by=...，--fields-enclosed-by=...，--fields-optionally-enclosed-by=...，--fields-escaped-by=...，--lines-terminated-by=...

这些选项与LOAD DATA INFILE相应子句的含义相同。参见13.2.5节，“LOAD DATA INFILE语法”。

· --force，-f

忽视错误。例如，如果某个文本文件的表不存在，继续处理其它文件。不使用--force，如果表不存在则**mysqlimport**退出。

· --host=*host_name*，-h *host_name*

将数据导入给定主机上的MySQL服务器。默认主机是localhost。

· --ignore，-i

参见--replace选项的描述。

· --ignore-lines=*n*

忽视数据文件的前*n*行。

· --local，-L

从本地客户端读入输入文件。

· --lock-tables，-l

处理文本文件前锁定所有表以便写入。这样可以确保所有表在服务器上保持同步。

· --password[=*password*]，-p[*password*]

当连接服务器时使用的密码。如果使用短选项形式(-p)，选项和 密码之间不能有空格。如果在命令行中--password或-p选项后面没有 密码值，则提示输入一个密码。

· --port=*port_num*，-P port_*num*

用于连接的TCP/IP端口号。

· --protocol={TCP | SOCKET | PIPE | MEMORY}

使用的连接协议。

· --replace，-r

--replace和--ignore选项控制复制唯一键值已有记录的输入记录的处理。如果指定--replace，新行替换有相同的唯一键值的已有行。如果指定--ignore，复制已有的唯一键值的输入行被跳过。如果不指定这两个选项，当发现一个复制键值时会出现一个错误，并且忽视文本文件的剩余部分。

· --silent，-s

沉默模式。只有出现错误时才输出。

· --socket=*path*，-S *path*

当连接localhost时使用的套接字文件(为默认主机)。

· --user=*user_name*，-u *user_name*

当连接服务器时MySQL使用的用户名。

- --verbose，-v

冗长模式。打印出程序操作的详细信息。

- --version，-V

显示版本信息并退出。

这里是一个示例会话，显示如何使用**mysqlimport**：

```
shell> mysql -e 'CREATE TABLE imptest(id INT, n VARCHAR(30))' test
shell> ed
a
100     Max Sydow
101     Count Dracula
.
w imptest.txt
32
q
shell> 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
shell> mysqlimport --local test imptest.txt
test.imptest: Records: 2  Deleted: 0  Skipped: 0  Warnings: 0
shell> mysql -e 'SELECT * FROM imptest' test
+------+---------------+
| id   | n             |
+------+---------------+
|  100 | Max Sydow     |
|  101 | Count Dracula |
+------+---------------+
```

8.11. mysqlshow－显示数据库、表和列信息

mysqlshow客户可用来很快地查找存在哪些数据库，数据库中的表，表中的列或索引。

mysqlshow为一些SQL显示语句提供了一个命令行界面。相同的信息可以通过直接使用那些语句获得。例如，你可以从**mysql**客户程序发出它们。参见13.5.4节，“SHOW语法”。

象这样调用**mysqlshow**：

```
shell> mysqlshow[选项] [db_name [tbl_name [col_name]]]
```

- 如果没有给出数据库，显示所有匹配的数据库。

- 如果没有给出表，显示数据库中所有匹配的表。
- 如果没有给出列，显示表中所有匹配的列和列类型。

请注意，在MySQL的新版本中，你只能看到有部分权限的那些数据库、表或者列。

如果最后的参数包含shell或SQL通用字符 ('*'、'?'、'%'或者'_')，只显示那些名匹配通用字符的名字。如果一个数据库名包含下划线，应该使用一个反斜线（部分Unix shells需要双反斜线）对它们进行转义，以便得到一个正确的表或列的列表。'*'和'?'字符被转换为SQL的'%'和'_'通用字符。你试图用'_'显示一个表的列名时，这可能会造成一些混淆，因为在这种情况下 **mysqlshow**显示与模式相匹配的表名。这很容易通过在命令行添加一个'%'参量来解决。

mysqlshow支持下面的选项:

- ---help，-?

显示一个帮助消息并退出。

- --character-sets-dir=*path*

字符集安装目录。参见5.10.1节，"数据和排序用字符集"。

- --compress，-C

压缩所有的客户和服务器之间发送的信息（如果它们都支持压缩）。

- ---debug[=*debug_options*]，-# [*debug_options*]

写调试日志。*debug_options*字符串通常是'd:t:o,*file_name*'。

- --default-character-set=*charset*

使用*charset*为默认字符集。参见5.10.1节，"数据和排序用字符集"。

- --host=*host_name*，-h *host_name*

在给出的主机上连接 MySQL服务器。

- --keys, -k

显示表索引。

- --password[=*password*]，-p[*password*]

连接服务器时使用的密码。如果你使用短选项形式(-p)，不能在选项和 密码之间有一个空格。如果在命令行中，你忽略了--password或-p选项后面的 密码值，将提示你输入一个。

- --port=*port_num*, -P *port_num*

连接时使用的TCP/IP端口号。

- --protocol={TCP | SOCKET | PIPE | MEMORY}

连接时使用的协议。

- --show-table-type

显示一列来指示表类型，与SHOW FULL TABLES语句显示相同。

· --socket=*path*，-S *path*

当连接localhost时使用的套接字文件（它时默认主机）。

· --status, -i

显示关于每个表的额外信息。

· --user=*user_name*，-u *user_name*

连接服务器时使用的MySQL用户名。

· --verbose，-v

冗长模式。打印出程序操作的详细信息。该选项可以多次使用以便增加信息总量。

· --version，-V

显示版本信息并退出。

8.12. myisamlog：显示MyISAM日志文件内容

myisamlog处理MyISAM日志文件内容。

象这样调用**myisamlog**：

```
shell> myisamlog [options] [logfile-name [tbl_name] ...]
```

常用操作是更新（-u）。如果恢复完成（-r），所有写和可能的更新以及删除操作完成并且只统计错误。如果没有给出日志文件名，使用myisam.log作为日志文件名。如果在命令行中指定了表名，只更新那些指定的表。

myisamlog理解下面的选项：

· -?，-I

显示一个帮助消息并退出。

· -c *N*

只执行*N*个命令。

· -f *N*

指定打开文件的最大数目。

· -F *filepath/*

TODO

· -i

退出之前显示详细信息。

· -o *offset*

指定起始偏移量。

· -p *N*

从路径中移除*N*个组件。

· -r

恢复。

· -R *record-pos-file record-pos*

指定记录位置文件和记录位置。

· -u

更新。

· -v

Verbose模式。产生更多输出。该选项可以多次使用以便产生越来越多的输出。

· -w *write-file*

指定写文件。

· -V

显示版本信息。

8.13. perror：解释错误代码

对于大多数系统错误，除内部文本信息之外，MySQL还按下面的风格显示的系统错误代码：

message ... (errno: #)

message ... (Errcode: #)

通过检查系统文档或使用**perror**工具，可以检查错误代码的意义。

perror为系统错误代码或存储引擎（表处理）错误代码打印其描述信息。

象这样调用**perror**：

```
shell> perror [options] errorcode ...
```

例如：

```
shell> perror13 64
```

Error code 13: Permission denied

Error code 64: Machine is not on the network

注释：要想显示MySQL簇的错误代码，用--ndb选项调用**perror**：

```
shell> perror --ndb errorcode
```

请注意，系统错误信息的含义可能与操作系统有关。在不同的操作系统中错误代码的含义可能不同。

8.14. replace：字符串替换实用工具

replace实用工具可以及时更改文件或标准输入中的字符串。它首先使用有限状态机来匹配长的字符串。该工具可以用来交换字符串。例如，下面的命令交换给定文件file1和file2中的a和b：

```
shell> replace a b b a -- file1 file2 ...
```

使用--选项来指示字符串-替换列表的结束位置和开始的文件名。

在命令行中的任何文件名被及时修改，因此在转换之前，可能要对源文件进行备份。

如果在命令行中没有指定文件名，**replace**读取标准输入并写到标准输出中。在这种情况下，不需要--选项。

replace程序用于**msql2mysql**。参见25.9.1节，“msql2mysql：转换mSQL程序以用于MySQL”。

replace支持下面的选项：

- -?，-I

显示一条帮助消息并退出。

- *-# debug_options*

写调试日志。*debug_options*字符串通常是'd:t:o,*file_name*'。

- -s

沉默模式。打印出程序执行的少量信息。

- -v

冗长模式。打印出程序执行的大量信息。

- -V

显示版本信息并退出。

8.15. mysql_zap：杀死符合某一模式的进程

mysql_zap杀死匹配某一模式的进程。象这样调用它：

```
shell> mysql_zap [-signal] [-?Ift] pattern
```

如果进程的**ps**命令的输出行包含该模式，则认为其匹配。默认情况，**mysql_zap**对每个进程请求确认。响应y则杀死该进程，或者q退出 **mysql_zap**。对于其它响应，**mysql_zap**不试图杀死进程。

如果给出-*signal*选项，它指定发送到每个进程的信号名或信号编号。否则，**mysql_zap**首先尝试TERM（信号15）然后用KILL（信号9）。

mysql_zap理解下面的额外选项：

- ---help，-?，-I

显示一条帮助消息并退出。

- -f

强制模式。**mysql_zap** 视图杀死每个匹配的进程而不需要确认。

- -t

测试模式。显示每个进程的信息但不杀死它。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第9章：语言结构

目录

9.1. 文字值

9.1.1. 字符串
9.1.2. 数值
9.1.3. 十六进制值
9.1.4. 布尔值
9.1.5. 位字段值
9.1.6. NULL值

9.2. 数据库、表、索引、列和别名

9.2.1. 识别符限制条件
9.2.2. 识别符大小写敏感性

9.3. 用户变量
9.4. 系统变量

9.4.1. 结构式系统变量

9.5. 注释语法
9.6. MySQL中保留字的处理

本章讨论了使用MySQL编写SQL语句的下面元素时所使用的规则：

- 字符串和数字等文字值
- 识别符，例如表和列名
- 用户和系统变量
- 注释
- 保留字

9.1. 文字值

9.1.1. 字符串
9.1.2. 数值
9.1.3. 十六进制值
9.1.4. 布尔值
9.1.5. 位字段值
9.1.6. NULL值

该节描述了如何在MySQL中写文字值。包括字符串、数值、十六进制值、布尔值和NULL。本节还包括在MySQL中处理这些基本类型时会遇到的各种细微差别和"影印版"。

9.1.1. 字符串

字符串指用单引号('')或双引号('")引起来的字符序列。例如:

```
'a string'
"another string"
```

如果SQL服务器模式启用了NSI_QUOTES，可以只用单引号引用字符串。用双引号引用的字符串被解释为一个识别符。

字符串可以有一个可选字符集引介词和COLLATE子句:

```
[_charset_name]'string' [COLLATE collation_name]
```

例如:

```
SELECT _latin1'string';
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

关于这些字符串语法形式的详细信息，参见10.3.7节，"字符串文字字符集和校对"。

在字符串中，某些序列具有特殊含义。这些序列均用反斜线('\')开始，即所谓的转义字符。MySQL识别下面的转义序列:

\0	ASCII 0(NUL)字符。
\'	单引号('')。
\"	双引号('")。
\b	退格符。
\n	换行符。
\r	回车符。
\t	tab字符。
\Z	ASCII 26(控制（Ctrl）-Z)。该字符可以编码为'\Z'，以允许你解决在Windows中ASCII 26代表文件结尾这一问题。(如果你试图使用mysql *db_name* < *file_name*，ASCII 26会带来问题）。
\\	反斜线('\')字符。
\%	'%'字符。参见表后面的注解。
_	'_'字符。参见表后面的注解。

这些序列对大小写敏感。例如，'\b'解释为退格，但'\B'解释为'B'。

'\%'和'_'序列用于搜索可能会解释为通配符的模式匹配环境中的'%'和'_'文字实例。参见12.3.1节，"字符串比较函数"。请注意如果你在其它环境中使用'\%'或'_'，它们返回字符串'\%'和'_'，而不是'%'和'_'。

在其它转义序列中，反斜线被忽略。也就是说，转义字符解释为仿佛没有转义。

有几种方式可以在字符串中包括引号:

- 在字符串内用''引用的''可以写成''。
- 在字符串内用"引用的"可以写成"。
- 可以在引号前加转义字符('\')。
- 在字符串内用"引用的''不需要特殊处理，不需要用双字符或转义。同样，在字符串内用''引用的"也不需要特殊处理。

下面的SELECT语句显示了引用和转义如何工作:

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

mysql> SELECT 'disappearing\ backslash';
+------------------------+
| disappearing backslash |
+------------------------+
```

如果你想要在字符串列内插入二进制数据(例如BLOB)，必须通过转义序列表示下面的字符:

NUL	NUL字节(ASCII 0)。用'\0'表示该字符(反斜线后面跟一个ASCII'0'字符)。
\	反斜线(ASCII 92)。用'\\'表示该字符。
'	单引号(ASCII 39)。用'\'表示该字符。
"	双引号(ASCII 34)。用'\"'表示该字符。

当编写应用程序时，在包含这些特殊字符的字符串用于发送到MySQL服务器的SQL语句中的数据值之前，必须对它们正确进行转义。可以用两种方法来完成:

- 用转义特殊字符的函数处理字符串。例如，在C程序中，可以使用mysql_real_escape_string() C API函数来转义字符。参见25.2.3.52节，“mysql_real_escape_string()”。Perl DBI接口提供一个quote方法来将特殊字符转换为正确的转义序列。参见25.4节，“MySQL Perl API”。

- 显式转义特殊字符，许多MySQL API提供了占位符功能，允许你在查询字符串中插入特殊标记，然后当你发出查询时将数据值同它们绑定起来。在这种情况下，API关注转义值中的特殊字符。

9.1.2. 数值

整数用一系列阿拉伯数字表示。浮点数使用'.'作为十进制间隔符。两种类型的数值均可以在前面加一个'-'来表示负值。

合法整数的例子:

```
1221
0
-32
```

合法浮点数的例子:

```
294.42
-32032.6809e+10
148.00
```

整数可以用在浮点环境中; 它被解释为与浮点数等效。

9.1.3. 十六进制值

MySQL支持十六进制值。在数字上下文中, 十六进制数如同整数(64位精度)。在字符串上下文, 如同二进制字符串, 每对十六进制数字被转换为一个字符:

```
mysql> SELECT x'4D7953514C';
        -> 'MySQL'
mysql> SELECT 0xa+0;
        -> 10
mysql> SELECT 0x5061756c;
        -> 'Paul'
```

十六进制值的默认类型是字符串。如果想要确保该值作为数字处理, 可以使用CAST(...AS UNSIGNED):

```
mysql> SELECT 0x41, CAST(0x41 AS UNSIGNED);
        -> 'A', 65
```

0x语法基于ODBC。十六进制字符串通常用于ODBC以便为BLOB列提供值。x'hexstring'语法基于标准SQL。

可以用HEX()函数将一个字符串或数字转换为十六进制格式的字符串:

```
mysql> SELECT HEX('cat');
        -> '636174'
mysql> SELECT 0x636174;
        -> 'cat'
```

9.1.4. 布尔值

常量TRUE等于1, 常量FALSE等于0。常量名可以写成大写或小写。

```
mysql> SELECT TRUE, true, FALSE, false;
        -> 1, 1, 0, 0
```

9.1.5. 位字段值

可以使用b'*value*'符号写位字段值。*value*是一个用0和1写成的二进制值。

位字段符号可以方便指定分配给BIT列的值:

```
mysql> CREATE TABLE t (b BIT(8));
mysql> INSERT INTO t SET b = b'11111111';
mysql> INSERT INTO t SET b = b'1010';
+------+----------+----------+----------+
| b+0  | BIN(b+0) | OCT(b+0) | HEX(b+0) |
+------+----------+----------+----------+
|  255 | 11111111 | 377      | FF       |
|   10 | 1010     | 12       | A        |
+------+----------+----------+----------+
```

9.1.6. NULL值

NULL值表示"没有数据"。NULL可以写成大写或小写。

请注意NULL值不同于数字类型的0或字符串类型的空字符串。参见A.5.3节，"与NULL值有关的问题"。

对于用LOAD DATA INFILE或SELECT ...INTO OUTFILE执行的文本文件导入或导出操作，NULL用序列\N表示。参见13.2.5节，"LOAD DATA INFILE语法"。

9.2. 数据库、表、索引、列和别名

9.2.1. 识别符限制条件
9.2.2. 识别符大小写敏感性

数据库、表、索引、列和别名是识别符。该节描述了在MySQL中识别符的允许的语法。

下面的表描述了每类识别符的最大长度和允许的字符。

识别符	最大长度(字节)	允许的字符
数据库	64	目录名允许的任何字符，不包括'/'、'\'或者'。'
表	64	文件名允许的任何字符，不包括'/'、'\'或者'。'
列	64	所有字符
索引	64	所有字符
别名	255	所有字符

除了表内注明的限制，识别符不可以包含ASCII 0或值为255的字节。数据库、表和列名不应以空格结尾。在识别符中可以使用引号识别符，尽管应尽可能避免这样使用。

识别符用Unicode(UTF8)保存。在.frm文件中保存的表定义的识别符和在mysql数据库中的授权表保存的识别符也用Unicode(UTF8)保存。在MySQL 5.1中授权表(和其它表)的字符串列的大小等于字符个数；这说明(不象以前版本的MySQL)你可以在这些列保存的值中使用多字节字符而不需要降低字符个数。

识别符可以引起来也可以不引起来。如果识别符是一个保留字或包含特殊字符，无论何时使用，必须将它引起来。关于保留字的列表参见9.6节，“MySQL中保留字的处理”。特殊字符指那些当前字符集、'_'和'$'之外的文字数字字符集。

识别符的引用符是反勾号(`)：

```
mysql> SELECT * FROM `select` WHERE `select`.id > 100;
```

如果SQL服务器模式包括ANSI_QUOTES模式选项，还可以用双引号将识别符引起来：

```
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)
```

参见5.3.2节，“SQL服务器模式”。

如果你引用识别符，可以在识别符内包括识别符引用符。如果识别符内包括的字符与引用识别符的字符相同，则需要用双字符。下面的语句创建一个名为a`b包含列c"d的表：

```
mysql> CREATE TABLE `a``b` (`c"d` INT);
```

建议不要使用*X*e*X*模式的名，例如1e或2e2，因为类似1e+1的表达式比较模糊。根据上下文，它可以解释为表达式1e + 1或数字1e+1。

使用MD5产生表名时应仔细，因为它可能产生不合法的表名，如上所述。

9.2.1. 识别符限制条件

MySQL允许使用由单个识别符或多个识别符组成的名字。多部分名各组件之间应以句点('.')间隔开。多部分名的开头部分作为限定词，后面的识别符被解释。

在MySQL中可以引用下面形式的列：

列参考	含义
col_name	列*col_name*，查询中使用的表包含有此名字的列。
tbl_name.col_name	默认数据库中的表*tbl_name*的列*col_name*。
db_name.tbl_name.col_name	数据库*db_name*中的表*tbl_name*的列*col_name*。

如果多部分名的组件需要引用，应分别将它们引起来而不要将整个名引起来。例如，`my-tables`.`my-column` 有效，而`my-tables.my-column`无效。

不需要在语句中为列指定*tbl_name*或*db_name.tbl_name*前缀，除非列会很模糊。假定表t1和t2各包含一个列c，你使用SELECT语句在t1和t2中搜索c。在这种情况下，c很模糊，因为它在语句中使用的表内不唯一。你必须用表名t1.c或t2.c限定它，表示指哪个表。同样，要想用同一语句搜索数据库db1中的表t和数据库db2中的表t，你必须将那些表中的列指为db1.t.col_name和db2.t.col_name。

限定名中句点后面的字必须为一个识别符，因此不需要将它引起来，即使是一个保留字。

语法.*tbl_name*表示当前数据库中的*tbl_name*。该语法与ODBC兼容，因为某些ODBC程序在表名前面加前缀'.'字符。

9.2.2. 识别符大小写敏感性

在MySQL中，数据库对应数据目录中的目录。数据库中的每个表至少对应数据库目录中的一个文件(也可能是多个，取决于存储引擎)。因此，所使用操作系统的大小写敏感性决定了数据库名和表名的大小写敏感性。这说明在大多数Unix中数据库名和表名对大小写敏感，而在Windows中对大小写不敏感。一个显著的例外情况是Mac OS X，它基于Unix但使用默认文件系统类型(HFS+)，对大小写不敏感。然而，Mac OS X也支持UFS卷，该卷对大小写敏感，就像Unix一样。参见1.8.4节，“MySQL对标准SQL的扩展”。

注释：尽管在某些平台中数据库名和表名对大小写不敏感，不应在同一查询中使用不同的大小写来引用给定的数据库或表。下面的查询不会工作，因为它同时引用了表my_tables和as MY_tables：

```
mysql> SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

列、索引、存储子程序和触发器名在任何平台上对大小写不敏感，列的别名也不敏感。

默认情况，表别名在Unix中对大小写敏感，但在Windows或Mac OS X中对大小写不敏感。下面的查询在Unix中不会工作，因为它同时引用了别名a和A：

```
mysql> SELECT col_name FROM tbl_name AS a
    -> WHERE a.col_name = 1 OR A.col_name = 2;
```

然而，该查询在Windows中是可以的。要想避免出现差别，最好采用一致的转换，例如总是用小写创建并引用数据库名和表名。在大多数移植和使用中建议使用该转换。

在MySQL中如何在硬盘上保存和使用表名和数据库名由lower_case_tables_name系统变量确定，可以在启动**mysqld**时设置。lower_case_tables_name可以采用下面的任一值：

值	含义
0	使用CREATE TABLE或CREATE DATABASE语句指定的大写和小写在硬盘上保存表名和数据库名。名称比较对大小写敏感。在Unix系统中的默认设置即如此。请注意如果在大小写不敏感的文件系统上用--lower-case-table-names=0强制设为0，并且使用不同的大小写访问MyISAM表名，会导致索引破坏。
1	表名在硬盘上以小写保存，名称比较对大小写敏感。MySQL将所有表名转换为小写以便存储和查找。该行为也适合数据库名和表的别名。该值为Windows和Mac OS X系统中的默认值。
2	表名和数据库名在硬盘上使用CREATE TABLE或CREATE DATABASE语句指定的大小写进行保存，但MySQL将它们转换为小写以便查找。名称比较对大小写敏感。注释：只在对大小写不敏感的文件系统上适用! InnoDB表名以小写保存，例如lower_case_tables_name=1。

在Windows和Mac OS X中，lower_case_tables_name的 默认值是1。

如果只在一个平台上使用MySQL，通常不需要更改lower_case_tables_name变量。然而，如果你想要在对大小写敏感不同的文件系统的平台之间转移表，会遇到困难。例如，在Unix中，my_tables和MY_tables是两个不同的表，但在Windows中，这两个表名相同。要想避免由于数据库或表名的大小写造成的数据转移问题，可使用两个选项：

· 在任何系统中可以使用lower_case_tables_name=1。使用该选项的不利之处是当使用SHOW TABLES或SHOW DATABASES时，看不出名字原来是用大写还是小写。

· 在Unix中使用lower_case_tables_name=0，在Windows中使用lower_case_tables_name=2。这样了可以保留数据库名和表名的大小写。不利之处是必须确保在Windows中查询总是用正确大小写引用数据库名和表
Unix Unix

名。如果将查询转移到　　中，由于在　　中大小写很重要，如果大小写不正确，它们不工作。

例外：如果你正使用InnoDB表，在任何平台上均应将lower_case_tables_name设置为1，以强制将名转换为小写。

请注意在Unix中将lower_case_tables_name设置为1之前，重启**mysqld**之前，必须先将旧的数据库名和表名转换为小写。

9.3. 用户变量

可以先在用户变量中保存值然后在以后引用它；这样可以将值从一个语句传递到另一个语句。用户变量与连接有关。也就是说，一个客户端定义的变量不能被其它客户端看到或使用。当客户端退出时，该客户端连接的所有变量将自动释放。

用户变量的形式为@*var_name*，其中变量名*var_name*可以由当前字符集的文字数字字符、'.'、'_'和'$'组成。默认字符集是cp1252 (Latin1)。可以用**mysqld**的--default-character-set选项更改字符集。参见5.10.1节，"数据和排序用字符集"。用户变量名对大小写不敏感。

设置用户变量的一个途径是执行SET语句：

```
SET @var_name = expr [, @var_name = expr] ...
```

对于SET，可以使用=或:=作为分配符。分配给每个变量的*expr*可以为整数、实数、字符串或者NULL值。

也可以用语句代替SET来为用户变量分配一个值。在这种情况下，分配符必须为:=而不能用=，因为在非SET语句中=被视为一个比较 操作符：

```
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 |

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

用户变量可以用于表达式中。目前不包括明显需要文字值的上下文中，例如SELECT语句的LIMIT子句，或者LOAD DATA语句的IGNORE number LINES子句。

如果使用没有初始化的变量，其值是NULL。

如果用户变量分配了一个字符串值，其字符集和校对规则与该字符串的相同。用户变量的可压缩性（coercibility）是隐含的。(即为表列值的相同的可压缩性（coercibility）。

注释：在SELECT语句中，表达式发送到客户端后才进行计算。这说明在HAVING、GROUP BY或者ORDER BY子句中，不能使用包含SELECT列表中所设的变量的表达式。例如，下面的语句不能按期望工作：

```
mysql> SELECT (@aa:=id) AS a, (@aa+3) AS b 从tbl_name HAVING b=5;
```

HAVING子句中引用了SELECT列表中的表达式的别名，使用@aa。不能按期望工作：@aa不包含当前行的值，而是前面所选的行的id值。

一般原则是不要在语句的一个部分为用户变量分配一个值而在同一语句的其它部分使用该变量。可能会得到期

望的结果，但不能保证。

设置变量并在同一语句中使用它的另一个问题是变量的默认结果的类型取决于语句前面的变量类型。下面的例子说明了该点：

```
mysql> SET @a='test';

mysql> SELECT @a,(@a:=20) FROM tbl_name;
```

对于该 SELECT语句，MySQL向客户端报告第1列是一个字符串，并且将@a的所有访问转换为字符串，即使@a在第2行中设置为一个数字。执行完SELECT语句后，@a被视为下一语句的一个数字。

要想避免这种问题，要么不在同一个语句中设置并使用相同的变量，要么在使用前将变量设置为0、0.0或者"以定义其类型。

未分配的变量有一个值NULL，类型为字符串。

9.4. 系统变量

9.4.1. 结构式系统变量

MySQL可以访问许多系统和连接变量。当服务器运行时许多变量可以动态更改。这样通常允许你修改服务器操作而不需要停止并重启服务器。

mysqld服务器维护两种变量。全局变量影响服务器整体操作。会话变量影响具体客户端连接的操作。

当服务器启动时，它将所有全局变量初始化为默认值。这些默认值可以在选项文件中或在命令行中指定的选项进行更改。服务器启动后，通过连接服务器并执行SET GLOBAL *var_name*语句，可以动态更改这些全局变量。要想更改全局变量，必须具有SUPER权限。

服务器还为每个连接的客户端维护一系列会话变量。在连接时使用相应全局变量的当前值对客户端的会话变量进行初始化。对于动态会话变量，客户端可以通过SET SESSION *var_name*语句更改它们。设置会话变量不需要特殊权限，但客户端只能更改自己的会话变量，而不能更改其它客户端的会话变量。

对于全局变量的更改可以被访问该全局变量的任何客户端看见。然而，它只影响更改后连接的客户的从该全局变量初始化的相应会话变量。不影响目前已经连接的客户端的会话变量(即使客户端执行SET GLOBAL语句也不影响)。

可以使用几种语法形式来设置或检索全局或会话变量。下面的例子使用了sort_buffer_sizeas作为示例变量名。

要想设置一个GLOBAL变量的值，使用下面的语法：

```
mysql> SET GLOBAL sort_buffer_size=value;

mysql> SET @@global.sort_buffer_size=value;
```

要想设置一个SESSION变量的值，使用下面的语法：

```
mysql> SET SESSION sort_buffer_size=value;

mysql> SET @@session.sort_buffer_size=value;

mysql> SET sort_buffer_size=value;
```

LOCAL是SESSION的同义词。

如果设置变量时不指定GLOBAL、SESSION或者LOCAL，默认使用SESSION。参见13.5.3节，“SET语法”。

要想检索一个GLOBAL变量的值，使用下面的语法：

```
mysql> SELECT @@global.sort_buffer_size;

mysql> SHOW GLOBAL VARIABLES like 'sort_buffer_size';
```

要想检索一个SESSION变量的值，使用下面的语法：

```
mysql> SELECT @@sort_buffer_size;

mysql> SELECT @@session.sort_buffer_size;

mysql> SHOW SESSION VARIABLES like 'sort_buffer_size';
```

这里，LOCAL也是SESSION的同义词。

当你用SELECT @@*var_name*搜索一个变量时(也就是说，不指定global.、session.或者local.)，MySQL返回SESSION值（如果存在），否则返回GLOBAL值。

对于SHOW VARIABLES，如果不指定GLOBAL、SESSION或者LOCAL，MySQL返回SESSION值。

当设置GLOBAL变量需要GLOBAL关键字但检索时不需要它们的原因是防止将来出现问题。如果我们移除一个与某个GLOBAL变量具有相同名字的SESSION变量，具有SUPER权限的客户可能会意外地更改GLOBAL变量而不是它自己的连接的SESSION变量。如果我们添加一个与某个GLOBAL变量具有相同名字的SESSION变量，想更改GLOBAL变量的客户可能会发现只有自己的SESSION变量被更改了。

关于系统启动选项和系统变量的详细信息参见5.3.1节，“**mysqld**命令行选项”和5.3.3节，“服务器系统变量”。在5.3.3.1节，“动态系统变量”中列出了可以在运行时设置的变量。

9.4.1. 结构式系统变量

结构式变量在两个方面不同于常规系统变量：

- 其值是一个含组件的结构，可以指定服务器参数，一般紧密相关。
- 可能是给定类型的结构式变量的几个实例。每个实例有一个不同的名，指向服务器维护的不同的资源。

MySQL 5.1支持结构式变量类型，可以指定监控键值缓存操作的参数。键值缓存结构式变量有以下组件：

- key_buffer_size
- key_cache_block_size
- key_cache_division_limit
- key_cache_age_threshold

该节描述了引用结构式变量的语法。在语法的例子中使用了键值缓存变量，但关于键值缓存如何操作的具体详情在其它章节中，如 7.4.6节，“MyISAM键高速缓冲”。

要引用结构式变量实例的组件，可以使用instance_name.component_name格式的复合名。例如：

```
hot_cache.key_buffer_size

hot_cache.key_cache_block_size

cold_cache.key_cache_block_size
```

对于每个结构式系统变量，总是预定义名为default的一个实例。如果不使用任何实例名引用结构式变量的一个

组件，default实例被使用。这样，default.key_buffer_size和key_buffer_sizeboth均指向同一系统变量。

结构式变量实例和组件的命名规则为:

- 对于给定类型的结构式变量，每个实例必须有一个在该类变量中唯一的一个名。但是，在不同类型的结构式变量中实例名不需要唯一。例如，每个结构式变量有一个实例default，因此在不同的变量类型中default不是唯一的。

- 每个结构式变量类型的组件名在所有系统变量名中必须是唯一的。如果不是这样(也就是说，如果两个不同类型的结构式变量能够共享组件成员名)，将不清楚使用哪个默认结构式变量来用作没有使用实例名限定的成员名。

- 如果结构式变量实例名作为未引起来的识别符不合法，将它用反勾号引起来作为识别符。例如，hot-cache不合法，但`hot-cache` 合法。

- global、session和local不是合法实例名。这样可以避免与引用非结构式系统变量的符号的冲突，例如@@global.*var_name*。

目前，这前两个规则不可能违背，因为唯一的结构式变量类型是键值缓存。在将来创建其它类型的结构式变量，这些规则将有重要的意义。

一个例外是，可以在可能出现简单变量名的上下文中使用复合名引用结构式变量组件。例如，可以使用一个命令行选项未某个结构式变量分配一个值:

```
shell> mysqld --hot_cache.key_buffer_size=64K
```

在选项文件中，使用:

```
[mysqld]
hot_cache.key_buffer_size=64K
```

如果用该选项启动服务器，除了默认大小为8MB的默认键值缓存，还创建一个名为hot_cache的键值缓存，大小为64KB。

假定你这样启动服务器:

```
shell> mysqld --key_buffer_size=256K \
         --extra_cache.key_buffer_size=128K \
         --extra_cache.key_cache_block_size=2048
```

在这种情况下，服务器将默认键值缓存的大小设定为256KB。(也可以写成--default.key_buffer_size=256K）。并且，服务器创建一个名为extra_cache的第2个键值缓存，大小为128KB，缓存表索引块的块缓存区的大小设置为2048字节。

在下面的例子中，用3个不同的键值缓存（大小比例为3:1:1）启动服务器:

```
shell> mysqld --key_buffer_size=6M \
         --hot_cache.key_buffer_size=2M \
         --cold_cache.key_buffer_size=2M
```

也可以在运行时设置和检索结构式变量值。例如，要想将名为hot_cache的一个键值缓存的大小设置为10MB，使用下面任何一个语句:

```
mysql> SET GLOBAL hot_cache.key_buffer_size = 10*1024*1024;
```

```
mysql> SET @@global.hot_cache.key_buffer_size = 10*1024*1024;
```

要想检索缓存大小，执行：

```
mysql> SELECT @@global.hot_cache.key_buffer_size;
```

但是，下面的语句不工作。变量不解释为一个复合名，而是解释为LIKE模式匹配操作的简单字符串：

```
mysql> SHOW GLOBAL VARIABLES LIKE 'hot_cache.key_buffer_size';
```

这是一个在可能出现简单的变量名时使用结构式变量名的例子。

9.5. 注释语法

MySQL服务器支持3种注释风格：

- 从'#'字符从行尾。
- 从'-- '序列到行尾。请注意'-- '(双破折号)注释风格要求第2个破折号后面至少跟一个空格符(例如空格、tab、换行符等等)。该语法与标准SQL注释语法稍有不同，后者将在1.8.5.7, "'--'作为注释起始标记"中讨论。
- 从/*序列到后面的*/序列。结束序列不一定在同一行中，因此该语法允许注释跨越多行。

下面的例子显示了3种风格的注释：

```
mysql> SELECT 1+1;     # This comment continues to the end of line
mysql> SELECT 1+1;     -- This comment continues to the end of line
mysql> SELECT 1 /* this is an in-line comment */ + 1;
mysql> SELECT 1+
/*
this is a
multiple-line comment
*/
1;
```

上述的注释语法适用于**mysqld**服务器如何分析SQL语句。发送到服务器之前，**mysql**客户程序也执行部分语句解析。(例如，它通过解析来确定在多语句行中的语句边界）。

在MySQL 5.1中，**mysql**解析/* ...*/注释的唯一局限性是结合该风格的注释定界符使用的叹号标记了有条件执行的SQL语句部分。适用于交互式运行**mysql**和将命令放入一个文件中，并以批处理模式使用**mysql**来处理**mysql <** *file_name*的文件。详细信息和例子参见1.8.4节，"MySQL对标准SQL的扩展"。

9.6. MySQL中保留字的处理

尝试使用一个识别符，例如使用嵌入式MySQL数据类型或函数名作为表名或列名，例如TIMESTAMP或GROUP，会造成一个常见问题。允许你这样操作(例如，ABS可以作为一个列名)。但是，默认情况下，在数调用中在函数名和后面的'('字符之间不允许有空格。该要求使函数调用与列名引用不同。

该行为的不利结果是在某些上下文中省略一个空格会使识别符解释为函数名。例如，该语句合法：

```
mysql> CREATE TABLE abs (val INT);
```

但省略abs后面的空格会造成语法错误，因为省略后该语句好像要调用ABS()函数：

```
mysql> CREATE TABLE abs(val INT);
```

如果SQL服务器模式包括IGNORE_SPACE模式值，服务器允许函数调用时在函数名和后面的'('字符之间有空格。这样使函数名被视为保留字。结果是，与函数名相同的识别符必须按照9.2节，"数据库、表、索引、列和别名"中所描述的引起来。SQL服务器模式按照5.3.2节，"SQL服务器模式"中所描述的进行控制。

限定名中句点后面的字必须为一个识别符，因此不需要将它引起来，即使它是一个保留字。

在MySQL中，下表中的字显式被保留。其中大多数字进制被标准SQL用作列名和/或表名(例如，GROUP)。少数被保留了，因为MySQL需要它们，(目前)使用**yacc**解析程序。保留字被引起来后可以用作识别符。

ADD	ALL	ALTER
ANALYZE	AND	AS
ASC	ASENSITIVE	BEFORE
BETWEEN	BIGINT	BINARY
BLOB	BOTH	BY
CALL	CASCADE	CASE
CHANGE	CHAR	CHARACTER
CHECK	COLLATE	COLUMN
CONDITION	CONNECTION	CONSTRAINT
CONTINUE	CONVERT	CREATE
CROSS	CURRENT_DATE	CURRENT_TIME
CURRENT_TIMESTAMP	CURRENT_USER	CURSOR
DATABASE	DATABASES	DAY_HOUR
DAY_MICROSECOND	DAY_MINUTE	DAY_SECOND
DEC	DECIMAL	DECLARE
DEFAULT	DELAYED	DELETE
DESC	DESCRIBE	DETERMINISTIC
DISTINCT	DISTINCTROW	DIV
DOUBLE	DROP	DUAL
EACH	ELSE	ELSEIF
ENCLOSED	ESCAPED	EXISTS
EXIT	EXPLAIN	FALSE
FETCH	FLOAT	FLOAT4
FLOAT8	FOR	FORCE
FOREIGN	FROM	FULLTEXT
GOTO	GRANT	GROUP
HAVING	HIGH_PRIORITY	HOUR_MICROSECOND
HOUR_MINUTE	HOUR_SECOND	IF
IGNORE	IN	INDEX
INFILE	INNER	INOUT

INSENSITIVE	INSERT	INT
INT1	INT2	INT3
INT4	INT8	INTEGER
INTERVAL	INTO	IS
ITERATE	JOIN	KEY
KEYS	KILL	LABEL
LEADING	LEAVE	LEFT
LIKE	LIMIT	LINEAR
LINES	LOAD	LOCALTIME
LOCALTIMESTAMP	LOCK	LONG
LONGBLOB	LONGTEXT	LOOP
LOW_PRIORITY	MATCH	MEDIUMBLOB
MEDIUMINT	MEDIUMTEXT	MIDDLEINT
MINUTE_MICROSECOND	MINUTE_SECOND	MOD
MODIFIES	NATURAL	NOT
NO_WRITE_TO_BINLOG	NULL	NUMERIC
ON	OPTIMIZE	OPTION
OPTIONALLY	OR	ORDER
OUT	OUTER	OUTFILE
PRECISION	PRIMARY	PROCEDURE
PURGE	RAID0	RANGE
READ	READS	REAL
REFERENCES	REGEXP	RELEASE
RENAME	REPEAT	REPLACE
REQUIRE	RESTRICT	RETURN
REVOKE	RIGHT	RLIKE
SCHEMA	SCHEMAS	SECOND_MICROSECOND
SELECT	SENSITIVE	SEPARATOR
SET	SHOW	SMALLINT
SPATIAL	SPECIFIC	SQL
SQLEXCEPTION	SQLSTATE	SQLWARNING
SQL_BIG_RESULT	SQL_CALC_FOUND_ROWS	SQL_SMALL_RESULT
SSL	STARTING	STRAIGHT_JOIN
TABLE	TERMINATED	THEN
TINYBLOB	TINYINT	TINYTEXT
TO	TRAILING	TRIGGER
TRUE	UNDO	UNION
UNIQUE	UNLOCK	UNSIGNED
UPDATE	USAGE	USE
USING	UTC_DATE	UTC_TIME
UTC_TIMESTAMP	VALUES	VARBINARY
VARCHAR	VARCHARACTER	VARYING

WHEN	WHERE	WHILE
WITH	WRITE	X509
XOR	YEAR_MONTH	ZEROFILL

MySQL允许部分关键字用做未引起来的识别符，因为许多人以前曾使用过它们。下面列出了一些例子:

- ACTION
- BIT
- DATE
- ENUM
- NO
- TEXT
- TIME
- TIMESTAMP

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com.。原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第10章：字符集支持

目录

10.1. 常规字符集和校对
10.2. MySQL中的字符集和校对
10.3. 确定默认字符集和校对

- 10.3.1. 服务器字符集和校对
- 10.3.2. 数据库字符集和校对
- 10.3.3. 表字符集和校对
- 10.3.4. 列字符集和校对
- 10.3.5. 字符集和校对分配示例
- 10.3.6. 连接字符集和校对
- 10.3.7. 字符串文字字符集和校对
- 10.3.8. 在SQL语句中使用COLLATE
- 10.3.9. COLLATE子句优先
- 10.3.10. BINARY操作符
- 10.3.11. 校对确定较为复杂的一些特殊情况
- 10.3.12. 校对必须适合字符集
- 10.3.13. 校对效果的示例

10.4. 字符集支持影响到的操作

- 10.4.1. 结果字符串
- 10.4.2. CONVERT()
- 10.4.3. CAST()
- 10.4.4. SHOW语句

10.5. Unicode支持
10.6. 用于元数据的UTF8
10.7. 与其它DBMS的兼容性
10.8. 新字符集配置文件格式
10.9. 国家特有字符集
10.10. MySQL支持的字符集和校对

- 10.10.1. Unicode字符集
- 10.10.2. 西欧字符集
- 10.10.3. 中欧字符集
- 10.10.4. 南欧与中东字符集
- 10.10.5. 波罗的海字符集
- 10.10.6. 西里尔字符集
- 10.10.7. 亚洲字符集

本章讨论以下主题：

- 什么是字符集和校对规则？
- 多级默认系统
- 字符集语法
- 相关函数和运算
- Unicode支持

- 每个字符集和校对规则的含义

MySQL5.1中的字符集支持包括在MyISAM、MEMORY和InnoDB存储引擎中。

10.1. 常规字符集和校对

字符集是一套符号和编码。校对规则是在字符集内用于比较字符的一套规则。让我们使用一个假想字符集的例子来区别清楚。

假设我们有一个字母表使用了四个字母：`A'、`B'、`a'、`b'。我们为每个字母赋予一个数值：`A'=0，`B'= 1，`a'= 2，`b'= 3。字母`A'是一个符号，数字0是`A'的编码，这四个字母和它们的编码组合在一起是一个字符集。

假设我们希望比较两个字符串的值：`A'和`B'。比较的最简单的方法是查找编码：`A'为0，`B'为1。因为0 小于1，我们可以说`A'小于`B'。我们做的仅仅是在我们的字符集上应用了一个校对规则。校对规则是一套规则（在这种情况下仅仅是一套规则）："对编码进行比较。"我们称这种全部可能的规则中的最简单的 校对规则为一个*binary*（二元）校对规则。

但是，如果我们希望小写字母和大写字母是等价的，应该怎样？那么，我们将至少有两个规则：（1）把小写字母`a'和`b'视为与`A'和`B'等价；（2）然后比较编码。我们称这是一个大小写不敏感的 校对规则。比二元校对规则复杂一些。

在实际生活中，大多数字符集有许多字符：不仅仅是`A'和`B'，而是整个字母表，有时候有许多种字母表，或者一个东方的使用上千个字符的书写系统，还有许多特殊符号和标点符号。并且在实际生活中，大多数 校对规则有许多个规则：不仅仅是大小写不敏感，还包括重音符不敏感（"重音符" 是附属于一个字母的符号，象德语的`Ö'符号）和多字节映射（例如，作为规则`Ö'=`OE'就是两个德语 校对规则的一种）。

MySQL5.1能够做这些事情：

- 使用多种字符集来存储字符串
- 使用多种校对规则来比较字符串
- 在同一台服务器、同一个数据库或甚至在同一个表中使用不同字符集或校对规则来混合字符串
- 允许定义任何级别的字符集和校对规则

在这些方面，MySQL5.1不仅比MySQL4.1以前的版本灵活得多，而且比其它大多数数据库管理系统超前许多。但是，为了有效地使用这些功能，你需要了解哪些字符集和 校对规则是可用的，怎样改变默认值，以及它们怎样影响字符操作符和字符串函数的行为。

10.2. MySQL中的字符集和校对

MySQL服务器能够支持多种字符集。可以使用SHOW CHARACTER SET语句列出可用的字符集：

```
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   | cp1252     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 |
| 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 |
...
```

（完整列表参见10.10节，“MySQL支持的字符集和校对”。）

任何一个给定的字符集至少有一个校对规则。它可能有几个校对规则。

要想列出一个字符集的校对规则，使用SHOW COLLATION语句。例如，要想查看latin1（“西欧ISO-8859-1”）字符集的 校对规则，使用下面的语句查找那些名字以latin1开头的 校对规则：

```
mysql> SHOW COLLATION LIKE 'latin1%';
+----------------------+---------+----+---------+----------+---------+
| Collation            | Charset | Id | Default | Compiled | Sortlen |
+----------------------+---------+----+---------+----------+---------+
| latin1_german1_ci    | latin1  |  5 |         |          |       0 |
| latin1_swedish_ci    | latin1  |  8 | Yes     | Yes      |       1 |
| latin1_danish_ci     | latin1  | 15 |         |          |       0 |
| latin1_german2_ci    | latin1  | 31 |         | Yes      |       2 |
| latin1_bin           | latin1  | 47 |         | Yes      |       1 |
| latin1_general_ci    | latin1  | 48 |         |          |       0 |
| latin1_general_cs    | latin1  | 49 |         |          |       0 |
| latin1_spanish_ci    | latin1  | 94 |         |          |       0 |
+----------------------+---------+----+---------+----------+---------+
```

latin1校对规则有下面的含义：

校对规则	含义
latin1_german1_ci	德国DIN-1
latin1_swedish_ci	瑞典/芬兰
latin1_danish_ci	丹麦/挪威
latin1_german2_ci	德国 DIN-2
latin1_bin	符合latin1编码的二进制
latin1_general_ci	多种语言(西欧)
latin1_general_cs	多种语言(西欧ISO),大小写敏感
latin1_spanish_ci	现代西班牙

校对规则一般有这些特征：

- 两个不同的字符集不能有相同的校对规则。
- 每个字符集有一个默认校对规则。例如，latin1默认校对规则是latin1_swedish_ci。
- 存在校对规则命名约定：它们以其相关的字符集名开始，通常包括一个语言名，并且以_ci（大小写不敏感）、_cs（大小写敏感）或_bin（二元）结束。

10.3. 确定默认字符集和校对

10.3.1. 服务器字符集和校对
10.3.2. 数据库字符集和校对
10.3.3. 表字符集和校对
10.3.4. 列字符集和校对
10.3.5. 字符集和校对分配示例
10.3.6. 连接字符集和校对
10.3.7. 字符串文字字符集和校对
10.3.8. 在SQL语句中使用COLLATE
10.3.9. COLLATE子句优先
10.3.10. BINARY操作符
10.3.11. 校对确定较为复杂的一些特殊情况
10.3.12. 校对必须适合字符集
10.3.13. 校对效果的示例

字符集和校对规则有4个级别的默认设置：服务器级、数据库级、表级和连接级。以下描述可能显得复杂，但是在实际应用中可以发现使用多种级别会使结果自然而明显。

10.3.1. 服务器字符集和校对

MySQL服务器有一个服务器字符集和一个服务器校对规则，它们均不能设置为空。

MySQL按照如下方法确定服务器字符集和服务器校对规则：

- 当服务器启动时根据有效的选项设置
- 根据运行时的设定值

在服务器级别，确定方法很简单。当启动**mysqld**时，根据使用的初始选项设置来确定服务器字符集和 校对规则。可以使用--default-character-set设置字符集，并且可以在字符集后面为 校对规则添加--default-collation。如果没有指定一个字符集，那就与--default-character-set=latin1相同。如果你仅指定了一个字符集（例如，latin1），但是没有指定一个 校对规则，那就与--default-charset=latin1 --default-collation=latin1_swedish_ci相同，因为latin1_swedish_ci是latin1的默认校对规则。因此，以下三个命令有相同的效果：

shell> **mysqld**

shell> **mysqld --default-character-set=latin1**

shell> **mysqld --default-character-set=latin1 **

--default-collation=latin1_swedish_ci

更改设定值的一个方法是通过重新编译。如果希望在从源程序构建时更改默认服务器字符集和校对规则，使用：--with-charset和--with-collation作为**configure**的参量。例如：

```
shell> ./configure --with-charset=latin1
```

或者：

shell> **./configure --with-charset=latin1 **

--with-collation=latin1_german1_ci

mysqld和**configure**都验证字符集/校对规则组合是否有效。如果无效，每个程序都显示一个错误信息，然后终止。

当前的服务器字符集和校对规则可以用作character_set_server和collation_server系统变量的值。在运行时能够改变这些变量的值。

10.3.2. 数据库字符集和校对

每一个数据库有一个数据库字符集和一个数据库校对规则，它不能够为空。CREATE DATABASE和ALTER DATABASE语句有一个可选的子句来指定数据库字符集和校对规则：

```
CREATE DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]

ALTER DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]
```

例如：

```
CREATE DATABASE db_name
    DEFAULT CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

MySQL这样选择数据库字符集和数据库校对规则：

- 如果指定了CHARACTER SET X和COLLATE Y，那么采用字符集*X*和校对规则*Y*。
- 如果指定了CHARACTER SET X而没有指定COLLATE Y，那么采用CHARACTER SET X和CHARACTER SET X的默认校对规则。
- 否则，采用服务器字符集和服务器校对规则。

MySQL的CREATE DATABASE ... DEFAULT CHARACTER SET ...语法与标准SQL的CREATE SCHEMA ... CHARACTER SET ...语法类似。因此，可以在同一个MySQL服务器上创建使用不同字符集和 校对规则的数据库。

如果在CREATE TABLE语句中没有指定表字符集和校对规则，则使用数据库字符集和校对规则作为默认值。它们没有其它目的。

默认数据库的字符集和校对规则可以用作character_set_database和 collation_database系统变量。无论何时默认数据库更改了，服务器都设置这两个变量的值。如果没有 默认数据库，这两个变量与相应的服务器级别的变量（character_set_server和collation_server）具有相同的值。

10.3.3. 表字符集和校对

每一个表有一个表字符集和一个校对规则，它不能为空。为指定表字符集和校对规则，CREATE TABLE 和ALTER TABLE语句有一个可选的子句：

```
CREATE TABLE tbl_name (column_list)
    [DEFAULT CHARACTER SET charset_name [COLLATE collation_name]]

ALTER TABLE tbl_name
    [DEFAULT CHARACTER SET charset_name] [COLLATE collation_name]
```

例如：

```
CREATE TABLE t1 ( ... )
    DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

MySQL按照下面的方式选择表字符集和 校对规则：

- 如果指定了CHARACTER SET X和COLLATE Y，那么采用CHARACTER SET X和COLLATE Y。
- 如果指定了CHARACTER SET X而没有指定COLLATE Y，那么采用CHARACTER SET X和CHARACTER SET X的默认校对规则。
- 否则，采用服务器字符集和服务器校对规则。

如果在列定义中没有指定列字符集和校对规则，则默认使用表字符集和校对规则。表字符集和校对规则是MySQL的扩展;在标准SQL中没有。

10.3.4. 列字符集和校对

每一个"字符"列（即，CHAR、VARCHAR或TEXT类型的列）有一个列字符集和一个列 校对规则，它不能为空。列定义语法有一个可选子句来指定列字符集和校对规则：

```
col_name {CHAR | VARCHAR | TEXT} (col_length)
    [CHARACTER SET charset_name [COLLATE collation_name]]
```

例如：

```
CREATE TABLE Table1
(
   column1 VARCHAR(5) CHARACTER SET latin1 COLLATE latin1_german1_ci
);
```

MySQL按照下面的方式选择列字符集和校对规则：

- 如果指定了CHARACTER SET X和COLLATE Y，那么采用CHARACTER SET X和COLLATE Y。
- 如果指定了CHARACTER SET X而没有指定COLLATE Y，那么采用CHARACTER SET X和CHARACTER SET X的默认校对规则。
- 否则，采用表字符集和服务器校对规则。

CHARACTER SET和COLLATE子句是标准的SQL。

10.3.5. 字符集和校对分配示例

以下例子显示了MySQL怎样确定默认字符集和校对规则。

示例**1**：表和列定义

```
CREATE TABLE t1
(
    c1 CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci
) DEFAULT CHARACTER SET latin2 COLLATE latin2_bin;
```

在这里我们有一个列使用latin1字符集和latin1_german1_ci校对规则。是显式的定义，因此简单明了。需要注意的是，在一个latin2表中存储一个latin1列不会存在问题。

示例**2**：表和列定义

```
CREATE TABLE t1
(
    c1 CHAR(10) CHARACTER SET latin1
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

这次我们有一个列使用latin1字符集和一个默认校对规则。尽管它显得自然，默认校对规则却不是表级。相反，因为latin1的默认校对规则总是latin1_swedish_ci，列c1有一个校对规

则latin1_swedish_ci（而不是latin1_danish_ci）。

示例**3**：表和列定义

```
CREATE TABLE t1
(
    c1 CHAR(10)
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

我们有一个列使用一个默认字符集和一个默认校对规则。在这种情况下，MySQL查找表级别来确定列字符集和 校对规则。因此，列c1的字符集是latin1，它的 校对规则是latin1_danish_ci。

示例**4**：数据库、表和列定义

```
CREATE DATABASE d1
    DEFAULT CHARACTER SET latin2 COLLATE latin2_czech_ci;
USE d1;
CREATE TABLE t1
(
    c1 CHAR(10)
);
```

我们创建了一个没有指定字符集和校对规则的列。我们也没有指定表级字符集和校对规则。在这种情况下，MySQL查找数据库级的相关设置。（数据库的设置变为表的设置，其后变为列的设置。）因此，列c1的字符集为是latin2，它的 校对规则是latin2_czech_ci。

10.3.6. 连接字符集和校对

一些字符集和校对规则系统变量与客户端和服务器的交互有关。在前面的章节中已经提到过部分内容：

- 服务器字符集和校对规则可以用作character_set_server和collation_server变量的值。
- 默认数据库的字符集和校对规则可以用作character_set_database和collation_database变量的值。

在客户端和服务器的连接处理中也涉及了字符集和校对规则变量。每一个客户端有一个连接相关的字符集和校对规则变量。

考虑什么是一个"连接"：它是连接服务器时所作的事情。客户端发送SQL语句，例如查询，通过连接发送到服务器。服务器通过连接发送响应给客户端，例如结果集。对于客户端连接，这样会导致一些关于连接的字符集和 校对规则的问题，这些问题均能够通过系统变量来解决：

- 当查询离开客户端后，在查询中使用哪种字符集？

服务器使用character_set_client变量作为客户端发送的查询中使用的字符集。

- 服务器接收到查询后应该转换为哪种字符集？

转换时，服务器使用character_set_connection和collation_connection系统变量。它将客户端发送的查询从character_set_client系统变量转换到character_set_connection（除非字符串文字具有象_latin1或_utf8的引介词）。collation_connection对比较文字字符串是重要的。对于列值的字符串比较，它不重要，因为列具有更高的 校对规则优先级。

- 服务器发送结果集或返回错误信息到客户端之前应该转换为哪种字符集？

character_set_results变量指示服务器返回查询结果到客户端使用的字符集。包括结果数据，例如列值和结果元数据（如列名）。

你能够调整这些变量的设置，或可以依赖默认值（这样，你可以跳过本章）。

有两个语句影响连接字符集：

```
SET NAMES 'charset_name'
```

```
SET CHARACTER SET charset_name
```

SET NAMES显示客户端发送的SQL语句中使用什么字符集。因此，SET NAMES 'cp1251'语句告诉服务器"将来从这个客户端传来的信息采用字符集cp1251"。它还为服务器发送回客户端的结果指定了字符集。（例如，如果你使用一个SELECT语句，它表示列值使用了什么字符集。）

SET NAMES '*x*'语句与这三个语句等价：

```
mysql> SET character_set_client = x;
mysql> SET character_set_results = x;
mysql> SET character_set_connection = x;
```

将x设置为character_set_connection也就设置了collation_connection是x的默认校对规则。

SET CHARACTER SET语句是类似的，但是为 默认数据库设置连接字符集和校对规则。SET CHARACTER SET x语句与这三个语句等价：

```
mysql> SET character_set_client = x;
mysql> SET character_set_results = x;
mysql> SET collation_connection = @@collation_database;
```

当一个客户端连接时，它向服务器发送希望使用的字符集名称。服务器为那个字符集设置character_set_client、character_set_results和 character_set_connection变量。（实际上，服务器为使用该字符集执行一个SET NAMES操作。）

对于**mysql**客户端，如果你希望使用与默认字符集不同的字符集，不需要每次启动时执行SET NAMES语句。可以在**mysql**语句行中或者选项文件中添加一个--default-character-set选项设置。例如，你每次运行**mysql**时，以下的选项文件设置把三个字符集变量修改为koi8r：

[mysql]

default-character-set=koi8r

例如：假设column1定义为CHAR(5) CHARACTER SET latin2。如果没有设定SET NAMES或SET CHARACTER SET，那么对于SELECT column1 FROM t，当连接后，服务器使用客户端指定的字符集返回列column1的所有值。另一方面，如果你设定SET NAMES 'latin1'或SET CHARACTER SET latin1，那么发送结果之前，服务器转换latin2值到latin1。转换可能会丢失那些不属于两种字符集的字符。

如果不希望服务器执行任何转换，设置character_set_results为NULL：

```
mysql> SET character_set_results = NULL;
```

10.3.7. 字符串文字字符集和校对

每一字符串字符文字有一个字符集和一个校对规则，它不能为空。

一个字符串文字可能有一个可选的字符集引介词和COLLATE子句：

```
[_charset_name]'string' [COLLATE collation_name]
```

例如：

```
SELECT 'string';
SELECT _latin1'string';
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

对于简单的语句SELECT '*string*'，字符串使用由character_set_connection和collation_connection系统变量定义的字符集和 校对规则。

*_charset_name*表达式正式称做一个引介词。它告诉解析程序，"后面将要出现的字符串使用字符集X。"因为以前人们对此感到困惑，我们强调引介词不导致任何转换；它仅是一个符号，不改变字符串的值。引介词在标准十六进制字母和数字十六进制符号（x'*literal*'和 0x*nnnn*）中是合法的，以及？（当在一个编程语言接口中使用预处理的语句时进行参数替换）。

例如:

```
SELECT _latin1 x'AABBCC';
SELECT _latin1 0xAABBCC;
SELECT _latin1 ?;
```

MySQL这样确定一个文字字符集和校对规则:

- 如果指定了CHARACTER SET X和COLLATE Y，那么使用CHARACTER SET X和COLLATE Y。
- 如果指定了CHARACTER SET X而没有指定COLLATE Y，那么使用CHARACTER SET X和CHARACTER SET X的默认校对规则。
- 否则，使用通过character_set_connection 和 collation_connection系统变量给出的字符集和 校对规则。

例如:

- 使用latin1字符集和latin1_german1_ci校对规则的字符串:
-
```
SELECT _latin1'Müller' COLLATE latin1_german1_ci;
```
- 使用latin1字符集和其默认校对规则的字符串（即，latin1_swedish_ci）:
-
```
SELECT _latin1'Müller';
```
- 使用连接默认字符集和校对规则的字符串:
-
```
SELECT 'Müller';
```

字符集引介词和COLLATE子句是根据标准SQL规范实现的。

10.3.8. 在SQL语句中使用COLLATE

- 使用COLLATE子句，能够为一个比较覆盖任何默认校对规则。COLLATE可以用于多种SQL语句中。下面是一些例子:
  - 使用ORDER BY:
  -
```
SELECT k
FROM t1
ORDER BY k COLLATE latin1_german2_ci;
```
  - 使用AS:
  -
```
SELECT k COLLATE latin1_german2_ci AS k1
FROM t1
ORDER BY k1;
```
  - 使用GROUP BY:
  -
```
SELECT k
FROM t1
GROUP BY k COLLATE latin1_german2_ci;
```
  - 使用聚合函数:
  -
```
SELECT MAX(k COLLATE latin1_german2_ci)
FROM t1;
```

- 使用DISTINCT：

```
SELECT DISTINCT k COLLATE latin1_german2_ci
FROM t1;
```

- 使用WHERE：

```
SELECT *
FROM t1
WHERE _latin1 'Müller' COLLATE latin1_german2_ci = k;
SELECT *
FROM t1
WHERE k LIKE _latin1 'Müller' COLLATE latin1_german2_ci;
```

- 使用HAVING：

```
SELECT k
FROM t1
GROUP BY k
HAVING k = _latin1 'Müller' COLLATE latin1_german2_ci;
```

10.3.9. COLLATE子句优先

COLLATE子句有较高的优先级（高于||），因此下面两个表达式是等价的：

```
x || y COLLATE z
x || (y COLLATE z)
```

10.3.10. BINARY操作符

BINARY操作符是COLLATE子句的一个速记符。BINARY '*x*'等价与'*x*' COLLATE *y*，这里*y*是字符集'*x*'二元 校对规则的名字。每一个字符集有一个二元校对规则。例如，latin1字符集的二元 校对规则是latin1_bin，因此，如果列a是字符集latin1，以下两个语句有相同效果：

```
SELECT * FROM t1 ORDER BY BINARY a;
SELECT * FROM t1 ORDER BY a COLLATE latin1_bin;
```

10.3.11. 校对确定较为复杂的一些特殊情况

在绝大多数查询中，MySQL使用哪种校对规则进行比较是很显然的。例如，在下列情况中，校对规则明显的是"列x的列校对规则"：

```
SELECT x FROM T ORDER BY x;
SELECT x FROM T WHERE x = x;
SELECT DISTINCT x FROM T;
```

但是，当涉及多个操作数时，可能不明确。例如：

```
SELECT x FROM T WHERE x = 'Y';
```

这个查询应该使用列x的 校对规则，还是字符串文字'Y'的 校对规则？

标准化SQL使用"可压缩性"规则解决这种问题。基本上，这个意思是：既然x和'Y'都有 校对规则，哪个校对规则优先？这可能比较难解决，但是以下规则适合大多数情况：

- 一个外在的COLLATE子句可压缩性是0（根本不能压缩。）

- 使用不同校对规则的两个字符串连接的可压缩性是1。
- 列校对规则的可压缩性是2。
- "系统常数"（如USER()或VERSION()函数返回的字符串）可压缩性是3。
- 文字规则的可压缩性是4。
- NULL或从NULL派生的表达式的可压缩性是 5。

上述可压缩性值是MySQL5.1当前所用的。

这样上述规则可以模糊解决:

- 使用最低的可压缩性值的校对规则。
- 如果两侧有相同的可压缩性，那么如果校对规则不同则发生错误。

例如:

column1 = 'A'	使用column1的校对规则
column1 = 'A' COLLATE x	使用'A'的校对规则
column1 COLLATE x = 'A' COLLATE y	错误

使用COERCIBILITY（）函数确定一个字符串表达式的可压缩性:

```
mysql> SELECT COERCIBILITY('A' COLLATE latin1_swedish_ci);
    -> 0
mysql> SELECT COERCIBILITY(VERSION());
    -> 3
mysql> SELECT COERCIBILITY('A');
    -> 4
```

见12.9.3节，"信息函数"。

没有系统常数或可忽略的压缩性。函数如USER()的可压缩性是2而不是3，文字的可压缩性是3而不是4。

10.3.12. 校对必须适合字符集

请注意每个字符集有一个或多个校对规则，并且每个校对规则只能属于一个字符集。因此，以下语句会产生一个错误信息，因为校对规则latin2_bin对于字符集latin1非法:

mysql> **SELECT _latin1 'x' COLLATE latin2_bin;**

ERROR 1251: COLLATION 'latin2_bin' is not valid

for CHARACTER SET 'latin1'

10.3.13. 校对效果的示例

假设表T中的列X有这些latin1列值:

Muffler

Müller

MX Systems

MySQL

假设使用下面的语句获取列值:

SELECT X FROM T ORDER BY X COLLATE *collation_name*;

使用不同校对规则的列值结果排序见下表:

latin1_swedish_ci	latin1_german1_ci	latin1_german2_ci
Muffler	Muffler	Müller
MX系统	Müller	Muffler
Müller	MX系统	MX系统
MySQL	MySQL	MySQL

本表显示了我们在ORDER BY字句中使用不同所校对规则的效果的示例。在本例中导致不同排序的字符是上面带有两个圆点的U（ü），它在德语中发音为"U-umlaut"。

- 第一列显示的是使用瑞典/芬兰校对规则的SELECT语句的结果，它被称作U-umlaut使用Y排序。
- 第二列显示的是使用德语DIN-1校对规则的SELECT语句的结果，它被称作U-umlaut使用U排序。
- 第三列显示的是使用德语DIN-2校对规则的SELECT语句的结果，它被称作U-umlaut使用UE排序。

10.4. 字符集支持影响到的操作

10.4.1. 结果字符串
10.4.2. CONVERT()
10.4.3. CAST()
10.4.4. SHOW语句

本节讨论在MySQL5.1中考虑到字符集信息的操作。

10.4.1. 结果字符串

MySQL中有许多操作符和函数可以返回字符串。本节回答这个问题：返回的字符串使用什么字符集和 校对规则？

对于简单的函数，即接收字符串输入然后返回一个字符串结果作为输出的函数，输出的字符集和校对规则与原始输入的相同。例如，UPPER（*X*）返回一个字符串，其字符和 校对规则与*X*相同。类似的函数还有INSTR()、LCASE()、LOWER()、LTRIM()、MID()、REPEAT()、REPLACE()、REVERSE()、RIGHT()、RPAD()、RTRIM()、SOUNDEX()、SUBSTRING()、TRIM()、UCASE()和UPPER()。（还需要注意：REPLACE()函数不同于其它函数，它总是忽略输入字符串的 校对规则，并且进行大小写不敏感的比较。）

对于合并多个字符串输入并且返回单个字符串输出的运算，应用标准SQL"聚合规则"：

- 如果存在显式的校对规则*X*，那么使用*X*。
- 如果存在显式的校对规则*X*和*Y*，那么产生一个错误。
- 否则，如果全部校对规则是*X*，那么使用*X*。
- 其它情况，结果没有校对规则。

例如，使用CASE ... WHEN a THEN b WHEN b THEN c COLLATE *X* END。结果校对规则是X。对于CASE、UNION、||、CONCAT()、ELT()、GREATEST()、IF()和LEAST()情况相同。

对于转换为字符数据的运算，从运算得到的结果字符串的字符集和校对规则由character_set_connection和collation_connection系统变量定义。这适用于CAST()、CHAR()、CONV()、FORMAT()、HEX()和SPACE()函数。

10.4.2. CONVERT()

CONVERT（）提供一个在不同字符集之间转换数据的方法。语法是:

```
CONVERT(expr USING transcoding_name)
```

在 MySQL中，转换代码名与相应的字符集名相同。

例子:

```
SELECT CONVERT(_latin1'Müller' USING utf8);
INSERT INTO utf8table (utf8column)
    SELECT CONVERT(latin1field USING utf8) FROM latin1table;
```

CONVERT(... USING ...)根据标准SQL规范实施。

在传统SQL模式中，如果你转换一个"0"日期字符串到日期类型，CONVERT（）函数返回NULL。在MySQL5.1中还产生一条警告。

10.4.3. CAST()

你也可以使用CAST()函数将一个字符串转换到一个不同的字符集。语法是:

CAST(*character_string* AS *character_data_type* CHARACTER SET *charset_name*)

例如:

SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8);

如果使用CAST()时没有指定CHARACTER SET，结果字符集和校对规则通过character_set_connection 和 collation_connection系统变量定义。如果用CAST()并带有CHARACTER SET X选项，那么结果字符集和校对规则是X和其 默认的校对规则。

你可能不能在CAST()中使用COLLATE子句，但是你可以在外部使用它。也就是说，不是CAST(... COLLATE ...)，而是CAST(...) COLLATE ...。

例如:

SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8) COLLATE utf8_bin;

在传统SQL模式中，如果你转换一个"0"日期字符串到日期类型，CAST()函数返回NULL。在MySQL5.1中还产生一条警告。

10.4.4. SHOW语句

一些SHOW语句提供额外的字符集信息。这些语句包括SHOW CHARACTER SET、SHOW COLLATION、SHOW CREATE DATABASE、SHOW CREATE TABLE和SHOW COLUMNS。

SHOW CHARACTER SET命令显示全部可用的字符集。它带有一个可选的LIKE子句来指示匹配哪些字符集名。例如:

mysql> **SHOW CHARACTER SET LIKE 'latin%';**

```
+---------+----------------------------+-------------------+--------+
| Charset | Description                | Default collation | Maxlen |
+---------+----------------------------+-------------------+--------+
| latin1  | cp1252 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 |

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

见13.5.4.1节，“SHOW CHARACTER SET语法”。

SHOW COLLATION语句的输出包括全部可用的字符集。它带有一个可选的LIKE子句来指示匹配哪些 校对规则名。例如：

```
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 |
| latin1_spanish_ci | latin1  | 94 |         |          |       0 |
+-------------------+---------+----+---------+----------+---------+
```

见13.5.4.2节，“SHOW COLLATION语法”。

SHOW CREATE DATABASE语句显示创建给定数据库的CREATE DATABASE语句。结果包括全部数据库选项。支持DEFAULT CHARACTER SET和COLLATE。全部数据库选项存储在命名为db.Opt的文本文件中，该文件能够在数据库目录中找到。

```
mysql> SHOW CREATE DATABASE test;
+----------+-----------------------------------------------------------------+
| Database | Create Database                                                 |
+----------+-----------------------------------------------------------------+
| test     | CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET latin1 */ |
+----------+-----------------------------------------------------------------+
```

见13.5.4.4节，“SHOW CREATE DATABASE语法”

SHOW CREATE TABLE与SHOW CREATE DATABASE相似，但是显示创建给定数据库的CREATE TABLE语句。列定义显示任何字符集规格，并且表选项包括字符集信息。

见13.5.4.5节，“SHOW CREATE TABLE语法”

当以SHOW FULL COLUMNS调用时，SHOW COLUMNS语句显示表中列的校对规则。具有CHAR、VARCHAR或TEXT数据类型的列有非NULL的 校对规则。数值列和其它非字符类型的列有NULL校对规则。例如：

mysql> **SHOW FULL COLUMNS FROM person\G**

********** 1. row **********

Field: id

Type: smallint(5) unsigned

```
 Collation: NULL
      Null: NO
       Key: PRI
   Default: NULL
     Extra: auto_increment
Privileges: select,insert,update,references
   Comment:
*************************** 2. row ***************************
     Field: name
      Type: char(60)
 Collation: latin1_swedish_ci
      Null: NO
       Key:
   Default:
     Extra:
Privileges: select,insert,update,references
   Comment:
```

字符集不是显示的部分。（字符集名隐含在校对规则名中。）

见13.5.4.3节，“SHOW COLUMNS语法”。

10.5. Unicode支持

MySQL 5.1支持两种字符集以保存Unicode数据：

- ucs2，UCS-2 Unicode字符集。
- utf8，Unicode字符集的UTF8编码。

在UCS-2（二进制Unicode表示法）中，每一个字符用一个双字节的Unicode编码来表示的，第一个字节表示重要的意义。例如："LATIN CAPITAL LETTER A"的Unicode编码是0x0041，它按顺序存储为两个字节：0x00 0x41。"CYRILLIC SMALL LETTER YERU"（Unicode 0x044B）顺序存储为两个字节：0x04 0x4B。对于Unicode字符和它们的编码，请参见Unicode 主页。

当前，UCS-2还不能够用作为客户端字符集，这意味着SET NAMES 'ucs2'不起作用。

UTF8字符集（转换Unicode表示）是存储Unicode数据的一种可选方法。它根据 RFC 3629执行。UTF8字符集的思想是不同Unicode字符采用变长字节序列编码：

- 基本拉丁字母、数字和标点符号使用一个字节。
- 大多数的欧洲和中东手写字母适合两个字节序列：扩展的拉丁字母（包括发音符号、长音符号、重音符号、低音符号和其它音符）、西里尔字母、希腊语、亚美尼亚语、希伯来语、阿拉伯语、叙利亚语和其它语言。

- 韩语、中文和日本象形文字使用三个字节序列。

RFC 3629说明了采用一到四个字节的编码序列。当前，MySQLUTF8不支持四个字节。（UTF8编码的旧标准是由RFC 2279给出，它描述了从一到六个字节的UTF8编码序列。RFC 3629补充了作废的RFC 2279；因此，不再使用5个字节和6个字节的编码序列。）

提示：使用UTF8时为了节省空间，使用VARCHAR而不要用CHAR。否则，MySQL必须为一个CHAR(10) CHARACTER SET utf8列预备30个字节，因为这是可能的最大长度。

10.6. 用于元数据的UTF8

元数据是"关于数据的数据"。描述数据库的任何数据—作为数据库内容的对立面—是元数据。因此，列名、数据库名、用户名、版本名以及从SHOW语句得到的结果中的大部分字符串是元数据。还包括INFORMATION_SCHEMA数据库中的表中的内容，因为定义的那些表存储关于数据库对象的信息。

元数据表述必须满足这些需求：

- 全部元数据必须在同一字符集内。否则，对INFORM一个TION_SCHEMA数据库中的表执行的SHOW命令和SELECT查询不能正常工作，因为这些运算结果中的同一列的不同行将会使用不同的字符集。

- 元数据必须包括所有语言的所有字符。否则，用户将不能够使用它们自己的语言来命名列和表。

为了满足这两个需求，MySQL使用Unicode字符集存储元数据，即UTF8。如果你从不使用重音字符，这不会导致任何破坏。但如果你使用重音字符，应该注意的是元数据是用UTF8存储。

这意味着，USER()、CURRENT_USER()、DATABASE()和VERSION()函数的返回值被 默认设置为UTF8字符集，这与同义函数如SESSION_USER() 和SYSTEM_USER()的结果相同。

服务器将character_set_system系统变量设置为元数据字符集的名：

```
mysql> SHOW VARIABLES LIKE 'character_set_system';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| character_set_system | utf8  |
+----------------------+-------+
```

存储元数据使用Unicode并不意味着列头和DESCRIBE函数的结果默认在character_set_system字符集中。当你使用SELECT column1 FROM t语句时，名字为column1的列从服务器返回客户端并使用由SET NAMES语句确定的字符集。更明确地说，使用的字符集是由character_set_results系统变量的值确定的。如果这个系统变量设置为NULL，不执行字符转换，服务器使用最初的字符集（字符集由character_set_system系统变量设置）返回元数据。

如果你希望服务器不使用UTF8字符集返回元数据结果，那么使用SET NAMES语句强制服务器执行字符集转换（见10.3.6节，"连接字符集和校对"），或者在客户端执行转换。在客户端执行转换效率较高，但这种选项并不能使用于全部客户端。

如果你正在一个语句中使用（例如）USER()函数进行比较或赋值，不要担心。MySQL为你执行一些原子转换。

```
SELECT * FROM Table1 WHERE USER() = latin1_column;
```

这是可以的，因为在比较之前latin1_column列的内容会自动转换到UTF8。

```
INSERT INTO Table1 (latin1_column) SELECT USER();
```

这是可以的，因为赋值之前USER()函数返回的内容自动转换为latin1。至今，自动转换没有全部实施，但是以后的版本中应该工作正常。

尽管自动转换不属于SQL标准，SQL标准化文档中说每一个字符集是（根据支持的字符）Unicode的"子集"。因此，一个知名的原则是，"适用超集的字符集能够应用于其子集"，我们相信Unicode的 校对规则能够应用于非Unicode字符串的比较。

注释：在MySQL5.1中，errmsg.txt文件全部使用UTF8。客户端字符集的转换是自动进行的，如同元数据。

10.7. 与其它DBMS的兼容性

对于MaxDB兼容性，下面两个语句是相同的：

```
CREATE TABLE t1 (f1 CHAR(n) UNICODE);
CREATE TABLE t1 (f1 CHAR(n) CHARACTER SET ucs2);
```

10.8. 新字符集配置文件格式

字符集配置存储在XML文件中，一个字符集对应一个文件。

10.9. 国家特有字符集

ANSI SQL定义了NCHAR或者NATIONAL CHAR作为一个方法来指示CHAR类型的列应该使用某些预定义的字符集。MySQL5.1使用utf8作为预定义的字符集。例如，这些列类型声明是等价的：

```
CHAR(10) CHARACTER SET utf8

NATIONAL CHARACTER(10)

NCHAR(10)
```

下面同样：

```
VARCHAR(10) CHARACTER SET utf8

NATIONAL VARCHAR(10)

NCHAR VARCHAR(10)

NATIONAL CHARACTER VARYING(10)

NATIONAL CHAR VARYING(10)
```

你能够使用N'*literal*'来创建一个使用国家特有字符集的字符串。这两个语句是等价的：

```
SELECT N'some text';

SELECT _utf8'some text';
```

关于MySQL从4.1以前的版本到5.1版本字符集升级的信息，请参见*MySQL4.1*参考手册。

10.10. MySQL支持的字符集和校对

10.10.1. Unicode字符集
10.10.2. 西欧字符集
10.10.3. 中欧字符集
10.10.4. 南欧与中东字符集
10.10.5. 波罗的海字符集
10.10.6. 西里尔字符集
10.10.7. 亚洲字符集

MySQL支持30多种字符集的70多种 校对规则。字符集和它们的默认校对规则可以通过SHOW CHARACTER SET语句显示：

```
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+
| Charset  | Description                 | Default collation   |
+----------+-----------------------------+---------------------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |
| dec8     | DEC West European           | dec8_swedish_ci     |
| cp850    | DOS West European           | cp850_general_ci    |
| hp8      | HP West European            | hp8_english_ci      |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |
| latin1   | cp1252 West European        | latin1_swedish_ci   |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |
| swe7     | 7bit Swedish                | swe7_swedish_ci     |
| ascii    | US ASCII                    | ascii_general_ci    |
```

```
| ujis      | EUC-JP Japanese              | ujis_japanese_ci     |
| sjis      | Shift-JIS Japanese           | sjis_japanese_ci     |
| hebrew    | ISO 8859-8 Hebrew            | hebrew_general_ci    |
| tis620    | TIS620 Thai                  | tis620_thai_ci       |
| euckr     | EUC-KR Korean                | euckr_korean_ci      |
| koi8u     | KOI8-U Ukrainian             | koi8u_general_ci     |
| gb2312    | GB2312 Simplified Chinese    | gb2312_chinese_ci    |
| greek     | ISO 8859-7 Greek             | greek_general_ci     |
| cp1250    | Windows Central European     | cp1250_general_ci    |
| gbk       | GBK Simplified Chinese       | gbk_chinese_ci       |
| latin5    | ISO 8859-9 Turkish           | latin5_turkish_ci    |
| armscii8  | ARMSCII-8 Armenian           | armscii8_general_ci  |
| utf8      | UTF-8 Unicode                | utf8_general_ci      |
| ucs2      | UCS-2 Unicode                | ucs2_general_ci      |
| cp866     | DOS Russian                  | cp866_general_ci     |
| keybcs2   | DOS Kamenicky Czech-Slovak   | keybcs2_general_ci   |
| macce     | Mac Central European         | macce_general_ci     |
| macroman  | Mac West European            | macroman_general_ci  |
| cp852     | DOS Central European         | cp852_general_ci     |
| latin7    | ISO 8859-13 Baltic           | latin7_general_ci    |
| cp1251    | Windows Cyrillic             | cp1251_general_ci    |
| cp1256    | Windows Arabic               | cp1256_general_ci    |
| cp1257    | Windows Baltic               | cp1257_general_ci    |
| binary    | Binary pseudo charset        | binary               |
| geostd8   | GEOSTD8 Georgian             | geostd8_general_ci   |
| cp932     | SJIS for Windows Japanese    | cp932_japanese_ci    |
| eucjpms   | UJIS for Windows Japanese    | eucjpms_japanese_ci  |
+----------+-----------------------------+---------------------+
```

10.10.1. Unicode字符集

MySQL有两种Unicode字符集。你能够使用这些字符集保存大约650种语言的文本。

· ucs2 (UCS-2 Unicode)校对规则:

- ```
  mysql> SHOW COLLATION LIKE 'ucs2%';
  +--------------------+---------+-----+---------+----------+---------+
  | Collation          | Charset | Id  | Default | Compiled | Sortlen |
  +--------------------+---------+-----+---------+----------+---------+
  | ucs2_general_ci    | ucs2    |  35 | Yes     | Yes      |       1 |
  | ucs2_bin           | ucs2    |  90 |         | Yes      |       1 |
  | ucs2_unicode_ci    | ucs2    | 128 |         | Yes      |       8 |
  | ucs2_icelandic_ci  | ucs2    | 129 |         | Yes      |       8 |
  | ucs2_latvian_ci    | ucs2    | 130 |         | Yes      |       8 |
  | ucs2_romanian_ci   | ucs2    | 131 |         | Yes      |       8 |
  | ucs2_slovenian_ci  | ucs2    | 132 |         | Yes      |       8 |
  | ucs2_polish_ci     | ucs2    | 133 |         | Yes      |       8 |
  | ucs2_estonian_ci   | ucs2    | 134 |         | Yes      |       8 |
  | ucs2_spanish_ci    | ucs2    | 135 |         | Yes      |       8 |
  | ucs2_swedish_ci    | ucs2    | 136 |         | Yes      |       8 |
  | ucs2_turkish_ci    | ucs2    | 137 |         | Yes      |       8 |
  | ucs2_czech_ci      | ucs2    | 138 |         | Yes      |       8 |
  | ucs2_danish_ci     | ucs2    | 139 |         | Yes      |       8 |
  | ucs2_lithuanian_ci | ucs2    | 140 |         | Yes      |       8 |
  | ucs2_slovak_ci     | ucs2    | 141 |         | Yes      |       8 |
  | ucs2_spanish2_ci   | ucs2    | 142 |         | Yes      |       8 |
  | ucs2_roman_ci      | ucs2    | 143 |         | Yes      |       8 |
  | ucs2_persian_ci    | ucs2    | 144 |         | Yes      |       8 |
  | ucs2_esperanto_ci  | ucs2    | 145 |         | Yes      |       8 |
  +--------------------+---------+-----+---------+----------+---------+
  ```

- utf8 (UTF-8 Unicode)校对规则:

  ```
  mysql> SHOW COLLATION LIKE 'utf8%';
  +--------------------+---------+-----+---------+----------+---------+
  | Collation          | Charset | Id  | Default | Compiled | Sortlen |
  +--------------------+---------+-----+---------+----------+---------+
  | utf8_general_ci    | utf8    |  33 | Yes     | Yes      |       1 |
  | utf8_bin           | utf8    |  83 |         | Yes      |       1 |
  | utf8_unicode_ci    | utf8    | 192 |         | Yes      |       8 |
  | utf8_icelandic_ci  | utf8    | 193 |         | Yes      |       8 |
  | utf8_latvian_ci    | utf8    | 194 |         | Yes      |       8 |
  | utf8_romanian_ci   | utf8    | 195 |         | Yes      |       8 |
  | utf8_slovenian_ci  | utf8    | 196 |         | Yes      |       8 |
  | utf8_polish_ci     | utf8    | 197 |         | Yes      |       8 |
  | utf8_estonian_ci   | utf8    | 198 |         | Yes      |       8 |
  | utf8_spanish_ci    | utf8    | 199 |         | Yes      |       8 |
  | utf8_swedish_ci    | utf8    | 200 |         | Yes      |       8 |
  | utf8_turkish_ci    | utf8    | 201 |         | Yes      |       8 |
  ```
  ```

utf8_czech_ci	utf8	202		Yes	8
utf8_danish_ci	utf8	203		Yes	8
utf8_lithuanian_ci	utf8	204		Yes	8
utf8_slovak_ci	utf8	205		Yes	8
utf8_spanish2_ci	utf8	206		Yes	8
utf8_roman_ci	utf8	207		Yes	8
utf8_persian_ci	utf8	208		Yes	8
utf8_esperanto_ci	utf8	209		Yes	8

utf8_unicode_ci校对规则是根据Unicode校对规则算法（UCA）执行的， 校对规则描述见 http://www.unicode.org/reports/tr10/。此校对规则使用UCA 4.0.0版本砝码键：http://www.unicode.org/Public/UC一个/4.0.0/一个llkeys-4.0.0.txt。（以下讨论使用utf8_unicode_ci，但同样适合ucs2_unicode_ci。）

当前，utf8_unicode_ci校对规则仅部分支持Unicode校对规则算法。一些字符还是不能支持。并且，不能完全支持组合的记号。这主要影响越南和俄罗斯的一些少数民族语言，如：Udmurt 、Tatar、Bashkir和Mari。

utf8_unicode_ci的最主要的特色是支持扩展，即当把一个字母看作与其它字母组合相等时。例如，在德语和一些其它语言中‘ß’等于‘ss’。

utf8_general_ci是一个遗留的 校对规则，不支持扩展。它仅能够在字符之间进行逐个比较。这意味着utf8_general_ci校对规则进行的比较速度很快，但是与使用utf8_unicode_ci的 校对规则相比，比较正确性较差）。

例如，使用utf8_general_ci和utf8_unicode_ci两种 校对规则下面的比较相等：

```
Ä = A
Ö = O
Ü = U
```

两种校对规则之间的区别是，对于utf8_general_ci下面的等式成立：

```
ß = s
```

但是，对于utf8_unicode_ci下面等式成立：

```
ß = ss
```

对于一种语言仅当使用utf8_unicode_ci排序做的不好时，才执行与具体语言相关的utf8字符集 校对规则。例如，对于德语和法语，utf8_unicode_ci工作的很好，因此不再需要为这两种语言创建特殊的utf8校对规则。

utf8_general_ci也适用与德语和法语，除了‘ß’等于‘s’，而不是‘ss’之外。如果你的应用能够接受这些，那么应该使用utf8_general_ci，因为它速度快。否则，使用utf8_unicode_ci，因为它比较准确。

utf8_swedish_ci，与其它语言相关的utf8的校对规则相似，来源于utf8_unicode_ci，使用额外的语言规则。例如，在瑞典语中，以下的关系式成立，它在德语和法语中不成立：

```
Ü = Y < Ö
```

utf8_spanish_ci和utf8_spanish2_ci校对规则分别适用于现代和古典西班牙语。在两种 校对规则中，ñ’（n-发音符）是‘n’和‘o’之间的间隔字母。另外，对于古典西班牙语，‘ch’是‘c’和d之间的间隔字母，并且‘ll’是‘l’和‘m’之间的间隔字母。

10.10.2. 西欧字符集

西欧字符集覆盖大多数西欧语言，如法语、西班牙语、加泰罗尼亚语、巴斯克人语、葡萄牙语、意大利语、阿而巴尼亚语、荷兰语、德语、丹麦语、瑞典语、挪威语、芬兰语、法罗人语、冰岛语、爱尔兰语、苏格兰语和英语。

- ascii（US ASCII）校对规则：

o ascii_bin

o ascii_general_ci（ 默认）

- cp850（DOS西欧） 校对规则：

- cp850_bin
- cp850_general_ci（默认）

· dec8（DEC 西欧）校对规则：

- dec8_bin
- dec8_swedish_ci（默认）

· hp8（HP 西欧）校对规则：

- hp8_bin
- hp8_english_ci（默认）

· latin1（cp1252 西欧）校对规则：

- latin1_bin
- latin1_danish_ci
- latin1_general_ci
- latin1_general_cs
- latin1_german1_ci
- latin1_german2_ci
- latin1_spanish_ci
- latin1_swedish_ci（默认）

latin1是 默认字符集。latin1_swedish_ci是 默认的校对规则，它用于大多数MySQL客户。虽然经常说它以瑞典/芬兰 校对规则为基础，但瑞典和芬兰人不同意这种说法。

latin1_german1_ci和latin1_german2_ci校对规则基于DIN-1和DIN-2标准，这里DIN代表*Deutsches Institut für Normung*（德语等价于ANSI）。DIN-1被叫做"字典校对规则"，DIN-2被叫做"电话簿校对规则"。

- latin1_german1_ci（字典）规则：

```
Ä = a
Ö = O
Ü = U
ß = s
```

- latin1_german2_ci（电话簿）规则：

```
Ä = aE
Ö = OE
Ü = UE
ß = ss
```

在 latin1_spanish_ci校对规则中，'ñ'（n-tilde）是'n'和'o'之间的间隔字母。

· macroma（Mac西欧） 校对规则：

- macroman_bin

- macroman_general_ci（默认）

- swe7（7位瑞典语） 校对规则:
  - swe7_bin
  - swe7_swedish_ci（默认）

10.10.3. 中欧字符集

我们还提供一些用于捷克共和国、斯洛伐克、匈牙利、罗马尼亚、斯罗纹尼亚、克罗地亚和波兰的字符集支持。

- cp1250（Windows中欧） 校对规则:
  - cp1250_bin
  - cp1250_croatian_ci
  - cp1250_czech_cs
  - cp1250_general_ci（默认）
- cp852（DOS 中欧）校对规则:
  - cp852_bin
  - cp852_general_ci（默认）
- keybcs2（DOS Kamenicky Czech-Slovak）校对规则:
  - keybcs2_bin
  - keybcs2_general_ci（默认）
- latin2（ISO 8859-2 中欧）校对规则:
  - latin2_bin
  - latin2_croatian_ci
  - latin2_czech_cs
  - latin2_general_ci（默认）
  - latin2_hungarian_ci
- macce（Mac 中欧）校对规则:
  - macce_bin
  - macce_general_ci（默认）

10.10.4. 南欧与中东字符集

MySQL支持的南欧和中东字符集包括亚美尼亚语、阿拉伯语、乔治亚语、希腊语、希伯莱语和土耳其语:

- armscii8（ARMSCII-8 亚美尼亚语）校对规则:

- o armscii8_bin
- o armscii8_general_ci（默认）
- · cp1256（阿拉伯语Windows） 校对规则:
- o cp1256_bin
- o cp1256_general_ci（默认）
- · geostd8（GEOSTD8乔治亚语） 校对规则:
- o geostd8_bin
- o geostd8_general_ci（默认）
- · greek（ISO 8859-7希腊语）校对规则:
- o greek_bin
- o greek_general_ci（默认）
- · hebrew（ISO 8859-8希伯莱语）校对规则:
- o hebrew_bin
- o hebrew_general_ci（默认）
- · latin5（ISO 8859-9 土耳其语）校对规则:
- o latin5_bin
- o latin5_turkish_ci（默认）

10.10.5. 波罗的海字符集

波罗的海字符集覆盖爱沙尼亚语、拉脱维亚语和立陶宛语言。当前支持的两种波罗的海字符集:

- · cp1257（Windows波罗的海） 校对规则:
- o cp1257_bin
- o cp1257_general_ci（默认）
- o cp1257_lithuanian_ci
- · latin7（ISO 8859-13波罗的海）校对规则:
- o latin7_bin
- o latin7_estonian_cs
- o latin7_general_ci（默认）
- o latin7_general_cs

10.10.6. 西里尔字符集

使用西里尔字符集和校对规则的有Belarusian、保加利亚、俄语和乌克兰语言。

- cp1251（Windows 西里尔）校对规则：
  - cp1251_bin
  - cp1251_bulgarian_ci
  - cp1251_general_ci（默认）
  - cp1251_general_cs
  - cp1251_ukrainian_ci
- cp866（DOS 俄语）校对规则：
  - cp866_bin
  - cp866_general_ci（默认）
- koi8r（KOI8-R Relcom 俄语）校对规则：
  - koi8r_bin
  - koi8r_general_ci（默认）
- koi8u（KOI8-U 乌克兰语）校对规则：
  - koi8u_bin
  - koi8u_general_ci（默认）

10.10.7. 亚洲字符集

10.10.7.1. cp932字符集

我们支持的亚洲字符集包括中文、日语、韩语和泰国语。这些可能比较复杂。例如，中文字符集必须考虑到上千种不同的字符。

- big5（Big5传统中文） 校对规则：
  - big5_bin
  - big5_chinese_ci（默认）
- cp932（SJIS Windows日语）校对规则：
  - cp932_bin
  - cp932_japanese_ci（默认）
- eucjpms（UJIS Windows日语）校对规则：
  - eucjpms_bin
  - eucjpms_japanese_ci（默认）
- euckr（EUC-KR 韩语）校对规则：
  - euckr_bin
  - euckr_korean_ci（默认）

- gb2312（GB2312 简体中文）校对规则：
  - gb2312_bin
  - gb2312_chinese_ci（ 默认）
- gbk（GBK简体中文） 校对规则：
  - gbk_bin
  - gbk_chinese_ci（ 默认）
- sjis（Shift-JIS 日语）校对规则：
  - sjis_bin
  - sjis_japanese_ci（ 默认）
- tis620（TIS620 泰国语）校对规则：
  - tis620_bin
  - tis620_thai_ci（ 默认）
- ujis（EUC-JP 日语）校对规则：
  - ujis_bin
  - ujis_japanese_ci（ 默认）

10.10.7.1. cp932字符集

为什么需要**cp932**？

在MySQL中，sjis字符集对应于由IANA定义的Shift_JIS字符集，它支持JIS X0201和JIS X0208字符。（见 http://www.iana.org/assignments/character-sets。）

但是，“SHIFT JIS”作为描述性术语的含义变得非常含糊不清，并且它常常包括由不同供应商定义的Shift_JIS扩展部分。

例如，使用在日本Windows环境中使用的“SHIFT JIS”是Microsoft对Shift_JIS的Microsoft扩展，它的准确名字是Microsoft Windows Codepage: 932或cp932。除由Shift_JIS支持的字符之外，cp932支持扩展字符，如NEC选择的IBM扩展字符和IBM扩展字符。

许多日本用户在使用这些扩展字符过程中碰到过一些问题。这些问题是由于以下情况引起的：

- MySQL自动转换字符集。
- 字符集通过Unicode转换（ucs2）。
- sjis字符集不支持这些扩展字符转换。
- 从号称“SHIFT JIS”到Unicode的转换，存在一些转换规则，并且一些字符转换到Unicode依赖不同的转换规则。MySQL仅支持这些转换规则中的一种（在后面描述）。

MySQLcp932字符集可以解决这些转换问题。

因为MySQL支持字符集转换，将IANA Shift_JIS 和 cp932分离为两种不同字符集是重要的，因为它们提供不同的转换规则。

cp932与**sjis**有什么不同？

cp932字符集与sjis存在以下不同点：

- cp932支持NEC特殊字符、NEC选择的IBM扩展字符和IBM选择的字符。
- 一些cp932字符有两个不同的编码点，这两种编码点转换为相同Unicode编码点。因此，当从Unicode转换回到cp932时，必须选择一个编码点。对于这种"相互转换"，使用由Microsoft推荐的转换规则。（见 http：//support.microsoft.com/kb/170559/EN-US/。）

转换规则如下：

- 如果字符在JIS X 0208 和NEC特殊字符中同时存在，使用JIS X 0208 的编码点。
- 如果字符在NEC特殊字符和IBM选择的字符中同时存在，使用NEC特殊字符的编码点。
- 如果字符在IBM选择的字符和NEC选择的IBM扩展字符中同时存在，使用IBM扩展字符的编码点。

关于cp932字符的Unicode 值的列表显示信息见http://www.microsoft.com/globaldev/reference/dbcs/932.htm。对于cp932表中的带有下面有四位数字出现的字符的实体，数字代表相应的Unicode（ucs2）编码。对于表中有两个带下划线的数字出现的实体，择有一个以那两个数字开头的cp932字符值的范围。点击一个这种表的实体，将带你到一个页，该页显示每个以那些数字开头的cp932字符的Unicode值。

以下连接很重要。它们与下列字符集的编码相对应：

- NEC特殊字符：

  http://www.microsoft.com/globaldev/reference/dbcs/932/932_87.htm

- NEC选择的IBM扩展字符：
- http://www.microsoft.com/globaldev/reference/dbcs/932/932_ED.htm
- http://www.microsoft.com/globaldev/reference/dbcs/932/932_EE.htm
- IBM选择的字符：
- 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

- cp932与eucjpms结合支持用户自定义字符的转换，并且解决sjis/ujis转换问题。详细信息，请参见http://www.opengroup.or.jp/jvc/cde/sjis-euc-e.html。
- 对于一些字符，与ucs2之间的转换与sjis和cp932之间的转换是不同的。下表举例说明了这些不同。

转换到ucs2：

sjis/cp932值	**sjis**→ **ucs2**转换	**cp932** → **ucs2**转换
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

从ucs2转换：

ucs2值	**ucs2** → **sjis**转换	**ucs2** → **cp932**转换

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

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第11章：列类型

目录

11.1. 列类型概述

11.1.1. 数值类型概述
11.1.2. 日期和时间类型概述
11.1.3. 字符串类型概述

11.2. 数值类型
11.3. 日期和时间类型

11.3.1. DATETIME、DATE和TIMESTAMP类型
11.3.2. TIME类型
11.3.3. YEAR类型
11.3.4. Y2K事宜和日期类型

11.4. String类型

11.4.1. CHAR和VARCHAR类型
11.4.2. BINARY和VARBINARY类型
11.4.3. BLOB和TEXT类型
11.4.4. ENUM类型
11.4.5. SET类型

11.5. 列类型存储需求
11.6. 选择正确的列类型
11.7. 使用来自其他数据库引擎的列类型

MySQL支持多种列类型：数值类型、日期/时间类型和字符串(字符)类型。本章首先对这些列类型进行了概述，然后更加详细地描述了各种列的类型，以及列类型存储需求的总结。概述很简单。关于具体列类型的详细信息应查阅详细的描述，例如指定值时允许使用的格式。

MySQL支持处理空间数据的扩展名。关于空间类型的信息参见第19章：*MySQL*中的空间扩展。

几种列类型描述使用了下述惯例：

· *M*

表示最大显示宽度。最大有效显示宽度是255。

· *D*

适用于浮点和定点类型，并表示小数点后面的位数。最大可能的值是30，但不应大于*M*-2。

· 方括号('['和']')表示可选部分。

11.1. 列类型概述

11.1.1. 数值类型概述
11.1.2. 日期和时间类型概述
11.1.3. 字符串类型概述

11.1.1. 数值类型概述

下面为数值列类型的概述。详细信息参见11.2节，“数值类型”。列存储需求参见11.5节，“列类型存储需求”。

*M*指示最大显示宽度。最大有效显示宽度是255。显示宽度与存储大小或类型包含的值的范围无关，相关描述见11.2节，“数值类型”。

如果为一个数值列指定ZEROFILL，MySQL自动为该列添加UNSIGNED属性。

SERIAL是BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE的一个别名。

在整数列定义中，SERIAL DEFAULT VALUE是NOT NULL AUTO_INCREMENT UNIQUE的一个别名。

警告：应当清楚，当使用在整数值（其中一个是UNSIGNED类型）之间使用减号时，结果是无符号。参见12.8节，“Cast函数和操作符”。

· BIT[(*M*)]

位字段类型。*M*表示每个值的位数，范围为从1到64。如果*M*被省略， 默认为1。

· TINYINT[(*M*)] [UNSIGNED] [ZEROFILL]

很小的整数。带符号的范围是-128到127。无符号的范围是0到255。

· BOOL，BOOLEAN

是TINYINT(1)的同义词。zero值被视为假。非zero值视为真。

在将来，将根据标准SQL引入完全布尔类型的处理。

· SMALLINT[(*M*)] [UNSIGNED] [ZEROFILL]

小的整数。带符号的范围是-32768到32767。无符号的范围是0到65535。

· MEDIUMINT[(*M*)] [UNSIGNED] [ZEROFILL]

中等大小的整数。带符号的范围是-8388608到8388607。无符号的范围是0到16777215。

· INT[(*M*)] [UNSIGNED] [ZEROFILL]

普通大小的整数。带符号的范围是-2147483648到2147483647。无符号的范围是0到4294967295。

· INTEGER[(*M*)] [UNSIGNED] [ZEROFILL]

这是INT的同义词。

· BIGINT[(*M*)] [UNSIGNED] [ZEROFILL]

大整数。带符号的范围是-9223372036854775808到9223372036854775807。无符号的范围是0到18446744073709551615。

应清楚BIGINT列的下述内容：

o 使用带符号的BIGINT或DOUBLE值进行所有算法，因此除了位函数，不应使用大于9223372036854775807(63位)的无符号的大整数! 如果这样做，结果中的最后几位可能出错，这是由于将BIGINT值转换为DOUBLE进行四舍五入时造成的错误。

MySQL可以在以下情况下处理BIGINT：

§ 当使用整数在一个BIGINT列保存大的无符号的值时。

§ 在MIN(*col_name*)或MAX(*col_name*)中，其中*col_name*指BIGINT列。

§ 使用操作符(+，-，*等等)并且两个操作数均为整数时。

o 总是可以使用一个字符串在BIGINT列中保存严格整数值。在这种情况下，MySQL执行字符串-数字转换，其间不存在双精度表示。

o 当两个操作数均为整数值时，-、+和* 操作符使用BIGINT算法。这说明如果乘两个大整数(或来自返回整数的函数)，当结果大于9223372036854775807时，会得到意想不到的结果。

· FLOAT[(*M*,*D*)] [UNSIGNED] [ZEROFILL]

小(单精度)浮点数。允许的值是-3.402823466E+38到-1.175494351E-38、0和1.175494351E-38到3.402823466E+38。这些是理论限制，基于IEEE标准。实际的范围根据硬件或操作系统的不同可能稍微小些。

*M*是小数纵位数，*D*是小数点后面的位数。如果*M*和*D*被省略，根据硬件允许的限制来保存值。单精度浮点数精确到大约7位小数位。

如果指定UNSIGNED，不允许负值。

使用浮点数可能会遇到意想不到的问题，因为在MySQL中的所有计算用双精度完成。参见A.5.7节，“解决与不匹配行有关的问题”。

· DOUBLE[(*M*,*D*)] [UNSIGNED] [ZEROFILL]

普通大小(双精度)浮点数。允许的值是-1.7976931348623157E+308到-2.2250738585072014E-308、0和2.2250738585072014E-308到 1.7976931348623157E+308。这些是理论限制，基于IEEE标准。实际的范围根据硬件或操作系统的不同可能稍微小些。

*M*是小数总位数，*D*是小数点后面的位数。如果*M*和*D*被省略，根据硬件允许的限制来保存值。双精度浮点数精确到大约15位小数位。

如果指定UNSIGNED，不允许负值。

· DOUBLE PRECISION[(*M*,*D*)] [UNSIGNED] [ZEROFILL], REAL[(*M*,*D*)] [UNSIGNED] [ZEROFILL]

为DOUBLE的同义词。除了：如果SQL服务器模式包括REAL_AS_FLOAT选项，REAL是FLOAT的同义词而不是DOUBLE的同义词。

· FLOAT(*p*) [UNSIGNED] [ZEROFILL]

浮点数。*p*表示精度（以位数表示），但MySQL只使用该值来确定是否结果列的数据类型为FLOAT或DOUBLE。如果*p*为从0到24，数据类型变为没有*M*或*D*值的FLOAT。如果*p*为从25到53，数据类型变为没有*M*或*D*值的DOUBLE。结果列范围与本节前面描述的单精度FLOAT或双精度DOUBLE数据类型相同。

FLOAT(*p*)语法与ODBC兼容。

· DECIMAL[(*M*[,*D*])] [UNSIGNED] [ZEROFILL]

压缩的“严格”定点数。*M*是小数位数(精度)的总数，*D*是小数点(标度)后面的位数。小数点和(负数)的‘-’符号不

包括在*M*中。如果*D*是0，则值没有小数点或分数部分。DECIMAL整数最大位数(*M*)为65。支持的十进制数的最大位数(*D*)是30。如果*D*被省略， 默认是0。如果*M*被省略， 默认是10。

如果指定UNSIGNED，不允许负值。

所有DECIMAL列的基本计算(+，-，*，/)用65位精度完成。

· DEC[(*M*[,*D*])] [UNSIGNED] [ZEROFILL], NUMERIC[(*M*[,*D*])] [UNSIGNED] [ZEROFILL], FIXED[(*M*[,*D*])] [UNSIGNED] [ZEROFILL]

是DECIMAL的同义词。FIXED同义词适用于与其它服务器的兼容性。

11.1.2. 日期和时间类型概述

本节综合讨论了临时列类型。详细信息，参见11.3节，“日期和时间类型”。列存储需求参见11.5节，“列类型存储需求”。

· DATE

日期。支持的范围为'1000-01-01'到'9999-12-31'。MySQL以'YYYY-MM-DD'格式显示DATE值，但允许使用字符串或数字为DATE列分配值。

· DATETIME

日期和时间的组合。支持的范围是'1000-01-01 00:00:00'到'9999-12-31 23:59:59'。MySQL以'YYYY-MM-DD HH:MM:SS'格式显示DATETIME值，但允许使用字符串或数字为DATETIME列分配值。

· TIMESTAMP[(*M*)]

时间戳。范围是'1970-01-01 00:00:00'到2037年。

TIMESTAMP列用于INSERT或UPDATE操作时记录日期和时间。如果你不分配一个值，表中的第一个TIMESTAMP列自动设置为最近操作的日期和时间。也可以通过分配一个NULL值，将TIMESTAMP列设置为当前的日期和时间。

TIMESTAMP值返回后显示为'YYYY-MM-DD HH:MM:SS'格式的字符串，显示宽度固定为19个字符。如果想要获得数字值，应在TIMESTAMP 列添加+0。

注释：MySQL 4.1以前使用的TIMESTAMP格式在MySQL 5.1中不支持；关于旧格式的信息参见*MySQL 4.1* 参考手册。

· TIME

时间。范围是'-838:59:59'到'838:59:59'。MySQL以'HH:MM:SS'格式显示TIME值，但允许使用字符串或数字为TIME列分配值。

· YEAR[(2|4)]

两位或四位格式的年。默认是四位格式。在四位格式中，允许的值是1901到2155和0000。在两位格式中，允许的值是70到69，表示从1970年到2069年。MySQL以YYYY 格式显示YEAR值，但允许使用字符串或数字为YEAR列分配值。

11.1.3. 字符串类型概述

本节综合讨论了字符串列类型。详细信息参见11.4节，“String类型”。列存储需求参见11.5节，“列类型存储需

求”。

在某些情况中，MySQL可以将一个字符串列更改为不同于CREATE TABLE或ALTER TABLE语句中所给出的类型。参见13.1.5.1节，“沉寂的列规格变更”。

MySQL 5.1字符串数据类型包括部分在MySQL 4.1之前的版本中没有的特性：

- 许多字符串数据类型的列定义可以包括指定字符集的CHARACTER SET属性，也可能包括校对规则。(CHARSET是CHARACTER SET的一个同义词）。这些属性适用于CHAR、VARCHAR、TEXT类型、ENUM和SET。例如：

```
CREATE TABLE t
(
    c1 CHAR(20) CHARACTER SET utf8,
    c2 CHAR(20) CHARACTER SET latin1 COLLATE latin1_bin
);
```

该表定义创建了一个名为c1的列，具有一个utf8字符集和该字符集的默认 校对规则，和一个名为c2的列以及latin1字符集和该字符集的二元 校对规则。二元校对规则对大小写不敏感。

- MySQL 5.1用字符单位解释在字符列定义中的长度规范。(以前的一些MySQL版本以字节解释长度）。

- 对于CHAR、VARCHAR和TEXT类型，BINARY属性可以为列分配该列字符集的 校对规则。

- 字符列的排序和比较基于分配给列的字符集。在以前的版本中，排序和比较基于服务器字符集的校对规则。对于CHAR和VARCHAR 列，你可以用BINARY属性声明列让排序和 校对规则使用当前的字符代码值而不是词汇顺序。

关于MySQL 5.1中字符集的支持，参见第10章：字符集支持。

- [NATIONAL] CHAR(*M*) [BINARY| ASCII | UNICODE]

固定长度字符串，当保存时在右侧填充空格以达到指定的长度。*M*表示列长度。*M*的范围是0到255个字符。

注释：当检索CHAR值时尾部空格被删除。

如果想要将某个CHAR的长度设为大于255，执行的CREATE TABLE或ALTER TABLE语句将失败并提示错误：

```
mysql> CREATE TABLE c1 (col1 INT, col2 CHAR(500));
ERROR 1074 (42000): Column length too big for column 'col' (max = 255); use BLOB or TEXT instead
mysql> SHOW CREATE TABLE c1;
ERROR 1146 (42S02): Table 'test.c1' doesn't exist
```

CHAR是CHARACTER的简写。NATIONAL CHAR(或其等效短形式NCHAR)是标准的定义CHAR列应使用 默认字符集的SQL方法。这在MySQL中为默认值。

BINARY属性是指定列字符集的二元 校对规则的简写。排序和比较基于数值字符值。

列类型CHAR BYTE是CHAR BINARY的一个别名。这是为了保证兼容性。

可以为CHAR指定ASCII属性。它分配latin1字符集。

可以为CHAR指定UNICODE属性。它分配ucs2字符集。

MySQL允许创建类型CHAR(0)的列。这主要用于必须有一个列但实际上不使用值的旧版本的应用程序相兼容。当你需要只能取两个值的列时也很好：没有定义为NOT NULL的一个CHAR(0)列只占用一位，只可以取值NULL和''(空字符串)。

· CHAR

这是CHAR(1)的同义词。

· [NATIONAL] VARCHAR(*M*) [BINARY]

变长字符串。*M* 表示最大列长度。*M*的范围是0到65,535。(VARCHAR的最大实际长度由最长的行的大小和使用的字符集确定。最大有效长度是65,532字节）。

注释：MySQL 5.1遵从标准SQL规范，并且不删除VARCHAR值的尾部空格。

VARCHAR是字符VARYING的简写。

BINARY属性是指定列的字符集的二元 校对规则的简写。排序和比较基于数值字符值。

VARCHAR保存时用一个字节或两个字节长的前缀+数据。如果VARCHAR列声明的长度大于255，长度前缀是两个字节。

· BINARY(*M*)

BINARY类型类似于CHAR类型，但保存二进制字节字符串而不是非二进制字符串。

· VARBINARY(*M*)

VARBINARY类型类似于VARCHAR类型，但保存二进制字节字符串而不是非二进制字符串。

· TINYBLOB

最大长度为255(2^8–1)字节的BLOB列。

· TINYTEXT

最大长度为255(2^8–1)字符的TEXT列。

· BLOB[(*M*)]

最大长度为65,535(2^{16}–1)字节的BLOB列。

可以给出该类型的可选长度*M*。如果给出，则MySQL将列创建为最小的但足以容纳*M*字节长的值的BLOB类型。

· TEXT[(*M*)]

最大长度为65,535(2^{16}–1)字符的TEXT列。

可以给出可选长度*M*。则MySQL将列创建为最小的但足以容纳*M*字符长的值的TEXT类型。

· MEDIUMBLOB

最大长度为16,777,215(2^{24}–1)字节的BLOB列。

· MEDIUMTEXT

最大长度为16,777,215(2^{24}–1)字符的TEXT列。

· LONGBLOB

最大长度为4,294,967,295或4GB(2^{32}–1)字节的BLOB列。LONGBLOB列的最大有效(允许的)长度取决于客户端/服务器协议中配置最大包大小和可用的内存。

· LONGTEXT

最大长度为4,294,967,295或4GB(2^{32}–1)字符的TEXT列。LONGTEXT列的最大有效(允许的)长度取决于客户端/服务器协议中配置最大包大小和可用的内存。

· ENUM('*value1*','*value2*',...)

枚举类型。只能有一个值的字符串，从值列'*value1*'，'*value2*'，...，NULL中或特殊 "错误值中选出。ENUM列最多可以有65,535个截然不同的值。ENUM值在内部用整数表示。

· SET('*value1*','*value2*',...)

一个设置。字符串对象可以有零个或多个值，每个值必须来自列值'*value1*'，'*value2*'，...SET列最多可以有64个成员。SET值在内部用整数表示。

11.2. 数值类型

MySQL支持所有标准SQL数值数据类型。这些类型包括严格数值数据类型(INTEGER、SMALLINT、DECIMAL和NUMERIC)，以及近似数值数据类型(FLOAT、REAL和DOUBLE PRECISION)。关键字INT是INTEGER的同义词，关键字DEC是DECIMAL的同义词。

BIT数据类型保存位字段值，并且支持MyISAM、MEMORY、InnoDB和BDB表。

作为SQL标准的扩展，MySQL也支持整数类型TINYINT、MEDIUMINT和BIGINT。下面的表显示了需要的每个整数类型的存储和范围。

类型	字节	最小值	最大值
		(带符号的**/**无符号的**)**	**(**带符号的**/**无符号的**)**
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
INT	4	-2147483648	2147483647
		0	4294967295
BIGINT	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615

MySQL还支持选择在该类型关键字后面的括号内指定整数值的显示宽度(例如，INT(4))。该可选显示宽度规定用于显示宽度小于指定的列宽度的值时从左侧填满宽度。

显示宽度并不限制可以在列内保存的值的范围，也不限制超过列的指定宽度的值的显示。

当结合可选扩展属性ZEROFILL使用时， 默认补充的空格用零代替。例如，对于声明为INT(5) ZEROFILL的列，

值4检索为00004。请注意如果在整数列保存超过显示宽度的一个值，当MySQL为复杂联接生成临时表时会遇到问题，因为在这些情况下MySQL相信数据适合原列宽度。

所有整数类型可以有一个可选(非标准)属性UNSIGNED。当你想要在列内只允许非负数和该列需要较大的上限数值范围时可以使用无符号值。

浮点和定点类型也可以为UNSIGNED。同数类型，该属性防止负值保存到列中。然而，与整数类型不同的是，列值的上范围保持不变。

如果为一个数值列指定ZEROFILL，MySQL自动为该列添加UNSIGNED属性。

对于浮点列类型，在MySQL中单精度值使用4个字节，双精度值使用8个字节。

FLOAT类型用于表示近似数值数据类型。SQL标准允许在关键字FLOAT后面的括号内选择用位指定精度(但不能为指数范围)。MySQL还支持可选的只用于确定存储大小的精度规定。0到23的精度对应FLOAT列的4字节单精度。24到53的精度对应DOUBLE列的8字节双精度。

MySQL允许使用非标准语法：FLOAT(*M*,*D*)或REAL(*M*,*D*)或DOUBLE PRECISION(*M*,*D*)。这里，"(*M*,*D*)"表示该值一共显示*M*位整数，其中*D*位位于小数点后面。例如，定义为FLOAT(7,4)的一个列可以显示为-999.9999。MySQL保存值时进行四舍五入，因此如果在FLOAT(7,4)列内插入999.00009，近似结果是999.0001。

MySQL将DOUBLE视为DOUBLE PRECISION(非标准扩展)的同义词。MySQL还将REAL视为DOUBLE PRECISION(非标准扩展)的同义词，除非SQL服务器模式包括REAL_AS_FLOAT选项。

为了保证最大可能的可移植性，需要使用近似数值数据值存储的代码应使用FLOAT或DOUBLE PRECISION，不规定精度或位数。

DECIMAL和NUMERIC类型在MySQL中视为相同的类型。它们用于保存必须为确切精度的值，例如货币数据。当声明该类型的列时，可以(并且通常要)指定精度和标度；例如：

```
salary DECIMAL(5,2)
```

在该例子中，5是精度，2是标度。精度表示保存值的主要位数，标度表示小数点后面可以保存的位数。

在MySQL 5.1中以二进制格式保存DECIMAL和NUMERIC值。

标准SQL要求salary列能够用5位整数位和两位小数保存任何值。因此，在这种情况下可以保存在salary列的值的范围是从-999.99到999.99。

在标准SQL中，语法DECIMAL(*M*)等价于DECIMAL(*M*,0)。同样，语法DECIMAL等价于DECIMAL(*M*,0)，可以通过计算确定*M*的值。在MySQL 5.1中支持DECIMAL和NUMERIC数据类型的变量形式。*M*默认值是10。

DECIMAL或NUMERIC的最大位数是65，但具体的DECIMAL或NUMERIC列的实际范围受具体列的精度或标度约束。如果此类列分配的值小数点后面的位数超过指定的标度允许的范围，值被转换为该标度。(具体操作与操作系统有关，但一般结果均被截取到允许的位数）。

BIT数据类型可用来保存位字段值。BIT(*M*)类型允许存储*M*位值。*M*范围为1到64。

要指定位值，可以使用b'*value*'符。*value*是一个用0和1编写的二进制值。例如，b'111'和b'100000000'分别表示7和128。参见9.1.5节，"位字段值"。

如果为BIT(M)列分配的值的长度小于*M*位，在值的左边用0填充。例如，为BIT(6)列分配一个值b'101'，其效果与分配b'000101'相同。

当要在一个数值列内保存一个超出该列允许范围的值时，MySQL的操作取决于此时有效的SQL模式。如果模式

未设置，MySQL将值裁剪到范围的相应端点，并保存裁减好的值。但是，如果模式设置为traditional("严格模式")，超出范围的值将被拒绝并提示错误，并且根据SQL标准插入会失败。参见5.3.2节，"SQL服务器模式"。

如果INT列是UNSIGNED，列范围的大小相同，但其端点会变为到0和4294967295。如果你试图保存-9999999999和9999999999，以非严格模式保存到列中的值是0和4294967296。

如果在浮点或定点列中分配的值超过指定(或默认)精度和标度规定的范围，MySQL以非严格模式保存表示范围相应端点的值。

当MySQL没有工作在严格模式时，对于ALTER TABLE、LOAD DATA INFILE、UPDATE和多行INSERT语句，由于裁剪发生的转换将报告为警告。当MySQL工作在严格模式时，这些语句将失败，并且部分或全部值不会插入或更改，取决于是否表为事务表和其它因素。详情参见5.3.2节，"SQL服务器模式"。

11.3. 日期和时间类型

11.3.1. DATETIME、DATE和TIMESTAMP类型
11.3.2. TIME类型
11.3.3. YEAR类型
11.3.4. Y2K事宜和日期类型

表示时间值的DATE和时间类型为DATETIME、DATE、TIMESTAMP、TIME和YEAR。每个时间类型有一个有效值范围和一个"零"值，当指定不合法的MySQL不能表示的值时使用"零"值。TIMESTAMP类型有专有的自动更新特性，将在后面描述。

如果试图插入一个不合法的日期，MySQL将给出警告或错误。可以使用ALLOW_INVALID_DATES SQL模式让MySQL接受某些日期，例如'1999-11-31'。当你想要保存一个"可能错误的"用户已经在数据库中指定(例如，以web形式)用于将来处理的值时很有用。在这种模式下，MySQL只验证月范围为从0到12，日范围为从0到31。这些范围可以包括零，因为MySQL允许在DATE或DATETIME列保存日/月和日是零的日期。这在应用程序需要保存一个你不知道确切日期的生日时非常有用。在这种情况下，只需要将日期保存为'1999-00-00'或'1999-01-00'。如果保存此类日期，DATE_SUB()或DATE_ADD等需要完整日期的函数不会得到正确的结果。(如果你不想在日期中出现零，可以使用NO_ZERO_IN_DATE SQL模式)。

MySQL还允许将'0000-00-00'保存为"伪日期"(如果不使用NO_ZERO_DATE SQL模式)。这在某些情况下比使用NULL值更方便(并且数据和索引占用的空间更小)。

将sql_mode系统变量设置为相应模式值，可以更确切你想让MySQL支持哪种日期。参见5.3.2节，"SQL服务器模式"。

当使用日期和时间类型时应记住以下几点:

- MySQL以标准输出格式检索给定日期或时间类型的值，但它尽力解释你指定的各种输入值格式(例如，当你指定一个分配给或与日期或时间类型进行比较的值时)。只支持下面章节中描述的格式。期望你能提供有效值。如果你使用其它格式的值会发生意想不到的结果。

- 包含两位年值的日期会令人模糊，因为世纪不知道。MySQL使用以下规则解释两位年值:

  - 70-99范围的年值转换为1970-1999。

  - 00-69范围的年值转换为2000-2069。

- 尽管MySQL尝试解释几种格式的值，日期总是以年-月-日顺序(例如，'98-09-04')，而不是其它地方常用的月-日-年或日-月-年顺序(例如，'09-04-98'，'04-09-98')。

- 如果值用于数值上下文中，MySQL自动将日期或时间类型的值转换为数字，反之亦然。

· 当 MySQL遇到一个日期或时间类型的超出范围或对于该类型不合法的值时(如本节开始所描述)，它将该值转换为该类的"零"值。一个例外是超出范围的TIME值被裁剪到TIME范围的相应端点。

下面的表显示了各类"零"值的格式。请注意如果启用NO_ZERO_DATE SQL模式，使用这些值会产生警告。

列类型	"零"值
DATETIME	'0000-00-00 00:00:00'
DATE	'0000-00-00'
TIMESTAMP	00000000000000
TIME	'00:00:00'
YEAR	0000

· "零"值是特殊值，但你可以使用表内显示的值显式保存或引用它们。你也可以使用值'0'或0来保存或引用，写起来更容易。

· MyODBC中使用的"零"日期或时间值在MyODBC 2.50.12和以上版本中被自动转换为NULL，因为ODBC不能处理此类值。

11.3.1. DATETIME、DATE和TIMESTAMP类型

11.3.1.1. 自MySQL 4.1以来的TIMESTAMP属性

DATETIME、DATE和TIMESTAMP类型是相关的。该节描述了它们的特征，它们的相似点和不同点。

当你需要同时包含日期和时间信息的值时则使用DATETIME类型。MySQL以'YYYY-MM-DD HH:MM:SS'格式检索和显示DATETIME值。支持的范围为'1000-01-01 00:00:00'到'9999-12-31 23:59:59'。("支持"表示尽管先前的值可能工作，但没有保证)。

当你只需要日期值而不需要时间部分时应使用DATE类型。MySQL用'YYYY-MM-DD'格式检索和显示DATE值。支持的范围是'1000-01-01'到 '9999-12-31'。

TIMESTAMP列类型的属性不固定，取决于MySQL版本和服务器运行的SQL模式。这些属性将在本节后面描述。

可以使用任何常见格式指定DATETIME、DATE和TIMESTAMP值:

· 'YYYY-MM-DD HH:MM:SS'或'YY-MM-DD HH:MM:SS'格式的字符串。允许"不严格"语法：任何标点符都可以用做日期部分或时间部分之间的间割符。例如，'98-12-31 11:30:45'、'98.12.31 11+30+45'、'98/12/31 11*30*45'和'98@12@31 11^30^45'是等价的。

· 'YYYY-MM-DD'或'YY-MM-DD'格式的字符串。这里也允许使用"不严格的"语法。例如，'98-12-31'、'98.12.31'、'98/12/31'和'98@12@31'是等价的。

· 'YYYYMMDDHHMMSS'或'YYMMDDHHMMSS'格式的没有间割符的字符串，假定字符串对于日期类型是有意义的。例如，'19970523091528'和'970523091528'被解释为'1997-05-23 09:15:28'，但'971122129015'是不合法的(它有一个没有意义的分钟部分)，将变为'0000-00-00 00:00:00'。

· 'YYYYMMDD'或'YYMMDD'格式的没有间割符的字符串，假定字符串对于日期类型是有意义的。例如，'19970523'和'970523'被解释为 '1997-05-23'，但'971332'是不合法的(它有一个没有意义的月和日部分)，将变为'0000-00-00'。

· YYYYMMDDHHMMSS或YYMMDDHHMMSS格式的数字，假定数字对于日期类型是有意义的。例如，19830905132800和830905132800被解释为 '1983-09-05 13:28:00'。

· YYYYMMDD或YYMMDD格式的数字，假定数字对于日期类型是有意义的。例如，19830905和830905被解释为'1983-09-05'。

· 函数返回的结果，其值适合DATETIME、DATE或者TIMESTAMP上下文，例如NOW()或CURRENT_DATE。

无效DATETIME、DATE或者TIMESTAMP值被转换为相应类型的"零"值('0000-00-00 00:00:00'、'0000-00-00'或者00000000000000)。

对于包括日期部分间割符的字符串值，如果日和月的值小于10，不需要指定两位数。'1979-6-9'与'1979-06-09'是相同的。同样，对于包括时间部分间割符的字符串值，如果时、分和秒的值小于10，不需要指定两位数。'1979-10-30 1:2:3'与'1979-10-30 01:02:03'相同。

数字值应为6、8、12或者14位长。如果一个数值是8或14位长，则假定为YYYYMMDD或YYYYMMDDHHMMSS格式，前4位数表示年。如果数字 是6或12位长，则假定为YYMMDD或YYMMDDHHMMSS格式，前2位数表示年。其它数字被解释为仿佛用零填充到了最近的长度。

指定为非限定符字符串的值使用给定的长度进行解释。如果字符串为8或14字符长，前4位数表示年。否则，前2位数表示年。从左向右解释字符串内出现的各部分，以发现年、月、日、小时、分和秒值。这说明不应使用少于6字符的字符串。例如，如果你指定'9903'，认为它表示1999年3月，MySQL将在你的表内插入一个"零"日期值。这是因为年和月值是99和03，但日部分完全丢失，因此该值不是一个合法的日期。但是，可以明显指定一个零值来代表缺少的月或日部分。例如，可以使用'990300'来插入值'1999-03-00'。

在一定程度上，可以将一个日期类型的值分配给一个不同的日期类型。但是，值可能会更改或丢失一些信息：

· 如果你为一个DATETIME或TIMESTAMP对象分配一个DATE值，结果值的时间部分被设置为'00:00:00'，因为DATE值未包含时间信息。

· 如果你为一个DATE对象分配一个DATETIME或TIMESTAMP值，结果值的时间部分被删除，因为DATE值未包含时间信息。

· 记住尽管可以使用相同的格式指定DATETIME、DATE和TIMESTAMP值，不同类型的值的范围却不同。例如，TIMESTAMP值不能早于1970或晚于2037。这说明一个日期，例如'1968-01-01'，虽然对于DATETIME或DATE值是有效的，但对于TIMESTAMP值却无效，如果分配给这样一个对象将被转换为0。

当指定日期值时请注意某些缺陷：

· 指定为字符串的值允许的非严格格式可能会欺骗。例如，值'10:11:12'由于':'间割符看上去可能象时间值，但如果用于日期上下文值则被解释为年'2010-11-12'。值'10:45:15'被转换为'0000-00-00'因为'45'不是合法月。

· 在非严格模式，MySQL服务器只对日期的合法性进行基本检查：年、月和日的范围分别是1000到9999、00到12和00到31。任何包含超出这些范围的部分的日期被转换成'0000-00-00'。请注意仍然允许你保存非法日期，例如'2002-04-31'。要想确保不使用严格模式时日期有效，应检查应用程序。

在严格模式，非法日期不被接受，并且不转换。

详细信息参见5.3.2节，"SQL服务器模式"。

· 包含两位年值的日期会令人模糊，因为世纪不知道。MySQL使用以下规则解释两位年值：

o 00-69范围的年值转换为2000-2069。

o 70-99范围的年值转换为1970-1999。

11.3.1.1. 自MySQL 4.1以来的TIMESTAMP属性

注释：在旧版本的MySQL中(4.1之前)，TIMESTAMP列类型的属性在许多方面于本节所描述的大大不同。如果你需要对旧的TIMESTAMP数据进行转化以便在MySQL 5.1中工作，详情请参见*MySQL 4.1* 参考手册。

TIMESTAMP列的显示格式与DATETIME列相同。换句话说，显示宽度固定在19字符，并且格式为YYYY-MM-DD HH:MM:SS。

MySQL服务器也可以以MAXDB模式运行。当服务器以该模式运行时，TIMESTAMP与DATETIME相等。也就是说，如果创建表时服务器以MAXDB模式运行，TIMESTAMP列创建为DATETIME列。结果是，该列使用DATETIME显示格式，有相同的值范围，并且没有自动对当前的日期和时间进行初始化或更新。

要想启用MAXDB模式，在启动服务器时使用--sql-mode=MAXDB服务器选项或在运行时通过设置全局sql_mode变量将SQL服务器模式设置为MAXDB：

```
mysql> SET GLOBAL sql_mode=MAXDB;
```

客户端可以按照下面方法让服务器为它的连接以MAXDB模式运行：

```
mysql> SET SESSION sql_mode=MAXDB;
```

MySQL不接受在日或月列包括一个零或包含非法日期值的时间戳值。该规则的唯一例外是特殊值'0000-00-00 00:00:00'。

你可以非常灵便地确定什么时候初始化和更新TIMESTAMP和对哪些列进行初始化和更新：

- 你可以将当前的时间戳指定为默认值和自动更新的值。但只能选择一个，或者两者都不选。(不可能一个列选择一个行为而另一个列选择另一个行为）。

- 你可以指定哪个TIMESTAMP列自动初始化或更新为当前的日期和时间。不再需要为第1个TIMESTAMP列。

请注意下面讨论所信息只适用于创建时未启用MAXDB模式的表的TIMESTAMP列。(如上所述，MAXDB模式使列创建为DATETIME列）。控制TIMESTAMP列的初始化和更新的规则如下所示：

- 如果一个表内的第1个TIMESTAMP列指定为一个DEFAULT值，则不能忽略。 默认值可以为CURRENT_TIMESTAMP或常量日期和时间值。

- DEFAULT NULL与第1个*TIMESTAMP* 列的DEFAULT CURRENT_TIMESTAMP相同。对于其它TIMESTAMP列，DEFAULT NULL被视为DEFAULT 0。

- 表内的任何一个TIMESTAMP列可以设置为自动初始化为当前时间戳和/或更新。

- 在CREATE TABLE语句中，可以用下面的任何一种方式声明第1个TIMESTAMP列：

o 用DEFAULT CURRENT_TIMESTAMP和ON UPDATE CURRENT_TIMESTAMP子句，列为默认值使用当前的时间戳，并且自动更新。

o 不使用DEFAULT或ON UPDATE子句，与DEFAULT CURRENT_TIMESTAMP ON UPDATECURRENT_TIMESTAMP相同。

o 用DEFAULT CURRENT_TIMESTAMP子句不用ON UPDATE子句，列为默认值使用当前的时间戳但是不自动更新。

o 不用DEFAULT子句但用ON UPDATE CURRENT_TIMESTAMP子句，列有默认值0并自动更新。

o 用常量DEFAULT值，列有给出的 默认值。如果列有一个ON UPDATE CURRENT_TIMESTAMP子句，它自动更新，否则不。

换句话说，你可以为初始值和自动更新的值使用当前的时间戳，或者其中一个使用，或者两个皆不使用。(例如，你可以指定ON UPDATE来启用自动更新而不让列自动初始化）。

· 在DEFAULT和ON UPDATE子句中可以使用CURRENT_TIMESTAMP、CURRENT_TIMESTAMP()或者NOW()。它们均具有相同的效果。

两个属性的顺序并不重要。如果一个TIMESTAMP列同时指定了DEFAULT和ON UPDATE，任何一个可以在另一个的前面。

例子，下面这些语句是等效的:

```
CREATE TABLE t (ts TIMESTAMP);

CREATE TABLE t (ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP

                             ON UPDATE CURRENT_TIMESTAMP);

CREATE TABLE t (ts TIMESTAMP ON UPDATE CURRENT_TIMESTAMP

                             DEFAULT CURRENT_TIMESTAMP);
```

· 要为TIMESTAMP列而不是第1列指定自动默认或更新，必须通过将第1个TIMESTAMP列显式分配一个常量DEFAULT值来禁用自动初始化和更新。(例如，DEFAULT 0或DEFAULT'2003-01-01 00:00:00')。然后，对于其它TIMESTAMP列，规则与第1个TIMESTAMP列相同，例外情况是不能忽略DEFAULT和ON UPDATE子句。如果这样做，则不会自动进行初始化或更新。

例如：下面这些语句是等效的:

```
CREATE TABLE t (

    ts1 TIMESTAMP DEFAULT 0,

    ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP

                  ON UPDATE CURRENT_TIMESTAMP);

CREATE TABLE t (

    ts1 TIMESTAMP DEFAULT 0,

    ts2 TIMESTAMP ON UPDATE CURRENT_TIMESTAMP

                  DEFAULT CURRENT_TIMESTAMP);
```

可以对每个连接设置当前的时区，相关描述参见5.10.8节，“MySQL服务器时区支持”。TIMESTAMP值以UTC格式保存，存储时对当前的时区进行转换，检索时再转换回当前的时区。只要时区设定值为常量，便可以得到保存时的值。如果保存一个TIMESTAMP值，应更改时区然后检索该值，它与你保存的值不同。这是因为在两个方向的转换中没有使用相同的时区。当前的时区可以用作time_zone系统变量的值。

可以在TIMESTAMP列的定义中包括NULL属性以允许列包含NULL值。例如:

```
CREATE TABLE t

(

  ts1 TIMESTAMP NULL DEFAULT NULL,

  ts2 TIMESTAMP NULL DEFAULT 0,

  ts3 TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP
```

```
);
```

如果未指定NULL属性，将列设置为NULL设置则会将它设置为当前的时间戳。请注意允许NULL值的TIMESTAMP列不会采用当前的时间戳，除非要么其 默认值定义为CURRENT_TIMESTAMP，或者NOW()或CURRENT_TIMESTAMP被插入到该列内。换句话说，只有使用如下定义创建，定义为NULL的TIMESTAMP列才会自动更新：

```
CREATE TABLE t (ts NULLDEFAULT CURRENT_TIMESTAMP);
```

否则-也就是说，如果使用NULL而不是DEFAULT TIMESTAMP来定义TIMESTAMP列，如下所示...

```
CREATE TABLE t1 (ts NULL DEFAULT NULL);

CREATE TABLE t2 (ts NULL DEFAULT '0000-00-00 00:00:00');
```

...则必须显式插入一个对应当前日期和时间的值。例如：

```
INSERT INTO t1 VALUES (NOW());

INSERT INTO t2 VALUES (CURRENT_TIMESTAMP);
```

11.3.2. TIME类型

MySQL以'HH:MM:SS'格式检索和显示TIME值(或对于大的小时值采用'HHH:MM:SS'格式)。TIME值的范围可以从'-838:59:59'到'838:59:59'。小时部分会因此大的原因是TIME类型不仅可以用于表示一天的时间(必须小于24小时)，还可能为某个事件过去的时间或两个事件之间的时间间隔(可以大于24小时，或者甚至为负)。

你可以用各种格式指定TIME值：

- 'D HH:MM:SS.fraction'格式的字符串。还可以使用下面任何一种"非严格"语法：'HH:MM:SS.fraction'、'HH:MM:SS'、'HH:MM'、'D HH:MM:SS'、'D HH:MM'、'D HH'或'SS'。这里D表示日，可以取0到34之间的值。请注意MySQL还不保存分数。

- 'HHMMSS'格式的没有间割符的字符串，假定是有意义的时间。例如，'101112'被理解为'10:11:12'，但'109712'是不合法的(它有一个没有意义的分钟部分)，将变为'00:00:00'。

- HHMMSS格式的数值，假定是有意义的时间。例如，101112被理解为'10:11:12'。下面格式也可以理解：SS、MMSS、HHMMSS、HHMMSS.fraction。请注意MySQL还不保存分数。

- 函数返回的结果，其值适合TIME上下文，例如CURRENT_TIME。

对于指定为包括时间部分间割符的字符串的TIME值，如果时、分或者秒值小于10，则不需要指定两位数。'8:3:2'与'08:03:02'相同。

为TIME列分配简写值时应注意。没有冒号，MySQL解释值时假定最右边的两位表示秒。(MySQL解释TIME值为过去的时间而不是当天的时间）。例如，你可能认为'1112'和1112表示'11:12:00'(11点过12分)，但MySQL将它们解释为'00:11:12'(11分，12 秒)。同样，'12'和12 被解释为 '00:00:12'。相反，TIME值中使用冒号则肯定被看作当天的时间。也就是说，'11:12'表示'11:12:00'，而不是'00:11:12'。

超出TIME范围但合法的值被裁为范围最接近的端点。例如，'-850:00:00'和'850:00:00'被转换为'-838:59:59'和'838:59:59'。

无效TIME值被转换为'00:00:00'。请注意由于'00:00:00'本身是一个合法TIME值，只从表内保存的一个'00:00:00'值还不能说出原来的值是 '00:00:00'还是不合法的值。

11.3.3. YEAR类型

YEAR类型是一个单字节类型用于表示年。

MySQL以YYYY格式检索和显示YEAR值。范围是1901到2155。

可以指定各种格式的YEAR值:

- 四位字符串，范围为'1901'到'2155'。
- 四位数字，范围为1901到2155。
- 两位字符串，范围为'00'到'99'。'00'到'69'和'70'到'99'范围的值被转换为2000到2069和1970到1999范围的YEAR值。
- 两位整数，范围为1到99。1到69和70到99范围的值被转换为2001到2069和1970到1999范围的YEAR值。请注意两位整数范围与两位字符串范围稍有不同，因为你不能直接将零指定为数字并将它解释为2000。你必须将它指定为一个字符串'0'或'00'或它被解释为0000。
- 函数返回的结果，其值适合YEAR上下文，例如NOW()。

非法YEAR值被转换为0000。

11.3.4. Y2K事宜和日期类型

MySQL本身对于2000年(Y2K)是安全的(参见1.4.5节，“2000年兼容性”)，但输入给MySQL的值可能不安全。任何包含两位年值的输入都会令人模糊，因为世纪不知道。这些值必须解释为四位形式，因为MySQL内部使用四位来保存年。

对于DATETIME、DATE、TIMESTAMP和YEAR类型，MySQL使用以下规则解释含模糊年值的日期:

- 00-69范围的年值转换为2000-2069。
- 70-99范围的年值转换为1970-1999。

请记住这些规则只是合理猜测数据值表示什么。如果MySQL使用的启发不能产生正确的值，你应提供包含四位年值的确切输入。

ORDER BY可以正确排序有两位年的TIMESTAMP或YEAR值。

部分函数如MIN()和MAX()将TIMESTAMP或YEAR转换为一个数字。这说明使用有两位年值的值，这些函数不能工作正确。在这种情况下的修复方法是将TIMESTAMP或YEAR转换为四位年格式或使用MIN(DATE_ADD(TIMESTAMP,INTERVAL 0 DAYS))。

11.4. String类型

11.4.1. CHAR和VARCHAR类型
11.4.2. BINARY和VARBINARY类型
11.4.3. BLOB和TEXT类型
11.4.4. ENUM类型
11.4.5. SET类型

字符串类型指CHAR、VARCHAR、BINARY、VARBINARY、BLOB、TEXT、ENUM和SET。该节描述了这些类型如何工作以及如何在查询中使用这些类型。

11.4.1. CHAR和VARCHAR类型

CHAR和VARCHAR类型类似，但它们保存和检索的方式不同。它们的最大长度和是否尾部空格被保留等方面也不同。在存储或检索过程中不进行大小写转换。

CHAR和VARCHAR类型声明的长度表示你想要保存的最大字符数。例如，CHAR(30)可以占用30个字符。

CHAR列的长度固定为创建表时声明的长度。长度可以为从0到255的任何值。当保存CHAR值时，在它们的右边填充空格以达到指定的长度。当检索到CHAR值时，尾部的空格被删除掉。在存储或检索过程中不进行大小写转换。

VARCHAR列中的值为可变长字符串。长度可以指定为0到65,535之间的值。(VARCHAR的最大有效长度由最大行大小和使用的字符集确定。整体最大长度是65,532字节）。

同CHAR对比，VARCHAR值保存时只保存需要的字符数，另加一个字节来记录长度(如果列声明的长度超过255，则使用两个字节)。

VARCHAR值保存时不进行填充。当值保存和检索时尾部的空格仍保留，符合标准SQL。

如果分配给CHAR或VARCHAR列的值超过列的最大长度，则对值进行裁剪以使其适合。如果被裁掉的字符不是空格，则会产生一条警告。如果裁剪非空格字符，则会造成错误(而不是警告)并通过使用严格SQL模式禁用值的插入。参见5.3.2节，“SQL服务器模式”。

下面的表显示了将各种字符串值保存到CHAR(4)和VARCHAR(4)列后的结果，说明了CHAR和VARCHAR之间的差别：

值	CHAR(4)	存储需求	VARCHAR(4)	存储需求
''	' '	4个字节	''	1个字节
'ab'	'ab '	4个字节	'ab '	3个字节
'abcd'	'abcd'	4个字节	'abcd'	5个字节
'abcdefgh'	'abcd'	4个字节	'abcd'	5个字节

请注意上表中最后一行的值只适用不使用严格模式时；如果MySQL运行在严格模式，超过列长度不的值不保存，并且会出现错误。

从CHAR(4)和VARCHAR(4)列检索的值并不总是相同，因为检索时从CHAR列删除了尾部的空格。通过下面的例子说明该差别：

```
mysql> CREATE TABLE vc (v VARCHAR(4), c CHAR(4));

Query OK, 0 rows affected (0.02 sec)


mysql> INSERT INTO vc VALUES ('ab  ', 'ab  ');

Query OK, 1 row affected (0.00 sec)


mysql> SELECT CONCAT(v, '+'), CONCAT(c, '+') FROM vc;

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

| CONCAT(v, '+') | CONCAT(c, '+') |

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

| ab  +          | ab+            |
```

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

1 row in set (0.00 sec)
```

根据分配给列的字符集校对规则对CHAR和VARCHAR列中的值进行排序和比较。

请注意所有MySQL校对规则属于PADSPACE类。这说明在MySQL中的所有CHAR和VARCHAR值比较时不需要考虑任何尾部空格。例如:

```
mysql> CREATE TABLE names (myname CHAR(10), yourname VARCHAR(10));

Query OK, 0 rows affected (0.09 sec)


mysql> INSERT INTO names VALUES ('Monty ', 'Monty ');

Query OK, 1 row affected (0.00 sec)


mysql> SELECT myname = 'Monty  ', yourname = 'Monty  ' FROM names;

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

| myname = 'Monty  ' | yourname = 'Monty  ' |

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

|                  1 |                    1 |

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

1 row in set (0.00 sec)
```

请注意所有MySQL版本均如此，并且它不受SQL服务器模式的影响。

对于尾部填充字符被裁剪掉或比较时将它们忽视掉的情形，如果列的索引需要唯一的值，在列内插入一个只是填充字符数不同的值将会造成复制键值错误。

CHAR BYTE是CHAR BINARY的别名。这是为了保证兼容性。

ASCII属性为CHAR列分配latin1字符集。UNICODE属性分配ucs2字符集。

11.4.2. BINARY和VARBINARY类型

BINARY和VARBINARY类类似于CHAR和VARCHAR，不同的是它们包含二进制字符串而不要非二进制字符串。也就是说，它们包含字节字符串而不是字符字符串。这说明它们没有字符集，并且排序和比较基于列值字节的数值值。

BINARY和VARBINARY允许的最大长度一样，如同CHAR和VARCHAR，不同的是BINARY和VARBINARY的长度是字节长度而不是字符长度。

BINARY和VARBINARY数据类型不同于CHAR BINARY和VARCHAR BINARY数据类型。对于后一种类型，BINARY属性不会将列视为二进制字符串列。相反，它致使使用列字符集的二元 校对规则，并且列自身包含非二进制字符字符串而不是二进制字节字符串。例如CHAR(5) BINARY被视为CHAR(5) CHARACTER SET latin1 COLLATE latin1_bin，假定默认字符集是latin1。这不同于BINARY(5)，它保存5字节二进制字符串，没有字符集或 校对规则。

当保存BINARY值时，在它们右边填充值以达到指定长度。填充值是0x00(零字节)。插入值时在右侧添加0x00 on，并且选择时不删除尾部的字节。比较时所有字节很重要，包括ORDER BY和DISTINCT操作。比较时0x00字节和空格是不同的，0x00<空格。

例如：对于一个BINARY(3)列，当插入时 'a' 变为 'a \0'。'a\0'插入时变为'a\0\0'。当选择时两个插入的值均不更改。

对于VARBINARY，插入时不填充字符，选择时不裁剪字节。比较时所有字节很重要，包括ORDER BY和DISTINCT操作。比较时0x00字节和空格是不同的，0x00<空格。

对于尾部填充字符被裁剪掉或比较时将它们忽视掉的情形，如果列的索引需要唯一的值，在列内插入一个只是填充字符数不同的值将会造成复制键值错误。

如果你计划使用这些数据类型来保存二进制数据并且需要检索的值与保存的值完全相同，应考虑前面所述的填充和裁剪特征。下面的例子说明了用0x00填充的BINARY值如何影响列值比较：

```
mysql> CREATE TABLE t (c BINARY(3));

Query OK, 0 rows affected (0.01 sec)


mysql> INSERT INTO t SET c = 'a';

Query OK, 1 row affected (0.01 sec)


mysql> SELECT HEX(c), c = 'a', c = 'a\0\0' from t;

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

| HEX(c) | c = 'a' | c = 'a\0\0' |

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

| 610000 |       0 |           1 |

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

1 row in set (0.09 sec)
```

如果检索的值必须与指定进行存储而没有填充的值相同，最好使用BLOB数据类型。

创建表时，MySQL可以默默更改BINARY或VARBINARY列的类型。参见13.1.5.1节，“沉寂的列规格变更”。

11.4.3. BLOB和TEXT类型

BLOB是一个二进制大对象，可以容纳可变数量的数据。有4种BLOB类型：TINYBLOB、BLOB、MEDIUMBLOB和LONGBLOB。它们只是可容纳值的最大长度不同。

有4种TEXT类型：TINYTEXT、TEXT、MEDIUMTEXT和LONGTEXT。这些对应4种BLOB类型，有相同的最大长度和存储需求。

参见11.5节，“列类型存储需求”。

BLOB 列被视为二进制字符串(字节字符串)。TEXT列被视为非二进制字符串(字符字符串)。BLOB列没有字符集，并且排序和比较基于列值字节的数值值。TEXT列有一个字符集，并且根据字符集的 校对规则对值进行排序和比较。

在TEXT或BLOB列的存储或检索过程中，不存在大小写转换。

当未运行在严格模式时，如果你为BLOB或TEXT列分配一个超过该列类型的最大长度的值值，值被截取以保证适合。如果截掉的字符不是空格，将会产生一条警告。使用严格SQL模式，会产生错误，并且值将被拒绝而不

是截取并给出警告。参见5.3.2节，“SQL服务器模式”。

在大多数方面，可以将BLOB列视为能够足够大的VARBINARY列。同样，可以将TEXT列视为VARCHAR列。BLOB和TEXT在以下几个方面不同于VARBINARY和VARCHAR：

- 当保存或检索BLOB和TEXT列的值时不删除尾部空格。(这与VARBINARY和VARCHAR列相同）。

请注意比较时将用空格对TEXT进行扩充以适合比较的对象，正如CHAR和VARCHAR。

- 对于BLOB和TEXT列的索引，必须指定索引前缀的长度。对于CHAR和VARCHAR，前缀长度是可选的。参见7.4.3节，“列索引”。

- BLOB和TEXT列不能有 默认值。

LONG和LONG VARCHAR对应MEDIUMTEXT数据类型。这是为了保证兼容性。如果TEXT列类型使用BINARY属性，将为列分配列字符集的二元 校对规则。

MySQL连接程序/ODBC将BLOB值定义为LONGVARBINARY，将TEXT值定义为LONGVARCHAR。

由于BLOB和TEXT值可能会非常长，使用它们时可能遇到一些约束：

- 当排序时只使用该列的前max_sort_length个字节。max_sort_length的 默认值是1024；该值可以在启动**mysqld**服务器时使用--max_sort_length选项进行更改。参见5.3.3节，“服务器系统变量”。

运行时增加max_sort_length的值可以在排序或组合时使更多的字节有意义。任何客户端可以更改其会话max_sort_length变量的值：

```
mysql> SET max_sort_length = 2000;

mysql> SELECT id, comment FROM tbl_name

    -> ORDER BY comment;
```

当你想要使超过max_sort_length的字节有意义，对含长值的BLOB或TEXT列使用GROUP BY或ORDER BY的另一种方式是将列值转换为固定长度的对象。标准方法是使用SUBSTRING函数。例如，下面的语句对comment列的2000个字节进行排序：

```
mysql> SELECT id, SUBSTRING(comment,1,2000) FROM tbl_name

    -> ORDER BY SUBSTRING(comment,1,2000);
```

- BLOB或TEXT对象的最大大小由其类型确定，但在客户端和服务器之间实际可以传递的最大值由可用内存数量和通信缓存区大小确定。你可以通过更改max_allowed_packet变量的值更改消息缓存区的大小，但必须同时修改服务器和客户端程序。例如，可以使用 **mysql**和**mysqldump**来更改客户端的max_allowed_packet值。参见7.5.2节，“调节服务器参数”、8.3节，“mysql：MySQL命令行工具”和8.8节，“mysqldump：数据库备份程序”。

每个BLOB或TEXT值分别由内部分配的对象表示。这与其它列类型形成对比，后者是当打开表时为每1列分配存储引擎。

11.4.4. ENUM类型

ENUM是一个字符串对象，其值来自表创建时在列规定中显式枚举的一列值。

在某些情况下，ENUM值也可以为空字符串(")或NULL：

- 如果你将一个非法值插入ENUM(也就是说，允许的值列之外的字符串)，将插入空字符串以作为特殊错误值。该字符串与"普通"空字符串不同，该字符串有数值值0。后面有详细讨论。

- 如果将ENUM列声明为允许NULL，NULL值则为该列的一个有效值，并且 默认值为NULL。如果ENUM列被声明为NOT NULL，其默认值为允许的值列的第1个元素。

每个枚举值有一个索引:

- 来自列规定的允许的值列中的值从1开始编号。

- 空字符串错误值的索引值是0。这说明你可以使用下面的SELECT语句来找出分配了非法ENUM值的行:

```
mysql> SELECT * FROM tbl_name WHERE enum_col=0;
```

- NULL值的索引是NULL。

例如，定义为ENUM的列('one'，'two'，'three')可以有下面所示任何值。还显示了每个值的索引:

值	索引
NULL	NULL
''	0
'one'	1
'two'	2
'three'	3

枚举最多可以有65,535个元素。

当创建表时，ENUM成员值的尾部空格将自动被删除。

当检索时，保存在ENUM列的值使用列定义中所使用的大小写来显示。请注意可以为ENUM列分配字符集和 校对规则。对于二进制或大小写敏感的校对规则，当为列分配值时应考虑大小写。

如果在数值上下文中检索一个ENUM值，将返回列值的索引。例如，你可以这样从ENUM列搜索数值值:

```
mysql> SELECT enum_col+0 FROM tbl_name;
```

如果将一个数字保存到ENUM列，数字被视为索引，并且保存的值是该索引对应的枚举成员。(但是，这不适合LOAD DATA，它将所有输入视为字符串）。不建议使用类似数字的枚举值来定义一个ENUM列，因为这很容易引起混淆。例如，下面的列含有字符串值'0'、'1'和'2'的枚举成员，但数值索引值为1、2和3:

```
numbers ENUM('0','1','2')
```

根据枚举成员在列定义中列出的顺序对ENUM值进行排序。(换句话说，ENUM值根据索引编号进行排序）。例如，对于ENUM('a'，'b')，'a'排在'b'前面，但对于ENUM('b'，'a')，'b'排在'a'前面。空字符串排在非空字符串前面，并且NULL值排在所有其它枚举值前面。要想防止意想不到的结果，按字母顺序规定ENUM列。还可以使用GROUP BY CAST(col AS CHAR)或GROUP BY CONCAT(col)来确保按照词汇对列进行排序而不是用索引数字。

如果你想要确定一个ENUM列的所有可能的值，使用SHOW COLUMNS FROM *tbl_name* LIKE *enum_col*，并解析输出中第2列的ENUM定义。

11.4.5. SET类型

SET是一个字符串对象，可以有零或多个值，其值来自表创建时规定的允许的一列值。指定包括多个SET成员

的SET列值时各成员之间用逗号(‘,’)间隔开。这样SET成员值本身不能包含逗号。

例如，指定为SET('one', 'two') NOT NULL的列可以有下面的任何值：

```
''
'one'
'two'
'one,two'
```

SET最多可以有64个不同的成员。

当创建表时，SET成员值的尾部空格将自动被删除。

当检索时，保存在SET列的值使用列定义中所使用的大小写来显示。请注意可以为SET列分配字符集和 校对规则。对于二进制或大小写敏感的校对规则，当为列分配值时应考虑大小写。

MySQL用数字保存SET值，所保存值的低阶位对应第1个SET成员。如果在数值上下文中检索一个SET值，检索的值的位设置对应组成列值的SET成员。例如，你可以这样从一个SET列检索数值值：

```
mysql> SELECT set_col+0 FROM tbl_name;
```

如果将一个数字保存到SET列中，数字中二进制表示中的位确定了列值中的SET成员。对于指定为SET('a','b','c','d')的列，成员有下面的十进制和二进制值：

SET成员	十进制值	二进制值
'a'	1	0001
'b'	2	0010
'c'	4	0100
'd'	8	1000

如果你为该列分配一个值9，其二进制形式为1001，因此第1个和第4个SET值成员'a'和'd'被选择，结果值为'a,d'。

对于包含多个SET元素的值，当插入值时元素所列的顺序并不重要。在值中一个给定的元素列了多少次也不重要。当以后检索该值时，值中的每个元素出现一次，根据表创建时指定的顺序列出元素。例如，假定某个列指定为SET('a','b','c','d')：

```
mysql> CREATE TABLE myset (col SET('a', 'b', 'c', 'd'));
```

插入值'a,d'、'd,a'、'a,d,d'、'a,d,a'和'd,a,d':

```
mysql> INSERT INTO myset (col) VALUES
-> ('a,d'), ('d,a'), ('a,d,a'), ('a,d,d'), ('d,a,d');
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

当检索时所有这些值显示为 'a,d'：

```
mysql> SELECT col FROM myset;
+------+
| col  |
```

```
+------+
| a,d  |
| a,d  |
| a,d  |
| a,d  |
| a,d  |
+------+
5 rows in set (0.04 sec)
```

如果将SET列设置为一个不支持的值，则该值被忽略并发出警告：

```
mysql> INSERT INTO myset (col) VALUES ('a,d,d,s');
Query OK, 1 row affected, 1 warning (0.03 sec)

mysql> SHOW WARNINGS;
+---------+------+-----------------------------------------+
| Level   | Code | Message                                 |
+---------+------+-----------------------------------------+
| Warning | 1265 | Data truncated for column 'col' at row 1 |
+---------+------+-----------------------------------------+
1 row in set (0.04 sec)

mysql> SELECT col FROM myset;
+------+
| col  |
+------+
| a,d  |
| a,d  |
| a,d  |
| a,d  |
| a,d  |
| a,d  |
+------+
6 rows in set (0.01 sec)
```

SET值按数字顺序排序。NULL值排在非NULL SET值的前面。

通常情况，可以使用FIND_IN_SET()函数或LIKE操作符搜索SET值：

```
mysql> SELECT * FROM tbl_name WHERE FIND_IN_SET('value',set_col)>0;
mysql> SELECT * FROM tbl_name WHERE set_col LIKE '%value%';
```

第1个语句找出*SET_col*包含*value* set成员的行。第2个类似，但有所不同：它在其它地方找出*set_col*包

含*value*的行，甚至是在另一个SET成员的子字符串中。

下面的语句也是合法的：

```
mysql> SELECT * FROM tbl_name WHERE set_col & 1;

mysql> SELECT * FROM tbl_name WHERE set_col = 'val1,val2';
```

第1个语句寻找包含第1个set成员的值。第2个语句寻找一个确切匹配的值。应注意第2类的比较。将set值与'*val1*,*val2*'比较返回的结果与同'*val2*,*val1*'比较返回的结果不同。指定值时的顺序应与在列定义中所列的顺序相同。

如果想要为SET列确定所有可能的值，使用SHOW COLUMNS FROM *tbl_name* LIKE *set_col*并解析输出中第2列的SET定义。

11.5. 列类型存储需求

根据类别列出了MySQL支持的每个列类型的存储需求。

MyISAM表中行的最大大小为65,534字节。每个BLOB和TEXT列 账户只占其中的5至9个字节。

如果MyISAM表包括变长列类型，记录格式也是可变长度。当创建表时，在某些条件下，MySQL可以将一个列从变长类型改为固定长度的类型或反之亦然。详细信息参见13.1.5.1节，“沉寂的列规格变更”。

数值类型存储需求

列类型	存储需求
TINYINT	1个字节
SMALLINT	2个字节
MEDIUMINT	3个字节
INT, INTEGER	4个字节
BIGINT	8个字节
FLOAT(*p*)	如果0 <= *p* <= 24为4个字节，如果25 <= *p* <= 53为8个字节
FLOAT	4个字节
DOUBLE [PRECISION], item REAL	8个字节
DECIMAL(*M*,*D*), NUMERIC(*M*,*D*)	变长；参见下面的讨论
BIT(*M*)	大约(*M*+7)/8个字节

DECIMAL(和NUMERIC)的存储需求与具体版本有关：

使用二进制格式将9个十进制(基于10)数压缩为4个字节来表示DECIMAL列值。每个值的整数和分数部分的存储分别确定。每个9位数的倍数需要4个字节，并且“剩余的”位需要4个字节的一部分。下表给出了超出位数的存储需求：

剩余的位数	字节数目
0	0
1	1
2	1
3	2

4	2
5	3
6	3
7	4
8	4
9	4

日期和时间类型的存储需求

列类型	存储需求
DATE	3个字节
DATETIME	8个字节
TIMESTAMP	4个字节
TIME	3个字节
YEAR	1个字节

字符串类型的存储需求

列类型	存储需求
CHAR(*M*)	*M*个字节，0 <= *M* <= 255
VARCHAR(*M*)	*L*+1个字节，其中*L* <= *M* 且0 <= *M* <= 65535(参见下面的注释)
BINARY(*M*)	*M*个字节，0 <= *M* <= 255
VARBINARY(*M*)	*L*+1个字节，其中*L* <= *M* 且0 <= *M* <= 255
TINYBLOB, TINYTEXT	*L*+1个字节，其中$L < 2^8$
BLOB, TEXT	*L*+2个字节，其中$L < 2^{16}$
MEDIUMBLOB, MEDIUMTEXT	*L*+3个字节，其中$L < 2^{24}$
LONGBLOB, LONGTEXT	*L*+4个字节，其中$L < 2^{32}$
ENUM('*value1*','*value2*',...)	1或2个字节，取决于枚举值的个数(最多65,535个值)
SET('*value1*','*value2*',...)	1、2、3、4或者8个字节，取决于set成员的数目(最多64个成员)

VARCHAR、BLOB和TEXT类是变长类型。每个类型的存储需求取决于列值的实际长度(用前面的表中的*L*表示)，而不是该类型的最大可能的大小。例如，VARCHAR(10)列可以容纳最大长度为10的字符串。实际存储需求是字符串(*L*)的长度，加上一个记录字符串长度的字节。对于字符串'abcd'，*L*是4，存储需要5个字节。

对于CHAR、VARCHAR和TEXT类型，前面的表中的值*L*和*M*应解释为字符数目，并且列定义中的这些类型的长度表示字符数目。例如，要想保存一个TINYTEXT值需要*L*字符+ 1个字节。

要想计算用于保存具体CHAR、VARCHAR或者TEXT列值的字节数，需要考虑该列使用的字符集。在具体情况中，当使用Unicode时，必须记住所有Unicode字符使用相同的字节数。为了细分用于不同类Unicode字符使用的存储，参见10.5节，“Unicode支持”。

注释：VARCHAR列的有效最大长度为65,532字符。

NDBCLUSTER引擎只支持固定宽度的列。这说明MySQL簇中的表中的VARCHAR列的行为如同类型CHAR(不同的是每个记录仍然有一个额外字节空间)。例如，在Cluster表中，声明为VARCHAR(100)的列中的每个记录存储时将占用101个字节，无论实际存储的记录中的字符串的长度为多少。

BLOB和TEXT类需要 1、2、3或者4个字节来记录列值的长度，取决于该类的最大可能的长度。参见11.4.3节，“BLOB和TEXT类型”。

在NDB Cluster存储引擎中，TEXT和BLOB列的实施是不同的，其中TEXT列中的每个记录由两个单独部分组成。一个是固定大小(256字节)，并且实际上保存在原表中。另一个包括超出256字节的任何数据，保存在隐含的表中。第2个表中的记录总是2,000字节长。这说明如果*size*<= 256，TEXT列的大小为256(其中*size*表示记录的大小)；否则，大小是256 +*size*+(2000–(*size*–256)%2000)。

ENUM对象的大小由不同的枚举值的数目确定。枚举用一个字节，可以有255个可能的值。当枚举的值位于256和65,535之间时，用两个字节。参见11.4.4节，“ENUM类型”。

SET对象的大小由不同的set成员的数量确定。如果set大小是*N*，对象占(*N*+7)/8个字节，四舍五入到1、2、3、4或者8个字节。SET最多可以有64个成员。参见11.4.5节，“SET类型”。

11.6. 选择正确的列类型

为了优化存储，在任何情况下均应使用最精确的类型。例如，如果列的值的范围为从1到99999，若使用整数，则MEDIUMINT UNSIGNED是好的类型。在所有可以表示该列值的类型中，该类型使用的存储最少。

用精度为65位十进制数(基于10)对DECIMAL 列进行所有基本计算(+、-、*、/)。参见11.1.1节，“数值类型概述”。

使用双精度操作对DECIMAL值进行计算。如果准确度不是太重要或如果速度为最高优先级，DOUBLE类型即足够了。为了达到高精度，可以转换到保存在BIGINT中的定点类型。这样可以用64位整数进行所有计算，根据需要将结果转换回浮点值。

11.7. 使用来自其他数据库引擎的列类型

为了使用由其它卖方编写的SQL执行代码，MySQL按照下表所示对列类型进行映射。通过这些映射，可以很容易地从其它数据库引擎将表定义导入到MySQL中：

其它卖方类型	**MySQL**类型
BOOL,	TINYINT
BOOLEAN	TINYINT
CHAR VARYING(*M*)	VARCHAR(*M*)
DEC	DECIMAL
FIXED	DECIMAL
FLOAT4	FLOAT
FLOAT8	DOUBLE
INT1	TINYINT
INT2	SMALLINT
INT3	MEDIUMINT
INT4	INT
INT8	BIGINT
LONG VARBINARY	MEDIUMBLOB
LONG VARCHAR	MEDIUMTEXT
LONG	MEDIUMTEXT
MIDDLEINT	MEDIUMINT
NUMERIC	DECIMAL

在创建表时对列类型进行映射，然后原来的类型定义被丢弃。如果你使用其它卖方的类型创建一个表，然后执行DESCRIBE *tbl_name*语句，MySQL使用等效的MySQL类型来报告表的结构。例如：

```
mysql> CREATE TABLE t (a BOOL, b FLOAT8, c LONG, d NUMERIC);

Query OK, 0 rows affected (0.08 sec)


mysql> DESCRIBE t;

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

| Field | Type          | Null | Key | Default | Extra |

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

| a     | tinyint(1)    | YES  |     | NULL    |       |

| b     | double        | YES  |     | NULL    |       |

| c     | mediumtext    | YES  |     | NULL    |       |

| d     | decimal(10,0) | YES  |     | NULL    |       |

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

4 rows in set (0.00 sec)
```

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第12章：函数和操作符

目录

12.1. 操作符

12.1.1. 操作符优先级
12.1.2. 圆括号
12.1.3. 比较函数和操作符
12.1.4. 逻辑操作符

12.2. 控制流程函数
12.3. 字符串函数

12.3.1. 字符串比较函数

12.4. 数值函数

12.4.1. 算术操作符
12.4.2. 数学函数

12.5. 日期和时间函数
12.6. MySQL使用什么日历？
12.7. 全文搜索功能

12.7.1. 布尔全文搜索
12.7.2. 全文搜索带查询扩展
12.7.3. 全文停止字
12.7.4. 全文限定条件
12.7.5. 微调MySQL全文搜索

12.8. Cast函数和操作符
12.9. 其他函数

12.9.1. 位函数
12.9.2. 加密函数
12.9.3. 信息函数
12.9.4. 其他函数

12.10. 与GROUP BY子句同时使用的函数和修改程序

12.10.1. GROUP BY（聚合）函数
12.10.2. GROUP BY修改程序
12.10.3. 具有隐含字段的GROUP BY

在SQL 语句中，表达式可用于一些诸如SELECT语句的ORDER BY 或HAVING子句、SELECT、 DELETE或UPDATE语句的WHERE 子句或 SET语句之类的地方。使用文本值、column值、NULL值、函数、 操作符来书写表达式。 本章叙述了可用于书写MySQL表达式的函数和操作符。

除非在文档编制中对一个函数或操作符另有指定的情况外，一个包含NULL 的表达式通常产生一个NULL 值。

注释： 在默认状态下, 在函数和紧随其后的括号之间不得存在空格。这能帮助 MySQL 分析程序区分一些同函数名相同的函数调用以及表或列。不过，函数自变量周围允许有空格出现。

可以通过选择--sql-mode=IGNORE_SPACE来打开MySQL服务器的方法使服务器接受函数名后的空格。 个人客户端程序可通过选择mysql_real_connect()的CLIENT_IGNORE_SPACE 实现这一状态。在以上两种情况中， 所有的函数名都成为保留字。请参见5.3.2节，“SQL服务器模式”.

为节省时间，本章中对大多数例子使用简写形式展示了 **mysql** 程序的输出结果。 对于以下格式的举例展示：

mysql> **SELECT MOD(29,9);**

```
+-----------+
| mod(29,9) |
+-----------+
|         2 |
+-----------+
1 rows in set (0.00 秒)
```

使用如下格式进行代替：

mysql> **SELECT MOD(29,9);**

```
    -> 2
```

12.1. 操作符

12.1.1. 操作符优先级
12.1.2. 圆括号
12.1.3. 比较函数和操作符
12.1.4. 逻辑操作符

12.1.1. 操作符优先级

以下列表显示了操作符优先级的由低到高的顺序。排列在同一行的操作符具有相同的优先级。

:=

||, OR, XOR

&&, AND

NOT

BETWEEN, CASE, WHEN, THEN, ELSE

=, <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN

|

&

<<, >>

-, +

*, /, DIV, %, MOD

^

- (一元减号), ~ (一元比特反转)

!

BINARY, COLLATE

注释：假如 HIGH_NOT_PRECEDENCE SQL 模式被激活，则 NOT 的优先级同 the ！操作符相同。请参见5.3.2节，“SQL服务器模式”。

12.1.2. 圆括号

- (...)

  使用括弧来规定表达式的运算顺序，例如：

```
mysql> SELECT 1+2*3;
        -> 7
mysql> SELECT (1+2)*3;
        -> 9
```

12.1.3. 比较函数和操作符

比较运算产生的结果为1(TRUE)、0 (FALSE)或 NULL。这些运算可用于数字和字符串。根据需要，字符串可自动转换为数字，而数字也可自动转换为字符串。

本章中的一些函数 (如LEAST()和GREATEST()) 的所得值不包括 1 (TRUE)、 0 (FALSE)和 NULL。然而，其所得值乃是基于按照下述规则运行的比较运算：

MySQL按照以下规则进行数值比较：

- 若有一个或两个参数为 NULL，除非NULL-safe <=> 等算符，则比较运算的结果为NULL。
- 若同一个比较运算中的两个参数都是字符串，则按照字符串进行比较。
- 若两个参数均为整数，则按照整数进行比较。
- 十六进制值在不需要作为数字进行比较时，则按照二进制字符串进行处理。
- 假如参数中的一个为 TIMESTAMP 或 DATETIME 列，而其它参数均为常数， 则在进行比较前将常数转为timestamp。这样做的目的是为了使ODBC的进行更加顺利。 注意，这不适合IN()中的参数!为了更加可靠，在进行对比时通常使用完整的 datetime/date/time字符串。
- 在其它情况下，参数作为浮点数进行比较。

在默认状态下，字符串比较不区分大小写，并使用现有字符集(默认为cp1252 Latin1，同时对英语也适合)。

为了进行比较，可使用CAST()函数将某个值转为另外一种类型。 使用CONVERT()将字符串值转为不同的字符集。请参见12.8节，“Cast函数和操作符”。

以下例子说明了比较运算中将字符串转为数字的过程：

mysql> **SELECT 1 > '6x';**

-> 0

mysql> **SELECT 7 > '6x';**

-> 1

mysql> **SELECT 0 > 'x6';**

-> 0

mysql> **SELECT 0 = 'x6';**

-> 1

注意，在将一个字符串列同一个数字进行比较时， MySQL 不能使用列中的索引进行快速查找。假如*str_col* 是一个编入索引的字符串列，则在以下语句中，索引不能执行查找功能：

SELECT * FROM *tbl_name* WHERE *str_col*=1;

其原因是许多不同的字符串都可被转换为数值 1: '1'、 ' 1'、 '1a'、 ……

- =

等于：

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

- <=>

NULL-safe equal.这个操作符和=操作符执行相同的比较操作，不过在两个操作码均为NULL时，其所得值为1而不为NULL，而当一个操作码为NULL时，其所得值为0而不为NULL。

mysql> **SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;**

-> 1, 1, 0

mysql> **SELECT 1 = 1, NULL = NULL, 1 = NULL;**

-> 1, NULL, NULL

- <> !=

不等于:

mysql> **SELECT '.01' <> '0.01';**

-> 1

mysql> **SELECT .01 <> '0.01';**

-> 0

mysql> **SELECT 'zapp' <> 'zappp';**

-> 1

- <=

小于或等于:

mysql> **SELECT 0.1 <= 2;**

-> 1

- <

小于:

mysql> **SELECT 2 < 2;**

-> 0

- >=

大于或等于:

mysql> **SELECT 2 >= 2;**

-> 1

- >

大于:

mysql> **SELECT 2 > 2;**

-> 0

- IS *boolean_value* IS NOT *boolean_value*

根据一个布尔值来检验一个值，在这里，布尔值可以是TRUE、FALSE或UNKNOWN。

mysql> **SELECT 1 IS TRUE, 0 IS FALSE, NULL IS UNKNOWN;**

-> 1, 1, 1

mysql> **SELECT 1 IS NOT UNKNOWN, 0 IS NOT UNKNOWN, NULL IS NOT UNKNOWN;**

-> 1, 1, 0

- IS NULL IS NOT NULL

检验一个值是否为 NULL。

mysql> **SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;**

-> 0, 0, 1

mysql> **SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;**

-> 1, 1, 0

为了能够顺利的使用ODBC 程序工作，在使用IS NULL时，MySQL支持一下额外特性：

- 在一个值产生后，立即运行一个以下格式的语句，从而找到最新AUTO_INCREMENT 值的所在行：

o SELECT * FROM *tbl_name* WHERE *auto_col* IS NULL

当设置SQL_AUTO_IS_NULL=0时，这项操作无法运行。请参见13.5.3节，“SET语法”。

- 对于命名为NOT NULL 的DATE 和DATETIME列，可通过使用如下的 语句找到特定日期 '0000-00-00'：

o SELECT * FROM *tbl_name* WHERE *date_column* IS NULL

运行这一步需要使用一些 ODBC 应用软件，因为 ODBC本身不支持 一个 '0000-00-00'的时间值。

- *expr* BETWEEN *min* AND *max*

假如*expr*大于或等于 *min* 且*expr* 小于或等于*max*, 则BETWEEN 的返回值为1,或是0。若所有参数都是同一类型，则上述关系相当于表达式 (*min* <= *expr* AND *expr* <= *max*)。其它类型的转换根据本章开篇所述规律进行，且适用于3种参数中任意一种。

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*

这相当于NOT(*expr* BETWEEN *min* AND *max*)。

· COALESCE(*value*,...)

返回值为列表当中的第一个非 NULL值，在没有非NULL 值得情况下返回值为 NULL 。

mysql> **SELECT COALESCE(NULL,1);**

-> 1

mysql> **SELECT COALESCE(NULL,NULL,NULL);**

-> NULL

· GREATEST(*value1*,*value2*,...)

当有2或多个参数时，返回值为最大(最大值的)参数。比较参数所依据的规律同LEAST()相同。

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'

在没有自变量为NULL的情况下，GREATEST()的返回值为NULL。

· *expr* IN (*value*,...)

- 若*expr* 为IN列表中的任意一个值，则其返回值为 1 , 否则返回值为0。假如所有的值都是常数，则其计算和分类根据 *expr* 的类型进行。这时，使用二分搜索来搜索信息。如IN值列表全部由常数组成，则意味着IN 的速度非常之快。如*expr* 是一个区分大小写的字符串表达式，则字符串比较也按照区分大小写的方式进行。

mysql> **SELECT 2 IN (0,3,5,'wefwf');**

-> 0

mysql> **SELECT 'wefwf' IN (0,3,5,'wefwf');**

-> 1

IN 列表中所列值的个数仅受限于 max_allowed_packet 值。

为了同SQL 标准相一致，在左侧表达式为NULL的情况下，或是表中找不到匹配项或是表中一个表达式为NULL的情况下，IN的返回值均为NULL。

IN() 语构也可用书写某些类型的子查询。请参见13.2.8.3节，“使用ANY、IN和SOME进行子查询”。

· *expr* NOT IN (*value*,...)

这与NOT (*expr* IN (*value*,...))相同。

- ISNULL(*expr*)

如*expr* 为NULL，那么ISNULL() 的返回值为 1，否则返回值为 0。

mysql> **SELECT ISNULL(1+1);**

-> 0

mysql> **SELECT ISNULL(1/0);**

-> 1

使用= 的NULL 值对比通常是错误的。

ISNULL() 函数同 IS NULL比较操作符具有一些相同的特性。请参见有关IS NULL 的说明。

· INTERVAL(*N*,*N1*,*N2*,*N3*,...)

假如*N* < *N1*，则返回值为0；假如*N* < *N2* 等等，则返回值为1；假如*N* 为NULL，则返回值为 -1 。所有的参数均按照整数处理。为了这个函数的正确运行，必须满足 *N1* < *N2* < *N3* < ……< *Nn* 。其原因是使用了二分查找(极快速)。

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

· LEAST(*value1*,*value2*,...)

在有两个或多个参数的情况下， 返回值为最小 (最小值) 参数。用一下规则将自变量进行对比:

- 假如返回值被用在一个 INTEGER 语境中，或是所有参数均为整数值，则将其作为整数值进行比较。
- 假如返回值被用在一个 REAL语境中，或所有参数均为实值，则 将其作为实值进行比较。
- 假如任意一个参数是一个区分大小写的字符串，则将参数按照区分大小写的字符串进行比较。
- 在其它情况下，将参数作为区分大小写的字符串进行比较。

假如任意一个自变量为NULL，则 LEAST()的返回值为NULL 。

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'

注意，上面的转换规则在一些边界情形中会产生一些奇特的结果:

mysql> **SELECT CAST(LEAST(3600, 9223372036854775808.0) as SIGNED);**

-> -9223372036854775808

发生这种情况的原因是MySQL在整数语境中读取9223372036854775808.0。整数表示法不利于保存数值，因此

它包括一个带符号整数。

12.1.4. 逻辑操作符

在SQL中，所有逻辑 操作符的求值所得结果均为 TRUE、FALSE或 NULL (UNKNOWN)。在 MySQL中，它们体现为 1 (TRUE)、 0 (FALSE)和 NULL。其大多数都与不同的数据库SQL通用，然而一些服务器对TRUE的返回值可能是任意一个非零值。

- NOT !

逻辑 NOT。当操作数为0 时，所得值为 1 ；当操作数为非零值时，所得值为 0 ，而当操作数为NOT NULL时，所得的返回值为 NULL。

mysql> **SELECT NOT 10;**

-> 0

mysql> **SELECT NOT 0;**

-> 1

mysql> **SELECT NOT NULL;**

-> NULL

mysql> **SELECT ! (1+1);**

-> 0

mysql> **SELECT ! 1+1;**

-> 1

最后一个例子产生的结果为 1，原因是表达式的计算方式和(!1)+1相同。

- AND &&

逻辑AND。当所有操作数均为非零值、并且不为NULL时，计算所得结果为 1 ，当一个或多个操作数为0 时，所得结果为 0 ，其余情况返回值为 NULL 。

mysql> **SELECT 1 && 1;**

-> 1

mysql> **SELECT 1 && 0;**

-> 0

mysql> **SELECT 1 && NULL;**

-> NULL

mysql> **SELECT 0 && NULL;**

-> 0

mysql> **SELECT NULL && 0;**

-> 0

- OR ||

逻辑 OR。当两个操作数均为非 NULL值时，如有任意一个操作数为非零值，则结果为1，否则结果为0。当有一个操作数为NULL时，如另一个操作数为非零值，则结果为1，否则结果为 NULL 。假如两个操作数均为 NULL，则所得结果为 NULL。

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

逻辑XOR。当任意一个操作数为 NULL时，返回值为NULL。对于非 NULL 的操作数，假如一个奇数操作数为非零值，则计算所得结果为 1 ，否则为 0 。

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 的计算等同于 (a AND (NOT b)) OR ((NOT a)和 b)。

同样见 12.1.1节，“操作符优先级”。

12.2. 控制流程函数

- 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

在第一个方案的返回结果中， *value=compare-value*。而第二个方案的返回结果是第一种情况的真实结果。如果没有匹配的结果值，则返回结果为ELSE后的结果，如果没有ELSE 部分，则返回值为 NULL。

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

一个CASE表达式的默认返回值类型是任何返回值的相容集合类型，但具体情况视其所在语境而定。如果用在字符串语境中，则返回结果味字符串。如果用在数字语境中，则返回结果为十进制值、实值或整数值。

- IF(*expr1*,*expr2*,*expr3*)

如果 *expr1* 是TRUE (*expr1* <> 0 and *expr1* <> NULL)，则 IF()的返回值为*expr2*; 否则返回值则为 *expr3*。IF() 的返回值为数字值或字符串值，具体情况视其所在语境而定。

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'

如果*expr2* 或*expr3*中只有一个明确是 NULL，则IF() 函数的结果类型 为非NULL表达式的结果类型。

expr1 作为一个整数值进行计算，就是说，假如你正在验证浮点值或字符串值， 那么应该使用比较运算进行检验。

mysql> **SELECT IF(0.1,1,0);**

-> 0

mysql> **SELECT IF(0.1<>0,1,0);**

-> 1

在所示的第一个例子中，IF(0.1)的返回值为0，原因是 0.1 被转化为整数值，从而引起一个对 IF(0)的检验。这或许不是你想要的情况。在第二个例子中，比较检验了原始浮点值，目的是为了了解是否其为非零值。比较结

果使用整数。

IF() (这一点在其被储存到临时表时很重要) 的默认返回值类型按照以下方式计算:

表达式	返回值
expr2 或*expr3* 返回值为一个字符串。	字符串
expr2 或*expr3* 返回值为一个浮点值。	浮点
expr2 或 *expr3* 返回值为一个整数。	整数

假如*expr2* 和*expr3* 都是字符串，且其中任何一个字符串区分大小写，则返回结果是区分大小写。

- IFNULL(*expr1*,*expr2*)

假如*expr1* 不为 NULL，则 IFNULL() 的返回值为 *expr1*; 否则其返回值为 *expr2*。IFNULL()的返回值是数字或是字符串，具体情况取决于其所使用的语境。

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'

IFNULL(*expr1*,*expr2*)的默认结果值为两个表达式中更加"通用"的一个，顺序为STRING、 REAL或 INTEGER。假设一个基于表达式的表的情况， 或MySQL必须在内存储器中储存一个临时表中IFNULL()的返回值:

CREATE TABLE tmp SELECT IFNULL(1,'test') AS test;

在这个例子中，测试列的类型为 CHAR(4)。

- NULLIF(*expr1*,*expr2*)

如果*expr1* = *expr2* 成立，那么返回值为NULL，否则返回值为 *expr1*。这和CASE WHEN *expr1* = *expr2* THEN NULL ELSE *expr1* END相同。

mysql> **SELECT NULLIF(1,1);**

-> NULL

mysql> **SELECT NULLIF(1,2);**

-> 1

注意，如果参数不相等，则 MySQL 两次求得的值为 expr1 。

12.3. 字符串函数

12.3.1. 字符串比较函数

假如结果的长度大于 max_allowed_packet 系统变量的最大值时，字符串值函数的返回值为NULL。请参见7.5.2节，“调节服务器参数”。

对于在字符串位置操作的函数，第一个位置的编号为 1。

- ASCII(*str*)

返回值为字符串*str* 的最左字符的数值。假如*str*为空字符串，则返回值为 0 。假如*str* 为NULL，则返回值为NULL。 ASCII()用于带有从 0到255的数值的字符。

mysql> **SELECT ASCII('2');**

-> 50

mysql> **SELECT ASCII(2);**

-> 50

mysql> **SELECT ASCII('dx');**

-> 100

见 ORD()函数。

- BIN(*N*)

返回值为*N*的二进制值的字符串表示，其中 *N* 为一个longlong (BIGINT) 数字。这等同于 CONV(*N*,10,2)。假如*N* 为NULL，则返回值为 NULL。

mysql> **SELECT BIN(12);**

-> '1100'

- BIT_LENGTH(*str*)

返回值为二进制的字符串*str* 长度。

mysql> **SELECT BIT_LENGTH('text');**

-> 32

- CHAR(*N*,... [USING *charset*])

CHAR()将每个参数*N*理解为一个整数，其返回值为一个包含这些整数的代码值所给出的字符的字符串。NULL值被省略。

mysql> **SELECT CHAR(77,121,83,81,'76');**

-> 'MySQL'

mysql> **SELECT CHAR(77,77.3,'77.3');**

-> 'MMM'

大于 255的CHAR()参数被转换为多结果字符。 例如，CHAR(256) 相当于 CHAR(1,0), 而CHAR(256*256) 则相当于 CHAR(1,0,0)：

mysql> **SELECT HEX(CHAR(1,0)), HEX(CHAR(256));**

```
+----------------+----------------+
| HEX(CHAR(1,0)) | HEX(CHAR(256)) |
+----------------+----------------+
| 0100           | 0100           |
+----------------+----------------+
```

mysql> **SELECT HEX(CHAR(1,0,0)), HEX(CHAR(256*256));**

```
+------------------+--------------------+
| HEX(CHAR(1,0,0)) | HEX(CHAR(256*256)) |
+------------------+--------------------+
| 010000           | 010000             |
+------------------+--------------------+
```

CHAR()的返回值为一个二进制字符串。可选择使用USING语句产生一个给出的字符集中的字符串：

mysql> **SELECT CHARSET(CHAR(0x65)), CHARSET(CHAR(0x65 USING utf8));**

```
mysql> SELECT CHARSET(CHAR(0x65)), CHARSET(CHAR(0x65 USING utf8));
+---------------------+-------------------------------+
| CHARSET(CHAR(0x65)) | CHARSET(CHAR(0x65 USING utf8)) |
+---------------------+-------------------------------+
| binary              | utf8                          |
+---------------------+-------------------------------+
```

如果 USING已经产生，而结果字符串不符合给出的字符集，则会发出警告。 同样，如果严格的SQL模式被激活，则CHAR()的结果会成为 NULL。

- CHAR_LENGTH(*str*)

返回值为字符串*str* 的长度，长度的单位为字符。一个多字节字符算作一个单字符。对于一个包含五个二字节字符集, LENGTH()返回值为 10, 而CHAR_LENGTH()的返回值为5。

- CHARACTER_LENGTH(*str*)

CHARACTER_LENGTH()是CHAR_LENGTH()的同义词。

- COMPRESS(*string_to_compress*)

压缩一个字符串。这个函数要求 MySQL已经用一个诸如zlib的压缩库压缩过。 否则，返回值始终是NULL。UNCOMPRESS() 可将压缩过的字符串进行解压缩。

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

压缩后的字符串的内容按照以下方式存储:

- 空字符串按照空字符串存储。
- 非空字符串未压缩字符串的四字节长度进行存储(首先为低字节),后面是压缩字符串。如果字符串以空格结尾，就会在后加一个"."号,以防止当结果值是存储在CHAR或VARCHAR类型的字段列时,出现自动把结尾空格去掉的现象。(不推荐使用 CHAR 或VARCHAR 来存储压缩字符串。最好使用一个 BLOB 列代替）。

- CONCAT(*str1*,*str2*,...)

返回结果为连接参数产生的字符串。如有任何一个参数为NULL ，则返回值为 NULL。或许有一个或多个参数。 如果所有参数均为非二进制字符串，则结果为非二进制字符串。 如果自变量中含有任一二进制字符串，则结果为一个二进制字符串。一个数字参数被转化为与之相等的二进制字符串格式；若要避免这种情况，可使用显式类型 cast, 例如： SELECT CONCAT(CAST(int_col AS CHAR), char_col)

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() 代表 CONCAT With Separator ，是CONCAT()的特殊形式。 第一个参数是其它参数的分隔符。分隔符的位置放在要连接的两个字符串之间。分隔符可以是一个字符串，也可以是其它参数。如果分隔符为 NULL，则结果为 NULL。函数会忽略任何分隔符参数后的 NULL 值。

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'

CONCAT_WS()不会忽略任何空字符串。 (然而会忽略所有的 NULL）。

- CONV(*N*,*from_base*,*to_base*)

不同数基间转换数字。返回值为数字的*N*字符串表示，由*from_base*基转化为 *to_base* 基。如有任意一个参数为NULL，则返回值为 NULL。自变量 *N* 被理解为一个整数，但是可以被指定为一个整数或字符串。最小基数为 2 ，而最大基数则为 36。 If *to_base* 是一个负数，则 *N* 被看作一个带符号数。否则， *N* 被看作无符号数。 CONV() 的运行精确度为 64比特。

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*,...)

若*N* = 1，则返回值为 *str1* ，若*N* = 2，则返回值为 *str2* ，以此类推。 若*N* 小于1或大于参数的数目，则返回值为 NULL 。 ELT() 是 FIELD()的补数。

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*]])

返回值为一个字符串，其中对于*bits*值中的每个位组，可以得到一个 *on* 字符串，而对于每个清零比特位，可以得到一个*off* 字符串。*bits* 中的比特值按照从右到左的顺序接受检验 (由低位比特到高位比特)。字符串被分隔字符串分开(默认为逗号‘,’)，按照从左到右的顺序被添加到结果中。*number_of_bits* 会给出被检验的二进制位数 (默认为 64)。

mysql> **SELECT EXPORT_SET(5,'Y','N',',',4);**

-> 'Y,N,Y,N'

mysql> **SELECT EXPORT_SET(6,'1','0',',',10);**

-> '0,1,1,0,0,0,0,0,0,0'

- FIELD(*str*,*str1*,*str2*,*str3*,...)

返回值为*str1*, *str2*, *str3*,……列表中的*str* 指数。在找不到*str* 的情况下，返回值为 0 。

如果所有对于FIELD() 的参数均为字符串，则所有参数均按照字符串进行比较。如果所有的参数均为数字，则按照数字进行比较。否则，参数按照双倍进行比较。

如果*str* 为NULL，则返回值为0 ，原因是NULL不能同任何值进行同等比较。FIELD() 是ELT()的补数。

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*)

假如字符串*str* 在由*N* 子链组成的字符串列表*strlist* 中， 则返回值的范围在 1 到 *N* 之间 。一个字符串列表就是一个由一些被‘,’符号分开的自链组成的字符串。如果第一个参数是一个常数字符串，而第二个是type SET列，则 FIND_IN_SET() 函数被优化，使用比特计算。如果*str*不在*strlist* 或*strlist* 为空字符串，则返回值为 0 。如任意一个参数为NULL，则返回值为 NULL。 这个函数在第一个参数包含一个逗号(‘,’)时将无法正常运行。

mysql> **SELECT FIND_IN_SET('b','a,b,c,d');**

-> 2

- FORMAT(*X*,*D*)

将number *X*设置为格式 '#,###,###.##', 以四舍五入的方式保留到小数点后*D*位, 而返回结果为一个字符串。详见12.9.4节，“其他函数”.

- HEX(*N_or_S*)

如果*N_OR_S* 是一个数字，则返回一个 十六进制值 *N* 的字符串表示，在这里， *N* 是一个longlong (BIGINT)数。这相当于 CONV(N,10,16)。

如果*N_OR_S* 是一个字符串，则返回值为一个*N_OR_S*的十六进制字符串表示， 其中每个*N_OR_S* 里的每个字符被转化为两个十六进制数字。

mysql> **SELECT HEX(255);**

-> 'FF'

mysql> **SELECT 0x616263;**

-> 'abc'

mysql> **SELECT HEX('abc');**

-> 616263

- INSERT(*str*,*pos*,*len*,*newstr*)

返回字符串 *str*, 其子字符串起始于 *pos* 位置和长期被字符串 *newstr*取代的*len* 字符。 如果*pos* 超过字符串长度，则返回值为原始字符串。 假如*len*的长度大于其它字符串的长度，则从位置*pos*开始替换。若任何一个参数为null，则返回值为NULL。

mysql> **SELECT INSERT('Quadratic', 3, 4, 'What');**

-> 'QuWhattic'

mysql> **SELECT INSERT('Quadratic', -1, 4, 'What');**

-> 'Quadratic'

mysql> **SELECT INSERT('Quadratic', 3, 100, 'What');**

-> 'QuWhat'

这个函数支持多字节字元。

- INSTR(*str*,*substr*)

返回字符串 *str* 中子字符串的第一个出现位置。这和LOCATE()的双参数形式相同，除非参数的顺序被颠倒。

mysql> **SELECT INSTR('foobarbar', 'bar');**

-> 4

mysql> **SELECT INSTR('xbar', 'foobar');**

-> 0

这个函数支持多字节字元，并且只有当至少有一个参数是二进制字符串时区分大小写。

- LCASE(*str*)

LCASE() 是 LOWER()的同义词。

- LEFT(*str*,*len*)

返回从字符串*str* 开始的*len* 最左字符。

mysql> **SELECT LEFT('foobarbar', 5);**

-> 'fooba'

- LENGTH(*str*)

返回值为字符串*str* 的长度，单位为字节。一个多字节字符算作多字节。这意味着 对于一个包含5个2字节字符的字符串， LENGTH() 的返回值为 10, 而 CHAR_LENGTH()的返回值则为5。

mysql> **SELECT LENGTH('text');**

-> 4

- LOAD_FILE(*file_name*)

读取文件并将这一文件按照字符串的格式返回。 文件的位置必须在服务器上,你必须为文件制定路径全名，而且你还必须拥有FILE 特许权。文件必须可读取，文件容量必须小于 max_allowed_packet字节。

若文件不存在，或因不满足上述条件而不能被读取， 则函数返回值为 NULL。

mysql> **UPDATE** *tbl_name*

SET *blob_column*=LOAD_FILE('/tmp/picture')

WHERE id=1;

- LOCATE(*substr*,*str*) , LOCATE(*substr*,*str*,*pos*)

第一个语法返回字符串 *str*中子字符串*substr*的第一个出现位置。第二个语法返回字符串 *str*中子字符串*substr*的第一个出现位置, 起始位置在*pos*。如若*substr* 不在*str*中，则返回值为0。

mysql> **SELECT LOCATE('bar', 'foobarbar');**

-> 4

mysql> **SELECT LOCATE('xbar', 'foobar');**

-> 0

mysql> **SELECT LOCATE('bar', 'foobarbar',5);**

-> 7

这个函数支持多字节字元，并且只有当至少有一个参数是二进制字符串时区分大小写。

- LOWER(*str*)

返回字符串 *str* 以及所有根据最新的字符集映射表变为小写字母的字符 (默认为 cp1252 Latin1)。

mysql> **SELECT LOWER('QUADRATICALLY');**

-> 'quadratically'

这个函数支持多字节字元。

- LPAD(*str*,*len*,*padstr*)

返回字符串 *str*, 其左边由字符串*padstr* 填补到*len* 字符长度。假如*str* 的长度大于*len*, 则返回值被缩短至 *len* 字符。

mysql> **SELECT LPAD('hi',4,'??');**

-> '??hi'

mysql> **SELECT LPAD('hi',1,'??');**

-> 'h'

- LTRIM(*str*)

返回字符串 *str* ，其引导空格字符被删除。

mysql> **SELECT LTRIM(' barbar');**

-> 'barbar'

这个函数支持多字节字元。

- MAKE_SET(*bits*,*str1*,*str2*,...)

返回一个设定值 (一个包含被‘,’号分开的字字符串的字符串) ，由在*bits* 组中具有相应的比特的字符串组成。*str1* 对应比特 0, *str2* 对应比特1,以此类推。*str1*, *str2*, ...中的 NULL值不会被添加到结果中。

mysql> **SELECT MAKE_SET(1,'a','b','c');**

-> 'a'

mysql> **SELECT MAKE_SET(1 | 4,'hello','nice','world');**

-> 'hello,world'

mysql> **SELECT MAKE_SET(1 | 4,'hello','nice',NULL,'world');**

-> 'hello'

mysql> **SELECT MAKE_SET(0,'a','b','c');**

-> ''

- MID(*str*,*pos*,*len*)

MID(*str*,*pos*,*len*) 是 SUBSTRING(*str*,*pos*,*len*)的同义词。

- OCT(*N*)

返回一个 *N*的八进制值的字符串表示，其中 *N* 是一个longlong (BIGINT)数。这等同于CONV(N,10,8)。若*N* 为NULL ，则返回值为NULL。

mysql> **SELECT OCT(12);**

-> '14'

- OCTET_LENGTH(*str*)

OCTET_LENGTH() 是 LENGTH()的同义词。

- ORD(*str*)

若字符串*str* 的最左字符是一个多字节字符，则返回该字符的代码， 代码的计算通过使用以下公式计算其组成字节的数值而得出:

```
  (1st byte code)
+ (2nd byte code × 256)
+ (3rd byte code × 256²) ...
```

假如最左字符不是一个多字节字符，那么 ORD()和函数ASCII()返回相同的值。

mysql> **SELECT ORD('2');**

-> 50

- POSITION(*substr* IN *str*)

POSITION(*substr* IN *str*)是 LOCATE(*substr*,*str*)同义词。

- QUOTE(*str*)

引证一个字符串，由此产生一个在SQL语句中可用作完全转义数据值的结果。 返回的字符串由单引号标注，每例都带有单引号 (''')、 反斜线符号 ('\')、 ASCII NUL以及前面有反斜线符号的Control-Z 。如果自变量的值

为NULL, 则返回不带单引号的单词 “NULL”。

mysql> **SELECT QUOTE('Don\'t!');**

-> 'Don\'t!'

mysql> **SELECT QUOTE(NULL);**

-> NULL

- REPEAT(*str*,*count*)

返回一个由重复的字符串*str* 组成的字符串，字符串*str*的数目等于*count* 。 若 *count* <= 0,则返回一个空字符串。若*str* 或 *count* 为 NULL，则返回 NULL 。

mysql> **SELECT REPEAT('MySQL', 3);**

-> 'MySQLMySQLMySQL'

- REPLACE(*str*,*from_str*,*to_str*)

返回字符串*str* 以及所有被字符串*to_str*替代的字符串*from_str* 。

mysql> **SELECT REPLACE('www.mysql.com', 'w', 'Ww');**

-> 'WwWwWw.mysql.com'

这个函数支持多字节字元。

- REVERSE(*str*)

返回字符串 *str* ，顺序和字符顺序相反。

mysql> **SELECT REVERSE('abc');**

-> 'cba'

这个函数支持多字节字元。

- RIGHT(*str*,*len*)

从字符串*str* 开始，返回最右*len* 字符。

mysql> **SELECT RIGHT('foobarbar', 4);**

-> 'rbar'

这个函数支持多字节字元。

- RPAD(*str*,*len*,*padstr*)

返回字符串*str*, 其右边被字符串 *padstr*填补至*len* 字符长度。假如字符串*str* 的长度大于 *len*,则返回值被缩短到与 *len* 字符相同长度。

mysql> **SELECT RPAD('hi',5,'?');**

-> 'hi???'

mysql> **SELECT RPAD('hi',1,'?');**

-> 'h'

这个函数支持多字节字元。

- RTRIM(*str*)

返回字符串 *str* ，结尾空格字符被删去。

mysql> **SELECT RTRIM('barbar ');**

-> 'barbar'

这个函数支持多字节字元。

- SOUNDEX(*str*)

从*str*返回一个soundex字符串。 两个具有几乎同样探测的字符串应该具有同样的 soundex 字符串。一个标准的soundex 字符串的长度为4个字符，然而SOUNDEX() 函数会返回一个人以长度的字符串。 可使用结果中的SUBSTRING() 来得到一个标准 soundex 字符串。在*str*中，会忽略所有未按照字母顺序排列的字符。 所有不在A-Z范围之内的国际字母符号被视为元音字母。

mysql> **SELECT SOUNDEX('Hello');**

-> 'H400'

mysql> **SELECT SOUNDEX('Quadratically');**

-> 'Q36324'

注意：这个函数执行原始的Soundex算法，而非更加流行的加强版本(如D. Knuth所述)。其区别在于原始版本首先会删去元音，其次是重复，而加强版则首先删去重复，而后删去元音。

- *expr1* SOUNDS LIKE *expr2*

这相当于SOUNDEX(*expr1*) = SOUNDEX(*expr2*)。

- SPACE(*N*)

返回一个由*N* 间隔符号组成的字符串。

mysql> **SELECT SPACE(6);**

-> ' '

- SUBSTRING(*str*,*pos*) , SUBSTRING(*str* FROM *pos*) SUBSTRING(*str*,*pos*,*len*) , SUBSTRING(*str* FROM *pos* FOR *len*)

不带有*len* 参数的格式从字符串*str*返回一个子字符串，起始于位置 *pos*。带有*len*参数的格式从字符串*str*返回一个长度同*len*字符相同的子字符串，起始于位置 *pos*。 使用 FROM的格式为标准 SQL 语法。也可能对*pos*使用一个负值。假若这样，则子字符串的位置起始于字符串结尾的*pos* 字符，而不是字符串的开头位置。在以下格式的函数中可以对*pos* 使用一个负值。

mysql> **SELECT SUBSTRING('Quadratically',5);**

-> 'ratically'

mysql> **SELECT SUBSTRING('foobarbar' FROM 4);**

-> 'barbar'

mysql> **SELECT SUBSTRING('Quadratically',5,6);**

-> 'ratica'

mysql> **SELECT SUBSTRING('Sakila', -3);**

-> 'ila'

mysql> **SELECT SUBSTRING('Sakila', -5, 3);**

-> 'aki'

mysql> **SELECT SUBSTRING('Sakila' FROM -4 FOR 2);**

-> 'ki'

这个函数支持多字节字元。

注意，如果对*len*使用的是一个小于1的值，则结果始终为空字符串。

SUBSTR()是 SUBSTRING()的同义词。

- SUBSTRING_INDEX(*str*,*delim*,*count*)

在定界符 *delim* 以及*count* 出现前，从字符串*str*返回自字符串。若*count*为正值,则返回最终定界符(从左边开始)左边的一切内容。若*count*为负值，则返回定界符（从右边开始）右边的一切内容。

mysql> **SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);**

-> 'www.mysql'

mysql> **SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);**

-> 'mysql.com'

这个函数支持多字节字元。

- TRIM([{BOTH | LEADING | TRAILING} [*remstr*] FROM] *str*) TRIM(*remstr* FROM] *str*)

返回字符串 *str* ， 其中所有*remstr* 前缀和/或后缀都已被删除。若分类符BOTH、LEADIN或TRAILING中没有一个是给定的,则假设为BOTH 。 *remstr* 为可选项，在未指定情况下，可删除空格。

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'

这个函数支持多字节字元。

- UCASE(*str*)

UCASE()是UPPER()的同义词。

- UNCOMPRESS(*string_to_uncompress*)

对经COMPRESS()函数压缩后的字符串进行解压缩。若参数为压缩值，则结果为 NULL。这个函数要求 MySQL 已被诸如zlib 之类的压缩库编译过。否则, 返回值将始终是 NULL。

mysql> **SELECT UNCOMPRESS(COMPRESS('any string'));**

-> 'any string'

mysql> **SELECT UNCOMPRESS('any string');**

-> NULL

- UNCOMPRESSED_LENGTH(*compressed_string*)

返回压缩字符串压缩前的长度。

mysql> **SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',30)));**

-> 30

- UNHEX(*str*)

执行从HEX(*str*)的反向操作。就是说，它将参数中的每一对十六进制数字理解为一个数字，并将其转化为该数字代表的字符。结果字符以二进制字符串的形式返回。

mysql> **SELECT UNHEX('4D7953514C');**

-> 'MySQL'

mysql> **SELECT 0x4D7953514C;**

-> 'MySQL'

mysql> **SELECT UNHEX(HEX('string'));**

-> 'string'

mysql> **SELECT HEX(UNHEX('1267'));**

-> '1267'

- UPPER(*str*)

返回字符串*str*， 以及根据最新字符集映射转化为大写字母的字符 (默认为cp1252 Latin1).

mysql> **SELECT UPPER('Hej');**

-> 'HEJ'

该函数支持多字节字元。

12.3.1. 字符串比较函数

根据，MySQL 会自动将数字转化为字符串，反之亦然。

mysql> **SELECT 1+'1';**

-> 2

mysql> **SELECT CONCAT(2,' test');**

-> '2 test'

若想要将数字明确地转化为字符串，可使用 CAST()或 CONCAT()函数:

mysql> **SELECT 38.8, CAST(38.8 AS CHAR);**

-> 38.8, '38.8'

mysql> **SELECT 38.8, CONCAT(38.8);**

-> 38.8, '38.8'

CAST() 比较可取。.

若已经对一个字符串函数给定一个二进制字符串作为参数， 则所得到的结果字符串也是一个二进制字符串。一个转化为字符串的数字被作为二进制字符串对待。这仅会对比较结果产生影响。

一般而言, 若字符串比较中任意一个表达式是区分大小写的，则执行比较时也区分大小写。

- *expr* LIKE *pat* [ESCAPE '*escape-char*']

模式匹配，使用SQL简单正规表达式比较。返回1 (TRUE) 或 0 (FALSE)。 若 *expr* 或 *pat* 中任何一个为 NULL,则结果为 NULL。

模式不需要为文字字符串。例如，可以被指定为一个字符串表达式或表列。

在模式中可以同LIKE一起使用以下两种通配符:

字符	说明
%	匹配任何数目的字符，甚至包括零字符
_	只能匹配一种字符

mysql> **SELECT 'David!' LIKE 'David_';**

-> 1

mysql> **SELECT 'David!' LIKE '%D%v%';**

-> 1

若要对通配符的文字实例进行检验, 可将转义字符放在该字符前面。如果没有指定 ESCAPE字符, 则假设为'\'。

字符串	说明
\%	匹配一个'%'字符
_	匹配一个'_'字符

mysql> **SELECT 'David!' LIKE 'David_';**

-> 0

mysql> **SELECT 'David_' LIKE 'David_';**

-> 1

要指定一个不同的转义字符,可使用ESCAPE语句:

mysql> **SELECT 'David_' LIKE 'David|_' ESCAPE '|';**

-> 1

转义序列可以为空，也可以是一个字符的长度。 从 MySQL 5.1.2开始, 如若 NO_BACKSLASH_ESCAPES SQL模式被激活, 则该序列不能为空。

以下两个语句举例说明了字符串比较不区分大小写，除非其中一个操作数为二进制字符串:

mysql> **SELECT 'abc' LIKE 'ABC';**

-> 1

mysql> **SELECT 'abc' LIKE BINARY 'ABC';**

-> 0

在MySQL中, LIKE 允许出现在数字表达式中。 (这是标准SQL LIKE 的延伸）。

mysql> **SELECT 10 LIKE '1%';**

-> 1

注释： 由于 MySQL在字符串中使用 C转义语法(例如, 用'\n'代表一个换行字符)，在LIKE字符串中，必须将用到的'\'双写。例如， 若要查找 '\n', 必须将其写成 '\\n'。而若要查找 '\', 则必须将其写成 it as '\\\\';原因是反斜线符号会被语法分析程序剥离一次，在进行模式匹配时，又会被剥离一次，最后会剩下一个反斜线符号接受匹配。

- *expr* NOT LIKE *pat* [ESCAPE '*escape-char*']

这相当于 NOT (*expr* LIKE *pat* [ESCAPE '*escape-char*'])。

- *expr* NOT REGEXP *pat* *expr* NOT RLIKE *pat*

这相当于NOT (*expr* REGEXP *pat*)。

- *expr* REGEXP *pat* *expr* RLIKE *pat*

执行字符串表达式 *expr* 和模式*pat* 的模式匹配。该模式可以被延伸为正规表达式。正规表达式的语法在附

录G：*MySQL*正则表达式中有详细讨论。若*expr* 匹配 *pat*，则返回 1; 否则返回0。若 *expr* 或 *pat* 任意一个为 NULL, 则结果为 NULL。 RLIKE 是REGEXP的同义词, 作用是为mSQL 提供兼容性。

模式不需要为文字字符串。例如,可以被指定为一个字符串表达式或表列。

注释：由于在字符串中， MySQL使用 C 转义语法 (例如, 用'\n'来代表换行字符),在REGEXP字符串中必须将用到的'\' 双写。

REGEXP 不区分大小写, 除非将其同二进制字符串同时使用。

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

在确定字符类型时，REGEXP 和 RLIKE 使用当前字符集 (默认为cp1252 Latin1)。 警告：这些操作符不支持多字节字元。

- STRCMP(*expr1*,*expr2*)

若所有的字符串均相同，则返回STRCMP()，若根据当前分类次序，第一个参数小于第二个，则返回 -1，其它情况返回 1 。

mysql> **SELECT STRCMP('text', 'text2');**

-> -1

mysql> **SELECT STRCMP('text2', 'text');**

-> 1

mysql> **SELECT STRCMP('text', 'text');**

-> 0

在执行比较时，STRCMP() 使用当前字符集。这使得默认的比较区分大小写，当操作数中的一个或两个都是二进制字符串时除外。

12.4. 数值函数

12.4.1. 算术操作符

12.4.2. 数学函数

12.4.1. 算术操作符

可使用常见的算术操作符。注意就 -、 +和 *而言，若两个参数均为正数，则其计算结果的精确度为 BIGINT (64比特)，若其中一个参数为无符号整数， 而其它参数也是整数，则结果为无符号整数。请参见12.8节，“Cast函数和操作符”。

- +

加号:

mysql> **SELECT 3+5;**

-> 8

- -

减号:

mysql> **SELECT 3-5;**

-> -2

- -

一元减号。更换参数符号。

mysql> **SELECT - 2;**

-> -2

注意：若该 操作符同一个BIGINT同时使用，则返回值也是一个BIGINT。这意味着你应当尽量避免对可能产生-2^{63}的整数使用 –。

- *

乘号:

mysql> **SELECT 3*5;**

-> 15

mysql> **SELECT 18014398509481984*18014398509481984.0;**

-> 324518553658426726783156020576256.0

mysql> **SELECT 18014398509481984*18014398509481984;**

-> 0

最后一个表达式的结果是不正确的。原因是整数相乘的结果超过了BIGINT 计算的 64比特范围。(见11.2节，“数值类型”.)

- /

除号:

mysql> **SELECT 3/5;**

-> 0.60

被零除的结果为 NULL:

mysql> **SELECT 102/(1-1);**

-> NULL

只有当执行的语境中，其结果要被转化为一个整数时 ，除法才会和 BIGINT 算法一起使用。

- DIV

整数除法。 类似于 FLOOR()，然而使用BIGINT 算法也是可靠的。

mysql> **SELECT 5 DIV 2;**

-> 2

12.4.2. 数学函数

若发生错误，所有数学函数会返回 NULL 。

- ABS(*X*)

返回*X* 的绝对值。

mysql> **SELECT ABS(2);**

-> 2

mysql> **SELECT ABS(-32);**

-> 32

该函数支持使用BIGINT值。

- ACOS(*X*)

返回*X* 反余弦, 即, 余弦是*X*的值。若*X* 不在-1到 1的范围之内，则返回 NULL 。

mysql> **SELECT ACOS(1);**

-> 0

mysql> **SELECT ACOS(1.0001);**

-> NULL

mysql> **SELECT ACOS(0);**

-> 1.5707963267949

- ASIN(*X*)

返回*X* 的反正弦，即，正弦为*X* 的值。若*X* 若*X* 不在-1到 1的范围之内，则返回 NULL 。

```
mysql> SELECT ASIN(0.2);
        -> 0.20135792079033
mysql> SELECT ASIN('foo');

+-------------+
| ASIN('foo') |
+-------------+
|           0 |
+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1292 | Truncated incorrect DOUBLE value: 'foo' |
+---------+------+----------------------------------------+
```

- ATAN(*X*)

返回*X* 的反正切，即，正切为*X* 的值。

mysql> **SELECT ATAN(2);**

-> 1.1071487177941

mysql> **SELECT ATAN(-2);**

-> -1.1071487177941

- ATAN(*Y*,*X*) , ATAN2(*Y*,*X*)

返回两个变量*X* 及*Y*的反正切。 它类似于 *Y* 或 *X*的反正切计算，除非两个参数的符号均用于确定结果所在象限。

mysql> **SELECT ATAN(-2,2);**

-> -0.78539816339745

mysql> **SELECT ATAN2(PI(),0);**

-> 1.5707963267949

- CEILING(*X*) CEIL(*X*)

返回不小于*X* 的最小整数值。

mysql> **SELECT CEILING(1.23);**

-> 2

mysql> **SELECT CEIL(-1.23);**

-> -1

这两个函数的意义相同。注意返回值会被转化为一个BIGINT。

- COS(*X*)

返回*X* 的余弦，其中*X*在弧度上已知。

mysql> **SELECT COS(PI());**

-> -1

- COT(*X*)

返回*X* 的余切。

mysql> **SELECT COT(12);**

-> -1.5726734063977

mysql> **SELECT COT(0);**

-> NULL

- CRC32(*expr*)

计算循环冗余码校验值并返回一个 32比特无符号值。若参数为NULL ，则结果为 NULL。该参数应为一个字符串，而且在不是字符串的情况下会被作为字符串处理（若有可能）。

mysql> **SELECT CRC32('MySQL');**

-> 3259397556

mysql> **SELECT CRC32('mysql');**

-> 2501908538

- DEGREES(*X*)

返回参数 *X*, 该参数由弧度被转化为度。

mysql> **SELECT DEGREES(PI());**

-> 180

mysql> **SELECT DEGREES(PI() / 2);**

-> 90

- EXP(*X*)

返回e的*X*乘方后的值(自然对数的底)。

mysql> **SELECT EXP(2);**

-> 7.3890560989307

mysql> **SELECT EXP(-2);**

-> 0.13533528323661

mysql> **SELECT EXP(0);**

-> 1

- FLOOR(*X*)

返回不大于*X*的最大整数值 。

mysql> **SELECT FLOOR(1.23);**

-> 1

mysql> **SELECT FLOOR(-1.23);**

-> -2

注意，返回值会被转化为一个 BIGINT。

- FORMAT(*X*,*D*)

将数字*X* 的格式写成'#,###,###.##'格式, 即保留小数点后 *D*位，而第D位的保留方式为四舍五入，然后将结果以字符串的形式返回。详见12.9.4节，“其他函数”。

- LN(*X*)

返回*X* 的自然对数,即, *X* 相对于基数e 的对数。

mysql> **SELECT LN(2);**

-> 0.69314718055995

mysql> **SELECT LN(-2);**

-> NULL

这个函数同LOG(*X*)具有相同意义。

- LOG(*X*) LOG(*B*,*X*)

若用一个参数调用，这个函数就会返回*X* 的自然对数。

mysql> **SELECT LOG(2);**

-> 0.69314718055995

mysql> **SELECT LOG(-2);**

-> NULL

若用两个参数进行调用，这个函数会返回*X* 对于任意基数*B* 的对数。

mysql> **SELECT LOG(2,65536);**

-> 16

mysql> **SELECT LOG(10,100);**

-> 2

LOG(*B*,*X*) 就相当于 LOG(*X*) / LOG(*B*)。

- LOG2(*X*)

返回*X* 的基数为2的对数。

mysql> **SELECT LOG2(65536);**

-> 16

mysql> **SELECT LOG2(-100);**

-> NULL

对于查出存储一个数字需要多少个比特，LOG2()非常有效。这个函数相当于表达式 LOG(*X*) / LOG(2)。

- LOG10(*X*)

返回*X*的基数为10的对数。

mysql> **SELECT LOG10(2);**

-> 0.30102999566398

mysql> **SELECT LOG10(100);**

-> 2

mysql> **SELECT LOG10(-100);**

-> NULL

LOG10(*X*)相当于LOG(10,*X*)。

- MOD(*N*,*M*) , *N* % *M* *N* MOD *M*

模操作。返回*N* 被 *M*除后的余数。

mysql> **SELECT MOD(234, 10);**

-> 4

mysql> **SELECT 253 % 7;**

-> 1

mysql> **SELECT MOD(29,9);**

-> 2

mysql> **SELECT 29 MOD 9;**

-> 2

这个函数支持使用BIGINT 值。

MOD() 对于带有小数部分的数值也起作用， 它返回除法运算后的精确余数:

mysql> **SELECT MOD(34.5,3);**

-> 1.5

- PI()

返回 ϖ (pi)的值。默认的显示小数位数是7位,然而 MySQL内部会使用完全双精度值。

mysql> **SELECT PI();**

-> 3.141593

mysql> **SELECT PI()+0.000000000000000000;**

-> 3.141592653589793116

- POW(*X*,*Y*) , POWER(*X*,*Y*)

返回*X* 的*Y*乘方的结果值。

mysql> **SELECT POW(2,2);**

-> 4

mysql> **SELECT POW(2,-2);**

-> 0.25

- RADIANS(*X*)

返回由度转化为弧度的参数 *X*, (注意 ϖ 弧度等于180度）。

mysql> **SELECT RADIANS(90);**

-> 1.5707963267949

- RAND() RAND(*N*)

返回一个随机浮点值 *v* ，范围在 0 到1 之间 (即, 其范围为 0 ≤ *v* ≤ 1.0)。若已指定一个整数参数 *N* ，则它被用作种子值，用来产生重复序列。

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

mysql> **SELECT RAND(20);**

-> 0.15888261251047

若要在$i \le R \le j$ 这个范围得到一个随机整数R ，需要用到表达式 FLOOR(i + RAND() * (j – i + 1))。例如， 若要在7 到 12 的范围（包括7和12）内得到一个随机整数, 可使用以下语句:

SELECT FLOOR(7 + (RAND() * 6));

在ORDER BY语句中，不能使用一个带有RAND()值的列，原因是 ORDER BY 会计算列的多重时间。然而，可按照如下的随机顺序检索数据行:

mysql> **SELECT * FROM** *tbl_name* **ORDER BY RAND();**

ORDER BY RAND()同 LIMIT 的结合从一组列中选择随机样本很有用:

mysql> **SELECT * FROM table1, table2 WHERE a=b AND c<d**

-> **ORDER BY RAND() LIMIT 1000;**

注意，在WHERE语句中，WHERE每执行一次， RAND()就会被再计算一次。

RAND()的作用不是作为一个精确的随机发生器，而是一种用来发生在同样的 MySQL版本的平台之间的可移动*ad hoc*随机数的快速方式。

- ROUND(*X*) ROUND(*X*,*D*)

返回参数*X*, 其值接近于最近似的整数。在有两个参数的情况下，返回 *X* ，其值保留到小数点后*D*位，而第*D*位的保留方式为四舍五入。若要接保留*X*值小数点左边的*D* 位，可将 *D* 设为负值。

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

返回值的类型同 第一个自变量相同(假设它是一个整数、双精度数或小数)。这意味着对于一个整数参数,结果也是一个整数(无小数部分)。

当第一个参数是十进制常数时，对于准确值参数，ROUND() 使用精密数学题库:

- 对于准确值数字, ROUND() 使用"四舍五入" 或"舍入成最接近的数" 的规则:对于一个分数部分为 .5或大于 .5的值，正数则上舍入到邻近的整数值， 负数则下舍入临近的整数值。(换言之, 其舍入的方向是数轴上远离零的方向）。对于一个分数部分小于.5 的值，正数则下舍入下一个整数值，负数则下舍入邻近的整数值，而正数则上舍入邻近的整数值。
- 对于近似值数字，其结果根据C 库而定。在很多系统中，这意味着 ROUND()的使用遵循"舍入成最接近的偶数"的规则： 一个带有任何小数部分的值会被舍入成最接近的偶数整数。

以下举例说明舍入法对于精确值和近似值的不同之处:

mysql> **SELECT ROUND(2.5), ROUND(25E-1);**

```
+------------+--------------+
| ROUND(2.5) | ROUND(25E-1) |
+------------+--------------+
| 3          |            2 |
+------------+--------------+
```

详见第24章：精度数学。

- SIGN(*X*)

返回参数作为-1、 0或1的符号，该符号取决于*X* 的值为负、零或正。

mysql> **SELECT SIGN(-32);**

-> -1

mysql> **SELECT SIGN(0);**

-> 0

mysql> **SELECT SIGN(234);**

-> 1

- SIN(*X*)

返回*X* 正弦，其中 *X* 在弧度中被给定。

mysql> **SELECT SIN(PI());**

-> 1.2246063538224e-16

mysql> **SELECT ROUND(SIN(PI()));**

-> 0

- SQRT(*X*)

返回非负数*X* 的二次方根。

mysql> **SELECT SQRT(4);**

-> 2

mysql> **SELECT SQRT(20);**

-> 4.4721359549996

mysql> **SELECT SQRT(-16);**

-> NULL

- TAN(*X*)

返回*X* 的正切，其中*X* 在弧度中被给定。

mysql> **SELECT TAN(PI());**

-> -1.2246063538224e-16

mysql> **SELECT TAN(PI()+1);**

-> 1.5574077246549

- TRUNCATE(*X*,*D*)

返回被舍去至小数点后*D*位的数字*X*。若*D* 的值为 0, 则结果不带有小数点或不带有小数部分。可以将D设为负数,若要截去(归零) X小数点左起第D位开始后面所有低位的值.

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> **SELECT TRUNCATE(122,-2);**

-> 100

mysql> **SELECT TRUNCATE(10.28*100,0);**

-> 1028

所有数字的舍入方向都接近于零。

12.5. 日期和时间函数

本章论述了一些可用于操作时间值的函数。关于每个时间和日期类型具有的值域及指定值的有效格式，请参见11.3节，“日期和时间类型”。

下面的例子使用了时间函数。以下询问选择了最近的 30天内所有带有date_col 值的记录：

mysql> **SELECT** *something* **FROM** *tbl_name*

-> **WHERE DATE_SUB(CURDATE(),INTERVAL 30 DAY) <=** *date_col***;**

注意，这个询问也能选择将来的日期记录。

用于日期值的函数通常会接受时间日期值而忽略时间部分。而用于时间值的函数通常接受时间日期值而忽略日期部分。

返回各自当前日期或时间的函数在每次询问执行开始时计算一次。这意味着在一个单一询问中，对诸如NOW()的函数多次访问总是会得到同样的结果(未达到我们的目的，单一询问也包括对存储程序或触发器和被该程序/触发器调用的所有子程序的调用)。这项原则也适用于 CURDATE()、 CURTIME()、 UTC_DATE()、UTC_TIME()、UTC_TIMESTAMP()，以及所有和它们意义相同的函数。

CURRENT_TIMESTAMP()、 CURRENT_TIME()、 CURRENT_DATE()以及FROM_UNIXTIME()函数返回连接当前时区内的值，这个值可用作time_zone系统变量的值。此外， UNIX_TIMESTAMP() 假设其参数为一个当前时区的时间日期值。请参见5.10.8节，“MySQL服务器时区支持”。

以下函数的论述中返回值的范围会请求完全日期。 若一个日期为“零” 值，或者是一个诸如'2001-11-00'之类的不完全日期， 提取部分日期值的函数可能会返回 0。 例如， DAYOFMONTH('2001-11-00') 会返回0。

- ADDDATE(*date*,INTERVAL *expr type*) ADDDATE(*expr*,*days*)

当被第二个参数的INTERVAL格式激活后， ADDDATE()就是DATE_ADD()的同义词。相关函数SUBDATE() 则是DATE_SUB()的同义词。对于INTERVAL参数上的信息 ，请参见关于DATE_ADD()的论述。

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'

若 *days* 参数只是整数值，则 MySQL 5.1将其作为天数值添加至 *expr*。

mysql> **SELECT ADDDATE('1998-01-02', 31);**

-> '1998-02-02'

- ADDTIME(*expr*,*expr2*)

ADDTIME()将 *expr2*添加至*expr* 然后返回结果。 *expr* 是一个时间或时间日期表达式，而*expr2* 是一个时间表达式。

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'

- CONVERT_TZ(*dt*,*from_tz*,*to_tz*)

CONVERT_TZ() 将时间日期值*dt* 从*from_tz* 给出的时区转到*to_tz*给出的时区，然后返回结果值。关于可能指定的时区的详细论述，请参见5.10.8节，“MySQL服务器时区支持”。若自变量无效，则这个函数会返回NULL。

在从若*from_tz* 到UTC的转化过程中，该值超出 TIMESTAMP 类型的被支持范围，那么转化不会发生。关于TIMESTAMP 范围的论述，请参见11.1.2节，“日期和时间类型概述”。

mysql> **SELECT CONVERT_TZ('2004-01-01 12:00:00','GMT','MET');**

-> '2004-01-01 13:00:00'

mysql> **SELECT CONVERT_TZ('2004-01-01 12:00:00','+00:00','+10:00');**

-> '2004-01-01 22:00:00'

注释：若要使用诸如 'MET'或 'Europe/Moscow'之类的指定时间区，首先要设置正确的时区表。详细说明见5.10.8节，“MySQL服务器时区支持”。

- CURDATE()

将当前日期按照'YYYY-MM-DD' 或YYYYMMDD 格式的值返回，具体格式根据函数用在字符串或是数字语境中而定。

mysql> **SELECT CURDATE();**

-> '1997-12-15'

mysql> **SELECT CURDATE() + 0;**

-> 19971215

- CURRENT_DATE CURRENT_DATE()

CURRENT_DATE和CURRENT_DATE()是的同义词.

- CURTIME()

将当前时间以'HH:MM:SS'或 HHMMSS 的格式返回， 具体格式根据函数用在字符串或是数字语境中而定。

mysql> **SELECT CURTIME();**

-> '23:50:26'

mysql> **SELECT CURTIME() + 0;**

-> 235026

- CURRENT_TIME, CURRENT_TIME()

CURRENT_TIME 和CURRENT_TIME() 是CURTIME()的同义词。

- CURRENT_TIMESTAMP, CURRENT_TIMESTAMP()

CURRENT_TIMESTAMP和 CURRENT_TIMESTAMP()是NOW()的同义词。

- DATE(*expr*)

提取日期或时间日期表达式*expr*中的日期部分。

mysql> **SELECT DATE('2003-12-31 01:02:03');**

-> '2003-12-31'

- DATEDIFF(*expr*,*expr2*)

DATEDIFF() 返回起始时间 *expr*和结束时间*expr2*之间的天数。*Expr*和*expr2* 为日期或 date-and-time 表达式。计算中只用到这些值的日期部分。

mysql> **SELECT DATEDIFF('1997-12-31 23:59:59','1997-12-30');**

-> 1

mysql> **SELECT DATEDIFF('1997-11-30 23:59:59','1997-12-31');**

-> -31

- DATE_ADD(*date*,INTERVAL *expr type*) DATE_SUB(*date*,INTERVAL *expr type*)

这些函数执行日期运算。 *date* 是一个 DATETIME 或DATE值，用来指定起始时间。 *expr* 是一个表达式，用来指定从起始日期添加或减去的时间间隔值。 *Expr*是一个字符串;对于负值的时间间隔，它可以以一个 '-'开头。 *type* 为关键词，它指示了表达式被解释的方式。

关键词INTERVA及 *type* 分类符均不区分大小写。

以下表显示了*type* 和*expr* 参数的关系:

type 值	预期的 *expr* 格式
MICROSECOND	MICROSECONDS
SECOND	SECONDS
MINUTE	MINUTES
HOUR	HOURS
DAY	DAYS
WEEK	WEEKS
MONTH	MONTHS
QUARTER	QUARTERS
YEAR	YEARS
SECOND_MICROSECOND	'SECONDS.MICROSECONDS'

MINUTE_MICROSECOND	'MINUTES.MICROSECONDS'
MINUTE_SECOND	'MINUTES:SECONDS'
HOUR_MICROSECOND	'HOURS.MICROSECONDS'
HOUR_SECOND	'HOURS:MINUTES:SECONDS'
HOUR_MINUTE	'HOURS:MINUTES'
DAY_MICROSECOND	'DAYS.MICROSECONDS'
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'
DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_HOUR	'DAYS HOURS'
YEAR_MONTH	'YEARS-MONTHS'

MySQL 允许任何*expr* 格式中的标点分隔符。表中所显示的是建议的 分隔符。若 *date* 参数是一个 DATE 值，而你的计算只会包括 YEAR、MONTH和DAY部分(即, 没有时间部分), 其结果是一个DATE 值。否则，结果将是一个 DATETIME值。

若位于另一端的表达式是一个日期或日期时间值 ， 则INTERVAL *expr type*只允许在 + 操作符的两端。对于 –操作符， INTERVAL *expr type* 只允许在其右端，原因是从一个时间间隔中提取一个日期或日期时间值是毫无意义的。 (见下面的例子）。

```
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'
```

若你指定了一个过于短的时间间隔值 (不包括*type* 关键词所预期的所有时间间隔部分), MySQL 假定你已经省去了时间间隔值的最左部分。 例如，你指定了一种类型的DAY_SECOND, *expr* 的值预期应当具有天、 小时、分钟和秒部分。若你指定了一个类似 '1:10'的值, MySQL 假定天和小时部分不存在，那么这个值代表分和秒。换言之, '1:10' DAY_SECOND 被解释为相当于 '1:10' MINUTE_SECOND。这相当于 MySQL将TIME 值解释为所耗费的时间而不是日时的解释方式。

假如你对一个日期值添加或减去一些含有时间部分的内容，则结果自动转化为一个日期时间值:

```
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'
```

假如你使用了格式严重错误的日期,则结果为 NULL。假如你添加了 MONTH、YEAR_MONTH或YEAR ，而结果日期中有一天的日期大于添加的月份的日期最大限度，则这个日期自动被调整为添加月份的最大日期:

```
mysql> SELECT DATE_ADD('1998-01-30', INTERVAL 1 MONTH);
    -> '1998-02-28'
```

- DATE_FORMAT(*date*,*format*)

根据*format* 字符串安排*date* 值的格式。

以下说明符可用在 *format* 字符串中:

说明符	说明
%a	工作日的缩写名称 (Sun..Sat)
%b	月份的缩写名称 (Jan..Dec)
%c	月份，数字形式(0..12)
%D	带有英语后缀的该月日期 (0th, 1st, 2nd, 3rd, ...)
%d	该月日期, 数字形式 (00..31)
%e	该月日期, 数字形式(0..31)
%f	微秒 (000000..999999)
%H	小时(00..23)

%h	小时(01..12)
%I	小时 (01..12)
%i	分钟,数字形式 (00..59)
%j	一年中的天数 (001..366)
%k	小时 (0..23)
%l	小时 (1..12)
%M	月份名称 (January..December)
%m	月份, 数字形式 (00..12)
%p	上午（AM）或下午（ PM）
%r	时间 , 12小时制 (小时hh:分钟mm:秒数ss 后加 AM或PM)
%S	秒 (00..59)
%s	秒 (00..59)
%T	时间 , 24小时制 (小时hh:分钟mm:秒数ss)
%U	周 (00..53), 其中周日为每周的第一天
%u	周 (00..53), 其中周一为每周的第一天
%V	周 (01..53), 其中周日为每周的第一天 ; 和 %X同时使用
%v	周 (01..53), 其中周一为每周的第一天 ; 和 %x同时使用
%W	工作日名称 (周日..周六)
%w	一周中的每日 (0=周日..6=周六)
%X	该周的年份，其中周日为每周的第一天, 数字形式,4位数;和%V同时使用
%x	该周的年份，其中周一为每周的第一天, 数字形式,4位数;和%v同时使用
%Y	年份, 数字形式,4位数
%y	年份, 数字形式 (2位数)
%%	'%'文字字符

所有其它字符都被复制到结果中，无需作出解释。

注意，'%'字符要求在格式指定符之前。

月份和日期说明符的范围从零开始，原因是 MySQL允许存储诸如 '2004-00-00'的不完全日期.

```
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() 和DAYOFMONTH()的意义相同。

- DAYNAME(*date*)

返回*date* 对应的工作日名称。

mysql> **SELECT DAYNAME('1998-02-05');**

-> '周四'

- DAYOFMONTH(*date*)

返回*date* 对应的该月日期，范围是从 1到31。

mysql> **SELECT DAYOFMONTH('1998-02-03');**

-> 3

- DAYOFWEEK(*date*)

返回*date* (1 = 周日, 2 = 周一, ..., 7 = 周六)对应的工作日索引。这些索引值符合 ODBC标准。

mysql> **SELECT DAYOFWEEK('1998-02-03');**

-> 3

- DAYOFYEAR(*date*)

返回*date* 对应的一年中的天数，范围是从 1到366。

mysql> **SELECT DAYOFYEAR('1998-02-03');**

-> 34

- EXTRACT(*type* FROM *date*)

EXTRACT()函数所使用的时间间隔类型说明符同 DATE_ADD()或DATE_SUB()的相同,但它从日期中提取其部分，而不是执行日期运算。

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*)

给定一个天数 *N*, 返回一个DATE值。

mysql> **SELECT FROM_DAYS(729669);**

-> '1997-10-07'

使用 FROM_DAYS()处理古老日期时，务必谨慎。他不用于处理阳历出现前的日期(1582)。请参见12.6节，“MySQL使用什么日历？”。

- FROM_UNIXTIME(*unix_timestamp*) , FROM_UNIXTIME(*unix_timestamp*,*format*)

返回'YYYY-MM-DD HH:MM:SS'或YYYYMMDDHHMMSS 格式值的*unix_timestamp*参数表示，具体格式取决于该函数是否用在字符串中或是数字语境中。

若*format* 已经给出，则结果的格式是根据*format* 字符串而定。 *format* 可以包含同DATE_FORMAT() 函数输入项列表中相同的说明符。

mysql> **SELECT FROM_UNIXTIME(875996580);**

-> '1997-10-04 22:23:00'

mysql> **SELECT FROM_UNIXTIME(875996580) + 0;**

-> 19971004222300

mysql> **SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(),**

-> **'%Y %D %M %h:%i:%s %x');**

-> '2003 6th August 06:22:58 2003'

- GET_FORMAT(DATE|TIME|DATETIME, 'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL')

返回一个格式字符串。这个函数在同DATE_FORMAT() 及STR_TO_DATE()函数结合时很有用。

第一个参数的3个可能值和第二个参数的5个可能值产生 15 个可能格式字符串 (对于使用的说明符，请参见DATE_FORMAT()函数说明表)。

函数调用	结果
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(DATETIME,'USA')	'%Y-%m-%d-%H.%i.%s'
GET_FORMAT(DATETIME,'JIS')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'ISO')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'EUR')	'%Y-%m-%d-%H.%i.%s'
GET_FORMAT(DATETIME,'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 格式为ISO 9075, 而非ISO 8601.

也可以使用TIMESTAMP, 这时GET_FORMAT()的返回值和DATETIME相同。

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'

或见13.5.3节，“SET语法”。

- HOUR(*time*)

返回*time* 对应的小时数。对于日时值的返回值范围是从 0 到 23 。

mysql> **SELECT HOUR('10:05:03');**

-> 10

然而，TIME 值的范围实际上非常大, 所以HOUR可以返回大于23的值。

mysql> **SELECT HOUR('272:59:59');**

-> 272

- LAST_DAY(*date*)

获取一个日期或日期时间值，返回该月最后一天对应的值。若参数无效，则返回NULL。

mysql> **SELECT LAST_DAY('2003-02-05');**

-> '2003-02-28'

mysql> **SELECT LAST_DAY('2004-02-05');**

-> '2004-02-29'

mysql> **SELECT LAST_DAY('2004-01-01 01:01:01');**

-> '2004-01-31'

mysql> **SELECT LAST_DAY('2003-03-32');**

-> NULL

- LOCALTIME, LOCALTIME()

LOCALTIME 及 LOCALTIME()和NOW()具有相同意义。

- LOCALTIMESTAMP, LOCALTIMESTAMP()

LOCALTIMESTAMP和LOCALTIMESTAMP()和NOW()具有相同意义。

- MAKEDATE(*year*,*dayofyear*)

给出年份值和一年中的天数值，返回一个日期。*dayofyear* 必须大于 0 ，否则结果为 NULL。

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

- MAKETIME(*hour*,*minute*,*second*)

返回由*hour*、 *minute*和*second* 参数计算得出的时间值。

mysql> **SELECT MAKETIME(12,15,30);**

-> '12:15:30'

- MICROSECOND(*expr*)

从时间或日期时间表达式*expr*返回微秒值，其数字范围从 0到 999999。

mysql> **SELECT MICROSECOND('12:00:00.123456');**

-> 123456

mysql> **SELECT MICROSECOND('1997-12-31 23:59:59.000010');**

-> 10

- MINUTE(*time*)

返回 *time* 对应的分钟数,范围是从 0 到 59。

mysql> **SELECT MINUTE('98-02-03 10:05:03');**

-> 5

- MONTH(*date*)

返回*date* 对应的月份，范围时从 1 到 12。

mysql> **SELECT MONTH('1998-02-03');**

-> 2

- MONTHNAME(*date*)

返回*date* 对应月份的全名。

mysql> **SELECT MONTHNAME('1998-02-05');**

-> 'February '

- NOW()

返回当前日期和时间值，其格式为 'YYYY-MM-DD HH:MM:SS' 或YYYYMMDDHHMMSS ， 具体格式取决于该函数是否用在字符串中或数字语境中。

mysql> **SELECT NOW();**

-> '1997-12-15 23:50:26'

mysql> **SELECT NOW() + 0;**

-> 19971215235026

在一个存储程序或触发器内, NOW() 返回一个常数时间，该常数指示了该程序或触发语句开始执行的时间。这同SYSDATE()的运行有所不同。

- PERIOD_ADD(*P*,*N*)

添加 *N* 个月至周期*P* (格式为YYMM 或YYYYMM)，返回值的格式为 YYYYMM。注意周期参数 *P* 不是日期值。

mysql> **SELECT PERIOD_ADD(9801,2);**

-> 199803

- PERIOD_DIFF(*P1*,*P2*)

返回周期*P1*和 *P2* 之间的月份数。*P1* 和*P2* 的格式应该为YYMM或YYYYMM。注意周期参数 *P1*和*P2* 不是日期值。

mysql> **SELECT PERIOD_DIFF(9802,199703);**

-> 11

- QUARTER(*date*)

返回*date* 对应的一年中的季度值，范围是从 1到 4。

mysql> **SELECT QUARTER('98-04-01');**

-> 2

- SECOND(*time*)

返回*time* 对应的秒数, 范围是从 0到59。

mysql> **SELECT SECOND('10:05:03');**

-> 3

- SEC_TO_TIME(*seconds*)

返回被转化为小时、 分钟和秒数的*seconds*参数值, 其格式为 'HH:MM:SS' 或HHMMSS，具体格式根据该函数

是否用在字符串或数字语境中而定。

mysql> **SELECT SEC_TO_TIME(2378);**

-> '00:39:38'

mysql> **SELECT SEC_TO_TIME(2378) + 0;**

-> 3938

- STR_TO_DATE(*str*,*format*)

这是DATE_FORMAT() 函数的倒转。它获取一个字符串 *str* 和一个格式字符串*format*。若格式字符串包含日期和时间部分，则 STR_TO_DATE()返回一个 DATETIME 值， 若该字符串只包含日期部分或时间部分，则返回一个 DATE 或TIME值。

*str*所包含的日期、时间或日期时间值应该在*format*指示的格式中被给定。对于可用在*format*中的说明符，请参见DATE_FORMAT() 函数说明表。 所有其它的字符被逐字获取，因此不会被解释。若 *str* 包含一个非法日期、时间或日期时间值，则 STR_TO_DATE()返回NULL。同时，一个非法值会引起警告。

对日期值部分的范围检查在11.3.1节，“DATETIME、DATE和TIMESTAMP类型”有详细说明。其意义是,例如, 只要具体日期部分的范围时从 1到 31之间，则允许一个日期中的具体日期部分大于一个月中天数值。并且，允许“零”日期或带有0值部分的日期。

mysql> **SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');**

-> '0000-00-00'

mysql> **SELECT STR_TO_DATE('04/31/2004', '%m/%d/%Y');**

-> '2004-04-31'

- SUBDATE(*date*,INTERVAL *expr type*) SUBDATE(*expr*,*days*)

当被第二个参数的 INTERVAL型式调用时, SUBDATE()和DATE_SUB()的意义相同。对于有关INTERVAL参数的信息， 见有关 DATE_ADD()的讨论。

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'

第二个形式允许对*days*使用整数值。在这些情况下，它被算作由日期或日期时间表达式 *expr*中提取的天数。

mysql> **SELECT SUBDATE('1998-01-02 12:00:00', 31);**

-> '1997-12-02 12:00:00'

注意不能使用格式 "%X%V" 来将一个 year-week 字符串转化为一个日期，原因是当一个星期跨越一个月份界限时，一个年和星期的组合不能标示一个唯一的年和月份。若要将year-week转化为一个日期，则也应指定具体工作日:

mysql> **select str_to_date('200442 Monday', '%X%V %W');**

-> 2004-10-18

- SUBTIME(*expr*,*expr2*)

SUBTIME()从*expr* 中提取*expr2* ，然后返回结果。*expr* 是一个时间或日期时间表达式，而*xpr2* 是一个时间表达式。

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'

- SYSDATE()

返回当前日期和时间值，格式为'YYYY-MM-DD HH:MM:SS' 或YYYYMMDDHHMMSS， 具体格式根据函数是否用在字符串或数字语境而定。

在一个存储程序或触发器中, SYSDATE()返回其执行的时间, 而非存储成都或触发语句开始执行的时间。这个NOW()的运作有所不同。

- TIME(*expr*)

提取一个时间或日期时间表达式的时间部分，并将其以字符串形式返回。

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'

- TIMEDIFF(*expr*,*expr2*)

TIMEDIFF() 返回起始时间 *expr* 和结束时间*expr2* 之间的时间。 *expr* 和*expr2* 为时间或 date-and-time 表达式,两个的类型必须一样。

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'

- TIMESTAMP(*expr*) , TIMESTAMP(*expr*,*expr2*)

对于一个单参数,该函数将日期或日期时间表达式 *expr* 作为日期时间值返回.对于两个参数, 它将时间表达式 *expr2* 添加到日期或日期时间表达式 *expr* 中，将theresult作为日期时间值返回。

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'

- TIMESTAMPADD(*interval*,*int_expr*,*datetime_expr*)

将整型表达式*int_expr* 添加到日期或日期时间表达式 *datetime_expr*中。 *int_expr* 的单位被时间间隔参数给定，该参数必须是以下值的其中一个： FRAC_SECOND、SECOND、 MINUTE、 HOUR、 DAY、 WEEK、 MONTH、 QUARTER或 YEAR。

可使用所显示的关键词指定*Interval*值，或使用SQL_TSI_前缀。例如, DAY或SQL_TSI_DAY 都是正确的。

mysql> **SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');**

-> '2003-01-02 00:01:00'

mysql> **SELECT TIMESTAMPADD(WEEK,1,'2003-01-02');**

-> '2003-01-09'

- TIMESTAMPDIFF(*interval*,*datetime_expr1*,*datetime_expr2*)

返回日期或日期时间表达式*datetime_expr1* 和*datetime_expr2*the 之间的整数差。其结果的单位由*interval* 参数给出。*interval* 的法定值同TIMESTAMPADD()函数说明中所列出的相同。

mysql> **SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');**

-> 3

mysql> **SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');**

-> -1

- TIME_FORMAT(*time*,*format*)

其使用和 DATE_FORMAT()函数相同, 然而*format* 字符串可能仅会包含处理小时、分钟和秒的格式说明符。其它说明符产生一个NULL值或0。

若*time* value包含一个大于23的小时部分，则 %H 和 %k 小时格式说明符会产生一个大于0..23的通常范围的值。另一个小时格式说明符产生小时值模数12。

mysql> **SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');**

-> '100 100 04 04 4'

- TIME_TO_SEC(*time*)

返回已转化为秒的*time*参数。

mysql> **SELECT TIME_TO_SEC('22:23:00');**

-> 80580

mysql> **SELECT TIME_TO_SEC('00:39:38');**

-> 2378

- TO_DAYS(*date*)

给定一个日期*date*, 返回一个天数 (从年份0开始的天数)。

mysql> **SELECT TO_DAYS(950501);**

-> 728779

mysql> **SELECT TO_DAYS('1997-10-07');**

-> 729669

TO_DAYS() 不用于阳历出现(1582)前的值，原因是当日历改变时，遗失的日期不会被考虑在内。请参见12.6节，“MySQL使用什么日历？”。

请记住， MySQL使用11.3节，“日期和时间类型”中的规则将日期中的二位数年份值转化为四位。例如，'1997-10-07'和 '97-10-07' 被视为同样的日期:

mysql> **SELECT TO_DAYS('1997-10-07'), TO_DAYS('97-10-07');**

-> 729669, 729669

对于1582 年之前的日期(或许在其它地区为下一年), 该函数的结果实不可靠的。详见12.6节，“MySQL使用什么日历？” 。

- UNIX_TIMESTAMP(), UNIX_TIMESTAMP(*date*)

若无参数调用，则返回一个Unix timestamp ('1970-01-01 00:00:00' GMT 之后的秒数) 作为无符号整数。若用*date* 来调用UNIX_TIMESTAMP()，它会将参数值以'1970-01-01 00:00:00' GMT后的秒数的形式返回。*date* 可以是一个DATE 字符串、一个 DATETIME字符串、一个 TIMESTAMP或一个当地时间的YYMMDD或YYYMMDD格式的数字。

mysql> **SELECT UNIX_TIMESTAMP();**

-> 882226357

mysql> **SELECT UNIX_TIMESTAMP('1997-10-04 22:23:00');**

-> 875996580

当 UNIX_TIMESTAMP被用在 TIMESTAMP列时, 函数直接返回内部时戳值， 而不进行任何隐含的 “string-to-Unix-timestamp”转化。假如你向UNIX_TIMESTAMP()传递一个溢出日期，它会返回 0,但请注意只有基本范围检查会被履行 (年份从1970 到 2037， 月份从01到12,日期从 01 到31)。

假如你想要减去 UNIX_TIMESTAMP() 列, 你或许希望删去带符号整数的结果。请参见12.8节，“Cast函数和操作符”。

- UTC_DATE, UTC_DATE()

返回当前 UTC日期值，其格式为 'YYYY-MM-DD' 或 YYYYMMDD，具体格式取决于函数是否用在字符串或数字语境中。

mysql> **SELECT UTC_DATE(), UTC_DATE() + 0;**

-> '2003-08-14', 20030814

- UTC_TIME, UTC_TIME()

返回当前 UTC 值，其格式为 'HH:MM:SS' 或HHMMSS，具体格式根据该函数是否用在字符串或数字语境而定。

mysql> **SELECT UTC_TIME(), UTC_TIME() + 0;**

-> '18:07:53', 180753

- UTC_TIMESTAMP, UTC_TIMESTAMP()

返回当前UTC日期及时间值，格式为 'YYYY-MM-DD HH:MM:SS' 或YYYYMMDDHHMMSS，具体格式根据该函数是否用在字符串或数字语境而定。

mysql> **SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;**

-> '2003-08-14 18:08:04', 20030814180804

- WEEK(*date*[,*mode*])

该函数返回*date* 对应的星期数。WEEK() 的双参数形式允许你指定该星期是否起始于周日或周一， 以及返回值的范围是否为从0 到53 或从1 到53。若 *mode*参数被省略，则使用default_week_format系统自变量的值。请参见5.3.3节，“服务器系统变量”。

以下表说明了*mode* 参数的工作过程：d

	第一天		
Mode	工作日	范围	**Week 1** 为第一周 **...**
0	周日	0-53	本年度中有一个周日
1	周一	0-53	本年度中有3天以上
2	周日	1-53	本年度中有一个周日
3	周一	1-53	本年度中有3天以上
4	周日	0-53	本年度中有3天以上
5	周一	0-53	本年度中有一个周一
6	周日	1-53	本年度中有3天以上
7	周一	1-53	本年度中有一个周一

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，假如有一个日期位于前一年的最后一周， 若你不使用2、3、6或7作为*mode* 参数选择，则MySQL返回

0:

mysql> **SELECT YEAR('2000-01-01'), WEEK('2000-01-01',0);**

-> 2000, 0

有人或许会提出意见，认为 MySQL 对于WEEK() 函数应该返回 52 ，原因是给定的日期实际上发生在1999年的第52周。我们决定返回0作为代替的原因是我们希望该函数能返回"给定年份的星期数"。这使得WEEK() 函数在同其它从日期中抽取日期部分的函数结合时的使用更加可靠。

假如你更希望所计算的关于年份的结果包括给定日期所在周的第一天，则应使用 0、2、5或 7 作为*mode*参数选择。

mysql> **SELECT WEEK('2000-01-01',2);**

-> 52

作为选择，可使用 YEARWEEK()函数:

mysql> **SELECT YEARWEEK('2000-01-01');**

-> 199952

mysql> **SELECT MID(YEARWEEK('2000-01-01'),5,2);**

-> '52'

- WEEKDAY(*date*)

返回*date* (0 = 周一, 1 = 周二, ... 6 = 周日)对应的工作日索引 weekday index for

mysql> **SELECT WEEKDAY('1998-02-03 22:23:00');**

-> 1

mysql> **SELECT WEEKDAY('1997-11-05');**

-> 2

- WEEKOFYEAR(*date*)

将该日期的阳历周以数字形式返回，范围是从1到53。它是一个兼容度函数，相当于WEEK(*date*,3)。

mysql> **SELECT WEEKOFYEAR('1998-02-20');**

-> 8

- YEAR(*date*)

返回*date* 对应的年份,范围是从1000到9999。

mysql> **SELECT YEAR('98-02-03');**

-> 1998

- YEARWEEK(*date*), YEARWEEK(*date*,*start*)

返回一个日期对应的年或周。*start*参数的工作同 *start*参数对 WEEK()的工作相同。结果中的年份可以和该年的

第一周和最后一周对应的日期参数有所不同。

mysql> **SELECT YEARWEEK('1987-01-01');**

-> 198653

注意，周数和WEEK()函数队可选参数0或 1可能会返回的(0) w有所不同，原因是此时 WEEK() 返回给定年份的语境中的周。

- -> '1997-10-07'

12.6. MySQL使用什么日历？

MySQL 使用通常所说的 *proleptic* 阳历。

每个将日历由朱利安改为阳历的国家在改变日历期间都不得不删除至少10天。 为了了解其运作，让我们看看1582年10月，这是由朱利安日历转换为阳历的第一次:

周一	周二	周三	周四	周五	周六	周日
1	2	3	4	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

在10月4 日到10月15日之间的日期为空白。这个中断被称为接入。接入前的日期均使用朱利安日历, 而接入后的日期均使用阳历。接入期间的日期是不存在的。

当一个用于日期的日历并为得到实际使用时被称为 *proleptic*。因此, 若我们假设从来没有接入期的存在，而阳历历法则始终被使用，我们会有一个预期的阳历 。这就是MySQL 所使用的，正如标准SQL所要求的。 鉴于这个原因，作为MySQL DATE 或 DATETIME值而被储存的接入前的日期必须调整这个差异。我们必须明白，接入的发生时间在不同的国家有所不同，而接入的时间越晚，遗失的日期越多。例如，在大不列颠, 接入发生在1752年,这时9月2日，周三后的第二天为9月14日，周二； 俄罗斯结束使用朱利安日历的时间为1918年,接入过程中遗失天数为 13天, 根据阳历，其普遍被称为"10月革命"的发生时间实际上是11月。

12.7. 全文搜索功能

12.7.1. 布尔全文搜索
12.7.2. 全文搜索带查询扩展
12.7.3. 全文停止字
12.7.4. 全文限定条件
12.7.5. 微调MySQL全文搜索

- MATCH (col1,col2,...) AGAINST (expr [IN BOOLEAN MODE | WITH QUERY EXPANSION])

MySQL支持全文索引和搜索功能。MySQL中的全文索引类型FULLTEXT的索引。 FULLTEXT 索引仅可用于 MyISAM 表；他们可以从CHAR、 VARCHAR或TEXT列中作为CREATE TABLE语句的一部分被创建，或是随后使用ALTER TABLE 或 CREATE INDEX被添加。对于较大的数据集，将你的资料输入一个没有FULLTEXT索引的表中，然后创建索引， 其速度比把资料输入现有FULLTEXT索引的速度更为快。

关于全文搜索的限制列表，请参见 12.7.4节，"全文限定条件".

全文搜索同MATCH()函数一起执行。

```
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 (title,body) VALUES
    -> ('MySQL Tutorial','DBMS stands for DataBase ...'),
    -> ('How To Use MySQL Well','After you went through a ...'),
    -> ('Optimizing MySQL','In this tutorial we will show ...'),
    -> ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
    -> ('MySQL vs. YourSQL','In the following database comparison ...'),
    -> ('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()函数对于一个字符串执行资料库内的自然语言搜索。一个资料库就是1套1个或2个包含在FULLTEXT内的列。搜索字符串作为对AGAINST()的参数而被给定。对于表中的每一行, MATCH() 返回一个相关值，即, 搜索字符串和 MATCH()表中指定列中该行文字之间的一个相似性度量。

在默认状态下, 搜索的执行方式为不区分大小写方式。然而,你可以通过对编入索引的列使用二进制排序方式执行区分大小写的全文搜索。 例如，可以向一个使用latin1字符集的列给定latin1_bin 的排序方式，对于全文搜索区分大小写。

如上述所举例子，当MATCH()被用在一个 WHERE 语句中时，相关值是非负浮点数。零相关的意思是没有相似性。相关性的计算是基于该行中单词的数目, 该行中独特子的数目，资料库中单词的总数，以及包含特殊词的文件(行)数目。

对于自然语言全文搜索，要求MATCH() 函数中命名的列和你的表中一些FULLTEXT索引中包含的列相同。对于前述问讯， 注意，MATCH()函数(题目及全文)中所命名的列和文章表的FULLTEXT索引中的列相同。若要分别搜索题目和全文，应该对每个列创建FULLTEXT索引。

或者也可以运行布尔搜索或使用查询扩展进行搜索。关于这些搜索类型的说明见12.7.1节，“布尔全文搜索”和12.7.2节，“全文搜索带查询扩展”。

上面的例子基本上展示了怎样使用返回行的相关性顺序渐弱的MATCH()函数。而下面的例子则展示了怎样明确地检索相关值。返回行的顺序是不定的，原因是 SELECT 语句不包含 WHERE或ORDER BY 子句：

```
mysql> SELECT id, MATCH (title,body) AGAINST ('Tutorial')
    -> FROM articles;
+----+-----------------------------------------+
| id | MATCH (title,body) AGAINST ('Tutorial') |
+----+-----------------------------------------+
|  1 |                       0.65545833110809 |
|  2 |                                      0 |
|  3 |                       0.66266459226608 |
|  4 |                                      0 |
|  5 |                                      0 |
|  6 |                                      0 |
+----+-----------------------------------------+
6 rows in set (0.00 sec)
```

下面的例子则更加复杂。询问返回相关值，同时对行按照相关性渐弱的顺序进行排序。为实现这个结果，你应该两次指定 MATCH()： 一次在 SELECT 列表中而另一次在 WHERE子句中。这不会引起额外的内务操作，原因是MySQL 优化程序注意到两个MATCH()调用是相同的，从而只会激活一次全文搜索代码。

```
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. ... | 1.5219271183014 |
|  6 | When configured properly, MySQL ... | 1.3114095926285 |
+----+-------------------------------------+-----------------+
2 rows in set (0.00 sec)
```

表中有2行(0.00 秒)

MySQL FULLTEXT 执行将任何单字字符原形 (字母、数字和下划线部分)的序列视为一个单词。这个序列或许也包含单引号 (')，但在一行中不会超过一个。 这意味着 aaa'bbb 会被视为一个单词，而 aaa''bbb则被视为2个单词。位于单词之前或其后的单引号会被FULLTEXT分析程序去掉； 'aaa'bbb' 会变成 aaa'bbb。

FULLTEXT分析程序会通过寻找某些分隔符来确定单词的起始位置和结束位置，例如' ' (间隔符号)、 , (逗号)以及 . (句号)。假如单词没有被分隔符分开，(例如在中文里)，则 FULLTEXT 分析程序不能确定一个词的起始位置和结束位置。为了能够在这样的语言中向FULLTEXT 索引添加单词或其它编入索引的术语，你必须对它们进行预处理，使其被一些诸如"之类的任意分隔符分隔开。

一些词在全文搜索中会被忽略:

- 任何过于短的词都会被忽略。 全文搜索所能找到的词的默认最小长度为 4个字符。
- 停止字中的词会被忽略。禁用词就是一个像"the" 或"some" 这样过于平常而被认为是不具语义的词。存在一个内置的停止字， 但它可以通过用户自定义列表被改写。请参见12.7.5节，"微调MySQL全文搜索"。

默认的停止字在12.7.3节，"全文停止字"中被给出。默认的最小单词长度和 停止字可以被改变，如12.7.5节，"微调MySQL全文搜索"中所述。

词库和询问中每一个正确的单词根据其在词库和询问中的重要性而被衡量。 通过这种方式，一个出现在许多文件中的单词具有较低的重要性(而且甚至很多单词的重要性为零),原因是在这个特别词库中其语义价值较低。反之，假如这个单词比较少见，那么它会得到一个较高的重要性。然后单词的重要性被组合，从而用来计算该行的相关性。

这项技术最适合同大型词库一起使用 (事实上, 此时它经过仔细的调整)。对于很小的表，单词分布并不能充分反映它们的语义价值， 而这个模式有时可能会产生奇特的结果。例如, 虽然单词 "MySQL" 出现在文章表中的每一行，但对这个词的搜索可能得不到任何结果:

mysql> **SELECT * FROM articles**

-> **WHERE MATCH (title,body) AGAINST ('MySQL');**

找不到搜索的词(0.00 秒)

这个搜索的结果为空，原因是单词 "MySQL" 出现在至少全文的50%的行中。 因此, 它被列入停止字。对于大型数据集，使用这个操作最合适不过了----一个自然语言问询不会从一个1GB 的表每隔一行返回一次。对于小型数据集，它的用处可能比较小。

一个符合表中所有行的内容的一半的单词查找相关文档的可能性较小。事实上, 它更容易找到很多不相关的内容。我们都知道，当我们在因特网上试图使用搜索引擎寻找资料的时候，这种情况发生的频率颇高。可以推论，包含该单词的行因其所在特别数据集 而被赋予较低的语义价值。 一个给定的词有可能在一个数据集中拥有超过其50%的域值，而在另一个数据集却不然。

当你第一次尝试使用全文搜索以了解其工作过程时，这个50% 的域值提供重要的蕴涵操作：若你创建了一个表，并且只将文章的1、2行插入其中， 而文中的每个单词在所有行中出现的机率至少为 50% 。那么结果是你什么也不会搜索到。一定要插入至少3行，并且多多益善。需要绕过该50% 限制的用户可使用布尔搜索代码；见12.7.1节，"布尔全文搜索"。

12.7.1. 布尔全文搜索

利用IN BOOLEAN MODE修改程序， MySQL 也可以执行布尔全文搜索:

```
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 Well | 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 ... |
+----+-----------------------+-------------------------------------+
```

这个问询检索所有包含单词"MySQL"的行，但不检索包含单词"YourSQL"的行。

布尔全文搜索具有以下特点:

- 它们不使用 50% 域值。.
- 它们不会按照相关性渐弱的顺序将行进行分类。你可以从上述问询结果中看到这一点：相关性最高的行是一个包含两个"MySQL" 的行，但它被列在最后的位置，而不是开头位置。
- 即使没有FULLTEXT，它们仍然可以工作，尽管这种方式的搜索执行的速度非常之慢。
- 最小单词长度全文参数和最大单词长度全文参数均适用。
- 停止字适用。

布尔全文搜索的性能支持以下操作符:

- +

一个前导的加号表示该单词必须 出现在返回的每一行的开头位置。

- -

一个前导的减号表示该单词一定不能出现在任何返回的行中。

- (无操作符)

在默认状态下(当没有指定 + 或–的情况下)，该单词可有可无，但含有该单词的行等级较高。这和MATCH() ... AGAINST()不使用IN BOOLEAN MODE修改程序时的运作很类似。

- > <

这两个操作符用来改变一个单词对赋予某一行的相关值的影响。 > 操作符增强其影响，而 <操作符则减弱其影响。请参见下面的例子。

- ( )

括号用来将单词分成子表达式。括入括号的部分可以被嵌套。

- ~

一个前导的代字号用作否定符， 用来否定单词对该行相关性的影响。 这对于标记"noise（无用信息）"的单词很有用。包含这类单词的行较其它行等级低，但因其可能会和-号同时使用，因而不会在任何时候都派出所有无用信息行。

- *

星号用作截断符。于其它符号不同的是，它应当被追加到要截断的词上。

- "

一个被括入双引号的短语 ('") 只和字面上包含该短语输入格式的行进行匹配。全文引擎将短语拆分成单词，在FULLTEXT索引中搜索该单词。　非单词字符不需要严密的匹配：短语搜索只要求符合搜索短语包含的单词且单词的排列顺序相同的内容。例如， "test phrase" 符合 "test, phrase"。

若索引中不存在该短语包含的单词，则结果为空。例如，若所有单词都是禁用词，或是长度都小于编入索引单

词的最小长度，则结果为空。

以下例子展示了一些使用布尔全文符号的搜索字符串:

- 'apple banana'

寻找包含至少两个单词中的一个的行。

- '+apple +juice'

寻找两个单词都包含的行。

- '+apple macintosh'

寻找包含单词"apple"的行，若这些行也包含单词"macintosh"， 则列为更高等级。

- '+apple -macintosh'

寻找包含单词"apple" 但不包含单词 "macintosh"的行。

- '+apple +(>turnover <strudel)'

寻找包含单词"apple"和"turnover" 的行，或包含"apple" 和"strudel"的行 (无先后顺序),然而包含 "apple turnover"的行较包含"apple strudel"的行排列等级更为高。

- 'apple*'

寻找包含"apple"、"apples"、"applesauce"或"applet"的行。

- '"some words"'

寻找包含原短语"some words"的行 (例如,包含"some words of wisdom" 的行，而非包含 "some noise words"的行)。注意包围词组的'"' 符号是界定短语的操作符字符。它们不是包围搜索字符串本身的引号。

12.7.2. 全文搜索带查询扩展

全文搜索支持查询扩展功能 (特别是其多变的"盲查询扩展功能")。若搜索短语的长度过短, 那么用户则需要依靠全文搜索引擎通常缺乏的内隐知识进行查询。这时，查询扩展功能通常很有用。例如, 某位搜索 "database" 一词的用户，可能认为"MySQL"、"Oracle"、"DB2" and "RDBMS"均为符合 "databases"的项，因此都应被返回。这既为内隐知识。

在下列搜索短语后添加WITH QUERY EXPANSION，激活盲查询扩展功能(即通常所说的自动相关性反馈)。它将执行两次搜索，其中第二次搜索的搜索短语是同第一次搜索时找到的少数顶层文件连接的原始搜索短语。这样,假如这些文件中的一个 含有单词 "databases" 以及单词 "MySQL", 则第二次搜索会寻找含有单词"MySQL" 的文件，即使这些文件不包含单词 "database"。下面的例子显示了这个不同之处:

```
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)


mysql> SELECT * FROM articles

    -> WHERE MATCH (title,body)

    -> AGAINST ('database' WITH QUERY EXPANSION);

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

| id | title             | body                                    |

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

|  1 | MySQL Tutorial    | DBMS stands for DataBase ...            |

|  5 | MySQL vs. YourSQL | In the following database comparison ... |

|  3 | Optimizing MySQL  | In this tutorial we will show ...       |

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

3 rows in set (0.00 sec)
```

另一个例子是Georges Simenon 搜索关于Maigret的书籍, 这个用户不确定“Maigret”一词的拼法。若不使用查询扩展而搜索“Megre and the reluctant witnesses” 得到的结果只能是的“Maigret and the Reluctant Witnesses” 。 而带有查询扩展的搜索会在第二遍得到带有“Maigret”一词的所有书名。

注释： 盲查询扩展功能很容易返回非相关文件而增加无用信息，因此只有在查询一个长度很短的短语时才有必要使用这项功能。

12.7.3. 全文停止字

以下表列出了默认的全文停止字:

a's	able	about	above	according
accordingly	across	actually	after	afterwards
again	against	ain't	all	allow
allows	almost	alone	along	already
also	although	always	am	among
amongst	an	and	another	any
anybody	anyhow	anyone	anything	anyway
anyways	anywhere	apart	appear	appreciate
appropriate	are	aren't	around	as
aside	ask	asking	associated	at
available	away	awfully	be	became
because	become	becomes	becoming	been
before	beforehand	behind	being	believe
below	beside	besides	best	better
between	beyond	both	brief	but
by	c'mon	c's	came	can
can't	cannot	cant	cause	causes
certain	certainly	changes	clearly	co

com	come	comes	concerning	consequently
consider	considering	contain	containing	contains
corresponding	could	couldn't	course	currently
definitely	described	despite	did	didn't
different	do	does	doesn't	doing
don't	done	down	downwards	during
each	edu	eg	eight	either
else	elsewhere	enough	entirely	especially
et	etc	even	ever	every
everybody	everyone	everything	everywhere	ex
exactly	example	except	far	few
fifth	first	five	followed	following
follows	for	former	formerly	forth
four	from	further	furthermore	get
gets	getting	given	gives	go
goes	going	gone	got	gotten
greetings	had	hadn't	happens	hardly
has	hasn't	have	haven't	having
he	he's	hello	help	hence
her	here	here's	hereafter	hereby
herein	hereupon	hers	herself	hi
him	himself	his	hither	hopefully
how	howbeit	however	i'd	i'll
i'm	i've	ie	if	ignored
immediate	in	inasmuch	inc	indeed
indicate	indicated	indicates	inner	insofar
instead	into	inward	is	isn't
it	it'd	it'll	it's	its
itself	just	keep	keeps	kept
know	knows	known	last	lately
later	latter	latterly	least	less
lest	let	let's	like	liked
likely	little	look	looking	looks
ltd	mainly	many	may	maybe
me	mean	meanwhile	merely	might
more	moreover	most	mostly	much
must	my	myself	name	namely
nd	near	nearly	necessary	need
needs	neither	never	nevertheless	new
next	nine	no	nobody	non
none	noone	nor	normally	not
nothing	novel	now	nowhere	obviously

of	off	often	oh	ok
okay	old	on	once	one
ones	only	onto	or	other
others	otherwise	ought	our	ours
ourselves	out	outside	over	overall
own	particular	particularly	per	perhaps
placed	please	plus	possible	presumably
probably	provides	que	quite	qv
rather	rd	re	really	reasonably
regarding	regardless	regards	relatively	respectively
right	said	same	saw	say
saying	says	second	secondly	see
seeing	seem	seemed	seeming	seems
seen	self	selves	sensible	sent
serious	seriously	seven	several	shall
she	should	shouldn't	since	six
so	some	somebody	somehow	someone
something	sometime	sometimes	somewhat	somewhere
soon	sorry	specified	specify	specifying
still	sub	such	sup	sure
t's	take	taken	tell	tends
th	than	thank	thanks	thanx
that	that's	thats	the	their
theirs	them	themselves	then	thence
there	there's	thereafter	thereby	therefore
therein	theres	thereupon	these	they
they'd	they'll	they're	they've	think
third	this	thorough	thoroughly	those
though	three	through	throughout	thru
thus	to	together	too	took
toward	towards	tried	tries	truly
try	trying	twice	two	un
under	unfortunately	unless	unlikely	until
unto	up	upon	us	use
used	useful	uses	using	usually
value	various	very	via	viz
vs	want	wants	was	wasn't
way	we	we'd	we'll	we're
we've	welcome	well	went	were
weren't	what	what's	whatever	when
whence	whenever	where	where's	whereafter
whereas	whereby	wherein	whereupon	wherever

whether	which	while	whither	who
who's	whoever	whole	whom	whose
why	will	willing	wish	with
within	without	won't	wonder	would
would	wouldn't	yes	yet	you
you'd	you'll	you're	you've	your
yours	yourself	yourselves	zero	

12.7.4. 全文限定条件

- 全文搜索只适用于 MyISAM 表。
- 全文搜索可以同大多数多字节字符集一起使用。Unicode属于例外情况; 可使用utf8 字符集 , 而非ucs2字符集。
- 诸如汉语和日语这样的表意语言没有自定界符。因此， FULLTEXT分析程序不能确定在这些或其它的这类语言中词的起始和结束的位置。其隐含操作及该问题的一些工作区在12.7节，“全文搜索功能”有详细论述。
- 若支持在一个单独表中使用多字符集，则所有 FULLTEXT索引中的列 必须使用同样的字符集和库。
- MATCH()列列表必须同该表中一些 FULLTEXT索引定义中的列列表完全符合,除非MATCH()在IN BOOLEAN MODE。
- 对AGAINST() 的参数必须是一个常数字符串。

12.7.5. 微调**MySQL**全文搜索

MySQL的全文搜索容量几乎不具有用户调节参数。假如你拥有一个 MySQL源分布，你就能对全文搜索性能行使更多控制，原因是一些变化需要源代码修改。请参见2.8节，“使用源码分发版安装MySQL”。

注意，为了更加有效，需要对全文搜索谨慎调节。实际上，在大多数情况下修改默认性能只能降低其性能。 除非你知道自己在做什么，否则不要改变 *MySQL*源。

下述的大多数全文变量必须在服务器启动时被设置。为了改变它们，还要重新启动服务器；在服务器正在运行期间，他们不会被改变。

一些变量的改变需要你重建表中的 FULLTEXT 索引。本章结尾部分给出了其有关操作说明。

- ft_min_word_len and ft_max_word_len系统自变量规定了被编入索引单词的最小长度和最大长度。(见5.3.3节，“服务器系统变量”.) 默认的最小值为四个字符；默认的最大值取决于使用的 MySQL 版本。假如你改变任意一个值，那么你必须重建你的 FULLTEXT索引。 例如，若你希望一个3字符的单词变为可查找项，则可以通过将以下行移动到一个供选择文件里，从而设置 ft_min_word_len 变量：

· [mysqld]

· ft_min_word_len=3

然后重新启动服务器，重建你的 FULLTEXT索引。同时还要特别注意该表后面的说明中的关于**myisamchk**的注释。

- 若要覆盖默认停止字，则可设置 ft_stopword_file 系统变量。 (见5.3.3节，“服务器系统变量”) 。 变量值应为包含停止字的文件路径名, 或是用来截止禁用词过滤的空字符串。在改变了这个变量的值或禁用词文件的内容后, 重建你的 FULLTEXT索引。

停止字是自由形态的，换言之，你可使用任何诸如newline、 space或comma这样的非字母数字字符来分隔禁用词。 下划线字符(_) 和被视为单词的一部分的单引号 (')例外。停止字字符集为服务器默认字符集；

见10.3.1节，“服务器字符集和校对”.

- 自然语言查询的50%阈值由所选择的特别权衡方案所决定。若要阻止它，myisam/ftdefs.h 中寻找以下行：

· #define GWS_IN_USE GWS_PROB

将该行改为：

#define GWS_IN_USE GWS_FREQ

然后重新编译 MySQL。此时不需要重建索引。注释：这样做你会严重的By 降低 MySQL为MATCH()函数提供合适的相关值得能力。假如你争得需要搜索这样的普通词，而使用IN BOOLEAN MODE代替的效果更好，因为它不遵循 50% 阈值。

- 要改变用于布尔全文搜索的操作符，设置 ft_boolean_syntax 系统变量。 这个变量也可以在服务器运行时被改变，但你必须有SUPER 特权才能这么做。在这种情况下不需要重建索引。 见5.3.3节，“服务器系统变量”, 它向我们说明了怎样使用这个变量的规则。

假如你改变了影响索引的全文变量 (ft_min_word_len、 ft_max_word_len或ft_stopword_file),或假如你改变了禁用词文件本身，则你必须在改变和重新启动服务器后重建你的 FULLTEXT索引。这时，要重建索引， 只需进行一个 QUICK 修理操作：

mysql> **REPAIR TABLE** *tbl_name* **QUICK;**

注意，假如你使用 **myisamchk** 来执行一项修改表索引的操作 (诸如修理或分析), 则使用最小单词长度和最大单词长度以及停止字的默认全文参数值重建FULLTEXT索引，除非你已另外指定。这会导致问询失败。

发生这个问题的原因是只有服务器认识这些参数。它们的存储位置不在 MyISAM 索引文件中。若你已经修改了最小单词长度或最大单词长度或服务器中的停止字，为避免这个问题，为你对**mysqld**所使用的**myisamchk**指定同样的 ft_min_word_len、 ft_max_word_len和ft_stopword_file值。例如，假如你已经将最小单词长度设置为 3, 则你可以这样修改一个带有**myisamchk**的表：

shell> **myisamchk --recover --ft_min_word_len=3** *tbl_name***.MYI**

为保证 **myisamchk** 及服务器对全文参数使用相同的值， 可将每一项都放在供选文件中的 [mysqld]和 [myisamchk] 部分：

[mysqld]

ft_min_word_len=3

[myisamchk]

ft_min_word_len=3

使用 REPAIR TABLE、 ANALYZE TABLE、OPTIMIZE TABLE或ALTER TABLE来代替使用 **myisamchk** 。这些语句通过服务器来执行，服务器知道使用哪个全文参数值更加合适。

12.8. Cast函数和操作符

- BINARY

BINARY操作符将后面的字符串抛给一个二进制字符串。这是一种简单的方式来促使逐字节而不是逐字符的进

行列比较。这使得比较区分大小写，即使该列不被定义为 BINARY或 BLOB。BINARY也会产生结尾空白，从而更加显眼。

mysql> **SELECT 'a' = 'A';**

-> 1

mysql> **SELECT BINARY 'a' = 'A';**

-> 0

mysql> **SELECT 'a' = 'a ';**

-> 1

mysql> **SELECT BINARY 'a' = 'a ';**

-> 0

BINARY影响整个比较；它可以在任何操作数前被给定，而产生相同的结果。

BINARY *str* 是CAST(*str* AS BINARY)的缩略形式。

注意，在一些语境中，假如你将一个编入索引的列派给BINARY, MySQL 将不能有效使用这个索引。

假如你想要将一个 BLOB值或其它二进制字符串进行区分大小写的比较，你可利用二进制字符串没有字符集这一事实实现这个目的，这样就不会有文书夹的概念。为执行一个区分大小写的比较，可使用 CONVERT()函数将一个字符串值转化为一个不区分大小写的字符集。其结果为一个非二进制字符串，因此 LIKE 操作也不会区分大小写：

SELECT 'A' LIKE CONVERT(*blob_col* USING latin1) FROM *tbl_name*;

若要使用一个不同的字符集, 替换其在上述语句中的latin1名。

CONVERT()一般可用于比较出现在不同字符集中的字符串。

- CAST(*expr* AS *type*), CONVERT(*expr*,*type*) , CONVERT(*expr* USING *transcoding_name*)

CAST() 和CONVERT() 函数可用来获取一个类型的值，并产生另一个类型的值。

这个类型 可以是以下值其中的 一个：

  - BINARY[(*N*)]
  - CHAR[(*N*)]
  - DATE
  - DATETIME
  - DECIMAL
  - SIGNED [INTEGER]
  - TIME
  - UNSIGNED [INTEGER]

BINARY 产生一个二进制字符串。关于它怎样影响比较结果的说明见本章中 BINARY操作符项。

假如给定了随意长度*N*，则 BINARY[*N*] 使 cast使用该参数的不多于 *N* 个字节。同样的， CHAR[*N*]会使 cast使用该参数的不多于*N* 个字符。

CAST() and CONVERT(... USING ...) 是标准 SQL语法。CONVERT()的非USING 格式是ofis ODBC语法。

带有USING的CONVERT() 被用来在不同的字符集之间转化数据。在 MySQL中, 自动译码名和相应的字符集名称相同。例如。 这个语句将服务器的默认字符集中的字符串 'abc'转化为utf8字符集中相应的字符串:

SELECT CONVERT('abc' USING utf8);

当你想要在一个CREATE ... SELECT 语句中创建一个特殊类型的列，则cast函数会很有用:

CREATE TABLE new_table SELECT CAST('2000-01-01' AS DATE);

该函数也用于ENUM 列按词法顺序的排序。通常ENUM列的排序在使用内部数值时发生。将这些值按照词法顺序派给 CHAR 结果:

SELECT *enum_col* FROM *tbl_name* ORDER BY CAST(*enum_col* AS CHAR);

CAST(*str* AS BINARY)和BINARY *str*相同。 CAST(*expr* AS CHAR) 将表达式视为一个带有默认字符集的字符串。

若用于一个诸如 CONCAT('Date: ',CAST(NOW() AS DATE))这样的比较复杂的表达式的一部分，CAST()也会改变结果。

你不应在不同的格式中使用 CAST() 来析取数据，但可以使用诸如LEFT() 或 EXTRACT() 的样的字符串函数来代替。请参见12.5节，“日期和时间函数”。

若要在数值语境中将一个字符串派给一个数值, 通常情况下，除了将字符串值作为数字使用外，你不需要做任何事:

mysql> **SELECT 1+'1';**

-> 2

若要在一个字符串语境中使用一个数字，该数字会被自动转化为一个BINARY 字符串。

mysql> **SELECT CONCAT('hello you ',2);**

-> 'hello you 2'

MySQL 支持带符号和无符号的64比特值的运算。若你正在使用数字操作符 (如 +) 而其中一个操作数为无符号整数，则结果为无符号。可使用SIGNED 和UNSIGNED cast 操作符来覆盖它。将运算分别派给带符号或无符号64比特整数。

mysql> **SELECT CAST(1-2 AS UNSIGNED)**

-> 18446744073709551615

mysql> **SELECT CAST(CAST(1-2 AS UNSIGNED) AS SIGNED);**

-> -1

注意，假如任意一个操作数为一个浮点值，则结果为一个浮点值， 且不会受到上述规则影响 (关于这一点, DECIMAL 列值被视为浮点值）。

mysql> **SELECT CAST(1 AS UNSIGNED) - 2.0;**

-> -1.0

若你在一个算术运算中使用了一个字符串，它会被转化为一个浮点数。

12.9. 其他函数

12.9.1. 位函数
12.9.2. 加密函数
12.9.3. 信息函数
12.9.4. 其他函数

12.9.1. 位函数

对于比特运算，MySQL 使用 BIGINT (64比特) 算法，因此这些操作符的最大范围是 64 比特。

- |

Bitwise OR:

mysql> **SELECT 29 | 15;**

-> 31

其结果为一个64比特无符号整数。

- &

Bitwise AND:

mysql> **SELECT 29 & 15;**

-> 13

其结果为一个64比特无符号整数。

- ^

Bitwise XOR:

mysql> **SELECT 1 ^ 1;**

-> 0

mysql> **SELECT 1 ^ 0;**

-> 1

mysql> **SELECT 11 ^ 3;**

-> 8

结果为一个64比特无符号整数。

- <<

把一个longlong (BIGINT)数左移两位。

mysql> **SELECT 1 << 2;**

-> 4

其结果为一个64比特无符号整数。

- >>

把一个longlong (BIGINT)数右移两位。

mysql> **SELECT 4 >> 2;**

-> 1

其结果为一个64比特无符号整数。

- ~

反转所有比特。

mysql> **SELECT 5 & ~1;**

-> 4

其结果为一个64比特无符号整数。

- BIT_COUNT(*N*)

返回参数*N* 中所设置的比特数

mysql> **SELECT BIT_COUNT(29);**

-> 4

12.9.2. 加密函数

本节介绍了加密和加密值。若你想要储存一些由可能包含任意字节值的加密函数返回的结果，使用BLOB列而不是 CHAR 或VARCHAR 列，从而避免由于结尾空格的删除而改变一些数据值的潜在问题。

- AES_ENCRYPT(*str*,*key_str*) , AES_DECRYPT(*crypt_str*,*key_str*)

这些函数允许使用官方AES进行加密和数据加密 (高级加密标准) 算法, 即以前人们所熟知的 "Rijndael"。 保密关键字的长度为128比特，不过你可以通过改变源而将其延长到 256 比特。我们选择了 128比特的原因是它的速度要快得多，且对于大多数用途而言这个保密程度已经够用。

输入参数可以为任何长度。若任何一个参数为NULL,则函数的结果也是NULL。

因为 AES 是块级算法，使用填充将不均衡长度字符串编码，这样结果字符串的长度的算法为 16 * (trunc(*string_length* / 16) + 1)。

若 AES_DECRYPT()检测到无效数据或不正确填充，它会返回 NULL。然而，若输入的资料或密码无效时，AES_DECRYPT()有可能返回一个非 NULL 值 (可能为无用信息)。

你可以通过修改你的问询，从而使用AES函数以加密形式来存储数据:

INSERT INTO t VALUES (1,AES_ENCRYPT('text','password'));

AES_ENCRYPT()和AES_DECRYPT() 可以被看作MySQL中普遍通用的密码最安全的加密函数。

- DECODE(*crypt_str*,*pass_str*)

使用 *pass_str* 作为密码，解密加密字符串 *crypt_str*， *crypt_str* 应该是由ENCODE()返回的字符串。

- ENCODE(*str*,*pass_str*)

使用*pass_str* 作为密码，解密 *str* 。 使用DECODE()解密结果。

结果是一个和*str*长度相同的二进制字符串。若你想要将其保留在一个列中，可使用 BLOB 列类型。

- DES_DECRYPT(*crypt_str*[,*key_str*])

使用DES_ENCRYPT()加密一个字符串。若出现错误，这个函数会返回 NULL。

注意，这个函数只有当MySQL在SSL 的支持下配置完毕时才会运作。请参见5.8.7节，“使用安全连接”.

假如没有给定 *key_str* 参数， DES_DECRYPT() 会首先检查加密字符串的第一个字节， 从而确定用来加密原始字符串的DES密码关键字数字，之后从DES关键字文件中读取关键字从而解密信息。为使其运行，用户必须享有 SUPER 特权。可以选择--des-key-file服务器指定关键字文件。

假如你向这个函数传递一个*key_str* 参数，该字符串被用作解密信息的关键字。

若 *crypt_str* 参数看起来不是一个加密字符串， MySQL 会返回给定的 *crypt_str*。

- DES_ENCRYPT(*str*[,(*key_num*|*key_str*)])

用Triple-DES 算法给出的关键字加密字符串。若出现错误，这个函数会返回NULL。

注意，这个函数只有当MySQL 在SSL的支持下配置完毕后才会运行。请参见5.8.7节，“使用安全连接”.

使用的加密关键字的选择基于第二个到 DES_ENCRYPT()的参数，假如给定:

参数	说明
无参数	使用来自DES关键字文件的第一个关键字。
key_num	使用DES 关键字文件给出的关键字数字(0-9)。
key_str	使用给出的关键字字符串为 *str* 加密。

选择--des-key-file服务器指定关键字文件。

返回字符串是一个二进制字符串，其中第一个字符为 CHAR(128 | key_num)。

加上 128使得识别加密关键字更加容易。若你使用一个字符串关键字，则 *key_num* 为127。

结果的字符串长度为 *new_len* = *orig_len* + (8-(*orig_len* % 8))+1。

DES关键字文件中的每一行都具有如下格式:

key_num des_key_str

每个*key_num* 必须是一个从0到0范围内的数字。文件中行的排列顺序是任意的。 *des_key_str* 是用来加密信息的字符串。在数字和关键字之间应该至少有一个空格。若你未指定任何到DES_ENCRYPT()的关键字参数，则第一个关键字为默认的使用关键字。

使用FLUSH DES_KEY_FILE语句，你可以让 MySQL从关键字文件读取新的关键字值。这要求你享有 RELOAD特

权。

拥有一套默认关键字的一个好处就是它向应用程序提供了一个检验加密列值的方式，而无须向最终用户提供解密这些值的权力。

mysql> **SELECT customer_address FROM customer_table**

> WHERE crypted_credit_card = DES_ENCRYPT('credit_card_number');

- ENCRYPT(*str*[,*salt*])

使用Unix crypt() 系统调用加密 *str*。 *salt* 参数应为一个至少包含2个字符的字符串。若没有给出 *salt* 参数，则使用任意值。

mysql> **SELECT ENCRYPT('hello');**

-> 'VxuFAJXVARROc'

至少在一些系统中，ENCRYPT()除了s*tr*的前八位字符之外会忽略所有内容。这个行为由下划线的crypt() 系统调用的执行所决定。

假如crypt()在你的系统中不可用(正如在 Windows系统), ENCRYPT() 则会始终返回NULL。鉴于这个原因，我们向你推荐使用 MD5() 或SHA1() 来代替,因为这两个函数适合所有的平台。

- MD5(*str*)

为字符串算出一个 MD5 128比特检查和。该值以32位十六进制数字的二进制字符串的形式返回, 若参数为 NULL 则会返回 NULL。例如，返回值可被用作散列关键字。

mysql> **SELECT MD5('testing');**

-> 'ae2b1fca515949e5d54fb22b8ed95575'

这是"RSA Data Security, Inc. MD5 Message-Digest Algorithm."

假如你想要将这个值转化为大写字母，参见12.8节，“Cast函数和操作符”中BINARY操作符项中给出的二进制字符串转换。

- OLD_PASSWORD(*str*)

当PASSWORD()的执行变为改善安全性时，OLD_PASSWORD()会被添加到 MySQL。OLD_PASSWORD()返回从前的PASSWORD()执行值(4.1之前)，同时允许你为任何4.1 之前的需要连接到你的5.1 版本MySQL服务器前客户端设置密码，从而不至于将它们切断。请参见5.7.9节，“MySQL 4.1中的密码哈希处理”。

- PASSWORD(*str*)

从原文密码*str* 计算并返回密码字符串，当参数为 NULL 时返回 NULL。这个函数用于用户授权表的Password列中的加密MySQL密码存储

mysql> **SELECT PASSWORD('badpwd');**

-> '7f84554057dd964b'

PASSWORD() 加密是单向的 (不可逆)。

PASSWORD() 执行密码加密与Unix 密码被加密的方式不同。请参见ENCRYPT()。

注释： PASSWORD()函数在MySQL服务器中的鉴定系统使用；你不应将它用在你个人的应用程序中。为达到同样目的，可使用 MD5()或SHA1() 代替。 更多关于在您的应用程序中处理密码及安全鉴定的信息见RFC 2195

- SHA1(*str*) SHA(*str*)

为字符串算出一个 SHA1 160比特检查和，如RFC 3174 (安全散列算法)中所述。该值被作为40位十六进制数字返回，而当参数为NULL 时则返回 NULL。这个函数的一个可能的用处就在于其作为散列关键字。你也可以将其作为存储密码的密码安全函数使用。

mysql> **SELECT SHA1('abc');**

-> 'a9993e364706816aba3e25717850c26c9cd0d89d'

SHA1()可以被视为一个密码更加安全的函数，相当于 MD5()。 SHA() 和SHA1()具有相同的意义。

12.9.3. 信息函数

- BENCHMARK(*count*,*expr*)

BENCHMARK() 函数重复*count* 次执行表达式 *expr* 。 它可以被用于计算 MySQL 处理表达式的速度。结果值通常为 0。另一种用处来自 **mysql**客户端内部,能够报告问询执行的次数:

```
mysql> SELECT BENCHMARK(1000000,ENCODE('hello','goodbye'));
+----------------------------------------------+
| BENCHMARK(1000000,ENCODE('hello','goodbye')) |
+----------------------------------------------+
|                                            0 |
+----------------------------------------------+
1 row in set (4.74 sec)
```

此处报告的时间是客户端上的共用时间，而不是服务器端上的CPU时间。建议执行多遍BENCHMARK()，并解释与服务器机器负荷程度有关的结果。

· CHARSET(*str*)

返回字符串自变量的字符集。

```
mysql> SELECT CHARSET('abc');
        -> 'latin1'
mysql> SELECT CHARSET(CONVERT('abc' USING utf8));
        -> 'utf8'
mysql> SELECT CHARSET(USER());
        -> 'utf8'
```

· COERCIBILITY(*str*)

返回字符串自变量的整序可压缩性值。

```
mysql> SELECT COERCIBILITY('abc' COLLATE latin1_swedish_ci);
        -> 0
```

```
mysql> SELECT COERCIBILITY(USER());
        -> 3
mysql> SELECT COERCIBILITY('abc');
        -> 4
```

返回值具有如下意义:

可压缩性	意义	举例
0	明确排序	带有COLLATE 子句的值
1	无排序	不同排序的字符串连接
2	明确排序	列值
3	系统常量	USER()返回值
4	可压缩	文字字符串
5	可忽略	NULL得来的NULL或一个表达式

下方值得优先级较高。

- COLLATION(*str*)

返回惠字符串参数的排序方式。

mysql> **SELECT COLLATION('abc');**

-> 'latin1_swedish_ci'

mysql> **SELECT COLLATION(_utf8'abc');**

-> 'utf8_general_ci'

- CONNECTION_ID()

返回对于连接的连接ID (线程ID)。每个连接都有各自的唯一 ID。

mysql> **SELECT CONNECTION_ID();**

-> 23786

- CURRENT_USER, CURRENT_USER()

返回当前话路被验证的用户名和主机名组合。这个值符合确定你的存取权限的MySQL 账户。在被指定SQL SECURITY DEFINER特征的存储程序内， CURRENT_USER() 返回程序的创建者。

CURRENT_USER()的值可以和USER()的值有所不同。

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'

这个例子解释了虽然客户端指定了一个 davida用户名 (正如USER()函数的值所指示的), 服务器却使用一个匿名的用户账户确认该客户端 (见CURRENT_USER()值得空用户名部分)。这种情况发生的一个原因是 One 在向davida的授权列表中没有足够的账户。

CURRENT_USER() 返回的字符串使用 utf8字符集。

- DATABASE()

返回使用 utf8 字符集的默认(当前)数据库名。在存储程序里，默认数据库是同该程序向关联的数据库，但并不一定与调用语境的默认数据库相同。

mysql> **SELECT DATABASE();**

-> 'test'

若没有默认数据库, DATABASE()返回 NULL。

- FOUND_ROWS()

A SELECT语句可能包括一个 LIMIT 子句，用来限制服务器返回客户端的行数。在有些情况下，需要不用再次运行该语句而得知在没有LIMIT 时到底该语句返回了多少行。为了知道这个行数, 包括在SELECT 语句中选择SQL_CALC_FOUND_ROWS ，随后调用 FOUND_ROWS() ：

mysql> **SELECT SQL_CALC_FOUND_ROWS * FROM** *tbl_name*

-> **WHERE id > 100 LIMIT 10;**

mysql> **SELECT FOUND_ROWS();**

第二个 SELECT返回一个数字，指示了在没有LIMIT子句的情况下，第一个SELECT返回了多少行 (若上述的SELECT语句不包括 SQL_CALC_FOUND_ROWS 选项,则使用LIMIT 和不使用时，FOUND_ROWS() 可能会返回不同的结果）。

通过 FOUND_ROWS()的有效行数是瞬时的，并且不用于越过SELECT SQL_CALC_FOUND_ROWS语句后面的语句。若你需要稍候参阅这个值，那么将其保存:

mysql> **SELECT SQL_CALC_FOUND_ROWS * FROM ... ;**

mysql> **SET @rows = FOUND_ROWS();**

假如你正在使用 SELECT SQL_CALC_FOUND_ROWS, MySQL 必须计算出在全部结果集合中有所少行。然而，这比不用LIMIT而再次运行问询要快，原因是结果集合不需要被送至客户端。

SQL_CALC_FOUND_ROWS 和 FOUND_ROWS() 在当你希望限制一个问询返回的行数时很有用，同时还能不需要再次运行问询而确定全部结果集合中的行数。一个例子就是提供页式显示的Web脚本，该显示包含显示搜索结果其它部分的页的连接。使用FOUND_ROWS() 使你确定剩下的结果需要多少其它的页。

SQL_CALC_FOUND_ROWS 和 FOUND_ROWS() 的应用对于UNION 问询比对于简单SELECT 语句更为复杂，原因是在UNION 中，LIMIT 可能会出现在多个位置。它可能适用于UNION中的个人 SELECT语句，或是总体上

到UNION 结果的全程。

SQL_CALC_FOUND_ROWS对于 UNION的意向是它应该不需要全程LIMIT而返回应返回的行数。SQL_CALC_FOUND_ROWS 和UNION 一同使用的条件是:

- SQL_CALC_FOUND_ROWS 关键词必须出现在UNION的第一个 SELECT中。
- FOUND_ROWS()的值只有在使用 UNION ALL时才是精确的。若使用不带ALL的UNION，则会发生两次删除， 而 FOUND_ROWS() 的指只需近似的。
- 假若UNION 中没有出现 LIMIT ，则SQL_CALC_FOUND_ROWS 被忽略，返回临时表中的创建的用来处理UNION的行数。

- LAST_INSERT_ID() LAST_INSERT_ID(*expr*)

自动返回最后一个INSERT或 UPDATE 问询为 AUTO_INCREMENT列设置的第一个 发生的值。

mysql> **SELECT LAST_INSERT_ID();**

-> 195

产生的ID 每次连接后保存在服务器中。这意味着函数向一个给定客户端返回的值是该客户端产生对影响AUTO_INCREMENT列的最新语句第一个 AUTO_INCREMENT值的。这个值不能被其它客户端影响，即使它们产生它们自己的 AUTO_INCREMENT值。这个行为保证了你能够找回自己的 ID 而不用担心其它客户端的活动，而且不需要加锁或处理。

假如你使用一个非“magic”值来更新某一行的AUTO_INCREMENT 列，则LAST_INSERT_ID() 的值不会变化(换言之, 一个不是 NULL也不是 0的值)。

重点: 假如你使用单INSERT语句插入多个行， LAST_INSERT_ID() 只返回插入的第一行产生的值。其原因是这使依靠其它服务器复制同样的 INSERT语句变得简单。

例如:

```
mysql> USE test;

Database changed

mysql> CREATE TABLE t (

    ->   id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,

    ->   name VARCHAR(10) NOT NULL

    -> );

Query OK, 0 rows affected (0.09 sec)


mysql> INSERT INTO t VALUES (NULL, 'Bob');

Query OK, 1 row affected (0.01 sec)


mysql> SELECT * FROM t;

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

| id | name |

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

|  1 | Bob  |

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

```
1 row in set (0.01 sec)

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                1 |
+------------------+
1 row in set (0.00 sec)

mysql> INSERT INTO t VALUES
    -> (NULL, 'Mary'), (NULL, 'Jane'), (NULL, 'Lisa');
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM t;
+----+------+
| id | name |
+----+------+
|  1 | Bob  |
|  2 | Mary |
|  3 | Jane |
|  4 | Lisa |
+----+------+
4 rows in set (0.01 sec)

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                2 |
+------------------+
1 row in set (0.00 sec)
```

虽然第二个问询将3 个新行插入 t, 对这些行的第一行产生的 ID 为 2, 这也是 LAST_INSERT_ID()返回的值。

假如你使用 INSERT IGNORE而记录被忽略，则AUTO_INCREMENT 计数器不会增量，而 LAST_INSERT_ID()返回0, 这反映出没有插入任何记录。

若给出作为到LAST_INSERT_ID()的参数*expr* ，则参数的值被函数返回，并作为被LAST_INSERT_ID()返回的下一个值而被记忆。这可用于模拟序列:

- 创建一个表，用来控制顺序计数器并使其初始化：

o mysql> **CREATE TABLE sequence (id INT NOT NULL);**

o mysql> **INSERT INTO sequence VALUES (0);**

- 使用该表产生这样的序列数 ：

o mysql> **UPDATE sequence SET id=LAST_INSERT_ID(id+1);**

o mysql> **SELECT LAST_INSERT_ID();**

UPDATE 语句会增加顺序计数器并引发向LAST_INSERT_ID() 的下一次调用，用来返回升级后的值。 SELECT 语句会检索这个值。 mysql_insert_id() C API函数也可用于获取这个值。
见25.2.3.36节，“mysql_insert_id()”.

你可以不用调用LAST_INSERT_ID()而产生序列，但这样使用这个函数的效用在于 ID值被保存在服务器中，作为自动产生的值。它适用于多个用户，原因是多个用户均可使用 UPDATE语句并用SELECT语句(或mysql_insert_id()),得到他们自己的序列值，而不会影响其它产生他们自己的序列值的客户端或被其它产生他们自己的序列值的客户端所影响。

注意， mysql_insert_id() 仅会在INSERT 和UPDATE语句后面被升级， 因此你不能在执行了其它诸如SELECT或 SET 这样的SQL语句后使用 C API 函数来找回 LAST_INSERT_ID(*expr*) 对应的值。

- ROW_COUNT()

ROW_COUNT()返回被前面语句升级的、插入的或删除的行数。 这个行数和 **mysql** 客户端显示的行数及mysql_affected_rows() C API 函数返回的值相同。

mysql> **INSERT INTO t VALUES(1),(2),(3);**

问询完成, 表中有3 行 (0.00秒)

记录: 3 重复: 0 警告: 0

mysql> **SELECT ROW_COUNT();**

```
+-------------+
| ROW_COUNT() |
+-------------+
|           3 |
+-------------+
```

表中有1行 (0.00 秒)

mysql> **DELETE FROM t WHERE i IN(1,2);**

问询完成, 找到2 行 (0.00 秒)

mysql> **SELECT ROW_COUNT();**

```
+-------------+
| ROW_COUNT() |
+-------------+
|           2 |
+-------------+
```

表中有1行 (0.00 秒)

- SCHEMA()

这个函数和 DATABASE()具有相同的意义。

- SESSION_USER()

SESSION_USER()和 USER()具有相同的意义。

- SYSTEM_USER()

SYSTEM_USER()合 USER()具有相同的意义。

- USER()

返回当前 MySQL用户名和机主名/

mysql> **SELECT USER();**

-> 'davida@localhost'

这个值指示了你指定的连接服务器时的用户名,及你所连接的客户主机。这个值可以和CURRENT_USER() 的值不同。

你可以这样提取用户名部分:

mysql> **SELECT SUBSTRING_INDEX(USER(),'@',1);**

-> 'davida'

由于 USER() 返回一个utf8 字符集中的值，你也应确保'@' 字符串文字在该字符集中得到解释:

mysql> **SELECT SUBSTRING_INDEX(USER(),_utf8'@',1);**

-> 'davida'

- VERSION()

返回指示 MySQL 服务器版本的字符串。这个字符串使用 utf8 字符集。

mysql> **SELECT VERSION();**

-> '5.1.2-alpha-standard'

注意，假如你的版本字符串以-log结尾，这说明登录已被激活。

12.9.4. 其他函数

- DEFAULT(*col_name*)

返回一个表列的默认值。若该列没有默认值则会产生错误。

mysql> **UPDATE t SET i = DEFAULT(i)+1 WHERE id < 100;**

- FORMAT(*X*,*D*)

将数字*X* 的格式写为'#,###,###.##',以四舍五入的方式保留小数点后 *D* 位， 并将结果以字符串的形式返回。若 *D* 为 0, 则返回结果不带有小数点，或不含小数部分。

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'

- GET_LOCK(*str*,*timeout*)

设法使用字符串*str* 给定的名字得到一个锁， 超时为*timeout* 秒。若成功得到锁，则返回 1，若操作超时则返回0 (例如,由于另一个客户端已提前封锁了这个名字),若发生错误则返回NULL (诸如缺乏记忆或线程**mysqladmin kill** 被断开)。假如你有一个用GET_LOCK()得到的锁，当你执行RELEASE_LOCK()或你的连接断开(正常或非正常)时，这个锁就会解除。

这个函数可用于执行应用程序锁或模拟记录锁定。名称被锁定在服务器范围内。假如一个名字已经被一个客户端封锁， GET_LOCK() 会封锁来自另一个客户端申请封锁同一个名字的任何请求。这使对一个封锁名达成协议的客户端使用这个名字合作执行建议锁。然而要知道它也允许不在一组合作客户端中的一个客户端封锁名字，不论是服役的还是非故意的，这样阻止任何合作中的客户端封锁这个名字。一个减少这种情况发生的办法就是使用数据库特定的或应用程序特定的封锁名。例如， 使用*db_name.str*或 *app_name.str* 形式的封锁名。

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，第二个 RELEASE_LOCK()调用返回 NULL ，原因是锁'lock1' 杯第二个GET_LOCK()调用解开。

- INET_ATON(*expr*)

给出一个作为字符串的网络地址的点地址表示，返回一个代表该地址数值的整数。地址可以是4或8比特地址。

mysql> **SELECT INET_ATON('209.207.224.40');**

-> 3520061480

产生的数字总是按照网络字节顺序。如上面的例子，数字按照 $209\times256^3 + 207\times256^2 + 224\times256 + 40$ 进行计算。

INET_ATON() 也能理解短格式 IP 地址：

mysql> **SELECT INET_ATON('127.0.0.1'), INET_ATON('127.1');**

-> 2130706433, 2130706433

注释： 在存储由INET_ATON() 产生的值时，推荐你使用 INT UNSIGNED 列。假如你使用 (带符号) INT列, 则相应的第一个八位组大于127的IP 地址值会被截至 2147483647 (即, INET_ATON('127.255.255.255') 所返回的值)。请参见11.2节，“数值类型”。

- INET_NTOA(*expr*)

给定一个数字网络地址 (4 或 8 比特),返回作为字符串的该地址的电地址表示。

mysql> **SELECT INET_NTOA(3520061480);**

-> '209.207.224.40'

- IS_FREE_LOCK(*str*)

检查名为*str*的锁是否可以使用 (换言之,没有被封锁)。若锁可以使用，则返回 1 (没有人在用这个锁), 若这个锁正在被使用，则返回0 ，出现错误则返回 NULL (诸如不正确的参数)。

- IS_USED_LOCK(*str*)

检查名为*str*的锁是否正在被使用(换言之,被封锁)。若被封锁，则返回使用该锁的客户端的连接标识符。否则返回 NULL。

- MASTER_POS_WAIT(*log_name*,*log_pos*[,*timeout*])

该函数对于控制主从同步很有用处。它会持续封锁，直到从设备阅读和应用主机记录中所有补充资料到指定的位置。返回值是其为到达指定位置而必须等待的记录事件的数目。若从设备SQL线程没有被启动、从设备主机信息尚未初始化、参数不正确或出现任何错误，则该函数返回 NULL。若超时时间被超过，则返回-1。若在MASTER_POS_WAIT() 等待期间，从设备SQL线程中止，则该函数返回 NULL。若从设备由指定位置通过，则函数会立即返回结果。

假如已经指定了一个超时时间值，当 超时时间 秒数经过后MASTER_POS_WAIT()会停止等待。超时时间 必须大于 0；一个为零或为负值的 超时时间 表示没有超市时间。

- NAME_CONST(*name*,*value*)

返回给定值。 当用来产生一个结果集合列时, NAME_CONST()促使该列使用给定名称。

mysql> **SELECT NAME_CONST('myname', 14);**

```
+--------+
| myname |
+--------+
|     14 |
+--------+
```

这个函数被添加进 MySQL 5.0.12。它只做内部使用。 服务器在书写来自包含局部程序变量的存储程序的语句时会用到它，详见20.4节，“存储子程序和触发程序的二进制日志功能”。你可能会在**mysqlbinlog** 的书橱中看到这个函数。

- RELEASE_LOCK(*str*)

解开被GET_LOCK()获取的，用字符串*str* 所命名的锁。若锁被解开，则返回 1，若改线程尚未创建锁，则返回0 (此时锁没有被解开), 若命名的锁不存在，则返回 NULL。若该锁从未被对GET_LOCK()的调用获取，或锁已经被提前解开，则该锁不存在。

DO 语句和RELEASE_LOCK()同时使用很方便。请参见13.2.2节，“DO语法”。

- SLEEP(*duration*)

睡眠(暂停) 时间为*duration* 参数给定的秒数，然后返回 0。若 SLEEP() 被中断,它会返回 1。 duration 或许或包括一个给定的以微秒为单位的分数部分。

- UUID()

返回一个通用唯一标识符(UUID) ，其产生的根据是《DCE 1.1: 远程过程调用》 (附录A) CAE (公共应用软件环境) 的说明，该作品于1997年10月由 The Open Group 出版 (文件编号 C706, http://www.opengroup.org/public/pubs/catalog/c706.htm).

UUID被设计成一个在时间和空间上都独一无二的数字。2个对UUID() 的调用应产生2个不同的值，即使这些调用的执行是在两个互不相连的单独电脑上进行。

UUID 是一个由5位十六进制数的字符串表示的128比特数字 ，其格式为 aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee ：

  - 前3个数字从一个时间戳产生。
  - 第4 个数字保持暂时唯一性，以防时间戳值失去单一性 (例如, 由于经济时)。
  - 第5个数字是一个 IEEE 802 节点号，它提供空间唯一性。若后者不可用，则用一个随机数字替换。(例如, 由于主机没有以太网卡，或我们不知道怎样在你的操作系统上找到界面的机器地址)。假若这样，空间唯一性就不能得到保证。尽管如此，一个冲突的发生机率还是非常低的。

目前,一个界面的 MAC 地址尽被FreeBSD 和 Linux考虑到。在其它操作系统中, MySQL使用随机产生的 48比特数字。

mysql> **SELECT UUID();**

```
-> '6ccd780c-baba-1026-9564-0040f4311e29'
```

注意， UUID() 不支持复制功能。

- VALUES(*col_name*)

在一个INSERT … ON DUPLICATE KEY UPDATE …语句中，你可以在UPDATE 子句中使用VALUES(*col_name*)函数，用来访问来自该语句的INSERT 部分的列值。换言之，UPDATE 子句中的VALUES(*col_name*) 访问需要被插入的*col_name* 的值,并不会发生重复键冲突。这个函数在多行插入中特别有用。 VALUES()函数只在INSERT ... UPDATE 语句中有意义，而在其它情况下只会返回 NULL。请参见13.2.4节，“INSERT语法”.

mysql> **INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)**

-> **ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);**

12.10. 与GROUP BY子句同时使用的函数和修改程序

12.10.1. GROUP BY（聚合）函数
12.10.2. GROUP BY修改程序
12.10.3. 具有隐含字段的GROUP BY

12.10.1. GROUP BY（聚合）函数

本章论述了用于一组数值操作的 group (集合)函数。除非另作说明， group 函数会忽略 NULL 值。

假如你在一个不包含 ROUP BY子句的语句中使用一个 group函数 ，它相当于对所有行进行分组。

- AVG([DISTINCT] *expr*)

返回*expr* 的平均值。 DISTINCT 选项可用于返回 *expr*的不同值的平均值。

若找不到匹配的行，则AVG()返回 NULL 。

mysql> **SELECT student_name, AVG(test_score)**

-> **FROM student**

-> **GROUP BY student_name;**

- BIT_AND(*expr*)

返回*expr*中所有比特的 bitwise AND 。计算执行的精确度为64比特(BIGINT) 。

若找不到匹配的行，则这个函数返回 18446744073709551615 。(这是无符号 BIGINT 值，所有比特被设置为1）。

- BIT_OR(*expr*)

返回*expr* 中所有比特的bitwise OR。计算执行的精确度为64比特(BIGINT) 。

若找不到匹配的行，则函数返回 0 。

- BIT_XOR(*expr*)

返回*expr* 中所有比特的bitwise XOR。计算执行的精确度为64比特(BIGINT) 。

若找不到匹配的行，则函数返回 0 。

- COUNT(*expr*)

返回SELECT语句检索到的行中非NULL值的数目。

若找不到匹配的行，则COUNT() 返回 0 。

mysql> **SELECT student.student_name,COUNT(*)**

-> **FROM student,course**

-> **WHERE student.student_id=course.student_id**

-> **GROUP BY student_name;**

COUNT(*) 的稍微不同之处在于，它返回检索行的数目， 不论其是否包含 NULL值。

SELECT 从一个表中检索，而不检索其它的列，并且没有 WHERE子句时， COUNT(*)被优化到最快的返回速度。例如:

mysql> **SELECT COUNT(*) FROM student;**

这个优化仅适用于 MyISAM表, 原因是这些表类型会储存一个函数返回记录的精确数量，而且非常容易访问。对于事务型的存储引擎(InnoDB, BDB), 存储一个精确行数的问题比较多，原因是可能会发生多重事物处理, 而每个都可能会对行数产生影响。

- COUNT(DISTINCT *expr*,[*expr*...])

返回不同的非NULL值数目。

若找不到匹配的项，则COUNT(DISTINCT)返回 0 。

mysql> **SELECT COUNT(DISTINCT results) FROM student;**

在MySQL中, 你通过给定一个表达式列表而获取不包含NULL 不同表达式组合的数目。在标准 SQL中,你将必须在COUNT(DISTINCT ...)中连接所有表达式。

- GROUP_CONCAT(*expr*)

该函数返回带有来自一个组的连接的非NULL值的字符串结果。其完整的语法如下所示:

GROUP_CONCAT([DISTINCT] *expr* [,*expr* ...]

[ORDER BY {*unsigned_integer* | *col_name* | *expr*}

[ASC | DESC] [,*col_name* ...]]

[SEPARATOR *str_val*])

mysql> **SELECT student_name,**

-> **GROUP_CONCAT(test_score)**

-> **FROM student**

```
    -> GROUP BY student_name;
```

Or:

```
mysql> SELECT student_name,
    -> GROUP_CONCAT(DISTINCT test_score
    ->          ORDER BY test_score DESC SEPARATOR ' ')
    -> FROM student
    -> GROUP BY student_name;
```

在MySQL中,你可以获取表达式组合的连接值。你可以使用DISTINCT删去重复值。假若你希望多结果值进行排序，则应该使用 ORDER BY子句。若要按相反顺序排列，将 DESC (递减) 关键词添加到你要用ORDER BY 子句进行排序的列名称中。默认顺序为升序；可使用ASC将其明确指定。 SEPARATOR 后面跟随应该被插入结果的值中间的字符串值。默认为逗号 (‘,’)。通过指定SEPARATOR " ，你可以删除所有分隔符。

使用group_concat_max_len系统变量，你可以设置允许的最大长度。 程序中进行这项操作的语法如下，其中val 是一个无符号整数：

```
SET [SESSION | GLOBAL] group_concat_max_len = val;
```

若已经设置了最大长度， 则结果被截至这个最大长度。

- MIN([DISTINCT] *expr*), MAX([DISTINCT] *expr*)

返回*expr* 的最小值和最大值。 MIN() 和 MAX() 的取值可以是一个字符串参数；在这些情况下， 它们返回最小或最大字符串值。请参见7.4.5节，“MySQL如何使用索引”。 DISTINCT关键词可以被用来查找*expr* 的不同值的最小或最大值，然而，这产生的结果与省略DISTINCT 的结果相同。

若找不到匹配的行，MIN()和MAX()返回 NULL 。

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
    -> FROM student
    -> GROUP BY student_name;
```

对于MIN()、 MAX()和其它集合函数， MySQL当前按照它们的字符串值而非字符串在集合中的相关位置比较ENUM和 SET 列。这同ORDER BY比较二者的方式有所不同。这一点应该在MySQL的未来版本中得到改善。

- STD(*expr*) STDDEV(*expr*)

返回*expr* 的总体标准偏差。这是标准 SQL 的延伸。这个函数的STDDEV() 形式用来提供和Oracle 的兼容性。可使用标准SQL函数 STDDEV_POP() 进行代替。

若找不到匹配的行，则这些函数返回 NULL 。

- STDDEV_POP(*expr*)

返回*expr* 的总体标准偏差(VAR_POP()的平方根)。你也可以使用 STD() 或STDDEV(), 它们具有相同的意义，然而不是标准的 SQL。

若找不到匹配的行，则STDDEV_POP()返回 NULL。

- STDDEV_SAMP(*expr*)

返回*expr* 的样本标准差 (VAR_SAMP()的平方根)。

若找不到匹配的行，则STDDEV_SAMP() 返回 NULL 。

- SUM([DISTINCT] *expr*)

返回*expr* 的总数。 若返回集合中无任何行，则 SUM() 返回NULL。DISTINCT 关键词可用于 MySQL 5.1 中，求得*expr* 不同值的总和。

若找不到匹配的行，则SUM()返回 NULL。

- VAR_POP(*expr*)

返回*expr* 总体标准方差。它将行视为总体，而不是一个样本， 所以它将行数作为分母。你也可以使用VARIANCE(),它具有相同的意义然而不是 标准的 SQL。

若找不到匹配的项，则VAR_POP()返回NULL。

- VAR_SAMP(*expr*)

返回*expr* 的样本方差。更确切的说，分母的数字是行数减去1。

若找不到匹配的行，则VAR_SAMP()返回NULL。

- VARIANCE(*expr*)

返回*expr* 的总体标准方差。这是标准SQL 的延伸。可使用标准SQL 函数 VAR_POP() 进行代替。

若找不到匹配的项，则VARIANCE()返回NULL。

12.10.2. GROUP BY修改程序

GROUP BY子句允许一个将额外行添加到简略输出端 WITH ROLLUP 修饰符。这些行代表高层(或高聚集)简略操作。ROLLUP 因而允许你在多层分析的角度回答有关问询的问题。例如，它可以用来向OLAP (联机分析处理) 操作提供支持。

设想一个名为sales 的表具有年份、国家、产品及记录销售利润的利润列：

```
CREATE TABLE sales
(
    year    INT NOT NULL,
    country VARCHAR(20) NOT NULL,
    product VARCHAR(32) NOT NULL,
    profit  INT
);
```

可以使用这样的简单GROUP BY，每年对表的内容做一次总结：

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year;
+------+-------------+
| year | SUM(profit) |
+------+-------------+
| 2000 |        4525 |
| 2001 |        3010 |
+------+-------------+
```

这个输出结果显示了每年的总利润， 但如果你也想确定所有年份的总利润，你必须自己累加每年的单个值或运行一个加法询问。

或者你可以使用 ROLLUP, 它能用一个问询提供双层分析。将一个 WITH ROLLUP修饰符添加到GROUP BY 语句，使询问产生另一行结果，该行显示了所有年份的总价值:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year WITH ROLLUP;
+------+-------------+
| year | SUM(profit) |
+------+-------------+
| 2000 |        4525 |
| 2001 |        3010 |
| NULL |        7535 |
+------+-------------+
```

总计高聚集行被年份列中的NULL值标出。

当有多重 GROUP BY 列时，ROLLUP产生的效果更加复杂。这时，每次在除了最后一个分类列之外的任何列出现一个 “break” (值的改变) ，则问讯会产生一个高聚集累计行。

例如，在没有 ROLLUP的情况下，一个以年、国家和产品为基础的关于 sales 表的一览表可能如下所示:

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

表示总值的输出结果仅位于年/国家/产品的分析级别。当添加了 ROLLUP后， 问询会产生一些额外的行:

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

对于这个问询， 添加ROLLUP 子句使村输出结果包含了四层分析的简略信息，而不只是一个下面是怎样解释ROLLUP输出:

- 一组给定的年份和国家的每组产品行后面, 会产生一个额外的总计行， 显示所有产品的总值。这些行将产品列设置为 NULL。
- 一组给定年份的行后面，会产生一个额外的总计行，显示所有国家和产品的总值。这些行将国家和产品列设置为 NULL。
- 最后, 在所有其它行后面，会产生一个额外的总计列，显示所有年份、国家及产品的总值。 这一行将年份、国家和产品列设置为 NULL。

使用**ROLLUP** 时的其它注意事项

以下各项列出了一些MySQL执行ROLLUP的特殊状态:

当你使用 ROLLUP时, 你不能同时使用 ORDER BY子句进行结果排序。换言之， ROLLUP 和ORDER BY 是互相排斥的。然而，你仍可以对排序进行一些控制。在 MySQL中， GROUP BY 可以对结果进行排序，而且你可以在GROUP BY列表指定的列中使用明确的 ASC和DESC关键词，从而对个别列进行排序。 (不论如何排序被ROLLUP添加的较高级别的总计行仍出现在它们被计算出的行后面）。

LIMIT可用来限制返回客户端的行数。LIMIT 用在 ROLLUP后面, 因此这个限制 会取消被ROLLUP添加的行。例如:

```
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 |
```

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

将ROLLUP同 LIMIT一起使用可能会产生更加难以解释的结果，原因是对于理解高聚集行，你所掌握的上下文较少。

在每个高聚集行中的NULL 指示符会在该行被送至客户端时产生。服务器会查看最左边的改变值后面的GROUP BY子句指定的列。对于任何结果集合中的，有一个词匹配这些名字的列, 其值被设为 NULL。(若你使用列数字指定了分组列，则服务器会通过数字确定将哪个列设置为 NULL）。

由于在高聚集行中的 NULL值在问询处理阶段被放入结果集合中，你无法将它们在问询本身中作为NULL值检验。例如，你无法将 HAVING product IS NULL 添加到问询中，从而在输出结果中删去除了高聚集行以外的部分。

另一方面, NULL值在客户端不以 NULL 的形式出现， 因而可以使用任何MySQL客户端编程接口进行检验。

12.10.3. 具有隐含字段的GROUP BY

MySQL 扩展了 GROUP BY的用途，因此你可以使用SELECT 列表中不出现在GROUP BY语句中的列或运算。这代表“对该组的任何可能值 ”。你可以通过避免排序和对不必要项分组的办法得到它更好的性能。例如，在下列问询中，你无须对customer.name 进行分组:

mysql> **SELECT order.custid, customer.name, MAX(payments)**

-> **FROM order,customer**

-> **WHERE order.custid = customer.custid**

-> **GROUP BY order.custid;**

在标准SQL中, 你必须将 customer.name添加到 GROUP BY子句中。在MySQL中, 假如你不在ANSI模式中运行，则这个名字就是多余的。

假如你从 GROUP BY 部分省略的列在该组中不是唯一的，那么不要使用这个功能! 你会得到非预测性结果。

在有些情况下,你可以使用MIN()和MAX() 获取一个特殊的列值，即使他不是唯一的。下面给出了来自包含排序列中最小值的列中的值:

SUBSTR(MIN(CONCAT(RPAD(sort,6,' '),column)),7)

See 3.6.4节，“拥有某个字段的组间最大值的行”.

注意，假如你正在尝试遵循标准 SQL, 你不能使用GROUP BY或 ORDER BY子句中的表达式。你可以通过使用表达式的别名绕过这一限制:

mysql> **SELECT id,FLOOR(value/100) AS val**

-> **FROM** *tbl_name*

-> **GROUP BY id, val ORDER BY val;**

然而, MySQL允许你使用GROUP BY 及 ORDER BY 子句中的表达式。例如:

mysql> **SELECT id, FLOOR(value/100) FROM** *tbl_name* **ORDER BY RAND();**

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第13章：SQL语句语法

目录

13.1. 数据定义语句

- 13.1.1. ALTER DATABASE语法
- 13.1.2. ALTER TABLE语法
- 13.1.3. CREATE DATABASE语法
- 13.1.4. CREATE INDEX语法
- 13.1.5. CREATE TABLE语法
- 13.1.6. DROP DATABASE语法
- 13.1.7. DROP INDEX语法
- 13.1.8. DROP TABLE语法
- 13.1.9. RENAME TABLE语法

13.2. 数据操作语句

- 13.2.1. DELETE语法
- 13.2.2. DO语法
- 13.2.3. HANDLER语法
- 13.2.4. INSERT语法
- 13.2.5. LOAD DATA INFILE语法
- 13.2.6. REPLACE语法
- 13.2.7. SELECT语法
- 13.2.8. Subquery语法
- 13.2.9. TRUNCATE语法
- 13.2.10. UPDATE语法

13.3. MySQL实用工具语句

- 13.3.1. DESCRIBE语法（获取有关列的信息）
- 13.3.2. USE语法

13.4. MySQL事务处理和锁定语句

- 13.4.1. START TRANSACTION, COMMIT和ROLLBACK语法
- 13.4.2. 不能回滚的语句
- 13.4.3. 会造成隐式提交的语句
- 13.4.4. SAVEPOINT和ROLLBACK TO SAVEPOINT语法
- 13.4.5. LOCK TABLES和UNLOCK TABLES语法
- 13.4.6. SET TRANSACTION语法
- 13.4.7. XA事务

13.5. 数据库管理语句

- 13.5.1. 账户管理语句
- 13.5.2. 表维护语句
- 13.5.3. SET语法
- 13.5.4. SHOW语法
- 13.5.5. 其它管理语句

13.6. 复制语句

13.6.1. 用于控制主服务器的SQL语句
13.6.2. 用于控制从服务器的SQL语句

13.7. 用于预处理语句的SQL语法

本章介绍了SQL语句的语法。

13.1. 数据定义语句

13.1.1. ALTER DATABASE语法
13.1.2. ALTER TABLE语法
13.1.3. CREATE DATABASE语法
13.1.4. CREATE INDEX语法
13.1.5. CREATE TABLE语法
13.1.6. DROP DATABASE语法
13.1.7. DROP INDEX语法
13.1.8. DROP TABLE语法
13.1.9. RENAME TABLE语法

13.1.1. ALTER DATABASE语法

```
ALTER {DATABASE | SCHEMA} [db_name]
    alter_specification [, alter_specification] ...

alter_specification:
    [DEFAULT] CHARACTER SET charset_name
  | [DEFAULT] COLLATE collation_name
```

ALTER DATABASE用于更改数据库的全局特性。这些特性储存在数据库目录中的db.opt文件中。要使用ALTER DATABASE，您需要获得数据库ALTER权限。

CHARACTER SET子句用于更改默认的数据库字符集。COLLATE子句用于更改默认的数据库整序。在第10章：字符集支持中对字符集和整序名称进行了讨论。

数据库名称可以忽略，此时，语句对应于默认数据库。也可以使用ALTER SCHEMA。

13.1.2. ALTER TABLE语法

```
ALTER [IGNORE] TABLE tbl_name
    alter_specification [, alter_specification] ...

alter_specification:
    ADD [COLUMN] column_definition [FIRST | AFTER col_name ]
  | ADD [COLUMN] (column_definition,...)
  | ADD INDEX [index_name] [index_type] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        PRIMARY KEY [index_type] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        UNIQUE [index_name] [index_type] (index_col_name,...)
  | ADD [FULLTEXT|SPATIAL] [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 column_definition
        [FIRST|AFTER col_name]
  | MODIFY [COLUMN] column_definition [FIRST | AFTER col_name]
  | DROP [COLUMN] col_name
  | DROP PRIMARY KEY
  | DROP INDEX index_name
  | DROP FOREIGN KEY fk_symbol
```

```
| DISABLE KEYS
| ENABLE KEYS
| RENAME [TO] new_tbl_name
| ORDER BY col_name
| CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
| [DEFAULT] CHARACTER SET charset_name [COLLATE collation_name]
| DISCARD TABLESPACE
| IMPORT TABLESPACE
| table_options
| partition_options
| ADD PARTITION partition_definition
| DROP PARTITION partition_names
| COALESCE PARTITION number
| REORGANIZE PARTITION partition_names INTO (partition_definitions)
| ANALYZE PARTITION partition_names
| CHECK PARTITION partition_names
| OPTIMIZE PARTITION partition_names
| REBUILD PARTITION partition_names
| REPAIR PARTITION partition_names
```

ALTER TABLE用于更改原有表的结构。例如，您可以增加或删减列，创建或取消索引，更改原有列的类型，或重新命名列或表。您还可以更改表的评注和表的类型。

允许进行的变更中，许多子句的语法与CREATE TABLE中的子句的语法相近。其中包括*table_options*修改，选项有ENGINE, AUTO_INCREMENT和AVG_ROW_LENGTH等。请见13.1.5节，“CREATE TABLE语法”。

存储引擎不支持有些操作，如果进行这些操作，会出现警告。使用SHOW WARNINGS可以显示出这些警告。请参见13.5.4.22节，“SHOW WARNINGS语法”。

如果您使用ALTER TABLE更改列规约，但是DESCRIBE *tbl_name*提示您列规约并没有改变，则可能是因为MySQL忽略了您所做的更改。忽略更改的原因见13.1.5.1节，“沉寂的列规格变更”。例如，如果您试图把VARCHAR列更改为CHAR列，此时，如果表包含其它长度可变的列，则MySQL仍会使用VARCHAR。

ALTER TABLE运行时会对原表进行临时复制，在副本上进行更改，然后删除原表，再对新表进行重命名。在执行ALTER TABLE时，其它用户可以阅读原表，但是对表的更新和修改的操作将被延迟，直到新表生成为止。新表生成后，这些更新和修改信息会自动转移到新表上。

注意，如果您在执行ALTER TABLE时使用除了RENAME以外的选项，则MySQL会创建一个临时表。即使数据并不需要进行复制（例如当您更改列的名称时），MySQL也会这么操作。对于MyISAM表，您可以通过把myisam_sort_buffer_size系统变量设置到一个较高的值，来加快重新创建索引（该操作是变更过程中速度最慢的一部分）的速度。

- 要使用ALTER TABLE，您需要获得表的ALTER, INSERT和CREATE权限。

- IGNORE是MySQL相对于标准SQL的扩展。如果在新表中有重复关键字，或者当STRICT模式启动后出现警告，则使用IGNORE控制ALTER TABLE的运行。如果没有指定IGNORE，当重复关键字错误发生时，复制操作被放弃，返回前一步骤。如果指定了IGNORE，则对于有重复关键字的行，只使用第一行，其它有冲突的行被删除。并且，对错误值进行修正，使之尽量接近正确值。

- 您可以在一个ALTER TABLE语句里写入多个ADD, ALTER, DROP和CHANGE子句，中间用逗号分开。这是MySQL相对于标准SQL的扩展。在标准SQL中，每个ALTER TABLE语句中每个子句只允许使用一次。例如，在一个语句中取消多个列：

- ```
  mysql> ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
  ```

- CHANGE *col_name*, DROP *col_name*和DROP INDEX是MySQL相对于标准SQL的扩展。

- MODIFY是Oracle对ALTER TABLE的扩展。

- COLUMN只是自选项目，可以忽略。

- 如果您使用ALTER TABLE *tbl_name* RENAME TO *new_tbl_name*并且没有其它选项，则MySQL只对
  ```

与table *tbl_name*相对应的文件进行重命名。不需要创建一个临时表。（您也可以使用RENAME TABLE语句对表进行重命名。请参见13.1.9节，“RENAME TABLE语法”。）

· *column_definition*子句使用与CREATE TABLE中的ADD和CHANGE子句相同的语法。注意，此语法包括列名称，而不只是列类型。请参见13.1.5节，“CREATE TABLE语法”。

· 您可以使用CHANGE *old_col_name column_definition*子句对列进行重命名。重命名时，需给定旧的和新的列名称和列当前的类型。例如：要把一个INTEGER列的名称从a变更到b，您需要如下操作：

```
mysql> ALTER TABLE t1 CHANGE a b INTEGER;
```

如果您想要更改列的类型而不是名称， CHANGE语法仍然要求旧的和新的列名称，即使旧的和新的列名称是一样的。例如：

```
mysql> ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
```

您也可以使用MODIFY来改变列的类型，此时不需要重命名：

```
mysql> ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
```

· 如果您使用CHANGE或MODITY缩短列长时，列中存在有索引，并且缩短后的列长小于索引长度，则MySQL会自动缩短索引的长度。

· 当您使用CHANGE或MODIFY更改列的类型时，MySQL会尽量把原有的列值转化为新的类型。

· 您可以使用FIRST或AFTER *col_name*在一个表行中的某个特定位置添加列。默认把列添加到最后。您也可以在CHANGE或MODIFY语句中使用FIRST和AFTER。

· AFTER COLUMN用于指定列的新默认值，或删除旧的默认值。如果旧的默认值被删除同时列值为NULL，则新的默认值为NULL。如果列值不能为NULL，MySQL会指定一个默认值，请参见13.1.5节，“CREATE TABLE语法”。

· DROP INDEX用于取消索引。这是MySQL相对于标准SQL的扩展。请参见13.1.7节，“DROP INDEX语法”。

· 如果列从表中被取消了，则这些列也从相应的索引中被取消。如果组成一个索引的所有列均被取消，则该索引也被取消。

· 如果一个表只包含一列，则此列不能被取消。如果您想要取消表，应使用DROP TABLE。

· DROP PRIMAY DEY用于取消主索引。注释：在MySQL较早的版本中，如果没有主索引，则DROP PRIMARY KEY会取消表中的第一个UNIQUE索引。在MySQL 5.1中不会出现这种情况。如果在MySQL 5.1中对没有主键的表使用DROP PRIMARY KEY，则会出现错误信息。

如果您向表中添加UNIQUE KEY或PRIMARY KEY，则UNIQUE KEY或PRIMARY KEY会被储存在非唯一索引之前，这样MySQL就可以尽早地检查出重复关键字。

· ORDER BY用于在创建新表时，让各行按一定的顺序排列。注意，在插入和删除后，表不会仍保持此顺序。当您知道多数情况下您会按照特定的顺序查询各行时，可以使用这个选项；在对表进行了大的改动后，通过使用此选项，您可以提高查询效率。在有些情况下，如果表按列排序，对于MySQL来说，排序可能会更简单。

· 如果您对一个MyISAM表使用ALTER TABLE，则所有非唯一索引会被创建到一个单独的批里（和REPAIR TABLE相同）。当您有许多索引时，这样做可以使ALTER TABLE的速度更快。

这项功能可以明确激活。ALTER TABLE...DISABLE KEYS让MySQL停止更新MyISAM表中的非唯一索引。然后使 ALTER TABLE ... ENABLE KEYS MySQL

用 重新创建丢失的索引。进行此操作时， 采用一种特殊的算法，比一个接一个地插入关键字要快很多。因此，在进行成批插入操作前先使关键字禁用可以大大地加快速度。使用ALTER TABLE ... DISABLE KEYS除了需要获得以前提到的权限以外，还需要获得INDEX权限。

- Innodb存储引擎支持FOREIGN KEY和REFERENCES子句。Innodb存储引擎执行ADD [CONSTRAINT [*symbol*]] FOREIGN KEY (...) REFERENCES ... (...)。请参见15.2.6.4节，“FOREIGN KEY约束”。对于其它存储引擎，这些子句会被分析，但是会被忽略。对于所有的存储引擎，CHECK子句会被分析，但是会被忽略。请参见13.1.5节，“CREATE TABLE语法”。接受这些子句但又忽略子句的原因是为了提高兼容性，以便更容易地从其它SQL服务器中导入代码，并运行应用程序，创建带参考数据的表。请参见1.8.5节，“MySQL与标准SQL的差别”。

- InnoDB支持使用ALTER TABLE来取消外键：

```
ALTER TABLE yourtablename DROP FOREIGN KEY fk_symbol;
```

要了解更多信息，请参见15.2.6.4节，“FOREIGN KEY约束”。

- ALTER TABLE忽略DATA DIRECTORY和INDEX DIRECTORY表选项。

- 如果您想要把表默认的字符集和所有字符列（CHAR, VARCHAR, TEXT）改为新的字符集，应使用如下语句：

```
ALTER TABLE tbl_name CONVERT TO CHARACTER SET charset_name;
```

警告：前面的操作转换了字符集之间的列类型。如果您有一列使用一种字符集（如latin1），但是存储的值实际上使用了其它的字符集（如utf8），这种情况不是您想要的。此时，您必须对这样的列进行以下操作。

```
ALTER TABLE t1 CHANGE c1 c1 BLOB;

ALTER TABLE t1 CHANGE c1 c1 TEXT CHARACTER SET utf8;
```

这种方法能够实现此功能的原因是，当您转换到BLOB列或从BLOB列转换过来时，并没有发生转换。

如果您指定CONVERT TO CHARACTER SET为二进制，则TEXT列被转换到相应的二进制字符串类型（BINARY, VARBINARY, BLOB）。这意味着这些列将不再有字符集，接下来的CONVERT TO操作也将不适用于这些列。

要仅仅改变一个表的默认字符集，应使用此语句：

```
ALTER TABLE tbl_name DEFAULT CHARACTER SET charset_name;
```

词语DEFAULT为自选项。如果您在向表中添加一个新列时（例如，使用ALTER TABLE...ADD column）没有指定字符集，则此时使用的字符集为默认字符集。

警告：ALTER TABLE...DEFAULT CHARACTER SET和ALTER TABLE...CHARACTER SET是等价的，只用于更改默认的表字符集。

- 如果InnoDB表在创建时，使用了.ibd文件中的自己的表空间，则这样的文件可以被删除和导入。使用此语句删除.ibd文件：

```
ALTER TABLE tbl_name DISCARD TABLESPACE;
```

此语句用于删除当前的.ibd文件，所以应首先确认您有一个备份。如果在表空间被删除后尝试打开表格，则会出现错误。

要把备份的.ibd文件还原到表中，需把此文件复制到数据库目录中，然后书写此语句：

```
ALTER TABLE tbl_name IMPORT TABLESPACE;
```

见15.2.6.6节，“使用按表的表空间”。

· 使用mysql_info() C API函数，您可以了解有多少记录已被复制，以及（当使用IGNORE时）有多少记录由于重复关键字的原因已被删除。请参见25.2.3.34节，“mysql_info()”。

· ALTER TABLE也可以用于对带分区的表进行重新分区，功能包括添加、取消、合并和拆分各分区，还可以用于进行分区维护。

对带分区的表使用*partition_options*子句和ALTER TABLE可以对表进行重新分区，使用时依据*partition_options*定义的分区方法。本子句以PARTITION BY为开头，然后使用与用于CREATE TABLE的*partition_options*子句一样的语法和规则（要了解详细信息，请参见13.1.5节，“CREATE TABLE语法”）。注释：MySQL 5.1服务器目前接受此语法，但是不实际执行；等MySQL 5.1开发出来后，将执行此语法。

用于ALTER TABLE ADD PARTITION的*partition_definition*子句支持用于CREATE TABLE语句的*partition_definition*子句的同样名称的选项。（要了解语法和介绍，请参见13.1.5节，“CREATE TABLE语法”。）例如，假设您有一个按照以下方式创建的带分区的表：

```
CREATE TABLE t1 (
    id INT,
    year_col INT
)
PARTITION BY RANGE (year_col) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1995),
    PARTITION p2 VALUES LESS THAN (1999)
);
```

您可以在表中增加一个新的分区p3，该分区用于储存小于2002的值。添加方法如下：

```
ALTER TABLE t1 ADD PARTITION p3 VALUES LESS THAN (2002);
```

注释：您不能使用ALTER TABLE向一个没有进行分区的表添加分区。

DROP PARTITION用于取消一个或多个RANGE或LIST分区。此命令不能用于HASH或KEY 分区；用于这两个分区时，应使用COALESCE PARTITION（见后）。如果被取消的分区其名称列于*partition_names*清单中，则储存在此分区中的数据也被取消。例如，如果以前已定义的表t1，您可以采用如下方法取消名称为p0和p1的分区：

```
ALTER TABLE DROP PARTITION p0, p1;
```

ADD PARTITION和DROP PARTITION目前不支持IF [NOT] EXISTS。也不可能对一个分区或一个已分区的表进行重命名。如果您希望对一个分区进行重命名，您必须取消分区，再重新建立；如果您希望对一个已分区的表进行重新命名，您必须取消所有分区，然后对表进行重命名，再添加被取消的分区。

COALESCE PARTITION可以用于使用HASH或KEY进行分区的表，以便使用*number*来减少分区的数目。例如，假设您使用下列方法创建了表t2：

```
CREATE TABLE t2 (
    name VARCHAR (30),
    started DATE
)
PARTITION BY HASH(YEAR(started))
```

```
PARTITIONS (6);
```

您可以使用以下命令，把t2使用的分区的数目由6个减少到4个：

```
ALTER TABLE t2 COALESCE PARTITION 2;
```

包含在最后一个*number*分区中的数据将被合并到其余的分区中。在此情况下，分区4和分区5将被合并到前4个分区中（编号为0、1、2和3的分区）。

如果要更改部分分区，但不更改所有的分区，您可以使用REORGANIZE PARTITION。这个命令有多种使用方法：

o 把多个分区合并为一个分区。通过把多个分区的名称列入*partition_names*清单，并为*partition_definition*提供一个单一的定义，可以实现这个功能。

o 把一个原有的分区拆分为多个分区。通过为*partition_names*命名一个分区，并提供多个*partition_definitions*，可以实现这个功能。

o 更改使用VALUES LESS THAN定义的分区子集的范围或更改使用VALUES IN定义的分区子集的值清单。

注释：对于没有明确命名的分区，MySQL会自动提供默认名称p0, p1, p2等。

要了解有关ALTER TALBE...REORANIZE PARTITION命令的详细信息，请参见18.3节，“分区管理”。

· 多个附加子句用于提供分区维护和修补功能。这些功能与用于非分区表的功能类似。这些功能由CHECK TABLE和REPAIR TABLE等命令（这些命令不支持用于分区表）执行。这些子句包括ANALYZE PARTITION, CHECK PARTITION, OPTIMIZE PARTITION, REBUILD PARTITION和REPAIR PARTITION.每个选项均为一个*partition_names*子句，包括一个或多个分区名称。需要更改的表中必须已存在这些分区。多个分区名称用逗号分隔。要了解更多信息，或要了解举例说明，请参见18.3.3节，“分区维护”。

以下例子展示了ALTER TABLE的使用。首先展示表t1。表t1采用如下方法创建：

```
mysql> CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

把表t1重新命名为t2：

```
mysql> ALTER TABLE t1 RENAME t2;
```

把列a从INTERGER更改为TINYINT NOT NULL（名称保持不变），并把列b从CHAR(10)更改为CHAR(20)，同时把列b重新命名为列c：

```
mysql> ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

添加一个新的TIMESTAMP列，名称为d：

```
mysql> ALTER TABLE t2 ADD d TIMESTAMP;
```

在列d和列a中添加索引：

```
mysql> ALTER TABLE t2 ADD INDEX (d), ADD INDEX (a);
```

删除列c：

```
mysql> ALTER TABLE t2 DROP COLUMN c;
```

添加一个新的AUTO_INCREMENT整数列，名称为c：

```
mysql> ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
```

```
    ->     ADD PRIMARY KEY (c);
```

注意我们为c编制了索引（作为PRIMARY KEY），因为AUTO_INCREMENT列必须编制索引。同时我们定义c为NOT NULL，因为主键列不能为NULL。

当您添加一个AUTO_INCREMENT列时，列值被自动地按序号填入。对于MyISAM表，您可以在ALTER TABLE之前执行SET INSERT_ID=*value*来设置第一个序号，也可以使用AUTO_INCREMENT=*value*表选项来设置。请参见13.5.3节，“SET语法”。

如果值大于AUTO_INCREMENT列中的最大值，则您可以使用用于InnoDB表的ALTER TALBE...AUTO_INCREMENT=*value*表选项，来为新行设置序号。如果值小于列中当前的最大值，不会出现错误信息，当前的序列值也不改变。

使用MyISAM表时，如果您不更改AUTO_INCREMENT列，则序列号不受影响。如果您取消一个AUTO_INCREMENT列，然后添加另一个AUTO_INCREMENT列，则序号重新排列，从1开始。

见A.7.1节，“与ALTER TABLE有关的问题”。

13.1.3. CREATE DATABASE语法

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_specification [, create_specification] ...]

create_specification:
    [DEFAULT] CHARACTER SET charset_name
  | [DEFAULT] COLLATE collation_name
```

CREATE DATABASE用于创建数据库，并进行命名。如果要使用CREATE DATABASE，您需要获得数据库CREATE权限。

有关合法数据库名称的规定列于9.2节，“数据库、表、索引、列和别名”。如果存在数据库，并且您没有指定IF NOT EXISTS，则会出现错误。

create_specification选项用于指定数据库的特性。数据库特性储存在数据库目录中的db.opt文件中。CHARACTER SET子句用于指定默认的数据库字符集。COLLATE子句用于指定默认的数据库整序。字符集和整序名称在第10章：字符集支持中讨论。

有些目录包含文件，这些文件与数据库中的表对应。MySQL中的数据库的执行方法与这些目录的执行方法相同。因为当数据库刚刚被创建时，在数据库中没有表，所以CREATE DATABASE只创建一个目录。这个目录位于MySQL数据目录和db.opt文件之下。

如果您手动在数据目录之下创建一个目录（例如，使用**mkdir**），则服务器会认为这是一个数据库目录，并在SHOW DATABASES的输出中显示出来。

也可以使用CREATE SCHEMA。

您还可以使用**mysqladmin**程序创建数据库。请参见8.5节，“mysqladmin：用于管理MySQL服务器的客户端”。

13.1.4. CREATE INDEX语法

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
```

```
    [USING index_type]
    ON tbl_name (index_col_name,...)


index_col_name:
    col_name [(length)] [ASC | DESC]
```

CREATE INDEX被映射到一个ALTER TABLE语句上，用于创建索引。请参见13.1.2节，“ALTER TABLE语法”。

通常，当使用CREATE TABLE创建表时，也同时在表中创建了所有的索引。请参见13.1.5节，“CREATE TABLE语法”。CREATE INDEX允许您向已有的表中添加索引。

格式为（col1, col2,...）的一个列清单创建出一个多列索引。通过串接给定列中的值，确定索引值的格式。

对于CHAR和VARCHAR列，只用一列的一部分就可创建索引。创建索引时，使用*col_name*(*length*)语法，对前缀编制索引。前缀包括每列值的前*length*个字符。BLOB和TEXT列也可以编制索引，但是必须给出前缀长度。

此处展示的语句用于创建一个索引，索引使用列名称的前10个字符。

```
CREATE INDEX part_of_name ON customer (name(10));
```

因为多数名称的前10个字符通常不同，所以此索引不会比使用列的全名创建的索引速度慢很多。另外，使用列的一部分创建索引可以使索引文件大大减小，从而节省了大量的磁盘空间，有可能提高INSERT操作的速度。

前缀最长为255字节。对于MyISAM和InnoDB表，前缀最长为1000字节。注意前缀的限长以字节计，而CREATE INDEX语句中的前缀长度指的是字符的数目。对于使用多字节字符集的列，在指定列的前缀长度时，要考虑这一点。

在MySQL 5.1中：

- 只有当您正在使用MyISAM, InnoDB或BDB表类型时，您可以向有NULL值的列中添加索引。
- 只有当您正在使用MyISAM, BDB或InnoDB表类型时，您可以向BLOB或TEXT列中添加索引。

一个*index_col_name*规约可以以ASC或DESC为结尾。这些关键词将来可以扩展，用于指定递增或递减索引值存储。目前，这些关键词被分析，但是被忽略；索引值均以递增顺序存储。

部分储存引擎允许在创建索引时指定索引类型。*index_type*指定语句的语法是USING *type_name*。不同的储存引擎所支持的*type_name*值已显示在下表中。如果列有多个索引类型，当没有指定*index_type*时，第一个类型是默认值。

存储引擎	允许的索引类型
MyISAM	BTREE
InnoDB	BTREE
MEMORY/HEAP	HASH, BTREE

示例：

```
CREATE TABLE lookup (id INT) ENGINE = MEMORY;
CREATE INDEX id_index USING BTREE ON lookup (id);
```

TYPE *type_name*可以作为USING *type_name*的同义词，用于指定索引类型。但是，USING是首选的格式。另外，在索引规约语法中，位于索引类型前面的索引名称不能使用TYPE。这是因为，与USING不同，TYPE不是保留词，因此会被认为是一个索引名称。

如果您指定的索引类型在给定的储存引擎中不合法，但是有其它的索引类型适合引擎使用，并且不会影响查询功能，则引擎应使用此类型。

要了解更多有关MySQL如何使用索引的信息，请参见7.4.5节，“MySQL如何使用索引”。

FULLTEXT索引只能对CHAR, VARCHAR和TEXT列编制索引，并且只能在MyISAM表中编制。请参见12.7节，“全文搜索功能”。

SPATIAL索引只能对空间列编制索引，并且只能在MyISAM表中编制。空间列类型在第19章：*MySQL*中的空间扩展中进行了描述。

13.1.5. CREATE TABLE语法

13.1.5.1. 沉寂的列规格变更

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(create_definition,...)]
    [table_options] [select_statement]
```

或:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(] LIKE old_tbl_name [)];

create_definition:
    column_definition
  | [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)
  | KEY [index_name] [index_type] (index_col_name,...)
  | INDEX [index_name] [index_type] (index_col_name,...)
  | [CONSTRAINT [symbol]] UNIQUE [INDEX]
        [index_name] [index_type] (index_col_name,...)
  | [FULLTEXT|SPATIAL] [INDEX] [index_name] (index_col_name,...)
  | [CONSTRAINT [symbol]] FOREIGN KEY
        [index_name] (index_col_name,...) [reference_definition]
  | CHECK (expr)

column_definition:
    col_name type [NOT NULL | NULL] [DEFAULT default_value]
        [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
        [COMMENT 'string'] [reference_definition]

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]
    | DATE
    | TIME
    | TIMESTAMP
    | DATETIME
    | CHAR(length) [BINARY | ASCII | UNICODE]
    | VARCHAR(length) [BINARY]
    | TINYBLOB
    | BLOB
    | MEDIUMBLOB
    | LONGBLOB
    | TINYTEXT [BINARY]
    | TEXT [BINARY]
    | MEDIUMTEXT [BINARY]
    | LONGTEXT [BINARY]
    | ENUM(value1,value2,value3,...)
    | SET(value1,value2,value3,...)
    | spatial_type

  index_col_name:
      col_name [(length)] [ASC | DESC]

  reference_definition:
      REFERENCES tbl_name [(index_col_name,...)]
                 [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
                 [ON DELETE reference_option]
                 [ON UPDATE reference_option]

  reference_option:
      RESTRICT | CASCADE | SET NULL | NO ACTION
```

```
table_options: table_option [table_option] ...

table_option:
    {ENGINE|TYPE} = engine_name
  | AUTO_INCREMENT = value
  | AVG_ROW_LENGTH = value
  | [DEFAULT] CHARACTER SET charset_name [COLLATE collation_name]
  | CHECKSUM = {0 | 1}
  | COMMENT = 'string'
  | CONNECTION = 'connect_string'
  | MAX_ROWS = value
  | MIN_ROWS = value
  | PACK_KEYS = {0 | 1 | DEFAULT}
  | PASSWORD = 'string'
  | DELAY_KEY_WRITE = {0 | 1}
  | ROW_FORMAT = {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}
  | UNION = (tbl_name[,tbl_name]...)
  | INSERT_METHOD = { NO | FIRST | LAST }
  | DATA DIRECTORY = 'absolute path to directory'
  | INDEX DIRECTORY = 'absolute path to directory'

partition_options:
    PARTITION BY
           [LINEAR] HASH(expr)
        |  [LINEAR] KEY(column_list)
        |  RANGE(expr)
        |  LIST(column_list)
    [PARTITIONS num]
    [  SUBPARTITION BY
           [LINEAR] HASH(expr)
         | [LINEAR] KEY(column_list)
      [SUBPARTITIONS(num)]
    ]
    [(partition_definition), [(partition_definition)], ...]

partition_definition:
    PARTITION partition_name
        [VALUES {
                  LESS THAN (expr) | MAXVALUE
```

```
                | IN (value_list) }]
        [[STORAGE] ENGINE [=] engine-name]
        [COMMENT [=] 'comment_text' ]
        [DATA DIRECTORY [=] 'data_dir']
        [INDEX DIRECTORY [=] 'index_dir']
        [MAX_ROWS [=] max_number_of_rows]
        [MIN_ROWS [=] min_number_of_rows]
        [TABLESPACE [=] (tablespace_name)]
        [NODEGROUP [=] node_group_id]
        [(subpartition_definition), [(subpartition_definition)], ...]


subpartition_definition:
    SUBPARTITION logical_name
        [[STORAGE] ENGINE [=] engine-name]
        [COMMENT [=] 'comment_text' ]
        [DATA DIRECTORY [=] 'data_dir']
        [INDEX DIRECTORY [=] 'index_dir']
        [MAX_ROWS [=] max_number_of_rows]
        [MIN_ROWS [=] min_number_of_rows]
        [TABLESPACE [=] (tablespace_name)]
        [NODEGROUP [=] node_group_id]


select_statement:
    [IGNORE | REPLACE] [AS] SELECT ...   (Some legal select statement)
```

CREATE TABLE用于创建带给定名称的表。您必须拥有表CREATE权限。

允许的表名称的规则列于9.2节，“数据库、表、索引、列和别名”中。默认的情况是，表被创建到当前的数据库中。如果表已存在，或者如果没有当前数据库，或者如果数据库不存在，则会出现错误。

表名称被指定为*db_name.tbl_name*，以便在特定的数据库中创建表。不论是否有当前数据库，都可以通过这种方式创建表。如果您使用加引号的识别名，则应对数据库和表名称分别加引号。例如，`mydb`.`mytbl`是合法的，但是`mydb.mytbl`不合法。

在创建表格时，您可以使用TEMPORARY关键词。只有在当前连接情况下，TEMPORARY表才是可见的。当连接关闭时，TEMPORARY表被自动取消。这意味着两个不同的连接可以使用相同的临时表名称，同时两个临时表不会互相冲突，也不与原有的同名的非临时表冲突。（原有的表被隐藏，直到临时表被取消时为止。）您必须拥有CREATE TEMPORARY TABLES权限，才能创建临时表。

如果表已存在，则使用关键词IF NOT EXISTS可以防止发生错误。注意，原有表的结构与CREATE TABLE语句中表示的表的结构是否相同，这一点没有验证。注释：如果您在CREATE TABLE...SELECT语句中使用IF NOT EXISTS，则不论表是否已存在，由SELECT部分选择的记录都会被插入。

MySQL通过数据库目录中的.frm表格式（定义）文件表示每个表。表的存储引擎也可能会创建其它文件。对于MyISAM表，存储引擎可以创建数据和索引文件。因此，对于每个MyISAM表*tbl_name*，有三个磁盘文件：

文件	作用
tbl_name.frm	表格式（定义）文件
tbl_name.MYD	数据文件
tbl_name.MYI	索引文件

用于表示表的由存储引擎创建的文件在第15章：存储引擎和表类型中描述。

要了解有关各种列类型的性质的一般说明，请参见第11章：列类型。要了解有关空间列类型的说明，请参见第19章：*MySQL*中的空间扩展。

- 如果没有指定是NULL或是NOT NULL，则列在创建时假定指定为NULL。

- 一个整数列可以拥有一个附加属性AUTO_INCREMENT。当您向一个已编入索引的AUTO_INCREMENT列中插入一个NULL值（建议）或0时，此列被设置为下一个序列的值。通常情况下为*value*+1，此处*value*是当前在表中的列的最大值。AUTO_INCREMENT序列从1开始。这样的列必须被定义为一种整数类型，请参见11.1.1节，“数值类型概述”中的叙述。（值1.0不是整数）。请参见25.2.3.36节，“mysql_insert_id()”。

为--sql-mode服务器选项或sql_mode系统变量指定NO_AUTO_VALUE_ON_ZERO特征位，这样可以把0存储到AUTO_INCREMENT列中，同时不生成一个新的序列值。请参见5.3.1节，“**mysqld**命令行选项”。

注释：有时候，每个表只有一个AUTO_INCREMENT列，此列必须编制索引，不能有DEFAULT值。一个AUTO_INCREMENT列只有在只包含正数的情况下，才能运行正常。插入一个负数会被认为是插入了一个非常大的正数。这样做是为了避免当数字由正数转为负数时出现精度问题，同时也为了确保AUTO_INCREMENT列中不会包含0。

对于MyISAM和BDB表，您可以在一个多列关键字中指定一个AUTO_INCREMENT次级列。请参见3.6.9节，“使用AUTO_INCREMENT”。

为了让MySQL与部分ODBC应用软件相兼容，您可以使用以下查询方法找到最后一个插入行的AUTO_INCREMENT值：

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

- 字符列的定义可以包括一个CHARACTER SET属性，用来指定字符集，也可以指定列的整序。要了解详细情况，请参见第10章：字符集支持。CHARSET是CHARACTER SET的同义词。

```
CREATE TABLE t (c CHAR(20) CHARACTER SET utf8 COLLATE utf8_bin);
```

MySQL 5.1理解，在字符列定义中的长度规约以字符为单位。（有些早期版本以字节为单位。）

- DEFAULT子句用于为列指定一个默认值。默认值必须为一个常数，不能为一个函数或一个表达式，有一种情况例外。例如，一个日期列的默认值不能被设置为一个函数，如NOW()或CURRENT_DATE。不过，有一种例外，您可以对TIMESTAMP列指定CURRENT_TIMESTAMP为默认值。请参见11.3.1.1节，“MySQL 4.1中的TIMESTAMP属性”。

BLOB和TEXT列不能被赋予默认值。

如果在列定义中没有明确的DEFAULT值，则MySQL按照如下规则确定默认值：

如果列可以使用NULL作为值，则使用DEFAULT NULL子句对列进行定义。（在MySQL的早期版本中也如此。）

如果列不能使用NULL作为值，则MySQL对列进行定义时不使用DEFAULT子句。输入数据时，如果INSERT或REPLACE语句不包括列的值，则MySQL依据当时的有效的SQL模式操作列：

o 如果严格模式没有被启用，则MySQL会根据列数据类型，把列设置为明确的默认值。

o 如果严格模式已被启用，则事务表会出现错误，语句被回滚。对于非事务表，会出现错误，不过，如果错误出现在一个多行语句中的第二行或后续行，则以前的各行将被插入。

假设表t按下面的方法进行定义：

```
CREATE TABLE t (i INT NOT NULL);
```

在这种情况下，i没有明确的默认值，所以在严格模式中，每个后续语句都会产生一个错误，并且没有行被插入。当未使用严格模式时，只有第三个语句产生错误；明确的默认值被插入到前两个语句中，但是第三个语句会出现错误，因为DEFAULT(i)不会产生一个值：

```
INSERT INTO t VALUES();

INSERT INTO t VALUES(DEFAULT);

INSERT INTO t VALUES(DEFAULT(i));
```

见5.3.2节，“SQL服务器模式”。

对于一个给定的表，您可以使用SHOW CREATE TABLE语句来查看那些列有明确的DEFAULT子句。

· 对于列的评注可以使用COMMENT选项来进行指定。评注通过SHOW CREATE TABLE和SHOW FULL COLUMNS语句显示。

· 属性SERIAL可以用作BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE的别名。

· KEY通常是INDEX同义词。如果关键字属性PRIMARY KEY在列定义中已给定，则PRIMARY KEY也可以只指定为KEY。这么做的目的是与其它数据库系统兼容。

· 在UNIQUE索引中，所有的值必须互不相同。如果您在添加新行时使用的关键字与原有行的关键字相同，则会出现错误。例外情况是，如果索引中的一个列允许包含NULL值，则此列可以包含多个NULL值。此例外情况不适用于BDB表。在BDB中，带索引的列只允许一个单一NULL。

· PRIMARY KEY是一个唯一KEY，此时，所有的关键字列必须定义为NOT NULL。如果这些列没有被明确地定义为NOT NULL，MySQL应隐含地定义这些列。一个表只有一个PRIMARY KEY。如果您没有PRIMARY KEY并且一个应用程序要求在表中使用PRIMARY KEY，则MySQL返回第一个UNIQUE索引，此索引没有作为PRIMARY KEY的NULL列。

· 在已创建的表中，PRIMARY KEY的位置最靠前，然后是所有的UNIQUE索引，然后是非唯一索引。这可以帮助MySQL优化程序选择优先使用哪个索引，并且更快速的检测出重复的UNIQUE关键字。

· PRIMARY KEY可以是一个多列索引。但是，在列规约中使用PRIMARY KEY关键字属性无法创建多列索引。这么做只能把一个列标记为主列。您必须使用一个单独的PRIMARY KEY（index_col_name, ...）子句。

· 如果PRIMARY KEY或UNIQUE索引只包括一个列，并且此列为整数类型，则您也可以在SELECT语句中把此列作为_rowid引用。

· 在MySQL中，PRIMARY KEY的名称为PRIMARY。对于其它索引，如果您没有赋予名称，则索引被赋予的名称与第一个已编入索引的列的名称相同，并自选添加后缀（_2, _3,...），使名称为唯一名称。您可以使用SHOW INDEX FROM *tbl_name*来查看表的索引名称。请参见13.5.4.11节，“SHOW INDEX语法”。

· 部分存储引擎允许您在创建索引时指定索引类型。*index_type*指示语句的语法是USING type_name。

示例：

```
CREATE TABLE lookup
  (id INT, INDEX USING BTREE (id))
```

```
  ENGINE = MEMORY;
```

要了解有关USING的详细说明，请参见13.1.4节，“CREATE INDEX语法”。

要了解有关MySQL如何使用索引的更多信息，请参见7.4.5节，“MySQL如何使用索引”。

- 在MySQL 5.1中，只有MyISAM，InnoDB, BDB和MEMORY存储引擎支持在含有NULL值的列中编索引。在其它情况下，您必须定义已编索引的列为NOT NULL，否则会出现错误。

- 在一个索引规约中使用*col_name*(*length*)语法，您可以创建一个索引，此索引只使用一个CHAR或VARCHAR列的第一个*length*字符。只对列值的前缀编制索引可以使索引文件大大减小。请参见7.4.3节，“列索引”。

MyISAM和InnoDB存储引擎也支持对BLOB和TEXT列编索引。当对BLOB或TEXT列编索引时，您必须为索引指定一个前缀长度。例如：

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

对于MyISAM和InnoDB表，前缀最长可以为1000字节，对于其它表格类型，最长可以为255字节。注意前缀长度限值以字节为单位，而在CREATE TABLE语句中的前缀长度用字符数目来表述。当为一个使用多字节字符集的列指定前缀长度时，一定要考虑到这一点。

- 一个*index_col_name*规约可以以ASC或DESC结尾。这些关键词可以在将来进行扩展，用于指定升序或降序的索引值存储。当前，这些关键词被分析但是被忽略；索引值均以升序储存。

- 当您在SELECT中的TEXT列或BLOB列中使用ORDER BY或GROUP BY时，服务器只使用初始的字节数目对值进行分类。字节数目由max_sort_length系统变量进行指示。请参见11.4.3节，“BLOB和TEXT类型”。

- 您可以创建特殊的FULLTEXT索引，用于全文搜索。只有MyISAM表类型支持FULLTEXT索引。FULLTEXT索引只可以从CHAR, VARCHAR和TEXT列中创建。整个列都会被编入索引；不支持对部分列编索引。如果已指定，前缀长度会被忽略。要了解运行的详细说明，请参见12.7节，“全文搜索功能”。

- 您可以为空间列类型创建SPATIAL索引。只有MyISAM表支持空间类型，已编索引的列必须声明为NOT NULL。请参见第19章：*MySQL*中的空间扩展。

- InnoDB表支持对外键限制条件进行检查。请参见15.2节，“InnoDB存储引擎”。注意，在InnoDB中，FOREIGN KEY语法比本节开始时介绍的CREATE TABLE语句的语法更严格：被引用的表中的列必须有明确的命名。InnoDB支持外键的ON DELETE和ON UPDATE两种操作。有关精确语法的说明，请参见15.2.6.4节，“FOREIGN KEY约束”。

对于其它存储引擎，MySQL服务器对CREATE TABLE语句中的FOREIGN KEY和REFERENCES语法进行分析，但不采取进一步的行动。所有的存储引擎均对CHECK子句进行分析，但是忽略CHECK子句。请参见1.8.5.5节，“外键”。

- 对于MyISAM表，每个NULL列要多占用一位，进位到距离最近的字节。最大记录长度（以字节为单位）按照如下方法计算：

```
row length = 1
           + (sum of column lengths)
           + (number of NULL columns + delete_flag + 7)/8
           + (number of variable-length columns)
```

对于采用静态记录格式的表，*delete_flag*为1。静态表在行记录中使用一位用作位标记。位标记指示该行是否已被删除。对于动态表，*delete_flag*为0，因为在动态行标题中已存储了位标记。

这些计算方法不适用于InnoDB表。对于InnoDB表，NULL列的存储量与NOT NULL列的存储量没有区别。

ENGINE和TYPE选项用于为表指定存储引擎。ENGINE是首选的选项名称。

ENGINE和TYPE选项采用以下值:

存储引擎	说明
ARCHIVE	档案存储引擎。请参见15.8节，“ARCHIVE存储引擎”。
BDB	带页面锁定的事务安全表。也称为BerkeleyDB。请参见15.5节，“BDB (BerkeleyDB)存储引擎”。
CSV	值之间用逗号隔开的表。请参见15.9节，“CSV存储引擎。
EXAMPLE	示例引擎。请参见15.6节，“EXAMPLE存储引擎”。
FEDERATED	可以访问远程表的存储引擎。请参见15.7节，“FEDERATED存储引擎”。
HEAP	见15.4节，“MEMORY (HEAP)存储引擎”。
(*OBSOLETE*) ISAM	在MySQL 5.1中没有此引擎。如果您要从以前的版本升级到MySQL 5.1，您应该在进行升级前把原有的ISAM表转换为MyISAM表。请参见第15章：存储引擎和表类型。
InnoDB	带行锁定和外键的事务安全表。请参见15.2节，“InnoDB存储引擎”。
MEMORY	本表类型的数据只保存在存储器里。（在早期MySQL版本中被称为HEAP。）
MERGE	MyISAM表的集合，作为一个表使用。也称为MRG_MyISAM。请参见15.3节，“MERGE存储引擎”。
MyISAM	二进制轻便式存储引擎，此引擎是MySQL所用的默认存储引擎。请参见15.1节，“MyISAM存储引擎”。
NDBCLUSTER	成簇表，容错表，以存储器为基础的表。也称为NDB。请参见第17章：*MySQL*簇。

要了解有关MySQL存储引擎的更多信息，请参见第15章：存储引擎和表类型。

如果被指定的存储引擎无法利用，则MySQL使用MyISAM代替。例如，一个表定义包括ENGINE=BDB选项，但是MySQL服务器不支持BDB表，则表被创建为MyISAM表。这样，如果您在主机上有事务表，但在从属机上创建的是非交互式表（以加快速度）时，可以进行复制设置。在MySQL 5.1中，如果没有遵守存储引擎规约，则会出现警告。

其它表选项用于优化表的性质。在多数情况下，您不必指定表选项。这些选项适用于所有存储引擎，另有说明除外:

- AUTO_INCREMENT

表的初始AUTO_INCREMENT值。在MySQL 5.1中，本选项只适用于MyISAM和MEMORY表。InnoDB也支持本选项。如果引擎不支持AUTO_INCREMENT表选项，则要设置引擎的第一个auto-increment值，需插入一个“假”行。该行的值比创建表后的值小一，然后删除该假行。

对于在CREATE TABLE语句中支持AUTO_INCREMENT表选项的引擎，您也可以使用ALTER TABLE *tbl_name* AUTO_INCREMENT = *n*来重新设置AUTO_INCREMENT值。

- AVG_ROW_LENGTH

表中平均行长度的近似值。只需要对含尺寸可变的记录的大型表进行此项设置。

当创建一个MyISAM表时，MySQL使用MAX_ROWS和AVG_ROW_LENGTH选项的乘积来确定得出的表有多大。如果有一个选项未指定，则表的最大尺寸为65,536TB数据。（如果操作系统不支持这么大的文件，则表的尺寸被限定在操作系统的限值处。）如果您想缩小指针尺寸使索引更小，速度更快，并且您不需要大文件，则您可以通过设置myisam_data_pointer_size系统变量来减少默认指针的尺寸。（见5.3.3节，“服务器系统变量”。）如果您希望所有的表可以扩大，超过默认限值，并且愿意让表稍微慢点，并稍微大点，则您可以通过设置此变

量增加默认指针的尺寸。

· [DEFAULT] CHARACTER SET

用于为表指定一个默认字符集。CHARSET是CHARACTER SET的同义词。

对于CHARACTER SET.

· COLLATE

用于为表指定一个默认整序。

· CHECKSUM

如果您希望MySQL随时对所有行进行实时检验求和（也就是，表变更后，MySQL自动更新检验求和），则应把此项设置为1。这样做，表的更新速度会略微慢些，但是更容易寻找到受损的表。CHECKSUM TABLE语句用于报告检验求和（仅限于MyISAM）。

· COMMENT

表的注释，最长60个字符。

· CONNECTION

FEDERATED表的连接字符串。（ 注释：较早版本的MySQL使用COMMENT选项用于连接字符串。

· MAX_ROWS

您打算储存在表中的行数目的最大值。这不是一个硬性限值，而更像一个指示语句，指示出表必须能存储至少这么多行。

· MIN_ROWS

您打算存储在表中的行数目的最小值。

· PACK_KEYS

如果您希望索引更小，则把此选项设置为1。这样做通常使更新速度变慢，同时阅读速度加快。把选项设置为0可以取消所有的关键字压缩。把此选项设置为DEFAULT时，存储引擎只压缩长的CHAR或VARCHAR列（仅限于MyISAM）。

如果您不使用PACK_KEYS，则默认操作是只压缩字符串，但不压缩数字。如果您使用PACK_KEYS=1，则对数字也进行压缩。

在对二进制数字关键字进行压缩时，MySQL采用前缀压缩：

o 每个关键字需要一个额外的字节来指示前一个关键字中有多少字节与下一个关键字相同。

o 指向行的指针以高位字节优先的顺序存储在关键字的后面，用于改进压缩效果。

这意味着，如果两个连续行中有许多相同的关键字，则后续的"相同"的关键字通常只占用两个字节（包括指向行的指针）。与此相比，常规情况下，后续的关键字占用storage_size_for_key + pointer_size（指针尺寸通常为4）。但是，只有在许多数字相同的情况下，前缀压缩才有好处。如果所有的关键字完全不同，并且关键字不能含有NULL值，则每个关键字要多使用一个字节。（在这种情况中，储存压缩后的关键字的长度的字节与用于标记关键字是否为NULL的字节是同一字节。）

· PASSWORD

使用密码对.frm文件加密。在标准MySQL版本中，本选项不起任何作用。

· DELAY_KEY_WRITE

如果您想要延迟对关键字的更新，等到表关闭后再更新，则把此项设置为1（仅限于MyISAM）。

· ROW_FORMAT

定义各行应如何储存。当前，此选项只适用于MyISAM表。对于静态行或长度可变行，此选项值可以为FIXED或DYNAMIC。**myisampack**用于把类型设置为COMPRESSED。请参见15.1.3节，“MyISAM表的存储格式”。

在默认情况下，InnoDB记录以压缩格式存储（ROW_FORMAT=COMPACT）。通过指定ROW_FORMAT=REDUNDANT，仍然可以申请用于较早版本的MySQL中的非压缩格式。

· RAID_TYPE

在MySQL 5.0中,RAID支持被删除了。要了解有关RAID的说明，请参见http://dev.mysql.com/doc/refman/4.1/en/create-table.html。

· UNION

当您想要把一组相同的表当作一个表使用时，采用UNION。UNION仅适用于MERGE表。请参见15.3节，“MERGE存储引擎”。

对于您映射到一个MERGE表上的表，您必须拥有SELECT, UPDATE和DELETE权限。（注释：以前，所有被使用的表必须位于同一个数据库中，并作为MERGE表。这些限制不再适用。）

· INSERT_METHOD

如果您希望在MERGE表中插入数据，您必须用INSERT_METHOD指定应插入行的表。INSERT_METHOD选项仅用于MERGE表。使用FIRST或LAST把行插入到第一个或最后一个表中；或者使用NO，阻止插入行。请参见15.3节，“MERGE存储引擎”。

· DATA DIRECTORY, INDEX DIRECTORY

通过使用DATA DIRECTORY='*directory*'或INDEX DIRECTORY='*directory*'，您可以指定MyISAM存储引擎放置表格数据文件和索引文件的位置。注意，目录应是通向目录的完整路径（不是相对路径）。

仅当您没有使用--skip-symbolic-links选项时，DATA DIRECTORY, INDEX DIRECTORY才能使用。操作系统必须有一个正在工作的、线程安全的realpath()调用。要了解全面信息，请参见7.6.1.2节，“在Unix平台上使用表的符号链接”。

· 对于用CREATE TABLE创建的表，可以使用*partition_options*控制分区。如果使用了*partition_options*，则其中必须包含至少一个PARTITION BY子句。本子句包含用于确定分区的函数；该函数会返回一个整值，范围从1到*num*。此处*num*为分区的数目。此函数中可以使用的选项显示在下面的清单中。 要点：在本节开始时介绍的用于*partition_options*的语法中显示的选项，并不是都能用于所有分区类型。要了解各种类型具体的信息 ，请参见以下各类型的清单。要了解有关在MySQL中的分区的操作和使用情况的全面说明，以及要了解表创建的示例和与MySQL分区有关的其它命令，请参见第18章：分区。

o HASH（*expr*）：用于混编一个或多个列，创建一个关键字，用于放置行，并确定行的位置。*expr*是一个表达式，使用一个或多个表中的列。该表达式可以是任何能够生成单一整值的合法的MySQL表达式（包括MySQL函数）。例如，这些都是有效的CREATE TABLE语句，语句中使用了PARTITION BY HASH：

o
```
CREATE TABLE t1 (col1 INT, col2 CHAR(5))
```

```
o                           PARTITION BY HASH(col1);
o
o                       CREATE TABLE t1 (col1 INT, col2 CHAR(5))
o                           PARTITION BY HASH( ORD(col2) );
o
o                       CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATETIME)
o                           PARTITION BY HASH ( YEAR(col3) );
```

VALUES LESS THAN或VALUES IN子句不能和PARTITION BY HASH一起使用。

PARTITION BY HASH使用*expr*被分区数目所除后的余数（也就是模数）。要了解示例和其它信息，请参见18.2.3节，“HASH分区”。

LENEAR关键词需要一种不同的算法。在这种情况下，通过一次或多次逻辑AND运算得出的结果，计算出存储记录的分区的数目。要了解线形混编的讨论和示例，请参见18.2.3.1节，“LINEAR HASH分区”。

o KEY(*column_list*)：与HASH近似，除了有一点不一样，即MySQL提供了混编函数，以保证均匀的数据分布。*column_list*自变量只是各列的一个清单。本示例显示了由关键字进行分区的一个简单的表，分为4个分区：

```
o                       CREATE TABLE tk (col1 INT, col2 CHAR(5), col3 DATE)
o                           PARTITION BY KEY(col3)
o                           PARTITIONS 4;
```

采用LINEAR关键词，您可以对由关键字分区的表进行线形分区。这与由HASH进行分区的表格有同样的效果；也就是说，使用&操作符查找分区数目，而不是使用模数（详细说明见18.2.3.1节，“LINEAR HASH分区”和18.2.4节，“KEY分区”）。本示例采用了关键字线形分区，用来在5个分区之间分配数据：

```
CREATE TABLE tk (col1 INT, col2 CHAR(5), col3 DATE)
    PARTITION BY LINEAR KEY(col3)
    PARTITIONS 5;
```

VALUES LESS THAN或VALUES IN子句不能和PARTITION BY KEY一起使用。

o RANGE：在此情况下，*expr*使用一套VALUES LESS THAN操作符显示了某一范围内的值。当使用范围分区时，您必须使用VALUES LESS THAN定义至少一个分区。VALUES IN不能和范围分区一起使用。

VALUES LESS THAN可以与一个文字值同时使用，或者与一个可以求算单一值的表达式同时使用。

举例说明，假设您有一个表，您希望采用以下方法对包含年份值的一列进行分区：

分区编号：	年份范围：
0	1990以前
1	1991 - 1994
2	1995 - 1998
3	1999 - 2002
4	2003 - 2005
5	2006年以后

采用这种分区方法的表可以通过如下CREATE TABLE语句实现：

```
CREATE TABLE t1 (
    year_col INT,
    some_data INT
)
PARTITION BY RANGE (year_col) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1995),
    PARTITION p2 VALUES LESS THAN (1999),
    PARTITION p3 VALUES LESS THAN (2002),
    PARTITION p4 VALUES LESS THAN (2006),
    PARTITION p5 VALUES LESS THAN MAXVALUE
);
```

PARTITION ... VALUES LESS THAN ...语句按顺序执行。VALUES LESS THAN MAXVALUE的作用是指定大于最大值的"其余"的值。

注意，VALUES LESS THAN子句按顺序执行，执行方式类似于switch ... case语段的一部分（许多编程语言，如C, Java和PHP也如此）。也就是说，子句必须按照这样一种方法排列，每一个后续的VALUES LESS THAN中指定的上限值大于前一个VALUES LESS THAN中指定的上限值，并在清单的最后加一个参照性的MAXVALUE。

VALUES IN与一系列的值同时使用。举例说明，您可以创建如下的分区方法：

```
CREATE TABLE client_firms (
    id INT,
    name VARCHAR(35)
)
PARTITION BY RANGE (id) (
    PARTITION r0 VALUES IN (1, 5, 9, 13, 17, 21),
    PARTITION r1 VALUES IN (2, 6, 10, 14, 18, 22),
    PARTITION r2 VALUES IN (3, 7, 11, 15, 19, 23),
    PARTITION r3 VALUES IN (4, 8, 12, 16, 20, 24)
);
```

当前，与VALUES IN...同时使用的值必须只包含整数值。

（因为此表只使用VALUES IN表达式进行分区，您也可以用PARTITION BY LIST代替，而不是使用PARTITION BY RANGE。请参见下一条。）

在使用VALUES LESS THAN或VALUES IN情况下，每个分区使用PARTITION *name*定义，此处*name*是分区的标识名，后面接VALUES...子句。

o LIST(*expr*)：当根据含有一系列限定性值（例如州代码或国家代码）的列进行分区时使用。在这种情况下，所有与特定的州或国家有关的记录都被分配到一个单一分区中，或者可以预留出一个分区，用于一系列特定的州或国家。LIST(*expr*)与RANGE类似，除了一点以外，即只有VALUES IN可以被用于为每个分区指定值。

当使用清单分区时，您必须使用VALUES IN定义至少一个分区。VALUES LESS THAN不能与PARTITION BY LIST一起使用。

o 分区数目可以使用PARTITION *num*子句，自选进行指定，此处，*num*是分区的数目。如果本子句和其它PARTITION子句同时使用，则*num*必须与使用PARTITION子句说明的分区的总数相等。

注释：不论您在创建一个由RANGE或LIST进行分区的表时是否使用了PARTITIONS子句，您必须在表定义中包括至少一个PARTITION VALUES（见后）。

o 一个分区可以自选分隔成多个子分区。使用自选的SUBPARTITION BY子句可以指示。子分区可以由HASH或KEY进行分隔。两种方法建立的子分区均为LINEAR。分隔子分区时的操作方式与以前描述的分区类型的操作方式一样。（无法由LIST或RANGE进行子分区分隔。）

使用SUBPARTITIONS关键词，后面接一个整值，可以对子分区的数目进行指示。

· 使用一个*partition_definition*子句可以对每个分区分别进行定义。下面是组成这个子句的各个部分：

o PARTITION *partition_name*：用于为分区指定一个逻辑名称。

o VALUE子句：对于范围分区，每个分区必须包括一个VALUES LESS THAN子句；对于清单分区，您必须为每个分区指定一个VALUES IN子句。本子句用于确定哪些行将被存储到此分区中。要了解语法示例，请参见第18章：分区中对分区类型的讨论。

o 自选的COMMENT子句可以用于描述分区。注释必须加单引号。举例说明：

```
o                    COMMENT = 'Data for the years previous to 1999'
```

o DATA DIRECTORY和INDEX DIRECTORY可以被用于指示本分区的数据和索引各自的存储位置的目录。*data_dir*和*index_dir*都必须是绝对系统路径。例如：

```
o                    CREATE TABLE th (id INT, name VARCHAR(30), adate DATE)

o                    PARTITION BY LIST(YEAR(adate))

o                    (

o                        PARTITION p1999 VALUES IN (1995, 1999, 2003) DATA DIRECTORY =
'/var/appdata/95/data' INDEX DIRECTORY = '/var/appdata/95/idx',

o                        PARTITION p2000 VALUES IN (1996, 2000, 2004) DATA DIRECTORY =
'/var/appdata/96/data' INDEX DIRECTORY = '/var/appdata/96/idx',

o                        PARTITION p2001 VALUES IN (1997, 2001, 2005) DATA DIRECTORY =
'/var/appdata/97/data' INDEX DIRECTORY = '/var/appdata/97/idx',

o                        PARTITION p2000 VALUES IN (1998, 2002, 2006) DATA DIRECTORY =
'/var/appdata/98/data' INDEX DIRECTORY = '/var/appdata/98/idx'

);
```

DATA DIRECTORY和INDEX DIRECTORY的操作方法与CREATE TABLE语句中的*table_option*子句的操作方法一样。此*table_option*子句用于位于MyISAM表管理程序下的各表。

可以为每个分区指定一个数据目录和一个索引目录。如果不指定，则数据和索引被存储在默认的MySQL数据目录中。

o MAX_ROWS和MIN_ROWS分别用于将被存储在分区中的行数目最大值和行数目最小值。*max_number_of_rows*和*min_number_of_rows*的值必须为正整数。和具有同样名称的桌面选项一样，*max_number_of_rows*和*min_number_of_rows*只作为对服务器的"建议"值，并不是硬性限值。

o 自选的TABLESPACE子句可以用于为分区指定一个桌面空间。仅用于MySQL Cluster。

o 自选的[STORAGE] ENGINE子句可以把本分区中表的类型改为指定的类型。表的类型可以是本MySQL服务器支持的所有类型。STORAGE关键字和等号(=)均为自选项。如果没有使用此选项设置分区存储引擎，则适用于整个表的引擎可以用于此分区。

注释：分区管理程序对于PARTITION和SUBPARTITION均接受[STORAGE] ENGINE选项。目前，此子句的使用方式仅限于对所有的分区或子分区设置同一个存储引擎，如果试图在同一个表内对不同的分区或子分区设置不同的存储引擎，则会出现错误ERROR 1469 (HY000)：在本版本的MySQL中，不允许在各分区中混用管理程序。我们打算在将来的MySQL 5.1版本中加入这种对分区的限定。

o NODEGROUP选项可以用于使本分区可以作为节点组的一部分，节点组使用*node_group_id*识别。本选项仅适用于MySQL Cluster。

o 分区定义可以自选地包含一个或多个*subpartition_definition*子句。每个这种子句至少包括SUBPARTITION *name*，此处，*name*是子分区的识别名称。除了用SUBPARTITION代替PARTITION关键词外，用于子分区定义的语法与用于分区定义的语法一样。

子分区必须由HASH或KEY完成，并且只能对RANGE或LIST分区进行子分区。请参见18.2.5节，“子分区”。

· 分区可以修改、合并、添加到表中，或从表中删去。要了解有关完成这些任务的MySQL命令的基本说明，请参见13.1.2节，“ALTER TABLE语法”。要了解详细的说明和示例，请参见18.3节，“分区管理”。

您可以在CREATE TABLE语句的末尾添加一个SELECT语句，在一个表的基础上创建表。

```
CREATE TABLE new_tbl SELECT * FROM orig_tbl;
```

MySQL会对SELECT中的所有项创建新列。举例说明：

```
mysql> CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,
    ->        PRIMARY KEY (a), KEY(b))
    ->        TYPE=MyISAM SELECT b,c FROM test2;
```

本语句用于创建含三个列（a, b, c）的MyISAM表。注意，用SELECT语句创建的列附在表的右侧，而不是覆盖在表上。参考以下示例：

```
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)
```

对应于表foo中的每一行，在表bar中插入一行，含有表foo中的值以及新列中的默认值。

在由CREATE TABLE...SELECT生成的表中，只在CREATE TABLE部分中命名的列首先出现。在两个部分中都命名的列和只在SELECT部分中命名的列随后出现。也可以通过指定CREATE TABLE部分中的列覆盖SELECT列中的数据类型。

如果在把数据复制到表中时出现错误，则表会自动被取消，不会被创建。

CREATE TABLE...SELECT不会自动创建任何索引。索引需要专门创建，以便使语句的灵活性更强。如果您希望为已创建的表建立索引，您应在SELECT语句前指定索引。

```
mysql> CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

列的类型会发生部分转化。例如，AUTO_INCREAMENT属性不会被保留，VARCHAR列会变成CHAR列。

当使用CREATE...SELECT创建表时，在查询时一定要对功能调用和表达式起别名。如果不起别名，则CREATE语句会出现错误或者生成不符合需要的列名称。

```
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;
```

您也可以明确地为一个已生成的列指定类型:

```
CREATE TABLE foo (a TINYINT NOT NULL) SELECT b+1 AS a FROM bar;
```

根据其它表的定义（包括在原表中定义的所有的列属性和索引），使用LIKE创建一个空表:

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

CREATE TABLE...LIKE不会复制对原表或外键定义指定的DATA DIRECTORY或INDEX DIRECTORY表选项。

您可以在SELECT前增加IGNORE或REPLACE，指示如何对复制唯一关键字值的记录进行操纵。使用IGNORE后，如果新记录复制了原有的唯一关键字值的记录，则新记录被丢弃。使用REPLACE后，新记录替换具有相同的唯一关键字值的记录。如果没有指定IGNORE或REPLACE，则出现多重唯一关键字值时会导致发生错误。

为了确保更新日志/二进位日志可以被用于再次创建原表，MySQL不允许在CREATE TABLE...SELECT过程中进行联合插入。

13.1.5.1. 沉寂的列规格变更

在有些情况下，较早版本的MySQL会静默地更改在CREATE TABLE或ALTER TABLE语句中给定的列规约。在MySQL 5.1中不会进行这类变更。如果使用指定的数据类型无法创建列，则会出现错误。

13.1.6. DROP DATABASE语法

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

DROP DATABASE用于取消数据库中的所用表格和取消数据库。使用此语句时要非常小心！如果要使用DROP DATABASE，您需要获得数据库DROP权限。

IF EXISTS用于防止当数据库不存在时发生错误。

也可以使用DROP SCHEMA。

如果您对一个带有符号链接的数据库使用DROP DATABASE，则链接和原数据库都被取消。

DROP DATABASE会返回已被取消的表的数目。此数目相当于被取消的.frm文件的数目。

在正常操作中MySQL自身会创建出一些文件和目录。DROP DATABASE语句会从给定的数据库目录中取消这些文件和目录：

- 所有带这些扩展名的文件：

.BAK	.DAT	.HSH	
.MRG	.MYD	.ISD	
.MYI	.db	.frm	

- 名称中包含两位16进制数00-ff的所有子目录。这些子目录用于RAID表。（当对RAID表的支持被取消时，在MySQL 5.0中，这些目录不会被取消。您应该在升级到MySQL 5.0或更新的版本前转化原有的RAID表，并人工取消这些目录。请参见MySQL 5.0参考手册中有关从较早版本升级到MySQL 5.0的章节。MySQL 5.0参考手册可以从MySQL网站中获取。）

- db.opt文件

如果在MySQL取消了上述这些文件之后，在数据库目录中仍保留有其它文件和目录，则数据库目录不能被取消。在这种情况下，您必须人工取消所有保留下的文件或目录，并再次发送DROP DATABASE语句。

您还可以使用**mysqladmin**来取消文件。请参见8.5节，“mysqladmin：用于管理MySQL服务器的客户端”。

13.1.7. DROP INDEX语法

```
DROP INDEX index_name ON tbl_name
```

DROP INDEX用于从表*tbl_name*中取消名称为*index_name*的索引。本语句被映射到一个ALTER TABLE语句中，用于取消索引。请参见13.1.2节，“ALTER TABLE语法”。

13.1.8. DROP TABLE语法

```
DROP [TEMPORARY] TABLE [IF EXISTS]
    tbl_name [, tbl_name] ...
    [RESTRICT | CASCADE]
```

DROP TABLE用于取消一个或多个表。您必须有每个表的DROP权限。所有的表数据和表定义会被取消，所以使用本语句要小心！

注意，对于一个带分区的表，DROP TABLE会永久性地取消表定义，取消各分区，并取消储存在这些分区中的所有数据。DROP TABLE还会取消与被取消的表有关联的分区定义（.par）文件。

对与不存在的表，使用IF EXISTS用于防止错误发生。当使用IF EXISTS时，对于每个不存在的表，会生成一个NOTE。请参见13.5.4.22节，“SHOW WARNINGS语法”。

RESTRICT和CASCADE可以使分区更容易。目前，RESTRICT和CASCADE不起作用。

注释：除非您使用TEMPORARY关键词，DROP TABLE会自动提交当前的有效的事务。

TEMPORARY关键词具有以下作用：

- 语句只取消TEMPORARY表。
- 语句不会终止正在进行中的事务。
- 不会查验存取权。（TEMPORARY表仅对于创建该表的客户端是可见的，所以查验是不必要的。）

使用TEMPORARY是确保您不会意外取消一个非TEMPORARY表的良好方法。

13.1.9. RENAME TABLE语法

```
RENAME TABLE tbl_name TO new_tbl_name
    [, tbl_name2 TO new_tbl_name2] ...
```

本语句用于对一个或多个表进行重命名。

重命名操作自动进行，这意味着当重命名正在运行时，其它线程不能读取任何表。例如，如果您有一个原有的表old_table，您可以创建另一个具有相同结构的空表new_table，然后用此空表替换原有的表：

```
CREATE TABLE new_table (...);
RENAME TABLE old_table TO backup_table, new_table TO old_table;
```

如果此语句用于对多个表进行重命名，则重命名操作从左至右进行。如果您想要交换两个表的名称，您可以这样做（假设不存在名称为tmp_table的表）：

```
RENAME TABLE old_table TO tmp_table,
             new_table TO old_table,
             tmp_table TO new_table;
```

只要两个数据库位于同一文件系统中，您还可以对表进行重命名，把表从一个数据库中移动到另一个数据库中：

```
RENAME TABLE current_db.tbl_name TO other_db.tbl_name;
```

当您执行RENAME时，您不能有被锁定的表，也不能有处于活性状态的事务。您还必须拥有原表的ALTER和DROP权限，以及新表的CREATE和INSERT权限。

如果MySQL对多个表进行重命名时遇到了错误，MySQL会对所有已被重命名的表进行反向重命名，返回到原来的状态。

只要您不尝试通过重命名把视图加入另一个数据库中，则RENAME TABLE也可以用于视图。

13.2. 数据操作语句

13.2.1. DELETE语法
13.2.2. DO语法
13.2.3. HANDLER语法
13.2.4. INSERT语法
13.2.5. LOAD DATA INFILE语法
13.2.6. REPLACE语法

13.2.7. SELECT语法
13.2.8. Subquery语法
13.2.9. TRUNCATE语法
13.2.10. UPDATE语法

13.2.1. DELETE语法

单表语法:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
    [WHERE where_definition]
    [ORDER BY ...]
    [LIMIT row_count]
```

多表语法:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    tbl_name[.*] [, tbl_name[.*] ...]
    FROM table_references
    [WHERE where_definition]
```

或:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    FROM tbl_name[.*] [, tbl_name[.*] ...]
    USING table_references
    [WHERE where_definition]
```

*tbl_name*中有些行满足由*where_definition*给定的条件。DELETE用于删除这些行，并返回被删除的记录的数目。

如果您编写的DELETE语句中没有WHERE子句，则所有的行都被删除。当您不想知道被删除的行的数目时，有一个更快的方法，即使用TRUNCATE TABLE。请参见13.2.9节，“TRUNCATE语法”。

如果您删除的行中包括用于AUTO_INCREMENT列的最大值，则该值被重新用于BDB表，但是不会被用于MyISAM表或InnoDB表。如果您在AUTOCOMMIT模式下使用DELETE FROM *tbl_name*（不含WHERE子句）删除表中的所有行，则对于所有的表类型（除InnoDB和MyISAM外），序列重新编排。对于InnoDB表，此项操作有一些例外，在15.2.6.3节，“AUTO_INCREMENT列如何在InnoDB中运行”中进行了讨论。

对于MyISAM和BDB表，您可以把AUTO_INCREMENT次级列指定到一个多列关键字中。在这种情况下，从序列的顶端被删除的值被再次使用，甚至对于MyISAM表也如此。请参见3.6.9节，“使用AUTO_INCREMENT”。

DELETE语句支持以下修饰符:

- 如果您指定LOW_PRIORITY，则DELETE的执行被延迟，直到没有其它客户端读取本表时再执行。

- 对于MyISAM表，如果您使用QUICK关键词，则在删除过程中，存储引擎不会合并索引端结点，这样可以加快部分种类的删除操作的速度。

- 在删除行的过程中，IGNORE关键词会使MySQL忽略所有的错误。（在分析阶段遇到的错误会以常规方式处理。）由于使用本选项而被忽略的错误会作为警告返回。

删除操作的速度会受到一些因素的影响，这些因素在7.2.18节，“DELETE语句的速度”中进行了讨论。

在MyISAM表中，被删除的记录被保留在一个带链接的清单中，后续的INSERT操作会重新使用旧的记录位置。要重新使用未使用的空间并减小文件的尺寸，则使用OPTIMIZE TABLE语句或**myisamchk**应用程序重新编排表。OPTIMIZE TABLE更简便，但是**myisamchk**速度更快。请参见13.5.2.5节，“OPTIMIZE TABLE语法”和第7章：优化。

QUICK修饰符会影响到在删除操作中索引端结点是否合并。当用于被删除的行的索引值被来自后插入的行的相近的索引值代替时，DELETE QUICK最为适用。在此情况下，被删除的值留下来的空穴被重新使用。

未充满的索引块跨越某一个范围的索引值，会再次发生新的插入。当被删除的值导致出现未充满的索引块时，DELETE QUICK没有作用。在此情况下，使用QUICK会导致未利用的索引中出现废弃空间。下面是此种情况的举例说明：

1. 创建一个表，表中包含已编索引的AUTO_INCREMENT列。
2. 在表中插入很多记录。每次插入会产生一个索引值，此索引值被添加到索引的高端处。
3. 使用DELETE QUICK从列的低端处删除一组记录。

在此情况下，与被删除的索引值相关的索引块变成未充满的状态，但是，由于使用了QUICK，这些索引块不会与其它索引块合并。当插入新值时，这些索引块仍为未充满的状态，原因是新记录不含有在被删除的范围内的索引值。另外，即使您此后使用DELETE时不包含QUICK，这些索引块也仍是未充满的，除非被删除的索引值中有一部分碰巧位于这些未充满的块的之中，或与这些块相邻。在这些情况下，如果要重新利用未使用的索引空间，需使用OPTIMIZE TABLE。

如果您打算从一个表中删除许多行，使用DELETE QUICK再加上OPTIMIZE TABLE可以加快速度。这样做可以重新建立索引，而不是进行大量的索引块合并操作。

用于DELETE的MySQL唯一的LIMIT *row_count*选项用于告知服务器在控制命令被返回到客户端前被删除的行的最大值。本选项用于确保一个DELETE语句不会占用过多的时间。您可以只重复DELETE语句，直到相关行的数目少于LIMIT值为止。

如果DELETE语句包括一个ORDER BY子句，则各行按照子句中指定的顺序进行删除。此子句只在与LIMIT联用是才起作用。例如，以下子句用于查找与WHERE子句对应的行，使用timestamp_column进行分类，并删除第一（最旧的）行：

```
DELETE FROM somelog

WHERE user = 'jcole'

ORDER BY timestamp_column

LIMIT 1;
```

您可以在一个DELETE语句中指定多个表，根据多个表中的特定条件，从一个表或多个表中删除行。不过，您不能在一个多表DELETE语句中使用ORDER BY或LIMIT。

*table_references*部分列出了包含在联合中的表。此语法在13.2.7.1节，“JOIN语法”中进行了说明。

对于第一个语法，只删除列于FROM子句之前的表中的对应的行。对于第二个语法，只删除列于FROM子句之中（在USING子句之前）的表中的对应的行。作用是，您可以同时删除许多个表中的行，并使用其它的表进行搜索：

```
DELETE t1, t2 FROM t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.id;
```

或：

```
DELETE FROM t1, t2 USING t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.id;
```

当搜索待删除的行时，这些语句使用所有三个表，但是只从表t1和表t2中删除对应的行。

以上例子显示了使用逗号操作符的内部联合，但是多表DELETE语句可以使用SELECT语句中允许的所有类型的联合，比如LEFT JOIN。

本语法允许在名称后面加.*，以便与Access相容。

如果您使用的多表DELETE语句包括InnoDB表，并且这些表受外键的限制，则MySQL优化程序会对表进行处理，改变原来的从属关系。在这种情况下，该语句出现错误并返回到前面的步骤。要避免此错误，您应该从单一表中删除，并依靠InnoDB提供的ON DELETE功能，对其它表进行相应的修改。

注释：当引用表名称时，您必须使用别名（如果已给定）：

```
DELETE t1 FROM test AS t1, test2 WHERE ...
```

进行多表删除时支持跨数据库删除，但是在此情况下，您在引用表时不能使用别名。举例说明：

```
DELETE test1.tmp1, test2.tmp2 FROM test1.tmp1, test2.tmp2 WHERE ...
```

目前，您不能从一个表中删除，同时又在子查询中从同一个表中选择。

13.2.2. DO语法

```
DO expr [, expr] ...
```

DO用于执行表达式，但是不返回任何结果。DO是SELECT *expr*的简化表达方式。DO有一个优势，就是如果您不太关心结果的话，DO的速度稍快。

DO主要用于执行有副作用的函数，比如RELEASE_LOCK()。

13.2.3. HANDLER语法

```
HANDLER tbl_name OPEN [ AS alias ]
HANDLER tbl_name READ index_name { = | >= | <= | < } (value1,value2,...)
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST }
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ { FIRST | NEXT }
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name CLOSE
```

HANDLER语句提供通往表存储引擎接口的直接通道。HANDLER可以用于MyISAM和InnoDB表。

HANDLER...OPEN语句用于打开一个表，通过后续的HANDLER...READ语句建立读取表的通道。本表目标不会被其它线程共享，也不会关闭，直到线程调用HANDLER...CLOSE或线程中止时为止。如果您使用一个别名打开表，则使用其它HANDLER语句进一步参阅表是必须使用此别名，而不能使用表名。

如果被指定的索引满足给定的值并且符合了WHERE条件，则第一个HANDLER...READ语法取出一行。如果您有一个多列索引，则指定索引列值为一个用逗号隔开的清单。既可以为索引中的所有列指定值，也可以为索引列的最左边的前缀指定值。假设一个索引包括三个列，名称为col_a, col_b,和col_c，并按此顺序排列。HANDLER语句可以为索引中的所有三个列指定值，或者为一个最左边前缀中的各列指定值。举例说明：

```
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) ...
```

第二个HANDLER...READ语法按索引的顺序从表中取出一行。索引的顺序符合WHERE条件。

第三个HANDLER...READ语法按自然行的顺序从表中取出一行。自然行的顺序符合WHERE条件。当想要对整个表进行扫描时，此语句比HANDLER *tbl_name* READ index_name更快。自然行的顺序指的是行存储在MyISAM表数据文件的顺序。本语句也适用于InnoDB表，但是因为没有独立的数据文件，所以没有这类概念。

不使用LIMIT子句时，所有形式的HANDLER...READ语句均只取出一行。 如果要返回多个行，应加入一个LIMIT子句。本语句于SELECT语句的语法一样。请参见13.2.7节，“SELECT语法”。

HANDLER...CLOSE用于关闭使用HANDLER...OPEN打开的表。

注释：要使用HANDLER接口来查阅一个表的PRIMARY KEY，应使用带引号的识别符`PRIMARY`：

```
HANDLER tbl_name READ `PRIMARY` > (...);
```

HANDLER是比较低级别的语句。例如，它不能提供一致性。也就是说，HANDLER...OPEN不能为表做快照，也不能锁定表。这意味着，当一个HANDLER...OPEN语句被编写后，表数据可以被更改（用此线程或用其它线程），并且这些更改只会部分地出现在HANDLER...NEXT或HANDLER...PREV扫描中。

使用HANDLER接口代替常规的SELECT语句有多个原因：

· HANDLER比SELECT更快：

o 一个指定的存储引擎管理程序目标为了HANDLER...OPEN进行整序。该目标被重新用于该表的后续的HANDLER语句；不需要对每个语句进行重新初始化。

o 涉及的分析较少。

o 没有优化程序或查询校验开销。

o 在两个管理程序请求之间，不需要锁定表。

o 管理程序接口不需要提供外观一致的数据（例如，允许无条理的读取），所以存储引擎可以使用优化，而SELECT通常不允许使用优化。

· 有些应用程序使用与ISAM近似的接口与MySQL连接。使用HANDLER可以更容易地与这些应用程序连接。

· HANDLER允许您采用一种特殊的方式进出数据库。而使用SELECT时难以采用（或不可能采用）这种方式。有些应用程序可以提供一个交互式的用户接口与数据库连接。当与这些应用程序同时使用时，用HANDLER接口观看数据更加自然。

13.2.4. INSERT语法

13.2.4.1. INSERT ... SELECT语法
13.2.4.2. INSERT DELAYED语法

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    VALUES ({expr | DEFAULT},...),(...),...
    [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

或:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    SET col_name={expr | DEFAULT}, ...
    [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

或:

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

INSERT用于向一个已有的表中插入新行。INSERT...VALUES和INSERT...SET形式的语句根据明确指定的值插入行。INSERT...SELECT形式的语句插入从其它表中选出的行。在13.2.4.1节，“INSERT ... SELECT语法”中对INSERT...SELECT进行了进一步的讨论。

行应被插入到*tbl_name*表中。可以按以下方法指定列。本语句向这些列提供值。

- 列名称清单或SET子句明确的指示了列。

- 如果您不为INSERT...VALUES或INSERT...SELECT指定列的清单，则表中每列的值必须在VALUES清单中提供，或由SELECT提供。如果您不知道表中各列的顺序，则使用DESCRIBE *tbl_name*查询。

列值可以采用多种方法给定:

- 如果不是在严格模式下运行，则所有没有明确给定值的列都被设置为默认值（明确的或隐含的）。例如，如果您指定了一个列清单，但此清单没有对表中所有的列进行命名，则未命名的各列被设置为默认值。默认值的赋值在13.1.5节，“CREATE TABLE语法”中进行了说明。也可参见1.8.6.2节，“对无效数据的约束”。

有时候，您需要对所有没有默认值的列明确地指定值。如果您希望，在没有明确指定值时，INSERT语句可以生成错误信息，则您应该使用STRICT模式。请参见5.3.2节，“SQL服务器模式”。

- 使用关键词DEFAULT，明确地把列设置为默认值。这样，编写向所有列赋值的INSERT语句时可以更容易，因为使用DEFAULT可以避免编写出不完整的、未包含全部列值的VALUES清单。如果不使用DEFUALT，您必须编写一个列名称清单，与VALUES清单中的每个值对应。

您还可以使用DEFAULT(*col_name*)作为一种更通用的形式，在表达式中使用，用于生成一个列的默认值。

- 如果列清单和VALUES清单均为空清单，则INSERT会创建一个行，每个列都被设置为默认值:

-
```
mysql> INSERT INTO tbl_name () VALUES();
```

在STRICT模式中，如果有一列没有默认值，则会出现错误。或者，MySQL会对所有没有明确定义默认值的列使用隐含的默认值。

- 您可以指定一个表达式*expr*来提供一个列值。如果表达式的类型与列值不匹配，这样做会造成类型转化。并且，给定值的转化会导致不同的插入值，插入何值由列类型而定。例如，向一个INT, FLOAT, DECIMAL(10,6)或YEAR列插入字符串'1999.0e-2'，插入值分别是1999，19.9921，19.992100和1999。存储在INT和YEAR列中的值为1999的原因是，在从字符串到整数的转化中，只把字符串的前面部分看作有效的整数或年份。对于浮点列和固定点列，在从字符串到浮点的转化中，把整个字符串均看作有效的浮点值。

表达式*expr*可以引用在值清单中已设置的所有列。例如，您可以这么操作，因为用于col2的值引用了col1，

而col1已经被赋值：

```
mysql> INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
```

但是以下语句不合法，因为用于col1的值引用了col2，而col2在col1之后被赋值：

```
mysql> INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
```

有一种例外情况，那就是含有AUTO_INCREMENT值的列。因为AUTO_INCREMENT值在其它值赋值之后被生成，所以任何在赋值时对AUTO_INCREMENT列的引用都会返回0。

INSERT语句支持下列修改符：

- 如果您使用DELAYED关键字，则服务器会把待插入的行放到一个缓冲器中，而发送INSERT DELAYED语句的客户端会继续运行。如果表正在被使用，则服务器会保留这些行。当表空闲时，服务器开始插入行，并定期检查是否有新的读取请求。如果有新的读取请求，则被延迟的行被延缓执行，直到表再次空闲时为止。请参见13.2.4.2节，“INSERT DELAYED语法”。

- 如果您使用LOW_PRIORITY关键词，则INSERT的执行被延迟，直到没有其它客户端从表中读取为止。当原有客户端正在读取时，有些客户端刚开始读取。这些客户端也被包括在内。此时，INSERT LOW_PRIORITY语句等候。因此，在读取量很大的情况下，发出INSERT LOW_PRIORITY语句的客户端有可能需要等待很长一段时间（甚至是永远等待下去）。（这与INSERT DELAYED形成对比，INSERT DELAYED立刻让客户端继续执行。请参见13.2.4.2节，“INSERT DELAYED语法”。）注意LOW_PRIORITY通常不应用于MyISAM表，因为这么做会取消同时进行的插入。请参见15.1节，“MyISAM存储引擎”。

- 如果您指定了HIGH_PRIORITY，同时服务器采用--low-priority-updates选项启动，则HIGH_PRIORITY将覆盖--low-priority-updates选项。这么做还会导致同时进行的插入被取消。

- 使用mysql_affected_rows() C API函数，可以获得用于INSERT的受影响行的值。请参见25.2.3.1节，“mysql_affected_rows()”。

- 如果您在一个INSERT语句中使用IGNORE关键词，在执行语句时出现的错误被当作警告处理。例如，没有使用IGNORE时，如果一个行复制了原有的UNIQUE索引或PRIMARY KEY值，会导致出现重复关键字错误，语句执行失败。使用IGNORE时，该行仍然未被插入，但是不会出现错误。IGNORE未被指定时，如果数据转化引发错误，则会使语句执行失败。使用IGNORE后，无效数据被调整到最接近的值，并被插入；此时，生成警告，但是语句执行不会失败。您可以使用mysql_info() C API函数测定有多少行被插入到表中。

如果您指定了ON DUPLICATE KEY UPDATE，并且插入行后会导致在一个UNIQUE索引或PRIMARY KEY中出现重复值，则执行旧行UPDATE。例如，如果列a被定义为UNIQUE，并且包含值1，则以下两个语句具有相同的效果：

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3)
    -> ON DUPLICATE KEY UPDATE c=c+1;


mysql> UPDATE table SET c=c+1 WHERE a=1;
```

如果行作为新记录被插入，则受影响行的值为1；如果原有的记录被更新，则受影响行的值为2。

注释：如果列b也是唯一列，则INSERT与此UPDATE语句相当：

```
mysql> UPDATE table SET c=c+1 WHERE a=1 OR b=2 LIMIT 1;
```

如果a=1 OR b=2与多个行向匹配，则只有一个行被更新。通常，您应该尽量避免对带有多个唯一关键字的表使用ON DUPLICATE KEY子句。

您可以在UPDATE子句中使用VALUES(col_name)函数从INSERT...UPDATE语句的INSERT部分引用列值。换句话说，如果没有发生重复关键字冲突，则UPDATE子句中的VALUES(*col_name*)可以引用被插入的*col_name*的值。本函数特别适用于多行插入。VALUES()函数只在INSERT...UPDATE语句中有意义，其它时候会返回NULL。

示例：

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
    -> ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

本语句与以下两个语句作用相同：

```
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;
```

当您使用ON DUPLICATE KEY UPDATE时，DELAYED选项被忽略。

您可以使用SQL LAST_INSERT_ID()函数查找用于AUTO_INCREMENT列的值。从C API的内部，使用mysql_insert_id()函数。不过，您应该注意，两个函数的作用并不总是相同的。在12.9.3节，“信息函数”和25.2.3.36节，“mysql_insert_id()”中进一步讨论了与AUTO_INCREMENT列有关的INSERT语句的作用。

如果您使用INSERT...VALUES语句时采用了多个值清单或INSERT...SELECT，则该语句按以下格式返回一个信息字符串：

```
Records: 100 Duplicates: 0 Warnings: 0
```

记录指示了经过语句处理的行的数目。（因为重复数目可以不是零，所以该数目不一定是实际被插入的行的数目。）重复数目指的是不能被插入的行的数目，因为这些行会复制部分原有的唯一索引值。警告指的是插入有错误或有问题的列值的次数。在以下情况下会出现警告：

- 向一个已定义为NOT NULL的列中插入NULL。对于一个多行INSERT语句或INSERT INTO...SELECT语句，根据列数据的类型，列被设置为隐含的默认值。对于数字类型，默认值为0；对于字符串类型，默认值为空字符串(')；对于日期和时间类型，默认值为“zero”值。对INSERT INTO...SELECT语句的处理方法与对多行插入的处理方法一样，因为服务器不能检测来自SELECT的结果，不能判断是否返回单一行。（对于单一行INSERT，当NULL被插入一个NOT NULL列时，不会出现警告，而是出现错误，并且语句运行失败。）

- 数字列的值被设置在列的值范围之外。此值被修改为未最接近的值范围端点。

- 向一个数字列赋予一个例如'10.34 a'的值。尾部的非数字文本被删节，其余的数字部分被插入，如果字符串值没有前导的数字部分，则该列被设置为0。

- 向一个字符串列（CHAR, VARCHAR, TEXT或BLOB）中插入的字符串超过了列的最大长度。此值被删节到列的最大长度。

- 向日期或时间列中插入的值对于该列的类型是不合法的。根据列的类型，该列被设置到相应的零值。

如果您正在使用C API，则可以通过调用mysql_info()函数获取信息字符串。请参见25.2.3.34节，“mysql_info()”。

13.2.4.1. INSERT ... SELECT语法

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

使用INSERT...SELECT，您可以快速地从一个或多个表中向一个表中插入多个行。

示例:

```
INSERT INTO tbl_temp2 (fld_id)
    SELECT tbl_temp1.fld_order_id
    FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

使用INSERT...SELECT语句时会出现以下情况:

- 明确地指定IGNORE，用于忽略会导致重复关键字错误的记录。
- 不要同时使用DELAYED和INSERT...SELECT。
- INSERT语句的目标表会显示在查询的SELECT部分的FROM子句中。（在有些旧版本的MySQL中不会出现这种情况。）
- AUTO_INCREMENT列照常运行。
- 为了确保二进制日志可以被用于再次创建原表，MySQL不允许在INSERT...SELECT运行期间同时进行插入操作。
- 目前，您不能在向一个表插入的同时，又在一个子查询中从同一个表中选择。

在ON DUPLICATE KEY UPDATE的值部分中，只要您不使用SELECT部分中的GROUP BY，您就可以引用在其它表中的列。有一个副作用是，您必须使值部分中的非唯一列的名称符合要求。

您可以使用REPLACE替代INSERT，来覆盖旧行。对于包含唯一关键字值，并复制了旧行的新行，在进行处理时，REPLACE可以作为INSERT IGNORE的同类子句：新行被用于替换旧行，而不是被丢弃。

13.2.4.2. INSERT DELAYED语法

```
INSERT DELAYED ...
```

用于INSERT语句的DELAYED选项是MySQL相对于标准SQL的扩展。如果您的客户端不能等待INSERT完成，则这个选项是非常有用的。当您使用MySQL进行日志编写时，这是非常常见的问题。您也可以定期运行SELECT和UPDATE语句，这些语句花费的时间较长。

当一个客户端使用INSERT DELAYED时，会立刻从服务器处得到一个确定。并且行被排入队列，当表没有被其它线程使用时，此行被插入。

使用INSERT DELAYED的另一个重要的好处是，来自许多客户端的插入被集中在一起，并被编写入一个块。这比执行许多独立的插入要快很多。

使用DELAYED时有一些限制:

- INSERT DELAYED仅适用于MyISAM, MEMORY和ARCHIVE表。对于MyISAM表，如果在数据文件的中间没有空闲的块，则支持同时采用SELECT和INSERT语句。在这些情况下，基本不需要对MyISAM使用INSERT

DELAYED。请参见15.1节，“MyISAM存储引擎”, 15.4节，“MEMORY (HEAP)存储引擎”和15.8节，“ARCHIVE存储引擎”。

· INSERT DELAYED应该仅用于指定值清单的INSERT语句。服务器忽略用于INSERT DELAYED...SELECT语句的DELAYED。

· 服务器忽略用于INSERT DELAYED...ON DUPLICATE UPDATE语句的DELAYED。

· 因为在行被插入前，语句立刻返回，所以您不能使用LAST_INSERT_ID()来获取AUTO_INCREMENT值。AUTO_INCREMENT值可能由语句生成。

· 对于SELECT语句，DELAYED行不可见，直到这些行确实被插入了为止。

· DELAYED在从属复制服务器中被忽略了，因为DELAYED不会在从属服务器中产生与主服务器不一样的数据。

注意，目前在队列中的各行只保存在存储器中，直到它们被插入到表中为止。这意味着，如果您强行中止了**mysqld**（例如，使用kill -9）或者如果**mysqld**意外停止，则所有没有被写入磁盘的行都会丢失。

以下详细描述了当您对INSERT或REPLACE使用DELAYED选项时会发生什么情况。在这些描述中，“线程”指的是已接受了一个INSERT DELAYED语句的线程，“管理程序”指的是为某个特定的表处理所有INSERT DELAYED语句的线程。

· 当一个线程对一个表执行DELAYED语句时，会创建出一个管理程序线程（如果原来不存在），对用于本表的所有DELAYED语句进行处理。

· 线程会检查是否管理程序以前已获取了DELAYED锁定；如果没有获取，则告知管理程序线程进行此项操作。即使其它线程对表有READ或WRITE锁定，也可以获得DELAYED锁定。但是管理程序会等待所有的ALTER TABLE锁定或FLUSH TABLE锁定，以确保表的结构是最新的。

· 线程执行INSERT语句，但不是把行写入表中，而是把最终行的拷贝放入一个由管理程序线程管理的队列中。线程会提示出现语法错误，这些错误会被报告到客户端中。

· 因为在插入操作之前，INSERT返回已经完成，所以客户端不能从服务器处获取重复记录的数目，也不能获取生成的行的AUTO_INCREMENT值。（如果您使用C API，则出于同样的原因，mysql_info()函数不会返回任何有意义的东西。）

· 当行被插入表中时，二进制日志被管理程序线程更新。在多行插入情况下，当第一行被插入时，二进制日志被更新。

· 每次delayed_insert_limit行被编写时，管理程序会检查是否有SELECT语句仍然未执行。如果有，则会在继续运行前，让这些语句先执行。

· 当管理程序的队列中没有多余的行时，表被解锁。如果在delayed_insert_timeout时间内，没有接收到新的INSERT DELAYED语句，则管理程序中止。

· 如果在某个特定的管理程序队列中，有超过delayed_queue_size的行未被执行，则申请INSERT DELAYED的线程会等待，直到队列中出现空间为止。这么做可以确保**mysqld**不会把所有的存储器都用于被延迟的存储队列。

· 管理程序线程会显示在MySQL进程清单中，其命令列中包含delayed_insert。如果您执行一个FLUSH TABLES语句或使用KILL thread_id进行删除，则会删除此线程。不过，在退出前，线程会首先把所有排入队列的行存储到表中。在这期间，该线程不会从其它线程处接受任何新的INSERT语句。如果您在此之后执行一个INSERT DELAYED语句，则会创建出一个新的管理程序线程。

注意，如果有一个INSERT DELAYED管理程序正在运行，则这意味着INSERT DELAYED语句比常规的INSERT语句具有更高的优先权。其它更新语句必须等待，直到INSERT DELAYED语句队列都运行完毕，或者管理程序线程被中止（使用KILL thread_id），或者执行了一个FLUSH TABLES时为止。

- 以下状态变量提供了有关INSERT DELAYED语句的信息：

状态变量	意义
Delayed_insert_threads	管理程序线程的数目
Delayed_writes	使用INSERT DELAYED写入的行的数目
Not_flushed_delayed_rows	等待被写入的行的数目

- 您可以通过发送一个SHOW STATUS语句，或者执行一个**mysqladmin extended-status**命令，来阅览这些变量。

注意，当没有使用表时，INSERT DELAYED比常规的INSERT要慢。对于服务器来说，为每个含有延迟行的表操纵一个独立的线程，也是一个额外的系统开销。这意味着只有当您确认您需要时，才应使用INSERT DELAYED。

13.2.5. LOAD DATA INFILE语法

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name.txt'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [FIELDS
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char' ]
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]
    [IGNORE number LINES]
    [(col_name_or_user_var,...)]
    [SET col_name = expr,...)]
```

LOAD DATA INFILE语句用于高速地从一个文本文件中读取行，并装入一个表中。文件名称必须为一个文字字符串。

要了解有关INSERT和LOAD DATA INFILE的效率的对比和有关LOAD DATA INFILE加速的更多信息，请参见7.2.16节，“INSERT语句的速度”。

由character_set_database系统变量指示的字符集被用于解释文件中的信息。SET NAMES和character_set_client的设置不会影响对输入的解释。

注意，目前不能载入UCS2数据文件。

您也可以通过使用**mysqlimport**应用程序载入数据文件；通过向服务器发送一个LOAD DATA INFILE语句实现此功能。--local选项用于使**mysqlimport**从客户主机中读取数据文件。如果客户端和服务器支持压缩协议，则您可以指定—compress选项提高在慢速网络中的性能。请参见8.10节，“mysqlimport：数据导入程序。

如果您使用LOW_PRIORITY，则LOAD DATA语句的执行被延迟，直到没有其它的客户端从表中读取为止。

如果一个MyISAM表满足同时插入的条件（即该表在中间有空闲块），并且您对这个MyISAM表指定了CONCURRENT，则当LOAD DATA正在执行时，其它线程会从表中重新获取数据。即使没有其它线程在同时使用本表格，使用本选项也会略微影响LOAD DATA的性能。

如果指定了LOCAL，则被认为与连接的客户端有关：

- 如果指定了LOCAL，则文件会被客户主机上的客户端读取，并被发送到服务器。文件会被给予一个完整的路径名称，以指定确切的位置。如果给定的是一个相对的路径名称，则此名称会被理解为相对于启动客户端时所在的目录。

- 如果LOCAL没有被指定，则文件必须位于服务器主机上，并且被服务器直接读取。

当在服务器主机上为文件定位时，服务器使用以下规则：

- 如果给定了一个绝对的路径名称，则服务器使用此路径名称。

- 如果给定了带有一个或多个引导组件的相对路径名称，则服务器会搜索相对于服务器数据目录的文件。

- 如果给定了一个不带引导组件的文件名称，则服务器会在默认数据库的数据库目录中寻找文件。

注意，这些规则意味着名为./myfile.txt的文件会从服务器数据目录中被读取，而名为myfile.txt的同样的文件会从默认数据库的数据库目录中读取。例如，下面的LOAD DATA语句会从db1数据库目录中读取文件data.txt，因为db1是当前数据库。即使语句明确把文件载入到db2数据库中的表里，也会从db1目录中读取。

```
mysql> USE db1;

mysql> LOAD DATA INFILE 'data.txt' INTO TABLE db2.my_table;
```

注意，使用正斜杠指定Windows路径名称，而不是使用反斜杠。如果您使用反斜杠，您必须使用两个。

出于安全原因，当读取位于服务器中的文本文件时，文件必须位于数据库目录中，或者是全体可读的。另外，要对服务器文件使用LOAD DATA INFILE，您必须拥有FILE权限。

见5.7.3节，“MySQL提供的权限”。

与让服务器直接读取文件相比，使用LOCAL速度略慢，这是因为文件的内容必须通过客户端发送到服务器上。不过，您不需要FILE权限来载入本地文件。

只有当您的服务器和您的客户端都许可时，LOCAL才可运行。例如，如果使用—local-infile=0启动**mysqld**，则LOCAL不运行。请参见5.6.4节，“LOAD DATA LOCAL安全问题”。

如果您需要LOAD DATA来从一个管道中读取，您可以使用以下方法（此处我们把/目录清单载入一个表格）：

```
mkfifo /mysql/db/x/x

chmod 666 /mysql/db/x/x

find / -ls > /mysql/db/x/x

mysql -e "LOAD DATA INFILE 'x' INTO TABLE x" x
```

有些输入记录把原有的记录复制到唯一关键字值上。REPLACE和IGNORE关键字用于控制这些输入记录的操作。

如果您指定了REPLACE，则输入行会替换原有行（换句话说，与原有行一样，对一个主索引或唯一索引具有相同值的行）。请参见13.2.6节，“REPLACE语法”。

如果您指定IGNORE，则把原有行复制到唯一关键字值的输入行被跳过。如果您这两个选项都不指定，则运行情况根据LOCAL关键词是否被指定而定。不使用LOCAL时，当出现重复关键字值时，会发生错误，并且剩下的文本文件被忽略。使用LOCAL时，默认的运行情况和IGNORE被指定时的情况相同；这是因为在运行中间，服务器没有办法中止文件的传输。

如果您希望在载入运行过程中忽略外键的限制，您可以在执行LOAD DATA前发送一个SET FOREIGN_KEY_CHECKS=0语句。

如果您对一个空的MyISAM表使用LOAD DATA INFILE，则所有的非唯一索引会被创建在一个独立批中（对于REPAIR TABLE）。当您有许多索引时，这通常会使LOAD DATA INFILE大大加快。通常，LOAD DATA INFILE的速度会非常快，但是在某些极端情况下，您可以在把文件载入到表中之前使用ALTER TABLE...DISABLE KEYS关闭LOAD DATA INFILE，或者在载入文件之后使用ALTER TABLE...ENABLE KEYS再次创建索引，使创建索引的速度更快。请参见7.2.16节，“INSERT语句的速度”。

LOAD DATA INFILE是SELECT...INTO OUTFILE的补语。（见13.2.7节，“SELECT语法”。）要从一个表中把数据写入一个文件中，应使用SELECT...INTO OUTFILE。要读取文件，放回到表中，应使用LOAD DATA INFILE。FIELDS和LINES子句的语法对于两个语句是一样的。两个子句都是自选的，但是如果两个都被指定了，FIELDS必须位于LINES的前面。

如果您指定了一个FIELDS子句，则每个亚子句（TERMINATED BY, [OPTIONALLY] ENCLOSED BY和ESCAPED BY）也是自选的。不过，您必须指定其中至少一个。

如果您不指定FIELDS子句，则默认值为假设您写下如下语句时的值：

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
```

如果您不指定LINES子句，则默认值为假设您写下如下语句时的值：

```
LINES TERMINATED BY '\n' STARTING BY ''
```

换句话说，当读取输入值时，默认值会使LOAD DATA INFILE按如下方式运行：

- 在新行处寻找行的边界。
- 不会跳过任何行前缀。
- 在制表符处把行分解为字段。
- 不希望字段被包含在任何引号字符之中。
- 出现制表符、新行、或在‘\’前有‘\’时，理解为作为字段值一部分的文字字符。

相反的，当编写输出值时，默认值会使SELECT...INTO OUTFILE按如下方式运行：

- 在字段之间写入制表符。
- 不把字段包含在任何引号字符中。
- 当字段值中出现制表符、新行或‘\’时，使用‘\’进行转义。
- 在行的末端写入新行。

注意，要写入FIELDS ESCAPED BY ‘\\’，您必须为待读取的值指定两个反斜杠，作为一个单反斜杠使用。

注释：如果您已经在Windows系统中生成了文本文件，您可能必须使用LINES TERMINATED BY '\r\n'来正确地读取文件，因为Windows程序通常使用两个字符作为一个行终止符。部分程序，比如**WordPad**，当编写文件时，可能会使用\r作为行终止符。要读取这样的文件，应使用LINES TERMINATED BY '\r'。

如果所有您希望读入的行都含有一个您希望忽略的共用前缀，则您可以使用'*prefix_string*'来跳过前缀（和前缀前的字符）。如果某行不包括前缀，则整个行被跳过。注释：*prefix_string*会出现在一行的中间。

示例：

```
mysql> LOAD DATA INFILE '/tmp/test.txt'
    -> INTO TABLE test LINES STARTING BY "xxx";
```

使用此语句，您可以读入包含有如下内容的文件：

```
xxx"row",1
something xxx"row",2
```

并只得到数据("row",1)和("row",2)。

IGNORE *number* LINES选项可以被用于在文件的开始处忽略行。例如，您可以使用IGNORE 1 LINES来跳过一个包含列名称的起始标题行：

```
mysql> LOAD DATA INFILE '/tmp/test.txt'
    -> INTO TABLE test IGNORE 1 LINES;
```

当您联合使用SELECT...INTO OUTFILE和LOAD DATA INFILE来从一个数据库中把数据写入一个文件中，然后再读取文件，返回到数据库中时，用于两个语句的field-和line-handling选项必须匹配。否则，LOAD DATA INFILE不会正确地理解文件的内容。假设您使用SELECT...INTO OUTFILE来编写一个的文件，字段由逗号分隔：

```
mysql> SELECT * INTO OUTFILE 'data.txt'
    ->          FIELDS TERMINATED BY ','
    ->          FROM table2;
```

要读取由逗号分隔的文件并返回，则正确的语句应该是：

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE table2
    ->           FIELDS TERMINATED BY ',';
```

如果您尝试使用以下所示的语句读入文件，则不会运行，因为该语句命令LOAD DATA INFILE寻找位于字段之间的制表符：

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE table2
    ->           FIELDS TERMINATED BY '\t';
```

结果很可能是，每个输入行被理解为一个单一字段。

LOAD DATA INFILE也可以被用于读取从外源中获取的文件。例如，一个dBASE格式的文件具有以逗号分隔并且包含在双引号中的字段。如果文件中的各行以新行为结尾，则此处所示的语句描述了您可以用于载入文件的field-和line-handling选项：

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE tbl_name
    ->           FIELDS TERMINATED BY ',' ENCLOSED BY '"'
```

```
    ->          LINES TERMINATED BY '\n';
```

所有field-或line-handling选项都可以指定一个空字符串("")。如果字符串不是空的，则FIELDS [OPTIONALLY] ENCLOSED BY和FIELDS ESCAPED BY值必须为单一字符。FIELDS TERMINATED BY, LINES STARTING BY和LINES TERMINATED BY值可以超过一个字符。例如，要编写由回车/换行成对字符作为结尾的行，或读取包含这类行的文件，则应指定一个LINES TERMINATED BY '\r\n'子句。

如果jokes被由%%组成的行分隔，要读取包含jokes的文件，您可以这么操作：

```
mysql> CREATE TABLE jokes
    ->     (a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->     joke TEXT NOT NULL);
mysql> LOAD DATA INFILE '/tmp/jokes.txt' INTO TABLE jokes
    ->     FIELDS TERMINATED BY ''
    ->     LINES TERMINATED BY '\n%%\n' (joke);
```

FIELDS [OPTIONALLY] ENCLOSED BY用于控制字段的引号。对于（SELECT...INTO OUTFILE），如果您忽略了词语OPTIONALLY，则所有的字段都被包含在ENCLOSED BY字符串中。此处展示了此类输出的一个示例（使用逗号作为字段分隔符）：

```
"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"
```

如果您指定了OPTINALLY，则ENCLOSED BY字符只被用于包含具有字符串数据类型（比如CHAR, BINARY, TEXT或ENUM）的列中的值：

```
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，如果在字段值内出现ENCLOSED BY字符，则通过使用ESCAPED BY字符作为前缀，对ENCLOSED BY字符进行转义。另外，要注意，如果您指定了一个空的ESCAPED BY值，则可能会生成不能被LOAD DATA INFILE正确读取的输出值。例如，如果转义符为空字符，则刚显示的先前输出值应显示如下。请观察，第四行中的第二个字段在引号后面包含一个逗号，该引号（错误地）显示出来，作为字段的结尾：

```
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
```

对于输入值，ENCLOSED BY字符被从字段字的末尾剥离。（不论OPTIONALLY是否被指定都会剥离；OPTIONALLY对输入值的解释没有影响。）如果ENCLOSED BY字符前面带有ESCAPED BY字符，则被理解为当前字段值的一部分。

如果字段以ENCLOSED BY字符为开始，当出现这类字符时，只有后面接着字段或行TERMINATED BY序列时，这类字符被认为是一个字段值的结尾。为了避免意思不明确，当在一个字段值中出现ENCLOSED BY字符时，此字符可以重复书写，并被理解为单一的字符。例如，如果指定了ENCLOSED BY ""，则按照以下方法操作引号：

```
"The ""BIG"" boss"  -> The "BIG" boss

The "BIG" boss      -> The "BIG" boss

The ""BIG"" boss    -> The ""BIG"" boss
```

FIELDS ESCAPED BY用于控制如何写入或读取特殊字符。如果FIELDS ESCAPED BY字符不是空字符，则可以在输出中用于对以下字符加前缀：

- FIELDS ESCAPED BY字符
- FIELDS [OPTIONALLY] ENCLOSED BY字符
- FIELDS TERMINATED BY和LINES TERMINATED BY值的第一个字符
- ASCII 0（在转义符之后编写的字符实际上是ASCII'0'，而不是一个值为0的字节）

如果FIELDS ESCAPED BY字符为空字符，则没有字符被转义，并且NULL被作为NULL输出，而不是\N。去指定一个空的转义符不是一个好办法，特别是如果数据的字段值包含任何刚给定的清单中的字符时，更不能这么做。

对于输入值，如果FIELDS ESCAPED BY字符不是空字符，则出现这种字符时会被剥离，然后以下字符被作为字段值的一部分。例外情况是，被转义的'0'或'N'（例如，\0或\N，此时转义符为'\'）。这些序列被理解为ASCII NUL（一个零值字节）和NULL。用于NULL处理的规则在本节的后部进行说明。

要了解有关'\'-escape语法的更多信息，请参见9.1节，“文字值”。

在特定情况下，field-和line-handling选项相互影响：

- 如果LINES TERMINATED BY是空字符串，并且FIELDS TERMINATED BY不是空字符串，则各行以FIELDS TERMINATED BY作为结尾。
- 如果FIELDS TERMINATED BY和FIELDS ENCLOSED BY值均为空值(")，则使用固定行（无分隔）格式。使用固定行格式时，在字段之间不使用分隔符（但是您仍然可以有行终止符）。列值使用列的显示宽度进行写入和读取。例如，如果某列被定义为INT(7)，则使用7字符字段写入列值。输出时，通过读取7个字符获取列值。

LINES TERMINATED BY仍然用于分隔行。如果某行不包含所有字段，则其余的各列被设置到默认值。如果您没有行终止符，您应该把终止符设置为"。在此情况下，文本文件必须包含每行的所有字段。

固定行格式也会影响NULL值的操作，这将在以后进行介绍。注意，如果您正在使用一个多字节字符集，则固定规格格式不会运行。

根据正在使用中的FIELDS和LINES选项的不同，NULL值的操作有所变化：

- 对于默认的FIELDS和LINES值，NULL被作为\N的字段值编写，用于输出；\N字段值被作为NULL读取，用于输入（假设ESCAPED BY字符为'\'）。
- 如果FIELDS ENCLOSED BY不是空值，则包含以文字词语NULL为值的字段被作为NULL值读取。这与被FIELDS ENCLOSED BY字符包围的词语NULL不同。该词语被作为字符串'NULL'读取。
- 如果FIELDS ESCAPED BY是空值，则NULL被作为词语NULL写入。
- 采用固定行格式时（当FIELDS TERMINATED BY和FIELDS ENCLOSED BY均为空值时采用），NULL被作为一个空字符串写入。注意，这会导致在被写入文件时，表中的NULL值和空字符串均无法辨别，这是因为两者都被作为空字符串写入。如果您需要在读取文件并返回时能够分辨两者，则您不应使用固定行格式。

LOAD DATA INFILE不支持有些情况:

- 固定规格行（FIELDS TERMINATED BY和FIELDS ENCLOSED BY均为空值）和BLOB或TEXT列。

- 如果您指定了一个分隔符，并且该分隔符与其它的前缀一样，则LOAD DATA INFILE不能正确地理解输入值。例如，下面的FIELDS子句会导致问题:

```
FIELDS TERMINATED BY '"' ENCLOSED BY '"'
```

- 如果FIELDS ESCAPED BY为空值，则包含FIELDS ENCLOSED BY或LINES TERMINATED BY的字段值后面再接FIELDS TERMINATED BY值会导致LOAD DATA INFILE过早地停止读取一个字段或行。出现这种情况的原因是LOAD DATA INFILE不能正确地决定字段或行值在哪里结束。

以下的例子载入了persondata表中的所有列:

```
mysql> LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

默认情况下，如果在LOAD DATA INFILE语句的末尾处没有设列清单时，则输入行预计会包含一个字段，用于表中的每个列。如果您只想载入一个表的部分列，则应指定一个列清单:

```
mysql> LOAD DATA INFILE 'persondata.txt'
    ->          INTO TABLE persondata (col1,col2,...);
```

如果输入文件中各字段的顺序与表中各列的顺序不同，您也必须指定一个列清单。否则，MySQL不能把输入字段和表中的列匹配起来。

列清单可以包含列名称或用户变量。支持SET子句。这使您可以把输入值赋予用户变量，然后在把结果赋予列之前，对这些值进行变换。

SET子句中的用户变量可以采用多种方式使用。以下例子使用数据文件中的第一列，直接用于t1.column1的值。在用户变量被用于t2.column2值之前，把第二列赋予用户变量。该变量从属于一个分割运行。

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, @var1)
  SET column2 = @var1/100;
```

SET子句可以被用于提供不是来源于输入文件的值。以下语句把column3设置为当前的日期和时间:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, column2)
  SET column3 = CURRENT_TIMESTAMP;
```

您也可以通过把输入值赋予一个用户变量，同时不把变量赋予表中的列，来丢弃此输入值:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, @dummy, column2, @dummy, column3);
```

列/变量清单和SET子句的使用受到以下限定:

- 在SET子句中的赋值应只含有位于赋值操作符的左侧的列名称。

- 您可以在SET赋值的右侧使用子查询。如果子查询可以返回一个值，并且此值将被赋予到一个列中，则此子查询只能是标量子查询。另外，您不能使用子查询从一个正在被载入的表中选择。
- 对于于列/变量清单或SET子句，被IGNORE子句忽略的行不被处理。
- 当载入采用固定行格式的数据时，不能使用用户变量，因为用户变量没有显示宽度。

当处理一个输入行时，LOAD DATA会依据列/变量清单和SET子句，把行拆分成字段，并使用值。然后，得到的行被插入表中。如果有用于表的BEFORE INSERT或AFTER INSERT触发器，则在插入行之前和插入行之后分别启动触发器。

如果一个输入行含有过多的字段，则多余的字段被忽略，并且警告的数量增加。

如果一个输入行含有的字段过少，则输入字段缺失的表中的列被设置为默认值。默认值赋值在13.1.5节，“CREATE TABLE语法”中进行了说明。

如果字段值缺失，则对一个空字段值会被按不同方式理解：

- 对于字符串类型，列被设置为空字符串。
- 对于数字类型，列被设置为0。
- 对于日期和时间类型，列被设置为该类型相应的“zero”。请参见11.3节，“日期和时间类型”。

如果您明确地把一个空字符串赋予一个INSERT或UPDATE语句中的字符串类型、数字类型或日期或时间类型，则产生的这些值相同。

只有在两种情况下TIMESTAMP列被设置为当前日期和时间。一种情况时当列有一个NULL值（也就是\N）时；另一种情况是（仅对于第一个TIMESTAMP列），当一个字段清单被指定时，TIMESTAMP列会从字段清单中被略去。

LOAD DATA INFILE把所有的输入值当作字符串，所以您不能按照使用INSERT语句的方式使用ENUM或SET列的数字值。所有的ENUM和SET值必须被指定为字符串。

当LOAD DATA INFILE语句结束时，会按以下格式返回一个信息字符串：

```
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
```

如果您正在使用C API，您可以通过调用mysql_info()函数获取有关语句的信息。请参见25.2.3.34节，“mysql_info()”。

当值通过INSERT语句被插入时或出现相同情况时，会发生警告（见13.2.4节，“INSERT语法”）。例外情况是，当输入行中字段过多或过少时，LOAD DATA INFILE也生成警告。这些警告并不存储；警告的数量只用于指示运行是否良好。

您可以使用SHOW WARNINGS来得到第一批max_error_count警告的清单，作为有关运行错误的信息。请参见13.5.4.22节，“SHOW WARNINGS语法”。

13.2.6. REPLACE语法

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name [(col_name,...)]
    VALUES ({expr | DEFAULT},...),(...),...
```

或:

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    SET col_name={expr | DEFAULT}, ...
```

或:

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
```

REPLACE的运行与INSERT很相像。只有一点除外，如果表中的一个旧记录与一个用于PRIMARY KEY或一个UNIQUE索引的新记录具有相同的值，则在新记录被插入之前，旧记录被删除。请参见13.2.4节，"INSERT语法"。

注意，除非表有一个PRIMARY KEY或UNIQUE索引，否则，使用一个REPLACE语句没有意义。该语句会与INSERT相同，因为没有索引被用于确定是否新行复制了其它的行。

所有列的值均取自在REPLACE语句中被指定的值。所有缺失的列被设置为各自的默认值，这和INSERT一样。您不能从当前行中引用值，也不能在新行中使用值。如果您使用一个例如"SET *col_name* = *col_name* + 1"的赋值，则对位于右侧的列名称的引用会被作为DEFAULT(*col_name*)处理。因此，该赋值相当于SET *col_name* = DEFAULT(*col_name*) + 1。

为了能够使用REPLACE，您必须同时拥有表的INSERT和DELETE权限。

REPLACE语句会返回一个数，来指示受影响的行的数目。该数是被删除和被插入的行数的和。如果对于一个单行REPLACE该数为1，则一行被插入，同时没有行被删除。如果该数大于1，则在新行被插入前，有一个或多个旧行被删除。如果表包含多个唯一索引，并且新行复制了在不同的唯一索引中的不同旧行的值，则有可能是一个单一行替换了多个旧行。

受影响的行数可以容易地确定是否REPLACE只添加了一行，或者是否REPLACE也替换了其它行：检查该数是否为1（添加）或更大（替换）。

如果您正在使用C API，则可以使用mysql_affected_rows()函数获得受影响的行数。

目前，您不能在一个子查询中，向一个表中更换，同时从同一个表中选择。

以下是所用算法的更详细的说明（该算法也用于LOAD DATA...REPLACE）：

1. 尝试把新行插入到表中

2. 当因为对于主键或唯一关键字出现重复关键字错误而造成插入失败时：

a. 从表中删除含有重复关键字值的冲突行

b. 再次尝试把新行插入到表中

13.2.7. SELECT语法

13.2.7.1. JOIN语法
13.2.7.2. UNION语法

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr, ...
    [INTO OUTFILE 'file_name' export_options
      | INTO DUMPFILE 'file_name']
    [FROM table_references
    [WHERE where_definition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_definition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC] , ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT用于恢复从一个或多个表中选择的行，并可以加入UNION语句和子查询。请参见13.2.7.2节，“UNION语法”和13.2.8节，“Subquery语法”。

- 每个*select_expr*都指示一个您想要恢复的列。
- *table_references*指示行从哪个表或哪些表中被恢复。在13.2.7.1节，“JOIN语法”中对该语法进行了说明。
- *where_definition*包括关键词WHERE，后面接一个表达式。该表达式指示被选择的行必须满足的条件。

有的行在计算时未引用任何表。SELECT也可以用于恢复这类行。

举例说明：

```
mysql> SELECT 1 + 1;
        -> 2
```

所有被使用的子句必须按语法说明中显示的顺序严格地排序。例如，一个HAVING子句必须位于GROUP BY子句之后，并位于ORDER BY子句之前。

- 使用AS *alias_name*可以为*select_expr*给定一个别名。此别名用作表达式的列名，可以用于GROUP BY、ORDER BY或HAVING子句。例如：

```
mysql> SELECT CONCAT(last_name,', ',first_name) AS full_name
    -> FROM mytable ORDER BY full_name;
```

在为*select_expr*给定别名时，AS关键词是自选的。前面的例子可以这样编写：

```
mysql> SELECT CONCAT(last_name,', ',first_name) full_name
    -> FROM mytable ORDER BY full_name;
```

因为AS是自选的，如果您忘记在两个*select_expr*表达式之间加逗号，则会出现一个小问题：MySQL会把第二个表达式理解为一个别名。例如，在以下语句中，columnb被作为别名对待：

```
mysql> SELECT columna columnb FROM mytable;
```

因此，使用AS明确地指定列的别名，把它作为习惯，是一个良好的操作规范。

- 在一个WHERE子句中使用列别名是不允许的，因为当执行WHERE子句时，列值可能还没有被确定。请参见A.5.4节，"与列别名有关的问题"。

- FROM *table_references*子句指示行从哪些表中被恢复。如果您命名的表多于一个，则您在进行一个联合操作。要了解有关联合语法的说明，请参见13.2.7.1节，"JOIN语法"。对于每一个被指定的表，您可以自选地指定一个别名。

```
tbl_name [[AS] alias]
    [{USE|IGNORE|FORCE} INDEX (key_list)]
```

使用USE INDEX、IGNORE INDEX、FORCE INDEX可以向优化符提示如何选择索引。这部分内容在13.2.7.1节，"JOIN语法"中进行了讨论。

您可以使用SET max_seeks_for_key=*value*作为一种替代方法，来促使MySQL优先采用关键字扫描，替代表扫描。

- 您可以把当前数据库中的一个表作为*tbl_name*（在当前数据库中）引用，或作为*db_name.tbl_name*引用，来明确地指定一个数据库。您可以把一列作为*col_name*, *tbl_name.col_name*引用或作为*db_name.tbl_name.col_name*引用。您不需要对一个列引用指定一个*tbl_name*或*db_name.tbl_name*前缀，除非此引用意义不明确。意义不明确时，要求指定明确的列引用格式。有关示例见9.2节，"数据库、表、索引、列和别名"。

- 在没有表被引用的情况下，允许您指定DUAL作为一个假的表名。

```
mysql> SELECT 1 + 1 FROM DUAL;
    -> 2
```

有些服务器要求一个FROM子句。DUAL仅用于与这些服务器兼容。如果没有表被引用，则MySQL不要求该子句，前面的语句可以按以下方法编写：

```
mysql> SELECT 1 + 1;
    -> 2
```

- 使用*tbl_name* AS *alias_name*或*tbl_name alias_name*可以为一个表引用起别名：

```
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;
```

- 在WHERE子句中，您可以使用MySQL支持的所有函数，不过总计（总结）函数除外。请参见第12章：函数和操作符。

- 被选择的用于输出的列可以使用列名称、列别名或列位置被引用到ORDER BY和GROUP BY子句中。列

位置为整数，从1开始：

```
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;
```

要以相反的顺序进行分类，应把DESC（降序）关键字添加到ORDER BY子句中的列名称中。默认值为升序；该值可以使用ASC关键词明确地指定。

不建议使用列位置，因为该语法已经从SQL标准中删除。

- 如果您使用GROUP BY，则输出行根据GROUP BY列进行分类，如同您对相同的列进行了ORDER BY。MySQL对GROUP BY进行了扩展，因此您可以在各列（在子句中进行命名）的后面指定ASC和DESC：

```
SELECT a, COUNT(b) FROM test_table GROUP BY a DESC
```

- MySQL对GROUP BY的使用进行了扩展，允许选择在GROUP BY子句中没有被提到的字段。如果您没有得到预期的结果，请阅读GROUP BY的说明，请参见12.10节，“与GROUP BY子句同时使用的函数和修改程序”。

- GROUP BY允许一个WITH ROLLUP修饰符。请参见12.10.2节，“GROUP BY修改程序”。

- HAVING子句基本上是最后使用，只位于被发送给客户端的条目之前，没有进行优化。（LIMIT用于HAVING之后。）

SQL标准要求HAVING必须引用GROUP BY子句中的列或用于总计函数中的列。不过，MySQL支持对此工作性质的扩展，并允许HAVING因为SELECT清单中的列和外部子查询中的列。

如果HAVING子句引用了一个意义不明确的列，则会出现警告。在下面的语句中，col2意义不明确，因为它既作为别名使用，又作为列名使用：

```
mysql> SELECT COUNT(col1) AS col2 FROM t GROUP BY col2 HAVING col2 = 2;
```

标准SQL工作性质具有优先权，因此如果一个HAVING列名既被用于GROUP BY，又被用作输出列清单中的起了别名的列，则优先权被给予GROUP BY列中的列。

- HAVING不能用于应被用于WHERE子句的条目。例如，不能编写如下语句：

```
mysql> SELECT col_name FROM tbl_name HAVING col_name > 0;
```

而应这么编写：

```
mysql> SELECT col_name FROM tbl_name WHERE col_name > 0;
```

- HAVING子句可以引用总计函数，而WHERE子句不能引用：

```
mysql> SELECT user, MAX(salary) FROM users
    -> GROUP BY user HAVING MAX(salary)>10;
```

（在有些较早版本的MySQL中，本语句不运行。）

- LIMIT子句可以被用于限制被SELECT语句返回的行数。LIMIT取一个或两个数字自变量，自变量必须是

非负的整数常数（当使用已预备的语句时除外）。

使用两个自变量时，第一个自变量指定返回的第一行的偏移量，第二个自变量指定返回的行数的最大值。初始行的偏移量为0（不是1）：

```
mysql> SELECT * FROM tbl LIMIT 5,10;  # Retrieve rows 6-15
```

为了与PostgreSQL兼容，MySQL也支持LIMIT *row_count* OFFSET *offset*语法。

如果要恢复从某个偏移量到结果集合的末端之间的所有的行，您可以对第二个参数是使用比较大的数。本语句可以恢复从第96行到最后的所有行：

```
mysql> SELECT * FROM tbl LIMIT 95,18446744073709551615;
```

使用1个自变量时，该值指定从结果集合的开头返回的行数：

```
mysql> SELECT * FROM tbl LIMIT 5;     # Retrieve first 5 rows
```

换句话说，LIMIT n与LIMIT 0,n等价。

对于已预备的语句，您可以使用位置保持符。以下语句将从tb1表中返回一行：

```
mysql> SET @a=1;

mysql> PREPARE STMT FROM "SELECT * FROM tbl LIMIT ?";

mysql> EXECUTE STMT USING @a;
```

以下语句将从tb1表中返回第二到第六行：

```
mysql> SET @skip=1; SET @numrows=5;

mysql> PREPARE STMT FROM "SELECT * FROM tbl LIMIT ?, ?";

mysql> EXECUTE STMT USING @skip, @numrows;
```

· SELECT...INTO OUTFILE '*file_name*'形式的SELECT可以把被选择的行写入一个文件中。该文件被创建到服务器主机上，因此您必须拥有FILE权限，才能使用此语法。*file_name*不能是一个原有的文件。原有文件会阻止例如"/etc/passwd"的文件和数据库表被销毁。

SELECT...INTO OUTFILE语句的主要作用是让您可以非常快速地把一个表转储到服务器机器上。如果您想要在服务器主机之外的部分客户主机上创建结果文件，您不能使用SELECT...INTO OUTFILE。在这种情况下，您应该在客户主机上使用比如"mysql –e "SELECT ..." > file_name"的命令，来生成文件。

SELECT...INTO OUTFILE是LOAD DATA INFILE的补语；用于语句的exort_options部分的语法包括部分FIELDS和LINES子句，这些子句与LOAD DATA INFILE语句同时使用。请参见13.2.5节，"LOAD DATA INFILE语法"。

FIELDS ESCAPED BY用于控制如何写入特殊字符。如果FIELDS ESCAPED BY字符不是空字符，则被用于在输出中对以下字符设前缀：

- FIELDS ESCAPED BY字符
- FIELDS [OPTIONALLY] ENCLOSED BY字符
- FIELDS TERMINATED BY和LINES TERMINATED BY值的第一个字符
- ASCII 0（在编写时接在转义符后面的是ASCII '0'，而不是一个零值字节）

如果FIELDS ESCAPED BY字符是空字符，则没有字符被转义，并且NULL被作为NULL输出，而不是作为\N输

出。指定一个空的转义符不是一个好的主意。特别是当您的数据中的字段值包含刚被给予的清单中的字符时，更是如此。

其原因是您必须对所有FIELDS TERMINATED BY, ENCLOSED BY, ESCAPED BY或LINES TERMINATED BY字符进行转义，才能可靠地读取文件并返回。ASCII NUL被转义，以便更容易地使用调页程序观看。

生成的文件不必符合SQL语法，所以没有其它的字符需要被转义。

在下面的例子中，生成一个文件，各值用逗号隔开。这种格式可以被许多程序使用。

```
SELECT a,b,a+b INTO OUTFILE '/tmp/result.text'

FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'

LINES TERMINATED BY '\n'

FROM test_table;
```

· 如果您使用INTO DUMPFILE代替INTO OUTFILE，则MySQL只把一行写入到文件中，不对任何列或行进行终止，也不执行任何转义处理。如果您想要把一个BLOB值存储到文件中，则这个语句是有用的。

· 注释：任何由INTO OUTFILE或INTO DUMPFILE创建的文件都可以被服务器主机上的所有用户编写。原因是，MySQL服务器不能创建这样的文件，即文件的所有者不是该文件运行时所属的用户（任何时候，您都不能出于此原因或出于其它原因把**mysqld**作为根段运行）。该文件必须是全局可写的，这样您就可以操作其中的内容。

· 有的过程应在结果集合内处理数据。PROCEDURE子句用于对这些过程进行命名。要了解示例，请参见27.3.1节，“步骤分析”。

· 存储引擎使用页面或行锁。如果您对存储引擎使用FOR UPDATE，则受到查询检验的行会被进行写锁定，直到当前事务结束为止。使用LOCK IN SHARE MODE可以设置一个共享锁。共享锁可以防止其它事务更新或删除被检验的行。请参见15.2.10.5节，“锁定读取SELECT ... FOR UPDATE和SELECT ... LOCK IN SHARE MODE”。

在SELECT关键词的后面，您可以使用许多选项。这些选项可以影响语句的运行。

ALL, DISTINCT和DISTINCTROW选项指定是否重复行应被返回。如果这些选项没有被给定，则默认值为ALL（所有的匹配行被返回）。DISTINCT和DISTINCTROW是同义词，用于指定结果集合中的重复行应被删除。

HIGH_PRIORITY, STRAIGHT_JOIN和以SQL_为开头的选项是MySQL相对于标准SQL的扩展。

· HIGH_PRIORITY给予SELECT更高的优先权，高于用于更新表的语句。您应该只对查询使用HIGH_PRIORITY。查询速度非常快，而且立刻被执行。SELECT HIGH_PRIORITY查询在表被锁定用于读取时被发出。即使有一个新的语句正在等待表变为空闲，查询也会运行。

HIGH_PRIORITY不能和SELECT语句同时使用。SELECT语句是UNION的一部分。

· STRAIGHT_JOIN用于促使优化符把表联合在一起，顺序按照这些表在FROM子句中排列的顺序。如果优化符联合表时顺序不佳，您可以使用STRAIGHT_JOIN来加快查询的速度。请参见7.2.1节，“EXPLAIN语法（获取关于SELECT的信息）”。STRAIGHT_JOIN也可以被用于*table_references*清单中。请参见13.2.7.1节，“JOIN语法”。

· SQL_BIG_RESULT可以与GROUP BY或DISTINCT同时使用，来告知优化符结果集合有很多行。在这种情况下，MySQL直接使用以磁盘为基础的临时表（如果需要的话）。在这种情况下，MySQL还会优先进行分类，不优先使用临时表。临时表对于GROUP BY组分带有关键字。

· SQL_BUFFER_RESULT促使结果被放入一个临时表中。这可以帮助MySQL提前解开表锁定，在需要花费

较长时间的情况下，也可以帮助把结果集合发送到客户端中。

· SQL_SMALL_RESULT可以与GROUP BY或DISTINCT同时使用，来告知优化符结果集合是较小的。在此情况下，MySAL使用快速临时表来储存生成的表，而不是使用分类。在MySQL 5.1中，通常不需要这样。

· SQL_CALC_FOUND_ROWS告知MySQL计算有多少行应位于结果集合中，不考虑任何LIMIT子句。行的数目可以使用SELECT FOUND_ROWS()恢复。请参见12.9.3节，“信息函数”。

· 如果您正在使用一个query_cache_type值，值为2或DEMAND，则SQL_CACHE告知MySQL把查询结果存储在查询缓存中。对于使用UNION的查询或子查询，本选项会影响查询中的所有SELECT。请参见5.13节，“MySQL查询高速缓冲”。

· SQL_NO_CACHE告知MySQL不要把查询结果存储在查询缓存中。请参见5.13节，“MySQL查询高速缓冲”。对于一个使用UNION或子查询的查询，本选项会影响查询中的SELECT。

13.2.7.1. JOIN语法

MySQL支持以下JOIN语法。这些语法用于SELECT语句的*table_references*部分和多表DELETE和UPDATE语句：

```
table_references:
    table_reference [, table_reference] …

table_reference:
    table_factor
  | join_table

table_factor:
    tbl_name [[AS] alias]
        [{USE|IGNORE|FORCE} INDEX (key_list)]
  | ( table_references )
  | { OJ table_reference LEFT OUTER JOIN table_reference
        ON conditional_expr }

join_table:
    table_reference [INNER | CROSS] JOIN table_factor [join_condition]
  | table_reference STRAIGHT_JOIN table_factor
  | table_reference STRAIGHT_JOIN table_factor ON condition
  | table_reference LEFT [OUTER] JOIN table_reference join_condition
  | table_reference NATURAL [LEFT [OUTER]] JOIN table_factor
  | table_reference RIGHT [OUTER] JOIN table_reference join_condition
  | table_reference NATURAL [RIGHT [OUTER]] JOIN table_factor

join_condition:
    ON conditional_expr
  | USING (column_list)
```

一个表引用还被称为一个联合表达式。

与SQL标准相比，*table_factor*的语法被扩展了。SQL标准只接受*table_reference*，而不是圆括号内的一系列条目。

如果我们把一系列*table_reference*条目中的每个逗号都看作相当于一个内部联合，则这是一个稳妥的扩展。例如：

```
SELECT * FROM t1 LEFT JOIN (t2, t3, t4)

                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

相当于：

```
SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)

                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

在MySQL中，CROSS JOIN从语法上说与INNER JOIN等同（两者可以互相替换。在标准SQL中，两者是不等同的。INNER JOIN与ON子句同时使用，CROSS JOIN以其它方式使用。

通常，在只含有内部联合运行的联合表达式中，圆括号可以被忽略。MySQL也支持嵌套的联合（见7.2.10节，“MySQL如何优化嵌套Join”）。

通常，您不应对ON部分有任何条件。ON部分用于限定在结果集合中您想要哪些行。但是，您应在WHERE子句中指定这些条件。这条规则有一些例外。

在前面的清单中显示的{ OJ ... LEFT OUTER JOIN ...}语法的目的只是为了保持与ODBC的兼容性。语法中的花括号应按字面书写；该括号不是中间语法。中间语法用于语法描述的其它地方。

- 表引用可以使用*tbl_name* AS *alias_name*或*tbl_name alias_name*指定别名：

```
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;
```

- ON条件句是可以被用于WHERE子句的格式的任何条件表达式。

- 如果对于在LEFT JOIN中的ON或USING部分中的右表没有匹配的记录，则所有列被设置为NULL的一个行被用于右表。如果一个表在其它表中没有对应部分，您可以使用这种方法在这种表中查找记录：

```
mysql> SELECT table1.* FROM table1
    ->        LEFT JOIN table2 ON table1.id=table2.id
    ->        WHERE table2.id IS NULL;
```

本例查找在table1中含有一个id值的所有行。同时，在table2中没有此id值（即，table1中的所有行在table2中没有对应的行）。本例假设table2.id被定义为NOT NULL。请参见7.2.9节，“MySQL如何优化LEFT JOIN和RIGHT JOIN”。

- USING(*column_list*)子句用于为一系列的列进行命名。这些列必须同时在两个表中存在。如果表a和表b都包含列c1, c2和c3，则以下联合会对比来自两个表的对应的列：

```
a LEFT JOIN b USING (c1,c2,c3)
```

- 两个表的NATURAL [LEFT] JOIN被定义为与INNER JOIN语义相同，或与使用USING子句的LEFT JOIN语义相同。USING子句用于为同时存在于两个表中的所有列进行命名。

- INNER JOIN和，（逗号）在无联合条件下是语义相同的：两者都可以对指定的表计算出笛卡儿乘积（也就是说，第一个表中的每一行被联合到第二个表中的每一行）。

- RIGHT JOIN的作用与LEFT JOIN的作用类似。要使代码可以在数据库内移植，建议您使用LEFT JOIN代替RIGHT JOIN。

- STRAIGHT_JOIN与JOIN相同。除了有一点不一样，左表会在右表之前被读取。STRAIGH_JOIN可以被用于这样的情况，即联合优化符以错误的顺序排列表。

您可以提供提示，当从一个表中恢复信息时，MySQL应使用哪个索引。通过指定USE INDEX（key_list），您可以告知MySQL只使用一个索引来查找表中的行。另一种语法IGNORE INDEX（key_list）可以被用于告知MySQL不要使用某些特定的索引。如果EXPLAIN显示MySQL正在使用来自索引清单中的错误索引时，这些提示会有用处。

您也可以使用FORCE INDEX，其作用接近USE INDEX（key_list），不过增加了一项作用，一次表扫描被假设为代价很高。换句话说，只有当无法使用一个给定的索引来查找表中的行时，才使用表扫描。

USE KEY、IGNORE KEY和FORCE KEY是USE INDEX、IGNORE INDEX和FORCE INDEX的同义词。

注释：当MySQL决定如何在表中查找行并决定如何进行联合时，使用USE INDEX、IGNORE INDEX和FORCE INDEX只会影响使用哪些索引。当分解一个ORDER BY或GROUP BY时，这些语句不会影响某个索引是否被使用。

部分的联合示例：

```
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;
```

见7.2.9节，“MySQL如何优化LEFT JOIN和RIGHT JOIN”。

注释：自然联合和使用USING的联合，包括外部联合变量，依据SQL:2003标准被处理。这些变更时MySQL与标准SQL更加相符。不过，对于有些联合，这些变更会导致不同的输出列。另外，有些查询在旧版本（5.0.12以前）工作正常，但也必须重新编写，以符合此标准。对于有关当前联合处理和旧版本中的联合处理的效果的对比，以下列表提供了更详细的信息。

· NATURAL联合或USING联合的列会与旧版本不同。特别是，不再出现冗余的输出列，用于SELECT *扩展的列的顺序会与以前不同。

示例：

```
CREATE TABLE t1 (i INT, j INT);

CREATE TABLE t2 (k INT, j INT);

INSERT INTO t1 VALUES(1,1);

INSERT INTO t2 VALUES(1,1);

SELECT * FROM t1 NATURAL JOIN t2;

SELECT * FROM t1 JOIN t2 USING (j);
```

对于旧版本，语句会产生以下输出：

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

| i    | j    | k    | j    |

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

|    1 |    1 |    1 |    1 |

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

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

```
| i    | j    | k    | j    |
+------+------+------+------+
|    1 |    1 |    1 |    1 |
+------+------+------+------+
```

在第一个SELECT语句中，列i同时出现在两个表中，为一个联合列，所以，依据标准SQL，该列在输出中只出现一次。与此类似，在第二个SELECT语句中，列j在USING子句中被命名，应在输出中只出现一次。但是，在两种情况下，冗余的列均没被消除。另外，依据标准SQL，列的顺序不正确。

现在，语句产生如下输出：

```
+------+------+------+
| j    | i    | k    |
+------+------+------+
|    1 |    1 |    1 |
+------+------+------+
+------+------+------+
| j    | i    | k    |
+------+------+------+
|    1 |    1 |    1 |
+------+------+------+
```

冗余的列被消除，并且依据标准SQL，列的顺序是正确的：

- 第一，两表共有的列，按在第一个表中的顺序排列
- 第二，第一个表中特有的列，按该表中的顺序排列
- 第三，第二个表中特有的列，按该表中的顺序排列

· 对多方式自然联合的估算会不同。方式要求重新编写查询。假设您有三个表t1(a,b), t2(c,b)和t3(a,c)，每个表有一行：t1(1,2), t2(10,2)和t3(7,10)。同时，假设这三个表具有NATURAL JOIN：

```
·                SELECT … FROM t1 NATURAL JOIN t2 NATURAL JOIN t3;
```

在旧版本中，第二个联合的左操作数被认为是t2，然而它应该为嵌套联合（t1 NATURAL JOIN t2）。结果，对t3的列进行检查时，只检查其在t2中的共有列。如果t3与t1有共有列，这些列不被用作equi-join列。因此，在旧版本的MySQL中，前面的查询被转换为下面的equi-join：

```
SELECT … FROM t1, t2, t3
  WHERE t1.b = t2.b AND t2.c = t3.c;
```

此联合又省略了一个equi-join谓语（t1.a = t3.a）。结果是，该联合产生一个行，而不是空结果。正确的等价查询如下：

```
SELECT … FROM t1, t2, t3
  WHERE t1.b = t2.b AND t2.c = t3.c AND t1.a = t3.a;
```

如果您要求在当前版本的MySQL中获得和旧版本中相同的查询结果，应把自然联合改写为第一个equi-join。

· 在旧版本中，逗号操作符（，）和JOIN均有相同的优先权，所以联合表达式t1, t2 JOIN t3被理解((t1, t2) JOIN t3) JOIN (t1, (t2 JOIN t3))

为 。现在， 有更高的优先权，所以表达式被理解为 。这个变更会影响使用ON子句的语句，因为该子句只参阅联合操作数中的列。优先权的变更改变了对什么是操作数的理解。

示例：

```
CREATE TABLE t1 (i1 INT, j1 INT);
CREATE TABLE t2 (i2 INT, j2 INT);
CREATE TABLE t3 (i3 INT, j3 INT);
INSERT INTO t1 VALUES(1,1);
INSERT INTO t2 VALUES(1,1);
INSERT INTO t3 VALUES(1,1);
SELECT * FROM t1, t2 JOIN t3 ON (t1.i1 = t3.i3);
```

在旧版本中，SELECT是合法的，因为t1, t2被隐含地归为(t1,t2)。现在，JOIN取得了优先权，因此用于ON子句的操作数是t2和t3。因为t1.i1不是任何一个操作数中的列，所以结果是出现在'on clause'中有未知列't1.i1'的错误。要使联合可以被处理，用使用圆括号把前两个表明确地归为一组，这样用于ON子句的操作数为(t1,t2)和t3：

```
SELECT * FROM (t1, t2) JOIN t3 ON (t1.i1 = t3.i3);
```

本变更也适用于INNER JOIN，CROSS JOIN，LEFT JOIN和RIGHT JOIN。

- 在旧版本中，ON子句可以参阅在其右边命名的表中的列。现在，ON子句只能参阅操作数。

示例：

```
CREATE TABLE t1 (i1 INT);
CREATE TABLE t2 (i2 INT);
CREATE TABLE t3 (i3 INT);
SELECT * FROM t1 JOIN t2 ON (i1 = i3) JOIN t3;
```

在旧版本中，SELECT语句是合法的。现在该语句会运行失败，出现在'on clause'中未知列'i3'的错误。这是因为i3是t3中的一个表，而t3不是ON子句中的操作数。本语句应进行如下改写：

```
SELECT * FROM t1 JOIN t2 JOIN t3 ON (i1 = i3);
```

- 在旧版本中，一个USING子句可以被改写为一个ON子句。ON子句对比了相应的列。例如，以下两个子句具有相同的语义：

```
· 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
```

现在，这两个子句不再是一样的：

o 在决定哪些行满足联合条件时，两个联合保持语义相同。

o 在决定哪些列显示SELECT *扩展时，两个联合的语义不相同。USING联合选择对应列中的合并值，而ON联合选择所有表中的所有列。对于前面的USING联合，SELECT *选择这些值：

```
o COALESCE(a.c1,b.c1), COALESCE(a.c2,b.c2), COALESCE(a.c3,b.c3)
```

对于ON联合，SELECT *选择这些值：

```
a.c1, a.c2, a.c3, b.c1, b.c2, b.c3
```

使用内部联合时，COALESCE(a.c1,b.c1)与a.c1或b.c1相同，因为两列将具有相同的值。使用外部联合时（比如LEFT JOIN），两列中有一列可以为NULL。该列将会从结果中被忽略。

13.2.7.2. UNION语法

```
SELECT ...

UNION [ALL | DISTINCT]

SELECT ...

[UNION [ALL | DISTINCT]

SELECT ...]
```

UNION用于把来自许多SELECT语句的结果组合到一个结果集合中。

列于每个SELECT语句的对应位置的被选择的列应具有相同的类型。（例如，被第一个语句选择的第一列应和被其它语句选择的第一列具有相同的类型。）在第一个SELECT语句中被使用的列名称也被用于结果的列名称。

SELECT语句为常规的选择语句，但是受到如下的限定：

· 只有最后一个SELECT语句可以使用INTO OUTFILE。

· HIGH_PRIORITY不能与作为UNION一部分的SELECT语句同时使用。如果您对第一个SELECT指定了HIGH_PRIORITY，则不会起作用。如果您对其它后续的SELECT语句指定了HIGH_PRIORITY，则会产生语法错误。

如果您对UNION不使用关键词ALL，则所有返回的行都是唯一的，如同您已经对整个结果集合使用了DISTINCT。如果您指定了ALL，您会从所有用过的SELECT语句中得到所有匹配的行。

DISTINCT关键词是一个自选词，不起任何作用，但是根据SQL标准的要求，在语法中允许采用。（在MySQL中，DISTINCT代表一个共用体的默认工作性质。）

您可以在同一查询中混合UNION ALL和UNION DISTINCT。被混合的UNION类型按照这样的方式对待，即DISTICT共用体覆盖位于其左边的所有ALL共用体。DISTINCT共用体可以使用UNION DISTINCT明确地生成，或使用UNION（后面不加DISTINCT或ALL关键词）隐含地生成。

如果您想使用ORDER BY或LIMIT子句来对全部UNION结果进行分类或限制，则应对单个地SELECT语句加圆括号，并把ORDER BY或LIMIT放到最后一个的后面。以下例子同时使用了这两个子句：

```
(SELECT a FROM tbl_name WHERE a=10 AND B=1)

UNION

(SELECT a FROM tbl_name WHERE a=11 AND B=2)

ORDER BY a LIMIT 10;
```

这种ORDER BY不能使用包括表名称（也就是，采用*tbl_name.col_name*格式的名称）列引用。可以在第一个SELECT语句中提供一个列别名，并在ORDER BY中参阅别名，或使用列位置在ORDER BY中参阅列。（首选采用别名，因为不建议使用列位置。）

另外，如果带分类的一列有别名，则ORDER BY子句必须引用别名，而不能引用列名称。以下语句中的第一个语句必须运行，但是第二个会运行失败，出现在'order clause'中有未知列'a'的错误：

```
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY b;
```

```
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY a;
```

To apply ORDER BY or LIMIT to an individual SELECT, place the clause inside the parentheses that enclose the SELECT: 为了对单个SELECT使用ORDER BY或LIMIT，应把子句放入圆括号中。圆括号包含了SELECT：

```
(SELECT a FROM tbl_name WHERE a=10 AND B=1 ORDER BY a LIMIT 10)
UNION
(SELECT a FROM tbl_name WHERE a=11 AND B=2 ORDER BY a LIMIT 10);
```

圆括号中用于单个SELECT语句的ORDER BY只有当与LIMIT结合后，才起作用。否则，ORDER BY被优化去除。

UNION结果集合中的列的类型和长度考虑了被所有SELECT语句恢复的数值。例如，考虑如下语句：

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+---------------+
| REPEAT('a',1) |
+---------------+
| a             |
| bbbbbbbbbb    |
+---------------+
```

（在部分早期版本的MySQL中，第二行已被删节到长度为1。）

13.2.8. Subquery语法

13.2.8.1. 子查询作为标量操作数
13.2.8.2. 使用子查询进行比较
13.2.8.3. 使用ANY, IN和SOME进行子查询
13.2.8.4. 使用ALL进行子查询
13.2.8.5. 行子查询
13.2.8.6. EXISTS和NOT EXISTS
13.2.8.7. 关联子查询
13.2.8.8. FROM子句中的子查询
13.2.8.9. 子查询错误
13.2.8.10. 优化子查询
13.2.8.11. 把子查询作为用于早期MySQL版本的联合进行改写

子查询是另一个语句中的一个SELECT语句。

MySQL支持SQL标准要求的所有子查询格式和操作，也支持MySQL特有的几种特性。

以下是一个子查询的例子：

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

在本例中，SELECT * FROM t1...是外部查询（或外部语句），（SELECT column1 FROM t2）是子查询。我们可以说子查询嵌套在外部查询中。实际上，子查询也可以嵌套在其它子查询中，嵌套程度可以很深。子查询必须要位于圆括号中。

子查询的主要优势为：

- 子查询允许结构化的查询，这样就可以把一个语句的每个部分隔离开。
- 有些操作需要复杂的联合和关联。子查询提供了其它的方法来执行这些操作。
- 在许多人看来，子查询是可读的。实际上，子查询给予人们调用早期SQL"结构化查询语言"的原本的想法，这是子查询的创新。

以下是一个示例语句。该语句显示了有关子查询语法的要点。子查询语法由SQL标准指定并被MySQL支持。

```
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)));
```

一个子查询会返回一个标量（单一值）、一个行、一个列或一个表（一行或多行及一列或多列）。这些子查询被称为标量、列、行和表子查询。可返回一个特定种类结果的子查询经常只被用于特定的语境中，在后面各节中有说明。

有些语句可以使用子查询。对这些语句的类型基本没有限定。子查询可以包括普通SELECT可以包括的任何关键词或子句：DISTINCT, GROUP BY, ORDER BY, LIMIT, 联合, 索引提示, UNION结构化, 评注和函数等。

有一个限定是，一个子查询的外部语句必须是以下语句之一：SELECT, INSERT, UPDATE, DELETE, SET或DO。还有一个限定是，目前，您不能在一个子查询中修改一个表，又在同一个表中选择。这适用于DELETE, INSERT, REPLACE和UPDATE语句。在附录I：特性限制中给出了对子查询使用的更综合的讨论。

13.2.8.1. 子查询作为标量操作数

子查询最简单的形式是返回单一值的标量子查询。标量子查询是一个单一操作数。只要单一列值或文字是合法的，并且您希望子查询具有所有操作数都具有的特性，则您就可以使用子查询。操作数具有的特性包括：一个数据类型、一个长度、一个指示是否可以为NULL的标志等。举例说明：

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5) NOT NULL);

INSERT INTO t1 VALUES(100, 'abcde');

SELECT (SELECT s2 FROM t1);
```

在本SELECT中的子查询返回一个单一值('abcde')。该单一值的数据类型为CHAR，长度为5，字符集和整序与在CREATE TABLE时有效的默认值相同，并有一个指示符号，指示列中的值可以为NULL。实际上，基本上所有的子查询都为NULL。如果在本例中使用的表为空表，则子查询的值应为NULL。

在有些情况下，标量子查询不能使用。如果一个语句只允许一个文字值，您不能使用子查询。例如，LIMIT要求文字整数自变量，LOAD DATA要求一个文字字符串文件名。您不能使用子查询来提供这些值。

后面各节包括更简练的结构（SELECT column1 FROM t1）。当您在这些章节中观看例子时，请设想一下您自己的代码包含更多样、更复杂的结构。

举例说明，假设我们制作两个表：

```
CREATE TABLE t1 (s1 INT);
```

```
INSERT INTO t1 VALUES (1);

CREATE TABLE t2 (s1 INT);

INSERT INTO t2 VALUES (2);
```

然后执行一个SELECT：

```
SELECT (SELECT s1 FROM t2) FROM t1;
```

结果为2，因为t2中有一行包含s1，s1有一个值为2。

一个标量子查询可以为一个表达式的一部分。不要忘记圆括号。即使是子查询是一个为函数提供自变量的操作数时，也不要忘记圆括号。举例说明：

```
SELECT UPPER((SELECT s1 FROM t1)) FROM t2;
```

13.2.8.2. 使用子查询进行比较

子查询最常见的一种使用方式如下：

non_subquery_operand comparison_operator (*subquery*)

当*comparison_operator*是以下 操作符之一时：

```
=  >  <  >=  <=  <>
```

例如：

```
  ... 'a' = (SELECT column1 FROM t1)
```

有时，子查询的合法位置只能在比较式的右侧，您可以发现，在有些旧的DBMSs中仍保持这一点。

以下是一个常见格式的子查询比较的例子。您不能使用联合进行此类比较。表t1中有些值与表t2中的最大值相同。该比较可以查找出所有这类值：

```
SELECT column1 FROM t1

WHERE column1 = (SELECT MAX(column2) FROM t2);
```

下面还有另一个例子，该例子也不可能使用联合，因为该例子涉及对其中一个表进行总计。表t1中的有些行含有的值会在给定的列中出现两次。该例子可以查找出所有这些行：

```
SELECT * FROM t1 AS t

WHERE 2 = (SELECT COUNT(*) FROM t1 WHERE t1.id = t.id);
```

对于采用这些操作符之一进行的比较，子查询必须返回一个标量。有一个例外，那就是＝可以和行子查询同时使用。请参见13.2.8.5节，“行子查询”。

13.2.8.3. 使用ANY, IN和SOME进行子查询

语法：

operand comparison_operator ANY (*subquery*)

operand IN (*subquery*)

operand comparison_operator SOME (*subquery*)

ANY关键词必须后面接一个比较操作符。ANY关键词的意思是“对于在子查询返回的列中的任一数值，如果比较结果为TRUE的话，则返回TRUE”。例如：

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

假设表t1中有一行包含（10）。如果表t2包含（21，14，7），则表达式为TRUE，因为t2中有一个值为7，该值小于10。如果表t2包含（20，10），或者如果表t2为空表，则表达式为FALSE。如果表t2包含（NULL, NULL, NULL），则表达式为UNKNOWN。

词语IN是＝ANY的别名。因此，这两个语句是一样的：

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);

SELECT s1 FROM t1 WHERE s1 IN    (SELECT s1 FROM t2);
```

不过，NOT IN不是<> ANY的别名，但是是<> ALL的别名。请参见13.2.8.4节，“使用ALL进行子查询”。

词语SOME是ANY的别名。因此，这两个语句是一样的：

```
SELECT s1 FROM t1 WHERE s1 <> ANY  (SELECT s1 FROM t2);

SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);
```

使用词语SOME的机会很少，但是本例显示了为什么SOME是有用的。对于多数人来说，英语短语“a is not equal to any b”的意思是“没有一个b与a相等”，但是在SQL语法中不是这个意思。该语法的意思是“有部分b与a不相等”。使用<> SOME有助于确认每个人都理解该查询的真正含义。

13.2.8.4. 使用ALL进行子查询

语法：

```
operand comparison_operator ALL (subquery)
```

词语ALL必须接在一个比较操作符的后面。ALL的意思是“对于子查询返回的列中的所有值，如果比较结果为TRUE，则返回TRUE。”例如：

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

假设表1中有一行包含（10）。如果表t2包含（-5，0，＋5），则表达式为TRUE，因为10比t2中的所有三个值都大。如果表t2包含（12，6，NULL，－100），则表达式为FALSE，因为表t2中有一个值12大于10。如果表t2包含（0，NULL，1），则表达式为unknown。

最后，如果表t2为空表，则结果为TRUE。因此，当表t2为空表时，以下语句为TRUE：

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT s1 FROM t2);
```

但是，当表t2为空表时，本语句为NULL：

```
SELECT * FROM t1 WHERE 1 > (SELECT s1 FROM t2);
```

另外，当表t2为空表时，以下语句为NULL：

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT MAX(s1) FROM t2);
```

通常，包含NULL值的表和空表为“边缘情况”。当编写子查询代码时，都要考虑您是否把这两种可能性计算在内。

NOT IN是<> ALL的别名。因此，以下两个语句是相同的：

```
SELECT s1 FROM t1 WHERE s1 <> ALL (SELECT s1 FROM t2);

SELECT s1 FROM t1 WHERE s1 NOT IN (SELECT s1 FROM t2);
```

13.2.8.5. 行子查询

对于本点的讨论属于标量或列子查询，即返回一个单一值或一列值的子查询。行子查询是一个能返回一个单一行的子查询变量，因此可以返回一个以上的列值。以下是两个例子：

```
SELECT * FROM t1 WHERE (1,2) = (SELECT column1, column2 FROM t2);
SELECT * FROM t1 WHERE ROW(1,2) = (SELECT column1, column2 FROM t2);
```

如果在表t2的一个行中，column1=1并且column2=2，则查询结果均为TRUE。

表达式（1，2）和ROW（1，2）有时被称为行构造符。两者是等同的，在其它的语境中，也是合法的。例如，以下两个语句在语义上是等同的（但是目前只有第二个语句可以被优化）：

```
  SELECT * FROM t1 WHERE (column1,column2) = (1,1);
  SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

行构造符通常用于与对能返回两个或两个以上列的子查询进行比较。例如，以下查询可以答复请求，“在表t1中查找同时也存在于表t2中的所有的行”：

```
SELECT column1,column2,column3
FROM t1
WHERE (column1,column2,column3) IN
(SELECT column1,column2,column3 FROM t2);
```

13.2.8.6. EXISTS和NOT EXISTS

如果一个子查询返回任何的行，则EXISTS *subquery*为FALSE。例如：

```
SELECT column1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

过去，EXISTS子查询以SELECT *为开始，但是可以以SELECT 5或SELECT column1或其它的为开始。MySQL在这类子查询中忽略了SELECT清单，因此没有区别。

对于前面的例子，如果t2包含任何行，即使是只含有NULL值的行，EXISTS条件也为TRUE。这实际上是一个不可能的例子，因为基本上所有的[NOT] EXISTS子查询均包含关联。以下是一些更现实的例子：

- 哪些种类的商店出现在一个或多个城市里？

```
SELECT DISTINCT store_type FROM stores
  WHERE EXISTS (SELECT * FROM cities_stores
                WHERE cities_stores.store_type = stores.store_type);
```

- 哪些种类的商店没有出现在任何城市里？

```
SELECT DISTINCT store_type FROM stores
  WHERE NOT EXISTS (SELECT * FROM cities_stores
                    WHERE cities_stores.store_type = stores.store_type);
```

- 哪些种类的商店出现在所有城市里？

```
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));
```

最后一个例子是一个双嵌套NOT EXISTS查询。也就是，该查询包含一个NOT EXISTS子句，该子句又包含在一个NOT EXISTS子句中。该查询正式地回答了这个问题，“是否有某个城市拥有没有列在Stores中的商店？”。可以比较容易的说，一个带嵌套的NOT EXISTS可以回答这样的问题，“是否对于所有的y，x都为TRUE？”

13.2.8.7. 关联子查询

相关联的子查询是一个包含对表的引用的子查询。该表也显示在外部查询中。例如：

```
SELECT * FROM t1 WHERE column1 = ANY
(SELECT column1 FROM t2 WHERE t2.column2 = t1.column2);
```

注意，即使子查询的FROM子句不提及表t1，该子查询也会包含一个对t1中一列的引用。所以，MySQL看上去位于子查询的外部，并在外部查询中查找t1。

假设表t1包含一行，在此行中column1=5并且column2=6；同时，表t2包含一行，在此行中column1=5并且column2=7。简单的表达式... WHERE column1 = ANY (SELECT column1 FROM t2)会为TRUE。但是在本例中，在子查询中的WHERE子句为FALSE（因为（5，6）不等于（5，7）），所以子查询总体上为FALSE。

范围划分规则：MySQL从内到外进行评估。例如：

```
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));
```

在本语句中，x.column2必须是表t2中的列，因为SELECT column1 FROM t2 AS x ...对t2进行了重命名。它不是表t1中的列，因为SELECT column1 FROM t1 ...是一个更靠外的外部查询。

对于HAVING或ORDER BY子句中的子查询，MySQL也会在外部选择清单中寻找列名称。

对于特定的情况，相关联的子查询被优化。例如：

```
val IN (SELECT key_val FROM tbl_name WHERE correlated_condition)
```

否则，这些子查询效率不高，可能速度会慢。把查询作为联合进行改写可能会改进效率。

相关联的子查询不能从外部查询中引用总计函数的结果。

13.2.8.8. FROM子句中的子查询

在SELECT语句的FROM子句中，子查询是合法的。实际的语法是：

```
SELECT ... FROM (subquery) [AS] name ...
```

[AS] *name*子句是强制性的，因为FROM子句中的每个表必须有一个名称。在子查询选择列表中的任何列都必须有唯一的名称。您可以在本手册中的其它地方找到对本语法的说明。在该处，所用的词语是“导出表”。

为了进行详细说明，假设您有如下一个表：

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5), s3 FLOAT);
```

下面使用了示例表，解释了在FROM子句中如何使用子查询：

```
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;
```

结果：2, '2', 4.0。

下面是另一个例子：假设您想了解一个分类后的表的一组和的平均值。采用如下操作：

```
SELECT AVG(SUM(column1)) FROM t1 GROUP BY column1;
```

不过，本查询提供所需的信息：

```
SELECT AVG(sum_column1)
FROM (SELECT SUM(column1) AS sum_column1
FROM t1 GROUP BY column1) AS t1;
```

注意，在子查询中使用的列名称（sum_column1）被整理到外部查询中。

FROM子句中的子查询可以返回标量、列、行或表。FROM子句中的子查询不能为有关联的子查询。

即使对EXPLAIN语句（即建立临时导出表），FROM子句中的子查询也会被执行。这是因为在优化过程中，上一级的查询需要有关所有表的信息。

13.2.8.9. 子查询错误

以下错误只适用于子查询。本节把这些错误归在一起。

- 来自子查询的列的数目不正确

```
ERROR 1241 (ER_OPERAND_COL)
SQLSTATE = 21000
Message = "Operand should contain 1 column(s)"
```

在出现以下情况时，发生此错误：

```
SELECT (SELECT column1, column2 FROM t2) FROM t1;
```

如果您的目的是进行比较，您可以使用能返回多个列的子查询。请参见13.2.8.5节，“行子查询”。不过，在其它的语境下，子查询必须为标量操作数。

- 来自子查询的行的数目不正确：

```
ERROR 1242 (ER_SUBSELECT_NO_1_ROW)
SQLSTATE = 21000
Message = "Subquery returns more than 1 row"
```

如果在语句中，子查询返回的行多于一个，则发生此错误。请考虑以下例子：

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

如果SELECT column1 FROM t2只返回一行，则将执行以前的查询。如果子查询返回的行多于一个，则将出现错误1242。在这种情况下，该查询将被改写为：

```
SELECT * FROM t1 WHERE column1 = ANY (SELECT column1 FROM t2);
```

- 在子查询中表格使用不正确：

```
Error 1093 (ER_UPDATE_TABLE_USED)
SQLSTATE = HY000
Message = "You can't specify target table 'x'
for update in FROM clause"
```

在如下情况下，发生该错误：

```
UPDATE t1 SET column2 = (SELECT MAX(column1) FROM t1);
```

和SELECT语句一样，在UPDATE和DELETE语句中，子查询是合法的。所以您可以在UPDATE语句中使用子查询进行赋值。不过，您不能把同一个表（在本例中为表t1）既用于子查询的FROM子句，又用于更新目标。

对于事务存储引擎，子查询的错误会导致整个语句失效。对于非事务存储引擎，在遇到错误之前进行的数据修订会被保留。

13.2.8.10. 优化子查询

开发过程不断进展，所以从长远来看，没有一个可靠的优化技巧。有些技巧您可能会感兴趣，并原意采用：

- 有些子句会影响在子查询中的行的数量和顺序。使用这类子句。例如：

```
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);
```

- 用子查询替换联合。例如，试进行如下操作：

```
SELECT DISTINCT column1 FROM t1 WHERE t1.column1 IN (
SELECT column1 FROM t2);
```

代替如下操作：

```
SELECT DISTINCT t1.column1 FROM t1, t2
WHERE t1.column1 = t2.column1;
```

- 部分子查询可以被转换为联合，以便与不支持子查询的旧版本的MySQL相兼容。不过，在有些情况下，把子查询转化为联合可以提高效果。请参见13.2.8.11节，“把子查询作为用于早期MySQL版本的联合进行改写”。

- 把子句从子查询的外部转移到内部。例如，使用此查询：

```
SELECT * FROM t1
```

```
·                WHERE s1 IN (SELECT s1 FROM t1 UNION ALL SELECT s1 FROM t2);
```

代替此查询:

```
SELECT * FROM t1
WHERE s1 IN (SELECT s1 FROM t1) OR s1 IN (SELECT s1 FROM t2);
```

另一个例子是，使用此查询:

```
SELECT (SELECT column1 + 5 FROM t1) FROM t2;
```

代替此查询:

```
SELECT (SELECT column1 FROM t1) + 5 FROM t2;
```

· 使用行子查询，代替关联子查询。举例说明，使用此查询:

```
·                SELECT * FROM t1
·                WHERE (column1,column2) IN (SELECT column1,column2 FROM t2);
```

代替此查询:

```
SELECT * FROM t1
WHERE EXISTS (SELECT * FROM t2 WHERE t2.column1=t1.column1
AND t2.column2=t1.column2);
```

· Use NOT (a = ANY (...)) rather than a <> ALL (...).

· Use x = ANY (table containing (1,2)) rather than x=1 OR x=2.

· Use = ANY rather than EXISTS.

· 对于只返回一行的无关联子查询，IN的速度慢于＝。举例说明，使用此查询:

```
·                SELECT * FROM t1 WHERE t1.col_name
·                = (SELECT a FROM t2 WHERE b = some_const);
```

代替此查询:

```
SELECT * FROM t1 WHERE t1.col_name
IN (SELECT a FROM t2 WHERE b = some_const);
```

使用这些技巧可以使程序更快或更慢。使用BENCHMARK()函数等MySQL工具，您可以了解到在您所处的情况下，哪些技巧会有帮助。

MySQL本身进行的部分优化包括:

· MySQL只执行一次无关联子查询。使用EXPLAIN确认给定的子查询确实是无关联的。

· MySQL改写IN, ALL, ANY和SOME子查询，目的是如果子查询中的select-list列已编制索引，则能发挥出此优势。

· MySQL使用index-lookup函数代替以下格式的子查询。EXPLAIN把此函数描述为特殊的联合类型（unique_subquery或index_subquery）:

```
·                ... IN (SELECT indexed_column FROM single_table ...)
```

- 当表达式中不包含NULL值或空集时，MySQL使用一个包含MIN()或MAX()的表达式，对以下格式的表达式进行扩展：

- *value* {ALL|ANY|SOME} {> | < | >= | <=} (*non-correlated subquery*)

例如，本WHERE子句：

```
WHERE 5 > ALL (SELECT x FROM t)
```

可以用优化符进行如下处理：

```
WHERE 5 > (SELECT MAX(x) FROM t)
```

在MySQL内部手册中有一章名为"MySQL如何转换子查询"，可以从http://dev.mysql.com/doc/获取。

13.2.8.11. 把子查询作为用于早期MySQL版本的联合进行改写

在较早版本的MySQL中（早于MySQL 4.1），只支持INSERT...SELECT和REPLACE...SELECT...格式的带嵌套的查询。虽然在MySQL 5.1中没有这种情况，但有时，仍然有其它的方法测试一组值的从属关系。并且，在有些情况下，不仅可以在没有子查询时对查询进行改写，而且有时使用这些方法比使用子查询效率更高。这些方法之一是IN()结构：

举例说明，本查询：

```
SELECT * FROM t1 WHERE id IN (SELECT id FROM t2);
```

可以被改写为：

```
SELECT DISTINCT t1.* FROM t1, t2 WHERE t1.id=t2.id;
```

以下查询：

```
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);
```

也可以使用IN()进行改写：

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

WHERE table2.id IS NULL;
```

LEFT [OUTER] JOIN可以比对应的子查询更快，因为服务器可能对其进行更好的优化——这一点对于单独的MySQL服务器并不明确。在SQL-92之前，不存在外部联合，因此在做某些事情时，子查询是唯一的方法。现在，MySQL服务器和其它许多先进的数据库系统都能提供多种的外部联合类型。

MySQL支持multiple-table DELETE语句，该语句可以被用于高效地删除行。删除时依据来自一个表或同时来自多个表的信息。同时也支持Multiple-table UPDATE语句。

13.2.9. TRUNCATE语法

```
TRUNCATE [TABLE] tbl_name
```

TRUNCATE TABLE用于完全清空一个表。从逻辑上说，该语句与用于删除所有行的DELETE语句等同，但是在有些情况下，两者在使用上有所不同。

对于InnoDB表，如果有需要引用表的外键限制，则TRUNCATE TABLE被映射到DELETE上；否则使用快速删减（取消和重新创建表）。使用TRUNCATE TABLE重新设置AUTO_INCREMENT计数器，设置时不考虑是否有外

键限制。

对于其它存储引擎，在MySQL 5.1中，TRUNCATE TABLE与DELETE FROM有以下几处不同：

- 删减操作会取消并重新创建表，这比一行一行的删除行要快很多。
- 删减操作不能保证对事务是安全的；在进行事务处理和表锁定的过程中尝试进行删减，会发生错误。
- 被删除的行的数目没有被返回。
- 只要表定义文件*tbl_name*.frm是合法的，则可以使用TRUNCATE TABLE把表重新创建为一个空表，即使数据或索引文件已经被破坏。
- 表管理程序不记得最后被使用的AUTO_INCREMENT值，但是会从头开始计数。即使对于MyISAM和InnoDB也是如此。MyISAM和InnoDB通常不再次使用序列值。
- 当被用于带分区的表时，TRUNCATE TABLE会保留分区；即，数据和索引文件被取消并重新创建，同时分区定义（.par）文件不受影响。

TRUNCATE TABLE是在MySQL中采用的一个Oracle SQL扩展。

13.2.10. UPDATE语法

Single-table语法：

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
    SET col_name1=expr1 [, col_name2=expr2 ...]
    [WHERE where_definition]
    [ORDER BY ...]
    [LIMIT row_count]
```

Multiple-table语法：

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
    SET col_name1=expr1 [, col_name2=expr2 ...]
    [WHERE where_definition]
```

UPDATE语法可以用新值更新原有表行中的各列。SET子句指示要修改哪些列和要给予哪些值。WHERE子句指定应更新哪些行。如果没有WHERE子句，则更新所有的行。如果指定了ORDER BY子句，则按照被指定的顺序对行进行更新。LIMIT子句用于给定一个限值，限制可以被更新的行的数目。

UPDATE语句支持以下修饰符：

- 如果您使用LOW_PRIORITY关键词，则UPDATE的执行被延迟了，直到没有其它的客户端从表中读取为止。
- 如果您使用IGNORE关键词，则即使在更新过程中出现错误，更新语句也不会中断。如果出现了重复关键字冲突，则这些行不会被更新。如果列被更新后，新值会导致数据转化错误，则这些行被更新为最接近的合法的值。

如果您在一个表达式中通过*tbl_name*访问一列，则UPDATE使用列中的当前值。例如，以下语句把年龄列设置为比当前值多一：

```
mysql> UPDATE persondata SET age=age+1;
```

UPDATE赋值被从左到右评估。例如，以下语句对年龄列加倍，然后再进行增加：

```
mysql> UPDATE persondata SET age=age*2, age=age+1;
```

如果您把一列设置为其当前含有的值，则MySQL会注意到这一点，但不会更新。

如果您把被已定义为NOT NULL的列更新为NULL，则该列被设置到与列类型对应的默认值，并且累加警告数。对于数字类型，默认值为0；对于字符串类型，默认值为空字符串(")；对于日期和时间类型，默认值为"zero"值。

UPDATE会返回实际被改变的行的数目。Mysql_info() C API函数可以返回被匹配和被更新的行的数目，以及在UPDATE过程中产生的警告的数量。

您可以使用LIMIT *row_count*来限定UPDATE的范围。LIMIT子句是一个与行匹配的限定。只要发现可以满足WHERE子句的*row_count*行，则该语句中止，不论这些行是否被改变。

如果一个UPDATE语句包括一个ORDER BY子句，则按照由子句指定的顺序更新行。

您也可以执行包括多个表的UPDATE操作。*table_references*子句列出了在联合中包含的表。该语法在13.2.7.1节，"JOIN语法"中进行了说明。以下是一个例子：

```
UPDATE items,month SET items.price=month.price

WHERE items.id=month.id;
```

以上的例子显示出了使用逗号操作符的内部联合，但是multiple-table UPDATE语句可以使用在SELECT语句中允许的任何类型的联合，比如LEFT JOIN。

注释：您不能把ORDER BY或LIMIT与multiple-table UPDATE同时使用。

在一个被更改的multiple-table UPDATE中，有些列被引用。您只需要这些列的UPDATE权限。有些列被读取了，但是没被修改。您只需要这些列的SELECT权限。

如果您使用的multiple-table UPDATE语句中包含带有外键限制的InnoDB表，则MySQL优化符处理表的顺序可能与上下层级关系的顺序不同。在此情况下，语句无效并被 回滚。同时，更新一个单一表，并且依靠ON UPDATE功能。该功能由InnoDB提供，用于对其它表进行相应的修改。请参见15.2.6.4节，"FOREIGN KEY约束"。

目前，您不能在一个子查询中更新一个表，同时从同一个表中选择。

13.3. MySQL实用工具语句

13.3.1. DESCRIBE语法（获取有关列的信息）
13.3.2. USE语法

13.3.1. DESCRIBE语法（获取有关列的信息）

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

DESCRIBE可以提供有关表中各列的信息。它是SHOW COLUMNS FROM的快捷方式。这些语句也可以显示语句，用于阅览。

见13.5.4.3节，"SHOW COLUMNS语法"。

*col_name*可以是一个列名称，或一个包含'%'和'_'的通配符的字符串，用于获得对于带有与字符串相匹配的名称的各列的输出。没有必要在引号中包含字符串，除非其中包含空格或其它特殊字符。

```
mysql> DESCRIBE city;

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

| Field      | Type     | Null | Key | Default | Extra          |

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

| Id         | int(11)  | NO   | PRI | NULL    | auto_increment |

| Name       | char(35) | NO   |     |         |                |

| Country    | char(3)  | NO   | UNI |         |                |

| District   | char(20) | YES  | MUL |         |                |

| Population | int(11)  | NO   |     | 0       |                |

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

5 rows in set (0.00 sec)
```

NULL字段指示是否NULL可以被存储在列中。

Key字段指示是否该列已编制索引。PRI的值指示该列是表的主键的一部分。UNI指示，该列是UNIQUE索引的一部分。MUL值指示，在列中某个给定值多次出现是允许的。

MUL将被显示在UNIQUE索引中，原因之一是多个列会组合成一个复合UNIQUE索引；尽管列的组合是唯一的，但每个列仍可以多次出现同一个给定值。注意，在复合索引中，只有索引最左边的列可以进入Key字段中。

默认字段指示，默认值被赋予该列。

Extra字段包含可以获取的与给定列有关的附加信息。在我们的例子中，Extra字段指示，Id列使用AUTO_INCREMENT关键词创建。

如果列类型与您预计的依据CREATE TABLE语句得出的列类型不同，则请注意，MySQL有时会改变列类型。请参见13.1.5.1节，"沉寂的列规格变更"。

DESCRIBE语句被设立出来，用于与Oracle相兼容。

SHOW CREATE TABLE和SHOW TABLE STATUS语句也可以提供有关表的信息。请参见13.5.4节，"SHOW语法"。

13.3.2. USE语法

```
USE db_name
```

USE *db_name*语句可以通告MySQL把*db_name*数据库作为默认（当前）数据库使用，用于后续语句。该数据库保持为默认数据库，直到语段的结尾，或者直到发布一个不同的USE语句：

```
mysql> USE db1;

mysql> SELECT COUNT(*) FROM mytable;   # selects from db1.mytable

mysql> USE db2;

mysql> SELECT COUNT(*) FROM mytable;   # selects from db2.mytable
```

使用USE语句为一个特定的当前的数据库做标记，不会阻碍您访问其它数据库中的表。下面的例子可以从db1数据库访问作者表，并从db2数据库访问编辑表：

```
mysql> USE db1;

mysql> SELECT author_name,editor_name FROM author,db2.editor

    ->        WHERE author.editor_id = db2.editor.editor_id;
```

USE语句被设立出来，用于与Sybase相兼容。

13.4. MySQL事务处理和锁定语句

13.4.1. START TRANSACTION, COMMIT和ROLLBACK语法
13.4.2. 不能回滚的语句
13.4.3. 会造成隐式提交的语句
13.4.4. SAVEPOINT和ROLLBACK TO SAVEPOINT语法
13.4.5. LOCK TABLES和UNLOCK TABLES语法
13.4.6. SET TRANSACTION语法
13.4.7. XA事务

MySQL通过SET AUTOCOMMIT, START TRANSACTION, COMMIT和ROLLBACK等语句支持本地事务（在给定的客户端连接中）。请参见13.4.1节，“START TRANSACTION, COMMIT和ROLLBACK语法”。XA事务支持还可以允许MySQL参与分布式事务。请参见13.4.7节，“XA事务”。

13.4.1. START TRANSACTION, COMMIT和ROLLBACK语法

```
START TRANSACTION | BEGIN [WORK]

COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]

ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]

SET AUTOCOMMIT = {0 | 1}
```

START TRANSACTION或BEGIN语句可以开始一项新的事务。COMMIT可以提交当前事务，是变更成为永久变更。ROLLBACK可以 回滚当前事务，取消其变更。SET AUTOCOMMIT语句可以禁用或启用默认的autocommit模式，用于当前连接。

自选的WORK关键词被支持，用于COMMIT和RELEASE，与CHAIN和RELEASE子句。CHAIN和RELEASE可以被用于对事务完成进行附加控制。Completion_type系统变量的值决定了默认完成的性质。请参见5.3.3节，“服务器系统变量”。

AND CHAIN子句会在当前事务结束时，立刻启动一个新事务，并且新事务与刚结束的事务有相同的隔离等级。RELEASE子句在终止了当前事务后，会让服务器断开与当前客户端的连接。包含NO关键词可以抑制CHAIN或RELEASE完成。如果completion_type系统变量被设置为一定的值，使连锁或释放完成可以默认进行，此时NO关键词有用。

默认情况下，MySQL采用autocommit模式运行。这意味着，当您执行一个用于更新（修改）表的语句之后，MySQL立刻把更新存储到磁盘中。

如果您正在使用一个事务安全型的存储引擎（如InnoDB, BDB或NDB簇），则您可以使用以下语句禁用autocommit模式：

```
SET AUTOCOMMIT=0;
```

通过把AUTOCOMMIT变量设置为零，禁用autocommit模式之后，您必须使用COMMIT把变更存储到磁盘中，或着如果您想要忽略从事务开始进行以来做出的变更，使用ROLLBACK。

如果您想要对于一个单一系列的语句禁用autocommit模式，则您可以使用START TRANSACTION语句：

```
START TRANSACTION;

SELECT @A:=SUM(salary) FROM table1 WHERE type=1;

UPDATE table2 SET summary=@A WHERE type=1;

COMMIT;
```

使用START TRANSACTION，autocommit仍然被禁用，直到您使用COMMIT或ROLLBACK结束事务为止。然后autocommit模式恢复到原来的状态。

BEGIN和BEGIN WORK被作为START TRANSACTION的别名受到支持，用于对事务进行初始化。START TRANSACTION是标准的SQL语法，并且是启动一个ad-hoc事务的推荐方法。BEGIN语句与BEGIN关键词的使用不同。BEGIN关键词可以启动一个BEGIN...END复合语句。后者不会开始一项事务。请参见20.2.7节，“BEGIN ... END复合语句”。

您也可以按照如下方法开始一项事务：

```
START TRANSACTION WITH CONSISTENT SNAPSHOT;
```

WITH CONSISTENT SNAPSHOT子句用于启动一个一致的读取，用于具有此类功能的存储引擎。目前，该子句只适用于InnoDB。该子句的效果与发布一个START TRANSACTION，后面跟一个来自任何InnoDB表的SELECT的效果一样。请参见15.2.10.4节，“一致的非锁定读”。

开始一项事务会造成一个隐含的UNLOCK TABLES被执行。

为了获得最好的结果，事务应只使用由单一事务存储引擎管理的表执行。否则，会出现以下问题：

- 如果您使用的表来自多个事务安全型存储引擎（例如InnoDB和BDB），并且事务隔离等级不是SERIALIZABLE，则有可能当一个事务提交时，其它正在进行中的、使用同样的表的事务将只会发生由第一个事务产生的变更。也就是，用混合引擎不能保证事务的原子性，并会造成不一致。（如果混合引擎事务不经常有，则您可以根据需要使用SET TRANSACTION ISOLATION LEVEL把隔离等级设置到SERIALIZABLE。）

- 如果您在事务中使用非事务安全型表，则对这些表的任何变更被立刻存储，不论autocommit模式的状态如何。

如果您在更新了事务中一个事务表之后，发布一个ROLLBACK语句，则会出现一个ER_WARNING_NOT_COMPLETE_ROLLBACK警告。对事务安全型表的变更被 回滚，但是对非事务安全型表没有变更。

每个事务被存储在一个组块中的二进制日志中，在COMMIT之上。被回滚的事务不被计入日志。（例外情况：对非事务表的更改不会被 回滚。如果一个被回滚的事务包括对非事务表的更改，则整个事务使用一个在末端的ROLLBACK语句计入日志，以确保对这些表的更改进行复制。）见5.11.3节，“二进制日志”。

您可以使用SET TRANSACTION ISOLATION LEVEL更改事务的隔离等级。请参见13.4.6节，“SET TRANSACTION语法”。

回滚可以慢速运行。在用户没有明确要求时，也可以进行回滚（例如，当错误发生时）。因此，在明确地和隐含的（ROLLBACK SQL命令）回滚时，SHOW PROCESSLIST会在Stage列中显示Rolling back，用于连接。

13.4.2. 不能回滚的语句

有些语句不能被回滚。通常，这些语句包括数据定义语言（DDL）语句，比如创建或取消数据库的语句，和创建、取消或更改表或存储的子程序的语句。

您在设计事务时，不应包含这类语句。如果您在事务的前部中发布了一个不能被回滚的语句，则后部的其它语句会发生错误，在这些情况下，通过发布ROLLBACK语句不能 回滚事务的全部效果。

13.4.3. 会造成隐式提交的语句

以下语句（以及同义词）均隐含地结束一个事务，似乎是在执行本语句前，您已经进行了一个COMMIT。

· ALTER FUNCTION, ALTER PROCEDURE, ALTER TABLE, BEGIN, CREATE DATABASE, CREATE FUNCTION, CREATE INDEX, CREATE PROCEDURE, CREATE TABLE, DROP DATABASE, DROP FUNCTION, DROP INDEX, DROP PROCEDURE, DROP TABLE, LOAD MASTER DATA, LOCK TABLES, RENAME TABLE, SET AUTOCOMMIT=1, START TRANSACTION, TRUNCATE TABLE, UNLOCK TABLES.

· 当当前所有的表均被锁定时，UNLOCK TABLES可以提交事务。

· CREATE TABLE, CREATE DATABASE DROP DATABASE, TRUNCATE TABLE, ALTER FUNCTION, ALTER PROCEDURE, CREATE FUNCTION, CREATE PROCEDURE, DROP FUNCTION和DROP PROCEDURE等语句会导致一个隐含提交。

· InnoDB中的CREATE TABLE语句被作为一个单一事务进行处理。这意味着，来自用户的ROLLBACK不会撤销用户在事务处理过程中创建的CREATE TABLE语句。

事务不能被嵌套。这是隐含COMMIT的结果。当您发布一个START TRANSACTION语句或其同义词时，该COMMIT被执行，用于任何当前事务。

13.4.4. SAVEPOINT和ROLLBACK TO SAVEPOINT语法

```
SAVEPOINT identifier

ROLLBACK [WORK] TO SAVEPOINT identifier

RELEASE SAVEPOINT identifier
```

InnoDB支持SQL语句SAVEPOINT, ROLLBACK TO SAVEPOINT, RELEASE SAVEPOINT和自选的用于ROLLBACK的WORK关键词。

SAVEPOINT语句用于设置一个事务保存点，带一个标识符名称。如果当前事务有一个同样名称的保存点，则旧的保存点被删除，新的保存点被设置。

ROLLBACK TO SAVEPOINT语句会向以命名的保存点回滚一个事务。如果在保存点被设置后，当前事务对行进行了更改，则这些更改会在 回滚中被撤销。但是，InnoDB不会释放被存储在保存点之后的存储器中的行锁定。（注意，对于新插入的行，锁定信息被存储在行中的事务ID承载；锁定没有被分开存储在存储器中。在这种情况下，行锁定在撤销中被释放。）在被命名的保存点之后设置的保存点被删除。

如果语句返回以下错误，则意味着不存在带有指定名称的保存点：

```
ERROR 1181: Got error 153 during ROLLBACK
```

RELEASE SAVEPOINT语句会从当前事务的一组保存点中删除已命名的保存点。不出现提交或 回滚。如果保存点不存在，会出现错误。

如果您执行COMMIT或执行不能命名保存点的ROLLBACK，则当前事务的所有保存点被删除。

13.4.5. LOCK TABLES和UNLOCK TABLES语法

```
LOCK TABLES
```

```
    tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE}
    [, tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE}] ...
UNLOCK TABLES
```

LOCK TABLES可以锁定用于当前线程的表。如果表被其它线程锁定，则造成堵塞，直到可以获取所有锁定为止。UNLOCK TABLES可以释放被当前线程保持的任何锁定。当线程发布另一个LOCK TABLES时，或当与服务器的连接被关闭时，所有由当前线程锁定的表被隐含地解锁。

表锁定只用于防止其它客户端进行不正当地读取和写入。保持锁定（即使是读取锁定）的客户端可以进行表层级的操作，比如DROP TABLE。

注意，下面是对事务表使用LOCK TABLES的说明：

- 在尝试锁定表之前，LOCK TABLES不是事务安全型的，会隐含地提交所有活性事务。同时，开始一项事务（例如，使用START TRANSACTION），会隐含地执行UNLOCK TABLES。（见13.4.3节，“会造成隐式提交的语句”。

- 对事务表（如InnoDB）使用LOCK TABLES的正确方法是，设置AUTOCOMMIT=0并且不能调用UNLOCK TABLES，直到您明确地提交事务为止。当您调用LOCK TABLES时，InnoDB会内部地取其自己的表锁定，MySQL取其自己的表锁定。InnoDB在下一个提交时释放其表锁定，但是，对于MySQL，要释放表锁定，您必须调用UNLOCK TABLES。您不应该让AUTOCOMMIT=1，因为那样的话，InnoDB会在调用LOCK TABLES之后立刻释放表锁定，并且很容易形成死锁定。注意，如果AUTOCOMMIT=1，我们根本不能获取InnoDB表锁定，这样就可以帮助旧的应用软件避免不必要的死锁定。

- ROLLBACK不会释放MySQL的非事务表锁定。

要使用LOCK TABLES，您必须拥有相关表的LOCK TABLES权限和SELECT权限。

使用LOCK TABLES的主要原因是仿效事务，或在更新表时加快速度。这将在后面进行更详细的解释。

如果一个线程获得对一个表地READ锁定，该线程（和所有其它线程）只能从该表中读取。如果一个线程获得对一个表的WRITE锁定，只有保持锁定的线程可以对表进行写入。其它的线程被阻止，直到锁定被释放时为止。

READ LOCAL和READ之间的区别是，READ LOCAL允许在锁定被保持时，执行非冲突性INSERT语句（同时插入）。但是，如果您正打算在MySQL外面操作数据库文件，同时您保持锁定，则不能使用READ LOCAL。对于InnoDB表，READ LOCAL与READ相同。

当您使用LOCK TABLES时，您必须锁定您打算在查询中使用的所有的表。虽然使用LOCK TABLES语句获得的锁定仍然有效，但是您不能访问没有被此语句锁定的任何的表。同时，您不能在一次查询中多次使用一个已锁定的表——使用别名代替，在此情况下，您必须分别获得对每个别名的锁定。

```
mysql> LOCK TABLE t WRITE, t AS t1 WRITE;

mysql> INSERT INTO t SELECT * FROM t;

ERROR 1100: Table 't' was not locked with LOCK TABLES

mysql> INSERT INTO t SELECT * FROM t AS t1;
```

如果您的查询使用一个别名引用一个表，那么您必须使用同样的别名锁定该表。如果没有指定别名，则不会锁定该表。

```
mysql> LOCK TABLE t READ;

mysql> SELECT * FROM t AS myalias;

ERROR 1100: Table 'myalias' was not locked with LOCK TABLES
```

相反的，如果您使用一个别名锁定一个表，您必须使用该别名在您的查询中引用该表。

```
mysql> LOCK TABLE t AS myalias READ;

mysql> SELECT * FROM t;

ERROR 1100: Table 't' was not locked with LOCK TABLES

mysql> SELECT * FROM t AS myalias;
```

WRITE锁定通常比READ锁定拥有更高的优先权，以确保更新被尽快地处理。这意味着，如果一个线程获得了一个READ锁定，则另一个线程会申请一个WRITE锁定，后续的READ锁定申请会等待，直到WRITE线程获得锁定并释放锁定。您可以使用LOW_PRIORITY WRITE锁定来允许其它线程在该线程正在等待WRITE锁定时获得READ锁定。只有当您确定最终将有一个时机，此时没有线程拥有READ锁定时，您才应该使用LOW_PRIORITY WRITE锁定。

LOCK TABLES按照如下方式执行：

1. 按照内部定义的顺序，对所有要被锁定的表进行分类。从用户的角度，此顺序是未经定义的。
2. 如果使用一个读取和一个写入锁定对一个表进行锁定，则把写入锁定放在读取锁定之前。
3. 一次锁定一个表，直到线程得到所有锁定为止。

该规则确保表锁定不会出现死锁定。但是，对于该规则，您需要注意其它的事情：

如果您正在对一个表使用一个LOW_PRIORITY WRITE锁定，这只意味着，MySQL等待特定的锁定，直到没有申请READ锁定的线程时为止。当线程已经获得WRITE锁定，并正在等待得到锁定表清单中的用于下一个表的锁定时，所有其它线程会等待WRITE锁定被释放。如果这成为对于应用程序的严重的问题，则您应该考虑把部分表转化为事务安全型表。

您可以安全地使用KILL来结束一个正在等待表锁定的线程。请参见13.5.5.3节，“KILL语法”。

注意，您不能使用INSERT DELAYED锁定任何您正在使用的表，因为，在这种情况下，INSERT由另一个线程执行。

通常，您不需要锁定表，因为所有的单个UPDATE语句都是原子性的；没有其它的线程可以干扰任何其它当前正在执行的SQL语句。但是，在几种情况下，锁定表会有好处：

- 如果您正在对一组MyISAM表运行许多操作，锁定您正在使用的表，可以快很多。锁定MyISAM表可以加快插入、更新或删除的速度。不利方面是，没有线程可以更新一个用READ锁定的表（包括保持锁定的表），也没有线程可以访问用WRITE锁定的表（除了保持锁定的表以外）。

有些MyISAM操作在LOCK TABLES之下更快的原因是，MySQL不会清空用于已锁定表的关键缓存，直到UNLOCK TABLE被调用为止。通常，关键缓存在每个SQL语句之后被清空。

- 如果您正在使用MySQL中的一个不支持事务的存储引擎，则如果您想要确定在SELECT和UPDATE之间没有其它线程，您必须使用LOCK TABLES。本处所示的例子要求LOCK TABLES，以便安全地执行：

```
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;
```

如果没有LOCK TABLES，有可能另一个线程会在执行SELECT和UPDATE语句之间在trans表中插入一个新行。

通过使用相对更新（UPDATE customer SET *value*=*value*+*new_value*）或LAST_INSERT_ID()函数，您可以在许多情况下避免使用LOCK TABLES。请参见1.8.5.3节，“事务和原子操作”。

通过使用用户层级的顾问式锁定函数GET_LOCK()和RELEASE_LOCK()，您也可以在有些情况下避免锁定表。这些锁定被保存在服务器中的一个混编表中，使用pthread_mutex_lock() 和pthread_mutex_unlock()，以加快速度。请参见12.9.4节，“其他函数”。

要了解更多有关锁定规则的说明，请参见7.3.1节，“锁定方法”。

您可以使用FLUSH TABLES WITH READ LOCK语句锁定位于所有带有读取锁定的数据库中的所有表。请参见13.5.5.2节，“FLUSH语法”。如果您有一个可以及时拍摄快照的文件系统，比如Veritas，这是获得备份的一个非常方便的方式。

注释：如果您对一个已锁定的表使用ALTER TABLE，该表可能会解锁。请参见A.7.1节，“与ALTER TABLE有关的问题”。

13.4.6. SET TRANSACTION语法

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL

{ READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }
```

本语句用于设置事务隔离等级，用于下一个事务，或者用于当前会话。

在默认情况下，SET TRANSACTION会为下一个事务（还未开始）设置隔离等级。如果您使用GLOBAL关键词，则语句会设置全局性的默认事务等级，用于从该点以后创建的所有新连接。原有的连接不受影响。要进行此操作，您需要SUPER权限。使用SESSION关键测可以设置默认事务等级，用于对当前连接执行的所有将来事务。

要了解对每个InnoDB事务隔离等级的描述，请参见15.2.10.3节，“InnoDB和TRANSACTION ISOLATION LEVEL”。InnoDB支持MySQL 5.1中的各个等级。默认的等级是REPEATABLE READ。

您可以使用--transaction-isolation选项，对**mysqld**设置初始的默认全局隔离等级。请参见5.3.1节，“**mysqld**命令行选项”。

13.4.7. XA事务

13.4.7.1. XA事务SQL语法
13.4.7.2. XA事务状态

对于InnoDB存储引擎，可以获得对XA事务的支持。MySQL XA的执行依据X/Open CAE文件*Distributed Transaction Processing: The XA Specification*。本文件由Open Group出版，可以从http://www.opengroup.org/public/pubs/catalog/c193.htm获取。在I.5节，“对XA事务的限制”对当前XA执行的限制进行了描述。

在客户端方面，没有特殊要求。连接MySQL服务器的XA接口由以XA关键词开头的SQL语句组成。MySQL客户端必须能发送SQL语句，并能理解XA语句接口的语义，但是它们不需要被链接到特定的MySQL客户库上。

当前，在MySQL连接器当中，MySQL连接器/J 5.0.0直接支持XA（也就是，通过一个可以控制XA SQL语句接口的等级接口）。

XA支持分布式的事务，具备能力，让多个独立的事务资源参加全局的事务。事务资源通常是RDBMSs，不过也

可以是其它种类的资源。

一个全局事务会涉及到多个行动，这些行动本身是事务性的。不过，所有行动都必须作为一个群组成功完成，或者作为一个群组被回滚。实际上，这会延伸ACID性质，“提高等级”，这样，多个ACID事务就可以一起执行，相当于也拥有ACID性质的全局操作的组件。（但是，对于一个分布式事务，您必须使用SERAILIZABLE隔离等级，以实现ACID性质。对于一个非分布式事务，使用REPEATABLE READ就足够了。但是对于分布式事务，使用REPEATABLE READ是不够的。）

分布式事务的部分示例：

- 应用程序相当于一个把消息传递服务和RDBMS组合在一起的整合工具。应用程序可以确保，所有进行消息发送、回收和处理的事务（同时包含一个事务数据库）均在一个全局事务中发生。您可以把这看作是“事务电子邮件。”

- 应用程序执行的行动会涉及到不同数据库服务器，比如MySQL服务器和Oracle服务器（或多个MySQL服务器）。涉及到多个服务器的行动必须作为一个全局事务的一部分发生，而不是作为针对每个服务器的独立的本地事务发生。

- 银行把帐户信息保存在RDBMS中，并通过自动取款机（ATMs）分发和收取欠款。必须要确保ATM行动被正确地反映到帐户中，但是这不能只由RDBMS单独完成。全局事务管理器会整合ATM和数据库资源，以确保财务事务的整体一致性。

使用全局事务的应用程序涉及一个或多个资源管理器和一个事务管理器：

- 资源管理器（RM）用于提供通向事务资源的途径。数据库服务器是一种资源管理器。该管理器必须可以提交或 回滚由RM管理的事务。

- 事务管理器（TM）用于协调作为一个全局事务一部分的事务。TM与管理每个事务的RMs进行通讯。一个全局事务中各个单个事务均是全局事务的“分支”。全局事务和各分支通过一种命名方法进行标识。这种命名方法在后面进行讲述。

MySQL执行XA MySQL时，MySQL服务器相当于一个用于管理全局事务中的XA事务的资源管理器。与MySQL服务器连接的客户端相当于事务管理器。

要执行一个全局事务，必须知道涉及到了哪些组件，并且把每个组件引到一点，在此时，组件可以被提交或回滚时。根据每个组件报告的有关组件效能的内容，这些组件必须作为一个原子性群组全部提交或 回滚。即，要么所有的组件必须提交，要么所有的组件必须回滚。要管理一个全局事务，必须要考虑任何组件或连接网络可能会故障。

用于执行全局事务的过程使用两阶段提交（2PC），发生时间在由全局事务的分支进行的行动已经被执行之后。

1. 在第一阶段，所有的分支被预备好。即，它们被TM告知要准备提交。通常，这意味着用于管理分支的每个RM会记录对于被稳定保存的分支的行动。分支指示是否它们可以这么做。这些结果被用于第二阶段。

2. 在第二阶段，TM告知RMs是否要提交或 回滚。如果在预备分支时，所有的分支指示它们将能够提交，则所有的分支被告知要提交。如果在预备时，有任何分支指示它将不能提交，则所有分支被告知 回滚。

在有些情况下，一个全局事务可能会使用一阶段提交（1PC）。例如，当一个事务管理器发现，一个全局事务只由一个事务资源组成（即，单一分支），则该资源可以被告知同时进行预备和提交。

13.4.7.1. XA事务SQL语法

要在MySQL中执行XA事务，应使用以下语句：

```
XA {START|BEGIN} xid [JOIN|RESUME]

XA END xid [SUSPEND [FOR MIGRATE]]

XA PREPARE xid

XA COMMIT xid [ONE PHASE]

XA ROLLBACK xid

XA RECOVER
```

对于XA START，JOIN和RESUME子句不被支持。

对于XA END，SUSPEND [FOR MIGRATE]子句不被支持。

每个XA语句以XA关键词为开头，多数语句要求一个xid值。 xid是一个XA事务标识符。它指示该语句适用于哪个事务。xid值由客户端提供，或由MySQL服务器生成。xid值包含一到三个部分：

```
xid: gtrid [, bqual [, formatID ]]
```

gtrid是一个全局事务标识符，bqual是一个分支限定符，formatID是一个数字，用于标识由gtrid和bqual值使用的格式。根据语法的表示，bqual和formatID是自选的。如果没有给定，默认的bqual值是''。如果没有给定，默认的fromatID值是1。

gtrid和bqual必须为字符串文字，每个的长度最多为64字节（不是字符）。gtrid和bqual可以用多种方法指定。您可以使用带引号的字符串('ab')，十六进制字符串(0x6162, X'ab')，或位值(b'*nnnn*')。

formatID是一个无符号的整数。

通过MySQL服务器的带下划线的XA支持子程序，gtrid和bqual值被理解为以字节为单位。但是，在包含XA语句的SQL语句正在被分析的同时，服务器会去操作一些特定的字符集。为了安全，把gtrid和bqual作为十六进制字符串写入。

通常，xid值由事务管理器生成。由一个TM生成的值必须与由其它TMs生成的值不同。一个给定的TM必须能识别自己的xid值。这些值位于由XA RECOVER语句返回的值清单中。

XA START xid用于启动一个带给定xid值的XA事务。每个XA事务必须有一个唯一的xid值，因此该值当前不能被其它的XA事务使用。使用gtrid和bqual值评估唯一性。所有下列的用于XA事务的XA语句必须使用与XA START语句中给定的相同的xid值进行指定。如果您使用这些语句，但是指定的xid值与部分原有的XA事务不对应的话，会发生错误。

一项或多项XA事务可以是同一个全局事务的一部分。在一个给定的全局事务中的所有XA事务必须在xid值中使用同样的gtrid值。出于这个原因，gtrid值必须为全局唯一的，这样，有关一个给定的XA事务是哪个全局事务的一部分的问题就不会含糊不清。对于一个全局事务中的每个XA事务，xid值中的bqual部分必须不一样。（bqual值应不一样，这个要求是当前执行MySQL XA的一个限制条件。这不是XA规约的一部分。）

对于在MySQL服务器上的处于PREPARED状态的XA事务，XA RECOVER语句会返回信息。（见13.4.7.2节，“XA事务状态”.。）输出包括一个行，该行用于服务器上的每个这类XA事务，不论是哪个客户端启动了它。

XA RECOVER输出行看上去像这样（例如，xid值包括'abc', 'def'和 7等部分）：

```
mysql> XA RECOVER;
+----------+--------------+--------------+--------+
| formatID | gtrid_length | bqual_length | data   |
+----------+--------------+--------------+--------+
|        7 |            3 |            3 | abcdef |
+----------+--------------+--------------+--------+
```

输出列有以下意义：

- formatID是事务xid的formatID部分
- gtrid_length是xid的gtrid部分的长度，以字节为单位
- bqual_length是xid的bqual部分的长度，以字节为单位
- data是xid的gtrid部分和bqual部分的串联

13.4.7.2. XA事务状态

XA事务在以下状态下进展：

1. 使用XA START来启动一个XA事务，并把它放入ACTIVE状态。
2. 对于一个ACTIVE XA事务，发布构成事务的SQL语句，然后发布一个XA END语句。XA END把事务放入IDLE状态。
3. 对于一个IDLE XA事务，您可以发布一个XA PREPARE语句或一个XA COMMIT…ONE PHASE语句：

- XA PREPARE把事务放入PREPARED状态。在此点上的XA RECOVER语句将在其输出中包括事务的xid值，因为XA RECOVER会列出处于PREPARED状态的所有XA事务。
- XA COMMIT…ONE PHASE用于预备和提交事务。xid值将不会被XA RECOVER列出，因为事务终止。

4. 对于一个PREPARED XA事务，您可以发布一个XA COMMIT语句来提交和终止事务，或者发布XA ROLLBACK来回滚并终止事务。

下面是一个简单的XA事务，该事务把一行作为一个全局事务的一部分插入一个表中。

```
mysql> XA START 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO mytable (i) VALUES(10);
Query OK, 1 row affected (0.04 sec)

mysql> XA END 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA PREPARE 'xatest';
```

```
Query OK, 0 rows affected (0.00 sec)


mysql> XA COMMIT 'xatest';

Query OK, 0 rows affected (0.00 sec)
```

根据一个给定的客户端连接的语境，XA事务和本地（非XA）事务互相排斥。举例说明，如果已经发布了XA START来开始一项XA事务，则本地事务不会被启动，直到XA事务已经被提交或被 回滚为止。相反的，如果已经使用START TRANSACTION启动一个本地事务，则XA语句不能被使用，直到该事务被提交或被 回滚为止。

13.5. 数据库管理语句

13.5.1. 账户管理语句
13.5.2. 表维护语句
13.5.3. SET语法
13.5.4. SHOW语法
13.5.5. 其它管理语句

13.5.1. 账户管理语句

13.5.1.1. CREATE USER语法
13.5.1.2. DROP USER语法
13.5.1.3. GRANT和REVOKE语法
13.5.1.4. RENAME USER语法
13.5.1.5. SET PASSWORD语法

13.5.1.1. CREATE USER语法

```
CREATE USER user [IDENTIFIED BY [PASSWORD] 'password']

    [, user [IDENTIFIED BY [PASSWORD] 'password']] ...
```

CREATE USER用于创建新的MySQL账户。要使用CREATE USER，您必须拥有mysql数据库的全局CREATE USER权限，或拥有INSERT权限。对于每个账户，CREATE USER会在没有权限的mysql.user表中创建一个新记录。如果 账户已经存在，则出现错误。

使用自选的IDENTIFIED BY子句，可以为账户给定一个密码。user值和 密码的给定方法和GRANT语句一样。特别是，要在纯文本中指定密码，需忽略PASSWORD关键词。要把 密码指定为由PASSWORD()函数返回的混编值，需包含关键字PASSWORD。请参见13.5.1.3节，“GRANT和REVOKE语法”。

13.5.1.2. DROP USER语法

```
DROP USER user [, user] ...
```

DROP USER语句用于删除一个或多个MySQL账户。要使用DROP USER，您必须拥有mysql数据库的全局CREATE USER权限或DELETE权限。使用与GRANT或REVOKE相同的格式为每个 账户命名；例如，'jeffrey'@'localhost'。 账户名称的用户和主机部分与用户表记录的User和Host列值相对应。

使用DROP USER，您可以取消一个账户和其权限，操作如下：

```
DROP USER user;
```

该语句可以删除来自所有授权表的帐户权限记录。

要点：DROP USER不能自动关闭任何打开的用户对话。而且，如果用户有打开的对话，此时取消用户，则命令不会生效，直到用户对话被关闭后才生效。一旦对话被关闭，用户也被取消，此用户再次试图登录时将会失败。这是有意设计的。

13.5.1.3. GRANT和REVOKE语法

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)]] ...
    ON [object_type] {tbl_name | * | *.* | db_name.*}
    TO user [IDENTIFIED BY [PASSWORD] 'password']
        [, user [IDENTIFIED BY [PASSWORD] 'password']] ...
    [REQUIRE
        NONE |
        [{SSL| X509}]
        [CIPHER 'cipher' [AND]]
        [ISSUER 'issuer' [AND]]
        [SUBJECT 'subject']]
    [WITH with_option [with_option] ...]

object_type =
    TABLE
  | FUNCTION
  | PROCEDURE

with_option =
    GRANT OPTION
  | MAX_QUERIES_PER_HOUR count
  | MAX_UPDATES_PER_HOUR count
  | MAX_CONNECTIONS_PER_HOUR count
  | MAX_USER_CONNECTIONS count

REVOKE priv_type [(column_list)] [, priv_type [(column_list)]] ...
    ON [object_type] {tbl_name | * | *.* | db_name.*}
    FROM user [, user] ...

REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

GRANT和REVOKE语句允许系统管理员创建MySQL用户 账户，授予权限和撤销权限。

MySQL账户信息存储在mysql数据库的表中。在第5章：数据库管理中对本数据库和访问控制系统进行了详尽的讨论。要了解更多详细信息，您应该查询此章。

如果授权表拥有含有mixed-case数据库或表名称的权限记录，并且lower_case_table_names系统变量已设置，则不能使用REVOKE撤销权限，必须直接操纵授权表。（当lower_case_table_names已设置时，GRANT将不会创建此类记录，但是此类记录可能已经在设置变量之前被创建了。）

授予的权限可以分为多个层级：

- 全局层级

全局权限适用于一个给定服务器中的所有数据库。这些权限存储在mysql.user表中。GRANT ALL ON *.*和REVOKE ALL ON *.*只授予和撤销全局权限。

- 数据库层级

数据库权限适用于一个给定数据库中的所有目标。这些权限存储在mysql.db和mysql.host表中。GRANT ALL ON *db_name*.*和REVOKE ALL ON *db_name*.*只授予和撤销数据库权限。

- 表层级

表权限适用于一个给定表中的所有列。这些权限存储在mysql.talbes_priv表中。GRANT ALL ON *db_name.tbl_name*和REVOKE ALL ON *db_name.tbl_name*只授予和撤销表权限。

- 列层级

列权限适用于一个给定表中的单一列。这些权限存储在mysql.columns_priv表中。当使用REVOKE时，您必须指定与被授权列相同的列。

- 子程序层级

CREATE ROUTINE, ALTER ROUTINE, EXECUTE和GRANT权限适用于已存储的子程序。这些权限可以被授予为全局层级和数据库层级。而且，除了CREATE ROUTINE外，这些权限可以被授予为子程序层级，并存储在mysql.procs_priv表中。

当后续目标是一个表、一个已存储的函数或一个已存储的过程时，*object_type*子句应被指定为TABLE、FUNCTION或PROCEDURE。当从旧版本的MySQL升级时，要使用本子句，您必须升级您的授权表。请参见2.10.2节，“升级授权表”。

要使用GRANT或REVOKE，您必须拥有GRANT OPTION权限，并且您必须用于您正在授予或撤销的权限。

要撤销所有权限，需使用以下语法。此语法用于取消对于已命名的用户的所有全局层级、数据库层级、表层级和列层级的权限。

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

要使用本REVOKE语法，您必须拥有mysql数据库的全局CREATE USER权限或UPDATE权限。

对于GRANT和REVOKE语句，priv_type可以被指定为以下任何一种：

权限	意义
ALL [PRIVILEGES]	设置除GRANT OPTION之外的所有简单权限
ALTER	允许使用ALTER TABLE
ALTER ROUTINE	更改或取消已存储的子程序
CREATE	允许使用CREATE TABLE
CREATE ROUTINE	创建已存储的子程序
CREATE TEMPORARY TABLES	允许使用CREATE TEMPORARY TABLE
CREATE USER	允许使用CREATE USER, DROP USER, RENAME USER和REVOKE ALL PRIVILEGES。
CREATE VIEW	允许使用CREATE VIEW
DELETE	允许使用DELETE
DROP	允许使用DROP TABLE
EXECUTE	允许用户运行已存储的子程序
FILE	允许使用SELECT...INTO OUTFILE和LOAD DATA INFILE
INDEX	允许使用CREATE INDEX和DROP INDEX
INSERT	允许使用INSERT
LOCK TABLES	允许对您拥有SELECT权限的表使用LOCK TABLES
PROCESS	允许使用SHOW FULL PROCESSLIST
REFERENCES	未被实施
RELOAD	允许使用FLUSH
REPLICATION CLIENT	允许用户询问从属服务器或主服务器的地址

REPLICATION SLAVE	用于复制型从属服务器（从主服务器中读取二进制日志事件）
SELECT	允许使用SELECT
SHOW DATABASES	SHOW DATABASES显示所有数据库
SHOW VIEW	允许使用SHOW CREATE VIEW
SHUTDOWN	允许使用**mysqladmin shutdown**
SUPER	允许使用CHANGE MASTER, KILL, PURGE MASTER LOGS和SET GLOBAL语句，**mysqladmin debug**命令；允许您连接（一次），即使已达到max_connections。
UPDATE	允许使用UPDATE
USAGE	"无权限"的同义词
GRANT OPTION	允许授予权限

当从旧版本的MySQL升级时，要使用EXECUTE, CREATE VIEW, SHOW VIEW, CREATE USER, CREATE ROUTINE和ALTER ROUTINE权限，您必须首先升级您的授权表。请参见2.10.2节，"升级授权表"。

REFERENCES权限目前未被使用。

当您想要创建一个没有权限的用户时，可以指定USAGE。

使用SHOW GRANTS来确定帐户拥有什么权限。请参见13.5.4.10节，"SHOW GRANTS语法"。

您可以通过使用ON *.*语法赋予全局权限，或通过使用ON *db_name*.*语法赋予数据库层级权限。如果您指定了ON *并且您已经选择了一个默认数据库，则权限被赋予到这个数据库中。（警告：如果您指定了ON *同时您没有选择一个默认数据库，则权限是全局的。）

FILE, PROCESS, RELOAD, REPLICATION CLIENT, REPLICATION SLAVE, SHOW DATABASES, SHUTDOWN和SUPER权限是管理性权限，只能进行全局授权（使用ON *.*语法）。

其它权限可以被全局授权，或被赋予为其它层级。

对于一个表，您可以指定的priv_type值只能是SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, GRANT OPTION, INDEX和ALTER。

对于一个列（也就是，当您使用一个column_list子句时），您可以指定的priv_type值只能是SELECT, INSERT和UPDATE。

在子程序层级，您可以指定的priv_type值只能是ALTER ROUTINE, EXECUTE和GRANT OPTION。CREATE ROUTINE不是一个子程序层级的权限，因为您必须拥有此权限，才能创建一个子程序。

对于全局、数据库、表和子程序层级，GRANT ALL只能赋予在您正在授权的层级中存在的权限。例如，如果您使用GRANT ALL ON *db_name*.*，这是一个数据库层级语句，因此不会授予全局权限，如FILE等。

MySQL允许您对不存在的数据库目标授予权限。在此情况下，将被授予的权限必须包括CREATE权限。这个性质是有意设计的，目的是允许数据库管理员为将在此后被创建的数据库目标预备用户 账户和权限。

要点：当您取消一个表或数据库时，MySQL不会自动撤销任何权限。但是，如果您取消一个子程序，则被赋予该子程序的所有子程序层级的权限都被撤销。

注意：GRANT语句用于在全局层级或数据库层级赋予权限。当在GRANT语句中指定数据库名称时，允许使用'_'和'%'通配符。这意味着，如果您想要使用'_'字符作为一个数据库名称的一部分，您应该在GRANT语句中指定它为'_'，以防止用户可以访问其它符合此通配符格式的数据库；例如，GRANT ... ON `foo\_bar`.* TO ...。

为了接纳对来自任意主机的用户授权的权利，MySQL支持以*user_name*@*host_name*的形式指定user值。如果一个user_name或host_name与一个不加引号的标识符一样是合法的，那么您不需要对它加引号。不过，要指定一个包含特殊字符（如'-'）的user_name字符串，或一个包含特殊字符或通配字符（如'%'），则引号是必要的；例如，'test-user'@'test-hostname'。分别对username和hostname加引号。

您可以在hostname中指定通配符。例如*user_name*@'%.loc.gov'适用于在loc.gov域中的任何主机的user_name。同时*user_name*@'144.155.166.%'适用于144.155.166 C级子网中的任何主机的user_name。

简单形式user_name是*user_name*@'%'的同义词。

MySQL不支持usernames中的通配符。通过把带有User=''的登录项插入到mysql.user表中，或通过使用GRANT语句创建一个带有空名称的用户，可以定义匿名用户：

```
mysql> GRANT ALL ON test.* TO ''@'localhost' ...
```

当把带引号的值是，需使用反勾号(`)为数据库、表、列和子程序名称加引号。使用单引号('')为hostnames、usernames和 密码加引号。

警告：如果您允许匿名用户连接到MySQL服务器，则您应该同时向所有本地用户授予*user_name*@localhost权限。否则，当有名称的用户试图从本地机器登录MySQL服务器时，mysql.user表中的用于localhost的匿名用户帐户会被使用。

您可以通过执行以下查询来确定是否这适合于您。以下查询列举了所有匿名用户：

```
mysql> SELECT Host, User FROM mysql.user WHERE User='';
```

如果您想要删除本地匿名用户账户，以避免出现刚才谈到的问题，则需使用以下语句：

```
mysql> DELETE FROM mysql.user WHERE Host='localhost' AND User='';

mysql> FLUSH PRIVILEGES;
```

GRANT支持最长为60个字符的hostnames。数据库、表、列和子程序名称最长可为64个字符。Usernames最长可为16个字符。 注释：不能通过更改mysql.user表来改变usernames的允许长度。如果试图这么做，会导致出现不可预见的问题，可能会造成用户无法登录MySQL服务器。除了采用由MySQL公司提供的用于升级MySQL服务器的mysql_fix_privilege_tables原稿之外，请您不要以任何方式变更授权表。

对于表或列的权限是作为各个权限层级的逻辑OR权限被附加形成的。例如，如果mysql.user表指定一个用户拥有全局SELECT权限，则该权限不能被数据库、表或列层级的登录项定义。

可以按下列方法计算列权限：

```
global privileges

OR (database privileges AND host privileges)

OR table privileges

OR column privileges
```

在多数情况下，您只在一个权限层级下向用户授予权利，所以寿命通常不是那么复杂。有关权限检查规程的细节，请参见5.7节，“MySQL访问权限系统”。

如果您对一个在mysql.user表中不存在的username/hostname组合授予权限，则增加一个登录项并保持在此处，直到使用DELETE语句删除为止。换句话说，GRANT可以创建用户表登录项，但是REVOKE不会取消它们；您必须使用DROP USER或DELETE明确地操作。

如果创建了一个新的用户，或者如果您拥有全局授权权限，则用户密码被设置为由IDENTIFIED BY子句指定的密码（如果给定了一个）。如果用户已拥有了一个密码，则此密码被新密码替代。

警告：如果您创建了一个新用户，但是不指定IDENTIFIED BY子句，则用户没有 密码。这是很不安全的。不过，您可以启用NO_AUTO_CREATE_USER SQL模式，来防止GRANT创建一个新用户（否则GRANT会这么做），除非给定了IDENTIFIED BY来为新用户提供一个密码。

使用SET PASSWORD语句也可以设置密码。请参见13.5.1.5节，“SET PASSWORD语法”。

在IDENTIFIED BY子句中，密码应被作为文字密码只被给定。没有必要使用PASSWORD()函数，因为该函数用于SET PASSWORD语句。例如：

```
GRANT ... IDENTIFIED BY 'mypass';
```

如果您不想以明白的文字发送密码，并且您知道PASSWORD()返回给密码的混编值，则您可以指定混编值，前面加入关键词PASSWORD：

```
GRANT ...
IDENTIFIED BY PASSWORD '*6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4';
```

在一个C程序中，您可以通过使用make_scrambled_password() C API函数得到混编值。

如果您为一个数据库授予权限，则在mysql.db表中，会根据需要创建登录项。如果使用REVOKE删除了所有的数据库权限，则本登录项被删除。

如果一个用户不拥有表权限，则当用户申请表清单时（例如，使用SHOW TABLES语句），表名称不显示。

SHOW DATABASES权限允许账户通过发布SHOW DATABASE语句来观看数据名称。不拥有此权限的账户只能看到他们拥有部分权限的数据库，并且如果使用--skip-show-database选项启动服务器，则根本不能使用本语句。

WITH GRANT OPTION子句给予用户能力，可以在指定的权限层级，向其它用户给定其拥有的任何权限。您应该留心您给予了谁GRANT OPTION权限，因为拥有不同权限的两个用户可以联合使用权限！

您不能向其它用户授予您自己没有的权限；GRANT OPTION权限只允许您赋予您自己拥有的权限。

要注意，当您在某个特定权限层级向一个用户授予GRANT OPTION权限时，用户拥有的该层级的任何权限（或未来将被给定的权限）也可以由该用户授予。假设您向一个用户赋予了数据库INSERT权限。如果您然后赋予数据库SELECT权限，并指定了WITH GRANT OPTION，则该用户不仅可以向其它用户给予SELECT权限，还可以给予INSERT。如果您然后向用户授予数据库UPDATE权限，则用户可以授予INSERT, SELECT和UPDATE。

您不应该向一个常规用户授予ALTER权限。如果您这么做，则该用户可以尝试通过对表重新命名来破坏授权系统！

The MAX_QUERIES_PER_HOUR *count*, MAX_UPDATES_PER_HOUR *count*, and MAX_CONNECTIONS_PER_HOUR *count* options limit the number of queries, updates, and logins a user can perform during any given one-hour period. If *count* is 0 (the default), this means that there is no limitation for that user. MAX_QUERIES_PER_HOUR *count*, MAX_UPDATES_PER_HOUR *count*和MAX_CONNECTIONS_PER_HOUR *count*选项限制了在任何给定的一小时期间，用户可以执行的查询、更新和登录的数目。如果count是0（默认值），这意味着，对该用户没有限制。

MAX_USER_CONNECTIONS *count*选项限制了账户可以同时进行的连接的最大数目。如果count是0（默认值），则max_user_connections系统可以决定该 账户同时连接的数目。

注释：要对一个原有的用户指定任何这类资源限制型选项，同时又不影响原有的权限，需使用GRANT USAGE ON *.* ... WITH MAX_...。

见5.8.4节，“限制账户资源”。

除了根据username和密码进行常规鉴定外，MySQL还可以检查X509证明属性。要为MySQL账户指定与SSL有关的选项，需使用GRANT语句的REQUIRE子句。（要了解有关在MySQL中使用SSL的背景信息，请参见5.8.7节，“使用安全连接”。）

对于一个给定的账户，有多种可能性可以限制连接类型：

- 如果账户没有SSL或X509要求，并且如果username和 密码是有效的，则允许不加密连接。但是，如果客户端有正确的证明和关键文件，则根据客户端的选择，也可以使用加密连接。

- REQUIRE SSL选项用于告知服务器，对于该账户只允许SSL加密连接。注意，如果有允许任何非SSL连接的访问控制记录，则本选项可以被忽略。

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    -> IDENTIFIED BY 'goodsecret' REQUIRE SSL;
```

- REQUIRE X509意味着客户端必须拥有一个有效证明，除非不需要确切的证明、发布者和主题。唯一的要求是，应可以使用CA证明其中之一来验证签名。

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    -> IDENTIFIED BY 'goodsecret' REQUIRE X509;
```

- REQUIRE ISSUER '*issuer*'用于对连接尝试进行限定，客户端必须出示一个由CA’issuer’发布的有效的X509证明。如果客户端出示的证明是有效的，但是有一个不同的发布者，则服务器会拒绝连接。使用X509证明就意味着要加密，所以在这种情况下，SSL选项是不必要的。

```
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@example.com';
```

注意，ISSUER值应被作为一个单一字符串输入。

- REQUIRE SUBJECT '*subject*'用于对连接尝试进行限定，客户端必须出示一个包含主题subject的有效的X509证明。如果客户端出示的证明是有效的，但是有一个不同的主题，则服务器会拒绝连接。

```
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@example.com';
```

注意，SUBJECT值应被作为一个单一字符串输入。

- 需要REQUIRE CIPHER '*cipher*'来确认使用了密码和足够长度的关键字。如果使用了采用短型加密关键字的旧算法，SSL本身会比较脆弱。使用本选项，您可以要求使用特定的密码方法来许可一个连接。

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    -> IDENTIFIED BY 'goodsecret'
    -> REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

SUBJECT, ISSUER和CIPHER选项可以在REQUIRE子句中结合，如下：

```
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@example.com'
    -> AND ISSUER '/C=FI/ST=Some-State/L=Helsinki/
       O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com'
    -> AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

注意，SUBJECT和ISSUER值各自应被作为一个单一字符串输入。

在REQUIRE各选项之间，AND关键词是自选的。

选项的顺序无所谓，但是选项不能被指定两次。

当mysqld启动后，所有的权限被读入存储器中。要了解详细说明，请参见5.7.7节，“权限更改何时生效”。

注意，如果您正在使用表权限或列权限，即使只对一个用户使用，服务器也会对所有用户检查表权限和列权限，这会略微降低MySQL的速度。与此类似，如果您对某些用户限制查询、更新或连接的数目，则服务器必须监测这些值。

标准SQL版本和MySQL版本的GRANT之间的最大区别是：

- 在MySQL中，权限与hostname和username的组合有关，与 单一的username无关。
- 标准SQL不拥有全局层级或数据库层级权限，也不支持MySQL支持的所有权限类型。
- MySQL不支持标准SQL TRIGGER或UNDER权限。
- 标准SQL权限以一种分等级的方式进行组织。如果您取消一个用户，则用户被授予的所有权限都被撤销。在MySQL中，如果您使用DROP USER，也会如此。请参见13.5.1.2节，“DROP USER语法”。
- 在标准SQL中，当您取消一个表时，对一个表的所有权限会被撤销。在标准SQL中，当您撤销一个权限时，根据该权限被授予的所有权限也会被撤销。在MySQL中，只有使用明确的REVOKE语句，或通过操作存储在MySQL授权表中的值，才能取消权限。
- 在MySQL中，可以只对一个表中的部分列拥有INSERT权限。在此情况下，如果您忽略您不拥有INSERT权限的那些列，，您仍然可以对表执行INSERT语句。如果没有启用严格的SQL模式，则被忽略的列被设置为各自隐含的默认值。在严格模式下，如果某个被忽略的列没有默认值，则该语句被拒绝。5.3.2节，“SQL服务器模式”对严格模式进行了讨论。13.1.5节，“CREATE TABLE语法”对隐含默认值进行了讨论。

您不拥有INSERT权限的列被设置为各自的默认值。标准SQL要求您拥有所有列的INSERT权限。

在MySQL中，如果您只拥有一个表中的部分列的INSERT权限，同时，如果您从INSERT语句中忽略您不拥有权限的列，则您仍然可以对表执行INSERT语句；那些列将被设置为各自的默认值。在严格模式下（即当sql_mode='traditional'时，如果某些被忽略的列没有默认值，则INSERT语句将被拒绝。

13.5.1.4. RENAME USER语法

```
RENAME USER old_user TO new_user
```

```
    [, old_user TO new_user] ...
```

RENAME USER语句用于对原有MySQL账户进行重命名。要使用RENAME USER，您必须拥有全局CREATE USER权限或mysql数据库UPDATE权限。如果旧 账户不存在或者新账户已存在，则会出现错误。*old_user*和*new_user*值的给定方法与GRANT语句一样。

13.5.1.5. SET PASSWORD语法

```
SET PASSWORD = PASSWORD('some password')

SET PASSWORD FOR user = PASSWORD('some password')
```

SET PASSWORD语句用于向一个原有MySQL用户 账户赋予一个密码。

第一个语法为当前用户设置密码。已使用一个非匿名账户连接到服务器上的任何客户即都可以更改该账户的密码。

第二个语法为当前服务器主机上的一个特定账户设置密码。只有拥有mysql数据库UPDATE权限的客户端可以这么做。user值应以*user_name*@*host_name*的格式被给定，此处*user_name*和*host_name*与mysql.user表登录项的User和Host列中列出的完全一样。举例说明，如果您有一个登录项，User和Host列值为'bob'和'%.loc.gov'，您应该按如下方法写语句：

```
mysql> SET PASSWORD FOR 'bob'@'%.loc.gov' = PASSWORD('newpass');
```

这相当于以下语句：

```
mysql> UPDATE mysql.user SET Password=PASSWORD('newpass')

    -> WHERE User='bob' AND Host='%.loc.gov';

mysql> FLUSH PRIVILEGES;
```

注释：如果您正在使用一个4.1以前的客户端连接到一个MySQL 4.1或MySQL 4.1以后的服务器，则在阅读5.7.9节，“MySQL 4.1中的密码哈希处理”之前，不能使用前面的SET PASSWORD或UPDATE语句。 密码格式在MySQL 4.1中变更了，并且在特定情况下，如果您更改密码，您可能无法在连接到服务器上。

您可以通过执行SELECT CURRENT_USER()观看您当前的鉴定user@host登录项。

13.5.2. 表维护语句

13.5.2.1. ANALYZE TABLE语法
13.5.2.2. BACKUP TABLE语法
13.5.2.3. CHECK TABLE语法
13.5.2.4. CHECKSUM TABLE语法
13.5.2.5. OPTIMIZE TABLE语法
13.5.2.6. REPAIR TABLE语法
13.5.2.7. RESTORE TABLE语法

13.5.2.1. ANALYZE TABLE语法

```
ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...
```

本语句用于分析和存储表的关键字分布。在分析期间，使用一个读取锁定对表进行锁定。这对于MyISAM, BDB和InnoDB表有作用。对于MyISAM表，本语句与使用**myisamchk -a**相当。

MySQL使用已存储的关键字分布来决定，当您对除常数以外的对象执行联合时，表按什么顺序进行联合。

本语句会返回一个含有以下列的表：

列	值
Table	表名称
Op	进行分析
Msg_type	状态、错误、信息或警告之一
Msg_text	消息

您可以使用SHOW INDEX语句检查已存储的关键字分布。请参见13.5.4.11节，“SHOW INDEX语法”。

如果从上一个ANALYZE TABLE语句开始，表没有变化，则不再分析该表。

ANALYZE TABLE语句被写入二进制日志中，除非使用了自选的NO_WRITE_TO_BINLOG关键词（或其别名LOCAL）。

13.5.2.2. BACKUP TABLE语法

```
BACKUP TABLE tbl_name [, tbl_name] ... TO '/path/to/backup/directory'
```

注释：本语句不理想。我们正在努力寻找一种更好的替代方式，该方式将提供在线备份能力。同时，也可以使用**mysqlhotcopy**原本替代。

BACKUP TABLE用于在刷新了所有对磁盘的缓冲变更后，把恢复表所需的最少数目的表文件拷贝到备份目录中。本语句只对MyISAM表起作用。它可以拷贝.frm定义文件和.MYD数据文件。.MYI索引文件可以从这两个文件中重建。本目录应被指定为一个完整的路径名。

在使用本语句前，请参见5.9.1节，“数据库备份”。

在备份期间，为每个表保持一个读取锁定，每次一个，在正在备份时锁定。如果您想要把多个表作为一个快照来备份（防止它们在备份操作过程中被更改），您必须实现发布一个LOCK TABLES语句，以获得对一个组群中的每个表的读取锁定。

该语句会返回一个含有以下列的表：

列	值
Table	表名称
Op	进行备份
Msg_type	状态、错误、信息或警告之一
Msg_text	消息

13.5.2.3. CHECK TABLE语法

```
CHECK TABLE tbl_name [, tbl_name] ... [option] ...

option = {QUICK | FAST | MEDIUM | EXTENDED | CHANGED}
```

检查一个或多个表是否有错误。CHECK TABLE对MyISAM和InnoDB表有作用。对于MyISAM表，关键字统计数据被更新。

CHECK TABLE也可以检查视图是否有错误，比如在视图定义中被引用的表已不存在。

CHECK TABLE语句会返回一个含有以下列的表：

列	值
Table	表名称
Op	进行检查
Msg_type	状态、错误、信息或错误之一
Msg_text	消息

注意，该语句可能会为每个被检查的表产生多行信息。最后一行有一个Msg_type状态值。Msg_text通常应为OK。如果您没有得到OK，或表已经更新了，则您通常应该运行修复后的表。请参见5.9.4节，“表维护和崩溃恢复”。表已经更新了，这意味着表的存储引擎指示没有必要检查表。

可以给予的不同的检查选项列于下表中。这些选项只适用于检查MyISAM表。对于InnoDB表和视图，这些选项被忽略。

类型	意义
QUICK	不扫描行，不检查错误的链接。
FAST	只检查没有被正确关闭的表。
CHANGED	只检查上次检查后被更改的表，和没有被正确关闭的表。
MEDIUM	扫描行，以验证被删除的链接是有效的。也可以计算各行的关键字校验和，并使用计算出的校验和验证这一点。
EXTENDED	对每行的所有关键字进行一个全面的关键字查找。这可以确保表是100％一致的，但是花的时间较长。

如果没有指定QUICK, MEDIUM或EXTENDED选项，则对于动态格式MyISAM表，默认检查类型是MEDIUM。这与对表运行**myisamchk --medium-check** *tbl_name*的结果相同。对于静态格式MyISAM表，默认检查类型也是MEDIUM，除非CHANGED或FAST已被指定。在此情况下，默认值为QUICK。对于CHANGED和FAST，行扫描被跳过，因为行极少被破坏。

您可以组合检查选项，如下面的例子所示。该例子对表进行了一个快速检查，来查看该表是否被正确关闭：

```
CHECK TABLE test_table FAST QUICK;
```

注释：在有些情况下，CHECK TABLE会更改表。如果表被标记为“corrupted”或“not closed properly”，则出现这种情况。但是CHECK TABLE不会找出表中的问题。在这种情况下，CHECK TABLE会把表标记为良好。

如果一个表被破坏，很有可能问题在索引中，而不在数据部分中。所有前述的检查类型都可以彻底地检查索引，因此，可以找出多数的错误。

如果您只想要检查您假定的表是良好的，您应该不使用检查选项或QUICK选项。当您时间匆忙时，应使用QUICK。QUICK无法找出数据文件中的错误的风险非常小。（在多数情况下，在正常使用中，MySQL应能在数据文件中找出错误。如果找出了错误，表被标记为“corrupted”，并不能被使用，直到修复为止。）

如果您想要时常检查表，FAST和CHANGED多数情况下从原本中被使用（例如，从cron中被执行）。在多数情况下，FAST优先于CHANGED。（只有一种情况FAST不优先于CHANGED，那就是当您怀疑您在MyISAM代码中发现了错误。）

当MySQL试图通过关键字更新一行或查找一行时，如果您已经运行了一个常规检查后但仍得到来自表的奇怪的错误，此时使用EXTENDED。（如果常规的检查运行成功，则基本用不着EXTENDED。）

被CHECK TABLE报告的部分问题不会被自动修正：

- 发现行。此行中，auto_increment列有0值。

这意味着，您在表中有一行，该行的AUTO_INCREMENT索引列包含0值。（可以通过使用UPDATE语句，明确地把列设置为0，以创建一个AUTO_INCREMENT列为0的行。）

这本身不是一个错误，但是如果您决定转储表并恢复表，或对表进行ALTER TABLE，那么会导致出现麻烦。在此情况下，AUTO_INCREMENT列会根据AUTO_INCREMENT列的结果更改值，这会导致出现问题，如重复关键字错误等。

要消除警告，只需执行一个UPDATE语句，把列设置为除0以外的值。

13.5.2.4. CHECKSUM TABLE语法

```
CHECKSUM TABLE tbl_name [, tbl_name] ... [ QUICK | EXTENDED ]
```

报告一个表校验和。

如果指定了QUICK，则报告活性表校验和，否则报告NULL。这是非常快的。活性表通过指定CHECKSUM＝1表选项启用，目前只支持用于MyISAM表。请参见13.1.5节，“CREATE TABLE语法”。

在EXTENDED模式下，整个表被一行一行地读取，并计算校验和。对于大型表，这是非常慢的。

默认情况下，如果既没有指定QUICK，也没有指定EXTENDED，并且如果表存储引擎支持，则MySQL返回一个活性校验和，否则会对表进行扫描。

CHECKSUM TABLE对于不存在的表会返回NULL。对于这种情况，会生成一个警告。

13.5.2.5. OPTIMIZE TABLE语法

```
OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...
```

如果您已经删除了表的一大部分，或者如果您已经对含有可变长度行的表（含有VARCHAR, BLOB或TEXT列的表）进行了很多更改，则应使用OPTIMIZE TABLE。被删除的记录被保持在链接清单中，后续的INSERT操作会重新使用旧的记录位置。您可以使用OPTIMIZE TABLE来重新利用未使用的空间，并整理数据文件的碎片。

在多数的设置中，您根本不需要运行OPTIMIZE TABLE。即使您对可变长度的行进行了大量的更新，您也不需要经常运行，每周一次或每月一次即可，只对特定的表运行。

OPTIMIZE TABLE只对MyISAM, BDB和InnoDB表起作用。

对于MyISAM表，OPTIMIZE TABLE按如下方式操作：

1. 如果表已经删除或分解了行，则修复表。
2. 如果未对索引页进行分类，则进行分类。
3. 如果表的统计数据没有更新（并且通过对索引进行分类不能实现修复），则进行更新。

对于BDB表，OPTIMIZE TABLE目前被映射到ANALYZE TABLE上。对于InnoDB表，OPTIMIZE TABLE被映射到ALTER TABLE上，这会重建表。重建操作能更新索引统计数据并释放成簇索引中的未使用的空间。请参见13.5.2.1节，“ANALYZE TABLE语法”。

使用—skip-new或—safe-mode选项可以启动mysqld。通过启动mysqld，您可以使OPTIMIZE TABLE对其它表

类型起作用。

注意，在OPTIMIZE TABLE运行过程中，MySQL会锁定表。

OPTIMIZE TABLE语句被写入到二进制日志中，除非使用了自选的NO_WRITE_TO_BINLOG关键词（或其别名LOCAL）。已经这么做了，因此，用于MySQL服务器的OPTIMIZE TABLE命令的作用相当于一个复制主服务器，在默认情况下，这些命令将被复制到复制从属服务器中。

13.5.2.6. REPAIR TABLE语法

```
REPAIR [LOCAL | NO_WRITE_TO_BINLOG] TABLE
    tbl_name [, tbl_name] ... [QUICK] [EXTENDED] [USE_FRM]
```

REPAIR TABLE用于修复被破坏的表。默认情况下，REPAIR TABLE与**myisamchk --recover** *tbl_name*具有相同的效果。REPAIR TABLE对MyISAM和ARCHIVE表起作用。请参见15.1节，“MyISAM存储引擎”，15.8节，“ARCHIVE存储引擎”。

通常，您基本上不必运行此语句。但是，如果灾难发生，REPAIR TABLE很有可能从MyISAM表中找回所有数据。如果您的表经常被破坏，您应该尽力找到原因，以避免使用REPAIR TALBE。请参见A.4.2节，“如果MySQL依然崩溃，应作些什么”。同时也见15.1.4节，“MyISAM表方面的问题”。

本语句会返回一个含有以下列的表：

列	值
Table	表名称
Op	进行修复
Msg_type	状态、错误、信息或警告之一
Msg_text	消息

对于每个被修复的表，REPAIR TABLE语句会产生多行的信息。上一行含有一个Msg_type状态值。Msg_test通常应为OK。如果您没有得到OK，您应该尝试使用**myisamchk --safe-recover**修复表，因为REPAIR TABLE尚不会执行所有的**myisamchk**选项。我们计划在将来使它的灵活性更强。

如果给定了QUICK，则REPAIR TABLE会尝试只修复索引树。这种类型的修复与使用**myisamchk --recover --quick**相似。

如果您使用EXTENDED，则MySQL会一行一行地创建索引行，代替使用分类一次创建一个索引。这种类型的修复与使用**myisamchk --safe-recover**相似。

对于REPAIR TABLE，还有一种USE_FRM模式可以利用。如果.MYI索引文件缺失或标题被破坏，则使用此模式。在这种模式下，MySQL可以使用来自.frm文件重新创建.MYI文件。这种修复不能使用**myisamchk**来完成。 注释：只能在您不能使用常规REPAIR模式是，才能使用此模式。.MYI标题包含重要的表元数据（特别是，当前的AUTO_INCREMENT值和Delete链接）。这些元数据在REPAIR...USE_FRM中丢失。如果表被压缩，则不能使用USE_FRM。因为本信息也存储在.MYI文件中。

REPAIR TABLE语句被写入二进制日志中，除非使用了自选的NO_WRITE_TO_BINLOG关键词（或其别名LOCAL）。

警告：如果在REPAIR TABLE运行过程中，服务器停机，则在重新启动之后，在执行其它操作之前，您必须立刻对表再执行一个REPAIR TABLE语句。（通过制作一个备份来启动是一个好办法。）再最不利情况下，您可以有一个新的干净的索引文件，不含有关数据文件的信息。然后，您执行的下一个操作会覆盖数据文件。这很少发生，但是是有可能的。

13.5.2.7. RESTORE TABLE语法

```
RESTORE TABLE tbl_name [, tbl_name] ... FROM '/path/to/backup/directory'
```

用于恢复来自用BACKUP TABLE制作的备份的表。原有的表不会被覆盖；如果您试图覆盖一个原有的表，会发生错误。和BACKUP TABLE一样，RESTORE TABLE目前只对MyISAM表起作用。目录应被指定为一个完整路径名。

每个表的备份包括其.frm格式文件和.MYD数据文件。恢复操作会恢复这些文件，然后使用这些文件来重建.MYI索引文件。恢复操作比备份操作花的时间更长，这是因为需要重建索引。表含有的索引越多，花的时间就越长。

该语句会返回一个含有以下列的表：

列	值
Table	表名称
Op	进行恢复
Msg_type	状态、错误、信息或警告之一
Msg_text	消息

13.5.3. SET语法

```
SET variable_assignment [, variable_assignment] ...


variable_assignment:

      user_var_name = expr

    | [GLOBAL | SESSION] system_var_name = expr

    | @@[global. | session.]system_var_name = expr
```

SET用于设置不同类型的变量。这些变量会影响服务器或客户端的操作。SET可以用于向用户变量或系统变量赋值。

用于分配账户密码的SET PASSWORD语句在13.5.1.5节，“SET PASSWORD语法”中进行了讨论。

多数系统变量可以在运行时间被更改。可以被动态设置的系统变量在5.3.3.1节，“动态系统变量”中进行了讨论。

注释：旧版本的MySQL采用SET OPTION作为这个命令，但是由于有了SET，现在不赞成使用SET OPTION。

以下例子显示了您可以用于设置变量的不同语法。

用户变量可以被写作@*var_name*，并可以进行如下设置：

```
SET @var_name = expr;
```

在9.3节，“用户变量”中给出了有关用户变量的更多信息。

系统变量可以被作为*var_name*引用到SET语句中。在名称的前面可以自选地添加GLOBAL或@@global，以明确地指示该变量是全局变量。或者在名称前面添加SESSION, @@session，或@@，以指示它是一个会话变量。LOCAL和@@local是SESSION和@@session地同义词。如果没有修改符，则SET设置会话变量。

支持系统变量的@@*var_name*语法，以便使MySQL语法与其它数据库系统相兼容。

如果您在同一个语句中设置多个系统变量，则最后一个GLOBAL或SESSION选项被用于没有指定模式的变量。

```
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;
```

如果您使用SESSION（默认情况）设置一个系统变量，则该值仍然有效，直到当前会话结束为止，或者直到您吧变量设置为一个不同的值为止。如果您使用GLOBAL（要求SUPER权限）来设置一个系统变量，则该值被记住，并被用于新的连接，直到服务器重新启动为止。如果您想要进行永久式变量设置，您应该把它放入一个选项文件。请参见4.3.2节，“使用选项文件”。

为了防止不正确的使用，如果您使用SET GLOBAL时同时使用了一个只能与SET SESSION同时使用的变量，或者如果您在设置一个全局变量时未指定GLOBAL（或@@），则MySQL会产生一个错误。

如果您想要把一个SESSION变量设置为GLOBAL值或把一个GLOBAL值设置为内部MySQL默认值，需使用DEFAULT关键词。例如，在把max_join_size会话值设置为全局值时，以下两个语句是一样的：

```
SET max_join_size=DEFAULT;

SET @@session.max_join_size=@@global.max_join_size;
```

您可以使用SHOW VARIABLES来得到系统变量清单。（见13.5.4.21节，“SHOW VARIABLES语法”。）要获得与样式匹配的一个具体的变量名称或者名称清单，需使用LIKE子句，使用方法如下：

```
SHOW VARIABLES LIKE 'max_join_size';

SHOW GLOBAL VARIABLES LIKE 'max_join_size';
```

要得到名称与样式匹配的变量的清单，需使用通配符‘%’：

```
SHOW VARIABLES LIKE 'have%';

SHOW GLOBAL VARIABLES LIKE 'have%';
```

通配符可以被用于相匹配的样式中的任何位置。

您也可以通过使用@@[global.|local.]var_name语法和SELECT来得到值：

```
SELECT @@max_join_size, @@global.max_join_size;
```

当您使用SELECT @@var_name（即您不指定全局、会话或本地）来恢复一个变量时，则MySQL会返回SESSION值（如果存在）或者GLOBAL值。

以下清单用于描述带有非标准语法的变量，或描述在系统变量清单中（见5.3.3节，“服务器系统变量”。）中没有描述的变量。尽管这些变量没有被SHOW VARIABLES显示，但是您可以使用SELECT来获得它们的值（例外情况是，使用CHARACTER SET和SET NAMES）。例如：

```
mysql> SELECT @@AUTOCOMMIT;

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

| @@AUTOCOMMIT |

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

|            1 |
```

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

· AUTOCOMMIT = {0 | 1}

设置autocommit模式。如果设置为1，则所有对表的更改会立刻生效。如果设置为0，则您必须使用COMMIT来接受一个事务，或使用ROLLBACK来取消它。如果您把AUTOCOMMIT模式从0改为1，则MySQL会对开放事务执行一个自动COMMIT。开始一个事务的另一种方法是使用一个START TRANSACTION或BEGIN语句。请参见13.4.1节，“START TRANSACTION, COMMIT和ROLLBACK语法”。

· BIG_TABLES = {0 | 1}

如果设置为1，所有的临时表被存储在磁盘中，而不是存储在储存期中。这样会稍微慢些，但是对于需要一个大型临时表的SELECT操作，不会发生The table *tbl_name* is full错误。对于一个新连接，默认值为0（使用存储器内部临时表）。通常，您不必设置此变量，因为根据需要，存储器内部表会被自动转换为以磁盘为基础的表。（注释：本变量以前被命名为SQL_BIG_TABLES。）

· CHARACTER SET {*charset_name* | DEFAULT}

本语句使用给定的映射为所有来自客户端和指向客户端的字符串建立映射。您可以通过在MySQL源分布中编辑sql/convert.cc来添加新的映射。SET CHARACTER SET用于设定三个会话系统变量：character_set_client和character_set_results被设置为给定的字符集，character_set_connection被设置为character_set_database值。

可以通过使用DEFAULT值恢复默认的映射。

注意，SET CHARACTER SET的语法与设置其它选项的语法不同。

· FOREIGN_KEY_CHECKS = {0 | 1}

如果设置为1（默认情况），则检查InnoDB表的外键限制条件。如果设置为0，则限制条件被忽略。如果重新载入InnoDB表时按照的顺序与上级/下级目录所要求的顺序不同，此时禁用外键检查是有用的。请参见15.2.6.4节，“FOREIGN KEY约束”。

· IDENTITY = *value*

该变量是LAST_INSERT_ID变量的同义词。该变量的作用是保持与其它数据库兼容。您可以使用SELECT @@IDENTITY读取其值，并可以使用SET IDENTITY设置它。

· INSERT_ID = *value*

用于设置将被以下INSERT或ALTER TABLE语句使用的值。此值在插入一个AUTO_INCREMENT值时使用。本语句主要和二进制日志同时使用。

· LAST_INSERT_ID = *value*

用于设定将从LAST_INSERT_ID()被返回的值。当您在用于更新表的语句中使用LAST_INSERT_ID()时，它被存储在二进制日志中。设置此变量不会更新由mysql_insert_id() C API函数返回的值。

· NAMES {'*charset_name*' | DEFAULT}

SET NAMES用于把三个会话系统变量character_set_client, character_set_connection和character_set_results设置为给定的字符集。把character_set_connection设置为charset_name时，同时把collation_connection设置为charset_name的默认整序。

使用一个DEFAULT值可以恢复默认的映射。

注意，SET NAMES的语法与用于设置其它选项的语法不同。

· ONE_SHOT

这不是一个服务器系统变量，但是它可以被用来影响用于设置字符集、整序和时区的变量的效果。ONE_SHOT主要被用于复制：mysqlbinlog使用SET ONE_SHOT来暂时地修改字符集、整序和时区变量的值，以反映出它们原先的值。

您不能在使用ONE_SHOT时使用除允许的变量以外的变量；如果您这么做，您会得到如下错误：

```
mysql> SET ONE_SHOT max_allowed_packet = 1;

ERROR 1382 (HY000): The 'SET ONE_SHOT' syntax is reserved for purposes internal to the MySQL
server
```

如果同时使用ONE_SHOT和被允许的变量，则会根据要求更改变量，但是会在下一个语句后，重新设置所有的字符集、整序和与时区有关的服务器系统变量。唯一的例外是，当下一个语句是SET语句时，不会进行重新设置。换句话说，在下一个非SET语句之后，才会进行重新设置。例如：

```
mysql> SET ONE_SHOT character_set_connection = latin5;


mysql> SET ONE_SHOT collation_connection = latin5_turkish_ci;


mysql> SHOW VARIABLES LIKE '%_connection';

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

| Variable_name            | Value             |

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

| character_set_connection | latin5            |

| collation_connection     | latin5_turkish_ci |

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


mysql> SHOW VARIABLES LIKE '%_connection';

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

| Variable_name            | Value             |

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

| character_set_connection | latin1            |

| collation_connection     | latin1_swedish_ci |

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

· SQL_NOTES = {0 | 1}

当设置为1时（默认情况），“注意”一级的警报被记录下来。当设置为0时，“注意”警告被压制。Mysqldump包含输出，用于把此变量设置为0，这样，对于不会影响重新载入操作整体性的事件，重新载入转储文件时不会产生警告。

· SQL_AUTO_IS_NULL = {0 | 1}

如果设置为1（默认情况），您可以通过使用以下结构查找包含一个AUTO_INCREMENT列的表的最后插入的行：

```
WHERE auto_increment_column IS NULL
```

此性质被有些ODBC程序，比如Access使用。

· SQL_BIG_SELECTS = {0 | 1}

如果设定为0，则MySQL会放弃有可能会花很长时间来执行的SELECT语句（也就是，对于这些语句，优化程序估算被检查的行的数目超过了max_join_size的值）。当一个不妥当的WHERE语句被发布后，本语句有用。一个新连接的默认值为1，这可以允许所有的SELECT语句。

如果您把max_join_size系统变量设置为除DEFAULT以外的值，则SQL_BIG_SELECTS被设置为0。

· SQL_BUFFER_RESULT = {0 | 1}

SQL_BUFFER_RESULT会迫使来自SELECT语句的结果被放入临时表中。这可以帮助MySQL早点解除表锁定。当需要花较长时间把结果发送给客户端时，这是有好处的。

· SQL_LOG_BIN = {0 | 1}

如果设置为0，则客户端的二进制日志中不会记录日志。客户端必须拥有SUPER权限来设置此选项。

· SQL_LOG_OFF = {0 | 1}

如果设置为1，则此客户端的总查询日志中不会记录日志。客户端必须拥有SUPER权限来设置此选项。

· SQL_LOG_UPDATE = {0 | 1}

不赞成使用本变量。本变量被映射到SQL_LOG_BIN。

· SQL_QUOTE_SHOW_CREATE = {0 | 1}

如果设置为1，则SHOW CREATE TABLE会对表和列的名称加引号。如果设置为0，则加引号操作被禁用。默认情况下，本选项被启用，因此对于含有需要加引号的名称的表，复制操作起作用。请参见13.5.4.5节，“SHOW CREATE TABLE语法”。

· SQL_SAFE_UPDATES = {0 | 1}

如果设置为1，则MySQL会放弃在WHERE子句或LIMIT子句中不使用关键字的UPDATE或DELETE语句。这样，当关键字使用不正确时，也有可能理解UPDATE或DELETE语句。这样就可以更改或删除大量的行。

· SQL_SELECT_LIMIT = {*value* | DEFAULT}

从SELECT语句返回的记录的最大数目。对于一个新连接，默认值是“unlimited”。如果您更改了限值，可以使用SQL_SELECT_LIMIT DEFAULT值恢复默认值。

如果SELECT有一个LIMIT子句，则LIMIT优先于SQL_SELECT_LIMIT值。

SQL_SELECT_LIMT不适用于在被存储的子程序中执行的SELECT语句。它也不适用于不会产生将被返回到客户端的结果集合的SELECT语句。这些包括子查询中的SELECT语句，CREATE TABLE...SELECT和INSERT INTO...SELECT。

· SQL_WARNINGS = {0 | 1}

本变量用于控制当出现警告时，单行INSERT语句是否产生一个信息字符串。默认值为0。把值设置为1，来产生一个信息字符串。

· TIMESTAMP = {*timestamp_value* | DEFAULT}

用于为此客户端设置时间。当您使用二进制日志来恢复行时，本语句用于得到原始的时间标记。timestamp_value应为一个Unix时间标记，而不是MySQL时间标记。

- UNIQUE_CHECKS = {0 | 1}

如果设置为1（默认情况），则会对InnoDB表中的二级索引执行唯一性检查。如果设置为0，则对于被插入到InnoDB的插入缓冲器中的索引登录项，不执行唯一性检查。如果您可以肯定您的数据不违反唯一性要求，则您可以把此值设定为0，以加快向InnoDB导入大型表的速度。

13.5.4. SHOW语法

13.5.4.1. SHOW CHARACTER SET语法
13.5.4.2. SHOW COLLATION语法
13.5.4.3. SHOW COLUMNS语法
13.5.4.4. SHOW CREATE DATABASE语法
13.5.4.5. SHOW CREATE TABLE语法
13.5.4.6. SHOW DATABASES语法
13.5.4.7. SHOW ENGINE语法
13.5.4.8. SHOW ENGINES语法
13.5.4.9. SHOW ERRORS语法
13.5.4.10. SHOW GRANTS语法
13.5.4.11. SHOW INDEX语法
13.5.4.12. SHOW INNODB STATUS语法
13.5.4.13. SHOW LOGS语法
13.5.4.14. SHOW OPEN TABLES语法
13.5.4.15. SHOW PRIVILEGES语法
13.5.4.16. SHOW PROCESSLIST语法
13.5.4.17. SHOW STATUS语法
13.5.4.18. SHOW TABLE STATUS语法
13.5.4.19. SHOW TABLES语法
13.5.4.20. SHOW TRIGGERS语法
13.5.4.21. SHOW VARIABLES语法
13.5.4.22. SHOW WARNINGS语法

SHOW有多种形式，可以提供有关数据库、表、列或服务器状态的信息。本节叙述以下内容：

```
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']

SHOW CREATE DATABASE db_name

SHOW CREATE TABLE tbl_name

SHOW DATABASES [LIKE 'pattern']

SHOW ENGINE engine_name {LOGS | STATUS }

SHOW [STORAGE] ENGINES

SHOW ERRORS [LIMIT [offset,] row_count]

SHOW GRANTS FOR user

SHOW INDEX FROM tbl_name [FROM db_name]

SHOW INNODB STATUS

SHOW [BDB] LOGS

SHOW PRIVILEGES
```

```
SHOW [FULL] PROCESSLIST

SHOW [GLOBAL | SESSION] STATUS [LIKE 'pattern']

SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']

SHOW [OPEN] TABLES [FROM db_name] [LIKE 'pattern']

SHOW TRIGGERS

SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern']

SHOW WARNINGS [LIMIT [offset,] row_count]
```

SHOW语句还有一些形式，可以提供有关复制型主服务器和从属服务器的信息。这些形式在13.6节，“复制语句”中进行了叙述。

```
SHOW BINLOG EVENTS

SHOW MASTER LOGS

SHOW MASTER STATUS

SHOW SLAVE HOSTS

SHOW SLAVE STATUS
```

如果一个给定的SHOW语句的语法包括一个LIKE '*pattern*'部分，则'*pattern*'是一个可以包含SQL ‘%’和‘_’通配符的字符串。对于把语句输出值限定为匹配值，本样式是有用的。

13.5.4.1. SHOW CHARACTER SET语法

```
SHOW CHARACTER SET [LIKE 'pattern']
```

SHOW CHARACTER SET语句用于显示所有可用的字符集。该语句取一个自选的LIKE子句。该子句指示哪些字符集名称可以匹配。举例说明：

```
mysql> SHOW CHARACTER SET LIKE 'latin%';

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

| Charset | Description                 | Default collation | Maxlen |

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

| latin1  | cp1252 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 |

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

Maxlen列显示用于存储一个字符的最大的字节数目。

13.5.4.2. SHOW COLLATION语法

```
SHOW COLLATION [LIKE 'pattern']
```

来自SHOW COLLATION的输出包括所有可用的字符集。该语句取一个自选的LIKE子句。该子句的pattern指示哪些整序名称可以匹配。举例说明：

```
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 |
| latin1_spanish_ci | latin1  | 94 |         |          |       0 |
+-------------------+---------+----+---------+----------+---------+
```

Default列指示对于其字符集，整序值是否是默认值。Compiled指示字符集是否被编辑到服务器中。Sortlen与对字符串（在字符集中表达）分类所需的存储器的数量有关。

13.5.4.3. SHOW COLUMNS语法

```
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']
```

SHOW COLUMNS显示在一个给定表中的各列的信息。对于试图，本语句也起作用。

如果列类型与根据您的CREATE TABLE语句所预期的列类型不同，则需注意，当您创建或更改表时，MySQL有时会更改列类型。出现这种情况的条件在13.1.5.1节，“沉寂的列规格变更”中进行了描述。

FULL关键词会使得输出中包含您拥有的权限，并包含对每一列各自的评注。

您可以使用db_name.tbl_name作为tbl_name FROM db_name语法的另一种形式。换句话说，这两个语句是等价的：

```
mysql> SHOW COLUMNS FROM mytable FROM mydb;
mysql> SHOW COLUMNS FROM mydb.mytable;
```

SHOW FIELDS是SHOW COLUMNS的同义词。您也可以使用mysqlshow db_name tbl_name命令列举表的各列。

DESCRIBE语句提供与SHOW COLUMNS相近的信息。请参见13.3.1节，“DESCRIBE语法（获取关于列的信息）”。

13.5.4.4. SHOW CREATE DATABASE语法

```
SHOW CREATE {DATABASE | SCHEMA} db_name
```

显示用于创建给定数据库CREATE DATABASE语句。也可以使用SHOW CREATE SCHEMA。

```
mysql> SHOW CREATE DATABASE test\G
*************************** 1. row ***************************
```

```
       Database: test
Create Database: CREATE DATABASE `test`
                 /*!40100 DEFAULT CHARACTER SET latin1 */

mysql> SHOW CREATE SCHEMA test\G
*************************** 1. row ***************************
       Database: test
Create Database: CREATE DATABASE `test`
                 /*!40100 DEFAULT CHARACTER SET latin1 */
```

13.5.4.5. SHOW CREATE TABLE语法

```
SHOW CREATE TABLE tbl_name
```

显示用于创建给定表的CREATE TABLE语句。本语句对视图也起作用。

```
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)
) ENGINE=MyISAM
```

根据SQL_QUOTE_SHOW_CREATE选项，SHOW CREATE TABLE会对表名称和列名称加引号。请参见13.5.3节，“SET语法”。

13.5.4.6. SHOW DATABASES语法

```
SHOW {DATABASES | SCHEMAS} [LIKE 'pattern']
```

SHOW DATABASES可以在MySQL服务器主机上列举数据库。您也可以使用mysqlshow命令得到此清单。您只能看到您拥有某些权限的数据库，除非您拥有全局SHOW DATABASES权限。

如果服务器以--skip-show-database选项为起始，则您根本不能使用本语句，除非您拥有SHOW DATABASES权限。

也可以使用SHOW SCHEMAS。

13.5.4.7. SHOW ENGINE语法

```
SHOW ENGINE engine_name {LOGS | STATUS }
```

SHOW ENGINE显示存储引擎的日志或状态信息。目前支持以下语句：

```
SHOW ENGINE BDB LOGS

SHOW ENGINE INNODB STATUS
```

SHOW ENGINE BDB LOGS显示原有BDB日志文件的状态信息。它会返回以下字段:

- File

通向日志文件的完整路径。

- Type

日志文件类型（用于Berkeley DB日志文件的BDB）。

- Status

日志文件的状态（如果文件可以被取消，则为FREE。如果文件被事务子系统需要，则为IN USE）

SHOW ENGINE INNODB STATUS显示InnoDB存储引擎状态的全面信息。

这些语句的旧的同义词（现在不赞成使用）是SHOW [BDB] LOGS和SHOW INNODB STATUS。

SHOW ENGINE可以从MySQL 4.1.2起使用。

13.5.4.8. SHOW ENGINES语法

```
SHOW [STORAGE] ENGINES
```

SHOW ENGINES显示存储引擎的状态信息。对于检查一个存储引擎是否被支持，或者对于查看默认引擎是什么，本语句十分有用。SHOW TABLE TYPES是同义词，但不赞成使用。

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
 Engine: MyISAM
Support: DEFAULT
Comment: Default engine as of MySQL 3.23 with great performance
*************************** 2. row ***************************
 Engine: MEMORY
Support: YES
Comment: Hash based, stored in memory, useful for temporary tables
*************************** 3. row ***************************
 Engine: HEAP
Support: YES
Comment: Alias for MEMORY
*************************** 4. row ***************************
 Engine: MERGE
Support: YES
Comment: Collection of identical MyISAM tables
*************************** 5. row ***************************
 Engine: MRG_MYISAM
Support: YES
Comment: Alias for MERGE
*************************** 6. row ***************************
 Engine: ISAM
Support: NO
Comment: Obsolete storage engine, now replaced by MyISAM
*************************** 7. row ***************************
 Engine: MRG_ISAM
Support: NO
Comment: Obsolete storage engine, now replaced by MERGE
*************************** 8. row ***************************
 Engine: InnoDB
Support: YES
Comment: Supports transactions, row-level locking, and foreign keys
*************************** 9. row ***************************
 Engine: INNOBASE
Support: YES
Comment: Alias for INNODB
```

```
*************************** 10. row ***************************
 Engine: BDB
Support: NO
Comment: Supports transactions and page-level locking
*************************** 11. row ***************************
 Engine: BERKELEYDB
Support: NO
Comment: Alias for BDB
*************************** 12. row ***************************
 Engine: NDBCLUSTER
Support: DISABLED
Comment: Clustered, fault-tolerant, memory-based tables
*************************** 13. row ***************************
 Engine: NDB
Support: DISABLED
Comment: Alias for NDBCLUSTER
*************************** 14. row ***************************
 Engine: EXAMPLE
Support: NO
Comment: Example storage engine
*************************** 15. row ***************************
 Engine: ARCHIVE
Support: YES
Comment: Archive storage engine
*************************** 16. row ***************************
 Engine: CSV
Support: YES
Comment: CSV storage engine
*************************** 17. row ***************************
 Engine: FEDERATED
Support: NO
Comment: Federated MySQL storage engine
*************************** 18. row ***************************
 Engine: BLACKHOLE
Support: YES
Comment: /dev/null storage engine (anything you write to it disappears)
```

Support值指示某个存储引擎是否被支持，并指示哪个是默认引擎。例如，如果服务器以--default-table-type=InnoDB选项为起始，则InnoDB行的Support值为DEFAULT值。请参见第15章：存储引擎和表类型。

13.5.4.9. SHOW ERRORS语法

```
SHOW ERRORS [LIMIT [offset,] row_count]

SHOW COUNT(*) ERRORS
```

本语句与SHOW WARNINGS接近，不过该语句只显示错误，不同时显示错误、警告和注意。

LIMIT子句与SELECT语句具有相同的语法，请参见13.2.7节，“SELECT语法”。

SHOW COUNT(*) ERRORS语句显示错误的数目。您也可以从error_count变量中找回此数目：

```
SHOW COUNT(*) ERRORS;

SELECT @@error_count;
```

要了解更多信息，请参见13.5.4.22节，“SHOW WARNINGS语法”。

13.5.4.10. SHOW GRANTS语法

```
SHOW GRANTS FOR user
```

本语句列出了在为MySQL用户账户复制权限时必须发布的GRANT语句。

```
mysql> SHOW GRANTS FOR 'root'@'localhost';

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

```
| Grants for root@localhost                                          |
+--------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+--------------------------------------------------------------------+
```

要对当前的会话列出权限，您可以使用以下语句之一：

```
SHOW GRANTS;
SHOW GRANTS FOR CURRENT_USER;
SHOW GRANTS FOR CURRENT_USER();
```

13.5.4.11. SHOW INDEX语法

```
SHOW INDEX FROM tbl_name [FROM db_name]
```

SHOW INDEX会返回表索引信息。其格式与ODBC中的SQLStatistics调用相似。

SHOW INDEX会返回以下字段：

· Table

表的名称。

· Non_unique

如果索引不能包括重复词，则为0。如果可以，则为1。

· Key_name

索引的名称。

· Seq_in_index

索引中的列序列号，从1开始。

· Column_name

列名称。

· Collation

列以什么方式存储在索引中。在MySQL中，有值'A'（升序）或NULL（无分类）。

· Cardinality

索引中唯一值的数目的估计值。通过运行ANALYZE TABLE或**myisamchk -a**可以更新。基数根据被存储为整数的统计数据来计数，所以即使对于小型表，该值也没有必要是精确的。基数越大，当进行联合时，MySQL使用该索引的机会就越大。

· Sub_part

如果列只是被部分地编入索引，则为被编入索引的字符的数目。如果整列被编入索引，则为NULL。

· Packed

指示关键字如何被压缩。如果没有被压缩，则为NULL。

· Null

如果列含有NULL，则含有YES。如果没有，则该列含有NO。

· Index_type

用过的索引方法（BTREE, FULLTEXT, HASH, RTREE）。

· Comment

多种评注。

您可以使用db_name.tbl_name作为tbl_name FROM db_name语法的另一种形式。这两个语句是等价的：

```
mysql> SHOW INDEX FROM mytable FROM mydb;
mysql> SHOW INDEX FROM mydb.mytable;
```

SHOW KEYS是SHOW INDEX的同义词。您也可以使用**mysqlshow -k** *db_name tbl_name*命令列举一个表的索引。

13.5.4.12. SHOW INNODB STATUS语法

```
SHOW INNODB STATUS
```

在MySQL 5.1中，这是SHOW ENGINE INNODB STATUS的同义词，但不赞成使用。请参见13.5.4.7节，“SHOW ENGINE语法”。

13.5.4.13. SHOW LOGS语法

```
SHOW [BDB] LOGS
```

在MySQL 5.1中，这是SHOW ENGINE BDB LOGS的同义词，但是不赞成使用。请参见13.5.4.7节，“SHOW ENGINE语法”。

13.5.4.14. SHOW OPEN TABLES语法

```
SHOW OPEN TABLES [FROM db_name] [LIKE 'pattern']
```

SHOW OPEN TABLES列举在表缓存中当前被打开的非TEMPORARY表。请参见7.4.9节，“MySQL如何打开和关闭表”。

SHOW OPEN TABLES会返回以下字段：

· Database

含有该表的数据库。

· Table

表名称。

· In_use

表当前被查询使用的次数。如果该数为零，则表是打开的，但是当前没有被使用。

- Name_locked

表名称是否被锁定。名称锁定用于取消表或对表进行重命名等操作。

13.5.4.15. SHOW PRIVILEGES语法

```
SHOW PRIVILEGES
```

SHOW PRIVILEGES显示MySQL服务器支持的系统权限清单。确切的输出根据您的服务器的版本而定。

```
mysql> SHOW PRIVILEGES\G
*************************** 1. row ***************************
Privilege: Alter
Context: Tables
Comment: To alter the table
*************************** 2. row ***************************
Privilege: Alter routine
Context: Functions,Procedures
Comment: To alter or drop stored functions/procedures
*************************** 3. row ***************************
Privilege: Create
Context: Databases,Tables,Indexes
Comment: To create new databases and tables
*************************** 4. row ***************************
Privilege: Create routine
Context: Functions,Procedures
Comment: To use CREATE FUNCTION/PROCEDURE
*************************** 5. row ***************************
Privilege: Create temporary tables
Context: Databases
Comment: To use CREATE TEMPORARY TABLE
*************************** 6. row ***************************
Privilege: Create view
Context: Tables
Comment: To create new views
*************************** 7. row ***************************
Privilege: Create user
Context: Server Admin
Comment: To create new users
*************************** 8. row ***************************
```

```
  Privilege: Delete
  Context: Tables
  Comment: To delete existing rows
*************************** 9. row ***************************
  Privilege: Drop
  Context: Databases,Tables
  Comment: To drop databases, tables, and views
*************************** 10. row ***************************
  Privilege: Execute
  Context: Functions,Procedures
  Comment: To execute stored routines
*************************** 11. row ***************************
  Privilege: File
  Context: File access on server
  Comment: To read and write files on the server
*************************** 12. row ***************************
  Privilege: Grant option
  Context: Databases,Tables,Functions,Procedures
  Comment: To give to other users those privileges you possess
*************************** 13. row ***************************
  Privilege: Index
  Context: Tables
  Comment: To create or drop indexes
*************************** 14. row ***************************
  Privilege: Insert
  Context: Tables
  Comment: To insert data into tables
*************************** 15. row ***************************
  Privilege: Lock tables
  Context: Databases
  Comment: To use LOCK TABLES (together with SELECT privilege)
*************************** 16. row ***************************
  Privilege: Process
  Context: Server Admin
  Comment: To view the plain text of currently executing queries
*************************** 17. row ***************************
  Privilege: References
  Context: Databases,Tables
  Comment: To have references on tables
```

```
*************************** 18. row ***************************
Privilege: Reload
Context: Server Admin
Comment: To reload or refresh tables, logs and privileges
*************************** 19. row ***************************
Privilege: Replication client
Context: Server Admin
Comment: To ask where the slave or master servers are
*************************** 20. row ***************************
Privilege: Replication slave
Context: Server Admin
Comment: To read binary log events from the master
*************************** 21. row ***************************
Privilege: Select
Context: Tables
Comment: To retrieve rows from table
*************************** 22. row ***************************
Privilege: Show databases
Context: Server Admin
Comment: To see all databases with SHOW DATABASES
*************************** 23. row ***************************
Privilege: Show view
Context: Tables
Comment: To see views with SHOW CREATE VIEW
*************************** 24. row ***************************
Privilege: Shutdown
Context: Server Admin
Comment: To shut down the server
*************************** 25. row ***************************
Privilege: Super
Context: Server Admin
Comment: To use KILL thread, SET GLOBAL, CHANGE MASTER, etc.
*************************** 26. row ***************************
Privilege: Update
Context: Tables
Comment: To update existing rows
*************************** 27. row ***************************
Privilege: Usage
Context: Server Admin
```

```
Comment: No privileges - allow connect only
```

13.5.4.16. SHOW PROCESSLIST语法

```
SHOW [FULL] PROCESSLIST
```

SHOW PROCESSLIST显示哪些线程正在运行。您也可以使用**mysqladmin processlist**语句得到此信息。如果您有SUPER权限，您可以看到所有线程。否则，您只能看到您自己的线程（也就是，与您正在使用的MySQL账户相关的线程）。请参见13.5.5.3节，“KILL语法”。如果您不使用FULL关键词，则只显示每个查询的前100个字符。

本语句报告TCP/IP连接的主机名称（采用*host_name*:*client_port*格式），以方便地判定哪个客户端正在做什么。

如果您得到“too many connections”错误信息，并且想要了解正在发生的情况，本语句是非常有用的。MySQL保留一个额外的连接，让拥有SUPER权限的 账户使用，以确保管理员能够随时连接和检查系统（假设您没有把此权限给予所有的用户）。

在来自SHOW PROCESSLIST的输出中常见的一些状态:

- Checking table

线程正在执行（自动）表格检查。

- Closing tables

意味着线程正在刷新更改后的表数据，并正在关闭使用过的表。这应该是一个快速的操作。如果不快，则您应该验证您的磁盘没有充满，并且磁盘没有被超负荷使用。

- Connect Out

连接到主服务器上的从属服务器。

- Copying to tmp table on disk

临时结果集合大于tmp_table_size。线程把临时表从存储器内部格式改变为磁盘模式，以节约存储器。

- Creating tmp table

线程正在创建一个临时表，以保持部分结果。

- deleting from main table

服务器正在执行多表删除的第一部分，只从第一个表中删除。

- deleting from reference tables

服务器正在执行多表删除的第二部分，从其它表中删除匹配的行。

- Flushing tables

线程正在执行FLUSH TABLES，并正在等待所有线程，以关闭表。

- FULLTEXT initialization

服务器正在准备执行一个自然语言全文本搜索。

- Killed

有人已经向线程发送了一个KILL命令。在下一次检查终止标记时，应放弃。该标记在MySQL的每个大循环中都检查，但是在有些情况下，线程终止只需要较短的时间。如果该线程被其它线程锁定，则只要其它线程接触锁定，终止操作就会生效。

- Locked

该查询被其它查询锁定。

- Sending data

线程正在为SELECT语句处理行，同时正在向客户端发送数据。

- Sorting for group

线程正在进行分类，以满足GROUP BY要求。

- Sorting for order

线程正在进行分类，以满足ORDER BY要求。

- Opening tables

线程正在试图打开一个表。这应该是非常快的过程，除非打开操作受到阻止。例如，一个ALTER TABLE或一个LOCK TABLE语句可以阻止打开一个表，直到语句完成为止。

- Removing duplicates

查询正在使用SELECT DISTINCT。使用时，在早期阶段，MySQL不能优化不同的操作。因此，MySQL要求一个额外的阶段，以便在把结果发送给客户端之前取消所有的复制行。

- Reopen table

线程得到一个表锁定，但是在得到锁定后被通知带下方的表结构已更改了。它已经释放了锁定，关闭了表，并试图重新打开它。

- Repair by sorting

修复代码正在使用一个分类来创建索引。

- Repair with keycache

修复代码正在通过关键缓存一个接一个地使用创建关键字。这比通过分类修复要慢很多。

- Searching rows for update

线程正在进行第一阶段，以在更新之前，查找所有匹配的行。如果UPDATE正在更改用于查找相关行的索引，则必须这么做。

- Sleeping

线程正在等待客户端，以向它发送一个新语句。

- System lock

线程正在等待得到一个用于表的外部系统锁定。如果您没有正在使用多个正在访问同一个表的mysqld服务器，

则您可以使用--skip-external-locking选项禁用系统锁定。

· Upgrading lock

INSERT DELAYED管理程序正在试图得到一个表锁定，以插入行。

· Updating

线程正在搜索行，并正在更新这些行。

· User Lock

线程正在等待GET_LOCK()。

· Waiting for tables

线程得到一个通知，表的底层结构已经改变，需要重新打开表以得到新的结构。但是，为了能重新打开表，必须等待，直到所有其它的线程已经关闭了正在被质询的表。

如果其它线程已经对正在被质询的表使用了FLUSH TABLES或以下语句之一：FLUSH TABLES *tbl_name*, ALTER TABLE, RENAME TABLE, REPAIR TABLE, ANALYZE TABLE或OPTIMIZE TABLE；则会出现通知。

· waiting for handler insert

INSERT DELAYED管理程序已经处理了所有处于等待状态的插入，并正在等待新插入。

多数状态对应于非常快的操作。如果一个线程在这些状态下停留了数秒，则可能是有问题，需要进行调查。

有一些其它的状态，在前面的清单中没有提及，但是其中有很多状态对于查找服务器中的程序错误是有用的。

13.5.4.17. SHOW STATUS语法

```
SHOW [GLOBAL | SESSION] STATUS [LIKE 'pattern']
```

SHOW STATUS提供服务器状态信息。此信息也可以使用**mysqladmin extended-status**命令获得。

此处显示了局部的输出。对于您的服务器，变量和值的清单可以是不同的。在5.3.4节，“服务器状态变量”中给出了每个变量的意义。

```
mysql> SHOW STATUS;
+--------------------------+------------+
| 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         |
```

```
...                         ...          ...
| Open_tables               | 1          |
| Open_files                | 2          |
| Open_streams              | 0          |
| Opened_tables             | 44600      |
| Questions                 | 2026873    |
...                         ...          ...
| Table_locks_immediate     | 1920382    |
| Table_locks_waited        | 0          |
| Threads_cached            | 0          |
| Threads_created           | 30022      |
| Threads_connected         | 1          |
| Threads_running           | 1          |
| Uptime                    | 80380      |
+--------------------------+------------+
```

使用LIKE子句，该语句只显示匹配该样式的那些变量:

```
mysql> SHOW STATUS LIKE 'Key%';
+--------------------+----------+
| Variable_name      | Value    |
+--------------------+----------+
| Key_blocks_used    | 14955    |
| Key_read_requests  | 96854827 |
| Key_reads          | 162040   |
| Key_write_requests | 7589728  |
| Key_writes         | 3813196  |
+--------------------+----------+
```

使用GLOBAL选项，您可以得到所有MySQL连接的状态值。使用SESSION，您可以得到所有当前连接的状态值。如果您两个选项都不使用，则默认值为SESSION。LOCAL是SESSION的同义词。

注意，有些状态变量只有一个全局值。对于这些变量，使用GLOBAL和SESSION会得到同样的值。

13.5.4.18. SHOW TABLE STATUS语法

```
SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
```

SHOW TABLE STATUS的性质与SHOW TABLE类似，不过，可以提供每个表的大量信息。您也可以使用**mysqlshow --status** *db_name*命令得到此清单。

本语句也显示视图信息。

对于NDB Cluster表，本语句的输出显示Avg_row_length和Data_length列的适当值，不过BLOB列没有被考虑进来。另外，复制数量在Comment列中显示（作为number_of_replicas）。

SHOW TABLE STATUS会返回以下字段:

· Name

表的名称。

· Engine

表的存储引擎。在MySQL 4.1.2之前，本值被标记为Type。请参见第15章：存储引擎和表类型。

· Version

表的.frm文件的版本号。

· Row_format

行存储格式（Fixed, Dynamic, Compressed, Redundant, Compact）。InnoDB表的格式被报告为Redundant或Compact。

· Rows

行的数目。部分存储引擎，如MyISAM，存储精确的数目。

对于其它存储引擎，比如InnoDB，本值是一个大约的数，与实际值相差可达40到50%。在这些情况下，使用SELECT COUNT(*)来获得准确的数目。

对于在INFORMATION_SCHEMA数据库中的表，Rows值为NULL。

· Avg_row_length

平均的行长度。

· Data_length

数据文件的长度。

· Max_data_length

数据文件的最大长度。如果给定了数据指针的大小，这是可以被存储在表中的数据的字节总数。

· Index_length

索引文件的长度。

· Data_free

被整序，但是未使用的字节的数目。

· Auto_increment

下一个AUTO_INCREMENT值。

· Create_time

什么时候表被创建。

· Update_time

什么时候数据文件被最后一次更新。

· Check_time

什么时候表被最后一次检查。不是所有的存储引擎此时都更新，在此情况下，值为NULL。

· Collation

表的字符集和整序。

· Checksum

活性校验和值。

· Create_options

和CREATE TABLE同时使用的额外选项。

· Comment

创建表时使用的评注（或者有关为什么MySQL可以访问表信息的说明）。

在表评注中，InnoDB表报告表所属的表空间的空闲空间。对于一个位于共享表空间中的表，这是共享表空间中的空闲空间。如果您正在使用多个表空间，并且该表有自己的表空间，则空闲空间只用于此表。

对于MEMORY (HEAP)表，Data_length, Max_data_length和Index_length值近似于被整序的存储器的实际值。整序算法预留了大量的存储器，以减少整序操作的数量。

对于视图，由SHOW TABLE STATUS显示的所有字段均为NULL。例外情况是Name指示为视图名称同时Comment称为视图。

13.5.4.19. SHOW TABLES语法

```
SHOW [FULL] TABLES [FROM db_name] [LIKE 'pattern']
```

SHOW TABLES列举了给定数据库中的非TEMPORARY表。您也可以使用**mysqlshow** *db_name*命令得到此清单。

本命令也列举数据库中的其它视图。支持FULL修改符，这样SHOW FULL TABLES就可以显示第二个输出列。对于一个表，第二列的值为BASE TABLE；对于一个视图，第二列的值为VIEW。

注释：如果您对于一个表没有权限，则该表不会在来自SHOW TABLES或的**mysqlshow db_name**输出中显示。

13.5.4.20. SHOW TRIGGERS语法

```
SHOW TRIGGERS [FROM db_name] [LIKE expr]
```

SHOW TRIGGERS列出了目前被MySQL服务器定义的触发程序。

对于在21.3节，“使用触发程序”中定义的触发程序ins_sum，本语句的输出显示如下：

```
mysql> SHOW TRIGGERS LIKE 'acc%';

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

```
| Trigger | Event  | Table    | Statement                      | Timing | Created |
+---------+--------+----------+--------------------------------+--------+---------+
| ins_sum | INSERT | account |  SET @sum = @sum + NEW.amount | BEFORE | NULL    |
+---------+--------+----------+--------------------------------+--------+---------+
```

注释：当使用一个含有SHOW TRIGGERS的LIKE子句时，待匹配的表达式（expr）会与触发程序定义时所在的表的名称相比较，而不与触发程序的名称相比较：

```
mysql> SHOW TRIGGERS LIKE 'ins%';
Empty set (0.01 sec)
```

对本语句输出中的各列的简要解释如下：

- Trigger: 触发程序的名称。
- Event: 调用触发程序的时间。必须为'INSERT', 'UPDATE'或 'DELETE'.之一。
- Table: 触发程序定义时对应的表。
- Statement: 当触发程序被调用时执行的语句。这与在INFORMATION_SCHEMA.TRIGGERS的ACTION_STATEMENT列中显示的文本一样。
- Timing: 'BEFORE'或'AFTER'两个值之一。
- Created: 目前，本列的值为NULL。

为了执行SHOW TRIGGERS，您必须拥有SUPER权限。

同时也见23.1.16节，“INFORMATION_SCHEMA TRIGGERS表”。

13.5.4.21. SHOW VARIABLES语法

```
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern']
```

SHOW VARIABLES显示了部门MySQL系统变量的值。本信息也可以使用**mysqladmin variables**命令获得。

使用GLOBAL选项，您可以获得被用于MySQL新连接的值。使用SESSION，您可以得到对于当前连接起效的值。如果您两个选项都不使用，默认值为SESSION。

LOCAL是SESSION的同义词。

如果默认值不合适，当mysqld启动时或在SET语句运行过程中，您可以使用命令行选项设置多数的这类变量。请参见5.3.1节，“**mysqld**命令行选项”和13.5.3节，“SET语法。

此处显示了部分的输出。对于您的服务器，变量和值的清单会有所不同。在5.3.3节，“服务器系统变量”中给出了每个变量的意义。在7.5.2节，“调节服务器参数”中提供了有关调整变量的信息。

```
mysql> SHOW VARIABLES;
+---------------------------------+-----------------------------------------------+
| Variable_name                   | Value                                         |
+---------------------------------+-----------------------------------------------+
| auto_increment_increment        | 1                                             |
```

```
| auto_increment_offset           | 1                                          |
| automatic_sp_privileges         | ON                                         |
| back_log                        | 50                                         |
| basedir                         | /home/jon/bin/mysql-5.1/                   |
| binlog_cache_size               | 32768                                      |
| bulk_insert_buffer_size         | 8388608                                    |
| character_set_client            | latin1                                     |
| character_set_connection        | latin1                                     |
...                               ...
| max_user_connections            | 0                                          |
| max_write_lock_count            | 4294967295                                 |
| multi_range_count               | 256                                        |
| myisam_data_pointer_size        | 6                                          |
| myisam_max_sort_file_size       | 2147483647                                 |
| myisam_recover_options          | OFF                                        |
| myisam_repair_threads           | 1                                          |
| myisam_sort_buffer_size         | 8388608                                    |
| ndb_autoincrement_prefetch_sz   | 32                                         |
| ndb_cache_check_time            | 0                                          |
| ndb_force_send                  | ON                                         |
...                               ...
...
| time_zone                       | SYSTEM                                     |
| timed_mutexes                   | OFF                                        |
| tmp_table_size                  | 33554432                                   |
| tmpdir                          |                                            |
| transaction_alloc_block_size    | 8192                                       |
| transaction_prealloc_size       | 4096                                       |
| tx_isolation                    | REPEATABLE-READ                            |
| updatable_views_with_limit      | YES                                        |
| version                         | 5.1.2-alpha-log                            |
| version_comment                 | Source distribution                        |
| version_compile_machine         | i686                                       |
| version_compile_os              | suse-linux                                 |
| wait_timeout                    | 28800                                      |
+---------------------------------+--------------------------------------------+
```

使用LIKE子句，本语句只显示与样式相匹配的变量:

```
mysql> SHOW VARIABLES LIKE 'have%';
+-----------------------+----------+
```

```
| Variable_name          | Value     |
+------------------------+----------+
| have_archive           | YES       |
| have_bdb               | NO        |
| have_blackhole_engine | YES       |
| have_compress          | YES       |
| have_crypt             | YES       |
| have_csv               | YES       |
| have_example_engine    | NO        |
| have_federated_engine | NO        |
| have_geometry          | YES       |
| have_innodb            | YES       |
| have_isam              | NO        |
| have_ndbcluster        | DISABLED |
| have_openssl           | NO        |
| have_partition_engine | YES       |
| have_query_cache       | YES       |
| have_raid              | NO        |
| have_rtree_keys        | YES       |
| have_symlink           | YES       |
+------------------------+----------+
```

13.5.4.22. SHOW WARNINGS语法

```
SHOW WARNINGS [LIMIT [offset,] row_count]
SHOW COUNT(*) WARNINGS
```

SHOW WARNINGS显示由上一个生成消息的语句导致的错误、警告和注意消息。如果上一个使用表的语句未生成消息，则什么也不显示。SHOW ERRORS是其相关语句，只显示错误。请参见13.5.4.9节，“SHOW ERRORS语法”。

对于使用一个表的每个新语句，消息清单均重新设置。

SHOW COUNT(*) WARNINGS语句显示错误、警告和注意的总数。您也可以从warning_count变量中找回此数目。

```
SHOW COUNT(*) WARNINGS;
SELECT @@warning_count;
```

如果max_error_count系统变量设置得过低，以致于有的消息没有被存储，则warning_count值可能比由SHOW WARNINGS显示的消息数目要大。本节后部显示的例子展示了这类情况是如何发生的。

LIMIT子句具有与SELECT语句相同的语法。请参见13.2.7节，“SELECT语法”。

MySQL服务器会发回由上一个语句引起的错误、警告和注意的总数。如果您正在使用C API，则此值可以通过调用mysql_warning_count()来获得。请参见25.2.3.69节，“mysql_warning_count()”。

对于如LOAD DATA INFILE等语句和如INSERT, UPDATE, CREATE TABLE和ALTER TABLE等DML语句，会生成警告。

以下DROP TABLE语句会导致一个注意:

```
mysql> DROP TABLE IF EXISTS no_such_table;
mysql> SHOW WARNINGS;
+-------+------+-------------------------------+
| Level | Code | Message                       |
+-------+------+-------------------------------+
| Note  | 1051 | Unknown table 'no_such_table' |
+-------+------+-------------------------------+
```

以下是一个简单的例子，显示了对于CREATE TABLE的一个语法警告，和对于INSERT的转换警告:

```
mysql> CREATE TABLE t1 (a TINYINT NOT NULL, b CHAR(4)) TYPE=MyISAM;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1287
Message: 'TYPE=storage_engine' is deprecated, use
         'ENGINE=storage_engine' instead
1 row in set (0.00 sec)

mysql> INSERT INTO t1 VALUES(10,'mysql'),(NULL,'test'),
    -> (300,'Open Source');
Query OK, 3 rows affected, 4 warnings (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 4

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1265
Message: Data truncated for column 'b' at row 1
*************************** 2. row ***************************
  Level: Warning
   Code: 1263
Message: Data truncated, NULL supplied to NOT NULL column 'a' at row 2
*************************** 3. row ***************************
  Level: Warning
```

```
   Code: 1264
Message: Data truncated, out of range for column 'a' at row 3
*************************** 4. row ***************************
  Level: Warning
   Code: 1265
Message: Data truncated for column 'b' at row 3
4 rows in set (0.00 sec)
```

要存储的错误、警告和注意消息的最大数目由max_error_count系统变量控制。默认情况下，该值为64。要更改您想要存储的信息的数目，需更改max_error_count值。在下面的例子中，ALTER TABLE语句会产生三个警告消息，但是只有一个被存储，因为max_error_count被设置为1：

```
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> SELECT @@warning_count;
+-----------------+
| @@warning_count |
+-----------------+
|               3 |
+-----------------+
1 row in set (0.01 sec)

mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1263 | Data truncated for column 'b' at row 1 |
+---------+------+----------------------------------------+
```

```
1 row in set (0.00 sec)
```

要禁用警告，需把max_error_count设置为0。在此情况下，warning_count仍然指示有多少警告已经发生，但是这些消息不被存储。

您可以把SQL_NOTES会话变量设置为0，使"注意"级别的警告不被记录。

13.5.5. 其它管理语句

13.5.5.1. CACHE INDEX语法
13.5.5.2. FLUSH语法
13.5.5.3. KILL语法
13.5.5.4. LOAD INDEX INTO CACHE语法
13.5.5.5. RESET语法

13.5.5.1. CACHE INDEX语法

```
CACHE INDEX
  tbl_index_list [, tbl_index_list] ...
  IN key_cache_name

tbl_index_list:
  tbl_name [[INDEX|KEY] (index_name[, index_name] ...)]
```

CACHE INDEX语句把表索引分配给某个关键缓存。该语句只用于MyISAM表。

下列语句把索引从表t1, t2和t3分配到名为hot_cache的关键缓存：

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+---------+--------------------+----------+----------+
| Table   | Op                 | Msg_type | Msg_text |
+---------+--------------------+----------+----------+
| test.t1 | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
| test.t3 | assign_to_keycache | status   | OK       |
+---------+--------------------+----------+----------+
```

CACHE INDEX语法允许您指定，只有来自表的特定索引应被分配给缓存。但是，当前的实施会把所有的表索引分配给缓存，所以必须指定表名称，不能指定其它的。

被引用到CACHE INDEX语句中的关键缓存可以这样创建，即通过使用一个参数设置语句或在服务器参数设置中设置其大小。举例说明：

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

关键缓存参数可以被作为一个结构化系统变量的成分进行访问。请参见9.4.1节，"结构式系统变量"。

在您可以把索引分配给一个关键缓存以前，缓存必须存在：

```
mysql> CACHE INDEX t1 IN non_existent_cache;
```

```
ERROR 1284 (HY000): Unknown key cache 'non_existent_cache'
```

默认情况下，表索引被分配给在服务器启动时被创建的主（默认）键缓存。当一个键高速缓冲被破坏时，所有被分配到此缓存中的索引会再次被分配给默认的 键高速缓冲。

索引的分配会对服务器产生全局性影响：如果一个客户端把一个索引分配给一个给定的缓存，则不论什么客户端发布查询，本缓存都被用于所有涉及索引的查询。

13.5.5.2. FLUSH语法

```
FLUSH [LOCAL | NO_WRITE_TO_BINLOG] flush_option [, flush_option] ...
```

如果您想要清除MySQL使用的部分内部缓存，您应该使用FLUSH语句。要执行FLUSH，您必须拥有RELOAD权限。

flush_option可以为以下的任何一个：

- HOSTS

用于清空主机缓存表。如果有的主机更改了IP号或如果您得到了错误信息Host *host_name* is blocked，则您应该刷新主机表。当在连接到MySQL服务器时，如果对于一个给定的主机，接连出现错误"多于max_connect_errors"，此时，MySQL会假定出现了错误，并阻止主机后续的连接申请。刷新主机表允许主机尝试再次连接。请参见A.2.5节，"主机的host_name被屏蔽"。您可以使用max_connect_errors=999999999启动**mysqld**，以避免此错误信息。

- DES_KEY_FILE

用于在服务器启动时，从采用--des-key-file选项指定的文件重新载入DES关键字。

- LOGS

用于关闭并重新打开所有的日志文件。如果您已经指定了一个更新日志文件或一个二进制日志文件，同时没有扩展，则相对于前一个文件，日志文件的扩展号增加1。如果您在文件名称中使用了一个扩展，则MySQL会关闭并重新打开日志文件。在Unix中，当相mysqld服务器发送一个SIGHUP信号时，也会如此（例外情况是部分Mac OS X 10.3版本。在这些版本中，mysqld忽略SIGHUP和SIGQUIT）。

如果服务器使用--log-error选项，则FLUSH LOGS会导致错误日志被重命名（使用后缀-old），同时mysqld会创建一个新的空日志文件。如果没有给定--log-error选项，则不会进行重命名。

- PRIVILEGES

用于从mysql数据库中的授权表重新载入权限。

- QUERY CACHE

对查询缓存进行整理碎片，以更好得利用存储器。与RESET QUERY CACHE不同，本语句不会从缓存中取消任何查询。

- STATUS

用于把多数状态变量重新设置为零。只在当调试查询时，您才应该使用此项。请参见1.7.1.3节，"如何通报缺陷和问题"。

- {TABLE | TABLES} [*tbl_name* [, *tbl_name*] ...]

当没有表被命名时，关闭所有打开的表，并迫使所有正在使用的表关闭。这也会刷新查询缓存。此项含有一个或多个表名称，只刷新给定的表。和RESET QUERY CACHE语句一样，FLUSH TABLES还会取消来自查询缓存的所有查询结果。

· TABLES WITH READ LOCK

对于所有带读取锁定的数据库，关闭所有打开的表，并锁定所有的表，直到您执行UNLOCK TABLES为止。如果您拥有一个可以及时进行快照的文件系统，比如Veritas，则这是进行备份的非常方便的方法。

· USER_RESOURCES

用于把所有每小时用户资源重新设置为零。这可以使已经达到了每小时连接、查询或更新限值的客户端立刻重新恢复活性。FLUSH USER_RESOURCES不适用于同时连接的最大限值。请参见13.5.1.3节，“GRANT和REVOKE语法”。

FLUSH语句被写入二进制日志，除非使用了自选的NO_WRITE_TO_BINLOG关键字（或其别名LOCAL）。 注释：在任何情况下，FLUSH LOGS, FLUSH MASTER, FLUSH SLAVE和FLUSH TABLES WITH READ LOCK都不会被记入日志，因为如果它们被复制到一个从属服务器上，会导致出现问题。

您也可以使用flush-hosts, flush-logs, flush-privileges, flush-status或flush-tables命令访问含有mysqladmin应用程序的语句。

注释：在MySQL 5.1.2-alpha中，不可能在已存储的函数或触发程序中发布FLUSH语句。不过，您可以在已存储的过程中使用FLUSH，只要它们不会从已存储的函数或触发程序中被调用。请参见I.1节，“对存储子程序和触发程序的限制”。

要了解有关RESET语句与复制同时使用的信息，也可以见13.5.5.5节，“RESET语法”。

13.5.5.3. KILL语法

```
KILL [CONNECTION | QUERY] thread_id
```

每个与mysqld的连接都在一个独立的线程里运行，您可以使用SHOW PROCESSLIST语句查看哪些线程正在运行，并使用KILL *thread_id*语句终止一个线程。

KILL允许自选的CONNECTION或QUERY修改符：

· KILL CONNECTION与不含修改符的KILL一样：它会终止与给定的*thread_id*有关的连接。

· KILL QUERY会终止连接当前正在执行的语句，但是会保持连接的原状。

如果您拥有PROCESS权限，则您可以查看所有线程。如果您拥有SUPER权限，您可以终止所有线程和语句。否则，您只能查看和终止您自己的线程和语句。

您也可以使用**mysqladmin processlist**和**mysqladmin kill**命令来检查和终止线程。

注释：您不能同时使用KILL和Embedded MySQL Server库，因为内植的服务器只运行主机应用程序的线程。它不能创建任何自身的连接线程。

当您进行一个KILL时，对线程设置一个特有的终止标记。在多数情况下，线程终止可能要花一些时间，这是因为终止标记只会在在特定的间隔被检查：

· 在SELECT, ORDER BY和GROUP BY循环中，在读取一组行后检查标记。如果设置了终止标记，则该语句被放弃。

- 在ALTER TABLE过程中，在每组行从原来的表中被读取前，检查终止标记。如果设置了终止标记，则语句被放弃，临时表被删除。
- 在UPDATE或DELETE运行期间，在每个组读取之后以及每个已更行或已删除的行之后，检查终止标记。如果终止标记被设置，则该语句被放弃。注意，如果您正在使用事务，则变更不会被 回滚。
- GET_LOCK()会放弃和返回NULL。
- INSERT DELAYED线程会快速地刷新（插入）它在存储器中的所有的行，然后终止。
- 如果线程在表锁定管理程序中（状态：锁定），则表锁定被快速地放弃。
- 如果在写入调用中，线程正在等待空闲的磁盘空间，则写入被放弃，并伴随"disk full"错误消息。
- 警告：对MyISAM表终止一个REPAIR TABLE或OPTIMIZE TABLE操作会导致出现一个被损坏的没有用的表。对这样的表的任何读取或写入都会失败，直到您再次优化或修复它（不中断）。

13.5.5.4. LOAD INDEX INTO CACHE语法

```
LOAD INDEX INTO CACHE
  tbl_index_list [, tbl_index_list] ...


tbl_index_list:
  tbl_name
    [[INDEX|KEY] (index_name[, index_name] ...)]
    [IGNORE LEAVES]
```

LOAD INDEX INTO CACHE语句会把一个表索引预载入到某个关键缓存中。它已经被一个明确的CACHE INDEX语句分配到此关键缓存中。或者，表索引被预载入到默认的关键缓存中。LOAD INDEX INTO CACHE只用于MyISAM表。

IGNORE LEAVES修改符只会导致索引的非叶子节点被预载入。

对于表t1和t2，以下语句会预载入索引的节点（索引组）：

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+---------+--------------+----------+----------+
| Table   | Op           | Msg_type | Msg_text |
+---------+--------------+----------+----------+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+---------+--------------+----------+----------+
```

本语句会预载入所有来自t1的索引组。它只预载入来自t2的非叶子节点的组。

LOAD INDEX INTO CACHE语法允许您指定，只有来自表的特定的索引应被预载入。但是，当前实施会把所有的表索引预载入缓存中，所以一定要指定表名称，不能指定其它的。

13.5.5.5. RESET语法

```
RESET reset_option [, reset_option] ...
```

RESET语句被用于清除不同的服务器操作的状态。它也作为FLUSH语句的更强大的版本。请参见13.5.5.2节，“FLUSH语法”。

为了执行RESET，您必须拥有RELOAD权限。

reset_option可以为以下的任何一项：

- MASTER

可以删除列于索引文件中的所有二进制日志，把二进制日志索引文件重新设置为空，并创建一个新的二进制日志文件。（在以前版本的MySQL中，被称为FLUSH MASTER。）见13.6.1节，“用于控制主服务器的SQL语句”。

- QUERY CACHE

从查询缓存中取消所有的查询结果。

- SLAVE

可以使从属服务器忘记其在主服务器二进制日志中的复制位置，另外，也可以通过删除原有的中继日志文件和开始一个新文件来重新设置中继日志。请参见13.6.2节，“用于控制从服务器的SQL语句”。

13.6. 复制语句

13.6.1. 用于控制主服务器的SQL语句
13.6.2. 用于控制从服务器的SQL语句

本节叙述了与复制有关的SQL语句。一组语句被用于控制主服务器。其它的被用于控制从属服务器。

13.6.1. 用于控制主服务器的SQL语句

13.6.1.1. PURGE MASTER LOGS语法
13.6.1.2. RESET MASTER语法
13.6.1.3. SET SQL_LOG_BIN语法
13.6.1.4. SHOW BINLOG EVENTS语法
13.6.1.5. SHOW MASTER LOGS语法
13.6.1.6. SHOW MASTER STATUS语法
13.6.1.7. SHOW SLAVE HOSTS语法

可以通过SQL界面控制复制。本节讨论了用于管理主复制服务器的语句。13.6.2节，“用于控制从服务器的SQL语句”讨论了用于管理从属服务器的语句。

13.6.1.1. PURGE MASTER LOGS语法

```
PURGE {MASTER | BINARY} LOGS TO 'log_name'

PURGE {MASTER | BINARY} LOGS BEFORE 'date'
```

用于删除列于在指定的日志或日期之前的日志索引中的所有二进制日志。这些日志也会从记录在日志索引文件中的清单中被删除，这样被给定的日志成为第一个。

例如:

```
PURGE MASTER LOGS TO 'mysql-bin.010';

PURGE MASTER LOGS BEFORE '2003-04-02 22:46:26';
```

BEFORE变量的date自变量可以为'YYYY-MM-DD hh:mm:ss'格式。MASTER和BINARY是同义词。

如果您有一个活性的从属服务器，该服务器当前正在读取您正在试图删除的日志之一，则本语句不会起作用，而是会失败，并伴随一个错误。不过，如果从属服务器是休止的，并且您碰巧清理了其想要读取的日志之一，则从属服务器启动后不能复制。当从属服务器正在复制时，本语句可以安全运行。您不需要停止它们。

要清理日志，需按照以下步骤:

1. 在每个从属服务器上，使用SHOW SLAVE STATUS来检查它正在读取哪个日志。

2. 使用SHOW MASTER LOGS获得主服务器上的一系列日志。

3. 在所有的从属服务器中判定最早的日志。这个是目标日志。如果所有的从属服务器是更新的，这是清单上的最后一个日志。

4. 制作您将要删除的所有日志的备份。（这个步骤是自选的，但是建议采用。）

5. 清理所有的日志，但是不包括目标日志。

13.6.1.2. RESET MASTER语法

```
RESET MASTER
```

可以删除列于索引文件中的所有二进制日志，把二进制日志索引文件重新设置为空，并创建一个新的二进制日志文件。

13.6.1.3. SET SQL_LOG_BIN语法

```
SET SQL_LOG_BIN = {0|1}
```

如果客户端使用一个有SUPER权限的账户连接，则可以禁用或启用当前连接的二进制日志记录。如果客户端没有此权限，则语句被拒绝，并伴随有错误。

13.6.1.4. SHOW BINLOG EVENTS语法

```
SHOW BINLOG EVENTS
   [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
```

用于在二进制日志中显示事件。如果您不指定'log_name'，则显示第一个二进制日志。

LIMIT子句和SELECT语句具有相同的语法。请参见13.2.7节，“SELECT语法”。

注释：当服务器把二进制日志的完整内容（该日志包括多数的由MySQL执行的查询）转储到stdout时，发布一个不含LIMIT子句的SHOW BINLOG EVENTS可以启动一个过程，该过程非常消耗时间并消耗资源。要把二进制日志保存到一个文本文件中，用于以后的检查和分析，需使用mysqlbinlog应用程序。请参见8.6节，“mysqlbinlog：用于处理二进制日志文件的实用工具”。

13.6.1.5. SHOW MASTER LOGS语法

```
SHOW MASTER LOGS
SHOW BINARY LOGS
```

用于列出服务器中的二进制日志文件。本语句被用作13.6.1.1节，“PURGE MASTER LOGS语法”中所述的过程的一部分，用于确定哪些日志可以被清理。

```
mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000015 |    724935 |
| binlog.000016 |    733481 |
+---------------+-----------+
```

SHOW BINARY LOGS与SHOW MASTER LOGS相当。

13.6.1.6. SHOW MASTER STATUS语法

```
SHOW MASTER STATUS
```

用于提供主服务器二进制日志文件的状态信息。例如：

```
mysql > SHOW MASTER STATUS;
+---------------+----------+--------------+------------------+
| File          | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+---------------+----------+--------------+------------------+
| mysql-bin.003 | 73       | test         | manual,mysql     |
+---------------+----------+--------------+------------------+
```

13.6.1.7. SHOW SLAVE HOSTS语法

```
SHOW SLAVE HOSTS
```

用于显示当前使用主服务器注册的复制从属服务器的清单。不以--report-host=*slave_name*选项为开头的从属服务器不会显示在本清单中。

13.6.2. 用于控制从服务器的SQL语句

13.6.2.1. CHANGE MASTER TO语法
13.6.2.2. LOAD DATA FROM MASTER语法
13.6.2.3. LOAD TABLE *tbl_name* FROM MASTER语法
13.6.2.4. MASTER_POS_WAIT()语法
13.6.2.5. RESET SLAVE语法
13.6.2.6. SET GLOBAL SQL_SLAVE_SKIP_COUNTER语法
13.6.2.7. SHOW SLAVE STATUS语法
13.6.2.8. START SLAVE语法

13.6.2.9. STOP SLAVE语法

复制操作可以通过SQL界面控制。本节讨论了用于管理从属复制服务器的语句。13.6.1节，“用于控制主服务器的SQL语句”讨论了用于管理主服务器的语句。

13.6.2.1. CHANGE MASTER TO语法

```
  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'
```

可以更改从属服务器用于与主服务器进行连接和通讯的参数。

MASTER_USER, MASTER_PASSWORD, MASTER_SSL, MASTER_SSL_CA, MASTER_SSL_CAPATH, MASTER_SSL_CERT, MASTER_SSL_KEY和MASTER_SSL_CIPHER用于向从属服务器提供有关如何与主服务器连接的信息。

即使对于在编译时没有SSL支持的从属服务器，SSL选项(MASTER_SSL, MASTER_SSL_CA, MASTER_SSL_CAPATH, MASTER_SSL_CERT, MASTER_SSL_KEY和MASTER_SSL_CIPHER)也可以被更改。它们被保存到master.info文件中，但是会被忽略，直到您使用一个SSL支持已启用的服务器。

如果您不指定一个给定的参数，则它会保持其原有的值。例外情况在后面的讨论中进行了说明。举例说明，如果用于连接到您的MySQL主服务器的 密码被更改了，您只需发布这些语句，就可以告知从属服务器新的密码：

```
mysql> STOP SLAVE; -- if replication was running

mysql> CHANGE MASTER TO MASTER_PASSWORD='new3cret';

mysql> START SLAVE; -- if you want to restart replication
```

没有必要指定没有改变的参数（主机、接口、用户等）。

MASTER_HOST和MASTER_PORT是主服务器主机和其TCP/IP接口的主机名（或IP地址）。注意，如果MASTER_HOST与localhost相等，那么，和MySQL的其它部分一样，接口可以被忽略（例如，如果可以使用Unix插槽文件）。

如果您指定了MASTER_HOST或MASTER_PORT，则从属服务器会假定主服务器与以前不一样（即使您指定的主机或接口值与当前值是一样的。）在此情况下，主服务器二进制日志的名称和位置的原有值不再适用，因此，如果您不指定语句中的MASTER_LOG_FILE和MASTER_LOG_POS，MASTER_LOG_FILE="和MASTER_LOG_POS=4会被静默地添加。

在MASTER_LOG_FILE和MASTER_LOG_POS坐标点，从属服务器I/O线程在启动之后从主服务器读取。如果您只指定了其中一个，则从属服务器不能指定RELAY_LOG_FILE或RELAY_LOG_POS。如果MSATER_LOG_FILE和MASTER_LOG_POS都没有被指定，则从属服务器会使用在CHANGE MASTER被发布前的最后一个slave SQL thread坐标。当您只想改变要使用的 密码时，这可以确保复制的连续性。即使从属服务器SQL线程落后于从属服务器I/O线程，也可以确保复制的连续性。

CHANGE MASTER会删除所有的中继日志文件并启动一个新的日志，除非您指定了RELAY_LOG_FILE或RELAY_LOG_POS。在此情况下，中继日志被保持；relay_log_purge全局变量被静默地设置为0。

CHANGE MASTER TO可以更新master.info和relay-log.info文件的内容。

当您拥有主服务器快照并拥有日志和对应的偏移量时，CHANGE MASTER对于设置从属服务器是有用的。在把快照载入从属服务器之后，您可以在从属服务器上运行CHANGE MASTER TO MASTER_LOG_FILE='*log_name_on_master*', MASTER_LOG_POS=*log_offset_on_master*。

举例说明:

```
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;
```

第一个例子可以更改主服务器及其二进制日志坐标。当想要设置从属服务器来复制主服务器时使用。

第二个例子显示了较少被使用的一个操作。当从属服务器含有中继日志，并且您出于某种原因想要执行此日志时使用。要这么做时，不需要连接主服务器。您只需要使用CHANGE MASTER TO并启动SQL线程（START SLAVE SQL_THREAD）。

您甚至可以在一个用于独立非从属服务器的非复制型设置中使用第二种操作，在崩溃之后进行复原。假设您的服务器已崩溃，同时您已恢复了备份。您想要重新播放服务器自己的二进制日志（不是中继日志，而是正规的

二进制文件），例如名为myhost-bin.*。首先，应在安全的地方制作这些二进制日志的备份，以防您没有完全遵守以下步骤，意外地让服务器清理了二进制文件。使用SET GLOBAL relay_log_purge=0，进一步增加安全性。然后启动不含--log-bin选项的服务器。使用--replicate-same-server-id, --relay-log=myhost-bin（让服务器相信，这些正规的二进制日志是中继日志）和--skip-slave-start options选项。当服务器启动后，发布以下语句：

```
mysql> CHANGE MASTER TO
    ->     RELAY_LOG_FILE='myhost-bin.153',
    ->     RELAY_LOG_POS=410,
    ->     MASTER_HOST='some_dummy_string';
mysql> START SLAVE SQL_THREAD;
```

服务器会读取并执行自己的二进制日志，完成崩溃复原。当复原完成后，运行STOP SLAVE，关闭服务器，删除master.info和relay-log.info，并使用原来的选项重新启动服务器。

要让服务器认为它是一个从属服务器，需要指定MASTER_HOST（甚至使用假值）。

13.6.2.2. LOAD DATA FROM MASTER语法

```
LOAD DATA FROM MASTER
```

本命令用于对主服务器进行快照，并拷贝到从属服务器上。它可以更新MASTER_LOG_FILE和MASTER_LOG_POS的值，这样，从属服务器就可以从正确的位置开始进行复制。使用--replicate-*-do-*和--replicate-*-ignore-*选项指定的表和数据库排除规则均被兑现。--replicate-rewrite-db没有被考虑。这是因为使用本选项，用户就可以设置一个例如--replicate-rewrite-db=db1->db3和--replicate-rewrite-db=db2->db3的非唯一映射。当从主服务器载入表时，该映射会使从属服务器发生混淆。

本语句的使用受以下条件的制约：

- 只对MyISAM表起作用。如果试图载入一个非MyISAM表，会导致以下错误：

```
ERROR 1189 (08S01): Net error reading from master
```

- 当拍摄快照时，会获得对主服务器的全局读取锁定。在载入操作期间，该锁定会阻止对主服务器的更新。

如果您正在载入大表，您可能必须对主服务器和从属服务器均增加net_read_timeout和net_write_timeout值。请参见5.3.3节，“服务器系统变量”。

注意，LOAD DATA FROM MASTER不从mysql数据库拷贝任何表。这可以更容易地让主服务器和从属服务器拥有不同的用户和权限。

LOAD DATA FROM MASTER语句要求用于连接主服务器的复制帐户，以便让主服务器拥有RELOAD和SUPER权限，并让所有您想要载入的主服务器表拥有SELECT权限。所有的用户不拥有SELECT权限的主服务器表均被LOAD DATA FROM MASTER忽略。这是因为主服务器会对用户隐藏它们：LOAD DATA FROM MASTER会调用SHOW DATABASES以了解要载入的主服务器数据库，但是SHOW DATABASES只会返回用户有部分权限的数据库。请参见13.5.4.6节，“SHOW DATABASES语法”。在从属服务器方面，发布LOAD DATA FROM MASTER的用户应拥有授权，以取消或创建被复制的数据库和表。

13.6.2.3. LOAD TABLE *tbl_name* FROM MASTER语法

```
LOAD TABLE tbl_name FROM MASTER
```

用于把表的拷贝从主服务器转移到从属服务器。本语句的主要作用是调试LOAD DATA FROM MASTER。它要求用于连接主服务器的帐户拥有对主服务器的RELOAD和SUPER权限，并拥有对要载入的主服务器表的SELECT权限。在从属服务器方面，发布LOAD TABLE FROM MASTER的用户应拥有取消和创建表的权限。

用于LOAD DATA FROM MASTER的条件也适用于这里。举例说明，LOAD TABLE FROM MASTER仅对于MyISAM表起作用。对LOAD DATA FROM MASTER的暂停注意也适用。

13.6.2.4. MASTER_POS_WAIT()语法

```
SELECT MASTER_POS_WAIT('master_log_file', master_log_pos)
```

这实际上是一个函数，而不是一个语句。它被用于确认，从属服务器已读取并执行了到达主服务器二进制日志的给定位置。要了解完整的描述，请参见12.9.4节，“其他函数”

13.6.2.5. RESET SLAVE语法

```
RESET SLAVE
```

用于让从属服务器忘记其在主服务器的二进制日志中的复制位置。本语句被用于进行一个明确的启动：它会删除master.info和relay-log.info文件，以及所有的中继日志，并启动一个新的中继日志。

注释：所有的中继日志被删除，即使它们没有被从属服务器SQL线程完全的执行。（如果您已经发布了一个SLAVE语句或如果从属服务器的载入量很大，则这对于一个复制从属服务器是一个很可能出现的情况。）

存储在master.info文件中的连接信息通过使用在对应的启动选项中指定的值，被立即重新设置了。此信息包括主服务器主机、主服务器接口、主服务器用户和主服务器 密码等值。当从属服务器SQL线程被中止时，它位于正在复制的临时表的中间，并且发布了RESET SLAVE，则已被复制的临时表在从属服务器中被删除。

13.6.2.6. SET GLOBAL SQL_SLAVE_SKIP_COUNTER语法

```
SET GLOBAL SQL_SLAVE_SKIP_COUNTER = n
```

从主服务器中跳过后面的n个事件。要复原由语句导致的复制中止，这是有用的。

仅当从属线程没有正在运行时，本语句时有效的。否则，会产生一个错误。

13.6.2.7. SHOW SLAVE STATUS语法

```
SHOW SLAVE STATUS
```

用于提供有关从属服务器线程的关键参数的信息。如果您使用mysql客户端发布此语句，则您可以使用一个\G语句终止符来获得更便于阅读的竖向版面，而不是使用分号：

```
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
```

SHOW SLAVE STATUS会返回以下字段:

- Slave_IO_State

SHOW PROCESSLIST输出的State字段的拷贝。SHOW PROCESSLIST用于从属I/O线程。如果线程正在试图连接到主服务器，正在等待来自主服务器的时间或正在连接到主服务器等，本语句会通知您。在6.3节，“复制实施细节”中列出了可能的状态。旧版本的MySQL在连接主服务器不成功时，允许线程继续运行。对于旧版本的MySQL，观看此字段是必须的。如果它正在运行，则无问题；如果它没有运行，则您会在Last_Error字段中发现错误（后面有说明）。

- Master_Host

当前的主服务器主机。

- Master_User

被用于连接主服务器的当前用户。

- Master_Port

当前的主服务器接口。

· Connect_Retry

--master-connect-retry选项的当前值

· Master_Log_File

I/O线程当前正在读取的主服务器二进制日志文件的名称。

· Read_Master_Log_Pos

在当前的主服务器二进制日志中，I/O线程已经读取的位置。

· Relay_Log_File

SQL线程当前正在读取和执行的中继日志文件的名称。

· Relay_Log_Pos

在当前的中继日志中，SQL线程已读取和执行的位置。

· Relay_Master_Log_File

由SQL线程执行的包含多数近期事件的主服务器二进制日志文件的名称。

· Slave_IO_Running

I/O线程是否被启动并成功地连接到主服务器上。对于旧版本的MySQL（在4.1.14和5.0.12之前），如果I/O线程已被启动，即使从属服务器仍没有连接到主服务器上，Slave_IO_Running也将被设置到YES。

· Slave_SQL_Running

SQL线程是否被启动。

· Replicate_Do_DB, Replicate_Ignore_DB

使用--replicate-do-db和--replicate-ignore-db选项指定的数据库清单。

· Replicate_Do_Table, Replicate_Ignore_Table, Replicate_Wild_Do_Table, Replicate_Wild_Ignore_Table

使用--replicate-do-table, --replicate-ignore-table, --replicate-wild-do-table和--replicate-wild-ignore_table选项指定的表清单。

· Last_Errno, Last_Error

被多数最近被执行的查询返回的错误数量和错误消息。错误数量为0并且消息为空字符串意味着"没有错误"。如果Last_Error值不是空值，它也会在从属服务器的错误日志中作为消息显示。

举例说明：

```
Last_Errno: 1051

Last_Error: error 'Unknown table 'z'' on query 'drop table z'
```

该消息指示，表z曾经存在于在主服务器中并已被取消了，但是它没有在从属服务器中存在过，因此对于从属服务器，DROP TABLE失败。（举例说明，在设置复制时，如果您忘记了把此表拷贝到从属服务器中，则这有可能发生。）

· Skip_Counter

最近被使用的用于SQL_SLAVE_SKIP_COUNTER的值。

· Exec_Master_Log_Pos

来自主服务器的二进制日志的由SQL线程执行的上一个时间的位置（Relay_Master_Log_File）。在主服务器的二进制日志中的(Relay_Master_Log_File, Exec_Master_Log_Pos)对应于在中继日志中的(Relay_Log_File, Relay_Log_Pos)。

· Relay_Log_Space

所有原有的中继日志结合起来的总大小。

· Until_Condition, Until_Log_File, Until_Log_Pos

在START SLAVE语句的UNTIL子句中指定的值。

Until_Condition具有以下值:

o 如果没有指定UNTIL子句，则没有值

o 如果从属服务器正在读取，直到达到主服务器的二进制日志的给定位置为止，则值为Master

o 如果从属服务器正在读取，直到达到其中继日志的给定位置为止，则值为Relay

Until_Log_File和Until_Log_Pos用于指示日志文件名和位置值。日志文件名和位置值定义了SQL线程在哪个点中止执行。

· Master_SSL_Allowed, Master_SSL_CA_File, Master_SSL_CA_Path, Master_SSL_Cert, Master_SSL_Cipher, Master_SSL_Key

这些字段显示了被从属服务器使用的参数。这些参数用于连接主服务器。

Master_SSL_Allowed具有以下值:

o 如果允许对主服务器进行SSL连接，则值为Yes

o 如果不允许对主服务器进行SSL连接，则值为No

o 如果允许SSL连接，但是从属服务器没有让SSL支持被启用，则值为Ignored。

与SSL有关的字段的值对应于--master-ca, --master-capath, --master-cert, --master-cipher和--master-key选项的值。

· Seconds_Behind_Master

本字段是从属服务器"落后"多少的一个指示。当从属SQL线程正在运行时（处理更新），本字段为在主服务器上由此线程执行的最近的一个事件的时间标记开始，已经过的秒数。当此线程被从属服务器I/O线程赶上，并进入闲置状态，等待来自I/O线程的更多的事件时，本字段为零。总之，本字段测量从属服务器SQL线程和从属服务器I/O线程之间的时间差距，单位以秒计。

如果主服务器和从属服务器之间的网络连接较快，则从属服务器I/O线程会非常接近主服务器，所以本字段能够十分近似地指示，从属服务器SQL线程比主服务器落后多少。如果网络较慢，则这种指示不准确；从属SQL线程经常会赶上读取速度较慢地从属服务器I/O线程，因此，Seconds_Behind_Master经常显示值为0。即使I/O线程落后于主服务器时，也是如此。换句话说，本列只对速度快的网络有用。

即使主服务器和从属服务器不具有相同的时钟，时间差计算也会起作用（当从属服务器I/O线程启动时，计算时间差。并假定从此时以后，时间差保持不变）。如果从属SQL线程不运行，或者如果从属服务器I/O线程不运行或未与主服务器连接，则Seconds_Behind_Master为NULL（意义为"未知"）。举例说明，如果在重新连接之前，从属服务器I/O线程休眠了master-connect-retry秒，则显示NULL，因为从属服务器不知道主服务器正在做什么，也不能有把握地说落后多少。

本字段有一个限制。时间标记通过复制被保留，这意味着，如果一个主服务器M1本身是一个从属服务器M0，则来自M1的binlog的任何事件（通过复制来自M0的binlog的事件而产生），与原事件具有相同的时间标记。这可以使MySQL成功地复制TIMESTAMP。但是，Seconds_Behind_Master的缺点是，如果M1也收到来自客户端的直接更新，则值会随机变化，因为有时最近的M1时间来自M0，有时来自直接更新，最近的时间标记也是如此。

13.6.2.8. START SLAVE语法

```
START SLAVE [thread_type [, thread_type] ... ]

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_type: IO_THREAD | SQL_THREAD
```

不含选项的START SLAVE会同时启动两个从属服务器线程。I/O线程从主服务器中读取查询，并把它们存储在中继日志中。SQL线程读取中继日志并执行查询。START SLAVE要求SUPER权限。

如果START SLAVE成功地启动了从属服务器线程，则会返回，不会出现错误。但是，即使在此情况下，也有可能出现这样的现象——服务器线程启动了，然后又停止了（例如，因为它们没有成功地连接到主服务器上，或者没有能读取二进制日志，或者出现了其它问题）。START SLAVE对此不会发出警告。您必须检查从属服务器的错误日志，查看是否有由从属服务器线程产生的错误消息，或者使用SHOW SLAVE STATUS检查它们是否运行正常。

您可以把IO_THREAD和SQL_THREAD选项添加到语句中，指明哪些线程将要启动。

可以添加一个UNTIL子句，指定从属服务器应启动并运行，直到SQL线程达到主服务器二进制日志中的一个给定点为止。当SQL线程达到此点时，它会停止。如果在该语句中指定了SQL_THREAD选项，则它只会启动SQL线程。否则，它会同时启动两个从属服务器线程。如果SQL线程正在运行，则UNTIL子句被忽略，并发布一个警告。

对于一个UNTIL子句，您必须同时指定一个日志文件名和位置。不要把主服务器和中继日志选项混合在一起。

UNTIL条件由一个后续的STOP SLAVE语句，或一个不包括UNTIL子句的START SLAVE语句，或一个服务器重启命令重新设置。

UNTIL子句对于调试复制操作是有用的，或者可用于促使复制操作继续，直到接近一个特定的点时为止，在此点，您想要避免让从属服务器复制一个语句。举例说明，如果在主服务上执行了一个不明智的DROP TABLE语句，您可以使用UNTIL来告知从属服务器，执行到此点就停止，不要再继续了。要查找该事件是什么，需对主服务器日志或从属中继日志使用mysqlbinlog，或通过使用SHOW BINLOG EVENTS语句。

如果您正在使用UNTIL，让从属服务器成段地处理已复制的查询，则建议您使用--skip-slave-start选项来启动从属服务器，以防止当从属服务器启动时，SQL线程运行。最好在一个选项文件中使用此选项，而不是在命令行中使用，这样，如果发生了意料外的服务器重新启动，它也不会被忘记。

SHOW SLAVE STATUS语句包括了输出字段。这些字段显示了UNTIL条件的当前值。

在以前版本的MySQL中，本语句被称为SLAVE START。在MySQL 5.1中仍然接受这种用法，以便与以前版本兼容。但现在不赞成使用。

13.6.2.9. STOP SLAVE语法

```
STOP SLAVE [thread_type [, thread_type] ... ]

thread_type: IO_THREAD | SQL_THREAD
```

用于中止从属服务器线程。STOP SLAVE要求SUPER权限。

和START SLAVE相似，本语句在使用时可以加IO_THREAD和SQL_THREAD选项，指明将被中止的线程。

在以前版本的MySQL中，本语句被称为SLAVE STOP。在MySQL 5.1中仍然接受这种用法，以便与以前版本兼容。但是现在不赞成使用。

13.7. 用于预处理语句的SQL语法

MySQL 5.1对服务器一方的预制语句提供支持。如果您使用合适的客户端编程界面，则这种支持可以发挥在MySQL 4.1中实施的高效客户端/服务器二进制协议的优势。候选界面包括MySQL C API客户端库（用于C程序）、MySQL Connector/J（用于Java程序）和MySQL Connector/NET。例如，C API可以提供一套能组成预制语句API的函数调用。请参见25.2.4节，“C API预处理语句”。其它语言界面可以对使用了二进制协议（通过在C客户端库中链接）的预制语句提供支持。有一个例子是PHP 5.0中的mysqli扩展。

对预制语句，还有一个SQL界面可以利用。与在整个预制语句API中使用二进制协议相比，本界面效率没有那么高，但是它不要求编程，因为在SQL层级，可以直接利用本界面：

- 当您无法利用编程界面时，您可以使用本界面。
- 有些程序允许您发送SQL语句到将被执行的服务器中，比如mysql客户端程序。您可以从这些程序中使用本界面。
- 即使客户端正在使用旧版本的客户端库，您也可以使用本界面。唯一的要求是，您能够连接到一个支持预制语句SQL语法的服务器上。

预制语句的SQL语法在以下情况下使用：

- 在编代码前，您想要测试预制语句在您的应用程序中运行得如何。或者也许一个应用程序在执行预制语句时有问题，您想要确定问题是什么。
- 您想要创建一个测试案例，该案例描述了您使用预制语句时出现的问题，以便您编制程序错误报告。
- 您需要使用预制语句，但是您无法使用支持预制语句的编程API。

预制语句的SQL语法基于三个SQL语句：

```
PREPARE stmt_name FROM preparable_stmt;

EXECUTE stmt_name [USING @var_name [, @var_name] ...];
```

```
{DEALLOCATE | DROP} PREPARE stmt_name;
```

PREPARE语句用于预备一个语句，并赋予它名称stmt_name，借此在以后引用该语句。语句名称对案例不敏感。preparable_stmt可以是一个文字字符串，也可以是一个包含了语句文本的用户变量。该文本必须展现一个单一的SQL语句，而不是多个语句。使用本语句，'?'字符可以被用于制作参数，以指示当您执行查询时，数据值在哪里与查询结合在一起。'?'字符不应加引号，即使您想要把它们与字符串值结合在一起，也不要加引号。参数制作符只能被用于数据值应该出现的地方，不用于SQL关键词和标识符等。

如果带有此名称的预制语句已经存在，则在新的语言被预备以前，它会被隐含地解除分配。这意味着，如果新语句包含一个错误并且不能被预备，则会返回一个错误，并且不存在带有给定名称语句。

预制语句的范围是客户端会话。在此会话内，语句被创建。其它客户端看不到它。

在预备了一个语句后，您可使用一个EXECUTE语句（该语句引用了预制语句名称）来执行它。如果预制语句包含任何参数制造符，则您必须提供一个列举了用户变量（其中包含要与参数结合的值）的USING子句。参数值只能有用户变量提供，USING子句必须准确地指明用户变量。用户变量的数目与语句中的参数制造符的数量一样多。

您可以多次执行一个给定的预制语句，在每次执行前，把不同的变量传递给它，或把变量设置为不同的值。

要对一个预制语句解除分配，需使用DEALLOCATE PREPARE语句。尝试在解除分配后执行一个预制语句会导致错误。

如果您终止了一个客户端会话，同时没有对以前已预制的语句解除分配，则服务器会自动解除分配。

以下SQL语句可以被用在预制语句中：CREATE TABLE, DELETE, DO, INSERT, REPLACE, SELECT, SET, UPDATE和多数的SHOW语句。目前不支持其它语句。

以下例子显示了预备一个语句的两种方法。该语句用于在给定了两个边的长度时，计算三角形的斜边。

第一个例子显示如何通过使用文字字符串来创建一个预制语句，以提供语句的文本：

```
mysql> PREPARE stmt1 FROM 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';

mysql> SET @a = 3;

mysql> SET @b = 4;

mysql> EXECUTE stmt1 USING @a, @b;

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

| hypotenuse |

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

|          5 |

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

mysql> DEALLOCATE PREPARE stmt1;
```

第二个例子是相似的，不同的是提供了语句的文本，作为一个用户变量：

```
mysql> SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';

mysql> PREPARE stmt2 FROM @s;

mysql> SET @a = 6;

mysql> SET @b = 8;
```

```
mysql> EXECUTE stmt2 USING @a, @b;
+------------+
| hypotenuse |
+------------+
|         10 |
+------------+
mysql> DEALLOCATE PREPARE stmt2;
```

预制语句的SQL语法不能被用于带嵌套的风格中。也就是说，被传递给PREPARE的语句本身不能是一个PREPARE, EXECUTE或DEALLOCATE PREPARE语句。

预制语句的SQL语法与使用预制语句API调用不同。例如，您不能使用mysql_stmt_prepare() C API函数来预备一个PREPARE, EXECUTE或DEALLOCATE PREPARE语句。

预制语句的SQL语法可以在已存储的过程中使用，但是不能在已存储的函数或触发程序中使用。

当使用预制语句时，可以在LIMIT子句中使用占位符。请参见13.2.7节，“SELECT语法”。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第14章：插件式存储引擎体系结构

目录

14.1. 前言
14.2. 概述
14.3. 公共MySQL数据库服务器层
14.4. 选择存储引擎
14.5. 将存储引擎指定给表
14.6. 存储引擎和事务
14.7. 插入存储引擎
14.8. 拔出存储引擎
14.9. 插件式存储器的安全含义

14.1. 前言

在MySQL 5.1中，MySQL AB引入了新的插件式存储引擎体系结构，允许将存储引擎加载到正在运新的MySQL服务器中。

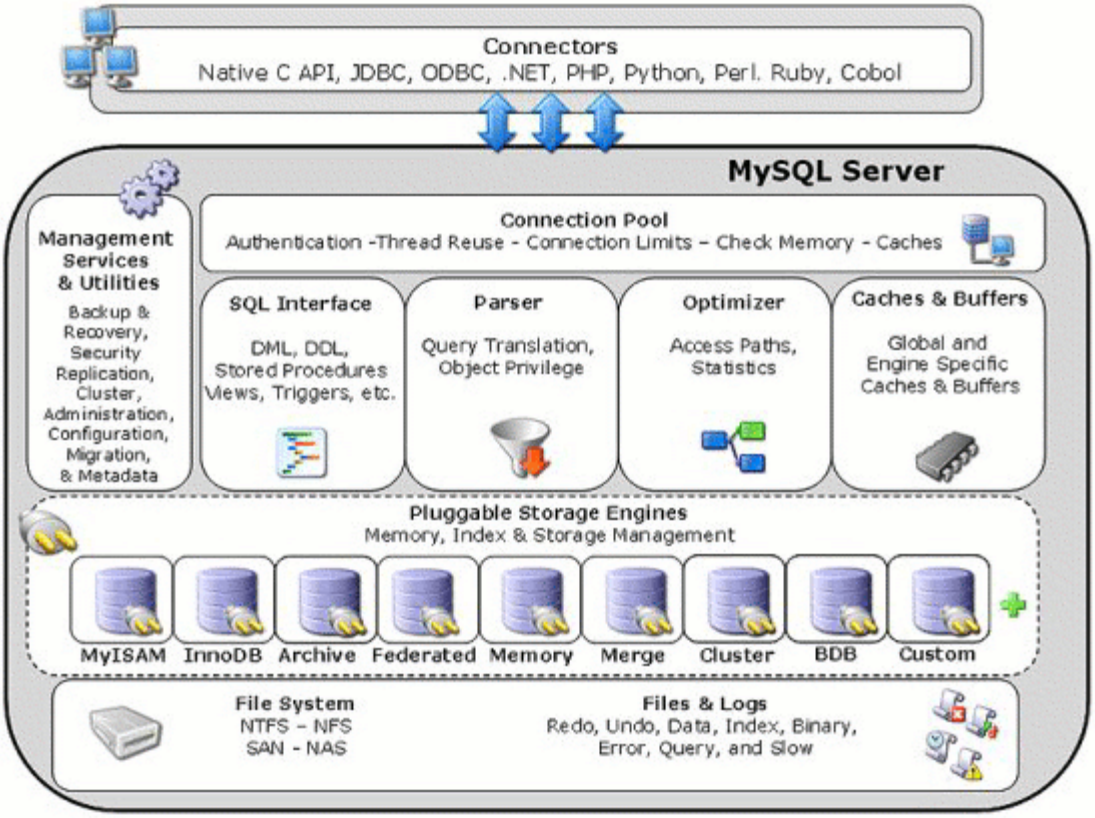
本章介绍了插件式存储引擎体系结构，概要介绍了与MySQL一起提供的各种存储引擎，介绍了将存储引擎赋给表的方法，以及插入和拔出存储引擎的方法。

14.2. 概述

使用MySQL插件式存储引擎体系结构，允许数据库专业人员为特定的应用需求选择专门的存储引擎，完全不需要管理任何特殊的应用编码要求。采用MySQL服务器体系结构，由于在存储级别上提供了一致和简单的应用模型和API，应用程序编程人员和DBA可不再考虑所有的底层实施细节。因此，尽管不同的存储引擎具有不同的能力，应用程序是与之分离的。

在下图中，以图形方式介绍了MySQL插件式存储引擎体系结构：

图**14.1**：**MySQL**插件式存储引擎的体系结构



插件式存储引擎体系结构提供了标准的管理和支持服务集合，它们对所有的基本存储引擎来说是共同的。存储引擎本身是数据库服务器的组件，负责对在物理服务器层面上维护的基本数据进行实际操作。

这是一种高效的模块化体系结构，它为那些希望专注于特定应用需求的人员提供了巨大的便利和益处，这类特殊应用需求包括数据仓储、事务处理、高可用性情形等，同时还能利用独立于任何存储引擎的一组接口和服务。

应用程序编程人员和DBA通过位于存储引擎之上的连接器API和服务层来处理MySQL数据库。如果应用程序的变化需要改变底层存储引擎，或需要增加1个或多个额外的存储引擎以支持新的需求，不需要进行大的编码或进程更改就能实现这类要求。MySQL服务器体系结构提供了一致和易于使用的API，这类API适用于多种存储引擎，通过该方式，该结构将应用程序与存储引擎的底层复杂性隔离开来。

14.3. 公共MySQL数据库服务器层

MySQL插件式存储引擎是MySQL数据库服务器中的组件，负责为数据库执行实际的数据I/O操作，并能允许和强制执行面向特殊应用需求的特定特性集合。使用特殊存储引擎的主要优点之一在于，仅需提供特殊应用所需的特性，因此，数据库中的系统开销较小，最终结果具有更有效和更高的数据库性能。这也是MySQL被始终视为具有高性能的原因之一，在行业标准基准方面，它能匹敌或击败专有的整体式数据库。

从技术角度上看，在存储引擎中，一些独特的支持底层结构的组件是什么呢？一些关键差别包括：

- 并发性：某些应用程序比其他应用程序具有很多的颗粒级锁定要求（如行级锁定）。选择正确的锁定策略能够减少开销，并有助于整体性能的提升。它还包括对多种能力的支持，如多版本并发性控制或"快照"读取等。

- 事务支持：并非所有的应用程序都需要事务，但对的确需要事务的应用程序来说，有着定义良好的需求，如ACID兼容等。

- 引用完整性：通过DDDL定义的 外键，服务器需要强制保持关联数据库的引用完整性。

· 物理存储：它包括各种各样的事项，从表和索引的总的页大小，到存储数据所需的格式，到物理磁盘。

· 索引支持：不同的应用程序倾向于采用不同的索引策略，每种存储引擎通常有自己的编制索引方法，但某些索引方法（如B-tree索引）对几乎所有的存储引擎来说是共同的。

· 内存高速缓冲：与其他应用程序相比，不同的应用程序对某些内存高速缓冲策略的响应更好，因此，尽管某些内存高速缓冲对所有存储引擎来说是共同的（如用于用户连接的高速缓冲，MySQL的高速查询高速缓冲等），其他高速缓冲策略仅当使用特殊的存储引擎时才唯一定义。

· 性能帮助：包括针对并行操作的多I/O线程，线程并发性，数据库检查点，成批插入处理等。

· 其他目标特性：可能包括对地理空间操作的支持，对特定数据处理操作的安全限制等。

每组插件式存储引擎基本组件均采用了相应的设计，能够为特定应用提供可选择的特性集合。从反面角度看，避免使用组件特性集合有助于避免不必要的开销。因此，显而易见，应理解特定应用程序的需求集合，并选择恰当的能大幅度改善系统整体效率和性能的MySQL存储引擎。

14.4. 选择存储引擎

与MySQL一起提供的各种存储引擎在设计时考虑了不同的使用情况。为了更有效地使用插件式存储体系结构，最好了解各种存储引擎的优点和缺点。

在下面的表格中，概要介绍了与MySQL一起提供的存储引擎：

图**14.2**：存储引擎比较

Feature	MyISAM	BDB	Memory	InnoDB	Archive	NDB
Storage Limits	No	No	Yes	64TB	No	Yes
Transactions (commit, rollback, etc.)		✔		✔		
Locking granularity	Table	Page	Table	Row	Row	Row
MVCC/Snapshot Read				✔	✔	✔
Geospatial support	✔					
B-Tree indexes	✔	✔	✔	✔		✔
Hash indexes			✔	✔		✔
Full text search index	✔					
Clustered index				✔		
Data Caches			✔	✔		✔
Index Caches	✔		✔	✔		✔
Compressed data	✔				✔	
Encrypted data (via function)	✔	✔	✔	✔	✔	✔
Storage cost (space used)	Low	Low	N/A	High	Very Low	Low
Memory cost	Low	Low	Medium	High	Low	High
Bulk Insert Speed	High	High	High	Low	Very High	High
Cluster database support						✔
Replication support	✔	✔	✔	✔	✔	✔
Foreign key support				✔		
Backup/Point-in-time recovery	✔	✔	✔	✔	✔	✔
Query cache support	✔	✔	✔	✔	✔	✔
Update Statistics for Data Dictionary	✔	✔	✔	✔	✔	✔

下述存储引擎是最常用的：

- MyISAM：默认的MySQL插件式存储引擎，它是在Web、数据仓储和其他应用环境下最常使用的存储引擎之一。注意，通过更改STORAGE_ENGINE配置变量，能够方便地更改MySQL服务器的默认存储引擎。

- InnoDB：用于事务处理应用程序，具有众多特性，包括ACID事务支持。

- BDB：可替代InnoDB的事务引擎，支持COMMIT、ROLLBACK和其他事务特性。

- Memory：将所有数据保存在RAM中，在需要快速查找引用和其他类似数据的环境下，可提供极快的访问。

- Merge：允许MySQL DBA或开发人员将一系列等同的MyISAM表以逻辑方式组合在一起，并作为1个对象引用它们。对于诸如数据仓储等VLDB环境十分适合。

- Archive：为大量很少引用的历史、归档、或安全审计信息的存储和检索提供了完美的解决方案。

- Federated：能够将多个分离的MySQL服务器链接起来，从多个物理服务器创建一个逻辑数据库。十分适合于分布式环境或数据集市环境。

- Cluster/NDB：MySQL的簇式数据库引擎，尤其适合于具有高性能查找要求的应用程序，这类查找需求还要求具有最高的正常工作时间和可用性。

- Other：其他存储引擎包括CSV（引用由逗号隔开的用作数据库表的文件），Blackhole（用于临时禁止对数据库的应用程序输入），以及Example引擎（可为快速创建定制的插件式存储引擎提供帮助）。

请记住，对于整个服务器或方案，你并不一定要使用相同的存储引擎，你可以为方案中的每个表使用不同的存储引擎，这点很重要。

关于MySQL中所包含存储引擎的详细信息，请参见14.4节，“选择存储引擎”。

14.5. 将存储引擎指定给表

可以在创建新表时指定存储引擎，或通过使用ALTER TABLE语句指定存储引擎。

要想在创建表时指定存储引擎，可使用ENGINE参数：

```
CREATE TABLE engineTest(
id INT
) ENGINE = MyISAM;
```

要想更改已有表的存储引擎，可使用ALTER TABLE语句：

```
ALTER TABLE engineTest ENGINE = ARCHIVE;
```

14.6. 存储引擎和事务

下述存储引擎支持事务：

- InnoDB：通过MVCC支持事务，允许COMMIT、ROLLBACK和保存点。

- NDB：通过MVCC支持事务，允许COMMIT和ROLLBACK。

- BDB：支持事务，允许COMMIT和ROLLBACK。

14.7. 插入存储引擎

能够使用存储引擎之前，必须使用INSTALL PLUGIN语句将存储引擎plugin（插件）装载到mysql。例如，要想加载example引擎，首先应加载ha_example.so模块：

```
INSTALL PLUGIN ha_example SONAME 'ha_example.so';
```

文件.so必须位于MySQL服务器库目录下（典型情况下是installdir/lib）。

14.8. 拔出存储引擎

要想拔出存储引擎，可使用UNINSTALL PLUGIN语句：

```
UNINSTALL PLUGIN ha_example;
```

如果拔出了正被已有表使用的存储引擎，这些表将成为不可访问的。拔出存储引擎之前，请确保没有任何表使用该存储引擎。

14.9. 插件式存储器的安全含义

为了安装插件式存储引擎，plugin文件必须位于恰当的MySQL库目录下，而且发出INSTALL PLUGIN语句的用户必须具有SUPER权限。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第15章：存储引擎和表类型

目录

15.1. MyISAM存储引擎

15.1.1. MyISAM启动选项

15.1.2.键所需的空间

15.1.3. MyISAM表的存储格式

15.1.4. MyISAM表的问题

15.2. InnoDB存储引擎

15.2.1. InnoDB概述

15.2.2. InnoDB联系信息

15.2.3. InnoDB配置

15.2.4. InnoDB启动选项

15.2.5. 创建InnoDB表空间

15.2.6. 创建InnoDB表

15.2.7. 添加和删除InnoDB数据和日志文件

15.2.8. InnoDB数据库的备份和恢复

15.2.9. 将InnoDB数据库移到另一台机器

15.2.10. InnoDB事务模型和锁定

15.2.11. InnoDB性能调节提示

15.2.12. 多版本的实施

15.2.13. 表和索引结构

15.2.14.文件空间管理和磁盘I/O

15.2.15. InnoDB错误处理

15.2.16. 对InnoDB表的限制

15.2.17. InnoDB故障诊断和排除

15.3. MERGE存储引擎

15.3.1. MERGE表 方面的问题

15.4. MEMORY(HEAP)存储引擎

15.5. BDB(BerkeleyDB)存储引擎

15.5.1. BDB支持的操作系统

15.5.2. 安装BDB

15.5.3. BDB启动选项

15.5.4. BDB表的特性

15.5.5. 修改BDB所需的事宜

15.5.6. 对BDB表的限制

15.5.7. 使用BDB表时可能 出现的错误

15.6. EXAMPLE存储引擎

15.7. FEDERATED存储引擎

15.7.1. 安装FEDERATED存储引擎

15.7.2. FEDERATED存储引擎的介绍

15.7.3. 如何使用FEDERATED表

15.7.4. FEDERATED存储引擎的局限

15.8. ARCHIVE存储引擎

15.9. CSV存储引擎

15.10.BLACKHOLE存储引擎

MySQL支持数个存储引擎作为对不同表的类型的处理器。MySQL存储引擎包括处理事务安全表的引擎和处理非事务安全表的引擎:

· MyISAM管理非事务表。它提供高速存储和检索，以及全文搜索能力。MyISAM在所有MySQL配置里被支持，它是默认的存储引擎，除非你配置MySQL默认使用另外一个引擎。

· MEMORY存储引擎提供"内存中"表。MERGE存储引擎允许集合将被处理同样的MyISAM表作为一个单独的表。就像MyISAM一样，MEMORY和MERGE存储引擎处理非事务表，这两个引擎也都被 默认包含在MySQL中。

注释：MEMORY存储引擎正式地被确定为HEAP引擎。

· InnoDB和BDB存储引擎提供事务安全表。BDB被包含在为支持它的操作系统发布的MySQL-Max二进制分发版里。InnoDB也 默认被包括在所有MySQL 5.1二进制分发版里，你可以按照喜好通过配置MySQL来允许或禁止任一引擎。

· EXAMPLE存储引擎是一个"存根"引擎，它不做什么。你可以用这个引擎创建表，但没有数据被存储于其中或从其中检索。这个引擎的目的是服务 ，在MySQL源代码中的一个例子，它演示说明如何开始编写新存储引擎。同样，它的主要兴趣是对开发者。

· NDB Cluster是被MySQL Cluster用来实现分割到多台计算机上的表的存储引擎。它在MySQL-Max 5.1二进制分发版里提供。这个存储引擎当前只被Linux, Solaris, 和Mac OS X 支持。在未来的MySQL分发版中，我们

想要添加其它平台对这个引擎的支持，包括Windows。

- ARCHIVE存储引擎被用来无索引地，非常小地覆盖存储的大量数据。
- CSV存储引擎把数据以逗号分隔的格式存储在文本文件中。
- BLACKHOLE存储引擎接受但不存储数据，并且检索总是返回一个空集。
- FEDERATED存储引擎把数据存在远程数据库中。在MySQL 5.1中，它只和MySQL一起工作，使用MySQL C Client API。在未来的分发版中，我们想要让它使用其它驱动器或客户端连接方法连接到另外的数据源。

寻求选择一个存储引擎的帮助，请参阅14.4节，“选择一个存储引擎”。

这一章讲述除NDB Cluster外的每一个MySQL存储引擎，NDB Cluster在第17章：*MySQL Cluster*中介绍。

当年创建一个新表的时候，你可以通过添加一个ENGINE 或TYPE 选项到CREATE TABLE语句来告诉MySQL你要创建什么类型的表：

```
CREATE TABLE t (i INT) ENGINE = INNODB;

CREATE TABLE t (i INT) TYPE = MEMORY;
```

虽然TYPE仍然在MySQL 5.1中被支持，现在ENGINE是首选的术语。

如果你省略掉ENGINE或TYPE选项，默认的存储引擎被使用。一般的默认是MyISAM，但你可以用--default-storage-engine或--default-table-type服务器启动选项来改变它，或者通过设置storage_engine或table_type系统变量来改变。

当MySQL被用MySQL配置向导安装在Windows平台上，InnoDB存储引擎替代MyISAM存储引擎作为替代，请参阅2.3.5.1节，“介绍”。

要把一个表从一个类型转到另一个类型，可使用ALTER TABLE语句，这个语句指明新的类型：

```
ALTER TABLE t ENGINE = MYISAM;

ALTER TABLE t TYPE = BDB;
```

请参阅13.1.5节，“CREATE TABLE语法”和13.1.2节，“ALTER TABLE语法”。

如果你试着使用一个未被编译进MySQL的存储引擎，或者试着用一个被编译进MySQL但没有被激活的存储引擎，MySQL取而代之地创建一个MyISAM类型的表。当你在支持不同存储引擎的MySQL服务器之间拷贝表的时候，上述的行为是很方便的。（例如，在一个复制建立中，可能你的主服务器为增加安全而支持 事务存储引擎，但从服务器为更快的速度而仅使用非事务存储引擎。）

在不可用的类型被指定时，自动用MyISAM表来替代，这会对MySQL的新用户造成混淆。无论何时一个表被自动改变之时，产生一个警告。

MySQL总是创建一个.frm文件来保持表和列的定义。表的索引和数据可能被存储在一个或多个文件里，这取决于表的类型。服务器在存储引擎级别之上创建.frm文件。单独的存储引擎创建任何需要用来管理表的额外文件。

一个数据库可以包含不同类型的表。

事务安全表(TST) 比起非事务安全表 (NTST)有几大优势：

- 更安全。即使MySQL崩溃或遇到硬件问题，要么自动恢复，要么从备份加事务日志恢复，你可以取回数据。

- 你可以合并许多语句，并用COMMIT语句同时接受它们全部（如果autocommit被禁止掉）。
- 你可以执行ROLLBACK来忽略你的改变（如果autocommit被禁止掉）。
- 如果更新失败，你的所有改变都变回原来。（用非事务安全表，所有发生的改变都是永久的）。
- 事务安全存储引擎可以给那些当前用读得到许多更新的表提供更好的部署。

虽然MySQL支持数个事务安全存储引擎，为获得最好结果，你不应该在一个事务那混合不同表类型。如果你混合表类型会发生问题，更多信息请参阅13.4.1节，“START TRANSACTION, COMMIT和ROLLBACK Syntax”。

如果你没有指定配置值的话，InnoDB使用默认的配置值。请参阅15.2.3节，“InnoDB配置”。

非事务安全表自身有几个优点，因为没有事务开支，所有优点都能出现：

- 更快
- 需要更少的磁盘空间
- 执行更新需要更少的内存

你可以在同一个语句中合并事务安全和非事务安全表来获得两者最好的情况。尽管如此，在autocommit被禁止掉的事务里，变换到非事务安全表依旧即时提交，并且不会被回滚。

15.1. MyISAM存储引擎

15.1.1. MyISAM启动选项

15.1.2.键需要的空间

15.1.3. MyISAM表存储格式

15.1.4. MyISAM表的问题

MyISAM是 默认存储引擎。它基于更老的ISAM代码，但有很多有用的扩展。（注意MySQL 5.1不支持ISAM）。

每个MyISAM在磁盘上存储成三个文件。第一个文件的名字以表的名字开始，扩展名指出文件类型。.frm文件存储表定义。数据文件的扩展名为.MYD (MYData)。索引文件的扩展名是.MYI (MYIndex)。

要明确表示你想要用一个MyISAM表格，请用ENGINE表选项指出来：

```
CREATE TABLE t (i INT) ENGINE = MYISAM;
```

注释：老版本的MySQL使用TYPE而不是ENGINE（例如，TYPE = MYISAM）。MySQL 5.1为向下兼容而支持这个语法，但TYPE现在被轻视，而ENGINE是首先的用法。

一般地，ENGINE选项是不必要的；除非默认已经被改变了，MyISAM是默认存储引擎。

你可以用**myisamchk**工具来检查或修复MyISAM表。请参阅5.9.5.6节，“使用**myisamchk**做崩溃恢复”。你也可以用**myisampack**来压缩MyISAM表，让它们占更少的空间。请参阅8.2节，“myisampack，产生压缩、只读的MyISAM表”。

如下是MyISAM存储引擎的一些特征：

- 所有数据值先存储低字节。这使得数据机和操作系统分离。二进制轻便性的唯一要求是机器使用补码（如最近20年的机器有的一样）和IEEE浮点格式（在主流机器中也完全是主导的）。唯一不支持二进制兼容性的机器是嵌入式系统。这些系统有时使用特殊的处理器。

先存储数据低字节并不严重地影响速度；数据行中的字节一般是未联合的，从一个方向读未联合的字节并不比从反向读更占用更多的资源。服务器上的获取列值的代码与其它代码相比并不显得时间紧。

- 大文件（达63位文件长度）在支持大文件的文件系统和操作系统上被支持。

- 当把删除和更新及插入混合的时候，动态尺寸的行更少碎片。这要通过合并相邻被删除的块，以及若下一个块被删除，就扩展到下一块来自动完成。

- 每个MyISAM表最大索引数是64。 这可以通过重新编译来改变。每个索引最大的列数是16个。

- 最大的键长度是1000字节。这也可以通过编译来改变。对于键长度超过250字节的情况，一个超过1024字节的的键块被用上。

- BLOB和TEXT列可以被索引。

- NULL值被允许在索引的列中。这个占每个键的0-1个字节。

- 所有数字键值以高字节为先被存储以允许一个更高地索引压缩。

- 当记录以排好序的顺序插入（就像你使用一个AUTO_INCREMENT列之时），索引树被劈开以便高节点仅包含一个键。这改善了索引树的空间利用率。

- 每表一个AUTO_INCREMEN列的内部处理。MyISAM为INSERT和UPDATE操作自动更新这一列。这使得AUTO_INCREMENT列更快（至少10%）。在序列顶的值被删除之后就不能再利用。（当AUTO_INCREMENT列被定义为多列索引的最后一列，可以出现重使用从序列顶部删除的值的情况）。AUTO_INCREMENT值可用ALTER TABLE或**myisamch**来重置。

- 如果数据文件中间的表没有自由块了，在其它线程从表读的同时，你可以INSERT新行到表中。（这被认识为并发操作）。自由块的出现是作为删除行的结果，或者是用比当前内容多的数据对动态长度行更新的结果。当所有自由块被用完（填满），未来的插入又变成并发。

- 你可以把数据文件和索引文件放在不同目录，用DATA DIRECTORY和INDEX DIRECTORY选项CREATE TABLE以获得更高的速度，请参阅13.1.5节，“CREATE TABLE语法”。

- 每个字符列可以又不同的字符集，请参阅第10章：“字符集支持”。

- 在MyISAM索引文件里又一个标志，它表明表是否被正确关闭。如果用--myisam-recover选项启动**mysqld**，MyISAM表在打开得时候被自动检查，如果被表被不恰当地关闭，就修复表。

- 如果你用--update-state选项运行**myisamchk**，它标注表为已检查。**myisamchk --fast**只检查那些没有这个标志的表。

- **myisamchk --analyze**为部分键存储统计信息，也为整个键存储统计信息。

- **myisampack**可以打包BLOB和VARCHAR列。

MyISAM也支持下列特征：

- 支持true VARCHAR类型；VARCHAR列以存储在2个字节中的长度来开始。

- 有VARCHAR的表可以有固定或动态记录长度。

- VARCHAR和CHAR列可以多达64KB。
- 一个被搞乱的已计算索引对可对UNIQUE来使用。这允许你在表内任何列的合并上有UNIQUE。（尽管如此，你不能在一个UNIQUE已计算索引上搜索）。

对MyISAM存储引擎，有一个更详细的论坛在http://forums.mysql.com/list.php?21。

15.1.1. MyISAM启动选项

下列对**mysqld** 的选项可用来改变MyISAM表的行为：

- --myisam-recover=*mode*

设置为崩溃MyISAM表自动恢复的模式。

- --delay-key-write=ALL

对任何MyISAM表的写操作之间不要刷新键缓冲区。

注释：如果你要这么做。当表在使用中之时，你应该不使用来自另一个程序的MyISAM表（比如从另一个MySQL服务器或用**myisamchk**）。这么做会导致索引被破坏。

对使用--delay-key-write的表，使用--external-locking没有帮助。

请参阅5.3.1节，“**mysqld**命令行选项”。

下列系统变量影响MyISAM表的行为：

- bulk_insert_buffer_size

用在块插入优化中的树缓冲区的大小。注释：这是一个*per thread*的限制。

- (*OBSOLETE*) myisam_max_extra_sort_file_size

这个参数已经不在MySQL中使用。

- myisam_max_sort_file_size

如果临时文件会变得超过索引，不要使用快速排序索引方法来创建一个索引。注释：这个参数以字节的形式给出。

- myisam_sort_buffer_size

设置恢复表之时使用的缓冲区的尺寸。

请参阅5.3.3节，“服务器系统变量”。

如果用--myisam-recover选项启动**mysqld**，自动恢复被激活。在这种情况下，当服务器打开一个MyISAM表之时，服务器会检查是否表被标注为崩溃，或者表的打开计数变量是否不为0且你正用--skip-external-locking运行服务器。如果这些条件的任何一个为真，下列情况发生：

- 表被查错。
- 如果服务器发现一个错误，它试着做快速表修复（排序且不重新创建数据文件）。
- 如果修复因为数据文件中的一个错误而失败（例如，一个重复键错误），服务器会再次尝试修复，这一

次重建数据文件。

· 如果修复仍然失败，服务器用旧修复选项方法再重试一次修复（一行接一行地写，不排序）。这个方法应该能修复任何类型的错误，并且需要很低的磁盘空间。

如果恢复不能够从先前完成的语句里恢复所有行，而且你不能在--myisam-recover选项值指定FORCE，自动修复会终止，并在错误日志里写一条错误信息：

```
Error: Couldn't repair table: test.g00pages
```

如果你指定FORCE，取而代之地，类似这样的一个警告被给出：

```
Warning: Found 344 of 354 rows when repairing ./test/g00pages
```

注释：如果自动恢复值包括BACKUP，恢复进程创建文件并用*tbl_name-datetime*.BAK形式取名。你应该有一个**cron**脚本，它自动把这些文件从数据库目录移到备份媒质上。

15.1.2.键所需的空间

MyISAM表使用B型树索引。你可以粗略地计算索引文件的大小为(key_length+4)/0.67, 加上所有的键之和。当所有键以排序的顺序插入并且表没有任何压缩的键之时，以上估计是对最坏的情况的。

字符串索引是被空间压缩的。如果第一个字符串索引部分是字符串，它也被加前缀压缩。如果字符串列有许多拖曳空间，或字符串 列是一个总是不用完全长度的VARCHAR列，空间压缩使得索引文件比最坏情况时的数值要小。前缀压缩被用在以字符串开始的键上。如果有许多具有同一前缀的字符串，前缀压缩是有帮助的。

在MyISAM表，你也可以在创建表的时候通过指定PACK_KEYS=1来前缀压缩数字。当数字被以高字节优先存储之时，若你有许多具有同一前缀的整数 键，上述方法是有帮助的。

15.1.3. MyISAM表的存储格式

15.1.3.1. 静态（固定长度）表特征

15.1.3.2. 动态表特征

15.1.3.3. 已压缩表特征

MyISAM支持三种不同存储格式。其中两个（固定格式和动态格式)根据正使用的列的类型来自动选择。第三个，即已压缩格式，只能使用**myisampack**工具来创建。

当你CREATE或ALTER一个没有BLOB或TEXT列的表，你可以用ROW_FORMAT表选项强制表的格式为FIXED或DYNAMIC。这 会导致CHAR和VARCHAR列因FIXED格式变成CHAR，或因DYNAMIC格式变成VARCHAR。

通过用ALTER TABLE指定ROW_FORMAT={COMPRESSED | DEFAULT}，你可以压缩或解压缩表，请参阅13.1.5节，“CREATE TABLE语法”。

15.1.3.1. 静态（固定长度）表特征

静态格式是MyISAM表的默认存储格式。当表不包含变量长度列（VARCHAR, BLOB, 或TEXT）时，使用这个格式。每一行用固定字节数存储。

MyISAM的三种存储格式中，静态格式就最简单也是最安全的（至少对于崩溃而言）。静态格式也是最快的on-disk格式。快速来自于数据文件中的行在磁盘上被找到的容易方式：当按照索引中的行号查找一个行时，用行长度乘以行号。同样，当扫描一个表的 时候，很容易用每个磁盘读操作读一定数量的记录。

当MySQL服务器正往一个固定格式MyISAM文件写的时候，如果计算机崩溃了，安全是显然的。在这种情况下，**myisamchk**可以容易地决定每行从哪里开始到哪里结束，所以它通常可以收回所有记录，除了写了一部分的记录。注意，基于数据行，MyISAM表索引可以一直被重新构建。

静态格式表的一般特征:

- CHAR列对列宽度是空间填补的。
- 非常快。
- 容易缓存。
- 崩溃后容易重建，因为记录位于固定位置。
- 重新组织是不必要的，除非你删除巨量的记录并且希望为操作系统腾出磁盘空间。为此，可使用OPTIMIZE TABLE或者**myisamchk -r**。
- 通常比动态格式表需要更多的磁盘空间。

15.1.3.2. 动态表特征

如果一个MyISAM表包含任何可变长度 列（VARCHAR, BLOB或TEXTDynamic），或者如果一个表被用ROW_FORMAT=DYNAMIC选项来创建，动态存储格式被使用。

这个格式更为复杂一点，因为每行有一个表明行有多长的头。当一个记录因为更新的结果被变得更长，该记录也可以在超过一个位置处结束。

你可以使用OPTIMIZE TABLE或**myisamchk**来对一个表整理碎片。如果在一个表中有你频繁访问或改变的固定长度 列，表中也有一些可变长度列，仅为避免碎片而把这些可变长度列移到其它表可能是一个好主意。

动态格式表的一般特征:

- 除了长度少于4的列外，所有的字符串列是动态的。
- 在每个记录前面是一个位图，该位图表明哪一列包含空字符串（对于字符串列）或者0（对于数字列）。注意，这并不包括包含NULL值的列。如果一个字符列在拖曳空间移除后长度为零，或者一个数字 列为零值，这都在位图中标注了且列不被保存到磁盘。 非空字符串被存为一个长度字节加字符串的内容。
- 通常比固定长度表需要更少的磁盘空间。
- 每个记录仅使用必需大小的空间。尽管如此，如果一个记录变大，它就按需要被分开成多片，造成记录碎片的后果。比如，你用扩展行长度的信息更新一行，该行就变得有碎片。在这种情况下，你可以时不时运行OPTIMIZE TABLE或**myisamchk -r**来改善性能。可使用**myisamchk -ei**来获取表的统计数据。
- 动态格式表在崩溃后要比静态格式表更难重建，因为一个记录可能被分为多个碎片且链接（碎片）可能被丢失。
- 动态尺寸记录期望的行长度用下列表达式来计算:

```
3
+ (number of columns + 7) / 8
+ (number of char columns)
+ (packed size of numeric columns)
```

- + (*length of strings*)
- + (*number of NULL columns* + 7) / 8

对每个链接需要额外的6字节。在一个更新导致一个记录的扩大之时，一个动态记录被链接了。每个新链接至少是20字节，所以下一个扩大可能在同样的链接里进行。如果不是，则另一个链接将被建立。你可以使用**myisamchk -ed**来找出链接的数目。所有的链接可以用**myisamchk -r**来移除。

15.1.3.3. 已压缩表特征

已压缩存储格式是由**myisampack**工具创建的只读格式。

所有MySQL分发版里都默认包括**myisampack**。已压缩表可以用**myisamchk**来解压缩。

已压缩表有下列特征:

- 已压缩表占据非常小的磁盘空间。这最小化了磁盘用量，当使用缓慢的磁盘（如CD-ROM)之时，这是很有用的。
- 每个记录是被单独压缩的，所以只有非常小的访问开支。依据表中最大的记录，一个记录的头在每个表中占据1到3个字节。每个 列被不同地压缩。通常每个列有一个不同的Huffman树。一些压缩类型如下:

o 后缀空间压缩。

- 前缀空间压缩。
- 零值的数用一个位来存储。
- 如果在一个整型列中的值有一个小的范围，列被用最小可能的类型来存储。比如，一个BIGINT列（8字节），如果所有它的值在-128到127范围内，它可以被存储为TINYINT列（1字节）
- 如果一个 列仅有一小组可能的值，列的类型被转化成ENUM。
- 一个 列可以使用先前压缩类型的任意合并。

- 可以处理固定长度或动态长度记录。

15.1.4. MyISAM表 方面的问题

15.1.4.1. 损坏的MyISAM表

15.1.4.2. 未被适当关闭的表的问题

MySQL用来存储数据的文件格式已经被广泛测试过，但总是有导致数据表变得损坏的环境。

15.1.4.1. 损坏的MyISAM表

即使MyISAM表格式非常可靠（SQL语句对表做的所有改变在语句返回之前被写下），如果下列任何事件发生，你依然可以获得损坏的表:

- **mysqld**进程在写中间被杀掉。
- 发生未预期的计算机关闭（例如，计算机被关闭）。
- 硬件故障。

- 你可以同时在正被服务器修改的表上使用外部程序（如**myisamchk**）。
- MySQL或MyISAM代码的软件缺陷。

一个损坏的表的典型症状如下：

- 当在从表中选择数据之时，你得到如下错误：
- `Incorrect key file for table: '...'. Try to repair it`
- 查询不能在表中找到行或返回不完全的数据。

你可以用CHECK TABLE statement语句来检查MyISAM表的健康，并用REPAIR TABLE修复一个 损坏的MyISAM表。当**mysqld**不运行之时，你也可以用**myisamchk**命令检查或修理一个表。请参阅13.5.2.3节，“CHECK TABLE语法”， 13.5.2.6节，“REPAIR TABLE语法”，和5.9.5节，“myisamchk — MyISAM表维护工具”。

如果你的表变得频繁损坏，你应该试着确定为什么会这样的原因。要明白的最重要的事是表变得损坏是不是因为服务器崩溃的结果。你可以在错误日志中查找最近的restarted mysqld消息来早期验证这个。如果存在这样一个消息，则表损坏是服务器死掉的一个结果是很有可能的。否则，损坏可能在正常操作中发生。这是一个缺陷。你应该试着创建一个展示这个问题的可重复生成的测试案例。请参阅A.4.2节，“如果MySQL保持崩溃，该怎么做” 及E.1.6节，“如果出现表崩溃，请生成测试案例”。

15.1.4.2. 未被适当关闭的表的问题

每个MyISAM索引文件(.MYI)在头有一个计数器，它可以被用来检查一个表是否被恰当地关闭。如果你从CHECK TABLE或**myisamchk**得到下列警告，意味着这个计数器已经不同步了：

```
clients are using or haven't closed the table properly
```

这个警告并不是完全意味着表已被破坏，但你至少应该检查表。

计数器的工作方式如下：

- 表在MySQL中第一次被更新，索引文件头的计数器加一。
- 在未来的更新中，计数器不被改变。
- 当表的最后实例被关闭（因为一个操作FLUSH TABLE或因为在表缓冲区中没有空间）之时，若表已经在任何点被更新，则计数器减一。
- 当你修理或检查表并且发现表完好之时，计数器被重置为零。
- 要避免与其它可能检查表的进程进行事务的问题，若计数器为零，在关闭时计数器不减一。

换句话来说，计数器只有在下列情况会不同步：

- MyISAM表不随第一次发出的LOCK TABLES和FLUSH TABLES被复制。
- MySQL在一次更新和最后关闭之间崩溃（注意，表可能依然完好，因为MySQL总是在每个语句之间为每件事发出写操作）。
- 一个表被**myisamchk --recover**或**myisamchk --update-state**修改，同时被**mysqld**使用。
- 多个**mysqld**服务器正使用表，并且一个服务器在一个表上执行REPAIR TABLE或CHECK TABLE，同时该表也被另一个服务器使用。在这个结构中，使用CHECK TABLE是安全的，虽然你可能从其它服务器上得到警

告。尽管如此，REPAIR TABLE应该被避免，因为当一个服务器用一个新的数据文件替代旧的之时，这并没有发送信号到其它服务器上。

总的来说，在多服务器之间分享一个数据目录是一个坏主意。请参阅5.12节，“在同一个机器上运行多个MySQL服务器” 获得更多地讨论

15.2. InnoDB存储引擎

15.2.1. InnoDB概述

15.2.2. InnoDB联系信息

15.2.3. InnoDB配置

15.2.4. InnoDB启动选项

15.2.5. 创建InnoDB表空间

15.2.6. 创建InnoDB表

15.2.7. 添加和删除InnoDB数据和日志文件

15.2.8. InnoDB数据库的备份和恢复atabase

15.2.9. 把InnoDB数据库移到另一台机器上

15.2.10. InnoDB事务模型和锁定

15.2.11. InnoDB性能调节提示

15.2.12. 多版本的实施

15.2.13. 表和索引结构

15.2.14. 文件空间管理和磁盘I/O

15.2.15. InnoDB错误处理

15.2.16. 对InnoDB表的限制

15.2.17. InnoDB故障诊断和排除

15.2.1. InnoDB概述

InnoDB给MySQL提供 了具有提交，回滚和崩溃恢复能力的事务安全（ACID兼容）存储引擎。InnoDB锁定在行级并且也在SELECT语句提供一个Oracle风格一致的非锁定读。这些特色增加 了多用户部署和性能。没有在InnoDB中扩大锁定的需要，因为在InnoDB中行级锁定适合非常小的空间。InnoDB也支持FOREIGN KEY强制。在SQL查询中，你可以自由地将InnoDB类型的表与其它MySQL的表的类型混合起来，甚至在同一个查询中也可以混合。

InnoDB是为处理巨大数据量时的最大性能设计。它的CPU效率可能是任何其它基于磁盘的关系数据库引擎所不能匹敌的。

InnoDB存储引擎被完全与MySQL服务器整合，InnoDB存储引擎为在主内存中缓存数据和索引而维持它自己的缓冲池。InnoDB存储它的表＆索引在一个表空间中，表空间可以包含数个文件（或原始磁盘分区）。这

与MyISAM表不同，比如在MyISAM表中每个表被存在分离的文件中。InnoDB 表可以是任何尺寸，即使在文件尺寸被限制为2GB的操作系统上。

InnoDB默认地被包含在MySQL二进制分发中。Windows Essentials installer使InnoDB成为Windows上MySQL的默认表。

InnoDB被用来在众多需要高性能的大型数据库站点上产生。著名的Internet新闻站点Slashdot.org运行在InnoDB上。Mytrix, Inc.在InnoDB上存储超过1TB的数据，还有一些其它站点在InnoDB上处理平均每秒800次插入/更新的负荷。

InnoDB在和MySQL一样在同一个GNU GPL证书，第2版（1991年6月版）下发行。更多有关MySQL证书的信息，请参阅http://www.mysql.com/company/legal/licensing/。

关于InnoDB存储引擎，在http://forums.mysql.com/list.php?22有一个详细的论坛。

15.2.2. InnoDB联系信息

InnoDB引擎的厂家的联系信息，Innobase Oy的联系方式如下：

```
Web site: http://www.innodb.com/

Email: <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
```

15.2.3. InnoDB配置

InnoDB存储引擎是默认地被允许的。如果你不想用InnoDB表，你可以添加skip-innodb选项到MySQL选项文件。

被InnoDB存储引擎管理的两个重要的基于磁盘的资源是InnoDB表空间数据文件和它的日志文件。

如果你指定无InnoDB配置选项，MySQL将在MySQL数据目录下创建一个名为ibdata1的10MB大小的自动扩展数据文件，以及两个名为ib_logfile0和ib_logfile1的5MB大小的日志文件。

注释：InnoDB给MySQL提供具有提交， 回滚和崩溃恢复能力的事务安全（ACID兼容）存储引擎。如果拟运行的操作系统和硬件不能如广告说的那样运行，InnoDB就不能实现如上能力。许多操作系统或磁盘子系统可能为改善性能而延迟或记录写操作。在一些操作系统上，就是系统调用（fsync()） 也要等着，直到所有未写入已被刷新文件的数据在被刷新到稳定内存之前可以确实返回了。因为这个，操作系统崩溃或掉电可能损坏当前提交的数据，或者在最坏的 情况，因为写操作已被记录了，甚至破坏了数据库。如果数据完整性对你很重要，你应该在用任何程序于生产中之前做一些“pull-the-plug”测试。Mac OS X 10.3 及以后版本，InnoDB使用一个特别的fcntl()文件 刷新方法。在Linux下，建议禁止回写缓存。

在ATAPI硬盘上，一个类似hdparm -W0 /dev/*hda*命令可能起作用。小心某些驱动器或者磁盘控制器可能不能

禁止回写缓存。

注释：要获得好的性能，你应该如下面例子所讨论那样，明确提供InnoDB参数。自然地，你应该编辑设置来适合你的硬件和要求。

要建立InnoDB表空间文件，在my.cnf选项文件里的[mysqld]节里使用innodb_data_file_path选项。在Windows上，你可以替代地使用my.ini文件。innodb_data_file_path的值应该为一个或多个 数据文件规格的列表。如果你命名一个以上的数据文件，用 分号(';')分隔它们：

```
innodb_data_file_path=datafile_spec1[;datafile_spec2]...
```

例如：把明确创建的具有相同特征的表空间作为默认设置的设置操作如下：

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend
```

这个设置配置一个可扩展大小的尺寸为10MB的单独文件，名为ibdata1。没有给出文件的位置，所以默认的是在MySQL的数据目录内。

尺寸大小用M或者G后缀来指定说明单位是MB或者GB。

一个表空间，它在数据目录里包含一个名为ibdata1的固定尺寸50MB的数据文件和一个名为ibdata2大小为50MB的自动扩展文件，其可以 像这样被配置：

```
[mysqld]
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

一个指定数据文件的完全后缀包括文件名，它的尺寸和数个可选属性：

```
file_name:file_size[:autoextend[:max:max_file_size]]
```

autoextend属性和后面跟着的属性只可被用来对innodb_data_file_path行里最后一个数据文件。

如果你对最后的数据文件指定autoextend选项。如果数据文件耗尽了表空间中的自由空间，InnoDB就扩展数据文件。扩展的幅度是每次8MB。

如果磁盘已满，你可能想要把其它数据添加到另一个硬盘上。重新配置一个已存在表空间的指令见15.2.7节，“添加和删除InnoDB数据和日志文件”。

InnoDB并不感知最大文件尺寸，所以要小心文件系统，在那上面最大的文件尺寸是2GB。要为一个自动扩展数据文件指定最大尺寸，请使用max属性。下列配置允许ibdata1涨到极限的500MB：

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend:max:500M
```

InnoDB默认地在MySQL数据目录创建表空间文件。要明确指定一个位置，请使用innodb_data_home_dir选项。比如，要使用两个名为ibdata1和ibdata2的文件，但是要把他们创建到/ibdata， 像如下一样配置InnoDB：

```
[mysqld]
innodb_data_home_dir = /ibdata
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

注释：InnoDB不创建目录，所以在启动服务器之前请确认/ibdata目录的确存在。这对你配置的任何日志文件目录来说也是真实的。使用Unix或DOS的mkdir命令来创建任何必需的目录。

通过把innodb_data_home_dir的值原原本本地部署到数据文件名，并在需要的地方添加斜杠或反斜杠，InnoDB为每个数据文件形成目录路径。如果innodb_data_home_dir选项根本没有在my.cnf中提到， 默认值是"dot"目录 ./，这意思是MySQL数据目录。

如果你指定innodb_data_home_dir为一个空字符串，你可以为列在innodb_data_file_path值里的数据文件指定绝对路径。下面的例子等价于前面那个例子：

```
[mysqld]

innodb_data_home_dir =

innodb_data_file_path=/ibdata/ibdata1:50M;/ibdata/ibdata2:50M:autoextend
```

一个简单的**my.cnf**例子。假设你有一台配备128MB内存和一个硬盘的计算机。下面的例子显示在my.cnf或my.ini里对InnoDB可能的配置参数，包括autoextend属性。

这个例子适合大多数在Unix和Windows上，不想分配InnoDB数据文件和日志文件到数个磁盘上的用户。它在MySQL数据目录创建一个自动扩展数据文件ibdata1和两个日志文件ib_logfile0及ib_logfile1。同样，InnoD在数据目录里自动创建的小型档案InnoDB日志文件ib_arch_log_0000000000也结束。

```
[mysqld]

# You can write your other MySQL server options here

# ...

# Data files must be able to hold your data and indexes.

# Make sure that you have enough free disk space.

innodb_data_file_path = ibdata1:10M:autoextend

#

# Set buffer pool size to 50-80% of your computer's memory

set-variable = innodb_buffer_pool_size=70M

set-variable = innodb_additional_mem_pool_size=10M

#

# Set the log file size to about 25% of the buffer pool size

set-variable = innodb_log_file_size=20M

set-variable = innodb_log_buffer_size=8M

#

innodb_flush_log_at_trx_commit=1
```

请确认MySQL服务器有适当的权限在数据目录里创建文件。更一般地，服务器必须对任何它需要创建数据文件或日志文件的目录有访问权限。

注意，在一些文件系统上，数据文件必需小于2GB。数据文件的合并尺寸必须至少10MB。

当你第一次创建一个InnoDB表空间，最好从命令行来启动MySQL服务器。InnoDB然后把数据库创建的信息打印到屏幕，所以你可以看见正在发生什么。比如，在Windows上，如果**mysqld-max**位于C:\mysql\bin，你可以如下来启动它：

```
C:\> C:\mysql\bin\mysqld-max --console
```

如果你不发送服务器输出到屏幕上，请检查服务器的错误日志来看在启动过程中InnoDB打印了什么。

请参阅15.2.5节，"创建InnoDB表空间"，以获得InnoDB显示的信息看起来应该 像什么的例子。

Windows上如何指定选项**?** 在Windows上选项文件的规则如下:

- 只应该创建一个my.cnf或文件。
- my.cnf文件应该被放在C盘根目录。
- my.ini文件应该被放置在WINDIR目录; 例如C:\WINDOWS或C:\WINNT。你可以在Windows控制台的命令提示符使用SET命令来打印WINDIR的值:

```
C:\> SET WINDIR
windir=C:\WINNT
```

- 如果你的PC在C盘不是启动盘的地方使用启动装载机, 你唯一的选择是使用my.ini文件。
- 如果你使用安装和配置向导安装的MySQL, my.ini文件被放在MySQL的安装目录。请参阅2.3.5.14节, "my.ini文件的位置"。

Unix上在哪里指定选项? 在Unix上, **mysqld**从下列文件, 如果它们存在的话。以下列的顺序读取选项:

- /etc/my.cnf

全局选项。

- $MYSQL_HOME/my.cnf

服务器专用选项。

- defaults-extra-file

--defaults-extra-file选项指定的文件。

- ~/.my.cnf

用户专用选项。

MYSQL_HOME代表环境变量, 它内含着到包含服务器专用my.cnf文件的目录的路径。

如果你确信**mysqld**只从指定文件读取选项, 你可以在启动服务器之时在命令行使用--defaults-option作为第一个选项:

```
mysqld --defaults-file=your_path_to_my_cnf
```

一个高级的**my.cnf**例子。假设你有一台Linux计算机, 有2GB内存和三个60GB硬盘 (在目录路径/, /dr2和/dr3)。下列例子显示 了在my.cnf里对InnoDB可能的配置参数。

```
[mysqld]
# You can write your other MySQL server options here
# ...
innodb_data_home_dir =
#
# Data files must be able to hold your data and indexes
innodb_data_file_path = /ibdata/ibdata1:2000M;/dr2/ibdata/ibdata2:2000M:autoextend
#
```

```
# Set buffer pool size to 50-80% of your computer's memory,
# but make sure on Linux x86 total memory usage is < 2GB
innodb_buffer_pool_size=1G
innodb_additional_mem_pool_size=20M
innodb_log_group_home_dir = /dr3/iblogs
#
innodb_log_files_in_group = 2
#
# Set the log file size to about 25% of the buffer pool size
innodb_log_file_size=250M
innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
innodb_lock_wait_timeout=50
#
# Uncomment the next lines if you want to use them
#innodb_thread_concurrency=5
```

注意，该例子把两个数据文件放在不同磁盘上。InnoDB开始用第一个数据文件填充表空间。在一些情况下，如果所有数据不被放置在同一物理磁盘上，这样将改善数据库的性能。把日志文件放在与数据文件不同的磁盘上对性能是经常很有好处的。你也可以使用原始磁盘分区（原始设备）作为InnoDB数据文件，这样可以加速I/O。请参阅15.2.14.2节，“为表空间使用原始设备”。

警告：在32位GNU/Linux x86上,你必须要小心不要设置过高的内存用量。glibc可能允许进程堆积在线程堆栈上发展，它会造成你的服务器崩溃。如果下列表达式的值接近或者超过2GB，系统会面临危机：

```
innodb_buffer_pool_size
+ key_buffer_size
+ max_connections*(sort_buffer_size+read_buffer_size+binlog_cache_size)
+ max_connections*2MB
```

每个线程使用一个堆栈（通常是2MB，但在MySQL AB二进制分发版里只有256KB）并且在最坏的情况下也使用sort_buffer_size + read_buffer_size附加内存。

你可以自己编译MySQL，在32位Windows上使用高达64GB物理内存。请参阅15.2.4节，“InnoDB启动选项”里对innodb_buffer_pool_awe_mem_mb的描述。

如何调整其它**mysqld**服务器参数？下列值是典型的，且适用于多数用户：

```
[mysqld]
skip-external-locking
max_connections=200
read_buffer_size=1M
sort_buffer_size=1M
#
# Set key buffer 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=value
```

15.2.4. InnoDB启动选项

这一节叙述InnoDB相关的服务器选项，所有这些选项可以以--*opt_name*=*value*的形式在命令行或在选项文件里被指定。

· innodb_additional_mem_pool_size

InnoDB用来存储数据目录信息＆其它内部数据结构的内存池的大小。你应用程序里的表越多，你需要在这里分配越多的内存。如果InnoDB用光了这个池内的内存，InnoDB开始从操作系统分配内存，并且往MySQL错误日志写警告信息。 默认值是1MB。

· innodb_autoextend_increment

当自动扩展表空间被填满之时，为扩展而增加的尺寸（MB为单位）。 默认值是8。这个选项可以在运行时作为全局系统变量而改变。

· innodb_buffer_pool_awe_mem_mb

如果缓冲池被放在32位Windows的AWE内存里，这个参数就是缓冲池的大小（MB为单位）。(仅在32位Windows上相关）如果你的32位Windows操作系统使用所谓的"地址窗口扩展（AWE)"支持超过4GB内存，你可以用这个参数把InnoDB缓冲池分配进AWE物理内存。这个参数最大的可能值是64000。如果这个参数被指定了，innodb_buffer_pool_size是在32位地址空间的**mysqld**内的窗口，InnoDB把那个AWE内存映射上去。对innodb_buffer_pool_size参数，一个比较好的值是500MB。

· innodb_buffer_pool_size

InnoDB用来缓存它的数据和索引的内存缓冲区的大小。你把这个值设得越高，访问表中数据需要得磁盘I/O越少。在一个专用的数据库服务器上，你可以设置这个参数达机器物理内存大小的80%。尽管如此，还是不要把它设置得太大，因为对物理内存的竞争可能在操作系统上导致内存调度。

· innodb_checksums

InnoDB在所有对磁盘的页面读取上使用校验和验证以确保额外容错防止硬件损坏或数据文件。尽管如此，在一些少见的情况下（比如运行标准检查之时）这个额外的安全特征是不必要的。在这些情况下，这个选项（ 默认是允许的）可以用--skip-innodb-checksums来关闭。

· innodb_data_file_path

到单独数据文件和它们尺寸的路径。通过把innodb_data_home_dir连接到这里指定的每个路径，到每个数据文件的完整目录路径可被获得。文件大小通过给尺寸值尾加M或G以MB或者GB（1024MB）为单位被指定。文件尺寸的和至少是10MB。在一些操作系统上，文件必须小于2GB。如果你没有指定innodb_data_file_path，开始的默认行为是创建一个单独的大小10MB名为ibdata1的自扩展数据文件。在那些支持大文件的操作系统上，你可以设置文件大小超过4GB。你也可以使用原始磁盘分区作为数据文件，请参阅15.2.14.2节，"为表空间使用原始设备"。

· innodb_data_home_dir

目录路径对所有InnoDB数据文件的共同部分。如果你不设置这个值， 默认是MySQL数据目录。你也可以指定这个值为一个空字符串，在这种情况下，你可以在innodb_data_file_path中使用绝对文件路径。

· innodb_doublewrite

默认地，InnoDB存储所有数据两次，第一次存储到doublewrite缓冲，然后存储到确实的数据文件。这个选项可以被用来禁止这个功能。类似于innodb_checksums，这个选项 默认是允许的；因为标准检查或在对顶级性能的需要超过对数据完整性或可能故障的关注之时，这个选项用--skip-innodb-doublewrite来关闭。

· innodb_fast_shutdown

如果你把这个参数设置为0，InnoDB在关闭之前做一个完全净化和一个插入缓冲合并。这些操作要花几分钟时间，设置在极端情况下要几个小时。如果你设置这个参数为1，InnoDB在关闭之时跳过这些操作。 默认值为1。如果你设置这个值为2 (在Netware无此值)， InnoDB将刷新它的日志然后冷关机，仿佛MySQL崩溃一样。已提交的事务不会被丢失，但在下一次启动之时会做一个崩溃恢复。

· innodb_file_io_threads

InnoDB中文件I/O线程的数量。正常地，这个参数是用 默认的，默认值是4，但是大数值对Windows磁盘I/O有益。在Unix上，增加这个数没有效果，InnoDB总是使用默认值。

· innodb_file_per_table

这个选项致使InnoDB用自己的.ibd文件为存储数据和索引创建每一个新表，而不是在共享表空间中创建。请参阅15.2.6.6节，“使用Per-Table表空间”。

· innodb_flush_log_at_trx_commit

当innodb_flush_log_at_trx_commit被 设置为0，日志缓冲每秒一次地被写到日志文件，并且对日志文件做到磁盘操作的刷新，但是在一个事务提交不做任何操作。当这个值为1（默认值）之时，在每个事务提交时，日志缓冲被写到日志文件，对日志文件做到磁盘操作的 刷新。当设置为2之时，在每个提交，日志缓冲被写到文件，但不对日志文件做到磁盘操作的刷新。尽管如此，在对日志文件的刷新在值为2的情况也每秒发生一次。我们必须注意到，因为进程安排问题，每秒一次的 刷新不是100%保证每秒都发生。你可以通过设置这个值不为1来获得较好的性能，但随之你会在一次崩溃中损失二分之一价值的事务。如果你设置这个值为0，那么任何**mysqld**进程的崩溃会删除崩溃前最后一秒的事务，如果你设置这个值为2，那么只有操作系统崩溃或掉电才会删除最后一秒的事务。尽管如此，InnoDB的崩溃恢复不受影响，而且因为这样崩溃恢复开始作用而不考虑这个值。注意，许多操作系统和一些磁盘硬件会欺骗 刷新到磁盘操作。尽管刷新没有进行，你可以告诉**mysqld**刷新已经进行。即使设置这个值为1，事务的持久程度不被保证，且在最坏情况下掉电甚至会破坏InnoDB数据库。在SCSI磁盘控制器中，或在磁盘自身中，使用有后备电池的磁盘缓存会加速文件 刷新并且使得操作更安全。你也可以试着使用Unix命令**hdparm**来在硬件缓存中禁止磁盘写缓存，或使用其它一些对硬件提供商专用的命令。这个选项的 默认值是1。

· innodb_flush_method

这个选项只在Unix系统上有效。如果这个选项被设置为fdatasync （默认值），InnoDB使用fsync()来刷新数据和日志文件。如果被设置为O_DSYNC，InnoDB使用O_SYNC来打开并刷新日志文件，但使用fsync()来 刷新数据文件。如果O_DIRECT被指定了（在一些GNU/Linux版本商可用），InnoDB使用O_DIRECT来打开数据文件，并使用fsync()来刷新数据和日志文件。注意，InnoDB使用fsync()来替代fdatasync()，并且它 默认不使用O_DSYNC，因为这个值在许多Unix变种上已经发生问题。

· innodb_force_recovery

警告：这个选项仅在一个紧急情况下被定义，当时你想要从损坏的数据库转储表。可能的值为从1到6。这些值的意思在15.2.8.1节，“强制恢复”中叙述。作为一个安全措施，当这个选项值大于零之时，InnoDB阻止用户修改数据。

· innodb_lock_wait_timeout

InnoDB事务在被回滚之前可以等待一个锁定的超时秒数。InnoDB在它自己的 锁定表中自动检测事务死锁并且

回滚事务。InnoDB用LOCK TABLES语句注意到锁定设置。默认值是50秒。

为在一个复制建立中最大可能的持久程度和连贯性，你应该在主服务器上的my.cnf文件里使用innodb_flush_log_at_trx_commit=1和sync-binlog=1。

- innodb_locks_unsafe_for_binlog

这个选项在InnoDB搜索和索引扫描中关闭下一键锁定。这个选项的 默认值是假（false）。

正常地，InnoDB使用一个被称为*next-key locking*的算法。当搜索或扫描一个表索引之时，InnoDB以这样一种方式实行行级锁定，它对任何遇到的索引记录设置共享的或独占的锁定。因此，行级锁定实际是索引记录锁定。InnoDB对索引记录设置的锁定也影响被锁定索引记录之前的"gap"。如果一个用户对某一索引内的记录*R*又共享的或独占的锁定，另一个用户不能立即在*R*之前以索引的顺序插入一个新的索引记录。这个选项导致InnoDB不在搜索或索引扫描中使用下一 键锁定。下一键锁定仍然被用来确保外键强制及重复键核查。注意，使用这个选项可能会导致一些诡异的问题：假设你想要用值大于100的标识符从子表里读取并锁定所有的子记录，同时 向随后在选定的行更新一些列：

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

假设在id列有一个索引。查询从id大于100的第一个记录开始扫描索引。如果在索引记录上的锁定不把在间隙处生成的插入排除锁定，同时一个新行被插进表中。如果你在同一个事务之内执行同样的SELECT，你会在查询返回的结果包里看到一个新行。这也意味着，如果新条目被加进数据库，InnoDB不保证连续性；尽管如此，对应连续性仍被保证。因此，如果这个选项被使用，InnoDB在大多数孤立级别保证READ COMMITTED。

这个选项甚至更不安全。InnoDB在一个UPDATE或DELETE中只锁定它更新或删除的行。这大大减少了死锁的可能性，但是可以发生死锁。注意，即使在当类似的操作影响不同行时的情况下，这个选项仍然不允许诸如UPDATE这样的操作压倒相似选项（比如另一个UPDATE）。考虑下列例子：

```
CREATE TABLE A(A INT NOT NULL, B INT);

INSERT INTO A VALUES (1,2),(2,3),(3,2),(4,3),(5,2);

COMMIT;
```

如果一个连接执行一个查询：

```
SET AUTOCOMMIT = 0;

UPDATE A SET B = 5 WHERE B = 3;
```

并且其它连接跟着第一个连接执行其它查询：

```
SET AUTOCOMMIT = 0;

UPDATE A SET B = 4 WHERE B = 2;
```

接着查询2要等查询1的提交或回滚，因为查询1对行（2，3）有一个独占的锁定，并且查询2在扫描行的同时也试着对它不能锁定的同一个行（2，3）采取一个独占的锁定。这是因为当innodb_locks_unsafe_for_binlog选项被使用之时，查询2首先对一个行采取一个独占的锁定，然后确定是否这个行属于结果包，并且如果不属于，就释放不必要的锁定。

因此，查询1按如下执行：

```
x-lock(1,2)

unlock(1,2)

x-lock(2,3)

update(2,3) to (2,5)

x-lock(3,2)
```

```
unlock(3,2)
x-lock(4,3)
update(4,3) to (4,5)
x-lock(5,2)
unlock(5,2)
```

并且查询2按如下执行:

```
x-lock(1,2)
update(1,2) to (1,4)
x-lock(2,3) - 等待查询1提交或回滚
```

- innodb_log_arch_dir

如果我们使用日志档案，被完整写入的日志文件所在的目录也被归档。这个参数值如果被使用了，应该被设置得与innodb_log_group_home_dir一样。尽管如此，它不是必需的。

- innodb_log_archive

这个值当前被设为0。因为MySQL使用它自己的日志文件从备份来恢复，所以当前没有必要来归档InnoDB日志文件。这个选项的 默认值是0。

- innodb_log_buffer_size

InnoDB用来往磁盘上的日志文件写操作的缓冲区的大小。明智的值是从1MB到8MB。 默认的是1MB。一个大的日志缓冲允许大型事务运行而不需要在事务提交之前往磁盘写日志。因此，如果你有大型事务，使日志缓冲区更大以节约磁盘I/O。

- innodb_log_file_size

在日志组里每个日志文件的大小。在32位计算机上日志文件的合并大小必须少于4GB。 默认是5MB。明智的值从1MB到*N*分之一缓冲池大小，其中*N*是组里日志文件的数目。值越大，在缓冲池越少需要检查点刷新行为，以节约磁盘I/O。但更大的日志文件也意味这在崩溃时恢复得更慢。

- innodb_log_files_in_group

在日志组里日志文件的数目。InnoDB以循环方式写进文件。默认是2（推荐）。

- innodb_log_group_home_dir

到InnoDB日志文件的目录路径。它必须有和innodb_log_arch_dir一样的值。如果你不指定任何InnoDB日志参数， 默认的是在MySQL数据目录里创建两个5MB大小名为ib_logfile0和ib_logfile1的文件。

- innodb_max_dirty_pages_pct

这是一个范围从0到100的整数。默认是90。InnoDB中的主线程试着从缓冲池写页面，使得脏页（没有被写的页面）的百分比不超过这个值。如果你有SUPER权限，这个百分比可以在服务器运行时按下面来改变:

```
SET GLOBAL innodb_max_dirty_pages_pct = value;
```

- innodb_max_purge_lag

这个选项控制在净化操作被滞后之时，如何延迟INSERT, UPDATE和DELETE操作。（请参阅15.2.12节，“多版本的实施”）。这个参数的 默认值是零，意为无延迟。这个选项可以在运行时作为全局系统变量而被改变。

InnoDB事务系统维持一个事务列表，该列表有被UPDATE或DELETE操作标志为删除的索引记录。让这个列表的长度为*purge_lag*。当*purge_lag*超过innodb_max_purge_lag之时，每个INSERT, UPDATE和DELETE操作延迟 ((*purge_lag*/innodb_max_purge_lag)*10)-5毫秒。在净化批处理的开始，延迟每隔10秒计算。如果因为一个旧的可以看到行被净化的一致的读查看， 删除操作不被延迟。

对有问题的工作量，典型设置可能是1百万，假设我们的事务很小，只有100字节大小，我们就可以允许在我们的表之中有100MB未净化的行。

· innodb_mirrored_log_groups

我们为数据库保持的日志组内同样拷贝的数量。当前这个值应该被设为1。

· innodb_open_files

在InnoDB中，这个选项仅与你使用多表空间时有关。它指定InnoDB一次可以保持打开的.ibd文件的最大数目。最小值是10。 默认值300。

对.ibd文件的文件描述符是仅对InnoDB的。它们独立于那些由--open-files-limit服务器选项指定的描述符，且不影响表缓存的操作。

· innodb_status_file

这个选项让InnoDB为周期的SHOW INNODB STATUS输出创建一个文件*<datadir>*/innodb_status.*<pid>*。

· innodb_support_xa

当被设置为ON或者1（默认地），这个变量允许InnoDB支持在XA事务中的 双向提交。允许innodb_support_xa导致一个额外的对事务准备的磁盘刷新。如果你对使用XA并不关心，你可以通过设置这个选项为OFF或0来禁止这个变量，以减少磁盘 刷新的次数并获得更好的InnoDB性能。

· innodb_table_locks

InnoDB重视LOCK TABLES，直到所有其它线程已经释放他们所有对表的锁定，MySQL才从LOCK TABLE .. WRITE返回。默认值是1，这意为LOCK TABLES让InnoDB内部锁定一个表。在使用AUTOCOMMIT=1的应用里，InnoDB的内部表锁定会导致死锁。你可以在my.cnf文件（Windows上是my.ini文件）里设置innodb_table_locks=0 来 消除这个问题。

· innodb_thread_concurrency

InnoDB试着在InnoDB内保持操作系统线程的数量少于或等于这个参数给出的限制。如果有性能问题，并且SHOW INNODB STATUS显示许多线程在等待信号，可以让线程“thrashing”，并且设置这个参数更小或更大。如果你的计算机有多个处理器和磁盘，你可以试着这个值更大以更好地利用计算机的资源。一个推荐的值是系统上处理器和磁盘的个数之和。值为500或比500大会禁止 调用并发检查。默认值是20，并且如果设置大于或等于20，并发检查将被禁止。

· innodb_status_file

这个选项让InnoDB为周期的SHOW INNODB STATUS输出创建一个文件*<datadir>*/innodb_status.*<pid>*。

15.2.5. 创建**InnoDB**表空间

15.2.5.1. 处理InnoDB初始化问题

假设你已经安装了MySQL，并且已经编辑了选项文件，使得它包含必要的InnoDB配置参数。在启动MySQL之前，你应该验证你为InnoDB数据文件和日志文件指定的目录是否存在，并且MySQL有访问这些目录的权 InnoDB

限。 不能创建目录，只能创建文件。也检查你有足够的空间来放数据和日志文件。

当创建InnoDB数据库时，最好从命令提示符运行MySQL服务器**mysqld**， 而不要从**mysqld_safe**包装或作为Windows的服务来运行。当你从命令提示符运行，你可看见**mysqld**打印什么以及发生了什么。在Unix上，只需要调用**mysqld**。在Windows上，使用--console选项。

当在选项文件里初始地配置InnoDB后，开始启动MySQL服务器之时，InnoDB创建一个数据文件和日志文件。InnoDB打印如下一些东西:

```
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: Doublewrite buffer not found: creating new

InnoDB: Doublewrite buffer created

InnoDB: Creating foreign key constraint system tables

InnoDB: Foreign key constraint system tables created

InnoDB: Started

mysqld: ready for connections
```

一个新的InnoDB数据库被创建了。你可以用**mysql**这样通常的MySQL客户端程序连接到MySQL服务器。当你用**mysqladmin shutdown**关闭MySQL服务器之时，输出类似如下:

```
010321 18:33:34  mysqld: Normal shutdown

010321 18:33:34  mysqld: Shutdown Complete

InnoDB: Starting shutdown...

InnoDB: Shutdown completed
```

你可以看数据文件和日志文件，并且你可以看见文件被创建。日志目录也包含一个名为ib_arch_log_0000000000的小文件。这个文件是数据库被创建的结果，数据库被创建之后InnoDB切断日志归档。当MySQL再次启动之时，数据文件＆日志文件已经被创建，所以输出更简洁:

```
InnoDB: Started
```

```
mysqld: ready for connections
```

你可以添加innodb_file_per_table选项到my.cnf文件，并且让InnoDB存储每一个表到MySQL数据库目录里自己的.ibd文件。请参阅15.2.6.6节，“使用Per-Table表空间”。

15.2.5.1. 处理InnoDB初始化问题

如果InnoDB在一个文件操作中打印一个操作系统错误，通常问题是如下中的一个：

- 你没有创建一个InnoDB数据文件目录或InnoDB日志目录。
- **mysqld**没有访问这些目录的权限 以创建文件。
- **mysqld**不能恰当地读取my.cnf或my.ini选项文件，因此不能看到你指定的选项。
- 磁盘已满，或者超出磁盘配额。
- 你已经创建一个子目录，它的名字与你指定的数据文件相同。
- 在innodb_data_home_dir或innodb_data_file_path有一个语法错误。

当InnoDB试着初始化它的表空间或日志文件之时，如果出错了，你应该删除InnoDB创建的所有文件。这意味着是所有ibdata文件和所有ib_logfiles文件。万一你创建了一些InnoDB表，为这些表也从MySQL数据库目录删除相应的.frm文件(如果你使用多重表空间的话，也删除任何.ibd文件）。然后你可以试着再次创建InnoDB数据库。最好是从命令提示符启动MySQL服务器 ，以便你可以查看发生了什么。

15.2.6. 创建InnoDB表

15.2.6.1. 如何在InnoDB用不同API来使用事务

15.2.6.2. 转换MyISAM表到InnoDB

15.2.6.3. AUTO_INCREMENT列如何在InnoDB中工作

15.2.6.4. 外键约束

15.2.6.5. InnoDB和MySQL复制

15.2.6.6. 使用Per-Table表空间

假如你用mysql test命令启动MySQL客户端。要创建一个InnoDB表，你必须在表创建SQL语句中指定ENGINE = InnoDB或者TYPE = InnoDB选项：

```
CREATE TABLE customers (a INT, b CHAR (20), INDEX (a)) ENGINE=InnoDB;

CREATE TABLE customers (a INT, b CHAR (20), INDEX (a)) TYPE=InnoDB;
```

SQL语句在表空间的列上创建一个表和索引，表空间包含你在my.cnf指定的数据文件。此外，MySQL在MySQL数据库目录下的test目录里创建一个名为customers.frm的文件。内部地，InnoDB为'test/customers'表往自己的数据目录添加一个条目。这意味这你可以在其它数据库创建一个具有相同名字customers的表，表的名字不会与InnoDB内的冲突。

你可以对任何InnoDB表，通过使用SHOW TABLE STATUS语句，查询在InnoDB表空间内空闲空间的数量。表空间内空闲空间的数量出现在SHOW TABLE STATUS的输出结果内的Comment节里。例如：

```
SHOW TABLE STATUS FROM test LIKE 'customers'
```

注意，统计的SHOW只给出关于InnoDB表的大概情况。它们被用于SQL优化。可是，表和索引保留的大小，以字节为单位是准确的。

15.2.6.1. 如何在InnoDB中用不同的API来使用事务

默认地，每个连接到MySQL服务器的客户端开始之时是允许自动提交模式的，这个模式自动提交你运行的每个SQL语句。要使用多语句事务，你可以用SQL语句SET AUTOCOMMIT = 0禁止自动提交，并且用COMMIT和ROLLBACK来提交或回滚你的事务。 如果你想要autocommit保持打开状态，可以在START TRANSACTION与COMMIT或ROLLBACK之间封装你的事务。下列的例子演示两个事务。第一个是被提交的，第二个是被 回滚的:

```
shell> mysql test

Welcome to the MySQL monitor.  Commands end with ; or \g.

Your MySQL connection id is 5 to server version: 3.23.50-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> CREATE TABLE CUSTOMER (A INT, B CHAR (20), INDEX (A))

    -> ENGINE=InnoDB;

Query OK, 0 rows affected (0.00 sec)

mysql> BEGIN;

Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO CUSTOMER VALUES (10, 'Heikki');

Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;

Query OK, 0 rows affected (0.00 sec)

mysql> SET AUTOCOMMIT=0;

Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO CUSTOMER VALUES (15, 'John');

Query OK, 1 row affected (0.00 sec)

mysql> ROLLBACK;

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM CUSTOMER;

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

| A    | B      |

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

|   10 | Heikki |

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

1 row in set (0.00 sec)

mysql>
```

在类似PHP, Perl DBI/DBD, JDBC, ODBC, 或者MySQL的标准C调用接口这样的API上，你能够以字符串形式发送事务控制语句，如COMMIT，到MySQL服务器，就像其它任何的SQL语句 那样，诸如SELECT或INSERT。一些API也提供单独的专门的事务提交和回滚函数或者方法。

15.2.6.2. 转换MyISAM表到InnoDB

要点：你不应该在mysql数据库（比如，user或者host）里把MySQL系统表转换为InnoDB类型。系统表总是MyISAM型。

如果你想要所有（非系统）表都被创建成InnoDB表，你可以简单地把default-table-type=innodb行添加到my.cnf或my.ini文件的[mysqld]节里。

InnoDB对MyISAM存储引擎采用的单独索引创建方法没有做专门的优化。因此，它不值得导出或导入表以及随后创建索引。改变一个表为InnoDB型最快的办法就是直接插入进一个InnoDB表。即，使用ALTER TABLE ... ENGINE=INNODB，或用相同的定义创建一个空InnoDB表，并且用INSERT INTO ... SELECT * FROM ...插入行。

如果你对第二个键有UNIQUE约束，你可以在导入阶段设置：SET UNIQUE_CHECKS=0，以临时关掉唯一性检查好加速表的导入。对于大表，这节省了大量的磁盘I/O，因为InnoDB随后可以使用它的插入缓冲区来第二个索引记录作为一批来写入。

为获得对插入进程的更好控制，分段插入大表可能比较好：

```
INSERT INTO newtable SELECT * FROM oldtable

   WHERE yourkey > something AND yourkey <= somethingelse;
```

所有记录已经本插入之后，你可以重命名表。

在大表的转换中，你应该增加InnoDB缓冲池的大小来减少磁盘I/O。尽管如此，不要使用超过80%的内部内存。你也可以增加InnoDB日志文件和日志文件的大小。

确信你没有填满表空间：InnoDB表比MyISAM表需要大得多的磁盘空间。如果一个ALTER TABLE耗尽了空间，它就开始一个 回滚，并且如果它是磁盘绑定的，回滚可能要几个小时。对于插入，InnoDB使用插入缓冲区来以成批地合并第二个索引记录到索引中。那样节省了大量磁盘I/O。在回滚中，没有使用这样的机制，而回滚要花比插入长30倍的时间来完成。

在失控的回滚情况下，如果你在数据库中没有有价值的数据，比较明智的是杀掉数据库进程而不是等几百万个磁盘I/O被完成。 完整的过程，请参阅15.2.8.1节，“强制恢复”。

15.2.6.3. AUTO_INCREMENT列在InnoDB里如何工作

如果你为一个表指定AUTO_INCREMENT列，在数据词典里的InnoDB表句柄包含一个名为自动增长计数器的计数器，它被用在为该 列赋新值。自动增长计数器仅被存储在主内存中，而不是存在磁盘上。

InnoDB使用下列算法来为包含一个名为ai_col的AUTO_INCREMENT列的表T初始化自动增长计数器：服务器启动之后，当一个用户对表T做插入之时，InnoDB执行等价如下语句的动作：

```
SELECT MAX(ai_col) FROM T FOR UPDATE;
```

语句取回的值逐次加一，并被赋给列和自动增长计数器。如果表是空的，值1被赋予该列。如果自动增长计数器没有被初始化，而且用户调用为表T显示输出的SHOW TABLE STATUS语句，则计数器被初始化（但不是增加计数）并被存储以供随后的插入使用。注意，在这个初始化中，我们对表做一个正常的独占读锁定，这个锁持续到事务的结束。

InnoDB对为新创建表的初始化自动增长计数器允许同样的过程。

注意，如果用户在INSERT中为AUTO_INCREMENT列指定NULL或者0，InnoDB处理行，就仿佛值还没有被指定，且为它生成一个新值。

自动增长计数器被初始化之后，如果用户插入一个明确指定该列值的行，而且该值大于当前计数器值，则计数器被设置为指定 列值。如果没有明确指定一个值，InnoDB给计数器增加一，并且赋新值给该列。

当访问自动增长计数器之时，InnoDB使用专用的表级的AUTO-INC锁定，该锁持续到当前SQL语句的结束而不是到业务的结束。 引入了专用锁释放策略，来为对一个含AUTO_INCREMENT列的表的插入改善部署。两个事务不能同时对同一表有AUTO-INC锁定。

注意，如果你回滚从计数器获得数的事务，你可能会在赋给AUTO_INCREMENT列的值的序列中发现间隙。

如果用户给列赋一个赋值，或者，如果值大过可被以指定整数格式存储的最大整数，自动增长机制的行为不被定义。

在CREATE TABLE和ALTER TABLE语句中，InnoDB支持AUTO_INCREMENT = *n* 表选项来设置计数器初始值或变更当前计数器值。因在本节早先讨论的原因，这个选项的影响在服务器重启后就无效了。

15.2.6.4.外键约束

InnoDB也支持外键约束。InnoDB中对外键约束定义的语法看起来如下:

```
[CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name, ...)
    [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
    [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

外键定义服从下列情况:

- 所有tables必须是InnoDB型，它们不能是临时表。

- 在引用表中，必须有一个索引，外键列以同样的顺序被列在其中作为第一列。这样一个索引如果不存在，它必须在 引用表里被自动创建。

- 在引用表中，必须有一个索引，被引用的列以同样的顺序被列在其中作为第一列。

- 不支持对外键列的索引前缀。这样的后果之一是BLOB和TEXT列不被包括在一个外键中，这是因为对这些列的索引必须总是包含一个前缀长度。

- 如果CONSTRAINT*symbol*被给出，它在数据库里必须是唯一的。如果它没有被给出，InnoDB自动创建这个名字。

InnoDB拒绝任何试着在子表创建一个外键值而不匹配在父表中的候选键值的INSERT或UPDATE操作。一个父表有一些匹配的行 的子表，InnoDB对任何试图更新或删除该父表中候选键值的UPDATE或DELETE操作有所动作，这个动作取决于用FOREIGN KEY子句的ON UPDATE和ON DETETE子句指定的*referential action*。当用户试图从一个父表删除或更新一行之时，且在子表中有一个或多个匹配的行，InnoDB根据要采取的动作有五个选择:

- CASCADE: 从父表删除或更新且自动删除或更新子表中匹配的行。ON DELETE CASCADE和ON UPDATE CASCADE都可用。在两个表之间，你不应定义若干在父表或子表中的同一列采取动作的ON UPDATE CASCADE子句。

- SET NULL: 从父表删除或更新行，并设置子表中的外键列为NULL。如果外键列没有指定NOT NULL限定词，这就是唯一合法的。ON DELETE SET NULL和ON UPDATE SET NULL子句被支持。

- NO ACTION: 在ANSI SQL-92标准中，NO ACTION意味这不采取动作，就是如果有一个相关的外键值在被参考的表里，删除或更新主要键值的企图不被允许进行（Gruber, 掌握*SQL*, 2000:181）。 InnoDB拒绝对父

表的删除或更新操作。

· RESTRICT: 拒绝对父表的删除或更新操作。NO ACTION和RESTRICT都一样，删除ON DELETE或ON UPDATE子句。（一些数据库系统有延期检查，并且NO ACTION是一个延期检查。在MySQL中，外键约束是被立即检查的，所以NO ACTION和RESTRICT是同样的）。

· SET DEFAULT: 这个动作被解析程序识别，但InnoDB拒绝包含ON DELETE SET DEFAULT或ON UPDATE SET DEFAULT子句的表定义。

当父表中的候选键被更新的时候，InnoDB支持同样选择。选择CASCADE，在子表中的外键 列被设置为父表中候选键的新值。以同样的方式，如果在子表更新的列参考在另一个表中的外键，更新级联。

注意，InnoDB支持外键在一个表内引用，在这些情况下，子表实际上意味这在表内附属的记录。

InnoDB需要对外键和被引用键的索引以便外键检查可以快速进行且不需要一个表扫描。对外键的索引被自动创建。这是相对于一些老版本，在老版本中索引必须明确创建，否则外键约束的创建会失败。

在InnoDB内，外键里和被引用列里相应的列必须有类似的内部数据类型，以便它们不需类型转换就可被比较。整数类型的大小和符号必须相同。字符串类型的长度不需要相同。如果你指定一个SET NULL动作，请确认你没有在子表中宣告该 列为为*NOT NULL*。

如果MySQL从CREATE TABLE语句报告一个错误号1005，并且错误信息字符串指向errno 150，这意思是因为一个外键约束被不正确形成，表创建失败。类似地，如果ALTER TABLE失败，且它指向errno 150， 那意味着对已变更的表，外键定义会被不正确的形成。你可以使用SHOW INNODB STATUS来显示一个对服务器上最近的InnoDB外键错误的详细解释。

注释：InnoDB不对那些 外键或包含NULL列的被引用键值检查外键约束。

对**SQL**标准的背离：如果在父表内有数个行，其中有相同的 被引用键值，然后InnoDB在外键检查中采取动作，就仿佛其它有相同键值的父行不存在一样。例如，如果你已定义一个RESTRICT类型的约束，并且有一个带数个父行的子行，InnoDB不允许任何对这些父行的删除。

居于对应外键约束的索引内的记录，InnoDB通过深度优先选法施行级联操作。

对**SQL**标准的背离： 如果ON UPDATE CASCADE或ON UPDATE SET NULL递归更新相同的表，之前在级联过程中该表一被更新过，它就象RESTRICT一样动作。这意味着你不能使用自引用ON UPDATE CASCADE或者ON UPDATE SET NULL操作。这将阻止级联更新导致的无限循环。另一方面，一个自引用的ON DELETE SET NULL是有可能的，就像一个自引用ON DELETE CASCADE一样。 级联操作不可以被嵌套超过15层深。

对**SQL**标准的背离： 类似一般的MySQL，在一个插入，删除或更新许多行的SQL语句内，InnoDB逐行检查UNIQUE和FOREIGN KEY约束。按照SQL的标准， 默认的行为应被延迟检查，即约束仅在整个SQL语句被处理之后才被检查。直到InnoDB实现延迟的约束检查之前，一些事情是不可能的，比如删除一个通过外键参考到自身的记录。

注释：当前，触发器不被级联外键的动作激活。

一个通过单列外键联系起父表和子表的简单例子如下：

```
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 CASCADE
) TYPE=INNODB;
```

如下是一个更复杂的例子，其中一个product_order表对其它两个表有外键。一个外键引用一个product表中的双列索引。另一个 引用在customer表中的单行索引：

```
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;
```

InnoDB允许你用ALTER TABLE往一个表中添加一个新的 外键约束：

```
ALTER TABLE yourtablename
    ADD [CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name, ...)
    [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
    [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

记住先创建需要的索引。你也可以用ALTER TABLE往一个表添加一个自引用外键约束。

InnoDB也支持使用ALTER TABLE来移除 外键：

```
ALTER TABLE yourtablename DROP FOREIGN KEY fk_symbol;
```

当年创建一个外键之时，如果FOREIGN KEY子句包括一个CONSTRAINT名字，你可以引用那个名字来移除 外键。另外，当外键被创建之时，fk_symbol值被InnoDB内部保证。当你想要移除一个外键之时，要找出标记，请使用SHOW CREATE TABLE语句。例子如下：

```
mysql> SHOW CREATE TABLE ibtest11c\G
*************************** 1. row ***************************
       Table: ibtest11c
Create Table: CREATE TABLE `ibtest11c` (
```

```
  `A` int(11) NOT NULL auto_increment,
  `D` int(11) NOT NULL default '0',
  `B` varchar(200) NOT NULL default '',
  `C` varchar(175) default NULL,
  PRIMARY KEY  (`A`,`D`,`B`),
  KEY `B` (`B`,`C`),
  KEY `C` (`C`),
  CONSTRAINT `0_38775` FOREIGN KEY (`A`, `D`)
REFERENCES `ibtest11a` (`A`, `D`)
ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `0_38776` FOREIGN KEY (`B`, `C`)
REFERENCES `ibtest11a` (`B`, `C`)
ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=INNODB CHARSET=latin1
1 row in set (0.01 sec)

mysql> ALTER TABLE ibtest11c DROP FOREIGN KEY 0_38775;
```

InnoDB解析程序允许你在FOREIGN KEY ... REFERENCES ...子句中用`（backticks)把表和列名名字围起来。InnoDB解析程序也考虑到lower_case_table_names系统变量的设置。

InnoDB返回一个表的外键定义作为SHOW CREATE TABLE语句输出的一部分：

```
SHOW CREATE TABLE tbl_name;
```

从这个版本起，**mysqldump**也将表的正确定义生成到转储文件中，且并不忘记 外键。

你可以如下对一个表显示外键约束：

```
SHOW TABLE STATUS FROM db_name LIKE 'tbl_name';
```

外键约束被列在输出的Comment列。

当执行外键检查之时，InnoDB对它照看着的子或父记录设置共享的行级锁。InnoDB立即检查外键约束，检查不对事务提交延迟。

要使得对有外键关系的表重新载入转储文件变得更容易，**mysqldump**自动在转储输出中包括一个语句设置FOREIGN_KEY_CHECKS为0。这避免在转储被重新装载之时，与不得不被以特别顺序重新装载的表相关的问题。也可以手动设置这个变量：

```
mysql> SET FOREIGN_KEY_CHECKS = 0;
mysql> SOURCE dump_file_name;
mysql> SET FOREIGN_KEY_CHECKS = 1;
```

如果转储文件包含对外键是不正确顺序的表，这就以任何顺序导入该表。这样也加快导入操作。设置FOREIGN_KEY_CHECKS为0，对于在LOAD DATA和ALTER TABLE操作中忽略外键限制也是非常有用的。

InnoDB不允许你删除一个被FOREIGN KEY表约束 引用的表，除非你做设置SET FOREIGN_KEY_CHECKS=0。当你移除一个表的时候，在它的创建语句里定义的约束也被移除。

如果你重新创建一个被移除的表，它必须有一个遵从于也引用它的外键约束的定义。它必须有正确的列名和类型，并且如前所述，它必须对被 引用的键有索引。如果这些不被满足，MySQL返回错误号1005 并在错误信息字符串中指向errno 150。

15.2.6.5. InnoDB和MySQL复制

MySQL复制就像对MyISAM表一样，也对InnoDB表起作用。以某种方式使用复制也是可能的，在这种方式中从服务器上表的类型不同于主服务器上 原始表的类型。例如，你可以复制修改到主服务器上一个InnoDB表，到从服务器上一个MyISAM表里。

要为一个主服务器建立一个新服务器，你不得不复制InnoDB表空间和日志文件，以及InnoDB表的.frm文件，并且移动复件到从服务器。 关于其恰当步骤请参阅15.2.9节，“移动InnoDB数据库到另一台机器”。

如果你可以关闭主服务器或者一个存在的从服务器。你可以采取InnoDB表空间和日志文件的冷备份，并用它来建立一个从服务器。要不关闭任何服务器来建立一个新的从服务器，你也可以使用非免费（商业的）InnoDB热备份工具。

InnoDB复制里一个小的限制是LOAD TABLE FROM MASTER不对InnoDB类型表起作用。有两种可能的工作区：

- 转储主服务器上的表并导入转储文件到从服务器。
- 在用LOAD TABLE *tbl_name* FROM MASTER建立复制之前，在主服务器上使用ALTER TABLE *tbl_name* TYPE=MyISAM，并且随后使用ALTER TABLE把主表转换回InnoDB。

在主服务器失败的事务根本不影响复制。MySQL复制基于二进制日志，MySQ在那里写修改数据的SQL语句。从服务器读主服务器的二进制日志，并执行同样的SQL语句。尽管如此，在事务里发生的语句不被写进二进制日志直到事务提交，在那一刻，事务里所有语句被一次性写进日志。如果语句失败了，比如，因为外键违例，或者，如果事务被回滚，没有SQL语句被写进二进制日志，并且事务根本不在从服务器上执行。

15.2.6.6. 使用Per-Table表空间

你可以存储每个InnoDB表和它的索引在它自己的文件在中，这个特征被称为“多表空间”，因为实际上每个表有它自己的表空间。

对那些想把特定表格移到分离物理磁盘的用户，或者那些希望快速恢复单个表的备份而无须打断其余InnoDB表的使用的用户，使用多表空间会是有益的。

你可以往my.cnf的[mysqld]节添加下面行来允许多表空间：

```
[mysqld]
innodb_file_per_table
```

重启服务器之后，InnoDB存储每个新创建的表到表格所属于的数据库目录下它自己的文件*tbl_name*.ibd里。这类似于MyISAM存储引擎所做的，但MyISAM 把表分成数据文件*tbl_name*.MYD和索引文件*tbl_name*.MYI。对于InnoDB，数据和所以被一起存到.ibd文件。*tbl_name*.frm文件照旧依然被创建。

如果你从my.cnf文件删除innodb_file_per_table行，并重启服务器，InnoDB在共享的表空间文件里再次创建表。

innodb_file_per_table只影响表的创建。如果你用这个选项启动服务器，新表被用.ibd文件来创建，但是你仍旧能访问在共享表空间里的表。如果你删掉这个选项，新表在共享表空间内创建，但你仍旧可以访问任何用多表空间创建的表。

InnoDB总是需要共享标空间。.ibd文件对InnoDB不足以去运行，共享表空间包含熟悉的ibdata文

件，InnoDB把内部数据词典和未作日志放在这个文件中。

注释：你不能 像对MyISAM一样，在数据目录之间随意地移动.ibd文件。这是因为表定义被存在InnoDB共享表空间内，而且InnoDB必须保持事务ID和日志顺序号的一致性。

在一个给定的MySQL安装里，你可以用RENAME TABLE语句把.ibd文件和关联的表从一个数据库移到另一个数据库：

```
RENAME TABLE old_db_name.tbl_name TO new_db_name.tbl_name;
```

如果你有.ibd文件的一个干净的备份，你可以按如下操作从被起源的地方恢复它到MySQL安装中：

1. 发出这个ALTER TABLE语句：

```
2.      ALTER TABLE tbl_name DISCARD TABLESPACE;
```

警告：这个语句删除当前.ibd文件。

3. 把备份的.ibd文件放回到恰当的数据库目录。

4. 发出这个ALTER TABLE语句:

```
5.      ALTER TABLE tbl_name IMPORT TABLESPACE;
```

在上下文中，一个.ibd文件干净的备份意为：

- .ibd文件里没有尚未提交的事务做的修改。
- .ibd文件里无未合并的插入混充条目。
- 净化已经从.ibd文件移除所有已标注删除的索引记录。
- **mysqld**已经把.ibd文件的所有已修改页面从缓冲池 刷新到文件。

你可以用下列方法生成一个.ibd文件的干净备份：

1. 停止所有来自**mysqld**服务器的活动，并提交所有事务。

2. 等待直至SHOW INNODB STATUS显示在数据库被已经没有激活的事务，并且InnoDB主线程的状态是Waiting for server activity。然后你就可以复制.ibd文件了。

生成一个.ibd文件的干净复制的另一个方法是使用商业的**InnoDB**热备份工具：

1. 使用**InnoDB**热备份工具备份InnoDB安装。

2. 在备份上启动第二个**mysqld**服务器，让它清洁备份里的.ibd文件。

15.2.7. 添加和删除**InnoDB**数据和日志文件

这一节描述在InnoDB表空间耗尽空间之时，或者你想要改变日志文件大小之时，你可以做的一些事情。

最简单的，增加InnoDB表空间大小的方法是从开始配置它为自动扩展的。为表空间定义里的最后一个数据文件指定autoextend属性。然后在文件耗尽空间之时，InnoDB以8MB为 增量自动增加该文件的大小。增加的大小可以通过设置innodb_autoextend_increment值来配置,这个值以MB为单位，默认的是8。

作为替代，你可以通过添加另一个数据文件来增加表空间的尺寸。要这么做的话，你必须停止MySQL服务器，编辑my.cnf文件 ，添加一个新数据文件到innodb_data_file_path的末尾，然后再次启动服务器。

如果最后一个数据文件是用关键字autoextend定义的，编辑my.cnf文件的步骤必须考虑最后一个数据文件已经增长到多大。获取数据文件的尺寸，把它四舍五入到最接近乘积1024 × 1024bytes (= 1MB)，然后在innodb_data_file_path中明确指定大致的尺寸。然后你可以添加另一个数据文件。记得只有innodb_data_file_path里最后一个数据可以被指定为自动扩展。

作为一个例子。假设表空间正好有一个自动扩展文件ibdata1：

```
innodb_data_home_dir =

innodb_data_file_path = /ibdata/ibdata1:10M:autoextend
```

假设这个数据文件过一段时间已经长到988MB。下面是添加另一个总扩展数据文件之后的配置行：

```
innodb_data_home_dir =

innodb_data_file_path = /ibdata/ibdata1:988M;/disk2/ibdata2:50M:autoextend
```

当你添加一个新文件到表空间的之后，请确信它并不存在。当你重启服务器之时，InnoDB创建并初始化这个文件。

当前，你不能从表空间删除一个数据文件。要增加表空间的大小，使用如下步骤：

1. 使用**mysqldump**转储所有InnoDB表。
2. 停止服务器。
3. 删除所有已存在的表空间文件。
4. 配置新表空间。
5. 重启服务器。
6. 导入转储文件。

如果你想要改变你的InnoDB日志文件的数量和大小，你必须要停止MySQL服务器，并确信它被无错误地关闭。随后复制旧日志文件到一个安全的地方以防万一某样东西在关闭时出错而你需要用它们来恢复表空间。从日志文件目录删除所有旧日志文件，编辑my.cnf来改变日志文件配置，并再次启动MySQL服务器。**mysqld**在启动之时发现没有日志文件，然后告诉你它正在创建一个新的日志文件。

15.2.8. 备份和恢复**InnoDB**数据库

15.2.8.1. 强制恢复

15.2.8.2. 检查点

安全数据库管理的关键是定期做备份。

InnoDB热备份工具是一个在线备份工具，你可以用它来在InnoDB数据库运行之时备份你的InnoDB数据库。**InnoDB**热备份工具不要求你关闭数据库，并且它不设置任何锁定或干扰你 正常的数据库处理。**InnoDB**热备份工具是非免费（商业的）附加软件，它每年的证书费用是每台MySQL服务器运行的计算机€390。请参阅**InnoDB**热备份主页以获得更详细的信息以及屏幕截图。

如果你可以关闭你的MySQL服务器，你可以生成一个包含InnoDB用来管理它的表的所有文件的二进制备份。使用如下步骤：

1. 关闭MySQL服务器，确信它是无错误关闭。

2. 复制你所有数据文件（ibdata文件和.ibd文件）到一个安全的地方。
3. 复制你所有ib_logfile文件到一个安全的地方。
4. 复制my.cnf配置文件或文件到一个安全的地方。
5. 为你InnoDB表复制.frm文件到一个安全的地方。

复制对InnoDB表起作用，所以你可以使用MySQL复制能力来在需要高可用性的数据库站点保有一份数据库的复制。

除了刚才描述的二进制备份，你也应该周期性地用**mysqldump**转储你的数据库。这么做的原因是，二进制文件可能被破坏而你没有注意到。转储的文件被存储成为人可读的文本文件，所以定点表的损坏 修复变得更容易。再者，因为形式更简单，严重数据损坏的机会就更小。**mysqldump** 也有一个--single-transaction选项，你可以用它来做一个一致的快照而不用把其它客户端排除在外面。

要能够从上述的二进制备份恢复InnoDB数据库到现在，你必须让二进制日志功能打开正在运行的MySQL服务器。 然后你可以应用二进制日志到备份数据库以实现point-in-time恢复：

```
mysqlbinlog yourhostname-bin.123 | mysql
```

要从MySQL服务器的崩溃恢复，唯一需要的是重新启动它。InnoDB自动检查日志并执行到现在的数据库前滚。InnoDB自动回滚在崩溃时 呈现的未提交的事务。在恢复过程中，**mysqld**显示如下一些输出：

```
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
```

如果数据库被损坏或磁盘出错，你必须从备份做恢复。在损坏的情况下，你首先应该找出一个没有被损坏的备份。恢复数据库备份之后，从二进制日志文件恢复。

在一些数据库损坏的情况下，仅仅转储，移除并重建一个或数个被破坏的表是足够的。你可以用CHECK TABLE SQL语句来检查表是否损坏，虽然CHECK TABLE正常地不检查每种可能的损坏，你可以使用innodb_tablespace_monitor来检查表空间文件内文件空间管理的完整性。

在一些情况下，明显地数据库损坏是因为操作系统损坏它自己的文件缓存，磁盘上的数据可能完好，最好是首先重启计算机。它可以消除那些显得是数据库页损坏的错误。

15.2.8.1. 强制恢复

如果数据库页被破坏，你可能想要用SELECT INTO OUTFILE从从数据库转储你的表，通常以这种方法获取的大多数数据是完好的。即使这样，损坏可能导致SELECT * FROM *tbl_name*或者InnoDB后台操作崩溃或断言，或者甚至使得InnoDB前滚恢复崩溃。 尽管如此，你可以用它来强制InnoDB存储引擎启动同时阻止后台操作运行，以便你能转储你的表。例如：你可以在重启服务器之前，在选项文件的[mysqld]节添加如下的行：

```
[mysqld]
innodb_force_recovery = 4
```

innodb_force_recovery被允许的非零值如下。一个更大的数字包含所有更小数字的预防措施。如果你能够用一个多数是4的选项值来转储你的表，那么你是比较安全的，只有一些在损坏的单独页面上的数据会丢失。一个为6的值更夸张，因为数据库页被留在一个陈旧的状态，这个状态反过来可以引发对B树和其它数据库结构的更多破坏。

· 1 (SRV_FORCE_IGNORE_CORRUPT)

即使服务器检测到一个损坏的页，也让服务器运行着；试着让SELECT * FROM *tbl_name* 跳过损坏的索引记录和页，这样有助于转储表。

· 2 (SRV_FORCE_NO_BACKGROUND)

阻止主线程运行，如果崩溃可能在净化操作过程中发生，这将阻止它。

· 3 (SRV_FORCE_NO_TRX_UNDO)

恢复后不运行事务回滚。

· 4 (SRV_FORCE_NO_IBUF_MERGE)

也阻止插入缓冲合并操作。如果你可能会导致一个崩溃。最好不要做这些操作，不要计算表统计表。

· 5 (SRV_FORCE_NO_UNDO_LOG_SCAN)

启动数据库之时不查看未完成日志：InnoDB把未完成的事务视为已提交的。

· 6 (SRV_FORCE_NO_LOG_REDO)

不要在恢复连接中做日志前滚。

数据库不能另外地带着这些选项中被允许的选项来使用。作为一个安全措施，当innodb_force_recovery被设置为大于0的值时，InnoDB阻止用户执行INSERT, UPDATE或DELETE操作.

即使强制恢复被使用，你也可以DROP或CREATE表。如果你知道一个给定的表正在导致回滚崩溃，你可以移除它。你也可以用这个来停止由失败的大宗导入或失败的ALTER TABLE导致的失控 回滚。你可以杀掉**mysqld**进程，然后设置innodb_force_recovery为3，使得数据库被 挂起而不需要回滚，然后舍弃导致失控回滚的表。

15.2.8.2. 检查点

InnoDB实现一种被认识为"模糊"检查点设置的检查点机制。InnoDB以小批量从缓冲池 刷新已修改的数据库页。没必要以单个批次刷新缓冲池，单批次刷新实际操作中可能会在检查点设置进程中停止用户SQL语句的处理。

在崩溃恢复中，InnoDB找寻被写进日志的检查点标签。它知道所有在该标签之前对数据库的修改被呈现在数据库的磁盘映像中。然后InnoDB从检查点往前扫描日志文件，对数据库应用已写入日志的修改。

InnoDB以循环方式写日志文件。所有使得缓冲池里的数据库页与磁盘上的映像不同的已提交修改必须出现在日志文件中 ，以备万一InnoDB需要做一个恢复。这意味着，当InnoDB开始重新使用一个日志文件，它需要确认在磁盘上的数据库页映像包含已写进InnoDB准备重新使用的日志文件里的修改。换句话 说，InnoDB必须创建一个检查点，这经常涉及已修改 数据库页到磁盘的刷新。

前面的叙述解释了为什么使你的日志文件非常大会在设置检查点中节约磁盘I/O。设置日志文件总的大小和缓冲池一样大或者甚至比缓冲池大通常是有意义的。大日志文件的缺点是崩溃恢复要花更长的时间，因为有更多写入日志的信息要应用到数据库上。

15.2.9. 把一个**InnoDB**数据库移到另一台机器

在Windows上， InnoDB 总是在内部以小写名字的方式存储数据库和表。要从Unix把二进制格式的数据库移到Windows，或者从Windows移到Unix，你应该让所有表和数据库的名字小写。要实现这个，一个方便的方式是在创建任何数据库 和表之前，在你的my.cnf或my.ini文件的[mysqld]节内添加如下行：

```
[mysqld]

lower_case_table_names=1
```

类似于MyISAM数据文件，InnoDB数据和日志文件在所有有相同浮点数格式的平台上是二进制兼容的。你可以拷贝所有列在15.2.8节，"InnoDB数据库的备份和恢复"里的相关文件来简单地移动一个InnoDB数据库。如果浮点格式不同，但你没有在表中使用FLOAT或DOUBLE数据类型，则过程是一样：简单地拷贝相关文件。如果格式不容，且你的表包含浮点数据，你必须使用**mysqldump**在一台机器转储你的表，然后在另一台机器导入转储文件。

假设表空间有足够的空间供导入事务产生的大型回滚片断使用，则提高性能的一个方法是在导入数据时关掉autocommit模式。仅在导入整个表或表的一个片断之后提交。

15.2.10. **InnoDB**事务模型和锁定

15.2.10.1. InnoDB锁定模式

15.2.10.2. InnoDB和AUTOCOMMIT

15.2.10.3. InnoDB和TRANSACTION ISOLATION LEVEL

15.2.10.4. 持续非锁定读

15.2.10.5. 锁定读SELECT ... FOR UPDATE和SELECT ... LOCK IN SHARE MODE

15.2.10.6. Next-Key锁定：避免匪夷所思的问题

15.2.10.7. 持续读如何在InnoDB中作用的例子

15.2.10.8. 在InnoDB中用不同的SQL语句设定锁

15.2.10.9. MySQL什么时候提交或回滚一个事务？

15.2.10.10. 死锁检测＆回滚

15.2.10.11. 如何应对死锁

在InnoDB事务模式中，目的是把多版本数据库的最好特性与传统的二相锁定合并起来。InnoDB以Oracle的风格，对行级进行锁定，并且 默认运行查询作为非锁定持续读。在InnoDB中锁定的表被存储得如此节省空间，以至于不需要锁定增大：典型地，数个用户被允许在数据库中锁定每一行，或者行的任何随机子集，而InnoDB不会耗尽内存。

15.2.10.1. InnoDB锁定模式

InnoDB实现标准行级锁定，在这里有两种类型的锁： locks:

- 共享的(*S*)锁允许一个事务去读一行（tuple）。
- 独占的锁(*X*)允许一个事务更新或删除一行。

如果事务*A* 在tuple *t*上持有独占锁定，来自不同事务*B*的对*t*上任一类型的锁的请求不被马上许可，取而代之地，事务*B* 不得不等待事务t释放在tuple *t*上的锁。

如果事务 *A* 在tuple *t*上持有一个共享的锁(*S*)，那么

- 来自不同的事务*B*对在*t* 上*X*的锁定请求不能被马上许可。
- 来自不同的事务*B*对在*t* 上*S*的锁定请求可以被马上获准。因此*A*和*B*持有*t*上的*S*锁定。

不仅如此，InnoDB支持多间隔尺寸锁定，它允许记录锁和对整个表的锁共存。要使得多间隔尺寸级别的锁定实际化，额外类型的锁，被称为*intention locks*被使用。在InnoDB中，意图锁定是表锁定。 对于一个事务，意图锁定之后理想的是指明在该表中对一个行随后需要哪一类型的锁定（共享还是独占）。有两种意图锁被用在InnoDB中（假设事务*T* 在表*R*中要求一个已指出的类型的锁）：

- 意图共享(*IS*)：事务*T* 意图给表*T*上单独的tuple设置*S* 锁定。
- 意图独占(*IX*)：事务*T* 意图给这些tuple设置*X* 锁定。

意图锁协议如下：

- 在假设的事务可以获得对某假定行的*S* 锁定之前，它必须首先获得对包含该行的表的一个*IS* 或者更强的锁定。
- 在假设的事务可以获得对某假定行的*X* 锁定之前，它必须首先获得对包含该行的表的一个*IX* 锁定。

这些结果可以方便地用一个锁类型兼容矩阵来总结：

	X	IX	S	IS
X	冲突	冲突	冲突	冲突
IX	冲突	兼容	冲突	兼容
S	冲突	冲突	兼容	兼容
IS	冲突	兼容	兼容	兼容

如果一个锁定与现在锁定兼容的话，它被授给一个委托事务。如果一个锁定与现存锁定冲突，它就不被授予一个委托事务。事务等待着直到冲突的现存锁定被释放掉。如果一个锁定请求与现存锁定相冲突，且不能被授

予，因为它可能会导致死锁，一个错误产生。

因此，意图锁定不阻碍任何东西，除了完全表请求（比如LOCK TABLES ... WRITE）。*IX* 和*IS*锁定的主要目的是显示某人正锁定一行，或将要在表中锁定一行。

下列的例子演示当锁定请求可能会导致死锁之时一个错误会如何发生。例子中包括两个客户端A和B。

首先客户端A创建一个包含一个行的表，然后开始一个事务。在这个事务内，A通过在共享模式选择行获得对行的*S* 锁定：

```
mysql> CREATE TABLE t (i INT) ENGINE = InnoDB;

Query OK, 0 rows affected (1.07 sec)


mysql> INSERT INTO t (i) VALUES(1);

Query OK, 1 row affected (0.09 sec)


mysql> START TRANSACTION;

Query OK, 0 rows affected (0.00 sec)


mysql> SELECT * FROM t WHERE i = 1 LOCK IN SHARE MODE;

+------+

| i    |

+------+

|    1 |

+------+

1 row in set (0.10 sec)
```

接着，客户端B开始一个事务并尝试从该表删除行：

```
mysql> START TRANSACTION;

Query OK, 0 rows affected (0.00 sec)


mysql> DELETE FROM t WHERE i = 1;
```

删除操作要求一个*X* 锁定。因为这个锁定不兼容客户端A持有的*S*锁定，所以*X* 锁定不被 允许，所以请求进入对行及客户端阻挡的锁定请求队列。

最后，客户端A也试图从表中删除该行：

```
mysql> DELETE FROM t WHERE i = 1;

ERROR 1213 (40001): Deadlock found when trying to get lock;

try restarting transaction
```

因为客户端A需要一个*X* 锁定来删除该行，所以在这里发生死锁。尽管如此，锁定请求不被允许，因为客户端B已经有一个对*X*锁定的请求并且它正等待客户端A释放*S*锁定。因为客户端B之前对*X* 锁定的请求，被客户端A持有的*S*锁定也不能升级到*X*锁定。因此，InnoDB对客户端A产生一个错误，并且释放它的锁定。在那一点上，客户端B的锁定请求可以被许可，并且客户端B从表中删除行。

15.2.10.2. InnoDB和 AUTOCOMMIT

在InnoDB中，所有用户行为都在事务内发生。如果自动提交模式被允许，每个SQL语句在它自己上形成一个单独的事务。MySQL总是带着允许自动提交来开始一个新连接。

如果自动提交模式被用SET AUTOCOMMIT = 0关闭，那么我们可以认为一个用户总是有一个事务打开着。一个SQL COMMIT或ROLLBACK语句结束当前事务并且一个新事务开始。两个语句都释放所有在当前事务中被设置的InnoDB锁定。一个COMMIT语句意味着在当前事务中做的改变被生成为永久的，并且变成其它用户可见的。一个ROLLBACK语句，在另一方面，撤销所有当前事务做的修改。

如果连接有被允许的自动提交，通过用明确的START TRANSACTION或BEGIN语句来开始一个事务，并用COMMIT或者ROLLBACK语句来结束它，这样用户仍旧可以执行一个多重语句事务。

15.2.10.3. InnoDB和TRANSACTION ISOLATION LEVEL

按照SQL:1992 事务隔离级别，InnoDB默认是可重复读的（REPEATABLE READ）。MySQL/InnoDB 提供SQL标准所描述的所有四个事务隔离级别。你可以在命令行用--transaction-isolation选项，或在选项文件里，为所有连接设置 默认隔离级别。例如，你可以在my.inf文件的[mysqld]节里类似如下设置该选项：globally

```
[mysqld]

transaction-isolation = {READ-UNCOMMITTED | READ-COMMITTED

                         | REPEATABLE-READ | SERIALIZABLE}
```

用户可以用SET TRANSACTION语句改变单个会话或者所有新进连接的隔离级别。它的语法如下：

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL

                       {READ UNCOMMITTED | READ COMMITTED

                        | REPEATABLE READ | SERIALIZABLE}
```

注意，对--transaction-isolation选项的级别名中有连字符，但在对SET TRANSACTION语句的级别名中没有。

默认的行为是为下一个（未开始）事务设置隔离级别。如果你使用GLOBAL关键字，语句在全局对从那点开始创建的所有新连接（除了不存在的连接）设置 默认事务级别。你需要SUPER全县来做这个。使用SESSION 关键字集为将来在当前连接上执行的事务设置默认事务级别 。

任何客户端都能自由改变会话隔离级别（甚至在事务的中间），或者为下一个事务设置隔离级别。

你可以用下列语句查询全局和会话事务隔离级别：

```
SELECT @@global.tx_isolation;

SELECT @@tx_isolation;
```

在行级锁定中，InnoDB使用next-key锁定。这意味着除了索引记录，InnoDB也可以把索引记录前的间隙锁定到其它用户所做的紧接该索引记录之前的块插入上。一个next-key锁定指向一个锁定住一个索引记录和它之前的间隙的锁定。一个间隙锁定指仅锁住一些索引记录之前的间隙的锁定。

InnoDB中每个隔离级别的详细描述如下：

- READ UNCOMMITTED

SELECT语句以非锁定方式被执行，但是一个可能更早期版本的记录会被用到。因此，使用这个隔离级别，比如，读是不连贯的。着也被称为“脏读”（dirty read）。另外，这个隔离级别象READ COMMITTED一样作用。

- READ COMMITTED

一个有些象Oracle的隔离级别。所有SELECT ... FOR UPDATE和SELECT ... LOCK IN SHARE MOD语句仅锁定索引记录，而不锁定记录前的间隙，因而允许随意紧挨着已锁定的记录插入新记录。UPDATE和DELETE语句使用一个带唯一搜索条件的唯一的索引仅锁定找到的索引记录，而不包括记录前的间隙。在范围类型UPDATE和DELETE语句，InnoDB必须对范围覆盖的间隙设置next-key锁定或间隙锁定以及其它用户做的块插入。这是很必要的，因为要让MySQL复制和恢复起作用，“幽灵行”必须被阻止掉。

持续读行为如同在Oracle中：即使在同一事务内， 每个持续读设置并读取它自己的新快照。请参阅15.2.10.4节，“持续非锁定读”。

- REPEATABLE READ

这是InnoDB的默认隔离级别。带唯一搜索条件使用唯一索引的SELECT ... FOR UPDATE, SELECT ... LOCK IN SHARE MODE, UPDATE 和DELETE语句只锁定找到的索引记录，而不锁定记录前的间隙。用其它搜索条件，这些操作采用next-key锁定，用next-key锁定或者间隙锁定锁住搜索的索引范围，并且阻止其它用户的新插入。

在持续读中，有一个与之前隔离级别重要的差别：在这个级别，在同一事务内所有持续读读取由第一次读所确定的同一快照。这个惯例意味着如果你在同一事务内发出数个无格式SELECT语句，这些SELECT语句对相互之间也是持续的，请参阅15.2.10.4节，“持续非锁定读”。

- SERIALIZABLE

这个级别类似REPEATABLE READ，但是所有无格式SELECT语句被 隐式转换成SELECT ... LOCK IN SHARE MODE。

15.2.10.4. 持续非锁定读

持续读意味着InnoDB使用它的多版本化来给一个查询展示某个时间点处数据库的快照。查询看到在那个时间点之前被提交的那些确切事务做的更改，并且没有其后的事务或未提交事务做的改变。这个规则的例外是，查询看到发布该查询的事务本身所做的改变。

如果你运行在默认的REPEATABLE READ隔离级别，则在同一事务内的所有持续读读取由该事务中第一个这样的读所确立的快照。你可以通过提交当前事务并在发布新查询的事务之后，为你的查询获得一个更新鲜的快照。

持续读是默认模式，在其中InnoDBzai在READ COMMITTED和REPEATABLE READ隔离级别处理SELECT语句。持续读不在任何它访问的表上设置锁定，因此，其它用户可自由地在持续读在一个表上执行的同一时间修改这些表。

注意，持续读不在DROP TABLE和ALTER TABLE上作用。持续读不在DROP TABLE上作用，因为MySQL不能使用已经被移除的表，并且InnoDB 破坏了该表。持续读不在ALTER TABLE上作用，因为它在某事务内执行，该事务创建一个新表，并且从旧表往新表中插入行。现在，当你重新发出持续读之时，它不能在新表中看见任何行，因为它们被插入到一个在持续读读取的快照中不可见的事务 里。

15.2.10.5. 锁定读SELECT ... FOR UPDATE和SELECT ... LOCK IN SHARE MODE

在一些环境中，一个持续读是不方便的。比如，你可能想要往表的子表里添加一个新行，并确信该子表在父表中有一个根。下列例子显示如何在你应用程序代码中实现参考的完整性。

假设你使用一个持续读去读取父表并且看到表中子表的根。不能安全地往子表添加子行吗？不，因为可能同时发生一些其它用户从父表删除父行，而你没有注意到它的情况。

解决办法是在使用LOCK IN SHARE MODE的锁定模式执行SELECT：

```
SELECT * FROM parent WHERE NAME = 'Jones' LOCK IN SHARE MODE;
```

在共享模式执行一个读意味着我们读最新的可用数据，并在我们读的行设置一个共享锁定。共享模式锁防止其它人更新或删除我们已读的行。同时，如果最新的数据属于其它客户端尚未提交的事务，我们等着知道那个事务被提交。我们看到前述的查询返回父'Jones'，我们可以安全地 往子表添加子记录并提交我们的事务。

让我们看另外一个例子：我们在表child_codes 中有一个整数计数器域，我们用该表给每个添加到子表里的子项指派一个唯一的识别符。显然，使用持续读或者共享模式读去读取当前计数器的值并是一个好主意， 因为数据库的两个用户可能看到计数器的同一个值，如果两个用户试着用同一识别符往该表添加子项，就会发生一个重复键（duplicate-key）错误。

在这里，如果两个用户同时读计数器，当试图更新计数器之时，至少它们中有一个会发生死锁错误并终止，因此LOCK IN SHARE MODE并不是一个好的解决方法。

在这种情况下，有两个好方法去实现读计数器和增长计数器值：(1) 先更新计数器，让计数器值增1，之后读计数器，或者(2)用锁定模式FOR UPDATE先读计数器，之后计数器值增加。后一个途径可被如下实现：

```
SELECT counter_field FROM child_codes FOR UPDATE;

UPDATE child_codes SET counter_field = counter_field + 1;
```

SELECT ... FOR UPDATE读最新的可见数据，在每个它读取的行设置独占锁定。因此，它设置与搜索的SQL UPDATE可能会在行上设置的锁定同样的锁定。

请注意，以上仅是一个SELECT ... FOR UPDATE如何起作用的例子。在MySQL中，事实上生成一个唯一识别符的特殊任务可被用对该表的单独访问来完成：

```
UPDATE child_codes SET counter_field = LAST_INSERT_ID(counter_field + 1);

SELECT LAST_INSERT_ID();
```

SELECT语句仅仅取回识别符信息（专门对当前连接）。它不访问任何表。

15.2.10.6. Next-Key锁定：避免幽灵问题

在行级锁定中，InnoDB 使用一个名为*next-key locking*的算法。InnoDB以这样一种方式执行行级锁定：当它搜索或扫描表的索引之时，它对遇到的索引记录设置共享或独占锁定。因此，行级锁定事实上是索引记录锁定。

InnoDB对索引记录设置的锁定也映像索引记录之前的"间隙"。如果一个用户对一个索引上的记录R有共享或独占的锁定，另一个用户 不能紧接在R之前以索引的顺序插入一个新索引记录。这个间隙的锁定被执行来防止所谓的"幽灵问题"。假设你想要从有一个标识符值大于100的子表读并锁定所有子记录，并想着随后在选定行中更新一些 列：

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

假设在id列有一个索引。查询从id大于100的第一个记录开始扫描。如果设置在索引记录上的锁定不把在间隙生成的插入排除在外，一个新行可能与此同时被插进表中。如果你在同一事务内执行同样的SELECT，你可能会在该查询返回的结果包里看到一个新行。这与事务的隔离原则是相反的：一个事务应该能够运行，以便它已经读的数据在事务过程中不改变。如果我们把一套行视为数据项，新的"幽灵"子记录可能会违反这一隔离原则。

当InnoDB扫描一个索引之时，它也锁定所以记录中最后一个记录之后的间隙。刚在前一个例子中发生：InnoDB设置的锁定防止任何插入到id可能大过100的表。

你可以用next-key锁定在你的应用程序上实现一个唯一性检查：如果你以共享模式读数据， 并且没有看到你将要插入的行的重复，则你可以安全地插入你的行，并且知道在读过程中对你的行的继承者设置的next-key锁定与此同时阻止任何人对你的 行插入一个重复。因此，the next-key锁定允许你锁住在你的表中并不存在的一些东西。

15.2.10.7. 持续读如何在InnoDB中工作的例子

假设你正运行在默认的REPEATABLE READ隔离级别。当你发出一个持续读之时，即一个普通的SELECT语句，InnoDB按照你的查询看到的数据库，给你的事务一个时间点。如果另一个事务在你的时间点被指定之后删除一行并提交，你不会看见已被删除的行。插入和更新被相似地处理。

你可以通过提交你的事务来提前你的时间点，然后做另一个SELECT。

这被称为多版本并发控制（*multi-versioned concurrency control*）。

```
                User A                  User B


            SET AUTOCOMMIT=0;       SET AUTOCOMMIT=0;

time

|           SELECT * FROM t;

|           empty set

|                                 INSERT INTO t VALUES (1, 2);

|

v           SELECT * FROM t;

            empty set

                                  COMMIT;


            SELECT * FROM t;

            empty set


            COMMIT;


            SELECT * FROM t;

            ---------------------

            |    1    |    2    |

            ---------------------

            1 row in set
```

在这个例子中，只有当用户B已经提交插入，并且用户A也已经提交之时，用户A可看见被用户B插入的行，因此时间点 是在用户B提交之前。

如果你想看数据库的最新状态，你应该用READ COMMITTED隔离级别或用一个锁定读：

```
SELECT * FROM t LOCK IN SHARE MODE;
```

15.2.10.8. 在InnoDB中不同SQL语句设置的锁定

在SQL查询处理中，一个锁定读，一个UPDATE或一个DELETE一般地对被扫描的每个索引记录设置记录锁定。如果在某查询中有一个WHERE条件是没什么关系的，而这个查询可能从查询的结果包中排除行。InnoDB不记得确切的WHERE条件，但是仅知道哪个索引范围被扫描。记录锁定是正常的next-key锁定，它也阻止对紧接着记录之前的间隙的插入。

如果锁定被设置为独占，则InnoDB总是取回集束的索引目录并对其设置锁定。

如果你没有适合查询的索引，MySQL不得不扫描整个表来处理查询，表的每行变成被锁定的，这样反过来阻止其它用户的所有对表的插入。创建一个好的索引让你的查询不必要扫描很多行是很重要的。

- SELECT ... FROM是一个持续读，读取数据库的快照并且设置不锁定，除非事务隔离级别被设为SERIALIZABLE。对于 SERIALIZABLE级别，这个设置对它遇到的索引记录设置共享的next-key锁定。

- SELECT ... FROM ... LOCK IN SHARE MODE对读遇到的所有索引记录设置共享的next-key锁定。

- SELECT ... FROM ... FOR UPDATE对读遇到的所有索引记录设置独占的next-key锁定。

- INSERT INTO ... VALUES (...)对被插入的行设置独占锁定。注意，这不是一个next-key锁定，并且不阻止其它用户在已插入行之前的间隙插入。如果发生重复键错误，对重复的索引记录设置共享锁定。

- 在一个表上初始化之前指定的AUTO_INCREMENT列之时，InnoDB在与AUTO_INCREMENT列相关联的索引的末尾设置独占锁定。在访问自动增长计数器中，InnoDB使用专用的表锁定模式AUTO-INC，其中锁定仅持续到当前SQL语句的结束，而不是到整个事务的结束。 请参阅15.2.10.2节，“InnoDB和AUTOCOMMIT”。

InnoDB取回先前初始化的AUTO_INCREMENT列的值而不设定任何锁定。

- INSERT INTO T SELECT ... FROM S WHERE ... 对每个插入到T的行设置独占（非next-key）锁定。它在S上把搜索当作一个持续读，但是如果MySQL二进制日志功能被打开，它就对S设置一个共享的next-key锁定。InnoDB在后一种情况不得不设置锁定：在从一个备份的前滚恢复中，每个SQL语句不得不以与它最初被执行的方式完全同样的方式执行。

- CREATE TABLE ... SELECT ... 把SELECT当作一个持续读来执行，或者带着共享锁定来执行，如前面的条目所述。

- 如果唯一键没有冲突，REPLACE象一个插入一样被做。另外，对必须更新的行设置一个独占的next-key锁定。

- UPDATE ... WHERE ... 对搜索遇到的每个记录设置一个独占的next-key锁定。

- DELETE FROM ... WHERE ... 对搜索遇到的每个记录设置一个独占的next-key锁定。

- 如果对一个表定义FOREIGN KEY约束，任何需要检查约束条件的插入，更新或删除对它看着检查约束的记录设置共享行级锁定。InnoDB在约束失败的情况下也设置这些锁定。

- LOCK TABLES设置表锁定，但是是InnoDB层之上更高的MySQL层设置这些锁定。如果innodb_table_locks=1并且 and AUTOCOMMIT=0，InnoDB意 识到表锁定，并且InnoDB之上的MySQL层知道行级锁定。另外，InooDB的自动死锁检测不能检测在这个表锁定被涉及之处的死锁。 同样，既然更高的MySQL层不知道行级锁定，很可能对另一个用户当前对其有行锁定的表获得一个表锁定。尽管如此，这并不破坏事务的完整性，如15.2.10.10节，“死锁检测和回滚”中讨论的一样。请参阅15.2.16节，“对InnoDB表的限制”。

15.2.10.9. MySQL何时 隐式提交或回滚一个事务？

MySQL以默认允许autocommit模式来开始每一个客户端连接。当autocommit被允许之时，如果SQL语句不返回错误的话，MySQL在每个SQL语句之后，做一个提交。

如果你关闭autocommit模式并且关闭一个连接而不调用你的事务的明确提交，则MySQL回滚你的事务。

如果SQL语句返回一个错误，提交/回滚行为取决于这个错误。请参阅15.2.15节，“InnoDB错误处理”。

下列每一个语句（以及它们的任何同义词）隐式结束一个事务，就好像在执行这个语句之前你做了一个COMMIT：

· ALTER FUNCTION, ALTER PROCEDURE, ALTER TABLE, BEGIN, CREATE DATABASE, CREATE FUNCTION, CREATE INDEX, CREATE PROCEDURE, CREATE TABLE, DROP DATABASE, DROP FUNCTION, DROP INDEX, DROP PROCEDURE, DROP TABLE, LOAD MASTER DATA, LOCK TABLES, RENAME TABLE, SET AUTOCOMMIT=1, START TRANSACTION, TRUNCATE, UNLOCK TABLES.

· UNLOCK TABLES 仅在如果任何表当前被锁定之时，提交一个事务。

· 在InnoDB中的CREATE TABLE语句被作为一个单独的事务来处理。这意味着来自用户的ROLLBACK不撤销用户在事务过程中生成的CREATE TABLE语句。

事务不能被嵌套。当你发出START TRANSACTION语句或与之同义的语句之时，这是对任何当前事务 隐式提交的一个结果。

15.2.10.10. 死锁检测和回滚

InnoDB自动检测事务的死锁，并回滚一个或几个事务来防止死锁。InnoDB试着挑选小事务来回滚，事务的大小通过被插入、更新或删除的行的数量来确定。

如果innodb_table_locks=1 （1是默认值），InnoDB意识到表锁定，其上的MySQL层知道row-level锁定。另外InnoDB不能在MySQL LOCK TABLES设定表锁定的地方或者涉及InnoDB之外的存储引擎设置锁定的地方检测死锁。你必须通过设定innodb_lock_wait_timeout系统变量的值来解决这些情况。

当InnoD执行完全的事务回滚之时，该事务的所有锁定被释放。尽管如此，如果单个SQL语句被因为错误的原因被 回滚，该SQL语句设定的部分锁定可能被保留。这是因为InnoDB以一种方式存储行锁定，在这种方式中它不能知道随后的哪个锁定是被哪个SQL语句设定的。

15.2.10.11. 如何处理死锁

死锁是事务型数据库典型的问题，但是除非它们频繁出现以至于你更本不能运行某个事务，它们一般是不危险的。正常地，你必须编写你的应用程序使得它们总是准备如果因为死锁而 回滚一个事务就重新发出一个事务。

InnoDB使用自动行级锁定。即使在只插入或删除单个行的事务的情况下，你可以遇到死锁。这是因为这些操作不是真正的“极小的”，它们自动对插入或删除的行的（可能是数个）索引记录设置锁定。

你可以用下列技术对付死锁减少它们发生的可能性：

· 用Use SHOW INNODB STATUS来确定最后一个死锁的原因。这样可以帮助你调节应用程序来避免死锁。

· 总是准备着重新发出事务，如果它因为死锁而失败了。死锁不危险，再试一次。

· 经常提交你的事务。小事务更少地倾向于冲突。

· 如果你正使用锁定读，（SELECT ... FOR UPDATE或 ... LOCK IN SHARE MODE），试着用更低的隔离级别，比如READ COMMITTED。

· 以固定的顺序访问你的表和行。则事务形成良好定义的查询并且没有死锁。

· 添加精心选定的索引到你的表。则你的查询需要扫描更少的索引记录并且因此设置更少的锁定。使用EXPLAIN SELECT来确定对于你的查询,MySQL认为哪个索引是最适当的。

· 使用更少的锁定。如果你可以接受允许一个SELECT从一个旧的快照返回数据，不要给它添加FOR

UPDATE或LOCK IN SHARE MODE子句。这里使用READ COMMITTED隔离级别是比较好的，因为每个在同一事务里的持续读从它自己新鲜的快照里读取。

· 如果没有别的有帮助的了，用表级锁定系列化你的事务。用LOCK TABLES对事务型表（如InnoDB)的正确方法是设置AUTOCOMMIT = 0 并且不调用UNLOCK TABLES直到你明确地提交了事务。例如，如果你需要写表t1并从表t读，你可以按如下做：

```
·               SET AUTOCOMMIT=0;
·               LOCK TABLES t1 WRITE, t2 READ, ...;
·               [do something with tables t1 and t2 here];
·               COMMIT;
·               UNLOCK TABLES;
```

表级锁定使得你的事务很好地排队，并且死锁被避免了。

· 领一个系列化事务的方法是创建一个辅助的"semaphore" 表，它只包含一个单行。让每个事务在访问其它表之前更新那个行。以这种方式，所有事务以序列的方式发生。注意，InnoDB即时死锁检测算法也能在这种情况下起租用，因为系列化锁定是行级锁定。超时方法，用MySQL表级锁定，必须被用来解决死锁。

· 在应用程序中使用LOCK TABLES命令，如果AUTOCOMMIT=1,MySQL不设定InnoDB表锁定。

15.2.11. InnoDB性能调节提示

15.2.11.1. SHOW INNODB STATUS和InnoDB Monitors

· 如果Unix顶层工具或者Windows任务管理器显示，你的数据库的工作负荷的CPU使用率小于70%，则你的工作负荷可能是磁盘绑定的，可能你正生成太多的事务和提交，或者缓冲池太小。使得缓冲池更大 一些会有帮助的，但不要设置缓冲池等于或超过物理内存的80%.

· 把数个修改裹进一个事务里。如果事务对数据库修改，InnoDB在该事务提交时必须刷新日志到磁盘。因为磁盘旋转的速度至多167转/秒，如果磁盘没有骗操作系统的话，这就限制提交的数目为同样的每秒167次。

· 如果你可以接受损失一些最近的已提交事务，你可以设置my.cnf文件里的参数innodb_flush_log_at_trx_commit为0。 无论如何InnoDB试着每秒 刷新一次日志，尽管刷新不被许可。

· 使用大的日志文件，让它甚至与缓冲池一样大。当InnoDB写满日志文件时，它不得不在一个检查点把缓冲池已修改的内容写进磁盘。小日志文件导致许多不必要的吸盘写操作。大日志文件的缺点时恢复时间更长。

· 也让日志缓冲相当大（与8MB相似的数量)。

· 如果你存储变长度字符串，或者列可能包含很多NULL值，则使用VARCHAR列类型而不是CHAR类型。一个CHAR(*N*)列总是占据*N* 字节来存储，即使字符串更短或字符串的值是NULL。越小的表越好地适合缓冲池并且减少磁盘I/O。

当使用row_format=compact （MySQL 5.1中 默认的InnoDB记录格式）和可变长度字符集，比如UTF-8或sjis，CHAR(*N*)将占据可变数量的空间，至少为*N* 字节。

· 在一些版本的GNU/Linux和Unix上，用Unix的fsync()（InnoDB默认使用的）把文件刷新到磁盘，并且其他相似的方法是惊人的慢。如果你不满意数据库的写性能，你可以试着设置my.cnf里的innodb_flush_method为O_DSYNC，虽然O_DSYNC在多数系统上看起来更慢。

· 当在Solaris 10上，为x86_64架构（AMD Opteron)使用InnoDB存储引擎，重要的是使用forcedirectio选

项来安装任何为存储与InnoDB相关的文件而使用的数据系统。（默认在Solaris 10/x86_64上不使用这个文件系统安装选项）。使用forcedirectio 失败会导致InnoDB在这个平台上的速度和性能严重退化。

- 当导入数据到InnoDB中之时，请确信MySQL没有允许autocommit模式，因为允许autocommit模式会需要每次插入都要 刷新日志到磁盘。要在导入操作规程中禁止autocommit模式，用SET AUTOCOMMIT和COMMIT语句来包住导入语句：

```
SET AUTOCOMMIT=0;
/* SQL import statements ... */
COMMIT;
```

如果你使用**mysqldump** 选项--opt，即使不用SET AUTOCOMMIT和COMMIT语句来包裹，你也使得快速的转储文件被导入到InnoDB表中。

- 小心大宗插入的大回滚：InnoDB在插入中使用插入缓冲来节约磁盘I/O， 但是在相应的回滚中没有使用这样的机制。一个磁盘绑定的回滚可以用相应插入花费时间的30倍来执行。杀掉数据库进程没有是帮助的，因为回滚在服务器启动时 会再次启动。除掉一个失控的回滚的唯一方法是增大缓冲池使得回滚变成CPU绑定且跑得快，或者使用专用步骤，请参阅15.2.8.1节，“强制恢复”。

- 也要小心其它大的磁盘绑定操作。用DROP TABLE或CREATE TABLE来清空一个表，而不是用DELETE FROM *tbl_name*。

- 如果你需要插入许多行，则使用多行插入语法来减少客户端和服务器之间的通讯开支：

```
INSERT INTO yourtable VALUES (1,2), (5,5), ...;
```

这个提示对到任何表类型的插入都是合法的，不仅仅是对InnoDB类型。

- 如果你在第二个键上有UNIQUE约束，你可以在导入会话中暂时关闭唯一性检查以加速表的导入：

```
SET UNIQUE_CHECKS=0;
```

对于大表，这节约了大量磁盘I/O，因为InnoDB可以使用它的插入缓冲来在一批内写第二个索引记录。

- 如果你对你的表有FOREIGN KEY约束，你可以在导入会话过程中通过关闭外键检查来提速表的导入：

```
SET FOREIGN_KEY_CHECKS=0;
```

对于大表，这可以节约大量的磁盘I/O。

- 如果你经常有对不经常更新的表的重发查询，请使用查询缓存：

```
[mysqld]
query_cache_type = ON
query_cache_size = 10M
```

15.2.11.1. SHOW INNODB STATUS和 InnoDB Monitors

InnoDB包含InnoDB Monitors，它打印有关InnoDB内部状态的信息。你可以使用SQL语句SHOW INNODB STATUS来取标准InnoDB Monitor的输出送到你的SQL客户端。这个信息在性能调节时有用。（如果你正使用**mysql**事务SQL客户端，如果你用\G替代通常的语句终止符分号，输出结果会更可读）。关于InnoDB锁定模式的讨论，请参阅15.2.10.1节，“InnoDB锁定模式”。

```
mysql> SHOW INNODB STATUS\G
```

另一个使用InnoDB Monitors的方法时让它们不断写数据到**mysqld**服务器的标准输出。在这种情况下，没有输

出被送到客户端。当被打开的时候，InnoDB Monitors每15秒打印一次数据。服务器输出通常被定向到MySQL数据目录里的.err日志。这个数据在性能调节时有用。在Windows上，如果你想定向输出到窗口 而不是到日志文件，你必须从控制台窗口的命令行使用--console选项来启动服务器。

监视器输出包括下列类型的信息：

- 被每个激活事务持有的表和记录锁定
- 事务的锁定等待
- 线程的信号等待
- 未解决的文件I/O请求
- 缓冲池统计数据
- 主InnoDB线程的净化和插入缓冲合并活动

要让标准InnoDB Monitor写到**mysqld**的标准输出，请使用下列SQL语句：

```
CREATE TABLE innodb_monitor(a INT) ENGINE=INNODB;
```

监视器可以通过发出下列语句来被停止：

```
DROP TABLE innodb_monitor;
```

CREATE TABLE语法正是通过MySQL的SQL解析程序往InnoDB引擎传递命令的方法：唯一有问题的事情是表名字innodb_monitor及它是一个InnoDB表。对于**InnoDB Monitor**, 表的结构根本不相关。如果你在监视器正运行时关闭服务器，并且你想要再次启动监视器，你必须在可以发出新CREATE TABLE语句启动监视器之前移除表。这个语法在将来的发行版本中可能会改变。

你可以以相似的方式使用innodb_lock_monitor。除了它也提供大量锁定信息之外，它与innodb_monitor相同。一个分离的 innodb_tablespace_monitor打印存在于表空间中已创建文件片断的一个列表，并且确认表空间分配数据结构。此外，有innodb_table_monitor，用它你可以打印InnoDB内部数据词典的内容。

InnoDB Monitor输出的一个例子：

```
mysql> SHOW INNODB STATUS\G
*************************** 1. row ***************************
Status:
=====================================
030709 13:00:59 INNODB MONITOR OUTPUT
=====================================
Per second averages calculated from the last 18 seconds
----------
SEMAPHORES
----------
OS WAIT ARRAY INFO: reservation count 413452, signal count 378357
--Thread 32782 has waited at btr0sea.c line 1477 for 0.00 seconds the semaphore:
X-lock on RW-latch at 41a28668 created in file btr0sea.c line 135
a writer (thread id 32782) has reserved it in mode wait exclusive
```

```
number of readers 1, waiters flag 1
Last time read locked in file btr0sea.c line 731
Last time write locked in file btr0sea.c line 1347
Mutex spin waits 0, rounds 0, OS waits 0
RW-shared spins 108462, OS waits 37964; RW-excl spins 681824, OS waits 375485
------------------------
LATEST FOREIGN KEY ERROR
------------------------
030709 13:00:59 Transaction:
TRANSACTION 0 290328284, ACTIVE 0 sec, process no 3195, OS thread id 34831 inser
ting
15 lock struct(s), heap size 2496, undo log entries 9
MySQL thread id 25, query id 4668733 localhost heikki update
insert into ibtest11a (D, B, C) values (5, 'khDk' ,'khDk')
Foreign key constraint fails for table test/ibtest11a:
,
  CONSTRAINT `0_219242` FOREIGN KEY (`A`, `D`) REFERENCES `ibtest11b` (`A`, `D`)
 ON DELETE CASCADE ON UPDATE CASCADE
Trying to add in child table, in index PRIMARY tuple:
 0: len 4; hex 80000101; asc ....;; 1: len 4; hex 80000005; asc ....;; 2: len 4;
 hex 6b68446b; asc khDk;; 3: len 6; hex 0000114e0edc; asc ...N..;; 4: len 7; hex
 00000000c3e0a7; asc .......;; 5: len 4; hex 6b68446b; asc khDk;;
But in parent table test/ibtest11b, in index PRIMARY,
the closest match we can find is record:
RECORD: info bits 0 0: len 4; hex 8000015b; asc ...[;; 1: len 4; hex 80000005; a
sc ....;; 2: len 3; hex 6b6864; asc khd;; 3: len 6; hex 0000111ef3eb; asc ......
;; 4: len 7; hex 800001001e0084; asc .......;; 5: len 3; hex 6b6864; asc khd;;
------------------------
LATEST DETECTED DEADLOCK
------------------------
030709 12:59:58
*** (1) TRANSACTION:
TRANSACTION 0 290252780, ACTIVE 1 sec, process no 3185, OS thread id 30733 inser
ting
LOCK WAIT 3 lock struct(s), heap size 320, undo log entries 146
MySQL thread id 21, query id 4553379 localhost heikki update
INSERT INTO alex1 VALUES(86, 86, 794,'aA35818','bb','c79166','d4766t','e187358f'
,'g84586','h794',date_format('2001-04-03 12:54:22','%Y-%m-%d %H:%i'),7
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
```

```
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 index symbole
trx id 0 290252780 lock mode S waiting
Record lock, heap no 324 RECORD: info bits 0 0: len 7; hex 61613335383138; asc a
a35818;; 1:
*** (2) TRANSACTION:
TRANSACTION 0 290251546, ACTIVE 2 sec, process no 3190, OS thread id 32782 inser
ting
130 lock struct(s), heap size 11584, undo log entries 437
MySQL thread id 23, query id 4554396 localhost heikki update
REPLACE INTO alex1 VALUES(NULL, 32, NULL,'aa3572','','c3572','d6012t','', NULL,'
h396', NULL, NULL, 7.31,7.31,7.31,200)
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 index symbole
trx id 0 290251546 lock_mode X locks rec but not gap
Record lock, heap no 324 RECORD: info bits 0 0: len 7; hex 61613335383138; asc a
a35818;; 1:
*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 index symbole
trx id 0 290251546 lock_mode X locks gap before rec insert intention waiting
Record lock, heap no 82 RECORD: info bits 0 0: len 7; hex 61613335373230; asc aa
35720;; 1:
*** WE ROLL BACK TRANSACTION (1)
------------
TRANSACTIONS
------------
Trx id counter 0 290328385
Purge done for trx's n:o < 0 290315608 undo n:o < 0 17
Total number of lock structs in row lock hash table 70
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0 0, not started, process no 3491, OS thread id 42002
MySQL thread id 32, query id 4668737 localhost heikki
show innodb status
---TRANSACTION 0 290328384, ACTIVE 0 sec, process no 3205, OS thread id 38929 in
serting
1 lock struct(s), heap size 320
MySQL thread id 29, query id 4668736 localhost heikki update
insert into speedc values (1519229,1, 'hgjhjgghggjgjgjgjgjgjggjgjgjgjgjgjggjgjgjlhh
gghgggggghhjhghgggggghjhghghghghghhhhghghghjhhjghjghjkghjghjghjghjfhjfh
---TRANSACTION 0 290328383, ACTIVE 0 sec, process no 3180, OS thread id 28684 co
```

```
mmitting
1 lock struct(s), heap size 320, undo log entries 1
MySQL thread id 19, query id 4668734 localhost heikki update
insert into speedcm values (1603393,1, 'hgjhjgghggjgjgjgjgjgjggjgjgjgjgjgjgggjgjgjlh
hgghgggggghhjhghgggggghjhghghghghghhhhghghghjhhjghjghjkghjghjghjghjfhjf
---TRANSACTION 0 290328327, ACTIVE 0 sec, process no 3200, OS thread id 36880 st
arting index read
LOCK WAIT 2 lock struct(s), heap size 320
MySQL thread id 27, query id 4668644 localhost heikki Searching rows for update
update ibtest11a set B = 'kHdkkkk' where A = 89572
------- TRX HAS BEEN WAITING 0 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 65556 n bits 232 table test/ibtest11a index PRIM
ARY trx id 0 290328327 lock_mode X waiting
Record lock, heap no 1 RECORD: info bits 0 0: len 9; hex 73757072656d756d00; asc
 supremum.;;
------------------
---TRANSACTION 0 290328284, ACTIVE 0 sec, process no 3195, OS thread id 34831 ro
llback of SQL statement
ROLLING BACK 14 lock struct(s), heap size 2496, undo log entries 9
MySQL thread id 25, query id 4668733 localhost heikki update
insert into ibtest11a (D, B, C) values (5, 'khDk' ,'khDk')
---TRANSACTION 0 290327208, ACTIVE 1 sec, process no 3190, OS thread id 32782
58 lock struct(s), heap size 5504, undo log entries 159
MySQL thread id 23, query id 4668732 localhost heikki update
REPLACE INTO alex1 VALUES(86, 46, 538,'aa95666','bb','c95666','d9486t','e200498f
','g86814','h538',date_format('2001-04-03 12:54:22','%Y-%m-%d %H:%i'),
---TRANSACTION 0 290323325, ACTIVE 3 sec, process no 3185, OS thread id 30733 in
serting
4 lock struct(s), heap size 1024, undo log entries 165
MySQL thread id 21, query id 4668735 localhost heikki update
INSERT INTO alex1 VALUES(NULL, 49, NULL,'aa42837','','c56319','d1719t','', NULL,
'h321', NULL, NULL, 7.31,7.31,7.31,200)
--------
FILE I/O
--------
I/O thread 0 state: waiting for i/o request (insert buffer thread)
I/O thread 1 state: waiting for i/o request (log thread)
I/O thread 2 state: waiting for i/o request (read thread)
I/O thread 3 state: waiting for i/o request (write thread)
```

```
Pending normal aio reads: 0, aio writes: 0,
 ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
Pending flushes (fsync) log: 0; buffer pool: 0
151671 OS file reads, 94747 OS file writes, 8750 OS fsyncs
25.44 reads/s, 18494 avg bytes/read, 17.55 writes/s, 2.33 fsyncs/s
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf for space 0: size 1, free list len 19, seg size 21,
85004 inserts, 85004 merged recs, 26669 merges
Hash table size 207619, used cells 14461, node heap has 16 buffer(s)
1877.67 hash searches/s, 5121.10 non-hash searches/s
---
LOG
---
Log sequence number 18 1212842764
Log flushed up to   18 1212665295
Last checkpoint at  18 1135877290
0 pending log writes, 0 pending chkp writes
4341 log i/o's done, 1.22 log i/o's/second
----------------------
BUFFER POOL AND MEMORY
----------------------
Total memory allocated 84966343; in additional pool allocated 1402624
Buffer pool size   3200
Free buffers       110
Database pages     3074
Modified db pages  2674
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages read 171380, created 51968, written 194688
28.72 reads/s, 20.72 creates/s, 47.55 writes/s
Buffer pool hit rate 999 / 1000
--------------
ROW OPERATIONS
--------------
0 queries inside InnoDB, 0 queries in queue
Main thread process no. 3004, id 7176, state: purging
Number of rows inserted 3738558, updated 127415, deleted 33707, read 755779
```

```
1586.13 inserts/s, 50.89 updates/s, 28.44 deletes/s, 107.88 reads/s
---------------------------
END OF INNODB MONITOR OUTPUT
============================
1 row in set (0.05 sec)
```

关于这个输出一些要注意的:

· 如果TRANSACTIONS节报告锁定等待，你的应用程序可能有锁定竞争。输出也帮助跟踪事务死锁的原因。

· SEMAPHORES节报告等待一个信号的线程和关于线程对一个互斥体或rw-lock信号需要多少次回滚或等待的统计数据。大量等待信号的线程可能是磁盘I/O的结果或者InnoDB内竞争问题的结果。竞争可能是因为查询的严重并行，或者操作系统线程计划的问题。设置innodb_thread_concurrency小于默认值在这种情况下是有帮助的。

· BUFFER POOL AND MEMORY节给你关于页面读写的统计数据。你可以从这些数计算你当前的查询正做多少文件数据I/O操作。

· ROW OPERATIONS节显示主线程正在做什么。

InnoDB发送诊断输出到stderr或文件，而不是到stdout或者固定尺寸内存缓冲，以避免 底层缓冲溢出。作为一个副效果，SHOW INNODB STATUS的输出每15秒钟写到一个状态文件。这个文件的名字是innodb_status.*pid*，其中*pid* 是服务器进程ID。这个文件在MySQL数据目录里创建。正常关机之时，InnoDB删除这个文件。如果发生不正常的关机， 这些状态文件的实例可能被展示，而且必须被手动删除。在移除它们之前，你可能想要检查它们来看它们是否包含有关不正常关机的原因的有用信息。仅在配置选项innodb_status_file=1被设置之时，innodb_status.*pid*文件被创建。

15.2.12. 多版本的实施

因为InnoDB是多版本化的数据库, 它必须保持关于表空间中旧版本行的信息。这个信息被存在名为*rollback segment*（在Oracle中模拟数据结构之后）的数据结构中。

内部地，InnoDB往存在数据库中的每一行中添加两个域。一个6字节的域说明插入或更新该行的最后一个事务的事务识别符。同时，一个删除也被内部处理为一个更新，其中行中一个特殊的位被设置来标注该行为已删除。每一行也包含一个称为滚动指针的7字节域。滚动指针指向一个被写到回滚片断的撤销日志记录。如果该行被更新，撤销日志记录包含在该行被更新之前重建该行的内容必需的的信息。

InnoDB使用在回滚片断中的信息来执行在事务回滚中需要的撤销操作。它也使用这个信息来为一个持续读构建更早版本的行。

在回滚片断中的撤销日志被分为插入和更新撤销日志。插入撤销日志仅在事务回滚中需要，且只要事务一提交就可以被丢弃。更新撤销日志也被用在持续读中，而且它们仅在 当前没有被InnoDB分配给一个快照的事务之后被丢弃，这个快照在持续读中可能会需要更新撤销日志的信息来建立一个数据库行的早期版本。

你必须记得规律地提交你的事务，包括那些只发布持续读的事务。否则, InnoDB不能从更新撤销日志丢弃数据，并且回滚片断可能变得太大，填满你的表空间。

在一个回滚片断里，一个撤销日志记录的物理尺寸典型地小于相应的已插入行或已更新行。你可以用这个信息来计算回滚片断需要的空间。

在InnoDB多版本化方案中，当你用SQL语句删除一行之时，该行没有被从数据库立即物理删除掉。 只有当InnoDB可以丢弃为删除而被写的更新撤销日志记录时，InnoDB也物理地从数据库删除相应行和它的索引记

录。这个删除操作被成为精华，它运行得很快，通常与做删除的SQL语句花的时间在一个数量级。

在某一情景下，在那里，用户以几乎相同的比率，小批次地在表中插入和删除行，净化线程开始滞后 是可能的，并且表变得越来越大，使得每样事都是磁盘绑定的而且非常慢。即使表仅载有10MB有用的数据，它可能变得用所有的死行占据10GB空间。在这种情况下，节流新操作，并分配更多的资源来净化线程可能是比较好的。启动选项和可设置全球变量innodb_max_purge_lag就是为这个目的而存在的。请参阅15.2.4节，“InnoDB启动选项” 以获得更多信息。

15.2.13. 表和索引结构

15.2.13.1. 索引的物理结构

15.2.13.2. 缓冲插入

15.2.13.3. 适应的哈希索引

15.2.13.4. 物理记录结构

MySQL为表把它的数据词典信息以.frm文件的形式存在数据库目录里，这对所有MySQL存储引擎都是真的。但是每个InnoDB表在表空间内的InnoDB内部数据词典里有它自己的条目。当MySQL移除表或数据库，它不得不删除.frm文件和InnoDB数据词典内的相应条目。这就是为什么你不能在数据库之间简单地移动.frm文件来移动InnoDB表。

每个InnoDB表有专门索引，被称为*clustered index*，对行的数据被存于其中。如果你对你的表定义一个PRIMARY KEY， 主键的索引是集束索引。

如果你没有为表定义PRIMARY KEY，MySQL拾取第一个仅有NOT NULL列的UNIQUE索引作为主键，并且InnoDB把它当作集束索引来用。如果表中没有这样一个索引，InnoDB内部产生一个集束索引，其中用InnoDB在这样一个表内指定给行的行ID来排序行。行ID是一个6字节的域，它在新行被插入的时候单一地增加。因此被行ID排序的行是物理地按照插入顺序排的。

通过集束索引访问一个行是较快的，因为行数据是在索引搜索引导的同一页面。如果表是巨大的，当对比于传统解决方案，集束索引构架经常节约磁盘I/O。（在许多数据库，数据传统地被存在与索引记录不同的页）。

在InnoDB中，非集束索引里的记录（也称为第二索引）包含对行的主键值。InnoDB用这个 主键值来从集束索引中搜索行。注意，如果主键是长的，第二索引使用更多空间。

InnoDB比较CHAR和VARCHAR字符串不同长度，以便在较短字符串中剩下的长度被处理视为用空格补上的。

15.2.13.1. 索引的物理结构

所有InnoDB的索引是B数，其中索引记录被存储在树的树叶页。一个索引页的默认大小是16KB。当新记录被插入，InnoDB试着为将来索引记录的插入和更新留下十六分之一的空白页。

如果索引记录以连续的顺序被插入（升序或者降序），结果索引页大约是15/16满。如果记录被以随机的顺序被插入，页面是从1/2到 15/16满。如果索引页的填充因子降到低于1/2，InnoDB试着搜索索引树来释放页。

15.2.13.2. 缓冲插入

在数据库应用中，主键是一个唯一的识别符，并且新行被以主键的升序来插入，这是个常见的情况。因此，到集束索引的插入不需要从一个磁盘随机读。

另一方面，第二索引通常是非唯一的，到第二索引的插入以相对随机次序发生。这可能会导致大量的随机磁盘I/O操作，而没有一个被用在InnoDB中的专用机制。

如果一个索引记录应该被插入到一个非唯一第二索引，InnoDB检查第二索引页是否在缓冲池中。如果是，InnoDB直接插入到索引页。如果索引页没有在缓冲池中被发现，InnoDB插入记录到一个专门的插入缓冲结构。插入缓冲被保持得如此小以至于它完全适合在缓冲池，并且可以非常快地做插入。

插入缓冲周期地被合并到数据库中第二索引树里。把数个插入合并到索引树的同一页，节省磁盘I/O操作，经常地这是有可能的。据测量，插入缓冲可以提高到表的插入速度达15倍。

在插入事务被提交之后，插入缓冲合并可能连续发生。实际上，服务器关闭和重启之后，这会连续发生。（请参阅15.2.8.1节，“强制恢复”）。

当许多第二索引必须被更新之时，并且许多行已被插入之时，插入缓冲合并可能需要数个小时。在这个时间内，磁盘I/O将会增加，这样会导致磁盘绑定查询明显缓慢。另一个明显的后台I/O操作是净化线程（请参阅15.2.12节，“实现多版本化”）。

15.2.13.3. 适应的哈希索引

如果一个表几乎完全配合主内存，在其上执行查询最快的方法就是使用哈希索引。InnoDB有一个自动机制，它监视对为一个表定义的索引的索引搜索。如果InnoDB注意到查询会从建立一个哈希索引中获益，它会自动地这么做。

注意，哈希索引总是基于表上已存在的B树索引来建立。根据InnoDB对B树索引观察的搜索方式，InnoDB会在为该B树定义的任何长度的 键的一个前缀上建立哈希索引。 哈希索引可以是部分的：它不要求整个B树索引被缓存在缓冲池。InnoDB根据需要对被经常访问的索引的那些页面建立哈希索引。

在某种意义上，InnoDB通过针对丰富主内存的适应的哈希索引机制来剪裁自己，更加靠近主内存数据库的架构。

15.2.13.4. 物理记录结构

InnoDB表中的记录有如下特征：

- InnoDB中每个索引记录包含一个6字节的头。这个头被用来将连续的记录连接在一起，并且也用在row-level锁定中。

- 集束索引中的记录包含对所有自定义列的域。此外，有一个6字节的域给事务ID以及一个7字节的域给滚动指针。

- 如果没有为一个表定义初级键，每个集束索引记录也包含一个6字节的行ID域。

- 每个第二索引记录也包含为集束索引键定义的所有域。

- 一个记录也包含一个指向该记录每一个域的指针，如果在一个记录中域的总长度小于128字节，该指针时一个字节；否则就是2字节。这些指针的阵列被称为记录目录。这些指针指向的区域被称为记录的数据部分。

- 内部地，InnoDB以固定长度格式存储固定长度字符列，比如CHAR(10)。InnoDB从VARCHAR列截短跟踪空间。注意，MySQL可以内部地把CHAR列转换为VARCHAR列。请参阅13.1.5.1节，“安静的列规格改变”。

- 一个SQL的NULL值在记录目录里保留1或2字节。此外，SQL的NULL值如果被保存在可变长度列，则在记录数据部分保留零字节。在一个固定长度列，它在记录的数据部分保留该列的固定长度。为NULL值保留固定空间背后的动机是之后该 列从NULL值到非NULL值的更新可以就地完成，且不会导致索引页的碎片。

15.2.14.文件空间管理和磁盘I/O

15.2.14.1. 磁盘I/O

15.2.14.2. 为表空间使用原始设备

15.2.14.3.文件空间管理

15.2.14.4. 整理表碎片

15.2.14.1. 磁盘 I/O

InnoDB使用模拟异步磁盘I/O：InnoDB创建许多线程来处理I/O操作，比如read-ahead.

在InnoDB中有两个read-ahead试探：

· 在连续read-ahead中，如果InnoDB注意到在表空间中对一个片断的访问方式是连续的，它就预先布置一批 数据库页的读 给I/O系统。

· 在随机read-ahead中，如果InnoDB注意到表空间中的一些区域看起来进入完全读进缓冲池中的处理中，它就布置剩余的读到I/O系统。

InnoDB使用一个被称为*doublewrite*的新颖的文件 刷新技术。它给操作体统崩溃或掉电后的恢复添加了安全，并且通过减少对fsync()操作的需要，它在多数Unix变种上改善了性能。.

Doublewrite意为在向一个数据文件写页之前，InnoDB首先把它们写到一个毗邻的表空间区域，称为doublewrite缓冲。仅在写然后 刷新到doublewrite已经完成之前，InnoDB写页面到它们在表空间中恰当的位置。如果操作系统在写页面的中间崩溃，在恢复过程中，InnoDB可以在随后从doublewrite缓冲中找到页面的一个良好复制。

15.2.14.2. 为表空间使用原始设备

你也可以使用原始磁盘分区作为表空间数据文件。通过使用原始磁盘，你可以在Windows和一些Unix系统上执行non-buffered I/O 而无须文件系统开支，这样可以改善性能

当你创建一个新数据文件之时，你必须在innodb_data_file_path里紧接着数据文件尺寸之后放置 关键字newraw。分区必须至少和你指定的尺寸一样大，注意，在InnoDB中，1MB是1024x1024字节， 但是在磁盘规格中，1MB通常意为1,000,000字节。

```
[mysqld]

innodb_data_home_dir=

innodb_data_file_path=/dev/hdd1:3Gnewraw;/dev/hdd2:2Gnewraw
```

下次你启动服务器之时，InnoDB注意到关键字newraw并初始化新分区。但是仍然并不创建或改变任何InnoDB表。另外，当你重启服务器之时，InnoDB重新初始化分区，你的改变会丢失。（从3.23.44启动，作为一个安全措施， 当用newraw指定任何分区之时，InnoDB阻止用户修改数据）。

InnoDB初始化新分区之后，停止服务器，在对行的数据文件规格中改变newraw：

```
[mysqld]

innodb_data_home_dir=

innodb_data_file_path=/dev/hdd1:5Graw;/dev/hdd2:2Graw
```

然后重启动服务器，并且InnoDB允许做改变。

在Windows上，你可以象这样分配磁盘分区为一个数据文件：

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=//./D::10Gnewraw
```

对于访问物理驱动器，//./ 相当于Windows语法的\\.\ 。

当你使用原始磁盘分区之时，确信它们有允许被用来运行MySQL服务器的帐号读和写访问的许可。

15.2.14.3.文件空间管理

你在配置文件中定义的数据文件形成InnoDB的表空间。文件被简单地连起来以形成表空间。没有条纹在使用。当前你不能定义你的表被分配到表空间中的位置。但是，在一个新创建的表中间中，InnoDB 从第一个数据文件开始分配空间。

表空间包含数据库页，默认大小是16KB。这些页被分组成64个连续页的范围。表空间内的文件在InnoDB中被称为片断。术语"rollback segment"有一些混淆，因为它确切地包含许多表空间片断。

在InnoDB中，每个索引分配两个片断。一个是给B树的非树叶节点的，另一个是给树叶节点的。在这里，理想的是为包含数据的树叶节点达到更好的有序性。

当一个片断在表空间内长大，InnoDB单独地分配最先的32页给它。此后InnoDB开始分配整个范围给该片断。InnoDB可以一次给一个大片断添加多达4个范围以确保数据良好的连续性。

在表空间中的一些页包含其它页的位图，因此在一个InnoDB表空间中的一些范围不能被整个地分配给片断，只能作为单个页被分配。

当你发出SHOW TABLE STATUS询问表空间里可用的自由空间之时，InnoDB报告在表空间中完全自由的范围。InnoDB总是为扫除和其它内部目的保留一些范围，这些保留的范围不包括在自由空间里。

当你从一个表中删除数据之时，InnoDB联系相应的B树索引。是否释放单独页或是范围到表空间取决删除的方式，因此被释放的空间变成对其它用户可用，但是记住，已删除的行仅在该行不再被事务 回滚或持续读所需要之后的一个（自动）净化操作中被物理删除。

15.2.14.4. 整理表碎片

如果有随机插入到表的索引或从表的索引随机删除，索引可能变成碎片的。碎片意思是索引页在磁盘上的物理排序并不接近页上记录的索引排序，或者在分配给索引的64页块上有许多没有被使用的页。

碎片的一个"同义词"是一个表占据的空间超过它应该占据的空间的大 小。确切是多少，这是很难去确定的。所有InnoDB数据和索引被存在B树中，并且它们的填充因子可能从50%到100%。碎片的另一个"同 义词"是一个表扫描例如：

```
SELECT COUNT(*) FROM t WHERE a_non_indexed_column <> 12345;
```

花了超过它应该花的时间。（在上面的查询中我们"欺骗"SQL优化器来扫描集束索引，而不是一个第二索引）。多数磁盘可以读10MB/s到50MB/s，这可以被用来评估一个表扫描可以多快地运行。

如果你周期地执行"null" ALTER TABLE操作，它就可以加速索引扫描：

```
ALTER TABLE tbl_name ENGINE=INNODB
```

这导致MySQL重建表。另一个执行碎片整理操作的办法是使用**mysqldump**来转储一个表到一个文本文件，移除表，并重新从转储文件重装载它。

如果到一个索引的插入总是升序的，并且记录仅从末尾被删除，InnoDB文件空间管理保证在索引中的碎片不会发生。

15.2.15. InnoDB错误处理

15.2.15.1. InnoDB错误代码

15.2.15.2. 操作系统错误代码

在InnoDB中错误处理不像SQL标准中指定的一样。按照标准，在SQL语句过程中的任何错误应该导致该语句的回滚。InnoDB有时仅回滚部分语句，或者整个事务。下列条目叙述InnoDB如何执行错误处理：

- 如果你耗尽表空间中的文件空间，你使得MySQL表完全错误，并且InnoDB返回SQL语句。
- 一个事务死锁导致InnoDB回滚整个事务，在锁定等待超时的情况，InnoDB仅回滚最近的SQL语句。

当一个事务回滚因为死锁或锁定等待超时而发生，它在事务中撤销语句的作用。但是如果事务是用START TRANSACTION或BEGIN语句开始的，它就不撤销该语句。进一步，SQL语句变成事务的一部分直到COMMIT, ROLLBACK或者导致暗地提交的SQL语句发生。

- 如果你没有在语句中指定IGNORE选项，重复键错误回滚SQL语句。
- 行太长错误回滚SQL语句。
- 其它错误主要被代码的MySQL层（在InnoDB存储引擎级别以上）探测，它们回滚相应的SQL语句。在单个SQL语句中的回滚中锁定不被释放。

在暗地回滚过程中，就像在一个明确的ROLLBACK SQL命令的执行过程中一样，SHOW PROCESSLIST在State列为有关的连接显示Rolling back。

15.2.15.1. InnoDB错误代码

下面的这个不完全列表是你可能遇见的常见的InnoDB专有错误，带着为什么发生的原因以及如何该解决问题的相关信息

- 1005 (ER_CANT_CREATE_TABLE)

不能创建表。如果错误信息字符串指向errno 150，因为 外键约束被不正确地形成，所以表创建失败。

- 1016 (ER_CANT_OPEN_FILE)

虽然对某表的.frm文件存在，但不能从InnoDB数据文件找到该InnoDB表。请参阅15.2.17.1节，“InnoDB数据词典操作 故障诊断和排除”。

- 1114 (ER_RECORD_FILE_FULL)

InnoDB耗尽表空间中的可用空间，你应该重新配置表空间来添加一个新数据文件。

- 1205 (ER_LOCK_WAIT_TIMEOUT)

锁定等待超时过期。事务被回滚。

- 1213 (ER_LOCK_DEADLOCK)

事务死锁。你应该重运行事务。

- 1216 (ER_NO_REFERENCED_ROW)

你正试着添加一行，但没有父行，并且一个外键约束失败。你应该先添加父行。

- 1217 (ER_ROW_IS_REFERENCED)

你正试图删除一个有子行的父行，并且一个外键约束失败。你应该先删除子行。

15.2.15.2. 操作系统错误代码

要打印一个操作系统错误号的意思，请使用MySQL分发版里的**perror**程序。

下面表提供一些常用Linux系统错误代码。更完整的列表请参阅Linux source code。

- 1 (EPERM)

操作不被允许

- 2 (ENOENT)

无此文件或目录

- 3 (ESRCH)

无此进程

- 4 (EINTR)

中断的系统调用

- 5 (EIO)

I/O 错误

- 6 (ENXIO)

无此设备或地址

- 7 (E2BIG)

Arg列表太长

- 8 (ENOEXEC)

Exec合适错误

- 9 (EBADF)

坏文件号

- 10 (ECHILD)

无子进程

- 11 (EAGAIN)

再试一次

- 12 (ENOMEM)

内存耗尽

- 13 (EACCES)

许可被否定

- 14 (EFAULT)

坏地址

- 15 (ENOTBLK)

阻止需求的设备

- 16 (EBUSY)

设备或资源忙

- 17 (EEXIST)

文件存在

- 18 (EXDEV)

交叉设备连接

- 19 (ENODEV)

无此设备

- 20 (ENOTDIR)

不是一个目录

- 21 (EISDIR)

是一个目录?

- 22 (EINVAL)

非法参量

- 23 (ENFILE)

文件表溢出

- 24 (EMFILE)

打开的文件过多

- 25 (ENOTTY)

设备不适合的ioctl

- 26 (ETXTBSY)

文本文件忙

- 27 (EFBIG)

文件太大

- 28 (ENOSPC)

设备上没空间了

- 29 (ESPIPE)

非法查找

- 30 (EROFS)

只读文件系统

- 31 (EMLINK)

太多连接

下列表提供一列常用Windows系统错误代码。完整列表请访问Microsoft website。

- 1 (ERROR_INVALID_FUNCTION)

不正确函数。

- 2 (ERROR_FILE_NOT_FOUND)

系统不能找到指定的文件。

- 3 (ERROR_PATH_NOT_FOUND)

系统不能找到指定的路径。

- 4 (ERROR_TOO_MANY_OPEN_FILES)

系统不能打开文件。

- 5 (ERROR_ACCESS_DENIED)

访问被拒绝。

- 6 (ERROR_INVALID_HANDLE)

句柄非法。

- 7 (ERROR_ARENA_TRASHED)

存储控制块被破坏。

- 8 (ERROR_NOT_ENOUGH_MEMORY)

无足够存储来处理这个命令。

- 9 (ERROR_INVALID_BLOCK)

存储控制块地址非法。

- 10 (ERROR_BAD_ENVIRONMENT)

环境不正确。

- 11 (ERROR_BAD_FORMAT)

试图用不正确的格式装载一个程序。

- 12 (ERROR_INVALID_ACCESS)

访问代码不合法。

- 13 (ERROR_INVALID_DATA)

数据不合法。

- 14 (ERROR_OUTOFMEMORY)

无足够的存储来完成这个操作。

- 15 (ERROR_INVALID_DRIVE)

系统不能找到指定的驱动器。

- 16 (ERROR_CURRENT_DIRECTORY)

目录不能被删除。

- 17 (ERROR_NOT_SAME_DEVICE)

系统不能移动此文件到一个不同的磁盘驱动器。

- 18 (ERROR_NO_MORE_FILES)

没有更多文件。

- 19 (ERROR_WRITE_PROTECT)

媒质写保护。

- 20 (ERROR_BAD_UNIT)

系统不能找到指定的设备。

- 21 (ERROR_NOT_READY)

设备未准备好。

- 22 (ERROR_BAD_COMMAND)

设备不能识别此命令。

- 23 (ERROR_CRC)

数据错误(循环冗余码校验).

- 24 (ERROR_BAD_LENGTH)

程序发出一个命令，但是命令长度不正确。

- 25 (ERROR_SEEK)

驱动器不能在磁盘上定位指定区域或磁道。

- 26 (ERROR_NOT_DOS_DISK)

指定的磁盘或软盘不能被访问。

- 27 (ERROR_SECTOR_NOT_FOUND)

驱动器不能找到请求的扇区。

- 28 (ERROR_OUT_OF_PAPER)

打印机缺纸。

- 29 (ERROR_WRITE_FAULT)

系统不能写指定设备。

- 30 (ERROR_READ_FAULT)

系统不能从指定设备读。

- 31 (ERROR_GEN_FAILURE)

附加到系统的设备不起作用。

- 32 (ERROR_SHARING_VIOLATION)

进程不能访问文件，因为它正被另一个进程使用。

- 33 (ERROR_LOCK_VIOLATION)

进程不能访问文件，因为另一个进程已经锁定文件的一部分。

- 34 (ERROR_WRONG_DISK)

驱动器的的磁盘不正确，请插入 %2 (卷系列号: %3) 到驱动器 %1.

- 36 (ERROR_SHARING_BUFFER_EXCEEDED)

太多文件被打开以共享。

- 38 (ERROR_HANDLE_EOF)

到达文件的末尾。

- 39 (ERROR_HANDLE_DISK_FULL)

磁盘已满。

- 87 (ERROR_INVALID_PARAMETER)

参数不正确。（如果你在Windows中得到这个错误，并且已经在my.cnf或my.ini文件中设置innodb_file_per_table，则添加innodb_flush_method=unbuffered到你的my.cnf或my.ini文件）。

· 112 (ERROR_DISK_FULL)

磁盘已满。

· 123 (ERROR_INVALID_NAME)

文件名，目录名或者卷标语法不正确。

· 1450 (ERROR_NO_SYSTEM_RESOURCES)

存在系统资源不够完成请求的服务。

15.2.16. 对InnoDB表的限制

· 一个表不能包含超过1000列。

· 内部最大键长度是3500字节，但MySQL自己限制这个到1024字节。

· 除了VARCHAR, BLOB和TEXT列，最大行长度稍微小于数据库页的一半。即，最大行长度大约8000字节。LONGBLOB和LONGTEXT列必须小于4GB, 总的行长度，页包括BLOB和TEXT列，必须小于4GB。InnoDB在行中存储VARCHAR，BLOB或TEXT列的前768字节，余下的存储的分散的页面中。

· 虽然InnoDB内部地支持行尺寸大于65535，你不能定义一个包含VARCHAR列的，合并尺寸大于65535的行。

```
mysql> CREATE TABLE t (a VARCHAR(8000), b VARCHAR(10000),
    -> c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
    -> f VARCHAR(10000), g VARCHAR(10000));
ERROR 1118 (42000): Row size too large. The maximum row size for the
used table type, not counting BLOBs, is 65535. You have to change some
columns to TEXT or BLOBs
```

· 在一些更老的操作系统上，数据文件必须小于2GB。

· InnoDB日志文件的合并尺寸必须小于4GB。

· 最小的表空间尺寸是10MB。最大的表空间尺寸是4,000,000,000个数据库页（64TB）。这也是一个表的最大尺寸。

· InnoDB表不支持FULLTEXT索引。

· ANALYZE TABLE 通过对每个索引树做八次随机深入并相应地更新索引集估值，这样来计数集。注意，因为这是仅有的估值，反复运行ANALYZE TABLE会产生不同数。这使得 ANALYZE TABLE 在 InnoDB 表上很快，不是百分百准确，因为它没有考虑所有的行。

MySQL 不仅在汇合优化中使用索引集估值。如果一些汇合没有以正确的方式优化，你可以试一下 ANALYZE TABLE 。很少有情况，ANALYZE TABLE 没有产生对你特定的表足够好的值，你可以使用 FORCE INDEX 在你查询中来强制使用特定索引，或者设置 max_seeks_for_key 来确保MySQL在表扫描之上运行索引查找。请参阅5.3.3节，“服务器系统变量”。请参阅A.6节，“优化器相关的问题”。

· 在Windows上，InnoDB总是内部地用小写字母存储数据库和表名字。要把数据库以二进制形式从Unix移到Windows，或者从Windows移到Unix，你应该让所有数据库和表的名字都是小写。

· 警告**:** 不要在MySQL数据库内的把MySQL系统表从MyISAM转为InnoDB表！这是一个不被支持的操作。如果你这么做了，MySQL直到你从备份恢复旧系统表，或用**mysql_install_db**脚本重建系统表才重启动。

· InnoDB在表内不保留行的内部计数。（因为多版本化，这可能确实有些复杂 ）。要处理一个SELECT COUNT(*) FROM t语句，InnoDB必须扫描表的一个索引，如果这个索引不在缓冲池中，扫描需要花一些时间。要获得快速计数，你不得不使用一个自己创建的计数器表，并让你的应用按照它做的插入和删除来更新它。如果你的表格不经常改变，使用MySQL查询缓存时一个好的解决方案。如果大致的行数就足够了，则SHOW TABLE STATUS也可被使用。请参阅15.2.11节，“InnoDB性能调节提示”。

· 对于AUTO_INCREMENT列，你必须总是为表定义一个索引，并且索引必须包含AUTO_INCREMENT列。在MyISAM表中，AUTO_INCREMENT列可能时多 列索引的一部分。

· 当你重启MySQL服务器之时，InnoDB可能为一个AUTO_INCREMENT列重使用一个旧值（即，一个被赋给一个老的已 回滚的事务的值）。

· 当一个AUTO_INCREMENT列用完值，InnoDB限制一个BIGINT到－9223372036854775808以及BIGINT UNSIGNED到1。尽管如此，BIGINT值有由64位，所以注意到，如果你要一秒输入100万个行，在BIGINT到达它上限之前，可能还需要将近30万年。用所有其它整数类型 列，产生一个重复键错误。这类似于MyISAM如何工作的，因为它主要是一般MySQL行为，并不特别关于任何存储引擎。

· DELETE FROM *tbl_name*不重新生成表，但取而代之地删除所有行，一个接一个地删除。

· TRUNCATE *tbl_name*为InnoDB而被映射到DELETE FROM *tbl_name* 并且不重置AUTO_INCREMENT计数器。

· SHOW TABLE STATUS不能给出关于InnoDB表准确的统计数据，除了被表保留的物理尺寸。行计数仅是在SQL优化中粗略的估计。

· 在MySQL 5.1中，如果innodb_table_locks=1(1是默认值） MySQL LOCK TABLES操作在每一个表上获取两个锁定。除了在MySQL层的表锁定，它也获得一个InnoDB表锁定。旧版的MySQL不获取InnoDB表锁定，旧行为可以通过设置innodb_table_locks=0 来选择。如果没有InnoDB表锁定被获得，即使表的一些记录被其它事务锁定，LOCK TABLES完成。

· 所有被一个事务持有的InnoDB锁定在该事务被提交或中止之时被释放。因此在AUTOCOMMIT=1模式，在InnoDB表上调用是没有太多意义的，因为被需求的InnoDB表锁定可能会被立即释放。

· 有时，在事务的过程中锁定更多的表可能是有用的。不幸地，MySQL中的LOCK TABLES执行一个暗地的COMMIT和UNLOCK TABLES。LOCK TABLES的一个InnoDB变量已经被计划， 该计划在事务的中间被执行。

· 为建立复制从服务器的LOAD TABLE FROM MASTER语句对InnoDB表不起作用。一个工作区在主服务器上更换表为MyISAM的，然后做负载，之后更换主服务器表回到InnoDB中。

· 在InnoDB中默认数据库页的大小是16KB。通过编译代码，你可以在8KB到64KB之间来设置这个值。你不得不更新在univ.i源文件中的UNIV_PAGE_SIZE和UNIV_PAGE_SIZE_SHIFT的值。

· 在MySQL 5.1中，触发器不被级联的外键行为激活。

15.2.17. InnoDB故障诊断和排除

15.2.17.1. InnoDB数据词典操作的错误诊断和排除

· 一个总的规则是，当一个操作失败或这你怀疑有一个缺陷。你应该查看MySQL服务器的错误日志，该日

志典型地有一个有些象*hostname*.err这样的名字，或者在Windows上是mysql.err这样的。

· 故障诊断与排除之时，通常最好从命令提示符运行MySQL服务器，而不是从**mysqld_safe**包运行，或不作为一个Windows服务来运行。你可以看**mysqld**打印到控制台上的内容，因此更好掌握发生了什么。在Windows上，你必须用--console选项启动服务器将输出定向到控制台窗口

· 使用InnoDB Monitors获取关于某问题的信息。如果问题是性能相关的，或者你的服务器看起来被挂起，你应该使用innodb_monitor来打印InnoDB内部状态的信息，如果问题是关于锁定，则使用innodb_lock_monitor。如果问题是在表的创建或其它数据词典操作，使用innodb_table_monitor来打印InnoDB内部数据词典的内容。

· 如果你猜测一个表被破坏，则在该表上运行CHECK TABLE。

15.2.17.1. InnoDB数据词典操作错误诊断和排除

表的一个特殊问题是MySQL服务器以.frm文件来保存数据词典信息，它被放在数据库目录，然而InnoDB也存储信息到表空间文件里它自己的数据词典里。如果你把.frm文件移来移去；或者，如果服务器在数据词典操作的中间崩溃，.frm文件可能结束与InnoDB内部数据词典的同步。

一个不同步的数据词典的症状是CREATE TABLE语句失败。如果发生这种情况，你应该查看服务器的错误日志。如果日志说表已经存在于InnoDB内部数据词典当中，你在InnoDB表空间文件内有一个孤表，它没有对应的.frm文件。错误信息看起来象如下的:

```
InnoDB: Error: table test/parent already exists in InnoDB internal

InnoDB: data dictionary. Have you deleted the .frm file

InnoDB: and not used DROP TABLE? Have you used DROP DATABASE

InnoDB: for InnoDB tables in MySQL version <= 3.23.43?

InnoDB: See the Restrictions section of the InnoDB manual.

InnoDB: You can drop the orphaned table inside InnoDB by

InnoDB: creating an InnoDB table with the same name in another

InnoDB: database and moving the .frm file to the current database.

InnoDB: Then MySQL thinks the table exists, and DROP TABLE will

InnoDB: succeed.
```

你可以按照错误日志里给的指示移除一个孤表。如果还是不能成功地使用DROP TABLE，问题可能是因为在**mysql**客户端里的名字完成。要解决这个问题，用--disable-auto-rehash选项来启动**mysql**客户端并再次尝试DROP TABLE 。（有名字完成打开着，**mysql**试着构建个表名字的列表，当一个正如描述的问题存在之时，这个列表就不起作用）。

不同步数据词典的另一个"同义词"是MySQL打印一个不能打开.InnoDB文件的错误:

```
ERROR 1016: Can't open file: 'child2.InnoDB'. (errno: 1)
```

在错误日志你可以发现一个类似于此的信息:

```
InnoDB: Cannot find table test/child2 from the internal data dictionary

InnoDB: of InnoDB though the .frm file for the table exists. Maybe you

InnoDB: have deleted and recreated InnoDB data files but have forgotten

InnoDB: to delete the corresponding .frm files of InnoDB tables?
```

这意味这有一个孤单的.frm文件，在InnoDB内没有相对应的表。你可以通过手动删除来移除这个孤单的.frm文

件。

如果MySQL在一个 ALTER TABLE操作的中间崩溃，你可以用InnoDB表空间内临时孤表来结束。你可以用innodb_table_monitor看一个列出的表，名为#sql-...。如果你把表的名字包在`(backticks)里，你可以在名字包含"#"字符的表上执行SQL语句。因此，你可以用前述的的方法象移除其它孤表一样移除这样一个孤表。注意，要在Unix外壳里复制或重命名一个文件，如果文件名包含"#"字符，你需要把文件名放在双引号里。

15.3. MERGE存储引擎

15.3.1. MERGE表的问题

MERGE存储引擎，也被认识为MRG_MyISAM引擎，是一个相同的可以被当作一个来用的MyISAM表的集合。"相同"意味着所有表同样的 列和索引信息。你不能合并列被以不同顺序列于其中的表，没有恰好同样列的表，或有不同顺序索引的表。而且，任何或者所有的表可以用**myisampack**来压缩。请参阅8.2节，"myisampack — 生成压缩的只读MyISAM表"。表选项的差异，比如AVG_ROW_LENGTH, MAX_ROWS或PACK_KEYS都不重要。

当你创建一个MERGE表之时，MySQL在磁盘上创建两个文件。文件名以表的名字开始，并且有一个扩展名来指明文件类型。一个.frm文件存储表定义，一个.MRG文件包含被当作一个来用的表的名字。这些表作为MERGE表自身，不必要在同一个数据库中。

你可以对表的集合用SELECT, DELETE, UPDATE和INSERT。你必须对你映射到一个MERGE表的这些表有SELECT, UPDATE和DELETE 的权限。

如果你DROP MERGE表，你仅在移除MERGE规格。底层表没有受影响。

当你创建一个MERGE表之时，你必须指定一个UNION=(*list-of-tables*)子句，它说明你要把哪些表当作一个来用。如果你想要对MERGE表的插入发生在UNION列表中的第一个或最后一个表上，你可以选择地指定一个INSERT_METHOD选项。使用FIRST或LAST值使得插入被相应地做在第一或最后一个表上。如果你没有指定INSERT_METHOD选项，或你用一个NO值指定该选项。往MERGE表插入记录的试图导致错误。

下面例子说明如何创建一个MERGE表:

```
mysql> CREATE TABLE t1 (
    ->    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->    message CHAR(20));
mysql> CREATE TABLE t2 (
    ->    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->    message CHAR(20));
mysql> INSERT INTO t1 (message) VALUES ('Testing'),('table'),('t1');
mysql> INSERT INTO t2 (message) VALUES ('Testing'),('table'),('t2');
mysql> CREATE TABLE total (
    ->    a INT NOT NULL AUTO_INCREMENT,
    ->    message CHAR(20), INDEX(a))
    ->    TYPE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;
```

注意，一个列在MERGEN表中被索引，但没有被宣告为一个PRIMARY KEY，因为它是在更重要的MyISAM表中。这是必要的，因为MERGE表在更重要的表中的设置上强制非唯一性。

创建MERGE表之后，你可以发出把一组表当作一体来操作的查询：

```
mysql> SELECT * FROM total;
+---+---------+
| a | message |
+---+---------+
| 1 | Testing |
| 2 | table   |
| 3 | t1      |
| 1 | Testing |
| 2 | table   |
| 3 | t2      |
+---+---------+
```

注意，你也可以直接从MySQL之外直接操作.MRG文件：

```
shell> cd /mysql-data-directory/current-database
shell> ls -1 t1 t2 > total.MRG
shell> mysqladmin flush-tables
```

要重映射一个MERGE表到一个不同的MyISAM表集，你可以执行下列之一：

- DROP MERGE表并且重建它。
- 使用ALTER TABLE *tbl_name* UNION=(...)来改变底层表的列表。
- 改变.MRG文件，并对MERGE表或者所有底层表发出一个FLUSH TABLE语句来强制存储引擎去读新的定义文件。

MERGE表可以帮你解决以下问题：

- 容易地管理一套日志表。比如，你可以把不同月的数据放进分离的表中，用**myisampack**压缩其中的一些，并随后创建一个MERGE表来把它们当作一个来使用。
- 获得更快的速度。你可以以一些标准来分割大的只读表，然后放进不同磁盘上的单个表中。基于此的一个MERGE表可比使用大表要快得多。
- 执行更有效的搜索。如果你确切知道要搜索什么，对一些查询你可以只在被分割的表的其中之一来搜索，并且对其它使用MERGE。你甚至有许多不同的MERGE表，它们使用有重叠的表套。
- 执行更有效的修补。修补被映射到一个MERGE表中的单个表比修补单个大型表要更轻松。
- 即刻映射许多表成一个。MERGE表不需要维护它自己的索引，因为它使用大哥表的所用。因此，MERGE表集合是非常块地创建或重映射。（注意，当你创建一个MERGE表之时，即使没有索引被创建，你必须仍然指定索引定义）。
- 如果根据需要或按照批次，你有一组要合起来作为一个大表的表，你应该根据需要对它们创建一个MERGE表来替代大表。这样要快得多而且节约大量的磁盘空间。
- 超过操作系统的文件尺寸限制。每个MyISAM表都受制于这个限制，但是一个MyISAM表的集合则不然。

· 你可以通过定义一个映射到单个表的MERGE表来为一个MyISAM表创建一个别名或"同物异名"。这样做应该没有真实的可察觉的性能影响 （对每个读只有一些间接调用和memcpy()调用）。

MERGE表的缺点:

· 你可以对MERGE表使用仅相同的MyISAM表。

· 你不能在MERGE表中使用很多MyISAM功能。比如，你不能在MERGE表上创建FULLTEXT索引。（当然，你可以在底层MERGE 表上创建FULLTEXT索引，但是你不能用全文搜索来搜索MERGE表）。

· 如果MERGE表是非临时的，所有底层MyISAM表也必须是永久的。如果MERGE表是临时的，MyISAM表可以是任何临时＆非临时的混合。

· MERGE表使用更多的文件描述符。如果是个客户端正使用一个映射到10个表的MERGE表，服务器使用(10*10)+10个文件描述符。（10个数据文件描述符给10个客户端每人一个，并且在客户端之间共享10个索引文件描述符）。

· 键读会更慢。当你读一个键的时候，MERGE存储引擎需要在所有 底层表上发出一个读以检查哪一个接近匹配给定的键。如果你随后做了一个read-next，MERGE存储引擎需要搜索读缓冲来找出下一个键。只有当一个键缓冲被耗尽，存储引擎才需要读下一个 键块。这使得MERGE键在eq_ref搜索中非常慢，但在ref搜索中不是太慢。请参阅7.2.1节，"EXPLAIN 语法(获取SELECT相关信息)" 以获得更多关于eq_ref和ref的信息。

15.3.1. MERGE表 方面的问题

下列是已知关于MERGE表的问题:

· 如果你使用ALTER TABLE 来把MERGE表变为其它表类型，到 底层表的映射就被丢失了。取而代之的，来自底层MyISAM表的行被复制到已更换的表中，该表随后被指定新类型。

· REPLACE不起作用。

· 没有WHERE子句，或者在任何被映射到一个打开的MERGE表上的任何一个表上的REPAIR TABLE，TRUNCATE TABLE, OPTIMIZE TABLE或ANALYZE TABLE，你不能使用DROP TABLE, ALTER TABLE, DELETE FROM。如果你这么做了，MERGE表将仍旧指向原始表，这样产生意外结果。解决这个不足最简单的办法是在执行任何一个这些操作之前发出一个FLUSH TABLES语句来确保没有MERGE表仍旧保持打开。

· 一个MERGE表不能在整个表上维持UNIQUE约束。当你执行一个INSERT, 数据进入第一个或者最后一个MyISAM表（取决于INSERT_METHOD选项的值）。MySQL确保唯一 键值在那个MyISAM表里保持唯一，但不是跨集合里所有的表。

· 当你创建一个MERGE表之时，没有检查去确保底层表的存在以及有相同的机构。当MERGE表被使用之时，MySQL检查每个被映射的表的记录长度是否相等，但这并不十分可靠。如果你从不相似的MyISAM表创建一个MERGE表，你非常有可能撞见奇怪的问题。

· 在MERGE表中的索引的顺序和它的 底层表中的索引应该一样。如果你使用ALTER TABLE给一个被用在MERGE表中的表添加一个UNIQUE索引，然后使用ALTER TABLE在MERGE表上添加一个非唯一索引，如果在底层表上已经有一个非唯一索引，对表的索引排序是不同的。（这是因为ALTER TABLE把UNIQUE索引放在非唯一索引之前以利于重复键的快速检测 ）。因此对使用这样索引的表的查询可能返回不期望的结果。

· 在Windows中，在一个被MERGE表使用的表上DROP TABLE不起作用，因为MERGE引擎的表映射对MySQL的更上层隐藏。因为Windows不允许已打开文件的删除，你首先必须 刷新所有MERGE表（使用FLUSH TABLES）或在移除该表之前移除MERGE表。

对于MERGE存储引擎，在http://forums.mysql.com/list.php?93上有一个专门的论坛。

15.4. MEMORY (HEAP)存储引擎

MEMORY存储引擎用存在内存中的内容来创建表。这些在以前被认识为HEAP表。MEMORY是一个首选的术语，虽然为向下兼容，HEAP依旧被支持。

每个MEMORY表和一个磁盘文件关联起来。文件名由表的名字开始，并且由一个.frm的扩展名来指明它存储的表定义。

要明确指出你想要一个MEMORY表，可使用ENGINE选项来指定：

```
CREATE TABLE t (i INT) ENGINE = MEMORY;
```

如它们名字所指明的，MEMORY表被存储在内存中，且默认使用哈希索引。这使得它们非常快，并且对创建临时表非常有用。可是，当服务器关闭之时，所有存储在MEMORY表里的数据被丢失。因为表的定义被存在磁盘上的.frm文件中，所以表自身继续存在，在服务器重启动时它们是空的。

这个例子显示你如何可以创建，使用并删除一个MEMORY表：

```
mysql> CREATE TABLE test ENGINE=MEMORY
    ->     SELECT ip,SUM(downloads) AS down
    ->     FROM log_table GROUP BY ip;
mysql> SELECT COUNT(ip),AVG(down) FROM test;
mysql> DROP TABLE test;
```

MEMORY表有下列特征：

- 给MEMORY表的空间被以小块来分配。表对插入使用100%动态哈希来。不需要溢出区或额外键空间。自由列表无额外的空间需求。已删除的行被放在一个以链接的列表里，并且在你往表里插入新数据之时被重新使用。MEMORY表也没有通常与在哈希表中删除加插入相关的问题。

- MEMORY表可以有多达每个表32个索引，每个索引16列，以及500字节的最大 键长度。

- MEMORY存储引擎执行HASH和BTREE索引。你可以通过添加一个如下所示的USING子句为给定的索引指定一个或另一个：

```
CREATE TABLE lookup
    (id INT, INDEX USING HASH (id))
    ENGINE = MEMORY;
CREATE TABLE lookup
    (id INT, INDEX USING BTREE (id))
    ENGINE = MEMORY;
```

B树的一般特征和哈希索引在7.4.5节，“MySQL如何使用索引”里描述。

- 你可以在一个MEMORY表中有非唯一键。（对哈希索引的实现，这是一个不常用的功能）。

- 你页可以对MEMORY表使用INSERT DELAYED。请参阅13.2.4.2节，“INSERT DELAYED语法”

- 如果你在一个有高度键重复的（许多索引条目包含同一个值）MEMORY表上有一个哈希索引，对影响键值的表的更新及所有删除都是明显地慢的。这个变慢的程度比例于重复的程度（或者反比于索引cardinality）。你可以使用一个B树索引来避免这个问题。

- MEMORY表使用一个固定的记录长度格式。
- MEMORY不支持BLOB或TEXT列。
- MEMORY支持AUTO_INCREMENT列和对可包含NULL值的 列的索引。
- MEMORY表在所有客户端之间共享（就像其它任何非TEMPORARY表）。
- MEMORY表内容被存在内存中，内存是MEMORY表和服务器在查询处理之时的空闲中创建的内部表共享。可是，两个类型的表不同在于MEMORY表不会遇到存储转换，而内部表是：

- 如果一个内部表变得太大，服务器自动把它转换为一个磁盘表。尺寸限制由tmp_table_size系统变量的值来确定。

- MEMORY表决不会转换成磁盘表。要确保你不会偶尔做点傻事，你可以设置max_heap_table_size系统变量给MEMORY表加以最大尺寸。对于单个的表，你也可以在CREATE TABLE语句中指定一个MAX_ROWS表选项。

- 服务器需要足够内存来维持所有在同一时间使用的MEMORY表。
- 当你不再需要MEMORY表的内容之时，要释放被MEMORY表使用的内存，你应该执行DELETE FROM或TRUNCATE TABLE，或者整个地删除表（使用DROP TABLE）。
- 当MySQL服务器启动时，如果你想开拓MEMORY表，你可以使用--init-file选项。例如，你可以把INSERT INTO ... SELECT 或LOAD DATA INFILE这样的语句放入这个文件中以便从持久稳固的的数据源装载表。请参阅5.3.1节，“**mysqld** 命令行选项” and 13.2.5节，“LOAD DATA INFILE 语法”。
- 如果你正使用复制，当主服务器被关闭且重启动之时，主服务器的MEMORY表变空。可是从服务器意识不到这些表已经变空，所以如果你从它们选择数据，它就返回过时的内容。自从服务器启动后，当一个MEMORY表在主服务器上第一次被使用之时，一个DELETE FROM语句被自动写进主服务器的二进制日志，因此再次让从服务器与主服务器同步。注意，即使使用这个策略，在主服务器的重启和它第一次使用该表之间的间隔中，从服务器仍旧在表中有过时数据。可是，如果你使用--init-file选项于主服务器启动之时在其上推行MEMORY表。它确保这个时间间隔为零。
- 在MEMORY表中，一行需要的内存使用下列表达式来计算：

```
SUM_OVER_ALL_BTREE_KEYS(max_length_of_key + sizeof(char*) * 4)
+ SUM_OVER_ALL_HASH_KEYS(sizeof(char*) * 2)
+ ALIGN(length_of_row+1, sizeof(char*))
```

ALIGN()代表round-up因子，它使得行的长度为char指针大小的确切倍数。sizeof(char*)在32位机器上是4，在64位机器上是8。

对于MEMORY存储引擎，在http://forums.mysql.com/list.php?92上有一个专门的论坛。

15.5. BDB (BerkeleyDB)存储引擎

15.5.1. BDB支持的操作系统

15.5.2. 安装BDB

15.5.3. BDB启动选项

15.5.4. BDB表的特征

15.5.5. 修改BDB所需 的事宜

15.5.6. BDB表的限制

15.5.7. 使用BDB表时可能发生的错误

Sleepycat Software给MySQL提供Berkeley DB事务性存储引擎。这个存储引擎典型被简称为BDB。对BDB存储引擎的支持包括在MySQL源码分发版里，在MySQL-Max二进制分发版里被激活。

BDB表可能有一个更大的崩溃幸存机会，并且也具有对事务COMMIT和ROLLBACK操作的能力。MySQL源码分发版和被补丁过可以与MySQL一起工作的BDB分发版一起提供。你不能使用一个未补丁过的BDB版本与MySQL一起工作。。

我们在MySQL AB上与Sleepycat紧密合作工作以保持MySQL/BDB接口的质量在高水平。（即使Berkeley DB其本身是非常能经受考验和非常可靠的。MySQL接口仍然被认为是第三等质量的。我们将继续改善和优化它）。

当它达到对所有涉及BDB表的问题的支持之时，我们答应负责帮助我们的用户定位问题并创建可重复产生的测试案例。任何这样的测试案例被转交给Sleepycat，它反过来帮助我们找到并解决问题。因为这是一个二阶段的操作，任何关于BDB表的问题我们可能要花比对其它存储引擎稍微更长一点的时间来解决它。可是，我们期望这个过程没有显著的困难，因为Berkeley DB 代码本身被用在MySQL之外许多的应用中。

要获得关于Berkeley DB的一般信息，请访问Sleepycat网站，http://www.sleepycat.com/。

15.5.1. BDB支持的操作系统

当前，我们知道BDB存储引擎支持下列操作系统：

- Linux 2.x Intel
- Sun Solaris (SPARC and x86)
- FreeBSD 4.x/5.x (x86, sparc64)
- IBM AIX 4.3.x
- SCO OpenServer
- SCO UnixWare 7.1.x
- Windows NT/2000/XP

BDB不支持下列操作系统：

- Linux 2.x Alpha
- Linux 2.x AMD64
- Linux 2.x IA-64
- Linux 2.x s390
- Mac OS X

注释：前一个列表还不完全，我们收到更多信息时我们会更新它。

如果你从支持BDB表的源码建立的MySQL，但是，当你启动**mysqld**之时，发生下列错误，这意味着对你的架

构BDB不被支持:

```
bdb: architecture lacks fast mutexes: applications cannot be threaded

Can't init databases
```

在这种情况下，你必须重建MySQL不带BDB表支持或用--skip-bdb选项启动服务器。

15.5.2. 安装BDB

如果你已经下载一个包括对Berkeley DB支持的的二进制版的MySQL， 只要简单地按照通常的二进制分发版安装指令。（MySQL-Max 分发版包括BDB支持）。

如果你从源码建立MySQL，你可以在其它任何常用选项之外用--with-berkeley-db选项运行**configure**来允许支持BDB。下载一个MySQL 5.1分发版，改变位置到它的最顶层目录，然后运行下面命令:

```
shell> ./configure --with-berkeley-db [other-options]
```

更多信息，请参阅2.7节，“在其它类似Unix系统上安装MySQL”, 5.1.2节，“**mysqld-max**扩展MySQL服务器”, 和2.8节，“使用源码分发版版安装MySQL”。

15.5.3. BDB启动选项

下列对**mysqld**的选项可被用来改变BDB存储引擎的行为:

- --bdb-home=*path*

BDB表的基础目录。这应该和你为--datadir使用的目录相同。

- --bdb-lock-detect=*method*

BDB 锁定检测方式。选项值应该为DEFAULT, OLDEST, RANDOM或YOUNGEST。

- --bdb-logdir=*path*

BDB日志文件目录。

- --bdb-no-recover

不在恢复模式启动Berkeley DB。

- --bdb-no-sync

不同步刷新BDB日志。这个选项不被赞成，取而代之地使用--skip-sync-bdb-logs（请参阅对--sync-bdb-logs的描述)。

- --bdb-shared-data

以多处理模式启动Berkeley DB。（初始化Berkeley DB之时，不要使用DB_PRIVATE）。

- --bdb-tmpdir=*path*

BDB临时文件目录。

- --skip-bdb

禁止BDB存储引擎。

- --sync-bdb-logs

同步刷新BDB日志。这个选项默认被允许，请使用--skip-sync-bdb-logs来禁止它。

请参阅5.3.1节，“**mysqld**命令行选项”。

如果你使用--skip-bdb选项，MySQL不初始化Berkeley DB库，而且这样节省大量的内存。尽管如此，如果你使用这个选项，你不能使用BDB表。如果你试着创建一个BDB表，MySQL取而代之地创建一个MyISAM。

通常，如果你象使用BDB表，你应该不用--bdb-no-recover选项启动**mysqld**。可是，如果BDB日志被破坏则当你试着启动**mysqld**时，上述办法启动服务器可能导致问题。请参阅2.9.2.3节，“MySQL服务器的启动和故障诊断排除”。

使用bdb_max_lock 变量，你可以指定在BDB表上被激活的锁定的最大数目。默认值是10,000。当你执行长事务或当**mysqld**不得不检查许多行来执行一个查询之时，如果发生如下错误，你应该增加这个数目：

```
bdb: Lock table is out of available locks

Got error 12 from ...
```

如果你正使用大型多语句事务，你可能也想改变binlog_cache_size和max_binlog_cache_size变量。请参阅5.11.3节，“二进制日志”。

也请参阅5.3.3节，“服务器系统变量”。

15.5.4. BDB表的特征

每个BDB表用两个文件被存在磁盘上。文件的名字用表的名字做开头，并且有一个扩展名来指明文件类型。一个.frm文件存储表定义，一个.db文件包含表数据和索引。

要明确指出你想要一个BDB表，用ENGINE或TYPE表选项来指明：

```
CREATE TABLE t (i INT) ENGINE = BDB;

CREATE TABLE t (i INT) TYPE = BDB;
```

BerkeleyDB是用ENGINE或者TYPE选项的BDB存储引擎的“同义词”。

BDB存储引擎提供事务性表，你使用这些表的方法取决于autocommit模式：

- 如果你正运行着，同时随着autocommit的被允许（这是默认的），对BDB表的改变被立即提交并且不能被 回滚。

- 如果你正运行着，同时随着autocommit的被禁止，改变不变成永久的直到你执行一个COMMIT语句。作为提交的替代，你可以执行ROLLBACK来忘记改变。

你可以用BEGIN WORK语句开始一个事务来挂起autocommit，或者用SET AUTOCOMMIT=0来明确禁止autocommit。

请参阅13.4.1节，“START TRANSACTION, COMMIT和ROLLBACK语法”。

BDB存储引擎有下列特征：

- BDB表可以有多达每表31个索引，每个索引16列，并且1024字节的最大 键尺寸。

- MySQL在每个BDB表中需要一个PRIMARY KEY以便每一行可以被唯一地识别。如果你不明确创建一个，MySQL为你创建并维持一个隐藏的PRIMARY KEY。隐藏的 键有一个5字节的长度，并且为每个插入的企图

而被增加。这个键不出现在SHOW CREATE TABLE或DESCRIBE的输出之中。

· PRIMARY KEY比任何其它索引都要快，因为PRIMARY KEY被与行的数据一起存储。其它索引被存储为键数据＋PRIMARY KEY，所以保持PRIMARY KEY尽可能地短以节约磁盘空间并获得更好速度是重要的。

这个行为类似于InnoDB的，在其中较短的primary keys不仅在主索引也在第二索引节约空间 。

· 如果在BDB表中，你访问的所有列是同一索引的一部分或主键的一部分，MySQL可以执行查询而不访问确实的行。在一个MyISAM表中，只有 列是同一索引的一部分之时，才可以这么做。

· 连续扫描比对MyISAM表的扫描更慢，因为在BDB表中的数据被存储在B树而不是在分离的数据文件中。

· 键值象MyISAM表中的 键值一样不是前缀或后缀压缩的。换句话说，在BDB表中键信息只比在MyISAM表中稍微多占据一点空间。

· 在BDB表中经常有洞允许你在索引树的中间插入新行。这个使得BDB表比MyISAM表稍微大一些

· SELECT COUNT(*) FROM *tbl_name*对BDB表很慢，因为在该表中没有行计数被维持。

· 优化器需要知道表中的大概行数。MySQL通过计数插入以及在每个BDB表中的单独片断内维持它来解决了问题。如果你不发出大量的DELETE或ROLLBACK语句，这个数对MySQL优化器来说是足够精确了。可是，MySQL仅在关闭的时候才存储这个数，所以，如果服务器被意外地终止，这个数可能是不正确的。即使数不是100%正确，它不是 明确的。你可以使用ANALYZE TABLE或者 OPTIMIZE TABLE语句来更新行计数。请参阅13.5.2.1节，“ANALYZE TABLE语法” 和13.5.2.5节，“OPTIMIZE TABLE语法”。

· BDB表上的内部锁定在页面级别上做。

· LOCK TABLES在BDB表上就想在其它表上一样工作。如果你不使用LOCK TABLES，MySQL对该表发出一个内部多重写锁定 （一个不阻止其它作者的锁定）来确保即使另一个线程发出一个表锁定，该表也被恰当地锁定了。

· 要能够回滚一个事务，BDB存储引擎维护日志文件。多实现最高性能，你可以使用--bdb-logdir选项来把BDB日志放在不同的磁盘上，而不是放在数据库被放置的那个磁盘。

· 每次新BDB日志文件被启动之时，MySQL执行一个检查点，并且删掉当前事务不需要的任何BDB日志文件。你也可以在任何时候使用FLUSH LOGS来给Berkeley DB表设置检查点。

对灾难恢复，你应该使用表备份加MySQL的二进制日志，请参阅5.9.1节，“数据库备份”。

警告：如果你删除仍在使用中的旧日志文件，BDB根本不能做恢复，并且如果有些事不对，你可能会丢失数据。

· 应用程序必须总是被准备来处理情况，即BDB表任何的改变可能导致一个自动回滚并且任何读可能会带着一个死锁错误而失败。

· 如果你在BDB表内遇到磁盘满，你得到一个错误（可能是错误28），并且事务应该回滚。这与MyISAM表相反，对于MyISAM 表，**mysqld**在继续之前等待足够的自由磁盘空间。

15.5.5. 修改**BDB**所需的 事宜

· 同时打开许多BDB表可能是非常慢的。如果你要使用BDB表，你不应使用非常大的表缓存（例如，大于256）并且当你使用**mysql**客户端之时，你应该使用--no-auto-rehash选项。

· SHOW TABLE STATUS 不为BDB表提供一些信息：

· `mysql>`

```
SHOW TABLE STATUS LIKE 'bdbtest'\G
*************************** 1. row ***************************
           Name: bdbtest
         Engine: BerkeleyDB
        Version: 10
     Row_format: Dynamic
           Rows: 154
 Avg_row_length: 0
    Data_length: 0
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: NULL
    Create_time: NULL
    Update_time: NULL
     Check_time: NULL
      Collation: latin1_swedish_ci
       Checksum: NULL
 Create_options:
        Comment:
```

- 优化性能。
- 改变为对表扫描操作使用无页面锁定。

15.5.6. 对**BDB**表的限制

下列表说明使用BDB表之时你必须要遵从的限制:

- 每个BDB表在.db文件里存储文件被创建之时到该文件的路径。这个被做来允许在支持symlinks的多用户环境里检测锁定。因此，从一个数据库目录往另一个目录移动BDB表是不能的。
- 当制作BDB表的备份之时，你必须要么使用**mysqldump**要么做一个包含对每个BDB表的文件（.frm和.db文件）及BDB日志文件的备份。BDB存储引擎在它的日志文件存储未完成的事务以及要求它们在**mysqld**启动的时候被呈出来。BDB日志在数据目录里，具有log.XXXXXXXXXX（10位数字）形式名字的文件。
- 如果允许NULL值的列有唯一的索引，只有单个NULL值是被允许的。这不同于其它存储引擎。

15.5.7. 使用**BDB**表时可能发生的错误

- 如果你升级之后启动**mysqld**时发生下列错误，它意味着新BDB版本不支持旧日志文件格式:

```
bdb:  Ignoring log file: .../log.XXXXXXXXXX:
unsupported log version #
```

BDB log.XXXXXXXXXX

在这种情况下，你必须删除从数据目录所有 日志（名字为 这样格式的文件）并重新启动**mysqld**。我们也推荐你随后用**mysqldump --opt**来转储你的BDB表，移除表，并且从转储文件恢复它们。

- 如果autocommit模式被禁止，你移除在另一个事务中被参考的BDB表，你会从你的MySQL错误日志得到如下的错误信息：

```
·               001119 23:43:56  bdb:  Missing log fileid entry
·               001119 23:43:56  bdb:  txn_abort: Log undo failed for LSN:
·                                     1 3644744: Invalid
```

这不是一个致命错误，但是知道问题被解决，我们推荐你不要移除BDB表，除了autocommit模式被允许之时。（修复不是微不足道的）。

15.6. EXAMPLE存储引擎

EXAMPLE存储引擎是一个不做任何事情的存根引擎。它的目的是作为MySQL源代码中的一个例子，用来演示如何开始编写一个新存储引擎。 同样，它的主要兴趣是对开发者。

要对EXAMPLE引擎检查源 码，请查看MySQL源码分发版的sql/examples目录。

要允许这个存储引擎，使用在建立MySQL之时使用--with-example-storage-engine选项来**configure**。

当你创建一个EXAMPLE之时，服务器创建在数据库目录创建一个表定义文件。文件由表名字开始，并由一个.frm的扩展名。没有其它文件被创建。没有数据被存储进表或者从中取回。

```
mysql> CREATE TABLE test (i INT) ENGINE = EXAMPLE;
Query OK, 0 rows affected (0.78 sec)

mysql> INSERT INTO test VALUES(1),(2),(3);
ERROR 1031 (HY000): Table storage engine for 'test' doesn't have this option

mysql> SELECT * FROM test;
Empty set (0.31 sec)
```

EXAMPLE存储引擎不支持编索引。

15.7. FEDERATED存储引擎

15.7.1. 安装FEDERATED存储引擎

15.7.2. FEDERATED存储引擎的 介绍

15.7.3. 如何使用FEDERATED表

15.7.4. FEDERATED存储引擎的 局限性

FEDERATED存储引擎访问在远程数据库的表中的数据，而不是本地的表。

FEDERATED存储引擎仅在－MAX版的MySQL上可用。

要为FEDERATED引擎检查 源码，请查看MySQL源码分发版的sql目录。

对于FEDERATED存储引擎，在http://forums.mysql.com/list.php?105上有一个专门的论坛。

15.7.1. 安装FEDERATED存储引擎

要允许这个存储引擎，当你构建MySQL时请使用--with-federated-storage-engine来**configure**。

15.7.2. FEDERATED存储引擎的描述

当你创建一个FEDERATED表的时候，服务器在数据库目录创建一个表定义文件。文件由表的名字开始，并有一个.frm扩展名。无其它表被创建，因为实际的数据在一个远程数据库上。这不同于为本地表工作的存储引擎的方式。

对于本地的数据录表，数据文件是本地的。例如，如果你创建一个名为user的MyISAM表，MyISAM处理器创建一个名为users.MYD的数据文件。对本地表读，插入，删除和更新在本地数据文件里的数据的处理器和记录被以对处理器的特定格式存储。 为了读记录，处理器必须把数据分解进列。为了写记录，列值必须被转换到被处理器使用的行格式并且被写进本地的数据文件。

使用MySQL FEDERATED存储引擎，没有对表的本地数据文件（比如，没有.MYD文件）。取而代之地，一个远程数据库存储那些正常地应该在表中的数据。这使得MySQL客户端API来读，删除，更新和插入数据的使用成为必要。数据取回被通过SELECT * FROM *tbl_name* SQL语句来初始化。要读这个结果，通过使用mysql_fetch_row() C API函数，行被一次取一个，然后从SELECT结果包中的列转换成FEDERATED处理器期望的格式。

基本流程如下:

1. SQL调用被本地发布
2. MySQL处理器API （数据以处理器格式）
3. MySQL客户端API （数据被转换成SQL调用）
4. 远程数据库-> MySQL客户端API
5. 转换结果包（如果有的话）到处理器格式
6. 处理器 API -> 结果行或受行影响的对本地的计数

15.7.3. 如何使用FEDERATED表

使用FEDERATED表的步骤是非常简单的。通常，你运行两个服务器，要么在同一个主机上，要么在不同主机上。（一个FEDERATED表使用其它被同一服务器管理的表也是可能的。虽然只有极少的点要这么做）。

首先，你必须在你想要用FEDERATED表访问的远程服务器上有一个表。假设，远程的表在FEDERATED数据库中并且被如下定义:

```
CREATE TABLE test_table (
    id     int(20) NOT NULL auto_increment,
    name   varchar(32) NOT NULL default '',
    other  int(20) NOT NULL default '0',
    PRIMARY KEY  (id),
    KEY name (name),
```

```
    KEY other_key (other)
)
ENGINE=MyISAM
DEFAULT CHARSET=latin1;
```

ENGINE表选项可能命名任何存储引擎，该表需要不是一个MyISAM表。

接着，在本地服务器上为访问远程表创建一个FEDERATED表：

```
CREATE TABLE federated_table (
    id     int(20) NOT NULL auto_increment,
    name   varchar(32) NOT NULL default '',
    other  int(20) NOT NULL default '0',
    PRIMARY KEY  (id),
    KEY name (name),
    KEY other_key (other)
)
ENGINE=FEDERATED
DEFAULT CHARSET=latin1
CONNECTION='mysql://root@remote_host:9306/federated/test_table';
```

（注意: CONNECTION 替代 用在先前版本的MySQL里的COMMENT）。

除了ENGINE表选项应该是FEDERATED，并且CONNECTION表选项是给FEDERATED指明如何连接到远程服务器上的连接字符串之外， 这个表的结构必须完全与远程表的结构相同。

FEDERATED引擎仅创建在已联盟数据库中的test_table.frm文件。

远程主机信息指明本地服务器要连接到的远程服务器，数据库和表信息指明哪一个远程表要被作为数据文件来用。在这个例子中。远程服务器被指定来作为远程主机在9306端口上运行，所以你要启动服务器，让它监听9306端口。

在CONNECTION选项中的连接字符串的一般形式如下：

scheme://*user_name*[:*password*]@*host_name*[:*port_num*]/*db_name*/*tbl_name*

只有mysql在这一点被支持为*scheme*，密码和端口号时可选的。

这里有一些连接字符串的例子：

```
CONNECTION='mysql://username:password@hostname:port/database/tablename'
CONNECTION='mysql://username@hostname/database/tablename'
CONNECTION='mysql://username:password@hostname/database/tablename'
```

为指定连接字符串使用CONNECTION是非可选，并且在将来可能会改变。当你使用FEDERATED表的时候，要记得这个，因为这意味着当将来发生那种改变之时，可能被要求。

因为任何被用的密码作为纯文本被存在连接字符串中，它可以被任何使对FEDERATED表使用SHOW CREATE TABLE或SHOW TABLE STATUS的用户，或者在INFORMATION_SCHEMA数据库中查询TABLES表的用户看见。

对于FEDERATED存储引擎，在http://forums.mysql.com/list.php?105上有一个专门的论坛。

15.7.4. FEDERATED存储引擎的局限 性

FEDERATED支持及不支持的如下:

- 在第一个版本中，远程服务器必须是一个MySQL服务器。FEDERATED对其它数据库引擎的支持可能会在将来被添加。
- FEDERATED表指向的远程表在你通过FEDERATED表访问它之前必须存在。
- 一个FEDERATED表指向另一个FEDERATED表是可能的，但是你必须小心不要创建一个循环。
- 没有对事务的支持。
- 如果远程表已经改变，对FEDERATED引擎而言是没有办法知道的。这个的原因是因为这个表必须象数据文件一样工作，除了数据库其它任何都不会被写入。如果有任何对远程数据库的改变，本地表中数据的完整性可能会被破坏。
- FEDERATED存储引擎支持SELECT, INSERT, UPDATE, DELETE和索引。它不支持ALTER TABLE, DROP TABLE或任何其它的数据定义语言语句。当前的实现不使用预先准备好的语句。
- 执行使用SELECT, INSERT, UPDATE和DELETE，但不用HANDLER。
- FEDERATED表不能对查询缓存不起作用。

这些限制中的一些在FEDERATED处理机的将来版本可能被消除。

15.8. ARCHIVE存储引擎

ARCHIVE存储引擎被用来以非常小的覆盖区存储大量无索引数据。

要允许这个存储引擎，在建立MySQL之时使用--with-archive-storage-engine选项来**configure**。如果这个存储引擎可带这个语句使用，你可以看见:

```
mysql> SHOW VARIABLES LIKE 'have_archive';
```

当年创建一个ARCHIVE表，服务器在数据库目录创建一个表定义文件。文件由表的名字开始，并由一个.frm的扩展名。存储引擎创建其它文件，所有都有由表名字开头的名字。数据和元数据文件有扩展名.ARZ和.ARM。一个.ARN文件可能在优化操作中出现。

ARCHIVE引擎仅支持INSERT和SELEC（无删除，替换或更新）。它支持ORDER BY操作，BLOB 域，以及基本地所有数据类型，除了几何数据类型（请参阅19.4.1节，“MySQL 空间数据类型”）。ARCHIVE引擎使用行级锁定。

存储**:** 当记录被插入时，它们被压缩。ARCHIVE引擎使用zlib无损数据压缩。OPTIMIZE TABLE的使用可以分析表，并把它打包为更小的格式（ 使用OPTIMIZE TABLE的原因，往下看）。引擎页支持CHECK TABLE。有数种被使用的插入类型:

- 直接插入INSERT之时把一行推仅压缩缓冲，并且缓冲在它需要的时候刷新。到缓冲的插入被锁定保护。一个SELECT强制一个 刷新发生，除非进来的唯一插入是INSERT DELAYED（那些刷新如同需要一样）。请参阅13.2.4.2节，“INSERT DELAYED 语法”。
- 块插入旨在它完成后可见，除非其它插入在同一时间发生，在这种情况下，块插入可以被部分看见。一个SELECT不会导致一个块插入的 刷新，除非SELECT在被装载时发生一个正常插入。

取回： 在取回时，记录根据需要被解压缩，没有行缓存。一个SELECT操作执行完全表格扫描：当一个SELECT发生之时，它找出当前有多少行可用，并读行的数量。SELECT被当作持续读来执行。注意，许多SELECT语句在插入过程中会损坏压缩，除非块插入或者延迟的插入被使用。要修复任何已发生压缩问题，你可以总是做OPTIMIZE TABLE（也支持REPAIR TABLE）。被SHOW TABLE STATUS报告的行数总是正确的。请参阅13.5.2.6节，“REPAIR TABLE语法”，13.5.2.5节，“OPTIMIZE TABLE语法”，13.5.4.18节，“SHOW TABLE STATUS 语法”。

对于ARCHIVE存储引擎，在 http://forums.mysql.com/list.php?112 上有专门论坛。

15.9. CSV存储引擎

CSV存储引擎使用逗号分隔值格式的文本文件存储数据。

要允许使用这个存储引擎，当你建立MySQL之时，使用--with-csv-storage-engine选项来**configure**。

当你创建一个CSV表之时，服务器在数据库目录创建一个表定义文件。文件由表的名字开始，并且由一个.frm的扩展名。存储引擎也创建一个数据文件。它的名字由表的名字开始，并且有一个.CSV的扩展名。数据文件是无格式文本文件。当你把数据存储进表时，存储引擎用CSV格式把它存进数据文件。

```
mysql> CREATE TABLE test(i INT, c CHAR(10)) ENGINE = CSV;
Query OK, 0 rows affected (0.12 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM test;
+------+------------+
| i    | c          |
+------+------------+
|    1 | record one |
|    2 | record two |
+------+------------+
2 rows in set (0.00 sec)
```

如果你检查在执行前述语句所创建的数据库目录里的test.CSV文件，它的内容应该看起来象如下所示：

```
"1","record one"
"2","record two"
```

CSV存储引擎不支持索引。

15.10. BLACKHOLE存储引擎

BLACKHOLE存储引擎就像“黑洞”一样，它接收数据但丢弃它而不是存储它。取回总是返回空集：

```
mysql> CREATE TABLE test(i INT, c CHAR(10)) ENGINE = BLACKHOLE;
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM test;
Empty set (0.00 sec)
```

当你创建一个BLACKHOLE表的时候，服务器在数据库目录创建一个表定义文件。文件用表的名字开头，并且有一个.frm扩展名。没有其它文件关联到这个表格。

BLACKHOLE存储引擎支持所有种类的索引。

要允许这个存储引擎，在你建立MySQL之时使用--with-blackhole-storage-engine选项来**configure**。BLACKHOLE存储引擎在MySQ供应的服务器二进制版里可以找到；通过查看SHOW ENGINES或SHOW VARIABLES LIKE 'have%'的输出，你可以确定你的版本是否支持这个引擎。

到BLACKHOLE表的插入不存储任何数据，但如果二进制日志被允许，SQL语句被写入日志（并被复制到从服务器）。这可以有用帮助地作为重复器或过滤器机制。例如，假设你的应用需要从服务器侧的过滤规则，但传输所有二进制日志数据到从服务器首先导致过多交通的结果。在这种情况下，在主服务器主机上建立一个伪从服务器进程，它的存储引擎是BLACKHOLE，描述如下：

主服务器写它的二进制日志。伪mysqld进程作为从服务器，应用期望的replicate-do和replicate-ignore规则的合并，并且写一个新的，被过滤的属于它自己的二进制日志 。（请参阅6.8节，“复制启动选项”）。这个已过滤日志被提供给从服务器。

既然伪进程不确实地存储任何数据，只有很小的由在复制主服务器主机上额外的mysqld进程招致的处理开支。这个类型的建立可以用额外复制从服务器来重复。

其它可能对BLACKHOLE存储引擎的使用包括：

- 转储文件语法的验证。
- 来自二进制日志记录的开销测量，通过比较允许二进制日志功能的BLACKHOLE的性能与禁止二进制日志功能的BLACKHOLE的性能。
- 因为BLACKHOLE本质上是一个“no-op” 存储引擎，它可能被用来查找与存储引擎自身不相关的性能瓶颈。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第16章：编写自定义存储引擎

目录

16.1. 前言
16.2. 概述
16.3. 创建存储引擎源文件
16.4. 创建handlerton
16.5. 对处理程序进行实例化处理
16.6. 定义表扩展
16.7. 创建表
16.8. 打开表
16.9. 实施基本的表扫描功能

16.9.1. 实施store_lock()函数
16.9.2. 实施external_lock()函数
16.9.3. 实施rnd_init()函数
16.9.4. 实施info()函数
16.9.5. 实施extra()函数
16.9.6. 实施rnd_next()函数

16.10. 关闭表
16.11. 为存储引擎添加对INSERT的支持
16.12. 为存储引擎添加对UPDATE的支持
16.13. 为存储引擎添加对DELETE的支持
16.14. API引用

16.14.1. bas_ext
16.14.2. close
16.14.3. create
16.14.4. delete_row
16.14.5. delete_table
16.14.6. external_lock
16.14.7. extra
16.14.8. info
16.14.9. open
16.14.10. rnd_init
16.14.11. rnd_next
16.14.12. store_lock
16.14.13. update_row
16.14.14. write_row

16.1. 前言

对于MySQL 5.1，MySQL AB公司引入了插件式存储引擎体系结构，这样，就能创建新的存储引擎，并将它们添加到正在运行的MySQL服务器上，而不必重新编译服务器本身。

该体系结构简化了新存储引擎的开发和部署。

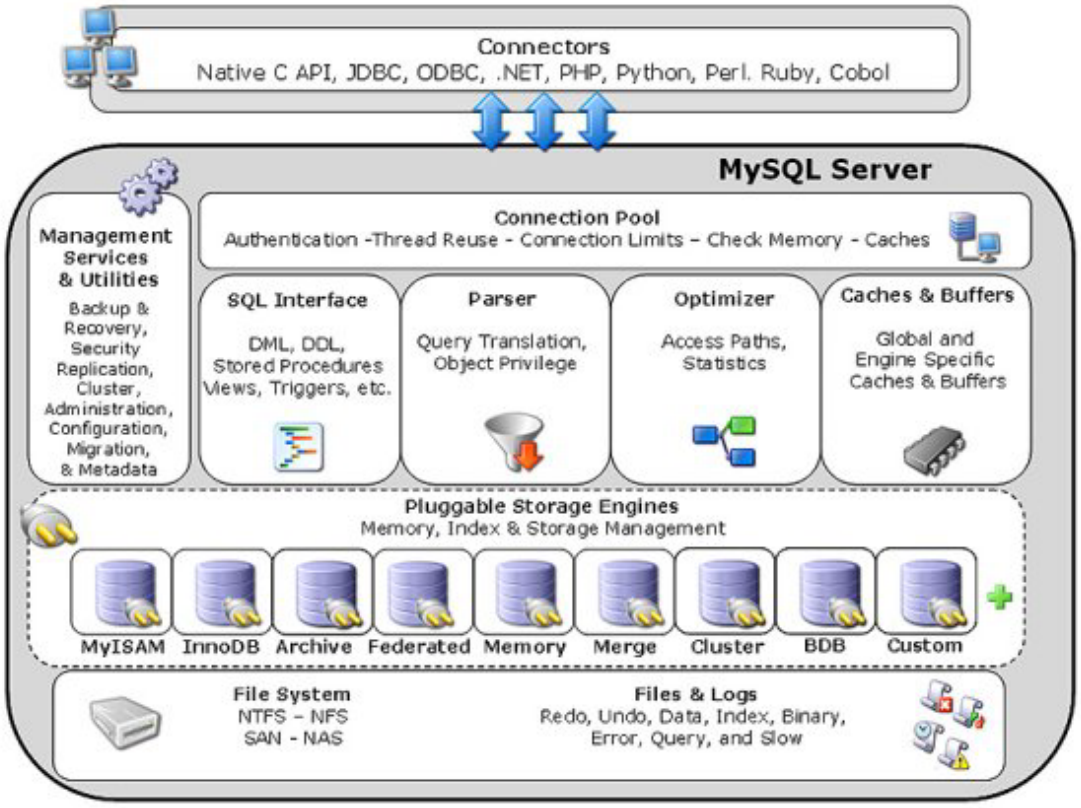
本章的意图是作为指南，用于帮助你为新的插件式存储引擎体系结构开发存储引擎。

关于MySQL插件式存储引擎体系结构的更多信息，请参见第14章：插件式存储引擎体系结构。

16.2. 概述

MySQL服务器采用了模块化风格。

图**16.1**：**MySQL**体系结构



存储引擎负责管理数据存储，以及MySQL的索引管理。通过定义的API，MySQL服务器能够与存储引擎进行通信。

每个存储引擎均是1个继承类，每个类实例作为处理程序而被引用。

针对需要与特殊表一起工作的每个线程，处理程序是在1个处理程序的基础上实例化的。例如，如果3个连接全都在相同的表上工作，需要创建3个处理程序实例。

一旦创建了处理程序实例，MySQL服务器将向处理程序发送命令，以便执行数据存储和检索任务，如打开表、操纵行和管理索引等。

能够以累进方式创建定制存储引擎：开发人员能够以只读存储引擎启动，随后添加对INSERT、UPDATE和DELETE操作的支持，甚至能够增加对索引功能、事务和其他高级操作的支持。

16.3. 创建存储引擎源文件

实施新存储引擎的最简单方法是，通过拷贝和更改EXAMPLE存储引擎开始。在MySQL 5.1源码树的sql/examples/目录下可找到文件ha_example.cc和ha_example.h。关于如何获得5.1源码树的说明，请参见2.8.3节，“从开发源码树安装”。

复制文件时，将名称从ha_example.cc和ha_example.h更改为与存储引擎相适应的名称，如ha_foo.cc和ha_foo.h。

拷贝并重命名了这些文件后，必须更换所有的EXAMPLE示例，以及具有存储引擎名称的示例。如果你熟悉sed，也能自动完成这些步骤：

```
sed s/EXAMPLE/FOO/g ha_example.h | sed s/example/foo/g ha_foo.h
```

```
sed s/EXAMPLE/FOO/g ha_example.cc | sed s/example/foo/g ha_foo.cc
```

16.4. 创建handlerton

handlerton（"单个处理程序"的简称）定义了存储引擎，并包含指向函数的函数指针，它以整体方式作用在引擎上，而函数工作在单独的处理程序实例中。在这类函数的一些示例中，包含用于处理注释和回滚的事务函数。

下面给出了一个来自EXAMPLE存储引擎的示例：

```
handlerton example_hton= {
  "EXAMPLE",
  SHOW_OPTION_YES,
  "Example storage engine",
  DB_TYPE_EXAMPLE_DB,
  NULL,    /* Initialize */
  0,       /* slot */
  0,       /* savepoint size. */
  NULL,    /* close_connection */
  NULL,    /* savepoint */
  NULL,    /* rollback to savepoint */
  NULL,    /* release savepoint */
  NULL,    /* commit */
  NULL,    /* rollback */
  NULL,    /* prepare */
  NULL,    /* recover */
  NULL,    /* commit_by_xid */
  NULL,    /* rollback_by_xid */
  NULL,    /* create_cursor_read_view */
  NULL,    /* set_cursor_read_view */
  NULL,    /* close_cursor_read_view */
  example_create_handler,    /* Create a new handler */
  NULL,    /* Drop a database */
  NULL,    /* Panic call */
  NULL,    /* Release temporary latches */
  NULL,    /* Update Statistics */
  NULL,    /* Start Consistent Snapshot */
  NULL,    /* Flush logs */
  NULL,    /* Show status */
  NULL,    /* Replication Report Sent Binlog */
  HTON_CAN_RECREATE
};
```

下面给出了来自handler.h的handlerton定义：

```
typedef struct
  {
    const char *name;
    SHOW_COMP_OPTION state;
    const char *comment;
    enum db_type db_type;
    bool (*init)();
    uint slot;
    uint savepoint_offset;
    int  (*close_connection)(THD *thd);
    int  (*savepoint_set)(THD *thd, void *sv);
    int  (*savepoint_rollback)(THD *thd, void *sv);
    int  (*savepoint_release)(THD *thd, void *sv);
    int  (*commit)(THD *thd, bool all);
    int  (*rollback)(THD *thd, bool all);
    int  (*prepare)(THD *thd, bool all);
    int  (*recover)(XID *xid_list, uint len);
    int  (*commit_by_xid)(XID *xid);
    int  (*rollback_by_xid)(XID *xid);
    void *(*create_cursor_read_view)();
    void (*set_cursor_read_view)(void *);
    void (*close_cursor_read_view)(void *);
    handler *(*create)(TABLE *table);
    void (*drop_database)(char* path);
    int (*panic)(enum ha_panic_function flag);
    int (*release_temporary_latches)(THD *thd);
    int (*update_statistics)();
    int (*start_consistent_snapshot)(THD *thd);
    bool (*flush_logs)();
    bool (*show_status)(THD *thd, stat_print_fn *print, enum ha_stat_type stat);
    int (*repl_report_sent_binlog)(THD *thd, char *log_file_name, my_off_t end_offset);
    uint32 flags;
  } handlerton;
```

共有30个handlerton元素，但只有少量元素是强制性的（明确地讲是前4个元素和第21个元素）。

1. 存储引擎的名称。这是创建表时将使用的名称（CREATE TABLE ... ENGINE = *FOO*;）。
2. 确定使用SHOW STORAGE ENGINES命令时是否列出存储引擎。
3. 存储引擎注释，对使用SHOW STORAGE ENGINES命令时显示的存储引擎的描述。

4. 在MySQL服务器内唯一识别存储引擎的整数。内置存储引擎使用的常数定义在handler.h文件中。作为创建常数的可选方法，可使用大于25的整数。

5. 指向存储引擎初始化程序的指针。仅当启动服务器时才调用该函数，以便在实例化处理程序之前，存储引擎类能执行必要的内务操作。

6. 插槽。保存每连接的信息时，每个存储引擎在thd中有自己的内存区域（实际上为指针）。它是作为thd->ha_data[*foo*_hton.slot]访问的。插槽编号在调用*foo*_init()后由MySQL初始化。

7. 保存点偏移。为了保存每个savepoint数据，为存储引擎提供了请求的大小（典型情况下为0）。

必须以静态方式初始化savepoint偏移，使其具有所有的内存大小，以便保存每个savepoint的信息。在*foo*_init之后，它被更改为savepoint存储区域的偏移，存储引擎不需要使用它。

8. 由事务性存储引擎使用，清理其存储段内分配的内存，和／或回滚任何未完成的事务。

9. 由事务性存储引擎选择性使用，创建savepoint（保存点），并将其保存到提供的内存中。

10.指向处理程序rollback_to_savepoint()函数的函数指针。它用于在事务期间返回savepoint。仅对支持保存点的存储引擎才会填充它。

11.指向处理程序release_savepoint()函数的函数指针。它用于在事务期间释放保存点的资源。仅对支持保存点的存储引擎才会填充它。

12.指向处理程序commit()函数的函数指针。它用于提交事务。仅对支持事务的存储引擎才会填充它。

13.指向处理程序rollback()函数的函数指针。它用于回滚交易。仅对支持事务的存储引擎才会填充它。

14.XA事务性存储引擎所需。为提交操作准备事务。将XID与事务关联起来。

15.XA事务性存储引擎所需。恢复由XID标识的事务。

16.XA事务性存储引擎所需。提交由XID标识的事务。

17.XA事务性存储引擎所需。回滚由XID标识的事务。

18.与服务器端光标一起使用，尚未实施。

19.与服务器端光标一起使用，尚未实施。

20.与服务器端光标一起使用，尚未实施。

21.*MANDATORY*：构造并返回处理程序实例。

22.撤销方案时，如果存储引擎需要执行特殊步骤时使用（如在使用表空间的存储引擎中使用）。

23.清理在服务器关闭和崩溃时调用的函数。

24.InnoDB特殊函数。

25.在启动SHOW STATUS时调用InnoDB特殊函数。

26.调用InnoDB特殊函数以开始连续读取。

27.调用它，指明应将日志刷新为可靠的存储。

28.在存储引擎上提供可被人员读取的状态信息。

29.InnoDB特殊函数用于复制。

30.Handlerton标志，通常与ALTER TABLE相关。可能的值定义于sql/handler.h文件中，并在此列出；

```
31.       #define HTON_NO_FLAGS                 0
32.       #define HTON_CLOSE_CURSORS_AT_COMMIT (1 << 0)
33.       #define HTON_ALTER_NOT_SUPPORTED     (1 << 1)
34.       #define HTON_CAN_RECREATE            (1 << 2)
35.       #define HTON_FLUSH_AFTER_RENAME      (1 << 3)
36.       #define HTON_NOT_USER_SELECTABLE     (1 << 4)
```

HTON_ALTER_NOT_SUPPORTED由FEDERATED存储引擎使用，用以指明存储引擎不接受AFTER TABLE语句。

HTON_FLUSH_AFTER_RENAME指明，重命名表后 ，必须调用FLUSH LOGS。

HTON_NOT_USER_SELECTABLE指明存储引擎不能由用户选择，而是用作系统存储引擎，如用于二进制日志的伪存储引擎。

16.5. 对处理程序进行实例化处理

调用存储引擎的第1个方法是调用新的处理程序实例。

在存储引擎源文件中定义handlerton之前，必须定义用于函数实例化的函数题头。下面给出了1个来自CSV引擎的示例：

```
static handler* tina_create_handler(TABLE *table);
```

正如你所见到的那样，函数接受指向处理程序准备管理的表的指针，并返回处理程序对象。

定义了函数题头后，用第21个handlerton元素中的函数指针命名函数，指明函数负责生成新的处理程序实例。

下面给出了MyISAM存储引擎的实例化函数示例：

```
static handler *myisam_create_handler(TABLE *table)
  {
    return new ha_myisam(table);
  }
```

该调用随后与存储引擎的构造程序一起工作。下面给出了来自FEDERATED存储引擎的1个示例：

```
ha_federated::ha_federated(TABLE *table_arg)
  :handler(&federated_hton, table_arg),
  mysql(0), stored_result(0), scan_flag(0),
  ref_length(sizeof(MYSQL_ROW_OFFSET)), current_position(0)
  {}
```

下面给出了来自EXAMPLE存储引擎的另一个示例：

```
ha_example::ha_example(TABLE *table_arg)
  :handler(&example_hton, table_arg)
  {}
```

FEDERATED示例中的附加元素是处理程序的额外初始化要素。所要求的最低实施是EXAMPLE示例中显示的handler()初始化。

16.6. 定义表扩展

就给定的表、数据和索引，要求存储引擎为MySQL服务器提供存储引擎所使用的扩展列表。

扩展应采用以Null终结的字符串数组形式。下面给出了CSV引擎使用的数组：

```
static const char *ha_tina_exts[] = {
  ".CSV",
  NullS
};
```

调用bas_ext()函数时返回该数组。

```
const char **ha_tina::bas_ext() const
{
  return ha_tina_exts;
}
```

通过提供扩展信息，你还能忽略DROP TABLE功能的实施，这是因为，通过关闭表并用你指定的扩展删除所有文件，MySQL服务器能实现该功能。

16.7. 创建表

一旦实例化了处理程序，所需的第1个操作很可能是创建表。

你的存储引擎必须实现create()虚拟函数：

```
virtual int create(const char *name, TABLE *form, HA_CREATE_INFO *info)=0;
```

该函数应创建所有必须的文件，然后关闭表。MySQL服务器将调用随后需打开的表。

*name参数是表的名称。*form参数是st_table结构，该结构定义了表并与MySQL服务器已创建的*tablename*.frm文件的内容匹配。在大多数情况下，存储引擎不需要更改*tablename*.frm文件，也没有支持该操作的预置功能。

*info参数是包含CREATE TABLE语句用于创建表所需信息的结构。该结构定义于handler.h文件中，并为了便于参考列于下面：

```
typedef struct st_ha_create_information
  {
    CHARSET_INFO *table_charset, *default_table_charset;
    LEX_STRING connect_string;
    const char *comment,*password;
    const char *data_file_name, *index_file_name;
    const char *alias;
    ulonglong max_rows,min_rows;
    ulonglong auto_increment_value;
```

```
    ulong table_options;
    ulong avg_row_length;
    ulong raid_chunksize;
    ulong used_fields;
    SQL_LIST merge_list;
    enum db_type db_type;
    enum row_type row_type;
    uint null_bits;                       /* NULL bits at start of record */
    uint options;                                                /* OR of HA_CREATE_ options */
    uint raid_type,raid_chunks;
    uint merge_insert_method;
    uint extra_size;                      /* length of extra data segment */
    bool table_existed;                                     /* 1 in create if table existed */
    bool frm_only;                        /* 1 if no ha_create_table() */
    bool varchar;                         /* 1 if table has a VARCHAR */
  } HA_CREATE_INFO;
```

基本的存储引擎能忽略*form和*info的内容，这是因为，真正所需的是创建存储引擎所使用的数据文件，以及对数据文件的可能初始化操作（假定存储文件是基于文件的）。

下面给出了来自CSV存储引擎的实施示例:

```
int ha_tina::create(const char *name, TABLE *table_arg,
  HA_CREATE_INFO *create_info)
  {
    char name_buff[FN_REFLEN];
    File create_file;
    DBUG_ENTER("ha_tina::create");

    if ((create_file= my_create(fn_format(name_buff, name, "", ".CSV",
          MY_REPLACE_EXT|MY_UNPACK_FILENAME),0,
          O_RDWR | O_TRUNC,MYF(MY_WME))) < 0)
    DBUG_RETURN(-1);

    my_close(create_file,MYF(0));

    DBUG_RETURN(0);
  }
```

在前面的例子中，CSV引擎未引用*table_arg或*create_info参数，而是简单地创建了所需的数据文件，关闭它们，并返回。

my_create和my_close函数是定义于src/include/my_sys.h文件中的助手函数。

16.8. 打开表

在表上执行任何读或写操作之前，MySQL服务器将调用open()方法打开表数据和索引文件（如果存在的话）。

```
int open(const char *name, int mode, int test_if_locked);
```

第1个参数是要打开的表的名称。第2个参数确定了要打开的文件或准备执行的操作。它们的值定义于handler.h中，并为了方便起见列在下面：

```
#define HA_OPEN_KEYFILE                 1
#define HA_OPEN_RNDFILE                 2
#define HA_GET_INDEX               4
#define HA_GET_INFO                  8     /* do a ha_info() after open */
#define HA_READ_ONLY               16    /* File opened as readonly */
#define HA_TRY_READ_ONLY         32    /* Try readonly if can't open with read and write */
#define HA_WAIT_IF_LOCKED        64      /* Wait if locked on open */
#define HA_ABORT_IF_LOCKED     128       /* skip if locked on open.*/
#define HA_BLOCK_LOCK              256   /* unlock when reading some records */
#define HA_OPEN_TEMPORARY        512
```

最后一个选项规定了是否要在打开表之前检查表上的锁定。

在典型情况下，存储引擎需要实施某种形式的共享访问控制，以防止在多线程环境下的文件损坏。关于如何实施文件锁定的示例，请参见sql/examples/ha_tina.cc的get_share()和free_share()方法。

16.9. 实施基本的表扫描功能

16.9.1. 实施store_lock()函数
16.9.2. 实施external_lock()函数
16.9.3. 实施rnd_init()函数
16.9.4. 实施info()函数
16.9.5. 实施extra()函数
16.9.6. 实施rnd_next()函数

最基本的存储引擎能实现只读表扫描功能。这类引擎可用于支持SQL日志查询、以及在MySQL之外填充的其他数据文件。

本节介绍的方法实施提供了创建更高级存储引擎的基础。

下面给出了在CSV引擎的9行表扫描过程中进行的方法调用：

```
ha_tina::store_lock
ha_tina::external_lock
ha_tina::info
ha_tina::rnd_init
ha_tina::extra - ENUM HA_EXTRA_CACHE   Cache record in HA_rrnd()
ha_tina::rnd_next
ha_tina::rnd_next
ha_tina::rnd_next
ha_tina::rnd_next
ha_tina::rnd_next
ha_tina::rnd_next
ha_tina::rnd_next
ha_tina::rnd_next
ha_tina::rnd_next
ha_tina::extra - ENUM HA_EXTRA_NO_CACHE   End cacheing of records (def)
ha_tina::external_lock
```

```
ha_tina::extra - ENUM HA_EXTRA_RESET   Reset database to after open
```

16.9.1. 实施store_lock()函数

在执行任何读取或写操作之前，调用store_lock()函数。

将锁定添加到表锁定处理程序之前（请参见thr_lock.c），mysqld将用请求的锁调用存储锁定。目前，存储锁定能将写锁定更改为读锁定（或其他锁定），忽略锁定（如果不打算使用MySQL锁定的话），或为很多表添加锁定（就像使用MERGE处理程序时作的那样）。

例如，Berkeley DB能将所有的WRITE锁定更改为TL_WRITE_ALLOW_WRITE（表示我们正在执行WRITES，但我们仍允许其他人员进行操作）。

释放锁定时，也将调用store_lock()，在这种情况下，通常不需做任何事。

在某些特殊情况下，MySQL可能会发送对TL_IGNORE的请求。这意味着我们正在请求与上次相同的锁定，这也应被忽略（当我们打开了表的某一部分时，如果其他人执行了表刷新操作，就会出现该情况，此时，mysqld将关闭并再次打开表，然后获取与上次相同的锁定）。我们打算在将来删除该特性。

可能的锁定类型定义于includes/thr_lock.h中，并列在下面：

```
enum thr_lock_type
{
        TL_IGNORE=-1,
                  TL_UNLOCK,                              /* UNLOCK ANY LOCK */
                  TL_READ,                                  /* Read lock */
                  TL_READ_WITH_SHARED_LOCKS,
        TL_READ_HIGH_PRIORITY,      /* High prior. than TL_WRITE. Allow concurrent insert */
        TL_READ_NO_INSERT,                /* READ, Don't allow concurrent insert */
        TL_WRITE_ALLOW_WRITE,                   /*   Write lock, but allow other threads to read
/ write. */
        TL_WRITE_ALLOW_READ,        /*       Write lock, but allow other threads to read / write.
*/
        TL_WRITE_CONCURRENT_INSERT, /* WRITE lock used by concurrent insert. */
        TL_WRITE_DELAYED,                 /* Write used by INSERT DELAYED.  Allows READ locks */
        TL_WRITE_LOW_PRIORITY,            /* WRITE lock that has lower priority than TL_READ */
        TL_WRITE,                         /* Normal WRITE lock */
        TL_WRITE_ONLY               /* Abort new lock request with an error */
};
```

实际的锁定处理因锁定实施的不同而不同，你可以选择某些请求的锁定类型或不选择任何锁定类型，并根据情况恰当地代入你自己的方法。下面给出了1个CSV存储引擎实施示例：

```
THR_LOCK_DATA **ha_tina::store_lock(THD *thd,
                                    THR_LOCK_DATA **to,
                                    enum thr_lock_type lock_type)
 {
   if (lock_type != TL_IGNORE && lock.type == TL_UNLOCK)
     lock.type=lock_type;
```

```
  *to++= &lock;

  return to;

}
```

16.9.2. 实施external_lock()函数

external_lock()函数是在事务开始时调用的，或发出LOCK TABLES语句时调用的，用于事务性存储引擎。

在sql/ha_innodb.cc和sql/ha_berkeley.cc文件中，可找到使用external_lock()的示例，但大多数存储引擎简单地返回0，就像EXAMPLE存储引擎那样：

```
int ha_example::external_lock(THD *thd, int lock_type)

{

  DBUG_ENTER("ha_example::external_lock");

  DBUG_RETURN(0);

}
```

16.9.3. 实施rnd_init()函数

在任何表扫描之前调用的函数是rnd_init()函数。函数rnd_init()用于为表扫描作准备，将计数器和指针复位为表的开始状态。

下述示例来自CSV存储引擎：

```
int ha_tina::rnd_init(bool scan)
  {
    DBUG_ENTER("ha_tina::rnd_init");

    current_position= next_position= 0;
    records= 0;
    chain_ptr= chain;

    DBUG_RETURN(0);
  }
```

16.9.4. 实施info()函数

执行表扫描操作之前，将调用info()函数，以便为优化程序提供额外信息。

优化程序所需的信息不是通过返回值给定的，你需填充存储引擎类的特定属性，当info()调用返回后，优化程序将读取存储引擎类。

除了供优化程序使用外，在调用info()函数期间，很多值集合还将用于SHOW TABLE STATUS语句。

在sql/handler.h中列出了完整的公共属性，下面给出了一些常见的属性：

```
ulonglong data_file_length;           /* Length off data file */

ulonglong max_data_file_length;       /* Length off data file */

ulonglong index_file_length;

ulonglong max_index_file_length;

ulonglong delete_length;              /* Free bytes */

ulonglong auto_increment_value;

ha_rows records;                      /* Records in table */
```

```
ha_rows deleted;                        /* Deleted records */

ulong raid_chunksize;

ulong mean_rec_length;          /* physical reclength */

time_t create_time;                     /* When table was created */

time_t check_time;

time_t update_time;
```

对于表扫描，最重要的属性是"records"，它指明了表中的记录数。当存储引擎指明表中有0或1行时，或有2行以上时，在这两种情况下，优化程序的执行方式不同。因此，当你在执行表扫描之前不清楚表中有多少行时，应返回大于等于2的值，这很重要（例如，数据是在外部填充的）。

16.9.5. 实施extra()函数

执行某些操作之前，应调用extra()函数，以便为存储引擎就如何执行特定操作予以提示。

额外调用中的提示实施不是强制性的，大多数存储引擎均返回0：

```
int ha_tina::extra(enum ha_extra_function operation)
 {
   DBUG_ENTER("ha_tina::extra");
   DBUG_RETURN(0);
 }
```

16.9.6. 实施rnd_next()函数

完成表的初始化操作后，MySQL服务器将调用处理程序的rnd_next()函数，每两个扫描行调用1次，直至满足了服务器的搜索条件或到达文件结尾为止，在后一种情况下，处理程序将返回HA_ERR_END_OF_FILE。

rnd_next()函数有一个名为*buf的单字节数组参数。对于*buf参数，必须按内部MySQL格式用表行的内容填充它。

服务器采用了三种数据格式：固定长度行，可变长度行，以及具有BLOB指针的可变长度行。对于每种格式，各列将按照它们由CREATE TABLE语句定义的顺序显示（表定义保存在.frm文件中，优化程序和处理程序均能从相同的源，即TABLE结构，访问表的元数据）。

每种格式以每列1比特的"NULL bitmap"开始。对于含6个列的表，其bitmap为1字节，对于含9～16列的表，其bitmap为2字节，依此类推。要想指明特定的值是NULL，应将该列NULL位设置为1。

当NULL bitmap逐个进入列后，每列将具有MySQL手册的"MySQL数据类型"一节中指定的大小。在服务器中，列的数据类型定义在sql/field.cc文件中。对于固定长度行格式，列将简单地逐个放置。对于可变长度行，VARCHAR列将被编码为1字节长，后跟字符串。对于具有BLOB列的可变长度行，每个blob由两部分表示：首先是表示BLOB实际大小的整数，然后是指向内存中BLOB的指针。

在任何表处理程序中从rnd_next()开始，可找到行转换（或"包装"）的示例。例如，在ha_tina.cc中，find_current_row()内的代码给出了使用TABLE结构（由表指向的）和字符串对象（命名缓冲）包装字符数据（来自CSV文件）的方法。将行写回磁盘需要反向转换，从内部格式解包。

下述示例来自CSV存储引擎：

```
int ha_tina::rnd_next(byte *buf)

 {

   DBUG_ENTER("ha_tina::rnd_next");


   statistic_increment(table->in_use->status_var.ha_read_rnd_next_count, &LOCK_status);
```

```
    current_position= next_position;
    if (!share->mapped_file)
      DBUG_RETURN(HA_ERR_END_OF_FILE);
    if (HA_ERR_END_OF_FILE == find_current_row(buf) )
      DBUG_RETURN(HA_ERR_END_OF_FILE);

    records++;
    DBUG_RETURN(0);
  }
```

对于从内部行格式到CSV行格式的转换，它是在find_current_row()函数中执行的。

```
int ha_tina::find_current_row(byte *buf)
 {
   byte *mapped_ptr= (byte *)share->mapped_file + current_position;
   byte *end_ptr;
   DBUG_ENTER("ha_tina::find_current_row");

   /* EOF should be counted as new line */
   if ((end_ptr=  find_eoln(share->mapped_file, current_position,
                            share->file_stat.st_size)) == 0)
     DBUG_RETURN(HA_ERR_END_OF_FILE);

   for (Field **field=table->field ; *field ; field++)
   {
     buffer.length(0);
     mapped_ptr++; // Increment past the first quote
     for(;mapped_ptr != end_ptr; mapped_ptr++)
     {
       // Need to convert line feeds!
       if (*mapped_ptr == '"' &&
           (((mapped_ptr[1] == ',') && (mapped_ptr[2] == '"')) ||
            (mapped_ptr == end_ptr -1 )))
       {
         mapped_ptr += 2; // Move past the , and the "
         break;
       }
       if (*mapped_ptr == '\\' && mapped_ptr != (end_ptr - 1))
       {
         mapped_ptr++;
```

```
          if (*mapped_ptr == 'r')
            buffer.append('\r');
          else if (*mapped_ptr == 'n' )
            buffer.append('\n');
          else if ((*mapped_ptr == '\\') || (*mapped_ptr == '"'))
            buffer.append(*mapped_ptr);
          else  /* This could only happed with an externally created file */
          {
            buffer.append('\\');
            buffer.append(*mapped_ptr);
          }
        }
        else
          buffer.append(*mapped_ptr);
      }
      (*field)->store(buffer.ptr(), buffer.length(), system_charset_info);
    }
    next_position= (end_ptr - share->mapped_file)+1;
    /* Maybe use \N for null? */
    memset(buf, 0, table->s->null_bytes); /* We do not implement nulls! */

    DBUG_RETURN(0);
  }
```

16.10. 关闭表

当MySQL服务器完成表操作时，它将调用close()方法关闭文件指针并释放任何其他资源。

对于使用共享访问方法的存储引擎（如CSV引擎和其他示例引擎中显示的方法），必须将它们自己从共享结构中删除：

```
int ha_tina::close(void)
 {
   DBUG_ENTER("ha_tina::close");
   DBUG_RETURN(free_share(share));
 }
```

对于使用其自己共享管理系统的存储引擎，应使用任何所需的方法，在它们的处理程序中，从已打开表的共享区删除处理程序实例。

16.11. 为存储引擎添加对INSERT的支持

一旦在你的存储引擎中有了读支持，下一个需要实施的特性是对INSERT语句的支持。有了INSERT支持，存储引

擎就能处理WORM（写一次，读多次）应用程序，如用于以后分析的日志和归档应用等。

所有的INSERT操作均是通过write_row()函数予以处理的：

```
int ha_foo::write_row(byte *buf)
```

*buf参数包含将要插入的行，采用内部MySQL格式。基本的存储引擎将简单地前进到数据文件末尾，并直接在末尾处添加缓冲的内容，这样就能使行读取变得简单，这是因为，你可以读取行并将其直接传递到rnd_next()函数的缓冲参数中。

写入行的进程与读取行的进程相反：从MySQL内部行格式获取数据，并将其写入数据文件。下述示例来自CSV存储引擎：

```
int ha_tina::write_row(byte * buf)
 {
   int size;
   DBUG_ENTER("ha_tina::write_row");

   statistic_increment(table->in_use->status_var.ha_write_count, &LOCK_status);

   if (table->timestamp_field_type & TIMESTAMP_AUTO_SET_ON_INSERT)
     table->timestamp_field->set_time();

   size= encode_quote(buf);

   if (my_write(share->data_file, buffer.ptr(), size, MYF(MY_WME | MY_NABP)))
     DBUG_RETURN(-1);

   if (get_mmap(share, 0) > 0)
     DBUG_RETURN(-1);
   DBUG_RETURN(0);
 }
```

前述示例中的两条注释包括，更新关于写入操作的表统计，以及在写入行之前设置时间戳。

16.12. 为存储引擎添加对UPDATE的支持

通过执行表扫描操作，在找到与UPDATE语句的WHERE子句匹配的行后，MySQL服务器将执行UPDATE语句，然后调用update_row()函数：

```
int ha_foo::update_row(const byte *old_data, byte *new_data)
```

*old_data参数包含更新前位于行中的数据，而*new_data参数包含行的新内容（采用MySQL内部行格式）。

更新的执行取决于行格式和存储实施方式。某些存储引擎将替换恰当位置的数据，而其他实施方案则会删除已有的行，并在数据文件末尾添加新行。

非事务性存储引擎通常会忽略*old_data参数的内容，仅处理*new_data缓冲。事务性存储引擎可能需要比较缓冲，以确定在上次回滚中出现了什么变化。

如果正在更新的表中包含时间戳列，对时间戳的更新将由update_row()调用管理。下述示例来自CSV引擎：

```
int ha_tina::update_row(const byte * old_data, byte * new_data)
 {
   int size;
   DBUG_ENTER("ha_tina::update_row");

   statistic_increment(table->in_use->status_var.ha_read_rnd_next_count,
                       &LOCK_status);

   if (table->timestamp_field_type & TIMESTAMP_AUTO_SET_ON_UPDATE)
     table->timestamp_field->set_time();

   size= encode_quote(new_data);

   if (chain_append())
     DBUG_RETURN(-1);

   if (my_write(share->data_file, buffer.ptr(), size, MYF(MY_WME | MY_NABP)))
     DBUG_RETURN(-1);
   DBUG_RETURN(0);
 }
```

请注意上例中的时间戳设置。

16.13. 为存储引擎添加对DELETE的支持

MySQL服务器采用了与INSERT语句相同的方法来执行DELETE语句：服务器使用rnd_next()函数跳到要删除的行，然后调用delete_row()函数删除行。

```
int ha_foo::delete_row(const byte *buf)
```

*buf参数包含要删除行的内容。对于大多数存储引擎，该参数可被忽略，但事务性存储引擎可能需要保存删除的数据，以供回滚操作使用。

下述示例来自CSV存储引擎：

```
int ha_tina::delete_row(const byte * buf)
 {
   DBUG_ENTER("ha_tina::delete_row");
   statistic_increment(table->in_use->status_var.ha_delete_count,
                       &LOCK_status);

   if (chain_append())
     DBUG_RETURN(-1);
```

```
  --records;

  DBUG_RETURN(0);
}
```

前述示例的步骤是更新delete_count统计，并记录计数。

16.14. API引用

16.14.1. bas_ext
16.14.2. close
16.14.3. create
16.14.4. delete_row
16.14.5. delete_table
16.14.6. external_lock
16.14.7. extra
16.14.8. info
16.14.9. open
16.14.10. rnd_init
16.14.11. rnd_next
16.14.12. store_lock
16.14.13. update_row
16.14.14. write_row

16.14.1. bas_ext

目的

定义存储引擎所使用的文件扩展。

概要

virtual const char ** **bas_ext** ();

;

描述

这是bas_ext方法。调用它，可为MySQL服务器提供存储引擎所使用的文件扩展列表。该列表将返回以Null终结的字符串数组。

通过提供扩展列表，在很多情况下，存储引擎能省略delete_table()函数，这是因为MySQL服务器将关闭所有对表的引用，并使用指定的扩展删除所有文件。

参数

该函数无参数。

返回值

返回值是存储引擎扩展的以Null终结的字符串数组。下面给出了CSV引擎的示例：

```
static const char *ha_tina_exts[] =
 {
   ".CSV",
   NullS
 };
```

用法

```
static const char *ha_tina_exts[] =
 {
   ".CSV",
   NullS
 };

const char **ha_tina::bas_ext() const
 {
   return ha_tina_exts;
 }
```

默认实施

```
static const char *ha_example_exts[] = {
   NullS
 };

const char **ha_example::bas_ext() const
 {
   return ha_example_exts;
 }
```

16.14.2. close

目的

关闭打开的表。

概要

virtual int **close** (*void*);

void ;

描述

这是close方法。

关闭表。这是释放任何已分配资源的恰当时机。

从sql_base.cc、sql_select.cc和table.cc调用它。在sql_select.cc中，它仅用于关闭临时表，或在将临时表转换为myisam表的过程中关闭表。关于sql_base.cc，请查看close_data_tables()。

参数

- void

返回值

无返回值。

用法

取自CSV引擎的示例:

```
int ha_example::close(void)
{
  DBUG_ENTER("ha_example::close");
  DBUG_RETURN(free_share(share));
}
```

16.14.3. create

目的

创建新表。

概要

virtual int **create** (*name*,
form,
info);

const char * *name* ;
TABLE * *form* ;
HA_CREATE_INFO * *info* ;

描述

这是create方法。

调用create()以创建表。变量名称为表的名称。调用create()时，不需要打开表。此外，由于已创建了.frm文件，不推荐调整create_info。

由ha_create_table()从handle.cc中调用。

参数

- name
- form
- info

返回值

无返回值。

用法

CSV搜索引擎示例:

```
int ha_tina::create(const char *name, TABLE *table_arg,
                    HA_CREATE_INFO *create_info)
{
  char name_buff[FN_REFLEN];
  File create_file;
```

```
  DBUG_ENTER("ha_tina::create");

  if ((create_file= my_create(fn_format(name_buff, name, "", ".CSV",
                                        MY_REPLACE_EXT|MY_UNPACK_FILENAME),0,
                              O_RDWR | O_TRUNC,MYF(MY_WME))) < 0)
    DBUG_RETURN(-1);

  my_close(create_file,MYF(0));

  DBUG_RETURN(0);
}
```

16.14.4. delete_row

目的

删除行。

概要

virtual int **delete_row** (*buf*);

const byte * *buf* ;

描述

这是delete_row方法。

*Buf*包含删除行的副本。调用了当前行后，服务器将立刻调用它（通过前一个rnd_next()或索引调用）。如果存在指向上一行的指针，或能够访问 主键，删除操作将更为容易。请记住，服务器不保证连续删除。可以使用ORDER BY子句。

在sql_acl.cc和sql_udf.cc中调用，以管理内部的表信息。在sql_delete.cc、sql_insert.cc和sql_select.cc中调用。在sql_select中，它用于删除副本，而在插入操作中，它用于REPLACE调用。

参数

- buf

返回值

无返回值。

用法

默认实施

```
{ return  HA_ERR_WRONG_COMMAND; }
```

16.14.5. delete_table

目的

用来自bas_ext()的扩展删除所有文件。

概要

virtual int **delete_table** (*name*);

const char * *name* ;

描述

这是delete_table方法。

用于删除表。调用delete_table()时，所有已打开的对该表的引用均将被关闭（并释放全局共享的引用）。变量名称为表名。此时，需要删除任何已创建的文件。

如果未实施它，将从handler.cc调用默认的delete_table()，并用bas_ext()返回的文件扩展删除所有文件。假定处理程序返回的扩展比文件实际使用的多。

由delete_table和ha_create_table()从handler.cc调用。如果为存储引擎指定了table_flag HA_DROP_BEFORE_CREATE，仅在创建过程中使用。

参数

- name: 表的基本名称

返回值

· 如果成功地从base_ext删除了至少1个文件而且未出现除ENOENT之外的错误，返回0。

· #: Error

用法

大多数存储引擎均会忽略该函数的实施。

16.14.6. external_lock

目的

为事务处理表锁定。

概要

virtual int **external_lock** (*thd*,
lock_type);

THD * *thd* ;
int *lock_type* ;

描述

这是external_lock方法。

在lock.cc中"用于mysql的锁定函数"一节，给出了关于该议题的额外注释，值的一读。

在表上创建锁定。如果实施了能处理事务的存储引擎，请查看ha_berkely.cc，以了解如何执行该操作的方法。否则，应考虑在此调用flock()。

由lock_external()和unlock_external()从lock.cc中调用。也能由copy_data_between_tables()从sql_table.cc中调用。

参数

- thd
- lock_type

返回值

无返回值。

默认实施

```
{ return 0; }
```

16.14.7. extra

目的

将提示从服务器传递给存储引擎。

概要

virtual int **extra** (*operation*);

enum ha_extra_function *operation* ;

描述

这是extra方法。

无论何时，当服务器希望将提示发送到存储引擎时，将调用extra()。MyISAM引擎实现了大多数提示。ha_innodb.cc给出了最详尽的提示列表。

参数

- operation

返回值

无返回值。

用法

默认实施

默认情况下，存储引擎倾向于不实施任何提示。

```
{ return 0; }
```

16.14.8. info

目的

提示存储引擎通报统计信息。

概要

virtual void **info** (*uint*);

uint ;

描述

这是info方法。

::info()用于将信息返回给优化程序。目前，该表处理程序未实施实际需要的大多数字段。SHOW也能利用该数据。注意，或许你打算在你的代码中包含下述内容"if (records > 2) records = 2"。原因在于，服务器仅优化具有一条记录的情形。如果在表扫描过程中，你不清楚记录的数目，最好将记录数设为2，以便能够返回尽可能多的所需记录。除了记录外，你或许还希望设置其他变量，包括：删除的记录，data_file_length，index_file_length，delete_length，check_time。更多信息，请参见handler.h中的公共变量。

在下述文件中调用：filesort.cc ha_heap.cc item_sum.cc opt_sum.cc sql_delete.cc sql_delete.cc sql_derived.cc sql_select.cc sql_select.cc sql_select.cc sql_select.cc sql_select.cc sql_show.cc sql_show.cc sql_show.cc sql_show.cc sql_table.cc sql_union.cc sql_update.cc

参数

- uint

返回值

无返回值。

用法

该示例取自CSV存储引擎：

```
void ha_tina::info(uint flag)
{
  DBUG_ENTER("ha_tina::info");
  /* This is a lie, but you don't want the optimizer to see zero or 1 */
  if (records < 2)
    records= 2;
  DBUG_VOID_RETURN;
}
```

16.14.9. open

目的

打开表。

概要

virtual int **open** (*name*,
mode,
test_if_locked);

const char * *name* ;
int *mode* ;
uint *test_if_locked* ;

描述

这是open方法。

用于打开表。名称是文件的名称。在需要打开表时打开它。例如，当请求在表上执行选择操作时（对于每一请求，表未打开并被关闭，对其进行高速缓冲处理）。

由handler::ha_open()从handler.cc中调用。通过调用ha_open()，然后调用处理程序相关的open()，服务器打开所有表。

对于处理程序对象，将作为初始化的一部分并在将其用于正常查询之前打开它（并非总在元数据变化之前）。如果打开了对象，在删除之前还将关闭它。

这是open方法。调用open以打开数据库表。

第1个参数是要打开的表的名称。第2个参数决定了要打开的文件或将要执行的操作。这类值定义于handler.h中，为了方便起见在此列出：

```
#define HA_OPEN_KEYFILE                 1
#define HA_OPEN_RNDFILE                 2
#define HA_GET_INDEX              4
#define HA_GET_INFO                 8     /* do a ha_info() after open */
#define HA_READ_ONLY              16    /* File opened as readonly */
#define HA_TRY_READ_ONLY        32    /* Try readonly if can't open with read and write */
#define HA_WAIT_IF_LOCKED       64      /* Wait if locked on open */
#define HA_ABORT_IF_LOCKED    128       /* skip if locked on open.*/
#define HA_BLOCK_LOCK             256   /* unlock when reading some records */
#define HA_OPEN_TEMPORARY       512
```

最后的选项规定了在打开表之前是否应检查表上的锁定。

典型情况下，存储引擎需要实现某种形式的共享访问控制，以防止多线程环境下的文件损坏。关于如何实现文件锁定的示例，请参见sql/examples/ha_tina.cc的get_share()和free_share()方法。

参数

- name
- mode
- test_if_locked

返回值

无返回值。

用法

该示例取自CSV存储引擎:

```
int ha_tina::open(const char *name, int mode, uint test_if_locked)
{
DBUG_ENTER("ha_tina::open");

if (!(share= get_share(name, table)))
DBUG_RETURN(1);
thr_lock_data_init(&share->lock,&lock,NULL);
ref_length=sizeof(off_t);

DBUG_RETURN(0);
}
```

16.14.10. rnd_init

目的

为表扫描功能初始化处理程序。

概要

virtual int **rnd_init** (*scan*);

bool *scan* ;

描述

这是rnd_init方法。

当系统希望存储引擎执行表扫描时，将调用rnd_init()。

与index_init()不同，rnd_init()可以调用两次，两次调用之间不使用rnd_end()（仅当scan=1时才有意义）。随后，第2次调用应准备好新的表扫描。例如，如果rnd_init分配了光标，第2次调用应将光标定位于表的开始部分，不需要撤销分配并再次分配。

从下述文件调用：filesort.cc, records.cc, sql_handler.cc, sql_select.cc, sql_table.cc, 和sql_update.cc。

参数

- scan

返回值

无返回值。

用法

该示例取自CSV存储引擎:

```
int ha_tina::rnd_init(bool scan)
{
  DBUG_ENTER("ha_tina::rnd_init");

  current_position= next_position= 0;
  records= 0;
  chain_ptr= chain;
  DBUG_RETURN(0);
}
```

16.14.11. rnd_next

目的

从表中读取下一行，并将其返回服务器。

概要

virtual int **rnd_next** (*buf*);

byte * *buf* ;

描述

这是rnd_next方法。

对于表扫描的每一行调用它。耗尽记录时，应返回HA_ERR_END_OF_FILE。用行信息填充buff。表的字段结构是以服务器能理解的方式将数据保存到buf中的键。

从下述文件调用：filesort.cc, records.cc, sql_handler.cc, sql_select.cc, sql_table.cc, 和sql_update.cc。

参数

- buf

返回值

无返回值。

用法

下述示例取自ARCHIVE存储引擎:

```
int ha_archive::rnd_next(byte *buf)
{
  int rc;
  DBUG_ENTER("ha_archive::rnd_next");

  if (share->crashed)
      DBUG_RETURN(HA_ERR_CRASHED_ON_USAGE);

  if (!scan_rows)
    DBUG_RETURN(HA_ERR_END_OF_FILE);
```

```
  scan_rows--;

  statistic_increment(table->in_use->status_var.ha_read_rnd_next_count,
                      &LOCK_status);
    current_position= gztell(archive);
  rc= get_row(archive, buf);

  if (rc != HA_ERR_END_OF_FILE)
    records++;

  DBUG_RETURN(rc);
}
```

16.14.12. store_lock

目的

创建和释放表锁定。

概要

virtual THR_LOCK_DATA ** **store_lock** (*thd*,
to,
lock_type);

THD * *thd* ;
THR_LOCK_DATA ** *to* ;
enum thr_lock_type *lock_type* ;

描述

这是store_lock方法。

下面介绍了关于handler::store_lock()的概念:

该语句决定了在表上需要何种锁定。对于updates/deletes/inserts，我们得到WRITE锁定；对于SELECT...，我们得到读锁定。

将锁定添加到表锁定处理程序之前（请参见thr_lock.c），mysqld将用请求的锁定调用存储锁定。目前，存储锁定能将写锁定更改为读锁定（或某些其他锁定），忽略锁定（如果不打算使用MySQL表锁定），或为很多表添加锁定（就像使用MERGE处理程序时那样）。

例如，Berkeley DB能够将所有的WRITE锁定更改为TL_WRITE_ALLOW_WRITE（表明正在执行WRITES操作，但我们仍允许其他人执行操作）。

释放锁定时，也将调用store_lock()。在这种情况下，通常不需要作任何事。

在某些特殊情况下，MySQL可能会发送对TL_IGNORE的请求。这意味着我们正在请求与上次相同的锁定，这也应被忽略（当我们打开了表的某一部分时，如果其他人执行了表刷新操作，就会出现该情况，此时，mysqld将关闭并再次打开表，然后获取与上次相同的锁定）。我们打算在将来删除该特性。

由get_lock_data()从lock.cc中调用。

参数

- thd
- to

- lock_type

返回值

无返回值。

用法

下述示例取自ARCHIVE存储引擎:

```
/*
  Below is an example of how to setup row level locking.
*/
THR_LOCK_DATA **ha_archive::store_lock(THD *thd,
                                       THR_LOCK_DATA **to,
                                       enum thr_lock_type lock_type)
{
  if (lock_type == TL_WRITE_DELAYED)
    delayed_insert= TRUE;
  else
    delayed_insert= FALSE;

  if (lock_type != TL_IGNORE && lock.type == TL_UNLOCK)
  {
    /*
      Here is where we get into the guts of a row level lock.
      If TL_UNLOCK is set
      If we are not doing a LOCK TABLE or DISCARD/IMPORT
      TABLESPACE, then allow multiple writers
    */

    if ((lock_type >= TL_WRITE_CONCURRENT_INSERT &&
         lock_type <= TL_WRITE) && !thd->in_lock_tables
        && !thd->tablespace_op)
      lock_type = TL_WRITE_ALLOW_WRITE;

    /*
      In queries of type INSERT INTO t1 SELECT ... FROM t2 ...
      MySQL would use the lock TL_READ_NO_INSERT on t2, and that
      would conflict with TL_WRITE_ALLOW_WRITE, blocking all inserts
      to t2. Convert the lock to a normal read lock to allow
      concurrent inserts to t2.
    */

    if (lock_type == TL_READ_NO_INSERT && !thd->in_lock_tables)
      lock_type = TL_READ;

    lock.type=lock_type;
  }

  *to++= &lock;

  return to;
}
```

16.14.13. update_row

目的

更新已有行的内容。

概要

virtual int **update_row** (*old_data*,
new_data);

const byte * *old_data* ;
byte * *new_data* ;

描述

这是update_row方法。

old_data将保存前一行的记录，而new_data将保存最新的数据。

如果使用了ORDER BY子句，服务器能够根据排序执行更新操作。不保证连续排序。

目前，new_data不会拥有已更新的auto_increament记录，或已更新的时间戳字段。你可以通过下述方式（例如）完成该操作：if (table->timestamp_field_type & TIMESTAMP_AUTO_SET_ON_UPDATE) table->timestamp_field->set_time(); if (table->next_number_field && record == table->record[0]) update_auto_increment();

从sql_select.cc, sql_acl.cc, sql_update.cc和sql_insert.cc调用。

参数

- old_data
- new_data

返回值

无返回值。

用法

默认实施

```
{ return  HA_ERR_WRONG_COMMAND; }
```

16.14.14. write_row

目的

为表添加新行。

概要

virtual int **write_row** (*buf*);

byte * *buf* ;

描述

这是write_row方法。

write_row()用于插入行。目前，如果出现大量加载，不会给出任何extra()提示。buf是数据的字节数组，大小为table->s->reclength。

可以使用字段信息从本地字节数组类型提取数据。例如:

for (Field **field=table->field ; *field ; field++) { ... }

BLOB必须特殊处理:

```
for (ptr= table->s->blob_field, end= ptr + table->s->blob_fields ; ptr != end ; ptr++)
  {
        char *data_ptr;
        uint32 size= ((Field_blob*)table->field[*ptr])->get_length();
        ((Field_blob*)table->field[*ptr])->get_ptr(&data_ptr);
        ...
  }
```

关于以字符串形式提取所有数据的示例，请参见ha_tina.cc。在ha_berkeley.cc中，对于ha_berkeley自己的本地存储类型，给出了一个通过"包装功能"完整保存它的例子。

请参见update_row()关于auto_increments和时间戳的注释。该情形也适用于write_row()。

从item_sum.cc、item_sum.cc、sql_acl.cc、sql_insert.cc、sql_insert.cc、sql_select.cc、sql_table.cc、sql_udf.cc、以及sql_update.cc调用。

参数

- 数据的buf字节数组

返回值

无返回值。

用法

默认实施

```
          { return  HA_ERR_WRONG_COMMAND; }
```

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第17章：MySQL簇

目录

17.1. MySQL簇概述
17.2. MySQL簇的基本概念
17.3. 多计算机的简单基础知识

- 17.3.1. 硬件、软件和联网
- 17.3.2. 安装
- 17.3.3. 配置
- 17.3.4. 首次启动
- 17.3.5. 加载示例数据并执行查询
- 17.3.6. 安全关闭和重启

17.4. MySQL簇的配置

- 17.4.1. 从源码创建MySQL簇
- 17.4.2. 安装软件
- 17.4.3. MySQL簇的快速测试设置
- 17.4.4. 配置文件

17.5. MySQL簇中的进程管理

- 17.5.1. 用于MySQL簇的MySQL服务器进程使用
- 17.5.2. **ndbd**，存储引擎节点进程
- 17.5.3. **ndb_mgmd**，“管理服务器”进程
- 17.5.4. **ndb_mgm**，“管理客户端”进程
- 17.5.5. 用于MySQL簇进程的命令选项

17.6. MySQL簇的管理

- 17.6.1. MySQL簇的启动阶段
- 17.6.2. “管理客户端”中的命令
- 17.6.3. MySQL簇中生成的事件报告
- 17.6.4. 单用户模式
- 17.6.5. MySQL簇的联机备份

17.7. 使用与MySQL簇的高速互连

- 17.7.1. 配置MySQL簇以使用SCI套接字
- 17.7.2. 理解簇互连的影响

17.8. MySQL簇的已知限制
17.9. MySQL簇发展的重要历程

- 17.9.1. MySQL 5.0中的MySQL簇变化
- 17.9.2. 关于MySQL簇的MySQL 5.1发展历程

17.10. MySQL簇常见问题解答
17.11. MySQL簇术语表

MySQL簇是MySQL适合于分布式计算环境的高实用、高冗余版本。它采用了NDB簇存储引擎，允许在1个簇中运行多个MySQL服务器。在MySQL 5.1二进制版本中、以及与最新的Linux版本兼容的RPM中提供了该存储引擎。（注意，要想获得MySQL簇的功能，必须安装mysql-server和mysql-max RPM）。

目前能够运行MySQL簇的操作系统有Linux、Mac OS X和Solaris。（一些用户通报成功地在FreeBSD上运行了MySQL簇，但MySQL AB公司尚未正式支持该特性）。我们正在努力，以便使MySQL簇能运行在MySQL支持的所有操作系统上，包括Windows，而且当支持新的平台时，将更新该页面。

本章介绍了正在进行的工作，其内容将随着MySQL簇的不断演化而变化。关于MySQL簇的更多信息，请访问MySQL AB公司的网站http://www.mysql.com/products/cluster/。

或许你也希望使用MySQL AB提供的两种额外资源：

- MySQL簇邮件列表。
- MySQL用户论坛上的簇主题区。

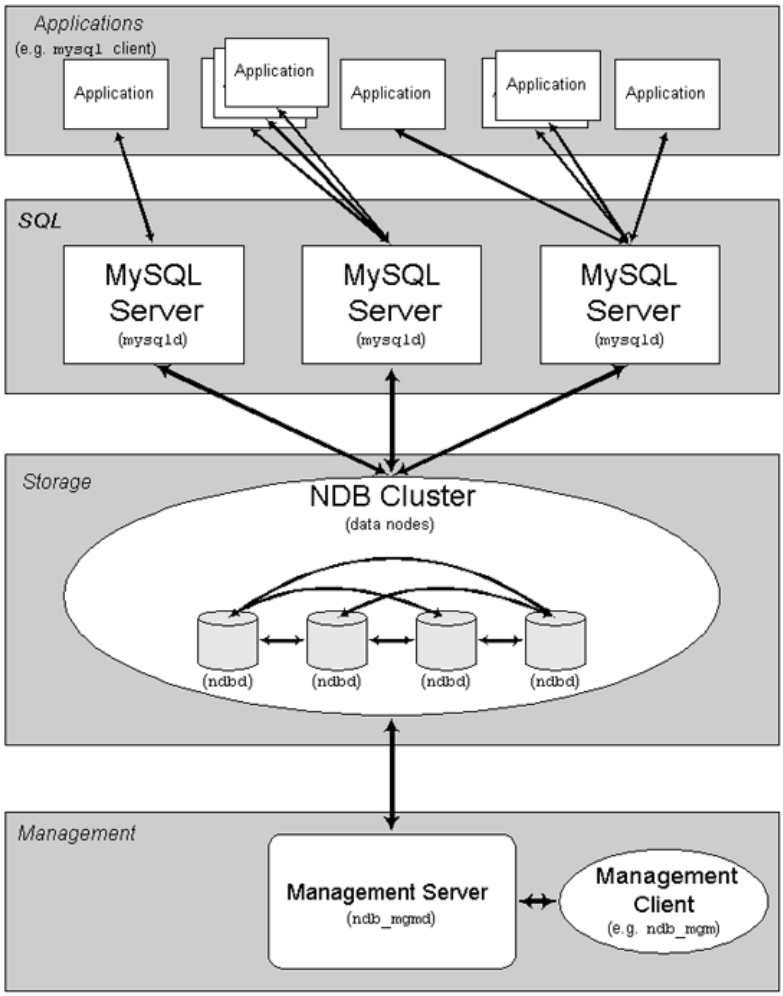
关于簇的一些常见问题，请参见17.10节，“MySQL簇常见问题解答”。如果你是MySQL簇的新手，请阅读我方开发人员区的文章如何为两个服务器设置MySQL簇，这会有所帮助。

17.1. MySQL簇概述

*MySQL*簇是一种技术，该技术允许在无共享的系统中部署“内存中”数据库的簇。通过无共享体系结构，系统能够使用廉价的硬件，而且对软硬件无特殊要求。此外，由于每个组件有自己的内存和磁盘，不存在单点故障。

MySQL簇将标准的MySQL服务器与名为NDB的"内存中"簇式存储引擎集成了起来。在我们的文档中，术语NDB指的是与存储引擎相关的设置部分，而术语"MySQL簇"指的是MySQL和NDB存储引擎的组合。

MySQL簇由一组计算机构成，每台计算机上均运行着多种进程，包括MySQL服务器，NDB簇的数据节点，管理服务器，以及（可能）专门的数据访问程序。关于簇中这些组件的关系，请参见下图：



所有这些程序一起构成了MySQL簇。将数据保存到NDB簇存储引擎中时，表将保存在数据节点内。能够从簇中所有其他MySQL服务器直接访问这些表。因此，在将数据保存在簇内的工资表应用程序中，如果某一应用程序更新了1位雇员的工资，所有查询该数据的其他MySQL服务器能立刻发现这种变化。

对于MySQL簇，保存在数据节点内的数据可被映射，簇能够处理单独数据节点的故障，除了少数事务将因事务状态丢失而被放弃外，不会产生其他影响。由于事务性应用程序能够处理事务失败事宜，因而它不是问题源。

通过将MySQL簇引入开放源码世界，MySQL为所有需要它的人员提供了具有高可用性、高性能和可缩放性的簇数据管理。

17.2. MySQL簇的基本概念

NDB是一种"内存中"存储引擎，它具有可用性高和数据一致性好的特点。

能够使用多种故障切换和负载平衡选项配置NDB存储引擎，但以簇层面上的存储引擎开始最简单。MySQL簇的NDB存储引擎包含完整的数据集，仅取决于簇本身内的其他数据。

下面，我们介绍了设置由NDB存储引擎和一些MySQL服务器构成的MySQL簇的设置方法。

目前，MySQL簇的簇部分可独立于MySQL服务器进行配置。在MySQL簇中，簇的每个部分被视为1个节点。

注释：在很多情况下，术语"节点"用于指计算机，但在讨论MySQL簇时，它表示的是进程。在单台计算机上可以有任意数目的节点，为此，我们采用术语簇主机。

有三类簇节点，在最低的MySQL簇配置中，至少有三个节点，这三类节点分别是：

- 管理(MGM)节点：这类节点的作用是管理MySQL簇内的其他节点，如提供配置数据、启动并停止节点、运行备份等。由于这类节点负责管理其他节点的配置，应在启动其他节点之前首先启动这类节点。MGM节点是用命令**ndb_mgmd**启动的。

- 数据节点：这类节点用于保存簇的数据。数据节点的数目与副本的数目相关，是片段的倍数。例如，对于两个副本，每个副本有两个片段，那么就有4个数据节点。没有必要有一个以上的副本。数据节点是用命令**ndbd**启动的。

· **SQL**节点：这是用来访问簇数据的节点。对于MySQL簇，客户端节点是使用NDB簇存储引擎的传统MySQL服务器。典型情况下，SQL节点是使用命令**mysqld –ndbcluster**启动的，或将ndbcluster添加到my.cnf后使用**mysqld**启动。

簇配置包括对簇中单独节点的配置，以及设置节点之间的单独通信链路。对于目前设计的MySQL簇，其意图在于，从处理器的能力、内存空间和带宽来讲，存储节点是同质的，此外，为了提供单一的配置点，作为整体，簇的所有配置数据均位于1个配置文件中。

管理服务器(MGM节点)负责管理簇配置文件和簇日志。簇中的每个节点从管理服务器检索配置数据，并请求确定管理服务器所在位置的方式。当数据节点内出现有趣的事件时，节点将关于这类事件的信息传输到管理服务器，然后，将这类信息写入簇日志。

此外，可以有任意数目的簇客户端进程或应用程序。它们分为两种类型：

· 标准**MySQL**客户端：对于MySQL簇，它们与标准的（非簇类）MySQL没有区别。换句话讲，能够从用PHP、Perl、C、C++、Java、Python、Ruby等编写的现有MySQL应用程序访问MySQL簇。

· 管理客户端：这类客户端与管理服务器相连，并提供了优雅地启动和停止节点、启动和停止消息跟踪（仅对调试版本）、显示节点版本和状态、启动和停止备份等的命令。

17.3. 多计算机的简单基础知识

17.3.1. 硬件、软件和联网
17.3.2. 安装
17.3.3. 配置
17.3.4. 首次启动
17.3.5. 加载示例数据并执行查询
17.3.6. 安全关闭和重启

本节介绍了如何规划、安装、配置和运行MySQL簇的基本知识。与17.4节，“MySQL簇的配置”中给出的示例不同，按照下面介绍的步骤和指南，所得的结果应是有用的MySQL簇，它满足对数据可用性和安全防护的最低要求。

在本节中，我们介绍了下述内容：硬件和软件要求，联网事宜，MySQL簇的安装，配置事宜，簇的启动、停止和重启，加载样本数据库，以及执行查询的方法。

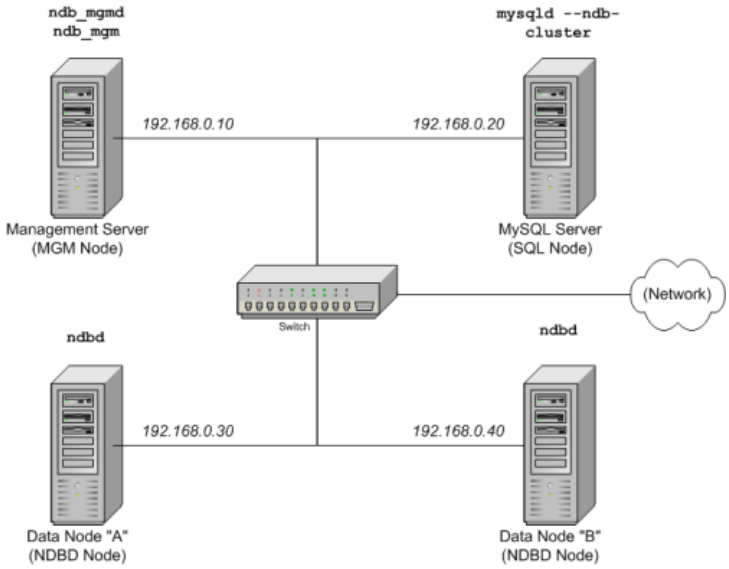
基本假定

本节作了如下假定：

1. 我们将建立具有4个节点的簇，每个节点位于不同的主机上，而且在典型的以太网中具有固定的网络地址，如下所述：

节点	**IP**地址
管理(MGM)节点	192.168.0.10
MySQL服务器(SQL)节点	192.168.0.20
数据(NDBD)节点"A"	192.168.0.30
数据(NDBD)节点"B"	192.168.0.40

2. 通过下图可更清楚的表明这点：



3.

4. 注释：出于简单性（以及可靠性）方面的考虑，在本基本知识介绍中我们仅使用数值IP地址。但是，如果在你的网络中具备DNS解析功能，在配置簇的过程中，可使用主机名代替IP地址。作为可选方式，也能使用/etc/hosts文件，或能提供主机查询的操作系统的等效物（如果可用的话）。

5. 在我们的场景中，每台主机均是基于Intel的桌面PC，PC上运行的是常见的一般性Linux版本，操作系统以标准配置安装在磁盘上，未运行任何不必要的服务。具备标准TCP/IP联网客户端的核心操作系统应足以符合我们的要求。此外，为了简单性，我们还假定所有主机上的文件系统是等同的。如果这些

主机上的文件系统不同，就需对这些说明作相应的调整。

6. 在每台机器上安装了标准的100 Mbps或1吉比特以太网卡，为每块网卡安装了恰当的驱动程序，并用标准的以太网联网装置（如交换器等）将4台主机连接起来（所有机器应使用具有相同容量的网卡，也就是说，簇中的所有4台机器应全部使用100M网卡，或全部使用1G网卡）。MySQL簇将工作在100 Mbps网络中，但吉比特以太网能提供更好的性能。

注意，MySQL簇不适合于连通性低于100 Mbps的网络。出于该原因（尤其是），在公共网络如Internet上运行MySQL簇很难成功，也不推荐这样做。

7. 对于样本数据，我们将使用世界数据库，该数据库可从MySQL AB公司的网站上下载。由于该数据库占用的空间相对较小，我们假定每台机器有256 MB RAM，这足以运行操作系统、主机NDB进程、以及存储数据库（对于数据节点）。

尽管在本基本介绍中采用的是Linux操作系统，但对这里给出的说明和步骤来说，仅过简单的修改，也能适用于Solaris或Mac OS X。此外，我们还假定你已掌握了安装和配置具备联网功能的操作系统的基本知识，或能够在需要的时候获得帮助。

下一节，我们更详细地讨论了MySQL簇的硬件、软件和联网要求。（请参见17.3.1节，“硬件、软件和联网”）。

17.3.1. 硬件、软件和联网

MySQL簇的一个强大优点在于，它能运行在普通硬件上，除了需要较大的RAM外在这点上没有特殊要求，这是因为实际的数据存储均是在内存中进行的。（注意，未来这点会改变，我们打算在未来的MySQL簇版本中实现基于磁盘的存储）。显然，多个CPU和更快的CPU能增强性能。对于簇进程来说，对内存的要求相对较少。

簇的软件要求程度适中。主机操作系统不需要任何特殊模块、服务、应用程序、或配置就能支持MySQL簇。对于Mac OS X或Solaris，标准安装就已足够。对于Linux，标准的“即开即用”安装应是所需的全部。MySQL软件要求很简单：MySQL-max 5.1的生产版就是所需的全部，要想获得簇支持，必须使用MySQL的-max版本。无需自己编译MySQL就能使用簇。在本节中，我们假定你使用了与Linux相适应的-max二进制版本。对于Solaris或Mac OS X操作系统，相应的部分可通过MySQL软件下载页面获得，http://dev.mysql.com/downloads/。

对于节点之间的通信，簇支持采用标准拓扑方案的TCP/IP联网，对于每台主机的预期最低要求是1块标准的100 Mbps以太网卡，对于作为整体的簇，还需加上交换器、网络集线器或路由器以提供网络连通性。我们强烈建议，应在其自己的子网内运行MySQL簇，不与非簇机器共享该子网，原因如下：

- 安全性：簇节点之间的通信未采用任何特殊加密或防护。对MySQL簇内传输的唯一保护方法是，在受保护的网络上运行簇。如果打算将MySQL簇用于Web应用，簇应明确地位于防火墙后面，而且不应位于网络的非军事区（DMZ）或其他地方。

- 效率：在专有的或受保护的网络上设置MySQL簇，这样，簇就能独享簇主机之间的带宽。为MySQL簇使用单独的交换器不仅能防止对簇数据的非法访问，而且还能确保簇节点不受网络上其他计算机之间信息传输的干扰。为了增强可靠性，可以使用双交换器和双卡，以防止网络出现单点故障，对于这类通信链路，很多设备驱动均支持故障切换功能。

也能与MySQL簇一起使用高速SCI（规模可扩展的计算机接口），但这不是要求的。关于该协议的更多信息，以及它与MySQL簇的用法，请参见17.7节，“使用与MySQL簇的高速互连”

17.3.2. 安装

对于每台运行存储或SQL节点的MySQL簇主机计算机，必须在其上安装MySQL-max二进制版本。对于管理节点，没有必要安装MySQL服务器二进制版本，但应安装MGM服务器端口监督程序和客户端二进制版本（分别是**ndb_mgmd**和**ndb_mgm**）。在本节中，我们介绍了为每种簇节点安装正确的二进制版本所需的步骤。

MySQL AB提供了预编译的二进制文件，它们支持簇，你不需要自己编译这些文件（如果你确实需要定制的二进制文件，请参见2.8.3节，“从开发源码树安装”）。因此，对于每台簇主机，安装进程的第一步是从MySQL下载区下载文件mysql-max-5.1.2-alpha-pc-linux-gnu-i686.tar.gz。我们假定你将该文件放在各机器的/var/tmp目录下。

对于32位和64位Linux平台，均有相应的RPM，RPM安装的-max二进制文件支持NDB簇存储引擎。如果你选择使用它们而不是二进制文件，务必在运行簇节点的所有机器上安装-server和-max软件包（关于使用RPM安装MySQL的更多信息，请参见2.4节，“在Linux下安装MySQL”）。使用RPM完成安装后，仍需对簇进行配置，请参见17.3.3节，“配置”。

注释：完成安装后，不启动任何二进制文件。配置完所有节点后，我们将向你介绍执行这类操作的方法。

存储节点和**SQL**节点安装

在设计为运行存储节点或SQL节点的三台机器的每一台上，以系统根用户身份执行下述步骤：

1. 检查你的/etc/passwd和/etc/group文件（或使用操作系统提供的用于管理用户和组的工具），查看在系统上是否已存在mysql组和mysql用户，这是因为某些操作系统会将其作为安装进程的一部分予以创建。如果它们不存在，创建新的mysql用户组，然后为该组添加1个mysql用户。

```
2.          groupadd mysql
3.          useradd -g mysql mysql
```

4. 进入包含下载文件的目录，解包档案文件，并创建与mysql-max可执行文件的symlink。注意，根据MySQL的版本号，实际的文件名和目录名会有所不同。

```
5.          cd /var/tmp
6.          tar -xzvf -C /usr/local/bin mysql-max-5.1.2-alpha-pc-linux-gnu-i686.tar.gz
7.          ln -s /usr/local/bin/mysql-max-5.1.2-alpha-pc-linux-gnu-i686 mysql
```

8. 进入mysql目录，运行所提供的用于创建系统数据库的脚本：

```
9.          cd mysql
```

```
10.       scripts/mysql_install_db --user=mysql
```

11.为MySQL服务器和数据目录设置必要的权限：

```
12.       chown -R root .
13.       chown -R mysql data
14.       chgrp -R mysql .
```

注意，在每台运行数据节点的机器上，数据目录是/usr/local/mysql/data。配置管理节点时将用到这类信息（请参见17.3.3节，“配置”）。

15.将MySQL启动脚本拷贝到恰当的目录下，使之成为可执行的脚本，并设置它以便在启动操作系统时启动：

```
16.       cp support-files/mysql.server /etc/rc.d/init.d/
17.       chmod +x /etc/rc.d/init.d/mysql.server
18.       chkconfig --add mysql.server
```

在此，我们使用Red Hat的**chkconfig**来创建与启动脚本的链接，请在你的操作系统上使用恰当的用于该目的的方式，如Debian上的**update-rc.d**。

请记住，对于存储节点或SQL节点所在的每台机器，必须分别指向上述步骤。

管理节点安装

对于MGM（管理）节点，不需要安装**mysqld**可执行文件，仅需安装用于MGM服务器和客户端的二进制文件，这类文件可在下载的-max档案中找到。再次假定你将该文件放在了/var/tmp目录下，引导系统时（也就是说使用**sudo, su root**或系统的等效命令后，假定具有系统管理员账户的权限），执行下述步骤，在簇管理节点主机上安装**ndb_mgmd**和**ndb_mgm**：

1. 即如/var/tmp目录，从档案文件中将**ndb_mgm**和**ndb_mgmd**提取到恰当的目录下，如/usr/local/bin：

```
2.          cd /var/tmp
3.          tar -zxvf mysql-max-5.1.2-alpha-pc-linux-gnu-i686.tar.gz /usr/local/bin '*/bin/ndb_mgm*'
```

4. 进入解包文件所在的目录，然后使这两个文件成为可执行的：

```
5.          cd /usr/local/bin
6.          chmod +x ndb_mgm*
```

在17.3.3节，“配置”中，我们将为示例簇中的所有节点创建和编写配置文件。

17.3.3. 配置

对于我们的4节点、4主机MySQL簇，需要编写4个配置文件，每个节点/主机1个。

- 每个数据节点或SQl节点需要1个my.cnf文件，该文件提供了两类信息：**connectstring**（连接字符串），用于通知节点到哪里找到MGM节点；以及一行，用于通知该主机（容纳数据节点的机器）上的MySQL服务器运行在NDB模式下。

关于连接字符串的更多信息，请参见17.4.4.2节，“MySQL簇连接字符串”。

- 管理节点需要config.ini文件，该文件通知节点有多少需要维护的副本，需要在每个数据节点上为数据和索引分配多少内存，数据节点的位置，在每个数据节点上保存数据的磁盘位置，以及SQL节点的位置。

配置存储节点和**SQL**节点

数据节点所需的my.cnf文件相当简单。配置文件应位于/etc目录下，并能用任何文本编辑器进行编辑（如有必要，创建该文件），例如：

```
vi /etc/my.cnf
```

对于本示例中的每个数据节点和SQL节点，my.cnf文件类似于：

```
# Options for mysqld process:
[MYSQLD]
ndbcluster                      # run NDB engine
ndb-connectstring=192.168.0.10  # location of MGM node

# Options for ndbd process:
[MYSQL_CLUSTER]
ndb-connectstring=192.168.0.10  # location of MGM node
```

输入上述内容后，保存文件并退出文本编辑器。在容纳数据节点“A”、数据节点“B”和SQL节点的机器上分别执行上述操作。

配置管理节点

配置MGM节点的第一步是创建目录，该目录用于存放配置文件，然后创建配置文件本身。例如（以根用户身份运行）：

```
mkdir /var/lib/mysql-cluster
cd /var/lib/mysql-cluster
```

```
vi config.ini
```

在此使用了vi来创建文件，不过，任何文本编辑器均应能胜任。

对于我们的典型设置，config.ini文件应类似于：

```
# Options affecting ndbd processes on all data nodes:
[NDBD DEFAULT]
NoOfReplicas=2    # Number of replicas
DataMemory=80M    # How much memory to allocate for data storage
IndexMemory=18M   # How much memory to allocate for index storage
                  # For DataMemory and IndexMemory, we have used the
                  # default values. Since the "world" database takes up
                  # only about 500KB, this should be more than enough for
                  # this example Cluster setup.

# TCP/IP options:
[TCP DEFAULT]
portnumber=2202   # This the default; however, you can use any
                  # port that is free for all the hosts in cluster
                  # Note: It is recommended beginning with MySQL 5.0 that
                  # you do not specify the portnumber at all and simply allow
                  # the default value to be used instead

# Management process options:
[NDB_MGMD]
hostname=192.168.0.10           # Hostname or IP address of MGM node
datadir=/var/lib/mysql-cluster  # Directory for MGM node logfiles

# Options for data node "A":
[NDBD]
                                # (one [NDBD] section per data node)
hostname=192.168.0.30           # Hostname or IP address
datadir=/usr/local/mysql/data   # Directory for this data node's datafiles

# Options for data node "B":
[NDBD]
hostname=192.168.0.40           # Hostname or IP address
datadir=/usr/local/mysql/data   # Directory for this data node's datafiles

# SQL node options:
[MYSQLD]
hostname=192.168.0.20           # Hostname or IP address
                                # (additional mysqld connections can be
                                # specified for this node for various
                                # purposes such as running ndb_restore)
```

（注释："world"数据库可从站点http://dev.mysql.com/doc/下载，它列在"示例"栏目下）。

一旦创建了所有的配置文件并指定了这些最低选项，可启动簇，并验证所有进程均能正常运行。关于这方面的讨论，请参见17.3.4节，"首次启动"。

关于可用MySQL簇配置参数以及其用法的更多信息，请参见17.4.4节，"配置文件"和17.4节，"MySQL簇的配置"。关于与进行备份有关的MySQL簇配置，请参见17.6.5.4节，"簇备份的配置"。

注释：簇管理节点的默认端口是1186，数据节点的默认端口2202。从MySQL 5.0.3开始，该限制已被放宽，簇能够根据空闲的端口自动地为数据节点分配端口。

17.3.4. 首次启动

完成配置后，启动簇并不很困难。必须在数据节点所在的主机上分别启动每个簇节点进程。尽管能够按任何顺序启动节点，但我们建议，应首先启动管理

节点，然后启动存储节点，最后启动SQL节点：

1. 在管理主机上，从系统shell发出下述命令以启动MGM节点进程：

2.
```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```

注意，必须用"-f"或"--config-file"选项，告诉**ndb_mgmd**到哪里找到配置文件（详情请参见17.5.3节，"**ndb_mgmd**，"管理服务器"进程"）。

3. 在每台数据节点主机上，对于首次启动，运行下述命令启动NDBD进程：

4.
```
shell> ndbd --initial
```

注意，仅应在首次启动**ndbd**时，或在备份／恢复或配置变化后重启**ndbd**时使用"--initial"参数，这很重要。原因在于，该参数会使节点删除由早期**ndbd**实例创建的、用于恢复的任何文件，包括恢复用日志文件。

5. 如果使用RPM在SQL节点所在的簇主机上安装了MySQL，能够（也应当）使用安装在/etc/init.d下的启动脚本在SQL节点上启动MySQL服务器进程。注意，要想运行"-max"服务器二进制文件，除了标准的RPM外，还需要安装-max服务器RPM。

如果一切顺利，并已正确设置了簇，那么簇现在应能运行。通过调用**ndb_mgm**管理节点客户端，可对其进行测试。其输出应类似于：

```
shell> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=2    @192.168.0.30  (Version: 5.1.2-alpha, Nodegroup: 0, Master)
id=3    @192.168.0.40  (Version: 5.1.2-alpha, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @192.168.0.10  (Version: 5.1.2-alpha)

[mysqld(SQL)]   1 node(s)
id=4   (Version: 5.1.2-alpha)
```

具体的输出内容可能会略有不同，这取决于你所使用的MySQL版本。

注释：如果你正在使用较早的MySQL版本，你或许会看到引用为'[mysqld(API)]'的SQL节点。这是一种早期的用法，现已放弃。

现在，应能在MySQL簇中处理数据库，表和数据。关于这方面的简要讨论，请参见17.3.5节，"加载示例数据并执行查询"。

17.3.5. 加载示例数据并执行查询

与没有使用簇的MySQL相比，在MySQL簇内操作数据的方式没有太大的区别。执行这类操作时应记住两点：

- 表必须用ENGINE=NDB或ENGINE=NDBCLUSTER选项创建，或用ALTER TABLE选项更改，以使用NDB簇存储引擎在簇内复制它们。如果使用**mysqldump**的输出从已有数据库导入表，可在文本编辑器中打开SQL脚本，并将该选项添加到任何表创建语句，或用这类选项之一替换任何已有的ENGINE（或TYPE）选项。例如，假定在另一个MySQL服务器（不支持MySQL簇）上有样本世界数据库，而且你打算导出城市表的定义：

-
```
shell> mysqldump --add-drop-table world City > city_table.sql
```

在所得的city_table.sql文件中，将包含这条表创建语句（以及导入表数据所需的INSERT语句）：

```
DROP TABLE IF EXISTS City;
CREATE TABLE City (
ID int(11) NOT NULL auto_increment,
Name char(35) NOT NULL default '',
CountryCode char(3) NOT NULL default '',
District char(20) NOT NULL default '',
Population int(11) NOT NULL default '0',
PRIMARY KEY  (ID)
) ENGINE=MyISAM;

INSERT INTO City VALUES (1,'Kabul','AFG','Kabol',1780000);
INSERT INTO City VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO City VALUES (3,'Herat','AFG','Herat',186800);
```

```
# (remaining INSERT statements omitted)
```

需要确认MySQL为该表使用了NDB存储引擎。有两种完成该任务的方法。其中一种方法是，在将表导入簇数据库之前更改其定义，使其类似于（仍使用"城市"作为示例）：

```
DROP TABLE IF EXISTS City;

CREATE TABLE City (

ID int(11) NOT NULL auto_increment,

Name char(35) NOT NULL default '',

CountryCode char(3) NOT NULL default '',

District char(20) NOT NULL default '',

Population int(11) NOT NULL default '0',

PRIMARY KEY  (ID)

) ENGINE=NDBCLUSTER;


INSERT INTO City VALUES (1,'Kabul','AFG','Kabol',1780000);

INSERT INTO City VALUES (2,'Qandahar','AFG','Qandahar',237500);

INSERT INTO City VALUES (3,'Herat','AFG','Herat',186800);

# (etc.)
```

对于将成为簇数据库组成部份的每个表，均需要为其定义执行上述操作。完成该任务的最简单方法是，简单地在world.sql文件上执行"查找-替换"，并用ENGINE=NDBCLUSTER替换所有的TYPE=MyISAM实例。如果你不打算更改该文件，也可使用ALTER TABLE。详情请参见下面的介绍。

假定你已在簇的SQL节点上创建了名为"world"的数据库，随后可使用**mysql**命令行客户端读取city_table.sql，并按通常方式创建和填充对应的表：

```
shell> mysql world < city_table.sql
```

请记住，上述命令必须在运行SQL节点的主机上执行，这点十分重要。对于本例，应在IP地址为**192.168.0.20**的机器上执行。

要想在SQL节点上创建世界数据库的副本，请将文件保存到/usr/local/mysql/data，然后运行：

```
shell> cd /usr/local/mysql/data

shell> mysql world < world.sql
```

当然，SQL脚本必须能被mysql系统用户读取。如果将文件保存到了不同的目录下，请作相应的调整。

注意，在MySQL 5.1中，NDB簇不支持自动发现数据库的功能，这点很重要（请参见17.8节，"MySQL簇的已知限制"）。这意味着，一旦在一个数据节点上创建了世界（world）数据库和它的表，在簇中的每个SQL节点上还需要发出命令**CREATE DATABASE world**（从MySQL 5.0.2开始，可以使用**CREATE SCHEMA world**取而代之），后跟**FLUSH TABLES**。这样，节点就能识别数据库并读取其表定义。

在SQL节点上运行**SELECT**查询与在MySQL服务器的任何其他实例上运行查询没有区别。要想从命令行运行查询，首先应按照通常方式登录到MySQL监视器：

```
shell> mysql -u root -p

Enter password:

Welcome to the MySQL monitor.  Commands end with ; or \g.

Your MySQL connection id is 1 to server version: 5.1.2-alpha


键入'help;'或'\h'获取帮助。键入'\c'清空缓冲区。


mysql>
```

如果在导入MySQL脚本之前未更改表定义中的ENGINE=子句，应在此时运行下述命令：

```
mysql> USE world;

mysql> ALTER TABLE City ENGINE=NDBCLUSTER;

mysql> ALTER TABLE Country ENGINE=NDBCLUSTER;

mysql> ALTER TABLE CountryLanguage ENGINE=NDBCLUSTER;
```

注意，在这里我们简单地使用了MySQL服务器密码为空的默认根用户账户。当然，在生产设置下，安装MySQL服务器时，总应遵守标准的安全方法措施，包括设置牢靠的根用户密码，并为用户创建具有完成任务所需的权限的用户账户。关于这方面的更多信息，请参见5.7节，"MySQL访问权限系统"。

需要关注的是，当簇节点彼此访问时不使用MySQL的权限系统，设置或更改MySQL用户账户（包括根用户账户）不影响节点之间的交互，它们仅对访问SQL节点的应用程序有效。

能够以通常的方式选择数据库，并对表执行SELECT查询，就像退出MySQL监视器一样：

```
mysql> USE world;

mysql> SELECT Name, Population FROM City ORDER BY Population DESC LIMIT 5;
```

```
+-----------+------------+
| 名称      | 人口 |
+-----------+------------+
| 孟买      |   10500000 |
| 汉城      |    9981619 |
| 圣保罗    |    9968485 |
| 上海      |    9696300 |
| 雅加达    |    9604900 |
+-----------+------------+
5 rows in set (0.34 sec)

mysql> \q
Bye

shell>
```

使用MySQL的应用程序能够使用标准的API。重要的是应记住，你的应用程序必须访问SQL节点，而不是MGM或存储节点。在下面的简单示例中，介绍了使用PHP 5的mysqli扩展（运行在位于网络中其他位置的Web服务器上）执行相同查询的方法：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1">
  <title>SIMPLE mysqli SELECT</title>
</head>
<body>
<?php
  # connect to SQL node:
  $link = new mysqli('192.168.0.20', 'root', '', 'world');
  # parameters for mysqli constructor are:
  #   host, user, password, database

  if( mysqli_connect_errno() )
    die("Connect failed: " . mysqli_connect_error());

  $query = "SELECT Name, Population
            FROM City
            ORDER BY Population DESC
            LIMIT 5";

  # if no errors...
  if( $result = $link->query($query) )
  {
?>
<table border="1" width="40%" cellpadding="4" cellspacing ="1">
  <tbody>
  <tr>
    <th width="10%">City</th>
    <th>Population</th>
  </tr>
<?
    # then display the results...
    while($row = $result->fetch_object())
      printf(<tr>\n  <td align=\"center\">%s</td><td>%d</td>\n</tr>\n",
```

```
                $row->Name, $row->Population);
?>
  </tbody
</table>
<?
  # ...and verify the number of rows that were retrieved
    printf("<p>Affected rows: %d</p>\n", $link->affected_rows);
  }
  else
    # otherwise, tell us what went wrong
    echo mysqli_error();

  # free the result set and the mysqli connection object
  $result->close();
  $link->close();
?>
</body>
</html>
```

我们假定运行在Web服务器上的进程能够访问SQL节点的IP地址。

采用类似的风格，可以使用MySQL C API、Perl-DBI、Python-mysql、或MySQL AB自己的连接器来执行数据定义和操控任务，就像正常使用MySQL那样。

- 另外还请记住，每个*NDB*表必须有一个主键。如果在创建表时用户未定义主键，NDB簇存储引擎将自动生成隐含的主键。（注释：该隐含 键也将占用空间，就像任何其他的表索引一样。由于没有足够的内存来容纳这些自动创建的键，出现问题并不罕见）。

17.3.6. 安全关闭和重启

要想关闭簇，可在MGM节点所在的机器上，在Shell中简单地输入下述命令：

```
shell> ndb_mgm -e shutdown
```

该命令将恰当地中止**ndb_mgm**、**ndb_mgmd**以及任何**ndbd**进程。使用**mysqladmin shutdown**或其他方法，可中止SQL节点。注意，这里的"-e"选项用于将命令从shell传递到**ndb_mgm**客户端。请参见4.3.1节，"在命令行上使用选项"。

要想重启簇，可简单地运行下述命令：

- 在管理主机上（本设置中为192.168.0.10）：
-
```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```
- 在每台数据节点主机上（192.168.0.30和192.168.0.40）：
-
```
shell> ndbd
```

请记住，正常重启NDBD节点时，不要用"--initial"选项调用该命令。

- 在SQL主机上（192.168.0.20）：
-
```
shell> mysqld &
```

关于创建簇备份的更多信息，请参见17.6.5.2节，"使用管理服务器创建备份"。

要想从备份中恢复簇，需要使用**ndb_restore**命令。请参见17.6.5.3节，"如何恢复簇备份"。

关于配置MySQL簇的更多信息，请参见17.4节，"MySQL簇的配置"。

17.4. MySQL簇的配置

17.4.1. 从源码创建MySQL簇
17.4.2. 安装软件
17.4.3. MySQL簇的快速测试设置
17.4.4. 配置文件

作为MySQL簇组成部份的MySQL服务器仅在一个方面上与正常的（非簇式）MySQL服务器不同，它采用了NDB簇存储引擎。该引擎也简单地称为NDB，这两个术语是同义词。

为了避免不必要的资源分配，默认情况下，在服务器的配置中将禁止NDB存储引擎。要想启用NDB，需要更改服务器的my.cnf配置文件，或使用"—ndbcluster"选项启动服务器。

由于MySQL服务器是簇的一部分，它也需要知道如何访问MGM节点，以便获得簇配置数据。默认行为是查找本地主机上MGM节点。但是，如果需要另外指定它的位置，可在my.cnf文件或MySQL服务器命令行上进行。能够使用NDB存储引擎之前，至少应有一个MGM节点是可操作的，而且还应有所需的数据节点。

17.4.1. 从源码创建MySQL簇

对于Linux、Mac OS X和Solaris，在其二进制分发版中均提供了NDB簇存储引擎。在Windows平台上尚不支持它，但我们打算在不远的将来使其能用于win32和其他平台。

如果选择从源码tarball或MySQL 5.1 BitKeeper树创建它，运行**configure**时，务必使用"--with-ndbcluster"选项。也可以使用**BUILD/compile-pentium-max**创建脚本。注意，该脚本包含OpenSSL，因此，要想成功创建，必须有或获得OpenSSL，如不然，需要更改"**compile-pentium-max**"以便将该要求排除在外，当然，也能采用标准步骤来编译你自己的二进制文件，然后执行常规测试和安装步骤。请参见2.8.3节，"从开发源码树安装"。

17.4.2. 安装软件

在下面的数节内，假定你已熟悉了MySQL的安装方法，在此，我们仅介绍了MySQL簇配置与不具备簇功能的MySQL配置之间的差别。如果希望了解关于后者的更多信息，请参见第2章：安装MySQL。

如果首先运行了所有的管理和数据节点，你将发现簇配置最简单，这或许是最花时间的配置部分。编辑my.cnf文件相对直接，在本节中，仅讨论与不具备簇功能的MySQL配置不同的部分。

17.4.3. MySQL簇的快速测试设置

为了帮助你熟悉基本概念，我们将介绍功能性MySQL簇的最简单的可能配置。然后，按照本章相关部分提供的信息，你应能设计自己所需的配置。

首先，应以系统根用户身份通过执行下述命令创建配置目录，如/var/lib/mysql-cluster：

```
shell> mkdir /var/lib/mysql-cluster
```

在该目录下，使用下述信息创建名为config.ini的文件，针对系统的情况，用恰当的值替换HostName和DataDir。

```
# file "config.ini" - showing minimal setup consisting of 1 data node,
# 1 management server, and 3 MySQL servers.
# The empty default sections are not required, and are shown only for
# the sake of completeness.
# Data nodes must provide a hostname but MySQL Servers are not required
# to do so.
# If you don't know the hostname for your machine, use localhost.
# The DataDir parameter also has a default value, but it is recommended to
# set it explicitly.
# Note: DB, API, and MGM are aliases for NDBD, MYSQLD, and NDB_MGMD
# respectively. DB and API are deprecated and should not be used in new
# installations.
[NDBD DEFAULT]
NoOfReplicas= 1

[MYSQLD DEFAULT]
[NDB_MGMD DEFAULT]
[TCP DEFAULT]

[NDB_MGMD]
HostName= myhost.example.com

[NDBD]
HostName= myhost.example.com
DataDir= /var/lib/mysql-cluster

[MYSQLD]
[MYSQLD]
[MYSQLD]
```

现在，能够按下述方式启动管理服务器：

```
shell> cd /var/lib/mysql-cluster
```

```
shell> ndb_mgmd
```

接下来，通过运行**ndbd**启动单个DB节点。首次为给定的DB节点启动**ndbd**时，应使用"—initial"选项，如下所示：

```
shell> ndbd --initial
```

对于后续的**ndbd**启动，通常不需要使用该选项：

```
shell> ndbd
```

这是因为，--initial选项将删除该数据节点的所有已有数据和日志文件（以及所有的表元数据），并创建新的数据和日志文件。该规则的一项例外是：添加了新数据节点后重启簇并从备份进行恢复之时。

默认情况下，**ndbd**将在端口1186上查找本地主机上的管理服务器。

注释：如果从二进制tarball安装了MySQL，需要明确指定**ndb_mgmd**和**ndbd**服务器的路径。（正常情况下，它们位于/usr/local/mysql/bin目录下）。

最后，进入MySQL数据目录（通常是/var/lib/mysql或/usr/local/mysql/data），并确保my.cnf文件包含启用NDB存储引擎所需的选项：

```
[mysqld]
ndbcluster
```

现在，你能按通常方式启动MySQL服务器：

```
shell> mysqld_safe --user=mysql &
```

等待一段时间，确认MySQL服务器正在恰当运行。如果发现通知用mysql停止，请检查服务器的.err文件，找出错误。

如果到目前为止一切正常，可使用簇启动它：

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.1.2-alpha-Max

键入'help;'或'\h'获取帮助。键入'\c'清空缓冲区。

mysql> SHOW ENGINES\G

...
*************************** 12. row ***************************
Engine: NDBCLUSTER
Support: YES
Comment: Clustered, fault-tolerant, memory-based tables
*************************** 13. row ***************************
Engine: NDB
Support: YES
Comment: Alias for NDBCLUSTER
...
```

（注意，上例输出中显示的行号可能与你的系统上显示的不同，具体情况取决于使用的MySQL版本，以及配置它的方式）。

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.1.2-alpha-Max

键入'help;'或'\h'获取帮助。键入'\c'清空缓冲区。

mysql> USE test;
Database changed

mysql> CREATE TABLE ctest (i INT) ENGINE=NDBCLUSTER;
Query OK, 0 rows affected (0.09 sec)

mysql> SHOW CREATE TABLE ctest \G
*************************** 1. row ***************************
       Table: ctest
```

```
Create Table: CREATE TABLE `ctest` (
  `i` int(11) default NULL
) ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

要想检查是否恰当设置了节点，可启动管理客户端：

```
shell> ndb_mgm
```

随后，为了获得关于簇状态的报告，可从管理客户端内使用**SHOW**命令：

```
NDB> SHOW
Cluster Configuration
---------------------
[ndbd(NDB)]     1 node(s)
id=2    @127.0.0.1  (Version: 3.5.3, Nodegroup: 0, Master)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @127.0.0.1  (Version: 3.5.3)

[mysqld(API)]   3 node(s)
id=3    @127.0.0.1  (Version: 3.5.3)
id=4 (not connected, accepting connect from any host)
id=5 (not connected, accepting connect from any host)
```

此时，你成功地设置了工作的MySQL簇。现在，你能使用由ENGINE=NDBCLUSTER或其别名ENGINE=NDB创建的表，将数据保存到簇中。

17.4.4. 配置文件

17.4.4.1. MySQL簇的配置示例
17.4.4.2. MySQL簇连接字符串
17.4.4.3. 定义构成MySQL簇的计算机
17.4.4.4. 定义MySQL簇管理服务器
17.4.4.5. 定义MySQL簇数据节点
17.4.4.6. 定义MySQL簇内的SQL节点
17.4.4.7. MySQL簇TCP/IP连接
17.4.4.8. 使用直接连接的MySQL簇TCP/IP连接
17.4.4.9. MySQL簇共享内存连接
17.4.4.10. MySQL簇SCI传输连接

配置MySQL簇需要与两个文件打交道：

- my.cnf：为所有的MySQL簇可执行文件指定了选项。你应熟悉了前面介绍的使用MySQL的方式，通过运行在簇中的每个可执行文件，必须能够访问该文件。

- config.ini：该文件仅由MySQL簇管理服务器读取，随后管理服务器会将包含该文件的信息分配给簇中的所有进程。config.ini文件包含对簇中各节点的描述。包括数据节点的配置参数，以及簇中所有节点间连接的配置参数。

我们正在不断改进簇配置，并努力简化该进程。尽管我们将尽量维护向后兼容性，但在某些时候，可能也需要引入不兼容的变动。在这种情况下，我们将尽量让簇用户事先了解该变动是否是向后兼容的。如果你发现了尚未记录在文档中的这类变动，请使用我们的缺陷数据库通报它。

17.4.4.1. MySQL簇的配置示例

为了支持MySQL簇，需要更新文件my.cnf，如下例所示。注意，不应将这里给出的选项与config.ini文件中出现的选项混淆起来。此外，从命令行调用可执行文件时，或许也应指定这些参数。

```
# my.cnf
# example additions to my.cnf for MySQL Cluster
# (valid in MySQL 5.1)

# enable ndbcluster storage engine, and provide connectstring for
# management server host (default port is 1186)
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com
```

```
# provide connectstring for management server host (default port: 1186)
[ndbd]
connect-string=ndb_mgmd.mysql.com

# provide connectstring for management server host (default port: 1186)
[ndb_mgm]
connect-string=ndb_mgmd.mysql.com

# provide location of cluster configuration file
[ndb_mgmd]
config-file=/etc/config.ini
```

（关于连接字符的更多信息，请参见17.4.4.2节，“MySQL簇连接字符串”）。

```
# my.cnf
# example additions to my.cnf for MySQL Cluster
# (will work on all versions)

# enable ndbcluster storage engine, and provide connectstring for management
# server host to the default port 1186
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

或许，你也可以使用簇my.cnf中单独的[mysql_cluster]部分，设置可被所有可执行文件读取的设置，并影响所有的可执行文件：

```
# cluster-specific settings
[mysql_cluster]
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

目前，配置文件采用的是INI格式，默认情况下被命名为config.ini。该文件在启动时由**ndb_mgmd**读取，并能被置于任何地方。在命令行上与**ndb_mgmd**一起使用--config-file=[*<path>*]*<filename>*，可指定其位置和名称。如果未指定配置文件，默认情况下，**ndb_mgmd**将尝试读取位于当前工作目录下的文件config.ini。

对于大多数参数，均定义了默认值，也能在config.ini文件中指定默认值。要想创建默认值部分，可简单地将单词DEFAULT添加到该部分的名称上。例如，数据节点是使用[NDBD]部分配置的。如果所有的数据节点使用相同大小的数据内存，而且该内存大小不同于默认的大小，应创建包含DataMemory行的[NDBD DEFAULT]部分，为所有数据节点指定默认的数据内存大小。

INI格式包含多个部分，每一部分以该部分的标题（用方括号括住）开始，后跟恰当的参数名和值。与标准格式的不同之处在于，不能用冒号“**:**”和等号“**=**”隔开参数名和值；另一处不同是，这些部分并不是用名称唯一标识的。其唯一性条目（如具有相同类型的两个不同节点）是由唯一ID标识的。

作为最低要求，配置文件必须定义簇中的计算机和节点，以及这些节点所在的计算机。下面给出了一个简单的簇配置文件示例，该簇包含1个管理服务器，2个数据节点和2个MySQL服务器：

```
# file "config.ini" - 2 data nodes and 2 SQL nodes
# This file is placed in the startup directory of ndb_mgmd (the management
# server)
# The first MySQL Server can be started from any host. The second can be started
# only on the host mysqld_5.mysql.com

[NDBD DEFAULT]
NoOfReplicas= 2
DataDir= /var/lib/mysql-cluster

[NDB_MGMD]
Hostname= ndb_mgmd.mysql.com
DataDir= /var/lib/mysql-cluster

[NDBD]
HostName= ndbd_2.mysql.com
```

```
[NDBD]
HostName= ndbd_3.mysql.com

[MYSQLD]
[MYSQLD]
HostName= mysqld_5.mysql.com
```

在该配置文件中，有6个不同部分：

- [COMPUTER]：定义了簇主机。
- [NDBD]：定义了簇的数据节点。
- [MYSQLD]：定义了簇的MySQL服务器节点。
- [MGM]或[NDB_MGMD]：定义了簇的管理服务器节点。
- [TCP]：定义了簇中节点间的TCP/IP连接，TCP/IP是默认的连接协议。
- [SHM]：定义了节点间的共享内存连接。以前，这类连接仅能在使用"--with-ndb-shm"选项创建的二进制文件中使用。在MySQL 5.1-Max中，默认情况下它是允许的，但仍应将其视为试验性的。

注意，每个节点在config.ini文件中有自己的部分。例如，由于该簇有两个数据节点，在配置文件中，也包含定义这些节点的部分。

可以为每个部分定义DEFAULT值。所有的簇参数名称均区分大小写。

17.4.4.2. MySQL簇连接字符串

除了MySQL簇管理服务器（**ndb_mgmd**），构成MySQL簇的每个节点均需要1个连接字符串，该连接字符串指向管理服务器所在的位置。它用于建立与管理服务器的连接，并执行其他任务，这类其他任务取决于节点在簇内扮演的角色。连接字符串的语法如下：

```
<connectstring> :=
    [<nodeid-specification>,]<host-specification>[,<host-specification>]

<nodeid-specification> := node_id

<host-specification> := host[:port]
```

node_id是大于1的整数，用于确定config.ini中的节点。*port*是引用正常Unix端口的整数。*host*是代表有效Internet地址的字符串。

```
example 1 (long):    "nodeid=2,myhost1:1100,myhost2:1100,192.168.0.3:1200"
example 2 (short):   "myhost1"
```

如果未提供，所有节点均将使用localhost:1186作为默认的连接字符串值。如果在连接字符串中省略了<port>，默认端口为1186。该端口在网络上总应是可用的，这是因为它是由IANA为该目的而指定的（详情请参见http://www.iana.org/assignments/port-numbers）。

通过列出多个<host-specification>值，能够指定数个冗余管理服务器。簇节点将按照指定的顺序尝试连接到每台主机上的连续管理服务器，直至成功建立起连接为止。

有多种指定连接字符串的不同方法：

- 每个可执行文件有自己的命令行选项，使用它，能够在启动时指定管理服务器（关于各可执行程序的介绍，请参见相应的文档）。
- 也能一次性地为簇中的所有节点设置连接字符串，方法是将其放在管理服务器的my.cnf文件的[mysql_cluster]部分。
- 为了向后兼容性，还提供了两种其他选项，其使用的语法相同：

1. 设置NDB_CONNECTSTRING环境变量，使之包含connectstring（连接字符串）。
2. 将针对各可执行文件的connectstring（连接字符串）写入名为Ndb.cfg的文本文件，并将该文件放在可执行文件的启动目录下。

但是，这些方法目前已不再受重视，对于新安装，不应使用它们。

指定连接字符串时，推荐的方法是在命令行上设置它，或为每个可执行文件在my.cnf文件中设置它。

17.4.4.3. 定义构成MySQL簇的计算机

除了用于避免为系统中的每个节点定义主机名外，[COMPUTER]部分没有实际的重要意义。这里所提到的所有参数都是需要的。

- [COMPUTER]Id

这是整数值，用于引用位于配置文件中别处的主机计算机。

- [COMPUTER]HostName

这是计算机的主机名或IP地址。

17.4.4.4. 定义MySQL簇管理服务器

[NDB_MGMD]部分（或其别名[MGM]）用于配置管理服务器的行为。下面列出的所有参数均能被忽略，如果是这样，将使用其默认值。注释：如果ExecuteOnComputer和HostName参数均未出现，会为它们指定默认值localhost。

· [NDB_MGMD]Id

簇中的每个节点都有唯一的标识，由从1到63的整数表示。所有的内部簇消息使用该ID来定址结点。

· [NDB_MGMD]ExecuteOnComputer

它引用在[COMPUTER]部分中定义的计算机之一。

· [NDB_MGMD]PortNumber

这是管理服务器用于监听配置请求和管理命令的端口号。

· [NDB_MGMD]LogDestination

该参数指定了将簇登录信息发送到哪里。有三种选项，CONSOLE、SYSLOG和FILE：

o CONSOLE，将日志输出到标准输出设备（stdout）：

o

```
CONSOLE
```

o SYSLOG，将日志发送到syslog（系统日志）软设备，可能的值包括：auth、authpriv、cron、daemon、ftp、kern、lpr、mail、news、syslog、user、uucp、local0、local1、local2、local3、local4、local5、local6或local7。

注释：并非所有的操作系统均支持所有的软设备。

```
SYSLOG:facility=syslog
```

o FILE，将簇日志输出导向相同机器上的正规文件。可指定下述值：

§ filename：日志文件的名称。

§ maxsize：日志记录切换到新文件之前，文件能增长到的最大尺寸。出现该情况时，将通过在文件名上添加.x，重命名日志文件，其中，x是该名称尚未使用的下一个数字。

§ maxfiles：日志文件的最大数目。

o

```
FILE:filename=cluster.log,maxsize=1000000,maxfiles=6
```

使用由分号分隔的字符串，可以指定多个日志目标，如下所示：

```
CONSOLE;SYSLOG:facility=local0;FILE:filename=/var/log/mgmd
```

FILE参数的默认值是FILE:filename=ndb_*node_id*_cluster.log,maxsize=1000000,maxfiles=6，其中，node_id是节点的ID。

· [NDB_MGMD]ArbitrationRank

该参数用于定义哪个节点将扮演仲裁程序的角色。只有MGM节点和SQL节点能扮演仲裁程序的角色。ArbitrationRank可以取下述值之一：

o 0：该节点永远不会用作仲裁程序。

o 1：该节点具有高的优先级，也就是说，与低优先级节点相比，它更容易成为仲裁程序。

o 2：表明节点具有低的优先级，仅当具有高优先级的节点无法用于该目的时，才能成为仲裁程序。

通常情况下，应将ArbitrationRank设置为1（默认值），并将所有的SQL节点设置为0，将管理服务器配置为仲裁程序。

· [NDB_MGMD]ArbitrationDelay

整数值，以毫秒为单位规定了管理服务器对仲裁请求的延迟时间。默认情况下，该值为0，通常不需要改变它。

· [NDB_MGMD]DataDir

它用于设置保存管理服务器输出文件的位置。这些文件包括簇日志文件、进程输出文件、以及端口监督程序的pid文件（对于日志文件，可通过设置[NDB_MGMD]LogDestination的FILE参数覆盖它，请参见本节前面的讨论）。

17.4.4.5. 定义MySQL簇数据节点

[NDBD]部分用于配置簇数据节点的行为。有很多可用于控制缓冲区大小、池大小、超时等的参数。强制性参数包括：

· ExecuteOnComputer或HostName.

· 参数NoOfReplicas

这些参数需要在[NDBD DEFAULT]部分中定义。

大多数数据节点参数是在[NDBD DEFAULT]部分中设置的。只有那些明确声明为能设置本地值的参数才能在[NDBD]部分中被更改。HostName、Id以及ExecuteOnComputer必须在本地[NDBD]部分中定义。

识别数据节点

启动节点时，可在命令行上分配Id值（即数据节点ID），也能在配置文件中分配Id值。

对于各参数，能够使用后缀k、M或G用于指明单位，分别表示1024、1024*1024或1024*1024*1024（例如，100k表示100 * 1024 = 102400）。目前，参数和值区分大小写。

· [NBDB]Id

这是用作节点地址的节点ID，供有的簇内部消息使用。这是介于1～63之间的整数。簇中的每个节点均有唯一的ID。

· [NDBD]ExecuteOnComputer

用于引用在COMPUTER部分中定义的计算机（主机）。

· [NDBD]HostName

指定该参数的效果类似于指定ExecuteOnComputer。它定义了存储节点所在计算机的主机名。指定除localhost之外的其他主机名时，需要该参数或ExecuteOnComputer。

· (*OBSOLETE*) [NDBD]ServerPort

簇中的各节点使用端口来与其他节点相连。该端口也用于连接建立阶段中的非TCP传输器。由于默认端口是动态分配的，同一台计算机上的两个节点具有不同的端口号，正常情况下不需要为该参数指定值。

· [NDBD]NoOfReplicas

该全局参数仅能在[NDBD DEFAULT]中设置，它定义了簇中每个表保存的副本数。该参数还指定了节点组的大小。节点组指的是保存相同信息的节点集合。

节点组是以隐式方式构成的。第1个节点组由具有最低节点ID的数据节点集合构成，下一个节点组由具有次低节点ID的数据节点集合构成，依此类推。作为示例，截顶我们有4个数据节点，并将NoOfReplicas设置为2。这四个数据节点的ID分别是2、3、4、5。那么第1个节点组由节点2和3构成，第2个节点组由节点4和5构成。重要的是对簇进行相应的配置，使得同一节点组中的节点位于不同的计算机上，这是因为，如果位于相同的计算机上，单个硬件故障会导致整个簇崩溃。

如果未提供节点ID，那么数据节点的顺序将是节点组的决定因素。无论是否进行了明确的分配，可在管理客户端SHOW命令的输出中查看它们。

NoOfReplicas没有默认值，最大的可能值为4。

· [NDBD]DataDir

该参数指定了存放跟踪文件、日志文件、pid文件以及错误日志的目录。

· [NDBD]FileSystemPath

该参数指定了存放为元数据创建的所有文件、REDO日志、UNDO日志和数据文件的目录。默认目录是由DataDir指定的。注意，启动**ndbd**进程之前，该目录必须已存在。

为MySQl簇推荐的目录层次包括/var/lib/mysql-cluster，在其下为节点的文件系统创建1个为目录。该子目录包含节点ID。例如，如果节点ID为2，该子目录的名称为ndb_2_fs。

· [NDBD]BackupDataDir

也能指定存放备份的目录。默认情况下，该目录是*FileSystemPath/*BACKUP（请参见前面的介绍）。

数据内存和索引内存

参数DataMemory和IndexMemory指定了存放实际记录及其索引的内存段的大小。这是它们的值时，重要的是应掌握使用DataMemory和IndexMemory的方式，这是因为，为了反映簇的实际使用情况，常常需要更新它们:

· [NDBD]DataMemory

该参数定义了用于保存数据库记录的空间大小。全部空间均是分配在内存中的，因此，机器应具有足够的物理内存来容纳该值，这点极其重要。

由DataMemory分配的内存用于保存实际记录和索引。目前，每条记录具有固定的大小（甚至VARCHAR列也保存为固定宽度列）。每条记录的开销为16字节，此外，每条记录还需要额外的空间，这是因为，这类记录保存在具有128字节页面开销的32KB页中（请参见下面的介绍）。由于每条记录仅保存在1个页中，因而每页有少量的浪费。目前，最大记录大小为8052字节。

由DataMemory定义的内存空间也用于保存有序索引，对于每条记录，索引约使用10字节。在有序索引中，表示了每个表行。用户常犯的一个错误是，想当然地认为所有的索引均保存在由IndexMemory分配的内存中，但情况并非如此：只有主键和唯一性混编索引使用该内存，有序索引使用的是由DataMemory分配的内存。然而，创建主键或唯一性混编索引时，也会在相同的 键上创建有序索引，除非在索引创建语句中指定了USING HASH。通过在管理客户端中运行**ndb_desc -d** *db_name table_name*，可对其进行验证。

DataMemory分配的内存空间由多个32KB页构成，它们是为表片段分配的。通常情况下，为每一表划分的表片段数目与簇中的节点数目相同。因此，对于每一节点，片段数目与在NoOfReplicas中设置的相同。一旦分配了1页，目前无法将其返回到自由页池中，除非删除表。执行节点恢复也将压缩分区，这是因为，所有记录均会被插入到其他活动节点的空分区中。

DataMemory内存空间也包含UNDO信息：对于每一更新，未改变记录的副本将被分配到DataMemory中。在有序表索引中，还有对每一副本的引用。仅当更新唯一性索引列时，才会更新唯一性混编索引，在该情况下，将在索引表中插入新的条目，并在提交时删除旧的条目。因此，也有必要分配足够的内存，以便处理由使用簇的应用程序执行的最大事务。在任何情况下，执行少量大的事务并不比使用众多小的事务占优，原因如下：

- 大事务的速度没有较小事务的速度快。
- 大的事务会增加丢失操作的数目，一旦事务失败，必须重复执行。
- 大的事务使用更多的内存。

DataMemory的默认值是80MB，最小为1MB。没有最大尺寸限制，但在实际使用过程中，最大限制应恰当，以便当达到最大限制时，进程不会启动交换功能。该限制由机器上可用的物理RAM量、以及操作系统能提交给任何进程的内存量决定。对于32位操作系统，该限制值为每进程2～4GB，对于64位操作系统，该限制值更大。对于大的数据库，出于该原因，最好使用64位操作系统。此外，在每台机器上也能运行一个以上的**ndbd**进程，在使用多CPU的机器上，该特性颇具优势。

- [NDBD]IndexMemory

该参数用于控制MySQL簇中哈希（混编）索引所使用的存储量。哈希（混编）索引总用于主键索引、唯一性索引、以及唯一性约束。注意，定义主键和唯一性索引时，将创建两条索引，其中一条是用于所有tuple访问和锁定处理的哈希（混编）索引。此外，它还能用于增强唯一性约束。

哈希（混编）索引的大小是每记录25字节，再加上主键的大小。对大于32字节的主键，还需加上8字节。

考虑下例定义的表：

```
CREATE TABLE example (
  a INT NOT NULL,
  b INT NOT NULL,
  c INT NOT NULL,
  PRIMARY KEY(a),
  UNIQUE(b)
) ENGINE=NDBCLUSTER;
```

有12字节的开销（无可空列将节省4字节的开销）加上每记录12字节的数据。此外，在列a和b上有两个有序索引，假定每记录分别耗用约10字节的空间。在每记录约使用29字节的基表上有1条主键哈希索引。唯一性约束由以b作为主键以及a作为列的单独表实现。对于该表，每记录将耗用额外的29字节索引内存，在示例表中，还包括12字节的开销再加上8字节的记录数据。

因此，对于100万条记录，需要58MB的索引内存来处理用于主键和唯一性约束的哈希索引。还需要64 MB来处理基表和唯一索引表、以及两个有序索引表的记录。

由此可见，哈希索引占用了相当大的内存空间，但作为回报，它们提供了对数据的极快访问。在MySQl簇中，它们也用于处理唯一性约束。

目前仅有的分区算法是散列法，有序索引对每个节点来说都是局部性的。因此，有序索引不能用于处理一般情况下的唯一性约束。

对于IndexMemory和DataMemory，重要的是，总的数据库大小是各节点组的所有数据内存和所有索引内存之和。每个节点组用于保存复制信息，因此，如果有4个节点和2个副本，将有2个节点组。对于每个数据节点，可用的总数据内存是2*DataMemory。

强烈建议为所有的节点设置相同的DataMemory值和IndexMemory值。由于数据是平均分布在簇中的所有节点上，任何节点可用的最大空间不超过簇中最小节点的可用空间。

DataMemory和IndexMemory可被更改，但降低任何一个的值均会导致危险，如果这样做，很容易使某一节点甚至整个簇因缺少足够的内存空间而无法重启。增加它们的值应是可接受的，但建议采用与软件升级相同的方式升级它，首先更新配置文件，然后重启管理服务器，最后依次重启每个数据节点。

更新不会增加所用的索引内存。插入将立刻生效，但是，在提交事务之前并不会实际删除行。

IndexMemory的默认值是18MB。最小值为1MB。

事务参数

下面讨论的三个参数十分重要，这是因为，它们会影响并发事务的数目，以及系统能够处理的事务的大小。MaxNoOfConcurrentTransactions用于设置节点内可能的并发事务数目。MaxNoOfConcurrentOperations用于设置能同时出现在更新阶段或同时锁定的记录数目。

对于打算设定特定值、不使用默认值的用户，这两个参数可能正是他们所需的（尤其是MaxNoOfConcurrentOperations）。默认值是为使用小型事务的系统而设置的，为的是确保这类事务不会使用过多的内存。

- [NDBD]MaxNoOfConcurrentTransactions

对于簇中的每个活动事务，必须在簇节点之一中有1条记录。对事务的协调任务是在各节点间进行的：在簇中，事务记录的总数等于任意给定节点中的事务数乘以簇中的节点数。

事务记录被分配给单独的MySQL服务器。正常情况下，对于使用簇中任何表的每个连接，必须为其分配至少1条事务记录。出于该原因，应确保簇中的事务记录数大于簇中所有MySQL服务器的并发连接数。

对于所有的簇节点，必须将该参数设置为相同的值。

更改该参数不安全，如果这样做，会导致簇崩溃。当某一节点崩溃时，簇中的一个节点（实际上是生存时间最久的节点）将为崩溃之时正在崩溃节点中运行的所有事务建立事务状态。因此，重要的是，该节点的事务记录数不低于失效节点中的事务记录数。

该参数的默认值为4096.

· [NDBD]MaxNoOfConcurrentOperations

根据事务的大小和数目调整该参数的值，这个想法不错。执行仅包含少量操作且不涉及很多记录的事务时，不需要将该参数设置得很高。但在执行涉及大量记录的大事务时，需要将该参数设置得较高。

对于每次事务更新的簇数据，均会保存记录，并会将它们保存在事务协调器中以及执行实际更新的节点中。这些记录包含所需的状态信息，这类信息可用于为回滚操作找到UNDO记录，用于锁定查询或其他目的。

该参数应被设置为：事务中同时更新的记录数除以簇数据节点的数目。例如，在包含4个数据节点的簇中，如果预期处理的、使用事务的并发更新数为1000000，就应将该值设置为1000000 / 4 = 250000。

设置锁定的读请求也会导致操作记录的创建。在单独节点内也会分配一些额外的空间，以便处理在节点间分配不完美的问题。

当查询使用唯一性哈希索引时，对于事务中的每条记录，实际上将使用两条操作记录。第1条记录代表在索引表中的读，第2条记录负责处理基表上的操作。

该参数的默认值为32768.

该参数实际上处理的是能分别配置的两个值。第1个值指定了将多少操作记录放到事务协调器中，第2个值指定了多少操作记录是数据库的本地记录。

对于在8节点簇上执行的特大事务，它要求事务协调器中的操作记录数不少于事务中涉及的读取、更新和删除次数。然而，簇中的操作记录分布在所有的8个节点上。因此，如果有必要为特大事务配置系统，良好的方法是分别配置该参数的两个部分。MaxNoOfConcurrentOperations总会被用于计算节点的事务协调器部分中的操作记录数。

应了解操作记录对内存的要求，这点也很重要。每记录约消耗1KB。

· [NDBD]MaxNoOfLocalOperations

默认情况下，将按照1.1 * MaxNoOfConcurrentOperations计算该参数，它适合于具有很多并发事务但不存在特大事务的系统。如果需要在某一时间处理特大事务而且有很多节点，最好通过明确指定该参数以覆盖默认值。

事务临时存储

下一组参数用于决定执行作为簇事务组成部分的查询时所需的临时存储空间。查询完成后将释放所有记录，簇将等待提交或回滚事件。

对于大多数情况，这些参数的默认值是恰当的。但是，如果需要支持涉及大量行或操作的事务，用户或许应增大这些参数的值，以便在系统中获得更好的平行性。对于需要相对较少事务的应用程序，用户可降低这些参数的值，以便节省内存。

· [NDBD]MaxNoOfConcurrentIndexOperations

对于使用唯一性哈希索引的查询，在查询执行期间，将使用操作记录的另一个临时集合。该参数用于设置记录池的大小。因此，仅当执行查询的某一部分时才会分配该记录，一旦该部分执行完成，将释放记录。对于处理放弃和提交所需的状态，它是由正常的操作记录负责处理的，这类记录的池大小由参数MaxNoOfConcurrentOperations设置。

该参数的默认值为8192。只有在极其罕见的情况下，需要使用唯一性哈希索引执行极高的并行操作时，才有必要增大该值。如果DBA（数据库管理员）确信该簇不需要高的并行操作，可以使用较小的值并节省内存。

· [NDBD]MaxNoOfFiredTriggers

MaxNoOfFiredTriggers的默认值是4000，它足以应付大多数情况。在某些情况下，如果DBA认为在簇中对并行操作的要求并不高，甚至还能降低它。

执行会影响唯一哈希索引的操作时，将创建记录。在具有哈希索引的表中插入或删除记录时，或更新作为唯一哈希索引组成部分的列时，均会触发索引表中的插入或删除操作。所获得的记录用于代表该索引表操作，同时等待促使其完成的初始操作。该操作的时间很短，但对于在基表（包含唯一哈希索引）上有很多并发写操作的情形，仍需要在记录池中有大量的记录。

· [NDBD]TransactionBufferMemory

该参数影响的内存用于跟踪更新索引表和读取唯一索引时执行的操作。该内存用于保存关于这类操作的键和列信息。几乎不需要更改该参数的默认值。

正常的读和写操作使用类似的缓冲区，其使用时间甚至更短。编译时间参数ZATTRBUF_FILESIZE（在ndb/src/kernel/blocks/Dbtc/Dbtc.hpp中）被设为4000*128字节（500KB）。用于 键信息的类似缓冲区，ZDATABUF_FILESIZE（也在Dbtc.hpp中）包含4000 * 16 = 62.5KB的缓冲空间。Dbtc是用于处理事务协调的模块。

扫描和缓冲

在Dblqh模块中（在ndb/src/kernel/blocks/Dblqh/Dblqh.hpp内）有很多附加参数，这些参数会影响读和写操作。这些参数包括：ZATTRINBUF_FILESIZE，默认值为10000*128字节（1250KB）；以及ZDATABUF_FILE_SIZE，默认的缓冲空间大小为10000*16字节（约156KB）。到目前为止，没有任何迹象表明应增加这类编译时间限制参数的值，无论是用户报告还是我们自己的大量测试。

TransactionBufferMemory的默认值是1MB。

· [NDBD]MaxNoOfConcurrentScans

该参数用于控制可在簇中执行的并行扫描的数目。每个事务协调程序均能处理为该参数定义的并行扫描。对于每次执行的扫描查询，将以并行方式扫描所有分区。每次分区扫描将使用分区所在节点内的扫描记录，记录数等于该参数的值乘以节点数。簇应能支持从簇内所有节点同时执行的MaxNoOfConcurrentScans扫描。

扫描实际上是在两种情况下执行的。第1种情况是，处理查询时不存在哈希或有序索引，在该情况下，查询是通过执行全表扫描进行的。第2种情况是，没有支持查询的哈希索引，但存在有序索引。使用有序索引意味着将执行并发范围扫描。由于顺序仅保存在本地分区上，需要在所有分区上执行索引扫描。

MaxNoOfConcurrentScans的默认值是256。最大值为500。

该参数指定了事务协调器中的可能扫描数。如果未提供本地扫描记录的数目，会对其进行计算，等于MaxNoOfConcurrentScans乘以系统中数据节点的数目。

· [NDBD]MaxNoOfLocalScans

如果很多扫描不是完全并行化的，指定本地扫描记录的数目。

· [NDBD]BatchSizePerLocalScan

该参数用于计算锁定记录的数目，要想处理很多并发扫描操作，需要这类记录。

默认值是64，该值与SQL节点中定义的ScanBatchSize关系密切。

· [NDBD]LongMessageBuffer

这是用于在单独节点内和节点之间传递消息的内部缓冲。尽管几乎不需要改变它，但它仍是可配置的。默认情况下，它被设置为1MB。

日志和**Checkpointing**

· [NDBD]NoOfFragmentLogFiles

该参数用于设置节点的REDO日志文件的大小。REDO日志文件是按循环方式组织的。第1个和最后1个日志文件（有时也分别称为"头"日志文件和"尾"日志文件）不应相遇，这点极其重要，当它们彼此过于接近时，由于缺少新日志记录的空间，节点将开始放弃所有事务，包括更新。

自插入日志记录开始，在三个本地检查点完成之前，不会删除REDO日志记录。检查点的频率由其自己的配置参数集决定，请参见本章的相应部分。

默认的参数值为8，它表示有8个集合，每个集合有4个16MB文件，总容量为512MB。换句话讲，REDO日志空间必须按64MB的块大小分配。在需要大量更新的情况下，可能需要将NoOfFragmentLogFiles的值增加到300或更高，以便为REDO日志提供足够的空间。

如果checkpointing很慢，并有很多对数据库的写操作以至于日志文件已满，而且在没有jeapo rdising恢复功能的情况下无法截断日志尾部，那么所有的更新日志均将被放弃，并给出错误代码410或缺少临时日志空间。该状况将一直持续，直至完成了检查点操作并能将日志尾部向前移动为止。

· [NDBD]MaxNoOfSavedMessages

该参数用于设置跟踪文件的最大数目，在覆盖旧文件之前，将保留这些跟踪文件。无论出于何种原因，当节点崩溃时将创建跟踪文件。

默认为25个跟踪文件。

元数据对象

下一组参数为元数据对象定义了池的大小，可用于定义最大属性数，表，索引，索引使用的触发程序对象，事件，以及簇之间的复制。注意，这些参数仅是对簇的"建议"，任何未指定的参数均将采用其默认值。

· [NDBD]MaxNoOfAttributes

定义了可在簇中定义的属性数目。

该参数的默认值为1000，最小的可能值为32。没有最大值限制。对于每一属性，每节点约需200字节的存储空间，这是应为，所有的元数据将完整地复制到服务器上。

设置MaxNoOfAttributes时，应实现准备好打算在将来执行的任何ALTER TABLE命令，这点很重要。这是因为下述事实，在簇表上执行ALTER TABLE的过程中，所使用的属性数目是原始表中的3倍。例如，如果某一表需要100个属性，而且你打算在以后更改它，那么就需要将MaxNoOfAttributes的值设为300。有一个良好的经验规则，如果你能在不出现问题的情况下创建所有所需的表，请将最大表中属性数目的两倍加到MaxNoOfAttributes上。完成该设置后，应通过执行实际的ALTER TABLE操作，验证该数目是足够的。如果失败，将原始值的倍数加到MaxNoOfAttributes上，并再次测试。

· [NDBD]MaxNoOfTables

表对象是为每个表、唯一哈希索引和有序索引分配的。该参数为作为整体的簇设置了最大表对象数目。

对于具有BLOB数据类型的每个属性，将使用额外的表来保存大部分BLOB数据。定义表的总数时，必须将这些表考虑在内。

该参数的默认值为128。最小值为8，最大值为1600。每个表对象每节点约需20KB的空间。

· [NDBD]MaxNoOfOrderedIndexes

对于簇中的每个有序索引，将分配1个对象，该对象描述了编入索引的是什么以及其存储段。默认情况下，每个这样定义的索引还将定义1个有序索引。每个唯一索引和主键既有1个有序索引还有1个哈希索引。

该参数的默认值为128。每个对象每节点约需10KB的数据。

· [NDBD]MaxNoOfUniqueHashIndexes

对于每个不是主键的唯一索引，将分配1个特殊表，该表将唯一键映射到索引表的主键上。默认情况下，对于每个唯一索引，还将定义1个有序索引。为了防止该情况，定义唯一索引时，必须使用USING HASH选项。

默认值是64。每个索引每节点约需15KB的空间。

· [NDBD]MaxNoOfTriggers

对于每个唯一性哈希索引，将分配内部更新、插入、和删除触发程序（这意味着对于每个唯一性哈希索引，将创建三个触发程序）。但是，1个有序索引仅需要1个触发程序对象。对于簇中每个正常表，备份也将使用三个触发程序对象。

注释：支持簇之间的复制时，也将使用内部触发程序。

该参数用于设置簇中触发程序对象的最大数目。

该参数的默认值为768.

· [NDBD]MaxNoOfIndexes

在MySQL 5.1中，该参数已被放弃，应使用MaxNoOfOrderedIndexes和MaxNoOfUniqueHashIndexes取而代之。

该参数仅供唯一性哈希索引使用。对于在簇中定义的每个唯一性哈希索引，在该池中需要有1条记录。

该参数的默认值为128.

布尔参数

数据节点的行为也会受具有布尔值的一组参数的影响。将其设为"1"或"Y"，可将这类参数设置为"真"，将其设为"0"或"N"，可将这类参数设置为"假"。

· [NDBD]LockPagesInMainMemory

对于包括Solaris和Linux在内的很多操作系统，能够将进程锁定在内存中，以避免与磁盘的交换。使用它，可确保簇的实时特性。

默认情况下，该特性是被禁止的。

· [NDBD]StopOnError

出现错误时，该参数指定了NDBD进程是退出还是执行自动重启。

默认情况下，允许该特性。

· [NDBD]Diskless

能够将MySQL簇的表指定为"无磁盘的"，这意味着不会在磁盘上对表执行检查点操作，也不会出现日志操作。这类表仅存在于主内存中。使用"无磁盘"表的一个结果是，出现崩溃侯，不会保留这类表，也不会保留这类表中的任何记录。但是，当工作在"无磁盘"模式下时，能够在无盘计算机上运行**ndbd**。

要点：该特性会使整个簇运行在"无磁盘"模式下。

允许该特性时，可执行备份操作，但不会实际保存备份数据。

将"Diskless"设置为"1"或"Y"可允许该特性。默认情况下，禁止该特性。

· [NDBD]RestartOnErrorInsert

仅当创建调试版时才能访问该特性，在执行作为测试组成部份的代码块的过程中，可以插入错误。

默认情况下，该特性是被禁止的。

控制超时、间隔、和磁盘分页

有多种用于指定超时以及簇数据节点中各种动作间时间间隔的参数。大多数超时值以毫秒为单位指定。任何例外均将在适用时指明。

· [NDBD]TimeBetweenWatchDogCheck

为了防止主线程在某一点上陷入无限循环，采用了"看门狗"线程来检查主线程。该参数以毫秒为单位指定了检查之间的时间间隔。如果三次检查之后进程仍保持在相同的状态，它将被"看门狗"线程中止。

出于试验目的，可方便地更改该参数，也可以对其进行调整以适合本地条件。也可以按节点指定它，虽然这样作的理由很少。

默认超时为4000毫秒（4秒）。

· [NDBD]StartPartialTimeout

该参数指定了在调用簇初始化子程序之前，簇等待所有存储节点出现的时间。该超时参数由于防止部分簇启动。

默认值是30000毫秒（30秒）。0表示无限超时，换句话讲，仅当所有节点均可能时才会启动簇。

· [NDBD]StartPartitionedTimeout

等待了StartPartialTimeout毫秒后，如果簇做好了启动准备但仍可能处于隔离状态，簇将等待该超时时间结束。

默认超时为60000毫秒（60秒）。

· [NDBD]StartFailureTimeout

如果数据节点在该参数指定的时间内未完成其启动序列，节点启动将失败。如果将该参数设置为0，表示不采用数据节点超时。

默认值是60000毫秒(60秒）。对于包含大量数据的数据节点，应增加该参数。例如，对于包含数GB数据的存储节点，为了执行节点重启，可能需要10～15分钟（即600000～1000000毫秒）。

· [NDBD]HeartbeatIntervalDbDb

发现失败节点的主要方法之一是使用"心跳"数。该参数指明了心跳信号的发送频率，以及接收它们的频率。如果在1行内丢失了三次心跳，节点将被宣告为死亡。因此，通过心跳机制发现故障的最大时间是心跳间隔的四倍。

默认的心跳间隔为1500毫秒（1.5秒）。不得大幅度更改该参数，各节点间该参数的变化范围也不得过宽。例如，如果某一节点使用了5000毫米的值，而观察它的节点采用1000毫秒，很显然，该节点很快就会被宣布为死亡。能够在软件升级期间更改该参数，但增量应较小。

· [NDBD]HeartbeatIntervalDbApi

每个数据节点会将心跳信号发送到各MySQL服务器（SQL节点），以确保保持接触。如果某一MySQL服务器未能及时发出心跳信号，它将被宣布为死亡。在这种情况下，所有正在进行的事务将结束并释放所有资源。SQL节点不能重新连接，直至由以前的MySQL实例初始化的所有活动完成为止。用于该判断的3心跳判据与HeartbeatIntervalDbDb描述的相同。

默认时间间隔为1500毫秒（1.5秒）。不同的数据节点之间，该间隔可以有所不同，这是因为，每个存储节点均会独立于所有其他数据节点观察与之相连的MySQL服务器。

· [NDBD]TimeBetweenLocalCheckpoints

该参数是一个例外，它未指定启动新的本地检查前应等待的时间，相反，它用于确保在出现相对较少更新的簇内未执行本地检查点操作。在具有较高更新率的大多数簇内，很可能在前一个本地检查点操作完成后立刻启动一个新的检查点操作。

从前一个本地检查点启动后，所有已执行写操作的大小将增加。该参数也是一个例外，原因在于它被指定为4字节字总数的以2为底数的对数，因此，默认值20表示4MB (4×2^{20})写操作，21表示8MB，依此类推，直至等同于8GB写操作的最大值31。

簇中所有的写操作将加在一起。将TimeBetweenLocalCheckpoints设置为6或更小表示本地检查点操作将不停顿地连续执行，与簇的工作负荷无关。

· [NDBD]TimeBetweenGlobalCheckpoints

提交事务时，它被提交到存有镜像数据的所有节点的主内存中。但是，事务日志记录不会作为提交进程的一部分写入磁盘。其原因在于，在至少两台独立主机机器上安全体提交事务应能满足关于关于持久性的合理标准。

另一个很重要的方面是，应确保即使在最差情况下（簇完全崩溃），也能进行恰当地处理。为了确保这点，在给定时间间隔内出现的所有事务均会被放到全局检查点，可将其视为写入磁盘的已提交事务的集合。换句话讲，作为提交进程的组成部分，事务将被放入全局检查点组；稍后，该组的日志记录将被写入磁盘，然后将整个事务组安全地提交到簇内所有计算机的磁盘上。。

该参数定义了全局检查点操作之间的时间间隔。默认值为2000毫秒。 milliseconds.

· [NDBD]TimeBetweenInactiveTransactionAbortCheck

对于该参数指定的每个时间间隔，通过检查每个事务的定时器来执行超时处理。因此，如果该参数被设为1000毫秒，每隔1秒就会对事务进行检查。

该参数的默认值为1000毫秒（1秒）。

· [NDBD]TransactionInactiveTimeout

如果事务目前未执行任何查询，而是等待进一步的用户输入，该参数指明了放弃事务之前用户能够等待的最长时间。

该参数的默认值是0（无超时）。对于需要确保无任何事务锁定了过长时间的数据库，应将参数设置为较小的值。单位为毫秒。

· [NDBD]TransactionDeadlockDetectionTimeout

当节点执行涉及事务的查询时，在继续之前，节点将等待簇中其他节点作出回应。如果出现下述原因，将无法予以回应:

1. 节点"死亡"。
2. 操作进入锁定队列。
3. 被请求执行动作的节点负荷过重。

该超时参数指明了放弃事务之前，事务协调器等候另一节点执行查询的时间长短，该参数在节点失败处理和死锁检测方面十分重要。在涉及死锁和节点失败的情形下，如果将其设置的过高，会导致不合需要的行为。

默认的超时值为1200毫秒（1.2秒）。

· [NDBD]NoOfDiskPagesToDiskAfterRestartTUP

执行本地检查点操作时，相应的算法会将所有数据页写入磁盘。如果追求尽快完成该操作而不是适中，很可能会对处理器、网络和磁盘带来过重负担。为了控制写入速度，该参数指明了每100毫秒可写入多少数据页。在本情形下，1个数据页定义为8KB，因而该参数的单位是每秒80KB。因此，如果将NoOfDiskPagesToDiskAfterRestartTUP设置为20，那么在执行本地检查点操作期间，要求每秒想磁盘写入1.6MB的数据。该值包括针对数据页的UNDO日志记录写入，也就是说，该参数能处理来自数据内存的写入限制。置于针对索引页的UNDO日志记录，它们是由参数NoOfDiskPagesToDiskAfterRestartACC处理的（关于索引页的更多信息，请参见关于IndexMemory的条目）。

简而言之，该参数指定了执行本地检查点操作的速度，并能与NoOfFragmentLogFiles、DataMemory和IndexMemory一起使用。

默认值是40（每秒3.2MB的数据页）。

· [NDBD]NoOfDiskPagesToDiskAfterRestartACC

该参数使用的单位与NoOfDiskPagesToDiskAfterRestartTUP的相同，工作方式也类似，但限制的是从索引内存进行的索引页写入速度。

该参数的默认值为每秒20个索引内存页（1.6MB每秒）。

· [NDBD]NoOfDiskPagesToDiskDuringRestartTUP

该参数的工作方式类似于NoOfDiskPagesToDiskAfterRestartTUP和NoOfDiskPagesToDiskAfterRestartACC，但仅对重启节点时在节点内执行的本地检查点操作有效。作为所有节点重启的组成部份，总会执行本地检查点操作。在节点重启过程中，能够以比其他时间更快的速度执行磁盘写入操作，这是因为，此时在节点内执行的活动数较少。

该参数涉及从数据内存写入的页。

默认值是40（3.2MB每秒）。

· [NDBD]NoOfDiskPagesToDiskDuringRestartACC

在节点重启的本地检查点阶段，对能够写入到磁盘的索引内存页的数目进行控制。

与NoOfDiskPagesToDiskAfterRestartTUP和NoOfDiskPagesToDiskAfterRestartACC一样，该参数的值采用的单位也是每100毫秒写入8KB（80KB/秒）。

默认值是20 (1.6MB每秒）。

· [NDBD]ArbitrationTimeout

该参数指定了数据节点等待仲裁程序对仲裁消息的回应的时间。如果超过了该时间，将假定网络已断开。

默认值是1000毫秒（1秒）。

缓冲和日志功能

一些与以前的编译时间参数对应的配置参数仍可用。使用这些参数，高级用户能够对节点进程使用的资源进行更多的控制，并能根据需要调整各种缓冲区大小。

将日志记录写入磁盘时，这些缓冲区用作文件系统的前端。如果节点运行在无盘模式下，那么可以将这些参数设置为它们的最小值而不会造成负面影响，这是因为，磁盘写入是由NDB存储引擎的文件系统提取层虚拟的。

· [NDBD]UndoIndexBuffer

该缓冲用于本地检查点操作执行期间。NDB存储引擎采用了一种恢复方案，该方案建立在检查点一致性以及操作性REDO日志值上。为了在不隔断整个系统的写操作的情况下获得一致的检查点，在执行本地检查点操作的同时，将执行UNDO日志操作。UNDO日志功能每次是在单个表偏短上触发的。由于表全部保存在主内存中，该优化是可能的。

UNDO索引缓冲用于主键哈希索引上的更新。插入和删除操作会导致哈希索引的重新排列，NDB存储引擎将映射了所有物理变化的UNDO日志记录写入索引页，以便能在系统重启时撤销这些变化。它还能记录启动本地检查点操作时对每个偏短的所有插入操作。

读取和更新能够设置锁定位，并更新哈希索引条目中的标题。这类变更由页写入算法负责处理，以确保这些操作不需要UNDO日志。

该缓冲的默认大小为2MB。最小值为1MB，对于大多数应用，最小值已足够。对于执行极大和／或大量插入和删除操作、并处理大事务和大主键的应用程序，或许有必要增大该缓冲。如果该缓冲过小，NDB存储引擎会发出错误代码677"索引UNDO缓冲过载"。

· [NDBD]UndoDataBuffer

UNDO数据缓冲的作用与UNDO索引缓冲的相同，不同之处在于，它作用在数据内存上而不是索引内存上。对于插入、删除和更新，该缓冲是在片段的本地检查点阶段使用的。

由于UNDO日志条目会随着所记录操作的增加而增大，该缓冲大于与之对应的索引内存缓冲，默认值为16MB。

对于某些应用程序，该内存可能过大。在这种情况下，可降低它的值，最小为1MB。

需要增加该缓冲的情况十分罕见。如果确实有这方面的要求，较好的方式是，检查磁盘是否能实际处理数据库更新活动所产生的负荷。如果缺少足够的磁盘空间，即使增加该缓冲的大小也不能解决问题。

如果该缓冲过小并变得"拥挤不堪"，NDB存储引擎将发出错误代码891"数据UNDO缓冲过载"。

· [NDBD]RedoBuffer

所有的更新活动也需要被记录到日志中。使用这类日志，当系统重启时，能够重现这类更新。NDB恢复算法采用了"模糊"数据检查点和UNDO日志，然后使用REDO日志再现所有变化直至到达恢复点。

该缓冲的默认大小是8MB。最小值为1MB。

如果该缓冲过小，NDB存储引擎将发出错误代码1221"REDO日志缓冲过载"。

在管理簇的过程中，应能控制为各种事件类型发送至标准输出装置的日志消息的数目，这点十分重要。有16种可能的事件级别（编号从0到15）。如果将给定事件类别的事件通报级别设置为15，那么该类别中的所有事件报告均会被发送至标准输出装置，如果将其设置为0，表示在该类别中的没有事件报告。

默认情况下，仅会将启动消息发送至标准输出装置，其余的事件通报级别默认为0。这样做的原因在于，这些消息也会被发送至管理服务器的簇日志。

对于管理客户端，也能设置类似的级别，用以确定在簇日志中记录哪些级别的事件。

· [NDBD]LogLevelStartup

通报级别，用于进程启动过程中生成的事件。

默认级别为1.

· [NDBD]LogLevelShutdown

通报级别，用于作为节点恰当关闭进程组成部分而生成的事件。

默认级别为0.

· [NDBD]LogLevelStatistic

通报级别，用于统计事件，如主键法读取次数，更新数目，插入数目，与缓冲使用有关的信息等。

默认级别为0.

· [NDBD]LogLevelCheckpoint

通报级别，用于由本地和全局检查点操作生成的事件。

默认级别为0.

· [NDBD]LogLevelNodeRestart

通报级别，用于在节点重启过程中生成的事件。

默认级别为0.

· [NDBD]LogLevelConnection

通报级别，用于由簇节点间的连接生成的事件。

默认级别为0.

· [NDBD]LogLevelError

通报级别，用于由在整个簇内的错误和警告生成的事件。这类错误不会导致任何节点失败，当仍值得通报。

默认级别为0.

· [NDBD]LogLevelInfo

通报级别，用于为簇的一般状态信息而生成的事件。

默认级别为0.

备份参数

本节讨论的参数定义了与在线备份执行有关的内存缓冲集。

· [NDBD]BackupDataBufferSize

在创建备份的过程中，为了将数据发送到磁盘，将使用两类缓冲。备份数据缓冲用于填充由扫描节点的表而记录的数据。一旦将该缓冲填充到了指定的水平BackupWriteSize（请参见下面的介绍），就会将页发送至磁盘。在将页写入磁盘的同时，备份进程能够继续填充该缓冲，直至其空间消耗完为止。出现该情况时，备份进程将暂停扫描，直至一些磁盘写入操作完成并释放了内存为止，然后扫描继续。

该参数的默认值为2MB。

· [NDBD]BackupLogBufferSize

备份日志缓冲扮演的角色类似于备份数据缓冲，不同之处在于，它用于生成备份执行期间进行的所有表写入的日志。相同的原理也适用于备份数据缓冲情形下的页写入，不同之处在于，当备份日志缓冲中没有多余空间时，备份将失败。出于该原因，备份日志缓冲的大小应足以处理执行备份时产生的负载。

该参数的默认值对于大多数应用程序均是适当的。事实上，备份失败的原因更可能是因为磁盘写入速度不够，而不是备份日志缓冲变满。如果没有为应用程序产生的写负载配置磁盘子系统，簇很可能无法执行所需的操作。

最好按恰当的方式配置簇，使得处理器成为瓶颈而不是磁盘或网络连接。

默认值是2MB。

· [NDBD]BackupMemory

该参数是BackupDataBufferSize和BackupLogBufferSize之和。

默认值是2MB + 2MB = 4MB。

· [NDBD]BackupWriteSize

该参数指定了由备份日志缓冲和备份数据缓冲写入磁盘的消息大小。

默认值是32KB.

17.4.4.6. 定义MySQL簇内的SQL节点

在config.ini文件的[MYSQLD]部分中，定义了用于访问簇数据的MySQL服务器（SQL节点）的行为。不需要其中所给出的参数。如果未提供计算机或主机名，那么任何主机均能使用该SQL节点。

· [MYSQLD]Id

该值用作节点的地址，供所有的簇内部消息使用，它必须是介于1和63之间的整数。在簇内，每个簇节点必须有唯一的ID。

· [MYSQLD]ExecuteOnComputer

它引用的是在配置文件的[COMPUTER]部分定义的主机（计算机）之一。

· [MYSQLD]ArbitrationRank

该参数用于定义可作为仲裁程序的节点。MGM节点和SQL节点均能成为仲裁程序。如果值为0，表明给定的节点永远不会用作仲裁程序，如果值为1，表明给定的节点在成为仲裁程序方面具有高的优先级，如果值为2，表明给定的节点在成为仲裁程序方面具有低的优先级。对于正常配置，使用管理服务器作为仲裁程序，将它的ArbitrationRank设置为1（默认），并将所有SQL节点的ArbitrationRank设置为0。

· [MYSQLD]ArbitrationDelay

如果将该参数设置为除0（默认值）以外的其他值，表示仲裁程序对仲裁请求的相应将被延迟设定的毫秒数。通常不需要更改该值。

· [MYSQLD]BatchByteSize

对于转换为全表扫描或对索引的范围扫描的查询，要想获得最佳性能，重要的是以恰当的大小获取记录。能够以记录数为单位（BatchSize）和字节为单位（BatchByteSize）设置恰当的大小。实际的批大小由两个参数限定。

查询的执行速度可能会出现40％的变化，具体情况取决于该参数的设置。在未来的版本中，MySQL服务器将根据查询类型恰当地设置与批大小相关的参数。

该参数以字节为单位，默认值是32KB。

· [MYSQLD]BatchSize

该参数以记录数为单位，默认值是64。最大值为992。

· [MYSQLD]MaxScanBatchSize

批大小指的是从各数据节点发送的每批数据的大小。大多数扫描均是以并行方式执行的，目的是为了防止MySQL服务器收到来自众多节点的过多数据，该参数对所有节点上的总的批大小进行了限制。

该参数的默认值为256KB。其最大大小为16MB。

17.4.4.7. MySQL簇TCP/IP连接

在MySQL簇中，TCP/IP是用于建立连接的默认传输协议。正常情况下不需要定义连接，这是因为，簇能自动建立数据节点间、数据节点与所有MySQL服务器节点、以及数据节点与管理服务器之间的连接（关于该规则的例外，，请参见17.4.4.8节，“使用直接连接的MySQL簇TCP/IP连接”)。

如果打算覆盖默认的连接参数，才需要定义连接。在这种情况下，至少需要定义*NodeId1*、*NodeId2*、以及打算更改的参数。

通过在[TCP DEFAULT]部分进行设置，也能更改这些参数的默认值。

· [TCP]*NodeId1* , [TCP]*NodeId2*

要想确定两个节点之间的连接，需要在配置文件的［TCP］部分中提供每个节点的ID。

· [TCP]SendBufferMemory

在向操作系统发出调用请求之前，TCP传输器采用缓冲来保存所有的消息。当该缓冲达到64KB时，将发送其内容，执行完一组消息循环后，也将发送这些内容。为了处理临时过载情况，也能定义一个较大的发送缓冲。发送缓冲的默认值是256KB。

· [TCP]SendSignalId

为了能够回扫分布式消息图，需要确定每条消息。将该参数设置为"Y"时，将通过网络传输消息ID。默认情况下禁止该特性。

· [TCP]Checksum

该参数也是一个布尔参数（Y/N或1/0），默认情况下是禁止的。启用了该参数时，在将所有消息置于发送缓冲之前，将为所有参数计算校验和。使用该特性，当消息等候在发送缓冲中时，可以确保消息不会损坏，也能确保消息不会被传输机制破坏。

· [TCP]PortNumber

（已过时）以前，该参数指定了用于监听来自其他节点的连接的端口号。不应再使用嘎参数。

· [TCP]ReceiveBufferMemory

指定了从TCP/IP套接字接收数据时所使用的缓冲大小。几乎不需要更改该参数的默认值，默认值为64KB，但是如果打算节省内存，也能更改它。

17.4.4.8. 使用直接连接的MySQL簇TCP/IP连接

使用数据节点之间的直接连接建立簇时，需要在簇config.ini文件的[TCP]部分中明确指定如此连接的数据节点的交叉IP地址。

在下面的示例中，假定簇具有至少4台主机，1台用于管理服务器，一台用于SQL节点，两台用于数据节点。作为整体，簇位于LAN的172.23.72.*子网内。除了通常的网络连接外，两个数据节点使用标准的交叉电缆直接相连，并使用范围在1.1.0.*的IP地址彼此间直接通信，如下所示：

```
# Management Server
[NDB_MGMD]
Id=1
HostName=172.23.72.20

# SQL Node
[MYSQLD]
Id=2
HostName=172.23.72.21

# Data Nodes
[NDBD]
Id=3
HostName=172.23.72.22

[NDBD]
Id=4
HostName=172.23.72.23

# TCP/IP Connections
[TCP]
NodeId1=3
NodeId2=4
HostName1=1.1.0.1
HostName2=1.1.0.2
```

使用数据节点间的直接连接能够改善簇的整体效率，使用该方式，数据节点能绕过以太网设备，如交换器、Hub、路由器等，从而减少了簇的等待时间。注意，对于两个以上的数据节点，要想充分利用这类直接连接的优点，需要为各数据节点建立与相同节点组内的其他数据节点间的直接连接。

17.4.4.9. MySQL簇共享内存连接

MySQL簇将尝试使用共享内存传输器，并在可能的情况下自动配置它，尤其是在相同的簇主机上同时运行着1个以上的节点时。在MySQL簇的早期版本中，仅当使用--with-ndb-shm创建了-max二进制版本时，才支持共享内存段。明确地将共享内存定义为连接方法时，至少需要定义NodeId1、NodeId2和ShmKey。对于所有其他参数，应具有在大多数情况下均良好工作的默认值。

注释：SHM支持仍应被视为试验性的。

· [SHM]NodeId1, [SHM]NodeId2

要想确定两个节点之间的连接，需要为每个节点提供节点ID，NodeId1和NodeId2。

- [SHM]ShmKey

设置共享内存段时，节点ID用于唯一地确定通信所用的共享内存段。它以整数表示，没有默认值。

- [SHM]ShmSize

每个SHM连接均有一个共享内存段，发送方将节点之间的消息置于该处，读取方从该处读取这类消息。gai 共享内存段的大小由ShmSize定义。默认值是1MB。

- [SHM]SendSignalId

为了回扫分布式消息的路径，需要为每条消息提供唯一性ID。如果将该参数设置为"Y"，也能在网络上传输这类消息ID。默认情况下，该特性是禁止的。

- [SHM]Checksum

该参数也是一种Y/N参数，默认情况下处于禁止状态。如果允许该参数，在将所有消息置于发送缓冲之前，对为所有消息计算校验和。

使用该特性，当消息等候在发送缓冲中时，能防止消息损坏。此外，它还能用于在传输过程中检查损坏的数据。

17.4.4.10. MySQL簇SCI传输连接

仅当使用--with-ndb-sci=*/your/path/to/SCI*创建了MySQL-Max二进制版本时，在MySQL簇中才支持使用SCI传输器来连接节点。*path*应指向包含最低库的目录，并应包括含SISCI库和头文件的目录。

此外，SCI要求专用硬件。

强烈建议，仅应为ndbd进程之间的通信使用SVI传输器。注意，使用SCI传输器意味着ndbd进程永不停止。因此，仅应在具有至少两块专供ndbd进程使用的CPU的机器上使用SCI传输器。每个ndbd进程至少应有1块CPU，至少还应有1块CPU用于处理操作系统的活动。

- [SCI]NodeId1, [SCI]NodeId2

为了确定两个节点之间的连接，需要为每个节点提供节点ID，NodeId1和NodeId2。

- [SCI]Host1SciId0

它用于确定第1个簇节点上的SCI节点ID（由NodeId1确定）。

- [SCI]Host1SciId1

能够为两块SCI卡间的故障切换设置SCI传输器，这两块卡应使用节点之间的不同网络。它用于确定节点ID，以及在第1个节点上使用的第2块SCI卡。

- [SCI]Host2SciId0

它用于确定第2个簇节点上的SCI节点ID（由NodeId2确定）。

- [SCI]Host2SciId1

使用两块SCI卡来提供故障切换功能时，该参数用于确定将在第2个节点上使用的第2块SCI卡。

- [SCI]SharedBufferSize

每个SCI传输器均有1个用于两节点间通信的共享内存段。可将该共享内存段设置为默认的1 MB，这足以应对大多数应用程序。如果使用较小的值，当执行大量并行插入操作时，会出现问题，如共享缓冲过小，还会导致ndbd进程崩溃。

- [SCI]SendLimit

SCI媒介前面的小缓冲用于保存消息，在通过SCI网络传输这类消息前，会将它们保存在该缓冲内。它的默认值为8kB。按照我们的基准，在64KB时性能最好，但16kB仅有少量提升，即使大于8KB有好处，好处也不大。

- [SCI]SendSignalId

为了跟踪分布式消息，需要唯一地确定每条消息。将该参数设置为"Y"时，就能在网络上传输消息ID。默认情况下禁止该特性。

- [SCI]Checksum

T该参数也是一种布尔值，默认情况下，该参数是被禁止的。启用了Checksum（校验和）时，在将所有消息置于发送缓冲之前，将为所有参数计算校验和。使用该特性，当消息等候在发送缓冲中时，可以确保消息不会损坏。此外，它还能用于在传输过程中检查损坏的数据。

17.5. MySQL簇中的进程管理

17.5.1. 用于MySQL簇的MySQL服务器进程使用
17.5.2. **ndbd**，存储引擎节点进程
17.5.3. **ndb_mgmd**，"管理服务器"进程
17.5.4. **ndb_mgm**，"管理客户端"进程
17.5.5. 用于MySQL簇进程的命令选项

要想掌握管理MySQL簇的方法，需要了解4种基本进程。在本章下面的几节内，介绍了这些进程在簇内的作用，它们的使用方法，以及每种进程可用的启动

选项。

17.5.1. 用于MySQL簇的MySQL服务器进程使用

mysqld是传统的MySQL服务器进程。要想与MySQL簇一起使用，所创建的**mysqld**应支持NDB簇存储引擎，就像在预编译的-max二进制版本中那样，http://dev.mysql.com/downloads/。

即使采用该方式创建了**mysqld**二进制版本，在默认情况下，NDB簇存储引擎仍处于禁止状态。要想启用NDB簇存储引擎，可使用两种可能的选项之一：

- 启动**mysqld**时，将"—ndbcluster"用作启动选项。
- 在my.cnf文件的[mysqld]部分插入包含ndbcluster的1行内容。

验证运行的服务器是否启用了NDB簇存储引擎的简单方法是，在MySQL监视器（mysql）中发出命令SHOW ENGINES。在列出NDBCLUSTER的行中应能看到值YES，如果在该行上看到NO（或在输出中未显示该行），你所运行的是未启用NDB功能的MySQl版本。如果在该行上看到DISABLED，就需采用上述两种方法之一启用它。

为了读取簇配置数据，MySQL服务器至少需要3种信息：

- MySQL服务器自己的簇节点ID。
- 管理服务器（MGM节点）的主机名或IP地址。
- 与管理服务器相连的端口。

节点ID可动态分配，因此不一定需要明确指定它们。

mysqld参数ndb-connectstring用于指定连接字符串，或是在启动**mysqld**时在命令行上指定，或是在my.cnf文件中指定。连接字符串包含主机名或IP地址，以及能够发现管理服务器的端口。

在下面的示例中，ndb_mgmd.mysql.com是管理服务器所在的主机，管理服务器在端口1186上监听簇消息。

```
shell> mysqld --ndb-connectstring=ndb_mgmd.mysql.com:1186
```

关于连接字符串的更多信息，请参见17.4.4.2节，"MySQL簇连接字符串"。

给定该信息，MySQL服务器将成为簇中的完全参与者。（有时，我们也将运行在该方式下的**mysqld**进程称为SQL节点）。它能完全了解所有的簇数据节点以及它们的状态，并能建立与所有数据节点的连接。在这种情形下，它能将任何数据节点用作事务协调器，并能访问数据节点以执行读取和更新操作。

17.5.2. ndbd，存储引擎节点进程

ndbd是使用NDB簇存储引擎处理表中所有数据的进程。通过该进程，存储节点能够实现分布式事务处理，节点恢复，对磁盘的检查点操作，在线备份，以及相关的任务。

在MySQL簇中，一组**ndbd**进程能够共同处理数据。这些进程可以在相同的计算机（主机）上执行，也能在不同的计算机上执行。数据节点和簇主机之间的通信是完全可配置的。

Ndbd将生成一组日志文件，这些文件位于由配置文件中DataDir指定的目录下。下面列出了这些日志文件。注意，*node_id*代表节点的唯一ID。例如，ndb_2_error.log是由节点ID为2的存储节点生成的错误日志。

- ndb_*node_id*_error.log是包含所引用ndbd进程所遇到的所有崩溃记录的文件。该文件中的每条记录均包含1个简要的错误字符串，以及对该崩溃跟踪文件的引用。该文件的典型条目与下面给出的类似：

```
Date/Time: Saturday 30 July 2004 - 00:20:01
Type of error: error
Message: Internal program error (failed ndbrequire)
Fault ID: 2341
Problem data: DbtupFixAlloc.cpp
Object of reference: DBTUP (Line: 173)
ProgramName: NDB Kernel
ProcessID: 14909
TraceFile: ndb_2_trace.log.2
***EOM***
```

注释：请记住，错误日志文件中的最后*1*个条目并不必然是最新的（也不太可能），这点很重要。错误日志中的条目不是按时间顺序排列的，而是与ndb_*node_id*_trace.log.next（请参见下面的介绍）中定义的跟踪文件的顺序对应。因此，错误日志条目是按循环方式而不是顺序方式覆盖的。

- ndb_*node_id*_trace.log.*trace_id*是准确描述了错误出现之时所发生情况的跟踪文件。该信息在MySQL簇开发团队进行分析时很有帮助。

能够对覆盖旧文件之前创建的跟踪文件的数目进行配置。*trace_id*是为每个连续的跟踪文件增加的编号。

- ndb_*node_id*_trace.log.next是记录了要指定的下一个跟踪文件编号的文件。
- ndb_*node_id*_out.log是包含ndbd进程的任何数据输出的文件。仅当将ndbd启动为端口监督程序时才会创建该文件。

- ndb_*node_id*.pid是包含启动时作为端口监督程序的ndbd进程的进程ID的文件。它还能起到锁定文件的作用，以防止启动具有相同ID的节点。
- ndb_*node_id*_signal.log是仅在ndbd调试版下使用的文件，它能跟踪ndbd进程中所有的入站、出站和内部消息。以及它们的数据。

建议不要使用通过NFS安装的目录，这是因为在某些情况下，如果pid-file上的锁定依旧有效，即使当进程中止后也会产生问题。

启动**ndbd**时，或许需要指定管理服务器的主机名以及监听的端口号。作为可选方式，也可以指定进程将使用的节点ID。

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

关于这方面的额外信息，请参见17.4.4.2节，“MySQL簇连接字符串”。

启动**ndbd**时，它实际上将启动两种进程。第1种进程称为“angel process”（天使进程），它的唯一任务是发现执行进程在何时完成，然后重启ndbd进程（如果作了该配置的话）。因此，如果你打算使用Unix的kill命令杀死ndbd进程，就需要杀死这两个进程。中止ndbd进程的更恰当方法是使用管理客户端，并通过该管理客户端停止进程。

执行进程采用了1个线程，用于读取、写入和扫描数据，以及所有其他任务。该线程是异步实施的，以便能方便地处理数以千计的并发活动。此外，看门狗线程负责监督执行线程，以确保执行线程不会陷入无限循环。线程池负责处理文件I/O，每个线程均能处理一个打开的文件。这些线程也能被ndbd进程中的传输器用作传输器连接。在执行包含更新在内的大量操作的系统中，如果允许，ndbd进程能占用2个CPU。对于拥有多CPU的机器，建议使用属于不同节点组的数个ndbd进程。

17.5.3. ndb_mgmd，“管理服务器”进程

管理服务器是这样一种进程，它读取簇配置文件，并将该信息分配给簇中所有请求该信息的节点。它还负责维护簇活动的日志。管理客户端能够连接到管理服务器，并检查簇的状态。

启动管理服务器时，不是一定需要指定连接字符串。但是，如果使用了1个以上的管理服务器，应提供连接字符串，而且簇中的每个节点应明确指定自己的节点ID。

下述文件是由**ndb_mgmd**在其启动目录下创建或使用的，并会被置于配置文件中指定的DataDir中。在下面的列表中，*node_id*是唯一性节点ID。

- config.ini是作为整体的簇的配置文件。该文件由用户创建并由管理服务器读取。在17.4节，“MySQL簇的配置”中，讨论了设置该文件的方法。
- ndb_*node_id*_cluster.log是簇事件日志文件。这类事件的例子包括：检查点操作的启动和完成，节点启动事件，节点故障，以及内存使用水平。关于簇事件的完整列表和描述，请参见17.6节，“MySQL簇的管理”。

当簇日志的大小达到1MB时，文件将被重命名为ndb_*node_id*_cluster.log.*seq_id*，其中*seq_id*是簇日志文件的序列号（例如，如果编号1、2、3已存在，下一个日志文件将用4命名）。

- ndb_*node_id*_out.log是将管理服务器用作端口监督程序时用于stdout和stderr的文件。
- ndb_*node_id*.pid是将管理服务器用作端口监督程序时所使用的PID文件。

17.5.4. ndb_mgm，“管理客户端”进程

对于簇的运行，实际上不需要管理客户端的进程。其价值在于它提供了一组命令，这组命令可用于检查簇的状态、启动备份、并执行其他管理功能。管理客户端使用C API来访问管理服务器，高级用户也能使用C API来编制专用的管理进程来执行任务，这类任务与**ndb_mgm**执行的类似。

启动管理客户端时，需要提供管理服务器的主机名和端口号，如下例所示。默认值是localhost和1186。

```
shell> ndb_mgm localhost 1186
```

关于使用**ndb_mgm**的更多信息，请参见17.5.5.4节，“ndb_mgm的命令选项”和17.6.2节，“管理客户端”中的命令。

17.5.5. 用于MySQL簇进程的命令选项

17.5.5.1. 用于mysqld的与MySQL有关的命令选项
17.5.5.2. ndbd命令选项
17.5.5.3. ndb_mgmd的命令选项
17.5.5.4. ndb_mgm的命令选项

所有MySQL簇的可执行文件（除**mysqld**）均使用下述选项。早期MySQL簇版本的用户应注意，这些选项开关中的一些与MySQL 4.1簇中的相比有所改变，为的是保持它们之间的一致性，以及与**mysqld**的一致性。可以使用-?开关来查看支持的选项列表。

- -?, --usage, --help

给出简明清单，以及可用命令选项的描述。

- -V, --version

给出**ndbd**进程的版本号。该版本号是MySQL簇的版本号。版本号有一定的作用，这是因为并非所有的版本均能一起使用，而且在启动时，MySQL簇进程将验证二进制文件的版本是否能在同一簇内共存。执行MySQL簇的在线升级时，它也很重要（请参见MySQL簇的软件升级）。

- -c *connect_string*, --connect-string

*connect_string*作为命令选项，用于设置与管理服务器的连接字符串。

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

· --debug[=*options*]

该选项仅能用于具有调试功能的版本。使用它，能够以与**mysqld**进程相同的方式允许来自调试调用的输出。

· -e, --execute

使用它，能够从系统shell将命令发送至簇执行程序，如：

```
shell> ndb_mgm -e show
```

或

```
shell> ndb_mgm --execute="SHOW"
```

等效于

```
NDB> SHOW;
```

它类似于"-e"选项与**mysql**命令行客户端一起工作的方式。请参见4.3.1节，"在命令行上使用选项"。

17.5.5.1. 用于**mysqld**的与**MySQL**有关的命令选项

· --ndbcluster

如果二进制版本包含对NDB簇存储引擎的支持，可使用该选项覆盖对NDB簇存储引擎（简称为NDB存储引擎）的默认禁止设置。使用MySQL簇时，NDB簇存储引擎是必要的。

· --skip-ndbcluster

禁止NDB簇存储引擎。对于包含该功能的二进制版本，在默认情况下，该功能是被禁止的，换句话讲，NDB簇存储引擎处于禁止状态，直至使用"—ndbcluster"选项激活了它为止。仅当所编译的服务器支持NDB簇存储引擎时，才能使用该选项。

· --ndb-connectstring=*connect_string*

使用NDB存储引擎时，通过设置该选项，能够指定分配簇配置数据的管理服务器。

17.5.5.2. **ndbd**命令选项

关于某些常见选项的更多信息，请参见17.5.5节，"用于MySQL簇进程的命令选项"。

· -d, --daemon

通知**ndbd**作为daemon（端口监督程序）进程执行（默认行为）。

· --nodaemon

指明**ndbd**不得作为daemon（端口监督程序）进程启动。调试ndbd以及希望将输出重定向到屏幕时，它很有用。

· --initial

通知**ndbd**执行初始化启动。初始化启动将删除以前ndbd实例为恢复目的创建的任何文件。它还能重新创建恢复用日志文件。注意，在某些操作系统上，该进程可能会占用较长的时间。

仅在首次启动ndbd进程时才应使用—initial启动，这是因为它将删除簇文件系统的所有文件，并再次创建所有的REDO日志文件。该规则的例外如下：

o 执行那些会更改文件内容的软件升级时。

o 用新的ndbd版本重启节点时。

o 出于某种原因，节点重启或系统重启不断失败时的最后手段。在这类情形下，请注意，由于数据文件的损坏，不能使用该节点来恢复数据。

该选项不影响那些已被受影响节点创建的备份文件。

· --nostart

指示**ndbd**不自动启动。使用该选项时，**ndbd**连接到管理服务器，从管理服务器获取配置数据，并初始化通信对象。但是，在管理服务器特别要求之前，它不会实际启动执行引擎。通过向管理客户端发出恰当的命令，可完成该任务。

17.5.5.3. **ndb_mgmd**的命令选项

关于某些常见选项的更多信息，请参见17.5.5节，"用于MySQL簇进程的命令选项"。

· -f *filename*, --config-file=*filename*, (**OBSOLETE**): -c *filename*

通知管理服务器应使用哪个文件作为其配置文件。必须指定该选项。文件名默认为config.ini。注意，"-c"快捷方式已过时，不应在新的安装实例中使用

它。

· -d, --daemon

指示**ndb_mgmd**作为端口监督程序启动。这是默认行为。

· --nodaemon

指示管理服务器不作为端口监督程序启动。

17.5.5.4. ndb_mgm的命令选项

关于某些常见选项的更多信息，请参见17.5.5节，“用于MySQL簇进程的命令选项”。

· [*host_name* [*port_num*]]

要想启动管理客户端，需要指定管理服务器所在的位置，即指定主机名和端口。默认的主机名是localhost，默认端口是1186。

· --try-reconnect=*number*

如果与管理服务器的连接断开，每隔5秒，节点将尝试再次连接到管理服务器，直至成功。使用该选项，能够将尝试的字数限制在number指定的值，超过该限制后，将放弃尝试并通报错误。

17.6. MySQL簇的管理

17.6.1. MySQL簇的启动阶段
17.6.2. “管理客户端”中的命令
17.6.3. MySQL簇中生成的事件报告
17.6.4. 单用户模式
17.6.5. MySQL簇的联机备份

管理MySQL簇涉及众多任务，首先是配置和启动MySQL簇。详情请参见17.4节，“MySQL簇的配置”和17.5节，“MySQL簇中的进程管理”。

下面介绍了MySQL簇的管理事宜。

有两种积极管理MySQL簇的基本方法。第1种方法是，使用在管理客户端中输入的命令，借此可检查簇的状态，更改日志级别，启动和停止备份，以及启动和停止节点。对于第2种方法，需要研究管理服务器DataDir目录下簇日志文件ndb_*node_id*_cluster.log的内容。（*node_id*代表其活动已被记录的节点的唯一ID）。簇日志包含由**ndbd**生成的事件报告。也能将簇日志条目发送到Unix的系统日志中。

17.6.1. MySQL簇的启动阶段

本节介绍了启动簇时涉及的步骤。

有数种不同的启动类型和模式，如下所述：

· 首次启动：在所有节点上与干净的文件系统一起启动簇。这或是出现在首次启动簇时，或是使用“--initial”选项重启簇时。

· 系统重启：簇启动并读取保存在数据节点中的数据。这出现在下述情况下：使用完后关闭了簇，并希望从簇的停止点恢复簇操作时。

· 节点重启：这是在簇运行的同时簇节点的在线重启。

· 首次节点重启：与节点重启类似，差别在于将再次初始化节点，并与干净的文件系统一起启动。

启动之前，必须对每个节点进行初始化操作（ndbd进程）。这包括下述步骤：

1. 获取节点ID。
2. 获取配置数据。
3. 为节点间的通信分配端口。
4. 根据从配置文件获得的设置分配内存。

一旦完成了对各节点的初始化操作，将进入簇启动进程。在该进程中，簇将经历下述阶段：

· 阶段**0**

清理簇文件系统。仅当使用“--initial”选项启动簇时，才会出现。

· 阶段**1**

建立簇连接，建立节点间的通信。启动簇“心跳”机制。

· 阶段**2**

选举仲裁程序节点。

如果这是系统重启阶段，簇将确定最近的可恢复全局检查点。

· 阶段**3**

该阶段包括众多内部簇变量的初始化。

· 阶段**4**

对于初始启动或初始节点重启，将创建redo日志文件。这类文件的数目等于NoOfFragmentLogFiles／

对于系统重启：

o 读取方案。

o 从本地检查点和undo日志读取数据。

o 应用所有的redo信息，直至到达最近的可恢复检查点为止。

对于节点重启，找到redo日志的末尾。

· 阶段**5**

如果这是首次启动，将创建SYSTAB_0和NDB$EVENTS内部系统表。

对于节点重启或首次节点重启：

o 节点包含在事务处理操作中。

o 将节点的方案与主服务器的方案进行比较，并与其同步。

o 对所收到的、来自本节点所在节点组内其他节点的、INSERT形式的数据进行同步。

o 在任何情形下，等待由仲裁程序判定的本地检查点操作的完成。

· 阶段**6**

更新内部变量。

· 阶段**7**

更新内部变量。

· 阶段**8**

在系统重启中，重建所有的索引。

· 阶段**9**

更新内部变量。

· 阶段**10**

在节点重启或首次节点重启的这一阶段，API可能会连接到节点，并接收事件。

· 阶段**11**

在节点重启或首次节点重启的这一阶段，将事件传递任务提交给加入簇的节点。新加入的节点负责将其主要数据传递给订方。

对于首次启动或系统重启，一旦该进程完成，将启用事务处理功能。对于节点重启或首次节点重启，启动进程的完成意味着节点现在能够成为事务协调程序。

17.6.2. “管理客户端”中的命令

除了中央配置文件外，还能通过管理客户端**ndb_mgm**提供的命令行界面对簇进行控制。这是运行簇的主要管理界面。

管理客户端提供了下述基本命令。在下面给出的清单中，*node_id*指的是数据库节点ID或关键字ALL，指明命令将应用到所有的簇数据节点上。

· HELP

显示关于所有可能命令的信息。

· SHOW

显示关于簇状态的信息。

注释：在使用多个管理节点的簇中，该命令仅显示与当前管理服务器实际相连的数据节点的信息。

· *node_id* START

启动由*node_id*标识的数据节点（或所有数据节点）。

- *node_id* STOP

停止由*node_id*标识的数据节点（或所有数据节点）。

- *node_id* RESTART [-N] [-I]

重启由*node_id*标识的数据节点（或所有数据节点）。

- *node_id* STATUS

显示由*node_id*标识的数据节点（或所有数据节点）的状态信息。

- ENTER SINGLE USER MODE *node_id*

进入单用户模式，仅允许由节点ID"*node_id*"标识的MySQL服务器访问数据库。

- EXIT SINGLE USER MODE

退出单用户模式，允许所有的SQL节点（即所有运行的**mysqld**进程）访问数据库。

- QUIT

中止管理客户端。

- SHUTDOWN

关闭除SQL节点之外的所有簇节点，并退出。

在下一节中，介绍了用于事件日志的命令。在这些议题的单独一节中，介绍了用于创建备份以及从备份中恢复的命令。

17.6.3. MySQL簇中生成的事件报告

17.6.3.1. 登记管理命令
17.6.3.2. 日志事件

在本节中，我们讨论了MySQL簇提供的事件日志的类型，以及记录的事件类型。

MySQL簇提供了两种事件日志。它们是**cluster log**和**node logs**，**cluster log**（簇日志）包括由所有簇节点生成的事件，**node logs**（节点日志）仅记录每个数据节点的本地事件。

由簇事件日志功能生成的输出可以有多个目的地，包括文件、管理服务器控制台窗口、或syslog。由节点事件日志功能生成的输出将被写入数据节点的控制台窗口。

可以对这两类事件日志进行设置，使之记录不同的事件子集。

注释：簇日志是为大多数使用场合推荐的日志，这是因为它在1个文件中提供了关于整个簇的日志信息。节点日志仅应在应用程序的开发过程中使用，或用于调试应用程序代码。

可根据三种不同的判据识别每个值得通报的事件：

- *Category*（类别）：可以是下述值之一：STARTUP, SHUTDOWN, STATISTICS, CHECKPOINT, NODERESTART, CONNECTION, ERROR,或INFO。
- *Priority*（优先级）：由从1到15的数字表示，"1"表示"最重要"，"15"表示"最不重要"。
- *Severity Level*（严重级别）：可以是下述值之一：ALERT, CRITICAL, ERROR, WARNING, INFO, 或DEBUG。

无论是簇日志还是节点日志，都能根据这些属性进行过滤。

17.6.3.1. 登记管理命令

下述管理命令与簇日志有关：

- CLUSTERLOG ON

打开簇日志。

- CLUSTERLOG OFF

关闭簇日志。

- CLUSTERLOG INFO

关于簇日志设置的信息。

- *node_id* CLUSTERLOG *category=threshold*

用小于或等于*threshold*的优先级将*category*事件记录到簇日志。

· CLUSTERLOG FILTER *severity_level*

将簇事件日志切换为指定的*severity_level*。

在下表中，介绍了簇日志类别阈值的默认设置（对于所有数据节点）。如果事件的优先级值低于或等于优先级阈值，就会在簇日志中记录。

注意，事件是按数据节点通报的，可在不同的节点上设置不同的阈值。

类别	默认阈值（所有数据节点）
STARTUP	**7**
SHUTDOWN	**7**
STATISTICS	**7**
CHECKPOINT	**7**
NODERESTART	**7**
CONNECTION	**7**
ERROR	**15**
INFO	**7**

阈值用于过滤每种类别中的事件。例如，对于优先级为3的STARTUP事件，不会将其记录到日志中，除非将STARTUP的阈值更改为3或更小。如果阈值为3，仅发送优先级等于或小于3的事件。

下面给出了事件的严重级别（注释：它们与Unix的syslog级别对应；但LOG_EMERG和LOG_NOTICE除外，未使用或未映射它们）：

1	ALERT	应立刻更正的状况，如损坏的系统数据库。
2	CRITICAL	临界状况，如设备错误或资源不足。
3	ERROR	应予以更正的状况，如配置错误等。
4	WARNING	不能称其为错误的状况，但仍需要特别处理。
5	INFO	通报性消息。
6	DEBUG	调试消息，用于NDB簇开发。

可以打开或关闭事件严重级别。如果打开了事件严重级别，那么优先级等于或低于类别阈值的事件均将被记录。如果关闭了事件严重级别，那么将不记录属于该严重级别的任何事件。

17.6.3.2. 日志事件

事件日志中记录的事件报告采用下述格式：*datetime* [*string*] *severity* – *message*。例如：

```
09:19:30 2005-07-24 [NDB] INFO -- Node 4 Start phase 4 completed
```

本节讨论了所有值得通报的事件，按类别以及每一类别中的严重级别排序。

CONNECTION事件

这类事件与簇节点之间的连接有关。

事件	优先级	严重级别	描述
DB节点已连接	8	INFO	数据节点已连接
DB节点断开连接	8	INFO	数据节点断开连接
通信关闭	8	INFO	SQL节点或数据节点的连接已关闭
通信打开	8	INFO	SQL节点或数据节点的连接已打开

CHECKPOINT事件

下面给出的日志消息与检查点有关。

（注释：GCP =全局检查点，LCP =本地检查点）。

事件	优先级	严重级别	描述
在calc keep GCI中，LCP已停止	0	ALERT	LCP已停止
本地检查点片段完成	11	INFO	片段上的LCP已完成
全局检查点完成	10	INFO	GCP完成
全局检查点启动	9	INFO	启动GCP：将REDO日志写入磁盘
本地检查点完成	8	INFO	LCP已正常完成
本地检查点启动	7	INFO	启动LCP：将数据写入磁盘
报告undo日志已封闭	7	INFO	UNDO日志已封闭：缓冲快要溢出

STARTUP事件

下述事件是在成功或失败的节点启动或簇启动时生成的。它们还提供了与启动进程进展状况有关的信息，包括与日志活动有关的信息。

事件	优先级	严重级别	描述
收到内部启动信号STTORRY	15	INFO	重启完成后收到的信息块
Undo记录已执行	15	INFO	
新的REDO日志已启动	10	INFO	GCI保持*X*，最新的可恢复GCI *Y*
新日志已启动	10	INFO	日志部分*X*，启动MB *Y*，停止MB *Z*
拒绝将节点纳入簇中	8	INFO	由于配置错误、无法建立通信、或其他问题，不能将节点包含在簇中。
DB节点邻居	8	INFO	显示附近的数据节点。
DB节点启动阶段X完成	4	INFO	数据节点启动阶段已完成。
节点已被簇成功接纳	3	INFO	显示节点，管理节点，以及动态ID。
DB节点启动阶段已开始	1	INFO	NDB簇节点正在启动。
DB节点的所有启动阶段已完成	1	INFO	NDB簇节点已启动。
DB节点关闭操作已启动	1	INFO	数据节点的关闭操作已开始
DB节点关闭操作失败	1	INFO	无法正常关闭数据节点。

NODERESTART事件

下述事件是在重启节点时产生的，并与节点重启进程的成功或失败相关。

事件	优先级	严重级别	描述
节点失败阶段完成	8	ALERT	通报节点失败阶段的完成
节点失败，节点状态为*X*	8	ALERT	通报节点已失败
通报仲裁程序结果	2	ALERT	对于仲裁尝试，有8种不同的可能结果： · 仲裁检查失败，剩余节点少于1/2。 · 仲裁检查成功，节点组多数 · 仲裁检查失败，丢失节点组 · 网络分区，要求仲裁 · 仲裁成功，来自节点*X*的正面回应 · 仲裁失败，来自节点*X*的负面回应 · 网络分区，无可用的仲裁程序 · 网络分区，未配置仲裁程序
完成了片段复制	10	INFO	
完成了目录信息复制	8	INFO	
完成了分配信息复制	8	INFO	
开始复制片段	8	INFO	
完成了所有片段的复制	8	INFO	
GCP接收已启动	7	INFO	
GCP接收已完成	7	INFO	
LCP接收已启动	7	INFO	
LCP接收已完成（状态= *X*）	7	INFO	
通报是否发现了仲裁程序	6	INFO	搜索仲裁程序时，有7种可能的结果： · 管理服务器重启仲裁线程[state=*X*] · 准备仲裁程序节点*X* [ticket=*Y*] · 接收仲裁程序节点*X* [ticket=*Y*] · 启动仲裁程序节点*X* [ticket=*Y*] · 丢失了仲裁程序节点*X* – 进程失败 [state=*Y*] · 丢失了仲裁程序节点*X* – 进程退出 [state=*Y*] · 丢失了仲裁程序节点*X* <error msg> [state=*Y*]

STATISTICS事件

下述事件具有统计特性。它们提供了相应的信息，如事务和其他操作的数目，低浓度节点发送或接收的数据量，以及内存使用率等。

事件	优先级	严重级别	描述

通报作业日程统计	9	INFO	平均的内部作业日程统计
发送的字节数	9	INFO	发送至节点*X*的平均字节数
接收的自己#	9	INFO	从节点*X*接收的平均字节数
通报事务统计	8	INFO	事务数目，提交次数，读取次数，简单读取次数，写入次数，并发操作数目。属性信息，以及放弃次数
通报操作	8	INFO	操作数目
通报表创建	7	INFO	
内存使用	5	INFO	数据内存和索引内存的使用率（80%、90%和100%）

ERROR事件

这些事件与簇错误和告警有关，如果出现1个或多个这类事件，表明出现了重大故障或失败。

事件	优先级	严重级别	描述
因失去心跳而死亡	8	ALERT	因失去心跳而声明节点*X*死亡。
传输器错误	2	ERROR	
传输器告警	8	WARNING	
失去心跳	8	WARNING	节点*X*失去心跳#*Y*
一般性告警事件	2	WARNING	

INFO事件

这些事件给出了关于簇状态和簇维护活动的一般信息，如日志和心跳传输等。

事件	优先级	严重级别	描述
发出心跳	12	INFO	将心跳发送至节点*X*
创建日志字节	11	INFO	日志部分，日志文件，MB
一般信息事件	2	INFO	

17.6.4. 单用户模式

采用单用户模式，数据库管理员能够将对数据库系统的访问限制在1个MySQL服务器（SQL节点）。进入单用户模式时，与所有其他MySQL服务器的所有连接均将恰当关闭，而且所有正在运行的事务均将被放弃。不允许启动任何新事务。

一旦簇进入单用户模式，只有指定的SQL节点才有权访问数据库。使用**ALL STATUS**命令，可查看簇进入单用户模式的时间。

示例：

```
NDB> ENTER SINGLE USER MODE 5
```

执行该命令而且簇进入单用户模式后，节点ID为5的SQL节点将成为簇中唯一允许的用户。

上述命令中指定的节点必须是MySQL服务器节点，如果指定任何其他类型的节点，将被拒绝。

注释：执行上述命令时，指定节点上所有正在运行的事务均将被放弃，连接关闭，而且必须重启服务器。

使用**EXIT SINGLE USER MODE**命令，能够将簇数据节点的状态从单用户模式更改为正常模式。对于等待连接的MySQL服务器（即，对于即将准备就绪并可用的簇），现在允许进行连接。在状态变化期间和变化之后，指定为单用户SQL节点的MySQL服务器将继续运行（如果仍连接的话）。

示例：

```
NDB> EXIT SINGLE USER MODE
```

运行在单用户模式下时，如果节点失败，推荐的处理方法是：

-

1. 结束所有的单用户模式事务。
2. 发出**EXIT SINGLE USER MODE**命令。
3. 重启簇的数据节点。

或

· 进入单用户模式之前重启数据库。

17.6.5. MySQL簇的联机备份

17.6.5.1. 簇备份概念
17.6.5.2. 使用管理服务器创建备份
17.6.5.3. 如何恢复簇备份
17.6.5.4. 簇备份的配置
17.6.5.5. 备份故障诊断与排除

在本节中，介绍了创建备份的方法，以及从备份恢复数据库的方法。

17.6.5.1. 簇备份概念

备份指的是在给定时间对数据库的快照。备份包含三个主要部分：

- **Metadata**（元数据）：所有数据库表的名称和定义。
- **Table records**（表记录）：执行备份时实际保存在数据库表中的数据。
- **Transaction log**（事务日志）：指明如何以及何时将数据保存在数据库中的连续记录。

每一部分均会保存在参与备份的所有节点上。在备份过程中，每个节点均会将这三个部分保存在磁盘上的三个文件中：

- BACKUP-*backup_id*.*node_id*.ctl

包含控制信息和元数据的控制文件。每个节点均会将相同的表定义（对于簇中的所有表）保存在自己的该文件中。

- BACKUP-*backup_id*-0.*node_id*.data

包含表记录的数据文件，它是按片段保存的，也就是说，在备份过程中，不同的节点会保存不同的片段。每个节点保存的文件以指明了记录所属表的标题开始。在记录清单后面有一个包含关于所有记录校验和的脚注。

- BACKUP-*backup_id*.*node_id*.log

包含已提交事务的记录的日志文件。在日志中，仅保存已在备份中保存的表上的事务。参与备份的节点将保存不同的记录，这是因为，不同的节点容纳了不同的数据库片段。

在上面所列的内容中，*backup_id*指的是备份ID，*node_id*是创建文件的节点的唯一ID。

17.6.5.2. 使用管理服务器创建备份

开始备份前，请确保已为备份操作恰当地配置了簇。（请参见17.6.5.4节，“簇备份的配置”）。

使用管理服务器创建备份包含以下步骤：

1. 启动管理服务器(**ndb_mgm**)。
2. 执行命令START BACKUP。
3. 管理服务器将用消息“指示备份开始”作出应答。这意味着管理服务器将请求提交给了簇，但尚未收到任何回应。
4. 管理服务器回复“备份*backup_id*开始”，其中，*backup_id*是该备份的唯一ID（如果未作其他配置，该ID还将保存在簇日志中）。这意味着簇已收到并开始处理备份请求。它不表示备份已完成。
5. 管理服务器发出消息“备份*backup_id*完成”，通知备份操作已结束。

要想放弃正在处理的备份：

1. 启动管理服务器。
2. 执行命令ABORT BACKUP *backup_id*。*backup_id*是当备份开始时包含在管理服务器应大中的备份ID（在消息“备份*backup_id*启动”中）。
3. 管理服务器用消息“放弃指示的备份*backup_id*”确认放弃请求，注意，尚未收到对该请求的实际回应。
4. 一旦放弃了备份，管理服务器将通报“备份*backup_id*因XYZ而放弃”。这意味着簇中止了备份，并从簇文件系统中删除了与该备份有关的所有文件。

在系统shell中使用下述命令，也能放弃正在执行的备份：

```
shell> ndb_mgm -e "ABORT BACKUP backup_id"
```

注释：执行放弃操作时，如果没有ID为*backup_id*的备份，管理服务器不会给出任何明确回应。但是，所发出的无效放弃命令将在簇日志中给出。

17.6.5.3. 如何恢复簇备份

簇恢复程序是作为单独的命令行实用工具**ndb_restore**实现的，它将读取由备份创建的文件，并将保存的信息插入数据库。必须为每组备份文件执行恢复程序，也就是说，执行次数与创建备份时运行的数据库节点数相同。

首次执行恢复程序时，还需要恢复元数据。换句话讲，必须重新创建数据库表（注意，开始执行恢复操作时，簇中应有一个空数据库）。恢复程序对于簇来说相当于API，因此，需要一个空闲连接，以便与簇相连。可使用**ndb_mgm**命令**SHOW**（在系统shell下使用**ndb_mgm -e SHOW**即可完成该操作）进行验证。可以使用开关“-c *connectstring*”来确定MGM节点的位置。（关于连接字符串的更多信息，请参见17.4.4.2节，“MySQL簇连接字符串”。备份文件必须位于恢复程序参量给定的目录下。

能够使用与创建是所用配置不同的配置将备份恢复到数据库。例如，对于备份ID为12的备份，该备份是在具有两个数据库节点（节点ID无恶2和3）的簇中创建的，可以将其恢复到具有4个节点的簇中。这样，**ndb_restore**必须运行两次，为创建备份时的每个数据库节点运行一次。

注释：对于快速恢复，能够以并行方式恢复数据，但必须有足够的可用簇连接。然而，数据文件必须始终前于日志文件。

17.6.5.4. 簇备份的配置

有4个用于备份的基本配置参数：

· BackupDataBufferSize

将数据写入磁盘之前用于对数据进行缓冲处理的内存量。

· BackupLogBufferSize

将日志记录写入磁盘之前用于对其进行缓冲处理的内存量。

· BackupMemory

在数据库节点中为备份分配的总内存。它应是分配给备份数据缓冲的内存和分配给备份日志缓冲的内存之和。

· BackupWriteSize

写入磁盘的块大小。它适用于备份数据缓冲和备份日志缓冲

关于这些参数的更多细节，请参见17.4节，“MySQL簇的配置”。

17.6.5.5. 备份故障诊断与排除

如果在发出备份请求时返回了错误代码，最可能的原因是内存不足，或磁盘空间不足。应检查是否为备份分配了足够的内存。此外，还应检查在备份目标的硬盘分区上是否有足够的空间。

NDB不支持重复的读取操作，对于恢复进程，这类操作会导致问题。尽管备份进程“hot”，但从备份恢复MySQL簇并不是100％“hot”的进程。其原因在于，在恢复进程执行期间，正在运行的事务会从已恢复的数据中获取不可重复的读。这意味着，当恢复正在进行时，数据的状态是不一致的。

17.7. 使用与MySQL簇的高速互连

17.7.1. 配置MySQL簇以使用SCI套接字
17.7.2. 理解簇互连的影响

即使在1996年开始NDB簇的设计之前，在创建并行数据库的过程中遇到的一个主要问题显然是网络中节点之间的通信问题。正因为如此，从开始设计时，NDB簇就允许使用多种不同的数据传输机制。在本手册中，我们使用了术语传输器。

目前，MySQL簇代码库包含对四种不同传输器的支持。目前的大多数用户采用的是以太网上的TCP/IP，这是因为它无处不在。它也是迄今为止在MySQL簇中经过最佳测试的传输器。

我们正在努力，以确保与**ndbd**进程的通信是尽可能大的“组块”，这是因为，它有利于所有的数据传输类型。

对于需要它的用户，也能使用簇互联以进一步增强性能。实现它的方式有两种：或使用能处理该情况的定制传输器，或使用能绕过TCP/IP堆栈的套接字实施方案。我们使用由Dolphin开发的SCI（规模可扩展的计算机连接接口）技术，对这两类技术进行了试验。

17.7.1. 配置MySQL簇以使用SCI套接字

在本节中，我们将介绍如何改编为常规TCP/IP通信配置的簇，以使用SCI套接字。本文档基于2004年10月1日发布的SCI套接字2.3.0版。

前提条件

对于任何打算使用SCI套接字的机器，都必须配备SCI卡。

能够与任何版本的MySQL簇一起使用SCI套接字。不需要特殊创建，这是因为它采用了MySQL簇中已提供的正常套接字调用。但是，目前仅在Linux 2.4和2.6内核上才支持SCI套接字。尽管到目前为止我们仅在Linux 2.4上核实了这点，在一些其他操作系统上也成功测试了SCI传输器。

对于SCI套接字，有四种基本要求：

1. 创建SCI套接字库。
2. 安装SCI套接字内核库。
3. 安装1个或2个配置文件。
4. 必须为整个机器或启动MySQL簇进程的shell启用SCI套接字库。

对于打算将SCI套接字用于节点间的通信的簇，需要为簇中的每台机器重复该进程。

要想使SCI套接字工作，需要获得两个软件包：

1. 包含用于SCI套接字库的DIS支持库的源码软件包。
2. 用于SCI套接字库本身的源码软件包。

目前，仅以源码格式提供了它们。编写本文档时可用的最新版本的软件包分别

是DIS_GPL_2_5_0_SEP_10_2004.tar.gz和SCI_SOCKET_2_3_0_OKT_01_2004.tar.gz。在下述站点，可找到它们（也可能找到更新的版本）：http://www.dolphinics.no/support/downloads.html。

软件包安装

获得库软件包后，接下来应将它们解包到恰当的目录下，将SCI套接字库解包到DIS代码下的目录中。然后，需要创建库。在下面的示例中，给出了在Linux/x86平台上执行该任务所需的命令：

```
shell> tar xzf DIS_GPL_2_5_0_SEP_10_2004.tar.gz
shell> cd DIS_GPL_2_5_0_SEP_10_2004/src/
shell> tar xzf ../../SCI_SOCKET_2_3_0_OKT_01_2004.tar.gz
shell> cd ../adm/bin/Linux_pkgs
shell> ./make_PSB_66_release
```

能够为64位处理器创建这些库。要想为使用64位扩展的Opteron CPU创建这些库，请运行**make_PSB_66_X86_64_release**而不是**make_PSB_66_release**。如果创建工作是在Itanium机器上进行的，应使用**make_PSB_66_IA64_release**。X86-64变体应能与Intel EM64T体系结构一起工作，但这点尚未测试（就我们所知）。

完成创建进程后，在zipped tar文件中可发现已编译好的库，文件的名称符合DIS-*<operating-system>-time-date*。现在，可将软件包安装到恰当位置。在本例中，我们使用的安装目录是/opt/DIS（注释：你很可能需要以系统根用户的身份运行下述命令）：

```
shell> cp DIS_Linux_2.4.20-8_181004.tar.gz /opt/
shell> cd /opt
shell> tar xzf DIS_Linux_2.4.20-8_181004.tar.gz
shell> mv DIS_Linux_2.4.20-8_181004 DIS
```

网络配置

至此，所有的库和二进制文件均已安装到了恰当的位置，还需确保SCI卡在SCI的地址空间范围内拥有恰当的节点ID。

进行后面的操作前，还需确定网络结构。对于该情形，有三种可使用的网络结构：

- 一维单环网络。
- 1个或多个SCI交换器，每个交换器端口有1个环形网。
- 2维或3维环面网。

每类拓扑结构均有自己的、用于提供节点ID的方法。下面给出了简要讨论。

单环网采用的节点ID是4的非0倍数，4，8，12…。

下一个可行的选择是SCI交换器。1个SCI交换器有8个端口，每个端口都能支持1个环形网。有必要确保不同的环形网使用了不同的节点ID空间。对于典型的配置，第1个端口使用的节点ID低于64（4～60），接下来的64个节点ID（68～124）指定给下一个端口，依此类推，节点ID 452～508将被指定给第8个端口。

对于2维或3维环面网结构，每个节点位于各维中，在第1维中，各节点的增量为4，在第2维中，增量为64，在第3维中（如果适用的话），增量为1024。关于更详尽的文档，请参见Dolphin的网站。

在我们的测试中，采用了交换器方式，但大多数大型簇安装应用使用的是2维或3维环面网结构。采用交换器方式具有的优点是：使用双SCI卡和双交换器，能够相对容易地创建冗余网络，SCI网络上的平均故障切换时间在100毫秒量级。对于SCI套接字实施，MySQl簇中的SCI传输器支持该结构，而且它也处于发展当中。

对于2D/3D环面网，故障切换也是可能的，但需要向所有节点发送发出新的路由索引。然而，这仅需要100毫秒左右就能完成，对于大多数高可用性情形，这是可接受的。

通过将簇数据节点恰当地置于交换式体系结构中，能够使用2个交换器来构建结构，将16台计算机互联在一起，任何单点故障均不会妨碍1台以上的计算机。对于32台计算机和2台交换器，也能以这样的方式配置簇，任何单点故障均不会导致2个以上的节点丢失，在此情况下，也能知道那一对节点收到影响。因此，通过将两个节点置于不同的节点组，就能创建"安全的"MySQL簇。

要想为SCI卡设置节点ID，可使用/opt/DIS/sbin目录中的下述命令。在本例中，"-c 1"指的是SCI卡的编号（如果在机器上只有1块卡，它总为1），"-a 0"指的是适配器0，"68"是节点ID：

```
shell> ./sciconfig -c 1 -a 0 -n 68
```

如果在同一台机器上有多块SCI卡，使用下述命令（假定当前的工作目录是/opt/DIS/sbin），可确定哪块卡使用哪个插槽：

```
shell> ./sciconfig -c 1 -gsn
```

它将给出SCI卡的序列号。然后用"-c 2"重复该步骤，依此类推（对机器上的每块卡）。一旦将每块卡与1个插槽匹配后，可为所有卡设置节点ID。

安装了必要的库和二进制文件，并设置了SCI节点ID后，下一步是设置从主机名（或IP地址）到SCI节点ID的映射。该任务可在SCI套接字配置文件中完成，该文件应被保存为/etc/sci/scisock.conf。在该文件中，通过恰当的SCI卡将SCI节点映射到与之通信的主机名或IP地址。这里给出了一个很简单的配置文件示例：

```
#host           #nodeId
```

```
alpha           8
beta            12
192.168.10.20   16
```

也能对配置进行限制，使其仅应用在这些主机的可用端口子集上。也能使用额外的配置文件/etc/sci/scisock_opt.conf完成该设置，如下所示：

```
#-key                        -type        -values
EnablePortsByDefault                yes
EnablePort                  tcp           2200
DisablePort                 tcp           2201
EnablePortRange             tcp           2202 2219
DisablePortRange            tcp           2220 2231
```

驱动程序安装

创建好了配置文件后，可安装驱动程序。

首先应安装底层驱动，然后安装SCI套接字驱动：

```
shell> cd DIS/sbin/
shell> ./drv-install add PSB66
shell> ./scisocket-install add
```

如果愿意，可调用脚本来核实SCI套接字配置文件中的所有节点是否均能访问，通过该方式检查安装：

```
shell> cd /opt/DIS/sbin/
shell> ./status.sh
```

如果发现错误并需要更改SCI套接字配置，需要使用**ksocketconfig**来完成该任务：

```
shell> cd /opt/DIS/util
shell> ./ksocketconfig -f
```

测试设置

为了确保SCI套接字已实际使用，可使用**latency_bench**测试程序。使用该实用工具的服务器组件，客户端能够连接服务器以测试连接的等待时间，通过观察等待时间，可方便地判定是否启用了SCI。（注释：使用**latency_bench**之前，需要设置环境变量LD_PRELOAD，请参见本节后面的介绍）。

要想设置服务器，可使用下述命令：

```
shell> cd /opt/DIS/bin/socket
shell> ./latency_bench -server
```

要想运行客户端，再次使用**latency_bench**，但此时采用"-client"选项：

```
shell> cd /opt/DIS/bin/socket
shell> ./latency_bench -client server_hostname
```

到目前为止，应完成了SCI套接字配置，而且MySQL簇已做好了使用SCI套接字和SCI传输器的准备（请参见17.4.4.10节，"MySQL簇SCI传输连接"）。

启动簇

该进程接下来的步骤是启动MySQL簇。要想启用SCI套接字，在启动**ndbd**、**mysqld**和**ndb_mgmd**之前，需要设置环境变量LD_PRELOAD。该变量应指向用于SCI套接字的内核库。

要想在bash shell下启动**ndbd**，可执行下述命令：

```
bash-shell> export LD_PRELOAD=/opt/DIS/lib/libkscisock.so
bash-shell> ndbd
```

在tcsh环境下，可使用下述命令完成相同的任务：

```
tcsh-shell> setenv LD_PRELOAD=/opt/DIS/lib/libkscisock.so
tcsh-shell> ndbd
```

注释：MySQL簇可以仅使用SCI套接字的内核变体。

17.7.2. 理解簇互连的影响

ndbd进程有很多简单的结构，它们可用于访问MySQL簇中的数据。我们创建了一个十分简单的基准测试来检查它们的性能，以及各种互联方式对其性能的影响。

有四种访问方法：

· 主键访问

通过其主键简单地访问记录。在最简单的情况下，一次仅访问一条记录，这意味着该单一请求需负担设置众多TCP/IP消息的全部开销以及众多的场景切换开销。对于在一批中发出多个主键访问请求的情况，这些访问请求将分担设置必要TCP/IP消息和场景切换的开销。如果TCP/IP消息是面向不同目标的，还须设置额外的TCP/IP消息。

· 唯一键访问

唯一键访问与主键访问类似，差别在于，唯一键访问是作为在索引表上的读取操作执行的，然后是作用在表上的主键访问。但是，MySQL服务器仅发送一条请求，而且对索引表的读取是由**ndbd**负责处理的。这类请求也受益于批处理法。

· 全表扫描

对于表查询，当不存在索引时，将执行全表扫描。这将作为单一请求被发送至**ndbd**进程，该进程随后将表扫描分为在所有簇ndbd进程上进行的一组并行扫描。在未来的MySQL簇版本中，SQL节点能够过滤某些扫描。

· 使用有序索引的范围扫描

使用有序索引时，它将使用与全表扫描相同的方式执行扫描，不同之处在于，它仅扫描位于MySQL服务器（SQL节点）所传输查询使用的范围内的记录。当所有绑定的索引属性包含分区 键中的所有属性时，将以并行方式扫描所有分区。

为了检查这些访问方法的基本性能，我们开发了一组基准。作为这类基准之一，**testReadPerf**能够测试简单的和批处理式主键和唯一键访问。通过发出返回单一记录的扫描请求，该基准还能测量范围扫描的设置开销。此外，还有1个该基准的变体，它采用了范围扫描来获取批量记录。

这样，我们就能确定单键访问的开销，以及单记录扫描访问的开销，并能根据基本访问方法测量通信媒介的影响。

在我们进行的测试中，为采用TCP/IP套接字的正常传输器和使用SCI套接字的类似设置进行了基本设置。下面给出的数字是关于少量访问的，每次访问20条记录。使用2KB的记录时，串行访问和批式访问之间的差异将降低3～4倍。对于2KB的记录，未测试SCI套接字。测试是在下述簇上进行的，该簇有两个数据节点，这两个数据节点运行在2台双CPU机器上，处理器为AMD MP1900+。

Access type:	TCP/IP sockets	SCI Socket
Serial pk access:	400 microseconds	160 microseconds
Batched pk access:	28 microseconds	22 microseconds
Serial uk access:	500 microseconds	250 microseconds
Batched uk access:	70 microseconds	36 microseconds
Indexed eq-bound:	1250 microseconds	750 microseconds
Index range:	24 microseconds	12 microseconds

我们还执行了另一组测试，以检查SCI套接字的性能以及SCI传输器的性能，并将这两者与TCP/IP传输器的性能进行了比较。所有测试均采用了主键访问方法，或采用串行多线程方式，或采用多线程批方式。

测试结果表明，SCI套接字比TCP/IP约快100％。与SCI套接字相比，在大多数情况下，SCI传输器的速度更快。在具有很多线程的测试程序中出现了1个值得关注的情况，它表明，与**mysqld**进程一起使用时，SCI传输器的执行性能并不很好。

我们得出的总体结论是，对于大多数基准，与TCP/IP相比，除了罕见的通信性能不成其为关注事宜的情况外，使用SCI套接字能将性能提升约100％。当扫描过滤器占用大多数处理时间，或当执行大量的主键访问，就会出现该情况。在这类情况下，ndbd进程中的CPU处理操作将占用开销的相当大部分。

对于使用SCI传输器来取代SCI套接字，它仅对ndbd进程之间的通信有意义。如果CPU能专门用于ndbd进程，使用SCI传输器也也很有意义，这是因为SCI传输器能确保该进程永不休息。另一个重要特性是，它能确保对ndbd进程的优先级进行特定的设置，即使运行了很长的时间，其优先级也不会丧失，就像在Linux 2.6中通过将进程锁定到CPU而能实现的那样。如果这类配置是可能的，与使用SCI套接字相比，ndbd进程将获得10～70％的性能提升（执行更新以及可能的并行扫描操作时，性能提升较大）。。

对于计算机簇，还有数种其他的优化套接字实施方案，包括Myrinet、吉比特以太网、Infiniband（无限带宽）和VIA接口。迄今为止，我们仅与SCI套接字一起测试了MySQL簇。我们还提供了上述文档，其中，介绍了如何使用针对MySQL簇的常规TCP/IP来设置SCI套接字的方法。

17.8. MySQL簇的已知限制

在本节中，我们列出了5.1.x系列中对MySQL簇的一些已知限制，并与使用MyISAM和InnoDB存储引擎时可用的特性进行了比较。目前，尚不打算在即将推出的MySQL 5.1版本中处理这些问题，但是，我们将在后续的版本系列中，提供这方面的补丁和修正。如果你检查了MySQL缺陷数据库（http://bugs.mysql.com）中的"簇"类别，可发现我们打算在即将推出的MyAQL 5.1版中更正的已知缺陷（如果标记为"5.1"的话）。

（注释：在本节末尾，列出了在当前版本中已解决的MySQL 5.0簇中的一些事宜）。

· 语法中的不兼容性（运行已有的应用程序时导致错误）：

o 不支持文本索引。

o 不支持Geometry数据类型（WKT和WKB）。

· 限制或行为中的不兼容性（运行已有的应用程序时可能导致错误）：

o 没有事务的部分回滚功能。重复键或类似错误会导致整个事务的回滚。

o 存在很多可配置的硬限制，但能使用簇中的可用主内存设置限制。关于配置参数的完整清单，请参见17.4.4节，"配置文件"。大多数配置参数可在线升级。这些硬限制包括：

§ 数据库内存大小和索引内存大小（分别是DataMemory和IndexMemory）。

§ 对于能够执行的最大事务数，可使用配置参数MaxNoOfConcurrentOperations进行设置。注意批量加载，TRUNCATE TABLE和ALTER TABLE是通过运行多个事务作为特殊情况进行处理的，因而不受该限制的约束。

§ 与表和索引有关的不同限制。例如，每表的最大有序索引数是由MaxNoOfOrderedIndexes确定的。

o 在NDB表中，数据库名称、表名称和属性名称不能与其他表处理程序中的一样长。属性名称将被截短至31个字符，截短后如果不是唯一的，将导致错误。数据库名称和表名的总最大长度为122个字符（也就是说，NDB簇表名的最大长度为122个字符减去该表所属的数据库的名称中的字符数）。

o 所有的簇表行具有固定长度。这意味着（例如），如果表中有仅包含相对较小值的1个或多个VARCHAR字段，与使用MyISAM引擎的相同表和数据相比，使用NDB存储引擎时需要更多的内存和磁盘空间。换句话讲，对于VARCHAR列，它所需的存储空间与具有相同大小的CHAR列所需的相同。

o 簇数据库中的最大表数目限制为1792。

o 每表的最大属性数限制为128。

o 任一行的最大允许大小为8K，不包括保存在*BLOB*列中的数据。

o 每键的最大属性数为32。

· 不支持的特性（不会导致错误，但不被支持或强制）：

o 外键结构将被忽略，就像在MyISAM表中那样。

o 保存点以及对保存点的回滚将被忽略，就像在MyISAM中那样。

· 性能以及与限制有关的事宜

o 由于对NDB存储引擎的连续访问，存在查询性能问题，与MyISAM或InnoDB的情形相比，执行很多范围扫描时，开销相对昂贵。

o 不支持范围统计中的记录，在某些情况下，这会造成非最优查询计划。可采用USE INDEX或FORCE INDEX规避该问题。

o 对于使用USING HASH创建的唯一性哈希索引，如果NULL是键的一部分，不能使用这类索引访问表。

o MySQL簇不支持磁盘上的持续提交。提交将被复制，但不保证在提交时会将日志写入磁盘。

▪ 丢失特性：

o 唯一支持的隔离级别是READ_COMMITTED（InnoDB支持READ_COMMITTED、READ_COMMITTED、REPEATABLE_READ和SERIALIZABLE）。关于其如何影响簇数据库备份和恢复的更多信息，请参见17.6.5.5节，“备份故障诊断与排除”。

o 不支持磁盘上的持续提交。提交将被复制，但不保证在提交时会将日志写入磁盘。

· 与多**MySQL**服务器有关的问题（与MyISAM或InnoDB无关）：

o 运行多个MySQL服务器时，ALTER TABLE未完全锁定（无分布式表锁定）。

o 如果在多个MySQL服务器上进行了更新，MySQL复制功能不能正确处理。但是，如果数据库分区方案是在应用级别上完成的，而且在这些分区上非发生事务，那么可使复制功能正确工作。

o 对于访问相同MySQL簇的多MySQL服务器，不支持数据库的自动发现（autodiscovery）功能。但是，在情况下，支持对表的自动发现（autodiscovery）功能。这意味着，创建了名为*db_name*的数据库后，或使用1个MySQL服务器导入了该数据库后，应在访问相同MySQL簇的每个额外MySQL服务器上发出CREATE DATABASE *db_name*语句（从MySQL 5.0.2开始，你还能使用CREATE SCHEMA *db_name;*）。一旦对给定MySQL服务器完成了该操作，服务器应能检测到数据库表，而不产生错误。

· 仅与**MySQL**簇有关的事宜（与MyISAM或InnoDB无关）：

o 簇中使用的所有机器必须具有相同的体系结构，也就是说，所有承载节点的机器必须是big-endian或little-endian，不能混合使用这两者。例如，不能用运行在PPC上的管理节点指挥运行在x86机器上的数据节点。该限制不适用于简单运行**mysql**或其他客户端（可能会访问簇的SQL节点）的机器。

o 不能像使用ALTER TABLE或CREATE INDEX完成的那样执行在线方案更改，这是因为NDB簇不支持这类变更的自动检测（但是，能够导入或创建使用不同存储引擎的表，然后使用“ALTER TABLE *tbl_name* ENGINE=NDBCLUSTER;”将其转换为NDB。在这类情况下，需要发出FLUSH TABLES命令，强制簇发现变化）。

o 不能在线增加或舍弃节点（此时必须重启簇）。

o 使用多个管理服务器时：

§ 必须在连接字符串中明确为节点指定ID，这是因为，在多个管理服务器上，自动分配的节点ID不能正常工作。

§ 必须十分小心，确保所有的管理服务器具有相同的配置。簇对此方面不进行任何特殊检查，

§ 要想使管理节点能发现彼此的存在，创建了簇后必须重启所有的数据节点（详细解释请参见Bug #13070）。

o 不支持用于数据节点的多个网络接口。如果使用了这类接口，很容易导致问题，原因在于，在出现某一数据节点失败的情况下，SQL节点将等待以确认该数据节点是否出现问题，但由于该数据节点的另一路径仍保持打开状态，SQL节点永远不会收到该确认信息。这会导致簇无法工作。

- 数据节点的最大数目为48。
- MySQL簇中总的最大节点数为63。在该数值中包括所有的MySQL服务器（SQL节点），数据节点和管理服务器。

考虑到本节开始时设定的条件，我们力图使上面列出的信息尽可能完全。你可以将任何遇到的差异通报到MySQL缺陷数据库，http://bugs.mysql.com/。如果我们不打算在MySQL 5.1中更正该问题，我们会将其添加到列表中。

17.9. MySQL簇发展的重要历程

17.9.1. MySQL 5.0中的MySQL簇变化
17.9.2. 关于MySQL簇的MySQL 5.1发展历程

在本节，我们讨论了MySQL 5.1在MySQL簇实施中的一些变化，并与MySQL 5.0进行了比较。我们还将讨论对MySQL簇的进一步重大改进，目前是为MySQL 5.2规划的。

在MySQL 5.0和5.1中，NDB簇存储引擎实施方案之间的变化相对较少，因此其升级相对快捷和简单。

为MySQL簇开发的主要新特性均归入了MySQL 5.1树。在本节中，我们还提供了一些提示，介绍了在MySQL 5.1中簇特性可能包含的方面（请参见17.9.2节，“关于MySQL簇的MySQL 5.1发展历程”）。

17.9.1. MySQL 5.0中的MySQL簇变化

MySQL 5.0.3-beta以及更新的版本包含一些有趣的新特性：

- 下推条件：一种查询，如：

```
SELECT * FROM t1 WHERE non_indexed_attribute = 1;
```

它将使用全表扫描，并会在簇的数据节点内评估条件。因此，不需要在网络中发送用于评估的记录（也就是说，采用函数传输而不是数据传输）。对于这种查询类型，其速度快5～10倍。请注意，模目前该特性是默认禁止的（有待更彻底的测试），但在某些情况下也能良好工作。使用命令“SET engine-condition-pushdown=On; command”可启用该特性。作为可选方式，你也可以用新的“--engine-condition-pushdown”选项标志启动MySQL服务器，运行启用了该特性的**mysqld**。

可以使用EXPLAIN来判定在何时使用了下推条件。

该变化的一项主要优点在于，能够以并行方式执行查询。这意味着，与以往相比，对非索引列的查询要快5～10倍，乘以数据节点的数目。原因在于，多个CPU能以并行方式执行查询。

- 降低了**IndexMemory**使用：在MySQL 5.1中，每条记录约消耗25字节的索引内存，每个唯一索引每记录将使用25字节的索引内存（不含某些数据内存，这是因为它们保存在单独表中）。这是因为在索引内存中，不再需要保存主键。
- 为**MySQL**簇启用了查询高速缓冲：关于配置和使用查询高速缓冲的更多信息，请参见5.13节，“MySQL查询高速缓冲”。
- 新的优化：有一个需要给予特别关注的优化，即，目前在某些查询中使用了批式读取接口。例如，考虑下述查询：

```
SELECT * FROM t1 WHERE primary_key IN (1,2,3,4,5,6,7,8,9,10);
```

与以往的MySQL簇版本相比，该查询的执行速度快2～3倍，原因在于，全部10个 键查找是在1个批次中发出的，而不是一次发送一个。

- 限制元数据对象的数目：在MySQL 4.1中，每个簇数据库最多可能包含1600个元数据对象，包括数据库表，系统表，索引和BLOB。在MySQL 5.0中，我们希望将该数值增加到20320。我们打算在2005年年中发布的MySQL 5.0.6 beta版中实现该增强特性。

17.9.2. 关于MySQL簇的MySQL 5.1发展历程

这里所介绍的内容是一份状态报告，它基于最近提交到MySQL 5.1源码树的内容。请注意，所有的5.1开发活动均可能随时变化。

目前，有四种为MySQL 5.1开发的主要新特性：

1. 将**MySQL**簇集成到了**MySQL**复制中：这样，就能从簇中的任何MySQL服务器执行更新，并由簇中的1个MySQL服务器处理MySQL复制，并保持了从服务器端安装的一致性。

2. 支持基于磁盘的记录：将支持磁盘上的记录。对于包含主键哈希索引的有索引字段，必须仍保存在RAM中，但可以将所有其他字段保存在磁盘上。

3. 大小可变的记录：定义为VARCHAR(255)的列目前使用260字节的存储空间，与任何特定记录中保存的内容无关。在MySQL 5.1簇表中，只保存被记录实际占用的字段部分。这样，在很多情况下，能够将这类列的空间需求量降低5倍。

4. 用户定义的分区功能：用户能够根据主关键的字段部分定义分区。MySQL服务器能够发现是否能将某些分区从WHERE子句中删除。能够根据KEY、HASH、RANGE和LIST处理程序执行分区操作和子分区操作。对于很多其他处理程序，也应能使用该特性。

此外，对于簇表中包含除BLOB或TEXT外的列类型的行，我们正准备增大8K的大小限制值。这是因为，行目前具有固定的大小，页大小为32768字节（减去用于行标题的128字节）。在目前，这意味着，如果允许每记录占用的大小超过8K，剩余的空间将为空（多达14000字节）。在MySQL 5.1中，我们打算更正该限制，从而使得在给定行中使用8K以上的空间不会导致页剩余部分的浪费。

17.10. MySQL簇常见问题解答

· 使用簇与使用复制的区别是什么？

在复制设置中，主MySQL服务器负责更新1个或多个从服务器。事务按顺序提交，较慢的事务会导致从服务器滞后于主服务器。这意味着，如果主服务器失败，从服务器可能无法记录最近的少数事务。如果使用了事务安全引擎，如InnoDB，要末是在从服务器上完成事务，要末是根本就不记录事务，但复制不保证主服务器和从服务器上的所有数据在任何时候都是一致的。在MySQL簇中，所有的数据节点保持同步，任何一个数据节点提交的事务将被提交给所有的数据节点。当某一数据节点失败时，所有剩余的数据节点仍能保持一致的状态。

简而言之，尽管标准的MySQL复制是异步的，但MySQL簇是同步的。

我们计划在MySQL 5.1中为簇实现（异步）复制功能。包括在两个簇之间，以及MySQL簇和非簇类MySQL服务器之间的复制能力。

· 为了运行簇，我是否需要进行特殊的组网呢？（簇中的计算机是如何通信的？）

MySQL簇是为高带宽环境下的使用而设计的，计算机通过TCP/IP相连。其性能直接取决于簇计算机之间的连接速度。对于簇，最低连接要求包括：典型的100MB以太网网络或等效网络。如果可能，建议使用GB以太网。

也支持更快的SCI 协议，但需要特殊硬件。关于SCI的更多信息，请参见17.7节，“使用与MySQL簇的高速互连”。

· 运行簇需要多少台计算机？为什么？

要想运行可行的簇，最少需要3台计算机。但在MySQL簇中，推荐的最低计算机数目为4：1台负责运行管理节点，1台负责运行SQL节点，2台用作存储节点。使用2个数据节点的目的是为了提供冗余性，管理节点必须运行在单独的机器上，这样，当1个数据节点失败时，仍能保证连续的仲裁服务。

· 簇中不同计算机所负责的任务是什么？

MySQL簇包含物理和逻辑结构，计算机是其物理部件。簇的逻辑或功能部件称为节点，容纳簇节点的计算机有时也称为簇主机。在理想情况下，每台簇主机将有1个节点，但在单个簇主机上可运行多个节点。簇中有三种节点，每一种节点均对应特定的角色。它们是：

1. 管理节点（**MGM**节点）：为整个簇提供管理服务，包括启动、停止、备份、以及为其他节点配置数据。管理节点服务器是作为应用程序**ndb_mgmd**实现的，用于通过MGM节点控制MySQL簇的管理客户端是**ndb_mgm**。

2. 数据节点：保存和复制数据。数据节点的功能由NDB数据节点进程ndbd的实例负责处理。

3. **SQL**节点：这是用“**--ndb-cluster**”选项启动的MySQL服务器（**mysqld**）的一个实例。

· 用什么操作系统才能使用簇呢？

在MySQL 5.1中，在Linux、Mac OS X和Solaris平台上均正式支持MySQL簇。我们正在努力为其他平台增加簇支持，包括Windows，我们的目标是，最终在支持MySQL本身的所有平台上实现MySQL簇。

在其他操作系统上运行簇也是可能的。用户告诉我们，他们在FreeBSD平台上超过运行了簇。但是，除了前面介绍的三种操作系统外，在其他平台上运行的簇应被视为alpha软件（最好情况），不保证在生产环境下的可靠性，而且不被*MySQL AB*支持。

· 运行*MySQL*簇的硬件要求是什么？

在簇运行的任何平台上，应具有具备NDB功能的二进制文件。显而易见，更快的CPU和更多的内存能够改善性能，64位CPU的效率很可能高于32位处理器。在用于数据节点的机器上必须有足够的内存，以便容纳各节点共享的数据库部分。（更多信息，请参见我需要多少内存？）。节点能通过标准的TCP/IP网络和硬件进行通信。对于SCI支持，需要特殊的组网硬件。

· 由于*MySQL*簇使用了*TCP/IP*，这是否意味着我能在*Internet*上运行*1*个或多个节点位于远程位置的簇？

请记住，在MySQL簇中，节点间的通信并不安全，这点极其重要，这类通信未加密，也未采用任何防护机制。对于簇，最安全的配置是位于防火墙后的专用网络，不能从外部直接访问任何簇数据或管理节点（对于SQL节点，应采取相同的防护措施，就像在MySQL服务器的其他实例中那样）。

无论是任何情况，在这类条件下簇的可靠运行十分令人怀疑，这是因为设计和实施MySQL簇时，假定它运行在能保证专用高速连通性的条件下，如使用100MB或GB以太网的LAN中（更倾向于后者）。对于低于该要求的任何环境，我们未作任何测试，也不保证其性能。

· 要想使用簇，我是否不得不学习新的编程语言或查询语言？

不。尽管使用了一些专用命令来管理和配置簇本身，但对于下述操作，仅需要标准的(My)SQL查询和命令：

o 创建、更改和撤销表。

o 插入、更新和删除表数据。

o 创建、更改和撤销主索引和唯一索引。

o 配置和管理SQL节点（MySQL服务器）。

· 使用簇时，如何了解错误或警告消息的含义呢？

有两种完成它的方式：

1. 出现错误或告警状况时，在MySQL监视器内，立刻使用**SHOW ERRORS**或**SHOW WARNINGS**。也能在MySQL Query Browser（查询浏览器）中显示它们。

2. 在系统shell提示符下，使用**perror --ndb** *error_code*。

· *MySQL*簇是事务安全的吗？支持什么隔离级别？

是。对于用NDB存储引擎创建的表，支持事务。在MySQL 5.1中，簇仅支持READ_COMMITTED事务隔离级别。

· 簇支持的表类型是什么？

NDB是仅有的支持簇功能的MySQL存储引擎。

（能够使用其他存储引擎在用于簇的MySQL服务器上创建表，如MyISAM或InnoDB，但这类非NDB表不在簇中）。

· *"NDB"*的含义是什么？

它是"**N**etwork **D**ata**b**ase"（网络数据库）的缩写。

· 哪些版本的*MySQL*软件支持簇？我必须从源码编译吗？

在5.1系列的所有MySQL-max二进制版本中均支持簇，但下面介绍的除外。使用命令SHOW VARIABLES LIKE 'have_%';或SHOW ENGINES;，可检查你的服务器是否支持NDB（更多信息，请参见5.1.2节，"mysq**ld-max**扩展**My**SQL服务器"）。

Linux用户请注意，NDB未包含在标准的MySQL服务器RPM中。对于NDB存储引擎以及所附的管理工具和其他工具，有单独的RPM软件包。请参见MySQL 5.1下载页上的NDB RPM下载部分（以前，你必须使用以.tar.gz档案方式提供的"-max"二进制文件。目前仍能这样，但却不是必须的，如果愿意，可使用Linux分发版的RPM管理器）。通过从源码编译-max二进制文件，也能获得NDB支持，但使用MySQL簇时，不需要这样。要想只下载最新的MySQL 5.1系列的二进制版本、RPM、或源码分发版，请访问http://dev.mysql.com/downloads/mysql/5.1.html。

· 我需要多少*RAM*？是否能全部使用磁盘内存？

在目前，簇是仅"内存中"的。这意味着所有的表数据（包括索引）均保存在RAM中。因此，如果你的数据占用了1GB的空间，而且打算在簇中将它复制1次，需要2GB的内存。还应加上操作系统以及在簇计算机上运行的应用程序所需的内存。

可以使用下述公司粗略估计簇中每个数据节点所需的RAM量：

```
(SizeofDatabase * NumberOfReplicas * 1.1 ) / NumberOfDataNodes
```

要想更准确地计算内存需求量，需要为簇数据库中的每个表确定每行所需的存储空间（详情请参见11.5节，"列类型存储需求"），然后乘以占行数。还必须对列索引进行计算，方式如下：

o 对于为NDB簇表创建的每个主键索引或哈希索引，每记录需要21-25字节。这类索引使用IndexMemory（索引内存）。

o 每个有序索引每记录需要10字节的存储空间，使用DataMemory（数据内存）。

o 创建主键或唯一索引时，还会创建1个有序索引，除非该索引是使用USING HASH创建的。换句话讲，如果未使用USING HASH创建它们，对于簇表上的每个键多因或唯一索引，在MySQL 5.1中，每记录占用31-35字节。

注意，使用USING HASH为所有主键和唯一索引创建MySQL簇表时，通常还能使表的更新速度更快。这是因为，它需要较少的内存（因未创建有序索引），对CPU的占用也较低（仅需读取或更新较少的索引）。

请记住，所有的MySQL簇表必须有主键，这点极其重要。如果未定义，NDB存储引擎将自动创建主键，而且该主键是在未使用USING HASH的条件下创建的。

很难准确确定簇索引在任何给定时间用于存储的内存量，但是，当可用DataMemory和/或IndexMemory的使用率到达80％时，会将警告消息写入簇日志，到达80％、90％等时，也会写入簇日志。

我们经常遇到用户通报的下述问题，当他们视图填充簇数据库时，加载进程提前终止，并显示下述错误消息：

```
错误1114：表'my_cluster_table'已满。
```

出现该情况时，很可能是因为在你的设置中未为所有的表数据和索引提供足够的RAM，包括*NDB*存储引擎所需的主键，以及表定义中不包含主键定义时自动创建的主键。

此外还应注意，所有的数据节点应具有相同的RAM量，这是因为，簇中任何数据节点所能使用的内存均不超过任意单独数据节点的最低可用内存。换句话讲，如果有三台运行簇数据节点的计算机，在其中两台计算机上，有3GB用于保存簇数据的RAM，另一台计算机只有1GB RAM，那么每个数据节点仅能为簇贡献1GB。

· 出现灾难性故障时，例如整个城市断电而且我的*UPS*也出现故障，我会丢失所有的数据吗？

所有已提交的事务将被记录。因此，在出现灾难的情况下，某些数据可能会丢失，但相当有限。通过减少每事务的操作数，能够进一步减少数据损失（在任何情形下，在每事务上执行大量操作都不是一个好主意）。

· 能够与簇一起使用*FULLTEXT*索引吗？

FULLTEXT索引功能目前不被NDB存储引擎支持，也不被除MyISAM之外的任何存储引擎支持。我们打算在未来的版本中增加该功能。

· 我能在一台计算机上运行多个节点吗？

可以，但不建议这样。运行簇的主要原因之一是提供冗余，为了获得冗余性的全部优点，每个节点应位于单独的计算机上。如果将多个节点置于同一台机器，当该机器出现故障时，所有节点均将丢失。考虑到MySQL簇能运行在常见的硬件上并利用廉价的操作系统（或不需费用），为了确保任务关键型数据的安全，值得为额外的机器户花费开销。此外还须注意，对运行管理节点的簇主机的要求最低，使用200 MHz Pentium CPU，操作系统所需的足够RAM，以及用于**ndb_mgmd**和**ndb_mgm**进程的少量开销，就能完成该任务。

- 我能在不重启的情况下为簇增加节点吗？

目前不行。对于在簇中添加新的MGM或SQL节点来说，简单的重启就是所需的一切。添加数据节点时，进程略微复杂些，需要采取下述步骤：

- 对所有簇数据进行完整备份。
- 彻底关闭簇和所有的簇节点进程。
- 使用"—initial"启动选项重启簇。
- 从备份中恢复所有簇数据。

在未来的MySQL簇版本中，我们希望为MySQL簇实现"热"重配置功能，以便能够将添加新节点时重启簇的要求降至最低（如果不能消除的话）。

- 使用簇时有需要我了解的限制吗？

MySQL中的NDB表服从下述限制：

- 并非所有的字符集和校对均被支持。
- 不支持FULLTEXT索引和前缀索引。只能为完整的列设置索引。不支持第19章：*MySQL*中的空间扩展中介绍的空间扩展。
- 仅支持对事务的完整回滚。不支持部分回滚以及回滚至保存点。
- 每表允许的最大属性数为128，而且属性名称不得超过31个字符。对于每个表，表和数据库名称的最大组合长度为122个字符。
- 表行的最大大小为8KB，不包括BLOB。对于每表中的行数没有限制，表的大小限制取决于多种因素，尤其是每个数据节点可用的RAM量。
- NDB引擎不支持外键约束。就像MyISAM表一样，这些约束将被忽略。
- 不支持查询高速缓冲功能。

关于簇限制的更多信息，请参见17.8节，"MySQL簇的已知限制"。

- 怎样将已有的*MySQL*数据库导入到簇中？

可以将数据库导入到MySQL簇中，就像用其他版本的MySQL那样。除了前一问题所提到的限制外，仅有的特殊要求是，准备包含到簇中的任何表必须使用NDB存储引擎。这意味着，表必须是用ENGINE=NDB或ENGINE=NDBCLUSTER创建的。使用ALTER TABLE，能够将使用其他存储引擎的现有表转换为NDB簇表，但需要采取额外避规措施，详情请参见17.8节，"MySQL簇的已知限制"。

- 簇节点是怎样彼此通信的？

簇节点能够通过下述三种协议中的任何一种进行通信：TCP/IP、SHM（共享内存）和SCI（规模可扩展的计算机连接接口）。在适用的场合，对于位于相同簇主机上的节点间通信，默认协议为SHM。SCI是高速（每秒1GB或更高）、高可用性协议，用于创建可扩展的多处理器系统，它需要特殊硬件和驱动。关于使用SCI作为簇中传输机制的更多信息，请参见17.7节，"使用与MySQL簇的高速互连"。

- 什么是"*arbitrator*"（仲裁程序）？

如果簇中的1个或多个节点失败，并非所有的簇节点均不能彼此"看到"，这是可能的。事实上，能够在网络分区中将两个节点集合彼此隔离，也称为"分裂大脑"。这类情形并不受欢迎，这是因为，每一个节点集合都试图表现为代表整个簇。

当簇节点出现问题时，有两种可能性。如果剩余节点的50％以上能够彼此通信，那么这就是我们有时称之为的"多数支配"情形，该节点集合将被视为簇。当节点数均等时，仲裁程序将介入：在该情形下，仲裁程序所属的节点集合将被当作簇，不属于该节点集合的节点将被关闭。

上述描述略为简化，更完整的解释需要考虑下面介绍的节点组：

当至少一个节点组中的所有节点均有效时，网络分区不是问题，这是因为，簇中的任一个部分均不能构成1个功能性的簇。当没有一个节点组的所有成员均是有效的时，问题产生，在该情况下，网络分区（"分裂大脑"情形）成为可能。随后就需要仲裁程序。所有的簇节点将相同的节点视为仲裁程序，通常是管理服务器。但是，也能将簇中的任何MySQL服务器配置为仲裁程序。仲裁程序接受第一个与其接触的簇节点集合，并通知剩余的集合关闭。对于MySQL服务器和管理服务器节点，仲裁程序的选择是由ArbitrationRank配置参数控制的（详情请参见17.4.4.4节，"定义MySQL簇管理服务器"）。此外还应注意，仲裁程序不会对指定主机施加过多的要求，因此，仲裁程序主机不需要特别块或拥有用于该目的的额外内存。

- *MySQL*簇支持的列类型是什么？

MySQL簇支持所有通常的MySQL列类型，但与MySQL空间扩展有关的例外（请参见第19章：*MySQL*中的空间扩展）。此外，对于索引，当与NDB表一起使用时存在一些差别。注释：MySQL簇表（即用ENGINE=NDBCLUSTER创建的表）仅有固定宽度列。这意味着（例如）含有VARCHAR(255)列的每一条记录需要256字节来保存列，无论列中保存的数据大小是多少。在未来的MySQL版本中，计划更正它。

关于这些方面的更多信息，请参见17.8节，"MySQL簇的已知限制"。

- 如何启动和停止*MySQL*簇？

需要按照下述顺序分别启动簇中的每个节点：

1. 用**ndb_mgmd**命令启动管理节点。
2. 用**ndbd**命令启动每个数据节点。

3. 使用**mysqld_safe --user=mysql &**，启动每个MySQL服务器（SQL节点）。

对于这类命令中的每一个，必须在受影响节点所在机器上的系统shell中执行。在容纳MGM节点的机器上启动管理客户端**ndb_mgm**，可验证簇是否正在运行。

· 当簇关闭时，会对簇数据有什么影响？

由簇数据节点保存在内存中的数据将被写入磁盘，并在下一次启动簇时重新装入内存。

要想关闭簇，请在MGM节点所在的机器上、在shell下输入下述命令：

```
shell> ndb_mgm -e shutdown
```

这样就能恰当地中止**ndb_mgm**、**ndb_mgm**、以及任何**ndbd**进程。对于用作簇SQL节点的MySQL服务器，可使用**mysqladmin shutdown**停止它。

更多信息，请参见17.6.2节，“管理客户端”中的命令和17.3.6节，“安全关闭和重启”。

· 簇中有*1*个以上的管理节点是否有用？

对于可靠性来说它确有帮助。在任何给定的时间，仅需1个MGM节点来控制簇，但能够将1个MGM配置为主节点，并配置1个或多个额外的管理节点，以便在主MGM节点出现故障时取代它。

· 在簇中能混合使用不同的硬件和操作系统吗？

是。只要所有的机器和操作系统均是相同的endian。也能在不同的节点上使用不同的MySQL簇版本，但是，我们建议仅应将其作为滚动升级的一部分使用。

· 我能在单台主机上运行两个数据节点吗？两个*SQL*节点？

是，能够这样。在有多个数据节点的情况下，每个节点必须使用不同的数据目录。如果打算在一台机器上运行多个SQL节点，那么每个**mysqld**实例必须使用不同的TCP/IP端口。

· 我能与*MySQL*簇一起使用主机名吗？

是，对于簇主机，能够使用DNS和DHCP。但是，如果你的应用程序要求“99.999％”的可用性，建议使用固定的IP地址。这是因为，依赖该服务的簇节点间的通信会引入额外的故障点，故障点越少越好。

17.11. MySQL簇术语表

下面给出的术语对理解MySQL簇有所帮助，而且在与MySQL簇一起使用时有一定的特殊含义。

· 簇：

按照通常的理解，簇是一个计算机集合，其功能相当于1个单位，一起工作以完成单一任务。

NDB簇：

这是MySQL中使用的存储引擎，用于实现数据存储、检索、以及多个计算机间分布式管理。

MySQL簇：

指的是使用NDB存储引擎一起工作的一组计算机，以便在使用“内存中”存储的非共享体系结构中支持分布式MySQL数据库。

· 配置文件：

包含与簇、其主机和节点有关的指令和信息的文本文件。当簇启动时，这类文件由簇的管理节点负责读取。详情请参见17.4.4节，“配置文件”。

· 备份：

所有簇数据、事务和日志的完整拷贝，保存在磁盘上或其他长期存储介质上。

· 恢复：

将簇返回以前的状态，与保存在备份中的相同。

· 检查点：

从广义上讲，将数据保存到磁盘时，即达到了检查点。具体对于簇来说，它是将所有已提交事务保存到磁盘的时间点。对于NDB存储引擎，有两种一起工作的检查点，以确保维护簇数据的一致性：

o 本地检查点（**LCP**）：

这是专门针对单一节点的检查点，但是，LCP也会在簇中的所有节点发生，同步性或强或弱。LCP包括将节点的所有数据保存到磁盘，通常每几分钟出现一次。准确的时间间隔会出现变化，具体情况取决于节点上保存的数据量，簇活动的级别，以及其他因素。

o 全局检查点（**GCP**）：

GCP每数秒出现一次，此时，将对所有节点的事务进行同步化处理，并将redo日志保存到磁盘。

· 簇主机:

构成MySQL簇组成部分的计算机。簇具有物理结构和逻辑结构。从物理意义上讲，簇由多个计算机构成，这些计算机也称为簇主机（或主机）另请参见下面的节点和节点组。

· 节点:

它指的是MySQL簇的逻辑或功能性单元，有时也称为簇节点。对于MySQL簇，我们使用术语"节点"表示进程而不是簇的物理部件。实施有效的MySQL簇需要三种节点。它们是:

o 管理（**MGM**）节点:

负责管理簇中的其他节点。它为其他节点提供了配置数据，负责启动和停止节点，处理网络分区事宜，创建备份和从备份恢复，等等。

o **SQL**（**MySQL**服务器）节点:

MySQL服务器的实例，用作簇数据节点所保存数据的前端。打算保存、检索或更新数据的客户端可访问SQL节点，就像访问其他MySQL服务器一样，采用通常的鉴定方法和API，节点组之间的基本数据分配对用户和应用程序来说是透明的。SQL节点访问作为整体的簇数据库，而不管数据在不同数据节点或簇主机上的分布情况。

o 数据节点:

这些节点负责保存实际数据。表数据片段保存在节点组集合中，每个节点组保存表数据的不同子集。构成节点组的每个节点均保存了该节点组所负责片段的副本。目前，1个簇能支持总数为48的数据节点。

1个以上的节点能共存于单台机器上（事实上，能够在一台机器上设置完成的簇，但在生产环境下几乎没人会这样做）。请记住，使用MySQL簇时，术语"主机"指的是簇的物理部件，而"节点"指的是逻辑或功能部件（即进程），这很有帮助。

关于过时术语的注释：在MySQL簇文档的早期版本中，数据节点有时也称为"数据库节点"、"DB节点"、或偶尔使用的"存储节点"。此外，SQL节点有时也称为"客户端节点"或"API节点"。为了将混淆降至最低，这类早期术语已被放弃，为此，应避免使用它们。

· 节点组:

数据节点的集合。位于节点组内的所有数据节点包含相同的数据（片段），而且单个组内的所有节点应位于不同的主机上。能够对哪个节点术语哪个节点组进行控制。

· 节点失败:

MySQL簇并不仅取决于构成构成簇的任一节点的功能，如果1个或多个节点失败，簇仍能运行。给定簇能容忍的失败节点的准确数目取决于节点的数目和簇的配置。

· 节点重启:

重启失败簇节点的进程。

· 首次节点重启:

不与其文件系统一起启动簇节点的进程。有时用于软件升级或其他特殊场合。

▪ 系统崩溃（或系统失败）:

当很多簇节点失败并无法保证簇的状态时，就会出现该情况。

· 系统重启:

重启簇并根据磁盘日志和检查点重新初始化簇状态的进程。在簇的计划关闭或非计划关闭后，通常需要执行该进程。

· 片段:

数据库表的一部分，在NDB存储引擎中，将表分为众多片段，并以片段方式保存。片段有时也称为"分区"，但"片段"是更可取的术语。在MySQL簇中，对表进行了片段化处理，从而简化了机器和节点间的负载平衡。

· 副本:

在NDB存储引擎下，每个表片段均有多个副本，这些副本保存在其他数据节点上，以提供冗余性。目前，每片段最多有4个副本。

· 传输器:

提供节点间数据传输的协议。MySQL簇目前提供了四种不同的传输器连接:

o TCP/IP（本地）

当然，这是广为人知的网络协议，它是Internet上HTTP、FTP等的基础。

o TCP/IP（远程）

同上，但用于远程通信。

- SCI

SCI（规模可扩展的计算机连接接口）是一种高速协议，由于构建多处理器系统和并行处理应用程序。与MySQL簇一起使用SCI时，需要专门的硬件，具体讨论请参见17.7.1节，“配置MySQL簇以使用SCI套接字”。关于SCI的基本介绍，请参见dolphinics.com上的文章。

- SHM

Unix风格共享内存（SHM）段。在任何支持的场合下，将自动使用SHM来连接运行在相同主机上的节点。要想了解这方面的更多信息，请参见关于shmop(2)的Unix手册页。

注释：簇传输器是簇内部的。使用MySQL簇的应用程序与SQL节点进行通信，就像与其他版本的MySQL服务器进行通信一样（通过TCP/IP、或使用Unix套接字、或Windows命名管道）。可以使用标准的MySQL API发出查询并检索结果。

- **NDB**：

它是网络数据库（**N**etwork **D**ata**b**ase）的缩写，而且指的是用于启用MySQL簇的存储引擎。NDB存储引擎支持所有通常的MySQL列类型和SQL语句，而且是ACID兼容的。该引擎还提供了对事务的完整支持（提交和回滚）。

- 无共享体系结构：

用于MySQL簇的理想体系结构。在真正的无共享设置下，每个节点运行在单独的主机上。这类安排的优点在于，单个主机或节点不会造成单点故障，也不会成为系统整体的性能瓶颈。

- 内存中存储：

保存在每个数据节点中的数据均保存在节点宿主计算机的内存中。对于簇中的每个数据节点，必须提供足够的RAM，所需的RAM量为：数据库的大小乘以副本数目，然后除以数据节点的数目。因此，如果数据库占用1GB的内存，而且你打算设置具有4个副本和8个数据节点的簇，每节点所需的最小内存为500 MB。注意，还应加上操作系统所需的内存以及运行在主机上的应用程序所需的内存。

- 表：

与关联数据库下的通常情况一样，术语“表”指的是具有等同结构的记录的集合。在MySQL簇中，数据库表以片段集合的方式保存在数据节点中，并将每个片段复制到额外的数据节点上。复制相同片段或片段集合的数据节点集合称为节点组。

- 簇程序：

它们是命令行程序，用于运行、配置和管理MySQL簇。它们包括两种服务器端口监督程序：

- **ndbd**：

数据节点端口监督程序（运行数据节点进程）。

- **ndb_mgmd**：

管理服务器端口监督程序（运行管理服务器进程）。

以及客户端程序：

- **ndb_mgm**：

管理客户端（为执行管理命令提供了1个界面）。

- **ndb_waiter**：

用于验证簇中所有节点的状态。

- **ndb_restore**：

从备份中恢复簇数据。

关于这些程序及其用法的更多信息，请参见17.5节，“MySQL簇中的进程管理”。

- 事件日志：

MySQL簇按照类别（启动、停止、错误、检查点等）、优先级和严重级别来记录事件。关于所有可通报事件的完整列表，请参见17.6.3节，“MySQL簇中生成的事件报告”。事件日志具有两种类型：

- 簇日志：

为作为整体的簇记录所有所需的、值得记录的事件。

- 节点日志：

为每个单独节点保存的单独日志。

在正常情况下，仅需保存和检查簇日志，这通常已足够。节点日志仅用于应用程序开发和调试目的。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第18章：分区

目录

18.1. MySQL中的分区概述
18.2. 分区类型

- 18.2.1. RANGE分区
- 18.2.2. LIST分区
- 18.2.3. HASH分区
- 18.2.4. KEY分区
- 18.2.5. 子分区
- 18.2.6. MySQL分区处理NULL值的方式

18.3. 分区管理

- 18.3.1. RANGE和LIST分区的管理
- 18.3.2. HASH和KEY分区的管理
- 18.3.3. 分区维护
- 18.3.4. 获取关于分区的信息

本章讨论MySQL 5.1.中实现的分区。关于分区和分区概念的介绍可以在18.1节，“MySQL中的分区概述”中找到。MySQL 5.1 支持哪几种类型的分区，在18.2节，“分区类型” 中讨论。关于子分区在18.2.5节，“子分区” 中讨论。现有分区表中分区的增加、删除和修改的方法在18.3节，“分区管理” 中介绍。 和分区表一同使用的表维护命令在18.3.3节，“分区维护” 中介绍。

请注意：MySQL 5.1中的分区实现仍然很新（pre-alpha品质），此时还不是可生产的（not production-ready）。 同样，许多也适用于本章：在这里描述的一些功能还没有实际上实现(分区维护和重新分区命令），其他的可能还没有完全如所描述的那样实现(例如, 用于分区的数据目录（DATA DIRECTORY）和索引目录（INDEX DIRECTORY）选项受到Bug #13520) 不利的影响). 我们已经设法在本章中标出这些差异。在提出缺陷报告前，我们鼓励参考下面的一些资源：

- MySQL 分区论坛

  这是一个为对MySQL分区技术感兴趣或用MySQL分区技术做试验提供的官方讨论论坛。来自MySQL 的开发者和其他的人，会在上面发表和更新有关的材料。它由分区开发和文献团队的成员负责监控。

- 分区缺陷报告

  已经归档在缺陷系统中的、所有分区缺陷的一个列表，而无论这些缺陷的年限、严重性或当前的状态如何。根据许多规则可以对这些缺陷进行筛选，或者可以从MySQL缺陷系统主页开始，然后查找你特别感兴趣的缺陷。

- Mikael Ronström's Blog

  MySQL分区体系结构和领先的开发者Mikael Ronström 经常在这里贴关于他研究MySQL 分区和MySQL簇的文章。

- PlanetMySQL

  一个MySQL 新闻网站，它以汇集MySQL相关的网誌为特点，那些使用我的MySQL的人应该对此有兴趣。我们鼓励查看那些研究MySQL分区的人的网誌链接，或者把你自己的网誌加到这些新闻报道中。

MySQL 5.1的二进制版本目前还不可用；但是，可以从BitKeeper知识库中获得源码。要激活分区，需要使用--with-分区选项编译服务器。关于建立MySQL 的更多信息，请参见2.8节，“使用源码分发版安装MySQL”。如果在编译一个激活分区的MySQL 5.1创建中碰到问题，可以在MySQL分区论坛中查找解决办法，如果在论坛中已经贴出的文章中没有找到问题的解决办法，可以在上面寻找帮助。

18.1. MySQL中的分区概述

本节提供了关于MySQL 5.1.分区在概念上的概述。

SQL标准在数据存储的物理方面没有提供太多的指南。SQL语言的使用独立于它所使用的任何数据结构或图表、表、行或列下的介质。但是，大部分高级数据库管理系统已经开发了一些根据文件系统、硬件或者这两者来确定将要用于存储特定数据块物理位置的方法。在MySQL中，InnoDB存储引擎长期支持表空间的概念，并且MySQL服务器甚至在分区引入之前，就能配置为存储不同的数据库使用不同的物理路径(关于如何配置的解释，请参见7.6.1节，“使用符号链接”)。

分区又把这个概念推进了一步，它允许根据可以设置为任意大小的规则，跨文件系统分配单个表的多个部分。实际上，表的不同部分在不同的位置被存储为单独的表。用户所选择的、实现数据分割的规则被称为分区函数，这在MySQL中它可以是模数，或者是简单的匹配一个连续的数值区间或数值列表，或者是一个内部HASH函数，或一个线性HASH函数。函数根据用户指定的分区类型来选择，把用户提供的表达式的值作为参数。该表达式可以是一个整数列值，或一个作用在一个或多个列值上并返回一个整数的函数。这个表达式的值传递给分区函数，分区函数返回一个表示那个特定记录应该保存在哪个分区的序号。这个函数不能是常数，也不能是任意数。它不能包含任何查询，但是实际上可以使用MySQL 中任何可用的SQL表达式，只要该表达式返回一个小于MAXVALUE（最大可能的正整数）的正数值。分区函数的例子可以在本章后面关于分区类型的讨论中找到 (请参见18.2节，“分区类型”)，也可在13.1.5节，“CREATE TABLE语法”的分区语法描述中找到。

当二进制码变成可用时（也就是说，5.1 -max 二进制码将通过--with-partition 建立），分区支持就将包含在MySQL 5.1的-max 版本中。如果MySQL二进制码是使用分区支持建立的，那么激活它不需要任何其他的东西 (例如，在my.cnf 文件中，不需要特殊的条目)。可以通过使用SHOW VARIABLES命令来确定MySQL是否支持分区，例如：

```
mysql> SHOW VARIABLES LIKE '%partition%';


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

| Variable_name         | Value |

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

| have_partition_engine | YES   |

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

1 row in set (0.00 sec)
```

在如上列出的一个正确的SHOW VARIABLES 命令所产生的输出中，如果没有看到变量have_partition_engine的值为YES，那么MySQL的版本就不支持分区。（注意：在显示任何有关分区支持信息的命令SHOW ENGINES的输出中，不会给出任何信息；必须使用SHOW VARIABLES命令来做出这个判断）。

对于创建了分区的表，可以使用你的MySQL 服务器所支持的任何存储引擎；MySQL 分区引擎在一个单独的层中运行，并且可以和任何这样的层进行相互作用。在MySQL 5.1版中，同一个分区表的所有分区必须使用同一个存储引擎；例如，不能对一个分区使用MyISAM，而对另一个使用InnoDB。但是，这并不妨碍在同一个MySQL 服务器中，甚至在同一个数据库中，对于不同的分区表使用不同的存储引擎。

要为某个分区表配置一个专门的存储引擎，必须且只能使用[STORAGE] ENGINE 选项，这如同为非分区表配置存储引擎一样。但是，必须记住[STORAGE] ENGINE（和其他的表选项）必须列在用在CREATE TABLE语句中

的其他任何分区选项之前。下面的例子给出了怎样创建一个通过HASH分成6个分区、使用InnoDB存储引擎的表：

```
CREATE TABLE ti (id INT, amount DECIMAL(7,2), tr_date DATE)
    ENGINE=INNODB
    PARTITION BY HASH(MONTH(tr_date))
    PARTITIONS 6;
```

（注释：每个PARTITION 子句可以包含一个 [STORAGE] ENGINE 选项，但是在MySQL 5.1版本中，这没有作用）。

创建分区的临时表也是可能的；但是，这种表的生命周期只有当前MySQL 的会话的时间那么长。对于非分区的临时表，这也是一样的。

注释：分区适用于一个表的所有数据和索引；不能只对数据分区而不对索引分区，反之亦然，同时也不能只对表的一部分进行分区。

可以通过使用用来创建分区表的CREATE TABLE语句的PARTITION子句的DATA DIRECTORY（数据路径）和INDEX DIRECTORY（索引路径）选项，为每个分区的数据和索引指定特定的路径。此外，MAX_ROWS和MIN_ROWS选项可以用来设定最大和最小的行数，它们可以各自保存在每个分区里。关于这些选项的更多信息，请参见18.3节，“分区管理”。注释：这个特殊的功能由于Bug #13250的原因，目前还不能实用。在第一个5.1二进制版本投入使用时，我们应该已经把这个问题解决了。

分区的一些优点包括：

- 与单个磁盘或文件系统分区相比，可以存储更多的数据。
- 对于那些已经失去保存意义的数据，通常可以通过删除与那些数据有关的分区，很容易地删除那些数据。相反地，在某些情况下，添加新数据的过程又可以通过为那些新数据专门增加一个新的分区，来很方便地实现。

通常和分区有关的其他优点包括下面列出的这些。MySQL 分区中的这些功能目前还没有实现，但是在我们的优先级列表中，具有高的优先级；我们希望在5.1的生产版本中，能包括这些功能。

- 一些查询可以得到极大的优化，这主要是借助于满足一个给定WHERE 语句的数据可以只保存在一个或多个分区内，这样在查找时就不用查找其他剩余的分区。因为分区可以在创建了分区表后进行修改，所以在第一次配置分区方案时还不曾这么做时，可以重新组织数据，来提高那些常用查询的效率。
- 涉及到例如SUM() 和 COUNT()这样聚合函数的查询，可以很容易地进行并行处理。这种查询的一个简单例子如 “SELECT salesperson_id, COUNT(orders) as order_total FROM sales GROUP BY salesperson_id；”。通过“并行”， 这意味着该查询可以在每个分区上同时进行，最终结果只需通过总计所有分区得到的结果。
- 通过跨多个磁盘来分散数据查询，来获得更大的查询吞吐量。

要经常检查本页和本章，因为它将随MySQL 5.1后续的分区进展而更新。

18.2. 分区类型

18.2.1. RANGE分区
18.2.2. LIST分区
18.2.3. HASH分区
18.2.4. KEY分区
18.2.5.

子分区
18.2.6. MySQL分区处理NULL值的方式

本节讨论在MySQL 5.1中可用的分区类型。这些类型包括：

- **RANGE** 分区：基于属于一个给定连续区间的列值，把多行分配给分区。参见18.2.1节，“RANGE分区”。
- **LIST** 分区：类似于按RANGE分区，区别在于LIST分区是基于列值匹配一个离散值集合中的某个值来进行选择。参见18.2.2节，“LIST分区”。
- **HASH**分区：基于用户定义的表达式的返回值来进行选择的分区，该表达式使用将要插入到表中的这些行的列值进行计算。这个函数可以包含MySQL 中有效的、产生非负整数值的任何表达式。参见18.2.3节，“HASH分区”。
- **KEY** 分区：类似于按HASH分区，区别在于KEY分区只支持计算一列或多列，且MySQL 服务器提供其自身的哈希函数。必须有一列或多列包含整数值。参见18.2.4节，“KEY分区”。

无论使用何种类型的分区，分区总是在创建时就自动的顺序编号，且从0开始记录，记住这一点非常重要。当有一新行插入到一个分区表中时，就是使用这些分区编号来识别正确的分区。例如，如果你的表使用4个分区，那么这些分区就编号为0, 1, 2, 和3。对于RANGE和LIST分区类型，确认每个分区编号都定义了一个分区，很有必要。对HASH分区，使用的用户函数必须返回一个大于0的整数值。对于KEY分区，这个问题通过MySQL服务器内部使用的 哈希函数自动进行处理。

分区的名字基本上遵循其他MySQL 标识符应当遵循的原则，例如用于表和数据库名字的标识符。但是应当注意，分区的名字是不区分大小写的。例如，下面的CREATE TABLE语句将会产生如下的错误：

```
mysql> CREATE TABLE t2 (val INT)
    -> PARTITION BY LIST(val)(
    ->     PARTITION mypart VALUES IN (1,3,5),
    ->     PARTITION MyPart VALUES IN (2,4,6)
    -> );
错误1488 (HY000): 表的所有分区必须有唯一的名字。
```

这是因为MySQL认为分区名字mypart和MyPart没有区别。

注释：在下面的章节中，我们没有必要提供可以用来创建每种分区类型语法的所有可能形式，这些信息可以在13.1.5节，“CREATE TABLE语法” 中找到。

18.2.1. RANGE分区

按照RANGE分区的表是通过如下一种方式进行分区的，每个分区包含那些分区表达式的值位于一个给定的连续区间内的行。这些区间要连续且不能相互重叠，使用VALUES LESS THAN操作符来进行定义。在下面的几个例子中，假定你创建了一个如下的一个表，该表保存有20家音像店的职员记录，这20家音像店的编号从1到20。

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
```

```
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
);
```

根据你的需要，这个表可以有多种方式来按照区间进行分区。一种方式是使用store_id 列。例如，你可能决定通过添加一个PARTITION BY RANGE子句把这个表分割成4个区间，如下所示:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
    PARTITION p0 VALUES LESS THAN (6),
    PARTITION p1 VALUES LESS THAN (11),
    PARTITION p2 VALUES LESS THAN (16),
    PARTITION p3 VALUES LESS THAN (21)
);
```

按照这种分区方案，在商店1到5工作的雇员相对应的所有行被保存在分区P0中，商店6到10的雇员保存在P1中，依次类推。注意，每个分区都是按顺序进行定义，从最低到最高。这是PARTITION BY RANGE 语法的要求；在这点上，它类似于C或Java中的“switch ... case”语句。

对于包含数据(72, 'Michael', 'Widenius', '1998-06-25', NULL, 13)的一个新行，可以很容易地确定它将插入到p2分区中，但是如果增加了一个编号为第21的商店，将会发生什么呢？在这种方案下，由于没有规则把store_id大于20的商店包含在内，服务器将不知道把该行保存在何处，将会导致错误。 要避免这种错误，可以通过在CREATE TABLE语句中使用一个“catchall” VALUES LESS THAN子句，该子句提供给所有大于明确指定的最高值的值:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
```

```
    PARTITION p0 VALUES LESS THAN (6),
    PARTITION p1 VALUES LESS THAN (11),
    PARTITION p2 VALUES LESS THAN (16),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

MAXVALUE 表示最大的可能的整数值。现在，store_id 列值大于或等于16（定义了的最高值）的所有行都将保存在分区p3中。在将来的某个时候，当商店数已经增长到25, 30, 或更多 ，可以使用ALTER TABLE语句为商店21-25, 26-30,等等增加新的分区 （关于如何实现的详细信息参见18.3节，“分区管理”)。

在几乎一样的结构中，你还可以基于雇员的工作代码来分割表，也就是说，基于job_code 列值的连续区间。例如——假定2位数字的工作代码用来表示普通（店内的）工人，三个数字代码表示办公室和支持人员，四个数字代码表示管理层，你可以使用下面的语句创建该分区表：

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
)
PARTITION BY RANGE (job_code) (
    PARTITION p0 VALUES LESS THAN (100),
    PARTITION p1 VALUES LESS THAN (1000),
    PARTITION p2 VALUES LESS THAN (10000)
);
```

在这个例子中, 店内工人相关的所有行将保存在分区p0中，办公室和支持人员相关的所有行保存在分区p1中，管理层相关的所有行保存在分区p2中。

在VALUES LESS THAN 子句中使用一个表达式也是可能的。这里最值得注意的限制是MySQL 必须能够计算表达式的返回值作为LESS THAN (<)比较的一部分；因此，表达式的值不能为NULL 。由于这个原因，雇员表的hired, separated, job_code,和store_id列已经被定义为非空（NOT NULL）。

除了可以根据商店编号分割表数据外，你还可以使用一个基于两个DATE （日期）中的一个的表达式来分割表数据。例如，假定你想基于每个雇员离开公司的年份来分割表，也就是说，YEAR(separated)的值。实现这种分区模式的CREATE TABLE 语句的一个例子如下所示：

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
```

```
    job_code INT,
    store_id INT
)
PARTITION BY RANGE (YEAR(separated)) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1996),
    PARTITION p2 VALUES LESS THAN (2001),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

在这个方案中，在1991年前雇佣的所有雇员的记录保存在分区p0中，1991年到1995年期间雇佣的所有雇员的记录保存在分区p1中， 1996年到2000年期间雇佣的所有雇员的记录保存在分区p2中，2000年后雇佣的所有工人的信息保存在p3中。

RANGE分区在如下场合特别有用：

- 当需要删除"旧的"数据时。如果你使用上面最近的那个例子给出的分区方案，你只需简单地使用"ALTER TABLE employees DROP PARTITION p0；"来删除所有在1991年前就已经停止工作的雇员相对应的所有行。(更多信息请参见13.1.2节，"ALTER TABLE语法" 和 18.3节，"分区管理"）。对于有大量行的表，这比运行一个如"DELETE FROM employees WHERE YEAR(separated) <= 1990；"这样的一个DELETE查询要有效得多。

- 想要使用一个包含有日期或时间值，或包含有从一些其他级数开始增长的值的列。

- 经常运行直接依赖于用于分割表的列的查询。例如，当执行一个如"SELECT COUNT(*) FROM employees WHERE YEAR(separated) = 2000 GROUP BY store_id；"这样的查询时，MySQL可以很迅速地确定只有分区p2需要扫描，这是因为余下的分区不可能包含有符合该WHERE子句的任何记录。注释：这种优化还没有在MySQL 5.1源程序中启用，但是，有关工作正在进行中。

18.2.2. LIST分区

MySQL中的LIST分区在很多方面类似于RANGE分区。和按照RANGE分区一样，每个分区必须明确定义。它们的主要区别在于，LIST分区中每个分区的定义和选择是基于某列的值从属于一个值列表集中的一个值，而RANGE分区是从属于一个连续区间值的集合。LIST分区通过使用"PARTITION BY LIST(*expr*)"来实现，其中"*expr*" 是某列值或一个基于某个列值、并返回一个整数值的表达式，然后通过"VALUES IN (*value_list*)"的方式来定义每个分区，其中"*value_list*"是一个通过逗号分隔的整数列表。

注释：在MySQL 5.1中，当使用LIST分区时，有可能只能匹配整数列表。

不像按照RANGE定义分区的情形，LIST分区不必声明任何特定的顺序。关于LIST分区更详细的语法信息，请参考13.1.5节，"CREATE TABLE语法" 。

对于下面给出的例子，我们假定将要被分区的表的基本定义是通过下面的"CREATE TABLE"语句提供的：

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
```

```
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
);
```

（这和18.2.1节，“RANGE分区” 中的例子中使用的是同一个表）。

假定有20个音像店，分布在4个有经销权的地区，如下表所示：

地区	商店**ID** 号
北区	3, 5, 6, 9, 17
东区	1, 2, 10, 11, 19, 20
西区	4, 12, 13, 14, 18
中心区	7, 8, 15, 16

要按照属于同一个地区商店的行保存在同一个分区中的方式来分割表，可以使用下面的“CREATE TABLE”语句：

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY LIST(store_id)
    PARTITION pNorth VALUES IN (3,5,6,9,17),
    PARTITION pEast VALUES IN (1,2,10,11,19,20),
    PARTITION pWest VALUES IN (4,12,13,14,18),
    PARTITION pCentral VALUES IN (7,8,15,16)
);
```

这使得在表中增加或删除指定地区的雇员记录变得容易起来。例如，假定西区的所有音像店都卖给了其他公司。那么与在西区音像店工作雇员相关的所有记录（行）可以使用查询“ALTER TABLE employees DROP PARTITION pWest；”来进行删除，它与具有同样作用的DELETE （删除）查询“DELETE query DELETE FROM employees WHERE store_id IN (4,12,13,14,18)；”比起来，要有效得多。

要点：如果试图插入列值（或分区表达式的返回值）不在分区值列表中的一行时，那么“INSERT”查询将失败并报错。例如，假定LIST分区的采用上面的方案，下面的查询将失败：

```
INSERT INTO employees VALUES
    (224, 'Linus', 'Torvalds', '2002-05-01', '2004-10-12', 42, 21);
```

这是因为“store_id”列值21不能在用于定义分区pNorth, pEast, pWest,或pCentral的值列表中找到。要重点注意的是，LIST分区没有类似如“VALUES LESS THAN MAXVALUE”这样的包含其他值在内的定义。将要匹配的任何

值都必须在值列表中找到。

LIST分区除了能和RANGE分区结合起来生成一个复合的子分区，与HASH和KEY分区结合起来生成复合的子分区也是可能的。 关于这方面的讨论，请参考18.2.5节，“子分区”。

18.2.3. HASH分区

18.2.3.1. LINEAR HASH分区

HASH分区主要用来确保数据在预先确定数目的分区中平均分布。在RANGE和LIST分区中，必须明确指定一个给定的列值或列值集合应该保存在哪个分区中；而在HASH分区中，MySQL 自动完成这些工作，你所要做的只是基于将要被哈希的列值指定一个列值或表达式，以及指定被分区的表将要被分割成的分区数量。

要使用HASH分区来分割一个表，要在CREATE TABLE 语句上添加一个“PARTITION BY HASH (*expr*)”子句，其中“*expr*”是一个返回一个整数的表达式。它可以仅仅是字段类型为MySQL 整型的一列的名字。此外，你很可能需要在后面再添加一个“PARTITIONS *num*”子句，其中*num* 是一个非负的整数，它表示表将要被分割成分区的数量。

例如，下面的语句创建了一个使用基于“store_id”列进行 哈希处理的表，该表被分成了4个分区：

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY HASH(store_id)
PARTITIONS 4;
```

如果没有包括一个PARTITIONS子句，那么分区的数量将默认为1。 例外： 对于NDB Cluster（簇）表，默认的分区数量将与簇数据节点的数量相同，这种修正可能是考虑任何MAX_ROWS 设置，以便确保所有的行都能合适地插入到分区中。（参见第17章：*MySQL*簇）。

如果在关键字“PARTITIONS”后面没有加上分区的数量，将会出现语法错误。

“*expr*”还可以是一个返回一个整数的SQL表达式。例如，也许你想基于雇用雇员的年份来进行分区。这可以通过下面的语句来实现：

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
```

```
    store_id INT
)
PARTITION BY HASH(YEAR(hired))
PARTITIONS 4;
```

"*expr*"还可以是MySQL 中有效的任何函数或其他表达式，只要它们返回一个既非常数、也非随机数的整数。（换句话说，它既是变化的但又是确定的）。但是应当记住，每当插入或更新（或者可能删除）一行，这个表达式都要计算一次；这意味着非常复杂的表达式可能会引起性能问题，尤其是在执行同时影响大量行的运算（例如批量插入）的时候。

最有效率的哈希函数是只对单个表列进行计算，并且它的值随列值进行一致地增大或减小，因为这考虑了在分区范围上的"修剪"。也就是说，表达式值和它所基于的列的值变化越接近，MySQL就可以越有效地使用该表达式来进行HASH分区。

例如，"date_col" 是一个DATE（日期）类型的列，那么表达式TO_DAYS(date_col)就可以说是随列"date_col"值的变化而发生直接的变化，因为列"date_col"值的每个变化，表达式的值也将发生与之一致的变化。而表达式YEAR(date_col)的变化就没有表达式TO_DAYS(date_col)那么直接，因为不是列"date_col"每次可能的改变都能使表达式YEAR(date_col)发生同等的改变。即便如此，表达式YEAR(date_col)也还是一个用于 哈希函数的、好的候选表达式，因为它随列date_col的一部分发生直接变化，并且列date_col的变化不可能引起表达式YEAR(date_col)不成比例的变化。

作为对照，假定有一个类型为整型（INT）的、列名为"int_col"的列。现在考虑表达式"POW(5-int_col,3) + 6"。这对于哈希函数就是一个不好的选择，因为"int_col"值的变化并不能保证表达式产生成比例的变化。列"int_col"的值发生一个给定数目的变化，可能会引起表达式的值产生一个很大不同的变化。例如，把列"int_col"的值从5变为6，表达式的值将产生"－1"的改变，但是把列"int_col"的值从6变为7时，表达式的值将产生"－7"的变化。

换句话说，如果列值与表达式值之比的曲线图越接近由等式"y=*n*x（其中n为非零的常数）描绘出的直线，则该表达式越适合于 哈希。这是因为，表达式的非线性越严重，分区中数据产生非均衡分布的趋势也将越严重。

理论上讲，对于涉及到多列的表达式，"修剪（pruning）"也是可能的，但是要确定哪些适于 哈希是非常困难和耗时的。基于这个原因，实际上不推荐使用涉及到多列的哈希表达式。

当使用了"PARTITION BY HASH"时，MySQL将基于用户函数结果的模数来确定使用哪个编号的分区。换句话，对于一个表达式"*expr*"，将要保存记录的分区编号为N ，其中"*N* = MOD(*expr*, *num*)"。例如，假定表t1 定义如下，它有4个分区：

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
    PARTITION BY HASH( YEAR(col3) )
    PARTITIONS 4;
```

如果插入一个col3列值为'2005-09-15'的记录到表t1中，那么保存该条记录的分区确定如下：

```
MOD(YEAR('2005-09-01'),4)
=  MOD(2005,4)
=  1
```

MySQL 5.1 还支持一个被称为"*linear hashing*（线性哈希功能）"的变量，它使用一个更加复杂的算法来确定新行插入到已经分区了的表中的位置。关于这种算法的描述，请参见18.2.3.1节，"LINEAR HASH分区" 。

每当插入或更新一条记录，用户函数都要计算一次。当删除记录时，用户函数也可能要进行计算，这取决于所处的环境。

注释：如果将要分区的表有一个唯一的键，那么用来作为HASH用户函数的自变数或者主键的*column_list*的自变数的任意列都必须是那个键的一部分。

18.2.3.1. LINEAR HASH分区

MySQL还支持线性哈希功能，它与常规哈希的区别在于，线性哈希功能使用的一个线性的2的幂（powers-of-two）运算法则，而常规 哈希使用的是求哈希函数值的模数。

线性哈希分区和常规哈希分区在语法上的唯一区别在于，在"PARTITION BY" 子句中添加"LINEAR"关键字，如下面所示：

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY LINEAR HASH(YEAR(hired))
PARTITIONS 4;
```

假设一个表达式*expr*, 当使用线性哈希功能时，记录将要保存到的分区是*num* 个分区中的分区N，其中N是根据下面的算法得到：

1. 找到下一个大于*num*.的、2的幂，我们把这个值称为*V* ，它可以通过下面的公式得到：

2. *V* = POWER(2, CEILING(LOG(2, *num*)))

（例如，假定num是13。那么LOG(2,13)就是3.7004397181411。 CEILING(3.7004397181411)就是4，则*V* = POWER(2,4), 即等于16）。

3. 设置 *N* = *F*(*column_list*) & (*V* - 1).

4. 当 *N* >= *num*:

- 设置 *V* = CEIL(*V* / 2)
- 设置 *N* = *N* & (*V* - 1)

例如，假设表t1，使用线性哈希分区且有4个分区，是通过下面的语句创建的：

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
    PARTITION BY LINEAR HASH( YEAR(col3) )
    PARTITIONS 6;
```

现在假设要插入两行记录到表t1中，其中一条记录col3列值为'2003-04-14'，另一条记录col3列值为'1998-10-19'。第一条记录将要保存到的分区确定如下：

```
V = POWER(2, CEILING(LOG(2,7))) = 8
```

```
N = YEAR('2003-04-14') & (8 - 1)
   = 2003 & 7
   = 3
```

(*3 >= 6* 为假（*FALSE*）**:** 记录将被保存到*#3*号分区中)

第二条记录将要保存到的分区序号计算如下:

```
V = 8
N = YEAR('1998-10-19') & (8-1)
  = 1998 & 7
  = 6
```

(*6 >= 4* 为真（*TRUE*）**:** 还需要附加的步骤)

```
N = 6 & CEILING(5 / 2)
  = 6 & 3
  = 2
```

(*2 >= 4* 为假（*FALSE*）**:** 记录将被保存到*#2*分区中)

按照线性哈希分区的优点在于增加、删除、合并和拆分分区将变得更加快捷，有利于处理含有极其大量（1000吉）数据的表。它的缺点在于，与使用常规HASH分区得到的数据分布相比，各个分区间数据的分布不大可能均衡。

18.2.4. KEY分区

按照KEY进行分区类似于按照HASH分区，除了HASH分区使用的用户定义的表达式，而KEY分区的 哈希函数是由MySQL 服务器提供。MySQL 簇（Cluster）使用函数MD5()来实现KEY分区；对于使用其他存储引擎的表，服务器使用其自己内部的 哈希函数，这些函数是基于与PASSWORD()一样的运算法则。

"CREATE TABLE ... PARTITION BY KEY"的语法规则类似于创建一个通过HASH分区的表的规则。它们唯一的区别在于使用的关键字是KEY而不是HASH，并且KEY分区只采用一个或多个列名的一个列表。

通过线性KEY分割一个表也是可能的。下面是一个简单的例子:

```
CREATE TABLE tk (
    col1 INT NOT NULL,
    col2 CHAR(5),
    col3 DATE
)
PARTITION BY LINEAR KEY (col1)
PARTITIONS 3;
```

在KEY分区中使用关键字LINEAR和在HASH分区中使用具有同样的作用，分区的编号是通过2的幂（powers-of-

two）算法得到，而不是通过模数算法。关于该算法及其蕴涵式的描述请参考 18.2.3.1节，“LINEAR HASH分区”。

18.2.5. 子分区

子分区是分区表中每个分区的再次分割。例如，考虑下面的CREATE TABLE 语句：

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE(YEAR(purchased))
    SUBPARTITION BY HASH(TO_DAYS(purchased))
    SUBPARTITIONS 2
    (
        PARTITION p0 VALUES LESS THAN (1990),
        PARTITION p1 VALUES LESS THAN (2000),
        PARTITION p2 VALUES LESS THAN MAXVALUE
    );
```

表ts 有3个RANGE分区。这3个分区中的每一个分区——p0, p1, 和 p2 ——又被进一步分成了2个子分区。实际上，整个表被分成了3 * 2 = 6个分区。但是，由于PARTITION BY RANGE子句的作用，这些分区的头2个只保存“purchased”列中值小于1990的那些记录。

在MySQL 5.1中，对于已经通过RANGE或LIST分区了的表再进行子分区是可能的。子分区既可以使用HASH希分区，也可以使用KEY分区。这也被称为复合分区（*composite partitioning*）。

为了对个别的子分区指定选项，使用SUBPARTITION 子句来明确定义子分区也是可能的。例如，创建在前面例子中给出的同一个表的、一个更加详细的方式如下：

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE(YEAR(purchased))
    SUBPARTITION BY HASH(TO_DAYS(purchased))
    (
        PARTITION p0 VALUES LESS THAN (1990)
        (
            SUBPARTITION s0,
            SUBPARTITION s1
        ),
        PARTITION p1 VALUES LESS THAN (2000)
        (
            SUBPARTITION s2,
            SUBPARTITION s3
        ),
        PARTITION p2 VALUES LESS THAN MAXVALUE
        (
            SUBPARTITION s4,
```

```
            SUBPARTITION s5
        )
    );
```

几点要注意的语法项:

- 每个分区必须有相同数量的子分区。
- 如果在一个分区表上的任何分区上使用SUBPARTITION 来明确定义任何子分区，那么就必须定义所有的子分区。换句话说，下面的语句将执行失败:

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE(YEAR(purchased))
    SUBPARTITION BY HASH(TO_DAYS(purchased))
    (
        PARTITION p0 VALUES LESS THAN (1990)
        (
            SUBPARTITION s0,
            SUBPARTITION s1
        ),
        PARTITION p1 VALUES LESS THAN (2000),
        PARTITION p2 VALUES LESS THAN MAXVALUE
        (
            SUBPARTITION s2,
            SUBPARTITION s3
        )
    );
```

即便这个语句包含了一个SUBPARTITIONS 2子句，但是它仍然会执行失败。

- 每个SUBPARTITION 子句必须包括 (至少)子分区的一个名字。否则，你可能要对该子分区设置任何你所需要的选项，或者允许该子分区对那些选项采用其默认的设置。
- 在每个分区内，子分区的名字必须是唯一的，但是在整个表中，没有必要保持唯一。例如，下面的CREATE TABLE 语句是有效的:

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE(YEAR(purchased))
    SUBPARTITION BY HASH(TO_DAYS(purchased))
    (
        PARTITION p0 VALUES LESS THAN (1990)
        (
            SUBPARTITION s0,
            SUBPARTITION s1
        ),
```

```
•                        PARTITION p1 VALUES LESS THAN (2000)
•                        (
•                            SUBPARTITION s0,
•                            SUBPARTITION s1
•                        ),
•                        PARTITION p2 VALUES LESS THAN MAXVALUE
•                        (
•                            SUBPARTITION s0,
•                            SUBPARTITION s1
•                        )
•                    );
```

子分区可以用于特别大的表，在多个磁盘间分配数据和索引。假设有6个磁盘，分别为/disk0， /disk1，/disk2等。现在考虑下面的例子:

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE(YEAR(purchased))
    SUBPARTITION BY HASH(TO_DAYS(purchased))
    (
        PARTITION p0 VALUES LESS THAN (1990)
        (
            SUBPARTITION s0
                DATA DIRECTORY = '/disk0/data'
                INDEX DIRECTORY = '/disk0/idx',
            SUBPARTITION s1
                DATA DIRECTORY = '/disk1/data'
                INDEX DIRECTORY = '/disk1/idx'
        ),
        PARTITION p1 VALUES LESS THAN (2000)
        (
            SUBPARTITION s0
                DATA DIRECTORY = '/disk2/data'
                INDEX DIRECTORY = '/disk2/idx',
            SUBPARTITION s1
                DATA DIRECTORY = '/disk3/data'
                INDEX DIRECTORY = '/disk3/idx'
        ),
        PARTITION p2 VALUES LESS THAN MAXVALUE
        (
            SUBPARTITION s0
                DATA DIRECTORY = '/disk4/data'
```

```
                INDEX DIRECTORY = '/disk4/idx',
            SUBPARTITION s1
                DATA DIRECTORY = '/disk5/data'
                INDEX DIRECTORY = '/disk5/idx'
        )
    );
```

在这个例子中，每个RANGE分区的数据和索引都使用一个单独的磁盘。还可能有许多其他的变化；下面是另外一个可能的例子：

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE(YEAR(purchased))
    SUBPARTITION BY HASH(TO_DAYS(purchased))
    (
        PARTITION p0 VALUES LESS THAN (1990)
        (
            SUBPARTITION s0a
                DATA DIRECTORY = '/disk0'
                INDEX DIRECTORY = '/disk1',
            SUBPARTITION s0b
                DATA DIRECTORY = '/disk2'
                INDEX DIRECTORY = '/disk3'
        ),
        PARTITION p1 VALUES LESS THAN (2000)
        (
            SUBPARTITION s1a
                DATA DIRECTORY = '/disk4/data'
                INDEX DIRECTORY = '/disk4/idx',
            SUBPARTITION s1b
                DATA DIRECTORY = '/disk5/data'
                INDEX DIRECTORY = '/disk5/idx'
        ),
        PARTITION p2 VALUES LESS THAN MAXVALUE
        (
            SUBPARTITION s2a,
            SUBPARTITION s2b
        )
    );
```

在这个例子中，存储的分配如下：

- 购买日期在1990年前的记录占了大量的存储空间，所以把它分为了四个部分进行存储，组成p0分区

的两个子分区（s0a 和s0b）的数据和索引都分别用一个单独的磁盘进行存储。换句话说：

o 子分区s0a 的数据保存在磁盘/disk0中。

o 子分区s0a 的索引保存在磁盘/disk1中。

o 子分区s0b 的数据保存在磁盘/disk2中。

o 子分区s0b 的索引保存在磁盘/disk3中。

· 保存购买日期从1990年到1999年间的记录（分区p1）不需要保存购买日期在1990年之前的记录那么大的存储空间。这些记录分在2个磁盘（/disk4和/disk5）上保存，而不是4个磁盘：

o 属于分区p1的第一个子分区（s1a）的数据和索引保存在磁盘/disk4上 — 其中数据保存在路径/disk4/data下，索引保存在/disk4/idx下。

o 属于分区p1的第二个子分区（s1b）的数据和索引保存在磁盘/disk5上 — 其中数据保存在路径/disk5/data下，索引保存在/disk5/idx下。

· 保存购买日期从2000年到现在的记录（分区p2）不需要前面两个RANGE分区那么大的空间。当前，在默认的位置能够足够保存所有这些记录。

将来，如果从2000年开始后十年购买的数量已经达到了默认的位置不能够提供足够的保存空间时，相应的记录（行）可以通过使用"ALTER TABLE ... REORGANIZE PARTITION"语句移动到其他的位置。关于如何实现的说明，请参见18.3节，"分区管理"。

18.2.6. MySQL分区处理NULL值的方式

MySQL 中的分区在禁止空值（NULL）上没有进行处理，无论它是一个列值还是一个用户定义表达式的值。一般而言，在这种情况下MySQL 把NULL视为0。如果你希望回避这种做法，你应该在设计表时不允许空值；最可能的方法是，通过声明列"NOT NULL"来实现这一点。

在本节中，我们提供了一些例子，来说明当决定一个行应该保存到哪个分区时，MySQL 是如何处理NULL值的。

如果插入一行到按照RANGE或LIST分区的表，该行用来确定分区的列值为NULL，分区将把该NULL值视为0。例如，考虑下面的两个表，表的创建和插入记录如下：

```
mysql> CREATE TABLE tnlist (
    ->     id INT,
    ->     name VARCHAR(5)
    -> )
    -> PARTITION BY LIST(id) (
    ->     PARTITION p1 VALUES IN (0),
    ->     PARTITION p2 VALUES IN (1)
    -> );
Query OK, 0 rows affected (0.09 sec)


mysql> CREATE TABLE tnrange (
    ->     id INT,
```

```
    ->     name VARCHAR(5)
    -> )
    -> PARTITION BY RANGE(id) (
    ->     PARTITION p1 VALUES LESS THAN (1),
    ->     PARTITION p2 VALUES LESS THAN MAXVALUE
    -> );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO tnlist VALUES (NULL, 'bob');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO tnrange VALUES (NULL, 'jim');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM tnlist;
+------+------+
| id   | name |
+------+------+
| NULL | bob  |
+------+------+
1 row in set (0.00 sec)

mysql> SELECT * FROM tnrange;
+------+------+
| id   | name |
+------+------+
| NULL | jim  |
+------+------+
1 row in set (0.00 sec)
```

在两个表中，id列没有声明为“NOT NULL”，这意味着它们允许Null值。可以通过删除这些分区，然后重新运行SELECT 语句，来验证这些行被保存在每个表的p1分区中：

```
mysql> ALTER TABLE tnlist DROP PARTITION p1;
Query OK, 0 rows affected (0.16 sec)

mysql> ALTER TABLE tnrange DROP PARTITION p1;
Query OK, 0 rows affected (0.16 sec)

mysql> SELECT * FROM tnlist;
```

```
Empty set (0.00 sec)


mysql> SELECT * FROM tnrange;

Empty set (0.00 sec)
```

在按HASH和KEY分区的情况下，任何产生NULL值的表达式都视同好像它的返回值为0。我们可以通过先创建一个按HASH分区的表，然后插入一个包含有适当值的记录，再检查对文件系统的作用，来验证这一点。假定有使用下面的语句在测试数据库中创建了一个表tnhash：

```
CREATE TABLE tnhash (

    id INT,

    name VARCHAR(5)

)

PARTITION BY HASH(id)

PARTITIONS 2;
```

假如Linux 上的MySQL 的一个RPM安装，这个语句在目录/var/lib/mysql/test下创建了两个.MYD文件，这两个文件可以在**bash** shell中查看，结果如下：

```
/var/lib/mysql/test> ls *.MYD -l

-rw-rw----  1 mysql mysql 0 2005-11-04 18:41 tnhash_p0.MYD

-rw-rw----  1 mysql mysql 0 2005-11-04 18:41 tnhash_p1.MYD
```

注意：每个文件的大小为0字节。现在在表tnhash 中插入一行id列值为NULL的行，然后验证该行已经被插入：

```
mysql> INSERT INTO tnhash VALUES (NULL, 'sam');

Query OK, 1 row affected (0.00 sec)


mysql> SELECT * FROM tnhash;

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

| id   | name |

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

| NULL | sam  |

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

1 row in set (0.01 sec)
```

回想一下，对于任意的整数N,NULL MOD *N* 的值总是等于NULL。这个结果在确定正确的分区方面被认为是0。回到系统shell(仍然假定**bash**用于这个目的)，通过再次列出数据文件，可以看出值被成功地插入到第一个分区（默认名称为p0）中：

```
var/lib/mysql/test> ls *.MYD -l

-rw-rw----  1 mysql mysql 20 2005-11-04 18:44 tnhash_p0.MYD

-rw-rw----  1 mysql mysql  0 2005-11-04 18:41 tnhash_p1.MYD
```

可以看出INSERT语句只修改了文件tnhash_p0.MYD，它在磁盘上的尺寸增加了，而没有影响其他的文件。

假定有下面的一个表：

```
CREATE TABLE tndate (
    id INT,
    dt DATE
)
PARTITION BY RANGE( YEAR(dt) ) (
    PARTITION p0 VALUES LESS THAN (1990),
    PARTITION p1 VALUES LESS THAN (2000),
    PARTITION p2 VALUES LESS THAN MAXVALUE
);
```

像其他的MySQL函数一样，YEAR(NULL)返回NULL值。一个dt列值为NULL的行，其分区表达式的计算结果被视为0，该行被插入到分区p0中。

18.3. 分区管理

18.3.1. RANGE和LIST分区的管理
18.3.2. HASH和KEY分区的管理
18.3.3. 分区维护
18.3.4. 获取关于分区的信息

MySQL 5.1 提供了许多修改分区表的方式。添加、删除、重新定义、合并或拆分已经存在的分区是可能的。所有这些操作都可以通过使用ALTER TABLE 命令的分区扩展来实现(关于语法的定义，请参见13.1.2节，“ALTER TABLE语法”)。也有获得分区表和分区信息的方式。在本节，我们讨论下面这些主题：

- 按RANGE或LIST分区的表的分区管理的有关信息，请参见18.3.1节，“RANGE和LIST分区的管理”。
- 关于HASH和KEY分区管理的讨论，请参见18.3.2节，“HASH和KEY分区的管理”。
- MySQL 5.1中提供的、获得关于分区表和分区信息的机制的讨论，请参见18.3.4节，“获取关于分区的信息” 。
- 关于执行分区维护操作的讨论，请参见18.3.3节，“分区维护”。

注释：在MySQL 5.1中，一个分区表的所有分区都必须有子分区同样的名字，并且一旦表已经创建，再改变子分区是不可能的。

要点：当前，从5.1系列起建立的MySQL 服务器就把“ALTER TABLE ... PARTITION BY ...”作为有效的语法，但是这个语句目前还不起作用。我们期望MySQL 5.1达到生产状态时，能够按照下面的描述实现该语句的功能。

要改变一个表的分区模式，只需要使用带有一个“*partition_options*”子句的ALTER TABLE 的命令。这个子句和与创建一个分区表的CREATE TABLE命令一同使用的子句有相同的语法，并且总是以关键字PARTITION BY 开头。例如，假设有一个使用下面CREATE TABLE语句建立的按照RANGE分区的表：

```
CREATE TABLE trb3 (id INT, name VARCHAR(50), purchased DATE)
    PARTITION BY RANGE(YEAR(purchased))
    (
        PARTITION p0 VALUES LESS THAN (1990),
        PARTITION p1 VALUES LESS THAN (1995),
        PARTITION p2 VALUES LESS THAN (2000),
```

```
        PARTITION p3 VALUES LESS THAN (2005)
    );
```

现在，要把这个表按照使用id列值作为键的基础，通过KEY分区把它重新分成两个分区，可以使用下面的语句：

```
ALTER TABLE trb3 PARTITION BY KEY(id) PARTITIONS 2;
```

这和先删除这个表、然后使用"CREATE TABLE trb3 PARTITION BY KEY(id) PARTITIONS 2；"重新创建这个表具有同样的效果。

18.3.1. RANGE和LIST分区的管理

关于如何添加和删除分区的处理，RANGE和LIST分区非常相似。基于这个原因，我们在本节讨论这两种分区的管理。关于HASH和KEY分区管理的信息，请参见18.3.2节，"HASH和KEY分区的管理"。删除一个RANGE或LIST分区比增加一个分区要更加简单易懂，所以我们先讨论前者。

从一个按照RANGE或LIST分区的表中删除一个分区，可以使用带一个DROP PARTITION子句的ALTER TABLE命令来实现。这里有一个非常基本的例子，假设已经使用下面的CREATE TABLE和INSERT语句创建了一个按照RANGE分区的表，并且已经插入了10条记录：

```
mysql> CREATE TABLE tr (id INT, name VARCHAR(50), purchased DATE)
    ->     PARTITION BY RANGE(YEAR(purchased))
    ->     (
    ->         PARTITION p0 VALUES LESS THAN (1990),
    ->         PARTITION p1 VALUES LESS THAN (1995),
    ->         PARTITION p2 VALUES LESS THAN (2000),
    ->         PARTITION p3 VALUES LESS THAN (2005)
    ->     );
Query OK, 0 rows affected (0.01 sec)


mysql> INSERT INTO tr VALUES
    ->     (1, 'desk organiser', '2003-10-15'),
    ->     (2, 'CD player', '1993-11-05'),
    ->     (3, 'TV set', '1996-03-10'),
    ->     (4, 'bookcase', '1982-01-10'),
    ->     (5, 'exercise bike', '2004-05-09'),
    ->     (6, 'sofa', '1987-06-05'),
    ->     (7, 'popcorn maker', '2001-11-22'),
    ->     (8, 'aquarium', '1992-08-04'),
    ->     (9, 'study desk', '1984-09-16'),
    ->     (10, 'lava lamp', '1998-12-25');
Query OK, 10 rows affected (0.01 sec)
```

可以通过使用下面的命令查看那些记录已经插入到了分区p2中：

```
mysql> SELECT * FROM tr
    -> WHERE purchased BETWEEN '1995-01-01' AND '1999-12-31';
+------+-----------+------------+
| id   | name      | purchased  |
+------+-----------+------------+
|    3 | TV set    | 1996-03-10 |
|   10 | lava lamp | 1998-12-25 |
+------+-----------+------------+
2 rows in set (0.00 sec)
```

要删除名字为p2的分区，执行下面的命令：

```
mysql> ALTER TABLE tr DROP PARTITION p2;
Query OK, 0 rows affected (0.03 sec)
```

记住下面一点非常重要：当删除了一个分区，也同时删除了该分区中所有的数据。可以通过重新运行前面的SELECT查询来验证这一点：

```
mysql> SELECT * FROM tr WHERE purchased
    -> BETWEEN '1995-01-01' AND '1999-12-31';
Empty set (0.00 sec)
```

如果希望从所有分区删除所有的数据，但是又保留表的定义和表的分区模式，使用TRUNCATE TABLE命令。（请参见13.2.9节，“TRUNCATE语法”）。

如果希望改变表的分区而又不丢失数据，使用“ALTER TABLE ... REORGANIZE PARTITION”语句。参见下面的内容，或者在13.1.2节，“ALTER TABLE语法” 中参考关于REORGANIZE PARTITION的信息。

如果现在执行一个SHOW CREATE TABLE命令，可以观察到表的分区结构是如何被改变的：

```
mysql> SHOW CREATE TABLE tr\G
*************************** 1. row ***************************
       Table: tr
Create Table: CREATE TABLE `tr` (
  `id` int(11) default NULL,
  `name` varchar(50) default NULL,
  `purchased` date default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
PARTITION BY RANGE (YEAR(purchased)) (
  PARTITION p0 VALUES LESS THAN (1990) ENGINE = MyISAM,
  PARTITION p1 VALUES LESS THAN (1995) ENGINE = MyISAM,
  PARTITION p3 VALUES LESS THAN (2005) ENGINE = MyISAM
)
1 row in set (0.01 sec)
```

如果插入购买日期列的值在'1995-01-01'和 '2004-12-31'之间（含）的新行到已经修改后的表中时，这些行将被保存在分区p3中。可以通过下面的方式来验证这一点：

```
mysql> INSERT INTO tr VALUES (11, 'pencil holder', '1995-07-12');

Query OK, 1 row affected (0.00 sec)


mysql> SELECT * FROM tr WHERE purchased

    -> BETWEEN '1995-01-01' AND '2004-12-31';

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

| id   | name           | purchased  |

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

|   11 | pencil holder  | 1995-07-12 |

|    1 | desk organiser | 2003-10-15 |

|    5 | exercise bike  | 2004-05-09 |

|    7 | popcorn maker  | 2001-11-22 |

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

4 rows in set (0.00 sec)


mysql> ALTER TABLE tr DROP PARTITION p3;

Query OK, 0 rows affected (0.03 sec)


mysql> SELECT * FROM tr WHERE purchased

    -> BETWEEN '1995-01-01' AND '2004-12-31';

Empty set (0.00 sec)
```

注意：由“ALTER TABLE ... DROP PARTITION”语句引起的、从表中删除的行数并没有被服务器报告出来，就好像通过同等的DELETE查询操作一样。

删除LIST分区使用和删除RANGE分区完全相同的“ALTER TABLE ... DROP PARTITION”语法。但是，在对其后使用这个表的影响方面，还是有重大的区别：在这个表中，再也不能插入这么一些行，这些行的列值包含在定义已经删除了的分区的值列表中 (有关示例，请参见18.2.2节，“LIST分区”）。

要增加一个新的RANGE或LIST分区到一个前面已经分区了的表，使用“ALTER TABLE ... ADD PARTITION”语句。对于使用RANGE分区的表，可以用这个语句添加新的区间到已有分区的序列的前面或后面。例如，假设有一个包含你所在组织的全体成员数据的分区表，该表的定义如下：

```
CREATE TABLE members (

    id INT,

    fname VARCHAR(25),

    lname VARCHAR(25),

    dob DATE

)

PARTITION BY RANGE(YEAR(dob)) (

    PARTITION p0 VALUES LESS THAN (1970),
```

```
    PARTITION p1 VALUES LESS THAN (1980),
    PARTITION p2 VALUES LESS THAN (1990)
);
```

进一步假设成员的最小年纪是16岁。随着日历接近2005年年底，你会认识到不久将要接纳1990年（以及以后年份）出生的成员。可以按照下面的方式，修改成员表来容纳出生在1990－1999年之间的成员：

```
ALTER TABLE ADD PARTITION (PARTITION p3 VALUES LESS THAN (2000));
```

要点：对于通过**RANGE**分区的表，只可以使用ADD PARTITION添加新的分区到分区列表的高端。设法通过这种方式在现有分区的前面或之间增加一个新的分区，将会导致下面的一个错误：

```
mysql> ALTER TABLE members ADD PARTITION (PARTITION p3 VALUES LESS THAN (1960));
错误1463 (HY000): 对每个分区，VALUES LESS THAN 值必须严格增长
```

采用一个类似的方式，可以增加新的分区到已经通过LIST分区的表。例如，假定有如下定义的一个表：

```
CREATE TABLE tt (
    id INT,
    data INT
)
PARTITION BY LIST(data) (
    PARTITION p0 VALUES IN (5, 10, 15),
    PARTITION p1 VALUES IN (6, 12, 18)
);
```

可以通过下面的方法添加一个新的分区，用来保存拥有数据列值7，14和21的行：

```
ALTER TABLE tt ADD PARTITION (PARTITION p2 VALUES IN (7, 14, 21));
```

注意：不能添加这样一个新的LIST分区，该分区包含有已经包含在现有分区值列表中的任意值。如果试图这样做，将会导致错误：

```
mysql> ALTER TABLE tt ADD PARTITION (PARTITION np VALUES IN (4, 8, 12));
错误1465 (HY000): 在LIST分区中，同一个常数的多次定义
```

因为带有数据列值12的任何行都已经分配给了分区p1，所以不能在表tt上再创建一个其值列表包括12的新分区。为了实现这一点，可以先删除分区p1，添加分区np，然后使用修正后的定义添加一个新的分区p1。但是，正如我们前面讨论过的，这将导致保存在分区p1中的所有数据丢失——而这往往并不是你所真正想要做的。另外一种解决方法可能是，建立一个带有新分区的表的副本，然后使用"CREATE TABLE ... SELECT ..."把数据拷贝到该新表中，然后删除旧表，重新命名新表，但是，当需要处理大量的数据时，这可能是非常耗时的。在需要高可用性的场合，这也可能是不可行的。

幸运地是，MySQL 的分区实现提供了在不丢失数据的条件下重新定义分区的方式。让我们首先看两个涉及到RANGE分区的简单例子。回想一下现在定义如下的成员表：

```
mysql> SHOW CREATE TABLE members\G
*************************** 1. row ***************************
       Table: members
Create Table: CREATE TABLE `members` (
  `id` int(11) default NULL,
```

```
  `fname` varchar(25) default NULL,
  `lname` varchar(25) default NULL,
  `dob` date default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
PARTITION BY RANGE (YEAR(dob)) (
  PARTITION p0 VALUES LESS THAN (1970) ENGINE = MyISAM,
  PARTITION p1 VALUES LESS THAN (1980) ENGINE = MyISAM,
  PARTITION p2 VALUES LESS THAN (1990) ENGINE = MyISAM.
  PARTITION p3 VALUES LESS THAN (2000) ENGINE = MyISAM
)
```

假定想要把表示出生在1960年前成员的所有行移入到一个分开的分区中。正如我们前面看到的，不能通过使用"ALTER TABLE ... ADD PARTITION"来实现这一点。但是，要实现这一点，可以使用ALTER TABLE上的另外一个与分区有关的扩展，具体实现如下：

```
ALTER TABLE members REORGANIZE PARTITION p0 INTO (
    PARTITION s0 VALUES LESS THAN (1960),
    PARTITION s1 VALUES LESS THAN (1970)
);
```

实际上，这个命令把分区p0分成了两个新的分区s0和s1。同时，它还根据包含在两个"PARTITION ... VALUES ..."子句中的规则，把保存在分区p0中的数据移入到两个新的分区中，所以分区s0中只包含YEAR(dob)小于1960的那些行，s1中包含那些YEAR(dob)大于或等于1960但是小于1970的行。

一个REORGANIZE PARTITION语句也可以用来合并相邻的分区。可以使用如下的语句恢复成员表到它以前的分区：

```
ALTER TABLE members REORGANIZE PARTITION s0,s1 INTO (
    PARTITION p0 VALUES LESS THAN (1970)
);
```

使用"REORGANIZE PARTITION"拆分或合并分区，没有数据丢失。在执行上面的语句中，MySQL 把保存在分区s0和s1中的所有数据都移到分区p0中。

"REORGANIZE PARTITION"的基本语法是：

```
ALTER TABLE tbl_name REORGANIZE PARTITION partition_list INTO (partition_definitions);
```

其中，*tbl_name* 是分区表的名称，*partition_list* 是通过逗号分开的、一个或多个将要被改变的现有分区的列表。*partition_definitions* 是一个是通过逗号分开的、新分区定义的列表，它遵循与用在"CREATE TABLE"中的*partition_definitions* 相同的规则 (请参见13.1.5节，"CREATE TABLE语法")。应当注意到，在把多少个分区合并到一个分区或把一个分区拆分成多少个分区方面，没有限制。例如，可以重新组织成员表的四个分区成两个分区，具体实现如下：

```
ALTER TABLE members REORGANIZE PARTITION p0,p1,p2,p3 INTO (
    PARTITION m0 VALUES LESS THAN (1980),
    PARTITION m1 VALUES LESS THAN (2000)
);
```

LIST REORGANIZE PARTITION

同样，对于按 分区的表，也可以使用 。让我们回到那个问题，即增加一个新的分区到已经按照LIST分区的表tt中，但是因为该新分区有一个值已经存在于现有分区的值列表中，添加新的分区失败。我们可以通过先添加只包含非冲突值的分区，然后重新组织该新分区和现有的那个分区，以便保存在现有的那个分区中的值现在移到了新的分区中，来处理这个问题：

```
ALTER TABLE tt ADD PARTITION (PARTITION np VALUES IN (4, 8));

ALTER TABLE tt REORGANIZE PARTITION p1,np INTO (

    PARTITION p1 VALUES IN (6, 18),

    PARTITION np VALUES in (4, 8, 12)

);
```

当使用"ALTER TABLE ... REORGANIZE PARTITION"来对已经按照RANGE和LIST分区表进行重新分区时，下面是一些要记住的关键点：

- 用来确定新分区模式的PARTITION子句使用与用在CREATE TABLE中确定分区模式的PARTITION子句相同的规则。

最重要的是，应该记住：新分区模式不能有任何重叠的区间（适用于按照RANGE分区的表）或值集合（适用于重新组织按照LIST分区的表）。

- *partition_definitions* 列表中分区的合集应该与在*partition_list* 中命名分区的合集占有相同的区间或值集合。

例如，在本节中用作例子的成员表中，分区p1和p2总共覆盖了1980到1999的这些年。因此，对这两个分区的重新组织都应该覆盖相同范围的年份。

- 对于按照RANGE分区的表，只能重新组织相邻的分区；不能跳过RANGE分区。

例如，不能使用以"ALTER TABLE members REORGANIZE PARTITION p0,p2 INTO ..."开头的语句，来重新组织本节中用作例子的成员表。因为，p0覆盖了1970年以前的年份，而p2覆盖了从1990到1999（包括1990和1999）之间的年份，因而这两个分区不是相邻的分区。

- 不能使用REORGANIZE PARTITION来改变表的分区类型；也就是说，例如，不能把RANGE分区变为HASH分区，反之亦然。也不能使用该命令来改变分区表达式或列。如果想在不删除和重建表的条件下实现这两个任务，可以使用"ALTER TABLE ... PARTITION BY ...."，例如：

```
ALTER TABLE members
    PARTITION BY HASH(YEAR(dob))
    PARTITIONS 8;
```

注释：在MySQL 5.1发布前的版本中，"ALTER TABLE ... PARTITION BY ..."还没有实现。作为替代，要么使用先删除表，然后使用想要的分区重建表，或者——如果需要保留已经存储在表中的数据——可以使用"CREATE TABLE ... SELECT ..."来创建新的表，然后从旧表中把数据拷贝到新表中，再删除旧表，如有必要，最后重新命名新表。

18.3.2. HASH和KEY分区的管理

在改变分区设置方面，按照HASH分区或KEY分区的表彼此非常相似，但是它们又与按照RANGE或LIST分区的表在很多方面有差别。所以，本节只讨论按照HASH或KEY分区表的修改。关于添加和删除按照RANGE或LIST进行分区的表的分区的讨论，参见18.3.1节，"RANGE和LIST分区的管理"。

不能使用与从按照RANGE或LIST分区的表中删除分区相同的方式，来从HASH或KEY分区的表中删除分区。但

是，可以使用"ALTER TABLE ... COALESCE PARTITION"命令来合并HASH或KEY分区。例如，假定有一个包含顾客信息数据的表，它被分成了12个分区。该顾客表的定义如下：

```
CREATE TABLE clients (
    id INT,
    fname VARCHAR(30),
    lname VARCHAR(30),
    signed DATE
)
PARTITION BY HASH( MONTH(signed) )
PARTITIONS 12;
```

要减少分区的数量从12到6，执行下面的ALTER TABLE命令：

```
mysql> ALTER TABLE clients COALESCE PARTITION 6;
Query OK, 0 rows affected (0.02 sec)
```

对于按照HASH，KEY，LINEAR HASH，或LINEAR KEY分区的表， COALESCE能起到同样的作用。下面是一个类似于前面例子的另外一个例子，它们的区别只是在于表是按照LINEAR KEY 进行分区：

```
mysql> CREATE TABLE clients_lk (
    ->     id INT,
    ->     fname VARCHAR(30),
    ->     lname VARCHAR(30),
    ->     signed DATE
    -> )
    -> PARTITION BY LINEAR KEY(signed)
    -> PARTITIONS 12;
Query OK, 0 rows affected (0.03 sec)

mysql> ALTER TABLE clients_lk COALESCE PARTITION 6;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

COALESCE不能用来增加分区的数量，如果你尝试这么做，结果会出现类似于下面的错误：

```
mysql> ALTER TABLE clients COALESCE PARTITION 18;
错误1478 (HY000): 不能移动所有分区，使用DROP TABLE代替
```

要增加顾客表的分区数量从12到18，使用"ALTER TABLE ... ADD PARTITION",具体如下：

```
ALTER TABLE clients ADD PARTITION PARTITIONS 18;
```

注释："ALTER TABLE ... REORGANIZE PARTITION"不能用于按照HASH或HASH分区的表。

18.3.3. 分区维护

注释：实际上，本节讨论的命令还没有在MySQL 5.1中实现， 在这里提出的目的，是为了在5.1版投产前的开发周期期间，引出来自用户测试该软件的反馈意见。(换句话说，就是"请不要反馈这样的缺陷，说这些命令不起作用"）。随着MySQL5.1版开发的继续，这些信息很有可能发生变化。随着分区功能的实现和提高，我们将更新本节的内容。

MySQL 5.1中可以执行许多分区维护的任务。对于分区表，MySQL不支持命令CHECK TABLE，OPTIMIZE TABLE，ANALYZE TABLE，或REPAIR TABLE。作为替代，可以使用ALTER TABLE 的许多扩展来在一个或多个分区上直接地执行这些操作，如下面列出的那样：

- 重建分区: 这和先删除保存在分区中的所有记录，然后重新插入它们，具有同样的效果。它可用于整理分区碎片。

示例：

```
ALTER TABLE t1 REBUILD PARTITION (p0, p1);
```

- 优化分区：如果从分区中删除了大量的行，或者对一个带有可变长度的行（也就是说，有VARCHAR，BLOB，或TEXT类型的列）作了许多修改，可以使用"ALTER TABLE ... OPTIMIZE PARTITION"来收回没有使用的空间，并整理分区数据文件的碎片。

示例：

```
ALTER TABLE t1 OPTIMIZE PARTITION (p0, p1);
```

在一个给定的分区表上使用"OPTIMIZE PARTITION"等同于在那个分区上运行CHECK PARTITION，ANALYZE PARTITION，和REPAIR PARTITION。

- 分析分区：读取并保存分区的键分布。

示例：

```
ALTER TABLE t1 ANALYZE PARTITION (p3);
```

- 修补分区： 修补被破坏的分区。

示例：

```
ALTER TABLE t1 REPAIR PARTITION (p0,p1);
```

- 检查分区： 可以使用几乎与对非分区表使用CHECK TABLE 相同的方式检查分区。

示例：

```
ALTER TABLE trb3 CHECK PARTITION (p1);
```

这个命令可以告诉你表t1的分区p1中的数据或索引是否已经被破坏。如果发生了这种情况，使用"ALTER TABLE ... REPAIR PARTITION"来修补该分区。

还可以使用**mysqlcheck**或**myisamchk** 应用程序，在对表进行分区时所产生的、单独的MYI文件上进行操作，来完成这些任务。请参见8.7节，"mysqlcheck：表维护和维修程序"。（在pre-alpha编码中，这个功能已经可以使用）。

18.3.4. 获取关于分区的信息

本节讨论获取关于现有分区的信息。这个功能仍然处于计划阶段，所以现阶段在这里描述的，实际上是我们想要在MySQL 5.1中实现的一个概观。

如在本章中别处讨论的一样，在SHOW CREATE TABLE的输出中包含了用于创建分区表的PARTITION BY子句。例如：

```
mysql> SHOW CREATE TABLE trb3\G
*************************** 1. row ***************************
       Table: trb3
Create Table: CREATE TABLE `trb3` (
  `id` int(11) default NULL,
  `name` varchar(50) default NULL,
  `purchased` date default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
PARTITION BY RANGE (YEAR(purchased)) (
  PARTITION p0 VALUES LESS THAN (1990) ENGINE = MyISAM,
  PARTITION p1 VALUES LESS THAN (1995) ENGINE = MyISAM,
  PARTITION p2 VALUES LESS THAN (2000) ENGINE = MyISAM,
  PARTITION p3 VALUES LESS THAN (2005) ENGINE = MyISAM
)
1 row in set (0.00 sec)
```

注释：当前，对于按HASH或KEY分区的表，PARTITIONS子句并不显示。 (Bug #14327)

SHOW TABLE STATUS用于分区表，它的输出与用于非分区表的输出相同，除了引擎（Engine）列总是包含'PARTITION'值。(关于这个命令的更多信息，参见13.5.4.18节，“SHOW TABLE STATUS语法”）。要获取单个分区的状态信息，我们计划实现一个SHOW PARTITION STATUS命令（请参见下面)。

计划用于分区表的、两个附加的SHOW命令是：

- SHOW PARTITIONS

这个命令预期其功能类似于SHOW TABLES和SHOW DATABASES，除了该命令将列出的是分区而不是表或数据库。这个命令的输出可能包含单个称为Partitions_in_*tbl_name* 的列，其中*tbl_name* 是分区表的名字。对于SHOW TABLES命令而言，如果一旦选择了一个数据库，随后该数据库将作为SHOW TABLES命令的默认数据库。但是由于SHOW PARTITIONS命令不可能用这样的方式来“选择”一个表，它很可能需要使用FROM子句，以便MySQL知道要显示的是哪个表的分区信息。

- SHOW PARTITION STATUS

这个命令将提供关于一个或多个分区的详细状态信息。它的输出很可能包含有与SHOW TABLE STATUS 的输出相同或类似的列，此外，还包括显示用于分区的数据和索引路径的附加列。这个命令可能支持LIKE和FROM子句，这样使得通过名字获得关于一个给定分区的信息，或者获得关于属于指定表或数据库的分区的信息，成为可能。

扩展INFORMATION_SCHEMA 数据库的计划也在进行中，以便提供关于分区表和分区的信息。这个计划当前还处一个在非常早的阶段；随着补充的信息变得可用，以及任何新的、与分区有关的INFORMATION_SCHEMA扩展得以实现，我们将更新手册相关部分的内容。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第19章：MySQL中的空间扩展

目录

19.1. 前言
19.2. OpenGIS几何模型

- 19.2.1. Geometry类的层次
- 19.2.2. 类Geometry
- 19.2.3. 类Point
- 19.2.4. 类Curve
- 19.2.5. 类LineString
- 19.2.6. 类Surface
- 19.2.7. 类Polygon
- 19.2.8. 类GeometryCollection
- 19.2.9. 类MultiPoint
- 19.2.10. 类MultiCurve
- 19.2.11. 类MultiLineString
- 19.2.12. 类MultiSurface
- 19.2.13. 类MultiPolygon

19.3. 支持的空间数据格式

- 19.3.1. 著名的文本(WKT)格式
- 19.3.2. 著名的二进制(WKB)格式

19.4. 创建具备空间功能的MySQL数据库

- 19.4.1. MySQL空间数据类型
- 19.4.2. 创建空间值
- 19.4.3. 创建空间列
- 19.4.4. 填充空间列
- 19.4.5. 获取空间数据

19.5. 分析空间信息

- 19.5.1. Geometry格式转换函数
- 19.5.2. Geometry函数
- 19.5.3. 从已有Geometry创建新Geometry的函数
- 19.5.4. 测试几何对象间空间关系的函数
- 19.5.5. 关于几何最小边界矩形（MBR）的关系
- 19.5.6. 测试几何类之间空间关系的函数

19.6. 优化空间分析

- 19.6.1. 创建空间索引
- 19.6.2. 使用空间索引

19.7. MySQL的一致性和兼容性

- 19.7.1. 尚未实施的GIS特性

MySQL支持空间扩展，允许生成、保存和分析地理特征。这些特征可用

于MyISAM、InnoDB、NDB、BDB和ARCHIVE表（但是，ARCHIVE引擎不支持索引功能，因此，不能为ARCHIVE列中的空间列编制索引）。

本章介绍了下述议题:

- OpenGIS几何模型中这些空间扩展的基础。
- 用于表示空间数据的数据格式。
- 如何在MySQL中使用空间数据。
- 使用关于空间数据的索引功能。
- MySQL与OpenGIS规范的差异。

如果在MySQL空间扩展的使用方面有任何问题，可在我们网站的GIS论坛中讨论。

19.1. 前言

遵从OpenGIS联盟（OGC）的规范，MySQL实施了空间扩展。OGC是一个由250多家公司、代理机构和大学参与的国际联盟，负责开发公开的概念解决方案，这类解决方案对所有用来管理空间数据的应用都是有用的。OGC的网站是http://www.opengis.org/。

1997年，OpenGIS联盟（OGC）发布了针对*SQL*的*OpenGIS®*简单特征规范，在该文档中，提出了扩展SQL RDBMS以支持空间数据的一些概念性方法。该规范可从OpenGIS网站上获得http://www.opengis.org/docs/99-049.pdf。其中包含与本章有关的额外信息。

MySQL实施了OGC建议的具有**Geometry**类型的**SQL**环境的一个子集。该术语指的是用一组集合类型扩展的环境。具有几何值的SQL列是作为拥有集合类型的列实施的。该规范描述了SQL几何类型集合，以及作用在这些类型上用于创建和分析几何值的函数。

地理特征指的是世界上具有地理位置的任何事物。它可以是:

- 实体，如山、池塘、城市。
- 空间，如邮政区域、热带。
- 可定义的位置，如两条街道相交的十字路口。

有些文件采用术语地理空间特征来指代地理特征。

几何是另一个表示地理特征的术语。最初，单词几何表示的是对大地的测量。来自制图学的另一个含义指的是制图人员用于绘制世界地图的地理特征。

本章将所有这些术语当作同义词对待：地理特征、地理空间特征、或几何。最常使用的术语是几何。

我们将几何定义为世界上具有地理位置的点或点的集合。

19.2. OpenGIS几何模型

19.2.1. Geometry类的层次
19.2.2. 类Geometry
19.2.3. 类Point

19.2.4. 类Curve
19.2.5. 类LineString
19.2.6. 类Surface
19.2.7. 类Polygon
19.2.8. 类GeometryCollection
19.2.9. 类MultiPoint
19.2.10. 类MultiCurve
19.2.11. 类MultiLineString
19.2.12. 类MultiSurface
19.2.13. 类MultiPolygon

OGC具有几何类型的**SQL**环境建议的几何类型集合，基于OpenGIS几何模型。在本模型中，每个几何对象均具有下述一般属性：

- 与空间参考系统相关，其中介绍了定义对象的坐标空间。
- 属于某种几何类。

19.2.1. Geometry类的层次

几何类定义了下述层次：

- Geometry（非实例化）
  - Point（可实例化的）
  - Curve（非实例化）
    - LineString（可实例化的）
    - Line
    - LinearRing
  - Surface（非实例化）
    - Polygon（可实例化的）
  - GeometryCollection（可实例化的）
    - MultiPoint（可实例化的）
    - MultiCurve（非实例化）
    - MultiLineString（可实例化的）
    - MultiSurface（非实例化）
    - MultiPolygon（可实例化的）

不能在非实例化类中创建对象。能够在可实例化类中创建对象。所有类均有属性，可实例化类还可以包含声明（定义有效类实例的规则）。

Geometry是一种基本类。它是一种抽象类。Geometry的可实例化子类限制为可在2维坐标空间中存在的0、1、2维几何对象。所有的可实例化几何类是这样定义的，从而使得几何类的实例从拓扑意义上讲是闭合

的（也就是说，所有定义的几何类包含其边界）。

基本Geometry类具有关于Point、Curve、Surface和GeometryCollection的子类：

- Point表示0维对象。
- Curve表示1维对象，具有子类LineString，以及次级子类Line和LinearRing。
- Surface是为2维对象设计的，具有子类Polygon。
- GeometryCollection具有特殊的0维、1维和2维类集合，名为MultiPoint、MultiLineString和MultiPolygon，分别用于为对应的Points、LineStrings和Polygons集合进行几何建模。MultiCurve和MultiSurface是作为抽象超类引入的，它们归纳了用于处理Curves和Surfaces的集合接口。

Geometry、Curve、Surface、MultiCurve和MultiSurface定义为非实例化类。它们为其子类定义了公用方法集合，而且是为扩展而包含在内的。

Point、LineString、Polygon、GeometryCollection、MultiPoint、MultiLineString和MultiPolygon定义为可实例化类。

19.2.2. 类Geometry

Geometry是层次结构的根类。它是一种非实例化类，但具有很多属性，这些属性对由任何Geometry子类创建的所有几何值来说是共同的。下面介绍了这些属性（尤其是具有自己特殊属性的子类）。

Geometry属性

Geometry值具有下述属性：

- 其**type**（类型）。每个geometry属于层次结构中可实例化类之一。
- 其**SRID**，或空间参考ID。该值确定了用于描述定义几何对象的坐标空间的空间坐标系统。

在MySQL中，SRID值仅是与geometry值相关的整数值。所有计算均是在欧几里得几何系（平面）中进行的。

- 它在其空间坐标系统中的**coordinates**（坐标），表示为双精度数值（8字节）。所有的非空几何对象至少包含一对坐标（X、Y）。空几何对象不含坐标。

坐标与SRID相关。例如，在不同的坐标系内，两个对象之间的距离会有所不同，即使这两个对象具有相同的坐标也同样。这是因为，平面坐标系中的距离和地心坐标系（地球表面上的坐标）中的距离是不同的事项。

- 其**interior**（内部）、**boundary**（边界）和**exterior**（外部）。

每个几何对象均占据空间中的某一位置。几何对象的exterior（外部）指的是未被该对象占据的所有空间。其interior（内部）指的是被该对象占据的空间。其boundary（边界）指的是几何对象内部和外部之间的界面。

- 其**MBR**（最小边界矩形）或包络面。这是一种边界几何值，由最小和最大坐标（X,Y）构成。

```
((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

- 无论值是简单的或非简单的。类型（LineString、MultiPoint、MultiLineString）的几何值或是简单的，或是非简单的。每个类型决定了其自己的简单或非简单声明。
- 无论值是封闭的或非封闭的。类型（LineString、MultiPoint）的几何值或是封闭的，或是非封闭的。每个类型决定了其自己的封闭或非封闭声明。

- 无论值是空的或非空的。如果没有任何点，几何对象是空的。空几何对象的内部、外部和边界未定义（也就是说，它们由Null值表示）。空的几何对象定义为总是简单的，面积为0。
- 其**dimension**（维数）。几何对象的维数为−1、0、1或2：
  - −1用于空几何对象。
  - 0用于无长度、无面积的几何对象。
  - 1用于具有非0长度和0面积的几何对象。
  - 2用于具有非0面积的几何对象。

Point对象的维数为0。LineString对象的维数为1。Polygon对象的维数为2。MultiPoint、MultiLineString和MultiPolygon对象的维数与构成它们的元素的维数相同。

19.2.3. 类Point

Point（点）指的是代表坐标空间中单个位置的几何类。

Point示例

- 想像一张具有众多城市的大世界地图。每个Point对象可代表1个城市。
- 在城市地图上，Point对象可代表1个公共汽车站。

Point属性

- X-坐标值。
- Y-坐标值。
- Point定义为0维几何对象。
- Point的边界为空集合。

19.2.4. 类Curve

Curve（曲线）是一种1维几何对象，通常由一系列点表示。Curve的特殊子类定义了点之间的内插类型。Curve是一种非实例化类。

Curve属性

- Curve具有其点的坐标。
- Curve定义为1维几何对象。
- 如果未通过相同的点两次，Curve就是简单的。
- 如果其起点等于其终点，Curve就是封闭的。
- 封闭Curve的边界为空。
- 非封闭Curve的边界由其两个端点构成。

- 简单且封闭的Curve是LinearRing。

19.2.5. 类LineString

LineString是具有点之间线性内插特性的Curve。

LineString示例

- 在世界地图上，LineString对象可表示河流。
- 在城市地图上，LineString对象可表示街道。

LineString属性

- LineString具有线段的坐标，由每个连续的点对（两点）定义。
- 如果仅包含两点，LineString为Line。
- 如果它既是简单的也是封闭的，LineString为LinearRing。

19.2.6. 类Surface

Surface是一种2维几何对象。它是一种非实例化类。其唯一的可实例化子类是Polygon.

Surface属性

- Surface定义为2维几何对象。
- 在OpenGIS规范中，将简单的Surface定义为由单一"patch"构成的几何对象，它与单个外部边界以及0或多个内部边界有关。
- 简单Surface的边界是一组与其内部和外部边界对应的封闭曲线的集合。

19.2.7. 类Polygon

Polygon是代表多边几何对象的平面Surface。它由单个外部边界以及0或多个内部边界定义，其中，每个内部边界定义为Polygon中的1个孔。

Polygon示例

- 在地区地图上，Polygon对象可表示森林、区等。

Polygon声明

- Polygon的边界由一组构成其外部边界和比内部边界的LinearRing归向集合构成（即，简单且封闭的LineString对象）。
- Polygon没有交叉的环。Polygon边界中的环可能会在Point处相交，但仅以切线方式相交。
- Polygon没有线、尖峰或穿孔。
- Polygon有由连接点集合构成的内部。
- Polygon可能包含孔。对于具有孔的Polygon，其外部不连接。每个孔定义了连接的外部部件。

前述声明使得Polygon成为简单的几何对象。

19.2.8. 类GeometryCollection

GeometryCollection是由1个或多个任意类几何对象构成的几何对象。

GeometryCollection中的所有元素必须具有相同的空间参考系（即相同的坐标系）。对GeometryCollection的元素无任何限制，但下面介绍的GeometryCollection的子类会限制其成员。这类限制可能基于：

- 元素类型（例如，MultiPoint可能仅包含Point元素）。
- 维数。
- 对元素间空间交迭程度的限制。

19.2.9. 类MultiPoint

MultiPoint是一种由Point元素构成的几何对象集合。这些点未以任何方式连接或排序。

MultiPoint示例

- 在世界地图上，MultiPoint可以代表岛链。
- 在城市地图上，MultiPoint可以表示售票处的出口。

MultiPoint属性

- MultiPoint是0维几何对象。
- 如果没有两个Point是相同的（具有等同的坐标值），MultiPoint是简单的。
- MultiPoint的边界为空集合。

19.2.10. 类MultiCurve

MultiCurve是一种由Curve元素构成的几何对象集合。MultiCurve是一种非实例化类。

MultiCurve属性

- MultiCurve是1维几何对象。
- 当且仅当其所有元素均是简单的时，MultiCurve才是简单的。任意两元素的唯一交叉仅出现在两元素边界的点上。
- MultiCurve边界是通过采用"模2联合规则"（也称为奇偶规则）获得的：如果某一点位于奇数编号MultiCurve元素的边界内，它将位于MultiCurve的边界内。
- 如果其所有元素均是封闭的，则MultiCurve为封闭的。
- 封闭MultiCurve的边界总为空。

19.2.11. 类MultiLineString

MultiLineString是一种由 LineString元素构成的MultiCurve几何对象集合。

MultiLineString示例

- 在地区地图上，MultiLineString可表示河流体系或高速路系统。

19.2.12. 类MultiSurface

MultiSurface是一种由Surface元素构成的几何对象集合。MultiSurface是一种非实例化类。其唯一的可实例化子类是MultiPolygon。

MultiSurface声明

- 2个MultiSurface面没有相交的内部。
- 2个MultiSurface元素具有最多在有限点上相交的边界。

19.2.13. 类MultiPolygon

MultiPolygon是一种由Polygon元素构成的几何对象集合。

MultiPolygon示例

- 在地区地图上，MultiPolygon可表示湖泊系统。

MultiPolygon声明

- MultiPolygon没有内部相交的的2个Polygon元素。
- MultiPolygon没有2个交叉的Polygon元素（前述声明也禁止交叉），也没有在无数点处相接触的2个Polugon元素。
- MultiPolygon不能含有有断开的线、尖峰或穿孔。MultiPolygon是一种正常的封闭点集合。
- 对于有1个以上Polygon元素的MultiPolygon，具有不连接的内部。MultiPolygon内部已连接部件的数目等于MultiPolygon中Polygon值的数目。

MultiPolygon属性

- MultiPolygon是2维几何对象。
- MultiPolygon边界是与其Polygon元素的边界对应的封闭曲线集合（LineString值）。
- MultiPolygon边界中的每个Curve准确位于1个Polygon元素的边界内。
- Polygon元素边界中的每个Curve位于MultiPolygon的边界中。

19.3. 支持的空间数据格式

19.3.1. 著名的文本(WKT)格式
19.3.2. 著名的二进制(WKB)格式

在本节中，介绍了用于表示查询中几何对象的标准空间数据格式。它们是：

- 著名的文本(WKT)格式

- 著名的二进制(WKB)格式

从其内部看，MySQL以不完全等同于WKT或WKB的格式保存几何对象值。

19.3.1. 著名的文本(WKT)格式

对于Geometry的著名文本（WKT）表示，它是为与采用ASCII格式的几何数据进行交换而设计的。

几何对象WKT表示的示例：

- Point：
- `POINT(15 20)`

注意，指定点坐标时不使用分隔用逗号。

- 具有4个点的LineString：
- `LINESTRING(0 0, 10 10, 20 25, 50 60)`

注意，点坐标对采用逗号隔开。

- 具有1个外部环和1个内部环的Polygon：
- `POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))`
- 具有三个Point值的MultiPoint：
- `MULTIPOINT(0 0, 20 20, 60 60)`
- 具有2个LineString值的MultiLineString：
- `MULTILINESTRING((10 10, 20 20), (15 15, 30 15))`
- 具有2个Polygon值的MultiPolygon：
- `MULTIPOLYGON(((0 0,10 0,10 10,0 10,0 0)),((5 5,7 5,7 7,5 7, 5 5)))`
- 由2个Point值和1个LineString构成的GeometryCollection：
- `GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30), LINESTRING(15 15, 20 20))`

在本章开始处给出的OGC规范文档中，可找到相应的Backus-Naur语法，它指定了用于编写WKT值的正式生产规则。

19.3.2. 著名的二进制(WKB)格式

用于几何值的著名二进制（WKB）表示是由OpenGIS规范定义的。它还定义在ISO “SQL/MM Part 3: Spatial”标准中。

WKB用于以二进制流的形式交换几何数据，二进制流由含几何WKB信息的BLOB值表示。

WKB使用1字节无符号整数、4字节无符号整数、以及8字节双精度数（IEEE 754格式）。1字节等于8比特。

例如，与POINT(1 1)对应的WKB值由下述21字节序列构成（在此，每个字节由2个十六进制数值表示）：

```
0101000000000000000000F03F000000000000F03F
```

该序列可分为下述部分：

```
Byte order : 01

WKB type   : 01000000

X          : 000000000000F03F

Y          : 000000000000F03F
```

表示如下：

- 字节顺序（Byte order）可以是0或1，分别表明little-endian或big-endian存储。little-endian和big-endian字节顺序也分别称为网络数据表示（NDR）和外部数据表示（XDR）。

- “WKB type”（WKB类型）是指明几何类型的代码。取值从1到7，分别表示Point、LineString、Polygon、MultiPoint、MultiLineString、MultiPolygon、和GeometryCollection。

- Point值具有X和Y坐标，每个值均用双精度值表示。

对于更复杂几何值的WKB值，它们是由更复杂的数据结构表示的，详情请参见OpenGIS规范。

19.4. 创建具备空间功能的MySQL数据库

19.4.1. MySQL空间数据类型
19.4.2. 创建空间值
19.4.3. 创建空间列
19.4.4. 填充空间列
19.4.5. 获取空间数据

本节介绍了可用于表示MySQL中空间数据的数据类型，以及用于创建和检索空间值的函数。

19.4.1. MySQL空间数据类型

MySQL具有与OpenGIS类对应的数据类型。某些类型只能保存单个几何值：

- GEOMETRY
- POINT
- LINESTRING
- POLYGON

GEOMETRY能够保存任何类型的几何值。其他的单值类型POINT、LINESTRING以及POLYGON只能保存特定几何类型的值。

其他数据类型能保存多个值：

- MULTIPOINT
- MULTILINESTRING
- MULTIPOLYGON
- GEOMETRYCOLLECTION

GEOMETRYCOLLECTION能保存任意类型的对象集合。对于其他集合类型，MULTIPOINT、MULTILINESTRING、MULTIPOLYGON和GEOMETRYCOLLECTION，仅限于具有特定几何类型的集合成员。

19.4.2. 创建空间值

19.4.2.1. 使用WKT函数创建Geometry（几何）值
19.4.2.2. 使用WKB函数创建Geometry（几何）值
19.4.2.3. 使用MySQL函数创建几何值

在本节中，介绍了使用OpenGIS标准中定义的WKT和WKB函数创建空间值的方法，以及使用相应MySQL函数的方法。

19.4.2.1. 使用WKT函数创建Geometry（几何）值

MySQL提供了众多以WKT表达式、以及可选的空间参考系ID（SRID）为输入参数的函数。它们将返回对应的几何对象。

GeomFromText()接受任何几何类型的WKT作为其第1个参量。在实施方案中也提供了与类型相关的构造函数，用于构造每一种几何类型的几何值。

- GeomCollFromText(*wkt*[,*srid*]) , GeometryCollectionFromText(*wkt*[,*srid*])

  使用其WKT表示和SRID构造GEOMETRYCOLLECTION值。

- GeomFromText(*wkt*[,*srid*]) , GeometryFromText(*wkt*[,*srid*])

  使用其WKT表示和SRID构造任何类型的几何值。

- LineFromText(*wkt*[,*srid*]) , LineStringFromText(*wkt*[,*srid*])

  使用其WKT表示和SRID构造LINESTRING值。

- MLineFromText(*wkt*[,*srid*]) , MultiLineStringFromText(*wkt*[,*srid*])

  使用其WKT表示和SRID构造MULTILINESTRING值。

- MPointFromText(*wkt*[,*srid*]) , MultiPointFromText(*wkt*[,*srid*])

  使用其WKT表示和SRID构造MULTIPOINT值。

- MPolyFromText(*wkt*[,*srid*]) , MultiPolygonFromText(*wkt*[,*srid*])

  使用其WKT表示和SRID构造MULTIPOLYGON值。

- PointFromText(*wkt*[,*srid*])

  使用其WKT表示和SRID构造POINT值。

- PolyFromText(*wkt*[,*srid*]) , PolygonFromText(*wkt*[,*srid*])

  使用其WKT表示和SRID构造POLYGON值。

OpenGIS规范还介绍了用于构造Polygon或MultiPolygon值的可选函数，这些值基于环和封闭LineString值集合的WKT表达式。这些值可以相交。MySQL未实施下述函数：

- BdMPolyFromText(*wkt*,*srid*)

  以包含已关闭LineString值任意集合的WKT格式，从MultiLineString值构造MultiPolygon值。

- BdPolyFromText(*wkt*,*srid*)

  以包含已关闭LineString值任意集合的WKT格式，从MultiLineString值构造Polygon值。

19.4.2.2. 使用**WKB**函数创建**Geometry**（几何）值

MySQL提供了众多函数，它们将包含WKT表达式、或可选的空间参考系统ID（SRID）的BLOB作为输入参数。它们返回对应的几何对象。

GeomFromWKB()接受任何几何类型的WKB作为其第1个参量。在实施方案中也提供了与类型相关的构造函数，用于构造每一种几何类型的几何值。

- GeomCollFromWKB(*wkb*[,*srid*]) , GeometryCollectionFromWKB(*wkb*[,*srid*])

  使用其WKB表示和SRID构造GEOMETRYCOLLECTION值。

- GeomFromWKB(*wkb*[,*srid*]) , GeometryFromWKB(*wkb*[,*srid*])

  使用其WKB表示和SRID构造任意类型的几何值。

- LineFromWKB(*wkb*[,*srid*]) , LineStringFromWKB(*wkb*[,*srid*])

  使用其WKB表示和SRID构造LINESTRING值。

- MLineFromWKB(*wkb*[,*srid*]) , MultiLineStringFromWKB(*wkb*[,*srid*])

  使用其WKB表示和SRID构造MULTILINESTRING值。

- MPointFromWKB(*wkb*[,*srid*]) , MultiPointFromWKB(*wkb*[,*srid*])

  使用其WKB表示和SRID构造MULTIPOINT值。

- MPolyFromWKB(*wkb*[,*srid*]) , MultiPolygonFromWKB(*wkb*[,*srid*])

  使用其WKB表示和SRID构造MULTIPOLYGON值。

- PointFromWKB(*wkb*[,*srid*])

  使用其WKB表示和SRID构造POINT值。

- PolyFromWKB(*wkb*[,*srid*]) , PolygonFromWKB(*wkb*[,*srid*])

  使用其WKB表示和SRID构造POLYGON值。

OpenGIS规范还介绍了用于构造Polygon或MultiPolygon值的可选函数，这些值基于环和封闭LineString值集合的WKB表达式。这些值可以相交。MySQL未实施下述函数：

- BdMPolyFromWKB(*wkb*,*srid*)

  以包含已关闭LineString值任意集合的WKB格式，从MultiLineString值构造MultiPolygon值。

- BdPolyFromWKB(*wkb*,*srid*)

以包含已关闭LineString值任意集合的WKB格式，从MultiLineString值构造Polygon值。

19.4.2.3. 使用MySQL函数创建几何值

注释：MySQL未实施本节所列的函数。

MySQL为创建几何WKB表达式提供了有用的函数集合。本节介绍的函数是对OpenGIS规范的MySQL扩展。这些函数的结果是包含几何值（无SRID）的BLOB值。这些函数的结果可被GeomFromWKB()函数系列中任意函数的第1个参量取代。

- GeometryCollection(*g1*,*g2*,...)

  构造WKB GeometryCollection。如果任何参量不是构造良好的几何对象WKB表达式，返回值为NULL。

- LineString(*pt1*,*pt2*,...)

  从多个WKB Point参量构造WKB LineString值。如果任何参量不是WKB Point，返回值为NULL。如果Point参量的数目小于2，返回值为NULL。

- MultiLineString(*ls1*,*ls2*,...)

  使用WKB LineString参量构造WKB MultiLineString值。如果任何参量不是WKB LineString，返回值为NULL。

- MultiPoint(*pt1*,*pt2*,...)

  使用WKB Point参量构造WKB MultiPoint值。如果任何参量不是WKB Point，返回值为NULL。

- MultiPolygon(*poly1*,*poly2*,...)

  从一组WKB Polygon参量构造WKB MultiPolygon值。如果任何参量不是WKB Polygon，返回值为NULL。

- Point(*x*,*y*)

  使用其坐标构造WKB Point。

- Polygon(*ls1*,*ls2*,...)

  从多个WKB LineString参量构造WKB Polygon值。如果任何参量未表示为LinearRing的WKB形式（即，非封闭和简单LineString），返回值为NULL。

19.4.3. 创建空间列

MySQL提供了为几何类型创建空间列的标准方法，例如，使用CREATE TABLE或ALTER TABLE。目前，仅对MyISAM标支持空间列。

- 使用CREATE TABLE语句创建具有空间列的表：

```
mysql> CREATE TABLE geom (g GEOMETRY);
Query OK, 0 rows affected (0.02 sec)
```

- 使用ALTER TABLE语句在已有表中增加空间列，或将空间列从已有表中删除：

```
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
```

19.4.4. 填充空间列

创建了空间列后，可用空间数据填充它们。

值应以内部几何格式保存，但你也能将其从WKT或WKB格式转换为内部几何格式。在下面的示例中，介绍了通过将WKT值转换为内部几何格式以便将几何值插入表中的方法。

你可以在INSERT语句中执行直接转换操作：

```
INSERT INTO geom VALUES (GeomFromText('POINT(1 1)'));


SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (GeomFromText(@g));
```

也可以在INSERT之前执行转换操作：

```
SET @g = GeomFromText('POINT(1 1)');
INSERT INTO geom VALUES (@g);
```

在下面的示例中，将多个复杂的几何值插入到了表中：

```
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));
```

在前面的所有示例中，均采用了GeomFromText()来创建几何值。你也可以使用与类型相关的函数：

```
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，如果客户端应用程序打算使用几何值的WKB表示，它需要在队列中将正确构造的WKB发送至服务器。但是，存在数种满足该要求的方法。例如：

- 用十六进制文字语法插入POINT(1 1)值：

```
mysql> INSERT INTO geom VALUES
    -> (GeomFromWKB(0x0101000000000000000000F03F000000000000F03F));
```

- ODBC应用程序能够发送WKB表达式，并使用具有BLOB类型的参量将其绑定到占位符：

```
INSERT INTO geom VALUES (GeomFromWKB(?))
```

其他编程接口或许也支持类似的占位符机制。

在C程序中，可以使用mysql_real_escape_string()转义二进制值，并将结果包含在将发送至服务器的查询字符串。请参见25.2.3.52节，“mysql_real_escape_string()”

19.4.5. 获取空间数据

19.4.5.1. 以内部格式获取空间数据
19.4.5.2. 以WKT格式获取空间数据
19.4.5.3. 以格式获取空间数据

对于表中保存的几何值，能够以内部格式获取。你也能将其转换为WKT或WKB格式。

19.4.5.1. 以内部格式获取空间数据

在表对表传输中，使用内部格式来获取几何值可能是有用的。

```
CREATE TABLE geom2 (g GEOMETRY) SELECT g FROM geom;
```

19.4.5.2. 以WKT格式获取空间数据

AsText()函数能够将几何值从内部格式转换为WKT字符串。

```
SELECT AsText(g) FROM geom;
```

19.4.5.3. 以格式获取空间数据

AsBinary()函数能够将几何值从内部格式转换为包含WKB值的BLOB。

```
SELECT AsBinary(g) FROM geom;
```

19.5. 分析空间信息

19.5.1. Geometry格式转换函数
19.5.2. Geometry函数
19.5.3. 从已有Geometry创建新Geometry的函数
19.5.4. 测试几何对象间空间关系的函数
19.5.5. 关于几何最小边界矩形（MBR）的关系

19.5.6. 测试几何类之间空间关系的函数

用值填充了空间列后，即可查询和分析它们。MySQL提供了一组在空间数据上执行各种操作的函数。根据它们所执行的操作类型，可以将这些函数分为四种主要类别：

- 用于在各种格式间转换几何值的函数。
- 用于访问几何值定性或定量属性的函数。
- 描述两种几何值之间关系的函数。
- 从已有Geometry创建新Geometry的函数

空间分析函数可用于很多场合下，如：

- 任何交互式SQL程序，如**mysql**或**MySQLCC**。
- 以任何语言编写的、支持MySQL客户端API的应用程序。

19.5.1. Geometry格式转换函数

MySQL支持下述用于在内部格式和WKT或WKB格式间转换几何值的函数：

- AsBinary(*g*)

  将采用内部几何格式的值转换为其WKB表示，并返回二进制结果。

  ```
  SELECT AsBinary(g) FROM geom;
  ```

- AsText(*g*)

  将采用内部几何格式的值转换为其WKT表示，并返回字符串结果。

  ```
  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)  |
  +--------------------------+
  ```

- GeomFromText(*wkt*[,*srid*])

  将字符串值从其WKT表示转换为内部几何格式，并返回结果。也支持多种与类型相关的函数，如PointFromText()和LineFromText()，请参见19.4.2.1节，“使用WKT函数创建Geometry（几何）值”。

- GeomFromWKB(*wkb*[,*srid*])

  将二进制值从其WKB表示转换为内部几何格式，并返回结果。也支持多种与类型相关的函数，如PointFromWKB()和LineFromWKB()，请参见19.4.2.2节，“使用WKB函数创建Geometry（几何）值”。

19.5.2. Geometry函数

19.5.2.1. 通用几何函数
19.5.2.2. Point函数

19.5.2.3. LineString函数
19.5.2.4. MultiLineString函数

19.5.2.5. Polygon函数
19.5.2.6. MultiPolygon函数
19.5.2.7. GeometryCollection函数

属于该组的每个函数均将几何值作为其参量，并返回几何值的定性或定量属性。某些函数限制了其参量类型。如果参量是不正确的几何类型，这类函数将返回NULL。例如，如果对象类型既不是Polygon也不是MultiPolygon，Area()将返回NULL。

19.5.2.1. 通用几何函数

本节列出的函数不限制其参量，可接受任何类型的几何值。

- Dimension(*g*)

  返回几何值*g*的固有维数。结果可以是-1、0、1或2。（关于这些值的含义，请参见19.2.2节，“类Geometry”）。

```
mysql> SELECT Dimension(GeomFromText('LineString(1 1,2 2)'));
+-----------------------------------------------+
| Dimension(GeomFromText('LineString(1 1,2 2)')) |
+-----------------------------------------------+
|                                             1 |
+-----------------------------------------------+
```

- Envelope(*g*)

  返回几何值g的最小边界矩形（MBR）。结果以Polygon值的形式返回。

  多边形（polygon）是由边界框的顶点定义的：

```
POLYGON((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

```
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))                        |
+-------------------------------------------------------+
```

- GeometryType(*g*)

  以字符串形式返回几何类型的名称，几何实例g是几何类型的成员。该名称与可实例化几何子类之一对应。

```
mysql> SELECT GeometryType(GeomFromText('POINT(1 1)'));
+-----------------------------------------+
| GeometryType(GeomFromText('POINT(1 1)')) |
+-----------------------------------------+
| POINT                                   |
+-----------------------------------------+
```

- SRID(*g*)

  返回指明了几何值g的空间参考系统ID的整数。

  在MySQL中，SRID值仅是与几何值相关的整数。所有计算均是在欧几里得（平面）几何中进行的。

```
mysql> SELECT SRID(GeomFromText('LineString(1 1,2 2)',101));
+-----------------------------------------------+
| SRID(GeomFromText('LineString(1 1,2 2)',101)) |
+-----------------------------------------------+
|                                           101 |
+-----------------------------------------------+
```

OpenGIS规范还定义了下述函数，MySQL未实施这类函数：

- Boundary(*g*)

  返回几何值g的组合边界的闭包的几何对象。

- IsEmpty(*g*)

  如果几何值g为空的几何对象，返回1，如果非空，返回0，如果参量为NULL，返回-1。如果几何对象是空的，它表示空的点集合。

- IsSimple(*g*)

  目前该函数是占位符，不应使用它。如果实施了它，其行为与下段所给出的描述类似。

  如果几何值g没有异常的几何点（如自相交或自相切），返回1。如果参量不是简单参量，IsSimple()返回0，如果参量是NULL，返回-1。

  对于本章前面介绍的每个可实例化几何类，均包含特定的条件，这类条件会使类实例被分类为非简单的。

19.5.2.2. Point函数

Point由X和Y坐标构成，可使用下述函数获得它们：

- X(*p*)

  以双精度数值返回点p的X坐标值。

```
mysql> SELECT X(GeomFromText('Point(56.7 53.34)'));
+--------------------------------------+
| X(GeomFromText('Point(56.7 53.34)')) |
+--------------------------------------+
|                                 56.7 |
+--------------------------------------+
```

- Y(*p*)

  以双精度数值返回点p的Y坐标值。

```
mysql> SELECT Y(GeomFromText('Point(56.7 53.34)'));
+--------------------------------------+
| Y(GeomFromText('Point(56.7 53.34)')) |
+--------------------------------------+
|                                53.34 |
+--------------------------------------+
```

19.5.2.3. LineString函数

LineString由Point值组成。你可以提取LineString的特定点，计数它所包含的点数，或获取其长度。

- EndPoint(*ls*)

  返回LineString值ls的最后一个点的Point。

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(EndPoint(GeomFromText(@ls)));
+-------------------------------------+
| AsText(EndPoint(GeomFromText(@ls))) |
+-------------------------------------+
```

```
| POINT(3 3)                            |
+---------------------------------------+
```

- GLength(*ls*)

  以双精度数值返回LineString值ls在相关的空间参考系中的长度。

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT GLength(GeomFromText(@ls));
+----------------------------+
| GLength(GeomFromText(@ls)) |
+----------------------------+
|            2.8284271247462 |
+----------------------------+
```

- NumPoints(*ls*)

  返回LineString值ls中的点数。

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT NumPoints(GeomFromText(@ls));
+------------------------------+
| NumPoints(GeomFromText(@ls)) |
+------------------------------+
|                            3 |
+------------------------------+
```

- PointN(*ls*,*n*)

  返回LineString值ls中的第*n*个点。点编号从1开始。

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(PointN(GeomFromText(@ls),2));
+-------------------------------------+
| AsText(PointN(GeomFromText(@ls),2)) |
+-------------------------------------+
| POINT(2 2)                          |
+-------------------------------------+
```

- StartPoint(*ls*)

  返回LineString值ls的第一个点的Point。

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(StartPoint(GeomFromText(@ls)));
+----------------------------------------+
| AsText(StartPoint(GeomFromText(@ls))) |
+----------------------------------------+
| POINT(1 1)                             |
+----------------------------------------+
```

OpenGIS规范还定义了下述函数，MySQL尚未实施这些函数：

- IsRing(*ls*)

  如果LineString值*ls*是封闭的（即其StartPoint()和EndPoint()值相同）和简单的（未通过相同的点1次以上）返回1。如果ls不是环，返回0，如果它是NULL，返回-1。

19.5.2.4. MultiLineString函数

- GLength(*mls*)

  以双精度数值形式返回MultiLineString值m1s的长度。*mls*的长度等于其元素的长度之和。

```
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT GLength(GeomFromText(@mls));
```

```
+-----------------------------+
| GLength(GeomFromText(@mls)) |
+-----------------------------+
|             4.2426406871193 |
+-----------------------------+
```

- IsClosed(*mls*)

  如果MultiLineString值m1s是封闭的（即StartPoint()和EndPoint()值对m1s中的每个LineString是相同的）返回1。如果mls是非封闭的，返回0，如果它是NULL，返回-1。

```
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT IsClosed(GeomFromText(@mls));
+------------------------------+
| IsClosed(GeomFromText(@mls)) |
+------------------------------+
|                            0 |
+------------------------------+
```

19.5.2.5. Polygon函数

- Area(*poly*)

  以双精度数值形式返回Polygon值*poly*的面积，根据在其空间参考系中的测量值。

```
mysql> SET @poly = 'Polygon((0 0,0 3,3 0,0 0),(1 1,1 2,2 1,1 1))';
mysql> SELECT Area(GeomFromText(@poly));
+---------------------------+
| Area(GeomFromText(@poly)) |
+---------------------------+
|                         4 |
+---------------------------+
```

- ExteriorRing(*poly*)

  以LineString形式返回Polygon值*poly*的外环。

```
mysql> SET @poly =
    -> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT AsText(ExteriorRing(GeomFromText(@poly)));
+-------------------------------------------+
| AsText(ExteriorRing(GeomFromText(@poly))) |
+-------------------------------------------+
| LINESTRING(0 0,0 3,3 3,3 0,0 0)           |
+-------------------------------------------+
```

- InteriorRingN(*poly*,*n*)

  以LineString形式返回Polygon值*poly*的第n个内环。环编号从1开始。

```
mysql> SET @poly =
    -> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT AsText(InteriorRingN(GeomFromText(@poly),1));
+-----------------------------------------------+
| AsText(InteriorRingN(GeomFromText(@poly),1)) |
+-----------------------------------------------+
| LINESTRING(1 1,1 2,2 2,2 1,1 1)               |
+-----------------------------------------------+
```

- NumInteriorRings(*poly*)

  返回Polygon值*poly*的内环的数目。

```
mysql> SET @poly =
    -> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT NumInteriorRings(GeomFromText(@poly));
+-------------------------------------+
| NumInteriorRings(GeomFromText(@poly)) |
```

```
+----------------------------------------+
|                                      1 |
+----------------------------------------+
```

19.5.2.6. MultiPolygon函数

- Area(*mpoly*)

  以双精度数值形式返回MultiPolygon值*mpoly*的面积，根据在其空间参考系中的测量结果。

```
mysql> SET @mpoly =
    -> 'MultiPolygon(((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1)))';
mysql> SELECT Area(GeomFromText(@mpoly));
+-----------------------------+
| Area(GeomFromText(@mpoly)) |
+-----------------------------+
|                           8 |
+-----------------------------+
```

OpenGIS规范还定义了下述函数，MySQL未实施这类函数：

- Centroid(*mpoly*)

  以Point形式返回用于MultiPolygon值*mpoly*的数学质心。不保证结果位于MultiPolygon上。

- PointOnSurface(*mpoly*)

  返回Point值，保证该值位于MultiPolygon值*mpoly*上。

19.5.2.7. GeometryCollection函数

- GeometryN(*gc*,*n*)

  返回GeometryCollection值*gc*中第n个几何对象。几何对象的编号从1开始。

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT AsText(GeometryN(GeomFromText(@gc),1));
+----------------------------------------+
| AsText(GeometryN(GeomFromText(@gc),1)) |
+----------------------------------------+
| POINT(1 1)                             |
+----------------------------------------+
```

- NumGeometries(*gc*)

  返回GeometryCollection值*gc*中几何对象的数目。

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT NumGeometries(GeomFromText(@gc));
+-----------------------------------+
| NumGeometries(GeomFromText(@gc)) |
+-----------------------------------+
|                                 2 |
+-----------------------------------+
```

19.5.3. 从已有Geometry创建新Geometry的函数

19.5.3.1. 生成新Geometry的Geometry函数
19.5.3.2. 空间操作符

19.5.3.1. 生成新Geometry的Geometry函数

在19.5.2节，“Geometry函数”中，我们讨论了一些可从已有几何对象构造新几何对象的函数：

- Envelope(*g*)
- StartPoint(*ls*)
- EndPoint(*ls*)
- PointN(*ls*,*n*)
- ExteriorRing(*poly*)
- InteriorRingN(*poly*,*n*)
- GeometryN(*gc*,*n*)

19.5.3.2. 空间操作符

OpenGIS建议了很多可生成几何对象的其他函数。它们是为实施空间操作符而设计的。

在MySQL中未实施这些函数。它们或许会在未来的版本中出现。

- Buffer(*g*,*d*)

  返回几何对象，该对象代表所有距几何值g的距离小于或等于d的所有点。

- ConvexHull(*g*)

  返回几何对象，该对象代表几何值*g*的凸包。

- Difference(*g1*,*g2*)

  返回几何对象，该对象表示了几何值g1与g2的点集合差异。

- Intersection(*g1*,*g2*)

  返回几何对象，该对象表示了几何值g1与g2的点集合交集。

- SymDifference(*g1*,*g2*)

  返回几何对象，该对象表示了几何值g1与g2的点集合对称差。

- Union(*g1*,*g2*)

  返回几何对象，该对象表示了几何值g1与g2的点集合联合。

19.5.4. 测试几何对象间空间关系的函数

这些节中所介绍的函数以2个几何对象作为输入参数，并返回它们之间的定量或定性关系。

19.5.5. 关于几何最小边界矩形（MBR）的关系

MySQL提供了一些可测试两个几何对象g1和g2最小边界矩形之间关系的函数。它们包括：

- MBRContains(*g1*,*g2*)

返回1或0以指明*g1*的最小边界矩形是否包含*g2*的最小边界矩形。

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

- MBRDisjoint(*g1*,*g2*)

  返回1或0以指明两个几何变量*g1*和*g2*的最小边界矩形是否不相交。

- MBREqual(*g1*,*g2*)

  返回1或0以指明两个几何变量*g1*和*g2*的最小边界矩形是否相同。

- MBRIntersects(*g1*,*g2*)

  返回1或0以指明两个几何变量*g1*和*g2*的最小边界矩形是否相交。

- MBROverlaps(*g1*,*g2*)

  返回1或0以指明两个几何变量*g1*和*g2*的最小边界矩形是否交迭。

- MBRTouches(*g1*,*g2*)

  返回1或0以指明两个几何变量*g1*和*g2*的最小边界矩形是否接触。

- MBRWithin(*g1*,*g2*)

  返回1或0以指明*g1*的最小边界矩形是否位于*g2*的最小边界矩形内。

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

19.5.6. 测试几何类之间空间关系的函数

OpenGIS规范定义了下述函数。目前在MySQL尚未按照规范实施它们。对于那些已实施的函数，它们返回的结果与对应的基于MBR的函数返回的相同。包括下面列出的函数，但Distance()和Related()除外。

在未来的版本中，可能会实施这些函数，为空间分析提供全部支持，而不仅仅是基于MBR的支持。

这些函数作用在2个几何值g1和g2上。

- Contains(*g1*,*g2*)

  返回1或0以指明*g1*是否完全包含*g2*。

- Crosses(*g1*,*g2*)

  如果g1在空间上与g2相交，返回1。如果g1为Polygon或MultiPolygon，返回NULL，或如

果g2为Point或MultiPoint返回NULL。否则，返回0。

术语"空间上交叉"指的是2个给定几何对象之间的空间关系，它具有下述属性：

- 2个结合对象交叉。
- 其交叉结果将导致其维数小于两个给定几何对象最大维数的几何对象。
- 其交叉不等于两个几何对象中的任何1个。

- Disjoint(*g1*,*g2*)

  返回1或0以指明*g1*是否与*g2*从空间上不相交。

- Distance(*g1*,*g2*)

  以双精度数值形式返回2个几何对象中2点间的最短距离。

- Equals(*g1*,*g2*)

  返回1或0以指明*g1*是否从空间上等同于*g2*。

- Intersects(*g1*,*g2*)

  返回1或0以指明*g1*是否从空间上与*g2*相交。

- Overlaps(*g1*,*g2*)

  返回1或0以指明*g1*是否从空间上与*g2*交迭。如果2个几何对象交叉而且其交叉将导致具有相同维数但并不等同于任一几何对象的几何对象，将使用术语"空间交迭"。

- Related(*g1*,*g2*,*pattern_matrix*)

  返回1或0以指明由*pattern_matrix*指定的空间关系是否在*g1*和*g2*间存在。如果参量为NULL返回-1。模式矩形为字符串。如果实施了该函数，其规范将在此给出。

- Touches(*g1*,*g2*)

  返回1或0以指明*g1*是否从空间上与*g2*接触。如果几何对象的内部不交叉，但1个几何对象的边界与另一个的边界或内部交叉，这两个几何对象是从空间上接触的。

- Within(*g1*,*g2*)

  返回1或0以指明*g1*是否从空间上位于*g2*内。

19.6. 优化空间分析

19.6.1. 创建空间索引
19.6.2. 使用空间索引

可以使用索引对2个非空间数据库中的搜索操作进行优化。对于空间数据库，这同样成立。有了以前设计的大量多维索引功能的帮助，能够对空间搜索进行优化。最典型的情况如下：

- 搜索包含给定点的所有对象的Point查询。
- 搜索与给定地区交迭的所有对象的地区查询。

MySQL采用了具有**2**次分裂特性的**R-Trees**来为空间列编制索引。使用几何对象的MBR来创建空间索引。对于大多数几何对象，MBR是包围几何对象的最小矩形。对于水平或垂直linestring，MBR退化为linestring的矩形。对于点，MBR是退化为点的矩形。

此外，还能在空间列上创建正常索引。需要为除POINT列之外的空间列上的任何索引（非空间）声明前缀。

19.6.1. 创建空间索引

MySQL能够使用与创建正规索引类似的语法创建空间索引，但使用了SPATIAL关键字进行了扩展。对于目前编制了索引的空间列，必须将其声明为NOT NULL。在下面的示例中，介绍了创建空间索引的方法。

- 对于CREATE TABLE：
- `mysql> CREATE TABLE geom (g GEOMETRY NOT NULL, SPATIAL INDEX(g));`
- 对于ALTER TABLE：
- `mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);`
- 对于CREATE INDEX：
- `mysql> CREATE SPATIAL INDEX sp_index ON geom (g);`

对于MyISAM表，SPATIAL INDEX负责创建R-tree索引。对于支持空间索引的其他存储引擎，SPATIAL INDEX能够创建B-tree索引。对于准确的值查找而不是范围扫描，作用在空间值上的B-tree索引很有用。

要想撤销空间索引，可使用ALTER TABLE或DROP INDEX：

- 对于ALTER TABLE：
- `mysql> ALTER TABLE geom DROP INDEX g;`
- 对于DROP INDEX：
- `mysql> DROP INDEX sp_index ON geom;`

示例：假定表geom包含32000以上的几何对象，它们保存在类型为GEOMETRY的列g中。该表还有用于保存对象ID值的AUTO_INCREMENT列。

```
mysql> DESCRIBE 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)
```

要想在列g上添加空间索引，可使用下述语句：

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
Query OK, 32376 rows affected (4.05 sec)
Records: 32376  Duplicates: 0  Warnings: 0
```

19.6.2. 使用空间索引

优化程序将调查可用的空间索引是否能包含在使用某些函数的查询搜索中，如WHERE子句中的MBRContains()或MBRWithin()函数。例如，假定我们打算找出位于给定矩形中的所有对象：

```
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)
```

我们使用EXPLAIN来检查该查询的执行方式（ID列已被删除，以便输出能更好地与页匹配）：

```
mysql> EXPLAIN SELECT fid,AsText(g) FROM geom WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000 16000,30000 16000,30000
15000))'),g);
+-------------+-------+-------+---------------+------+---------+------+------+-------------+
| select_type | table | type  | possible_keys | key  | key_len | ref  | rows | Extra       |
+-------------+-------+-------+---------------+------+---------+------+------+-------------+
| SIMPLE      | geom  | range | g             | g    |      32 | NULL |   50 | Using where |
+-------------+-------+-------+---------------+------+---------+------+------+-------------+
1 row in set (0.00 sec)
```

让我们检查一下在没有空间索引的情况下会出现什么：

```
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);
+-------------+-------+------+---------------+------+---------+------+-------+-------------+
| select_type | table | type | possible_keys | key  | key_len | ref  | rows  | Extra       |
+-------------+-------+------+---------------+------+---------+------+-------+-------------+
| SIMPLE      | geom  | ALL  | NULL          | NULL |    NULL | NULL | 32376 | Using where |
+-------------+-------+------+---------------+------+---------+------+-------+-------------+
1 row in set (0.00 sec)
```

执行SELECT语句，忽略空间键：

```
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)
```

未使用索引时，该查询的执行时间将从0.00秒上升到0.46秒。

在未来的版本中，空间索引也可能会用于优化其他函数。请参见19.5.4节，“测试几何对象间空间关系的函数”。

19.7. MySQL的一致性和兼容性

19.7.1. 尚未实施的GIS特性

19.7.1. 尚未实施的GIS特性

- 额外的元数据视图

  OpenGIS规范建议了数种额外的元数据视图。例如，包含几何列的描述的名为GEOMETRY_COLUMNS的系统视图，对于数据库中的每列有1行相关内容。

- 作用在LineString和MultiLineString上的OpenGIS函数Length()目前应在MySQL中以GLength()的方式调用。

  问题在于存在1个用于计算字符串值长度的已有SQL函数Length()，而且在某些情况下无法判断函数是在文本场景下还是空间场景下调用的。我们需要以某种方式解决该问题，或确定另一个函数名。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第20章：存储程序和函数

目录

20.1. 存储程序和授权表

20.2. 存储程序的语法

20.2.1. CREATE PROCEDURE和CREATE FUNCTION

20.2.2. ALTER PROCEDURE和ALTER FUNCTION

20.2.3. DROP PROCEDURE和和DROP FUNCTION

20.2.4. SHOW CREATE PROCEDURE和SHOW CREATE FUNCTION

20.2.5. SHOW PROCEDURE STATUS和SHOW FUNCTION STATUS

20.2.6. CALL语句

20.2.7. BEGIN ... END复合语句

20.2.8. DECLARE语句

20.2.9. 存储程序中的变量

20.2.10. 条件和处理程序

20.2.11. 光标

20.2.12. 流程控制构造

20.3. 存储程序、函数、触发程序和复制：常见问题

20.4. 存储子程序和触发程序的二进制日志功能

MySQL 5.1版支持存储程序和函数。一个存储程序是可以被存储在服务器中的一套SQL语句。一旦它被存储了，客户端不需要再重新发布单独的语句，而是可以引用存储程序来替代。

下面一些情况下存储程序尤其有用：

- 当用不同语言编写多客户应用程序，或多客户应用程序在不同平台上运行且需要执行相同的数据库操作之时。

- 安全极为重要之时。比如，银行对所有普通操作使用存储程序。这提供一个坚固而安全的环境，程序可以确保每一个操作都被妥善记入日志。在这样一个设置中，应用程序和用户不可能直接访问数据库表，但是仅可以执行指定的存储程序。

存储程序可以提供改良后的性能，因为只有较少的信息需要在服务器和客户算之间传送。代价是增加数据库服务器系统的负荷，因为更多的工作在服务器这边完成，更少的在客户端（应用程序）那边完成上。如果许多客户端机器（比如网页服务器）只由一个或少数几个数据库服务器提供服务，可以考虑一下存储程序。

存储程序也允许你在数据库服务器上有函数库。这是一个被现代应用程序语言共享的特征，它允许这样的内部设计，比如通过使用类。使用这些客户端应用程序语言特征对甚至于数据库使用范围以外的编程人员都有好处。

MySQL为存储程序遵循SQL:2003语法，这个语法也被用在IBM的DB2数据库上。

MySQL对存储程序的实现还在进度中。所有本章叙述的语法都被支持，在有限制或扩展的地方会恰当地指出来。有关使用 存储程序的限制的更多讨论在附录 I, 特性限制里提到。

如20.4节，“存储子程序和触发程序的二进制日志功能”里所说的， 存储子程序的二进制日志功能已经完成。

20.1. 存储程序和授权表

存储程序需要在mysql数据库中有proc表。这个表在MySQL 5.1安装过程中创建。如果你从早期的版本升级到MySQL 5.1 ，请确定更新你的授权表以确保proc表的存在。请参阅2.10.2节“升级授权表”。

在MySQL 5.1中，授权系统如下考虑存储子程序:

- 创建存储子程序需要CREATE ROUTINE权限。
- 提醒或移除存储子程序需要ALTER ROUTINE权限。这个权限自动授予子程序的创建者。
- 执行子程序需要EXECUTE权限。然而，这个权限自动授予 子程序的创建者。同样，子程序默认的SQL SECURITY 特征是DEFINER，它允许用该子程序访问数据库的用户与执行子程序联系到一起。

20.2. 存储程序的语法

20.2.1. CREATE PROCEDURE和CREATE FUNCTION

20.2.2. ALTER PROCEDURE和ALTER FUNCTION

20.2.3. DROP PROCEDURE和DROP FUNCTION

20.2.4. SHOW CREATE PROCEDURE和SHOW CREATE FUNCTION

20.2.5. SHOW PROCEDURE STATUS和SHOW FUNCTION STATUS

20.2.6. CALL语句

20.2.7. BEGIN ... END复合语句

20.2.8. DECLARE语句

20.2.9. 存储程序中的变量

20.2.10. 条件和处理程序

20.2.11. 光标

20.2.12. 流程控制构造

存储程序和函数是用CREATE PROCEDURE和CREATE FUNCTION语句创建的子程序。一个子程序要么是一个程序要么是一个函数。使用CALL语句来调用 程序，程序只能用输出变量传回值。就像别其它函数调用一样，函数可以被从语句外调用（即通过引用函数名），函数能返回标量值。 存储子程序也可以调用其它存储子程序。

在MySQL 5.1中，一个存储子程序或函数与特定的数据库相联系。这里有几个意思:

- 当一个子程序被调用时，一个隐含的USE *db_name* 被执行（当子程序终止时停止执行）。存储子程序

内的USE语句时不允许的。

- 你可以使用数据库名限定子程序名。这可以被用来引用一个不在当前数据库中的子程序。比如，要引用一个与test数据库关联的存储程序p或函数f，你可以说CALL test.p()或test.f()。

- 数据库移除的时候，与它关联的所有存储子程序也都被移除。

MySQL 支持非常有用的扩展，即它允许在存储程序中使用常规的SELECT语句（那就是说，不使用光标或 局部变量）。这个一个查询的结果包被简单地直接送到客户端。多SELECT语句生成多个结果包，所以客户端必须使用支持多结果包的MySQL客户端库。这意味这客户端必须 使用至少MySQL 4.1以来的近期版本上的客户端库。

下面一节描述用来创建，改变，移除和查询存储程序和函数的语法。

20.2.1. CREATE PROCEDURE和CREATE FUNCTION

```
CREATE PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body


CREATE FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body


    proc_parameter:
    [ IN | OUT | INOUT ] param_name type


    func_parameter:
    param_name type


type:
    Any valid MySQL data type


characteristic:
    LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'


routine_body:
    Valid SQL procedure statement or statements
```

这些语句创建存储子程序。要在MySQL 5.1中创建子程序，必须具有CREATE ROUTINE权限，并且ALTER ROUTINE和EXECUTE权限被自动授予它的创建者。如果二进制日志功能被允许，你也可能需要SUPER权限，请参阅20.4节，“存储子程序和触发程序的二进制日志功能”。

默认地，子程序与当前数据库关联。要明确地把子程序与一个给定数据库关联起来，可以在创建子程序的时候指定其名字为*db_name.sp_name*。

如果子程序名和内建的SQL函数名一样，定义子程序时，你需要在这个名字和随后括号中间插入一个空格，否则发生语法错误。当你随后调用子程序的时候也要插入。为此，即使有可能出现这种情况，我们还是建议最好避免给你自己的 存储子程序取与存在的SQL函数一样的名字。

由括号包围的参数列必须总是存在。如果没有参数，也该使用一个空参数列()。每个参数 默认都是一个IN参数。要指定为其它参数，可在参数名之前使用关键词 OUT或INOUT

注意: 指定参数为IN, OUT, 或INOUT 只对PROCEDURE是合法的。（FUNCTION参数总是被认为是IN参数）

RETURNS字句只能对FUNCTION做指定，对函数而言这是强制的。它用来指定函数的返回类型，而且函数体必须包含一个RETURN value语句。

routine_body 包含合法的SQL过程语句。可以使用复合语句语法，请参阅20.2.7节，“BEGIN ... END复合语句”。复合语句可以包含 声明，循环和其它控制结构语句。这些语句的语法在本章后免介绍，举例，请参阅20.2.8节，“DECLARE语句”和20.2.12节，“流程控制构造”。

CREATE FUNCTION语句被用在更早的MySQL版本上以支持UDF （自定义函数）。请参阅27.2节，“给MySQL添加新函数”。 UDF继续被支持，即使现在 有了存储函数。UDF会被认为一个外部存储函数。然而，不要让存储函数与UDF函数共享名字空间。

外部存储程序的框架将在不久的将来引入。这将允许你用SQL之外的语言编写存储程序。最可能的是，第一个被支持语言是PHP，因为核心PHP引擎很小，线程安全，且可以被方便地嵌入。因为框架是公开的，它希望许多其它语言也能被支持。

如果程序或线程总是对同样的输入参数产生同样的结果，则被认为它是“确定的”，否则就是“非确定”的。如果既没有给定DETERMINISTIC也没有给定NOT DETERMINISTIC，默认的就是NOT DETERMINISTIC。

为进行复制，使用NOW()函数（或它的同义词）或RAND()函数会不必要地使得一个子程序非确定。对NOW()而言，二进制日志包括时间戳并被正确复制。RAND() 只要在一个子程序被内应用一次也会被正确复制。（你可以把子程序执行时间戳和随机数种子认为强制输入，它们在主从上是同样的。）

当前来讲，DETERMINISTIC特征被接受，但还没有被优化程序所使用。然而如果二进制日志功能被允许了，这个特征影响到MySQL是否会接受子程序定义。请参阅20.4节，“存储子程序和触发程序的二进制日志功能”。

一些特征提供子程序使用数据的内在信息。CONTAINS SQL表示子程序不包含读或写数据的语句。NO SQL表示子程序不包含SQL语句。READS SQL DATA表示子程序包含读数据的语句，但不包含写数据的语句。MODIFIES SQL DATA表示子程序包含写数据的语句。如果这些特征没有明确给定，默认的是CONTAINS SQL。

SQL SECURITY特征可以用来指定 子程序该用创建子程序者的许可来执行，还是使用调用者的许可来执行。默认值是DEFINER。在SQL:2003中者是一个新特性。创建者或调用者必须由访问 子程序关联的数据库的许可。在MySQL 5.1中，必须有EXECUTE权限才能执行子程序。必须拥有这个权限的用户要么是定义者，要么是调用者，这取决于SQL SECURITY特征是如何设置的。

MySQL存储sql_mode系统变量设置，这个设置在子程序被创建的时候起作用，MySQL总是强制使用这个设置来执行 子程序。

COMMENT子句是一个MySQL的扩展，它可以被用来描述 存储程序。这个信息被SHOW CREATE PROCEDURE和 SHOW CREATE FUNCTION语句来显示。

MySQL允许子程序包含DDL语句，如CREATE和DROP。MySQL也允许存储程序（但不是 存储函数）包含SQL交互语句，如COMMIT。存储函数不可以包含那些做明确的和绝对的提交或者做回滚的语。SQL标准不要求对

这些语句的支持，SQL标准声明每个DBMS提供商可以决定是否允许支持这些语句。

存储子程序不能使用LOAD DATA INFILE。

返回结果包的语句不能被用在存储函数种。这包括不使用INTO给变量读取 列值的SELECT语句，SHOW 语句，及其它诸如EXPLAIN这样的语句。对于可在函数定义时间被决定要返回一个结果包的语句，发生一个允许从函数错误返回结果包的Not（ER_SP_NO_RETSET_IN_FUNC）。对于只可在运行时决定要返回一个结果包的语句， 发生一个不能在给定上下文错误返回结果包的PROCEDURE %s （ER_SP_BADSELECT)。

下面是一个使用OUT参数的简单的存储程序的例子。例子为，在 程序被定义的时候，用**mysql**客户端delimiter命令来把语句定界符从 ;变为//。这就允许用在 程序体中的;定界符被传递到服务器而不是被**mysql**自己来解释。

```
mysql> delimiter //


mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
    -> BEGIN
    ->   SELECT COUNT(*) INTO param1 FROM t;
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)


mysql> delimiter ;


mysql> CALL simpleproc(@a);
Query OK, 0 rows affected (0.00 sec)


mysql> SELECT @a;
+------+
| @a   |
+------+
| 3    |
+------+
1 row in set (0.00 sec)
```

当使用delimiter命令时，你应该避免使用反斜杠('\')字符，因为那是MySQL的 转义字符。

下列是一个例子，一个采用参数的函数使用一个SQL函数执行一个操作，并返回结果：

```
mysql> delimiter //


mysql> CREATE FUNCTION hello (s CHAR(20)) RETURNS CHAR(50)
    -> RETURN CONCAT('Hello, ',s,'!');
    -> //
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> delimiter ;


mysql> SELECT hello('world');
+----------------+
| hello('world') |
+----------------+
| Hello, world!  |
+----------------+
1 row in set (0.00 sec)
```

如果在存储函数中的RETURN语句返回一个类型不同于在函数的RETURNS子句中指定类型的值，返回值被强制为恰当的类型。比如，如果一个函数返回一个ENUM或SET值，但是RETURN语句返回一个整数，对于SET成员集的相应的ENUM成员，从函数返回的值是字符串。

20.2.2. ALTER PROCEDURE和ALTER FUNCTION

```
ALTER {PROCEDURE | FUNCTION} sp_name [characteristic ...]


characteristic:
    { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'
```

这个语句可以被用来改变一个存储程序或函数的特征。在MySQL 5.1中，你必须用ALTER ROUTINE权限才可用此子程序。这个权限被自动授予子程序的创建者。如20.4节，“存储子程序和触发程序的二进制日志功能”中所述， 如果二进制日志功能被允许了，你可能也需要SUPER权限。

在ALTER PROCEDURE和ALTER FUNCTION语句中，可以指定超过一个的改变。

20.2.3. DROP PROCEDURE和DROP FUNCTION

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

这个语句被用来移除一个存储程序或函数。即，从服务器移除一个制定的子程序。在MySQL 5.1中，你必须有ALTER ROUTINE权限才可用此子程序。这个权限被自动授予子程序的创建者。

IF EXISTS 子句是一个MySQL的扩展。如果程序或函数不存储，它防止发生错误。产生一个可以用SHOW WARNINGS查看的警告。

20.2.4. SHOW CREATE PROCEDURE和SHOW CREATE FUNCTION

```
SHOW CREATE {PROCEDURE | FUNCTION} sp_name
```

这个语句是一个MySQL的扩展。类似于SHOW CREATE TABLE，它返回一个可用来重新创建已命名 子程序的确切字符串。

```
mysql> SHOW CREATE FUNCTION test.hello\G
```

```
*************************** 1. row ***************************
       Function: hello
       sql_mode:
Create Function: CREATE FUNCTION `test`.`hello`(s CHAR(20)) RETURNS CHAR(50)
RETURN CONCAT('Hello, ',s,'!')
```

20.2.5. SHOW PROCEDURE STATUS和SHOW FUNCTION STATUS

```
SHOW {PROCEDURE | FUNCTION} STATUS [LIKE 'pattern']
```

这个语句是一个MySQL的扩展。它返回子程序的特征，如数据库，名字，类型，创建者及创建和修改日期。如果没有指定样式，根据你使用的语句，所有 存储程序和所有存储函数的信息都被列出。

```
mysql> SHOW FUNCTION STATUS LIKE 'hello'\G
*************************** 1. row ***************************
           Db: test
         Name: hello
         Type: FUNCTION
      Definer: testuser@localhost
     Modified: 2004-08-03 15:29:37
      Created: 2004-08-03 15:29:37
Security_type: DEFINER
      Comment:
```

你可以从INFORMATION_SCHEMA中的ROUTINES表获得有关存储子程序的信息。请参阅23.1.14节，“INFORMATION_SCHEMA ROUTINES 表”。

20.2.6. CALL语句

```
CALL sp_name([parameter[,...]])
```

CALL语句调用一个先前用CREATE PROCEDURE创建的程序。

CALL语句可以用 声明为OUT或的INOUT参数的参数给它的调用者传回值。它也"返回"受影响的行数，客户端程序可以在SQL级别通过调用ROW_COUNT()函数获得这个数，从C中是调用the mysql_affected_rows() C API函数来获得。

20.2.7. BEGIN ... END复合语句

```
[begin_label:] BEGIN
    [statement_list]
END [end_label]
```

存储子程序可以使用BEGIN ... END复合语句来包含多个语句。*statement_list* 代表一个或多个语句的列表。*statement_list*之内每个语句都必须用分号（；）来结尾。

复合语句可以被标记。除非*begin_label*存在,否则*end_label*不能被给出,并且如果二者都存在,他们必须是同样的。

请注意，可选的[NOT] ATOMIC子句现在还不被支持。这意味着在指令块的开始没有交互的存储点被设置，并且在上下文中用到的BEGIN子句对当前交互动作没有影响。

使用多重语句需要客户端能发送包含语句定界符;的查询字符串。这个符号在命令行客户端被用delimiter命令来处理。改变查询结尾定界符;（比如改变为//）使得; 可被用在子程序体中。

20.2.8. DECLARE语句

DECLARE语句被用来把不同项目局域到一个 子程序：局部变量（请参阅20.2.9节，“存储程序中的变量”)，条件和 处理程序（请参阅20.2.10节，“条件和处理程序”) 及光标（请参阅20.2.11节，“光标”)。SIGNAL和RESIGNAL语句当前还不被支持。

DECLARE仅被用在BEGIN ... END复合语句里，并且必须在复合语句的开头，在任何其它语句之前。

光标必须在声明处理程序之前被声明，并且变量和条件必须在声明光标或处理程序之前被声明。

20.2.9. 存储程序中的变量

20.2.9.1. DECLARE局部变量

20.2.9.2. 变量SET语句

20.2.9.3. SELECT ... INTO语句

你可以在子程序中声明并使用变量。

20.2.9.1. DECLARE局部变量

```
DECLARE var_name[,...] type [DEFAULT value]
```

这个语句被用来声明局部变量。要给变量提供一个默认值，请包含一个DEFAULT子句。值可以被指定为一个表达式，不需要为一个常数。如果没有DEFAULT子句，初始值为NULL。

局部变量的作用范围在它被声明的BEGIN ... END块内。它可以被用在嵌套的块中，除了那些用相同名字 声明变量的块。

20.2.9.2. 变量SET语句

```
SET var_name = expr [, var_name = expr] ...
```

在存储程序中的SET语句是一般SET语句的扩展版本。被参考变量可能是子程序内声明的变量，或者是全局服务器变量。

在存储程序中的SET语句作为预先存在的SET语法的一部分来实现。这允许SET a=x, b=y, ...这样的扩展语法。其中不同的变量类型（局域 声明变量及全局和集体变量）可以被混合起来。这也允许把局部变量和一些只对系统变量有意义的选项合并起来。在那种情况下，此选项被识别，但是被忽略了。

20.2.9.3. SELECT ... INTO语句

```
SELECT col_name[,...] INTO var_name[,...] table_expr
```

这个SELECT语法把选定的列直接存储到变量。因此，只有单一的行可以被取回。

```
SELECT id,data INTO x,y FROM test.t1 LIMIT 1;
```

注意，用户变量名在MySQL 5.1中是对大小写不敏感的。请参阅9.3节，“用户变量”。

重要: SQL变量名不能和列名一样。如果SELECT ... INTO这样的SQL语句包含一个对列的参考，并包含一个与列相同名字的 局部变量，MySQL当前把参考解释为一个变量的名字。例如，在下面的语句中，xname 被解释为到xname *variable* 的参考而不是到xname *column*的：

```
CREATE PROCEDURE sp1 (x VARCHAR(5))
  BEGIN
    DECLARE xname VARCHAR(5) DEFAULT 'bob';
    DECLARE newname VARCHAR(5);
    DECLARE xid INT;

    SELECT xname,id INTO newname,xid
      FROM table1 WHERE xname = xname;
    SELECT newname;
  END;
```

当这个程序被调用的时候，无论table.xname列的值是什么，变量newname将返回值'bob'。

请参阅I.1节，“存储子程序和触发程序的限制”。

20.2.10. 条件和处理程序

20.2.10.1. DECLARE条件

20.2.10.2. DECLARE处理程序

特定条件需要特定处理。这些条件可以联系到错误，以及子程序中的一般流程控制。

20.2.10.1. DECLARE条件

```
DECLARE condition_name CONDITION FOR condition_value

condition_value:
    SQLSTATE [VALUE] sqlstate_value
  | mysql_error_code
```

这个语句指定需要特殊处理的条件。它将一个名字和指定的错误条件关联起来。这个名字可以随后被用在DECLARE HANDLER语句中。请参阅20.2.10.2节，“DECLARE处理程序”。

除了SQLSTATE值，也支持MySQL错误代码。

20.2.10.2. DECLARE处理程序

```
DECLARE handler_type HANDLER FOR condition_value[,...] sp_statement

handler_type:
    CONTINUE
```

```
  | EXIT
  | UNDO

condition_value:
    SQLSTATE [VALUE] sqlstate_value
  | condition_name
  | SQLWARNING
  | NOT FOUND
  | SQLEXCEPTION
  | mysql_error_code
```

这个语句指定每个可以处理一个或多个条件的处理程序。如果产生一个或多个条件，指定的语句被执行。

对一个CONTINUE处理程序，当前子程序的执行在执行 处理程序语句之后继续。对于EXIT处理程序，当前BEGIN...END复合语句的执行被终止。UNDO 处理程序类型语句还不被支持。

- SQLWARNING是对所有以01开头的SQLSTATE代码的速记。
- NOT FOUND是对所有以02开头的SQLSTATE代码的速记。
- SQLEXCEPTION是对所有没有被SQLWARNING或NOT FOUND捕获的SQLSTATE代码的速记。

除了SQLSTATE值，MySQL错误代码也不被支持。

例如:

```
mysql> CREATE TABLE test.t (s1 int,primary key (s1));
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter //

mysql> CREATE PROCEDURE handlerdemo ()
    -> BEGIN
    ->   DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
    ->   SET @x = 1;
    ->   INSERT INTO test.t VALUES (1);
    ->   SET @x = 2;
    ->   INSERT INTO test.t VALUES (1);
    ->   SET @x = 3;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL handlerdemo()//
```

```
Query OK, 0 rows affected (0.00 sec)


mysql> SELECT @x//
    +------+
    | @x   |
    +------+
    | 3    |
    +------+
    1 row in set (0.00 sec)
```

注意到，@x是3，这表明MySQL被执行到程序的末尾。如果DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1; 这一行不在，第二个INSERT因PRIMARY KEY强制而失败之后,MySQL可能已经采取 默认(EXIT)路径，并且SELECT @x可能已经返回2。

20.2.11. 光标

20.2.11.1.声明光标

20.2.11.2. 光标OPEN语句

20.2.11.3. 光标FETCH语句

20.2.11.4. 光标CLOSE语句

简单光标在存储程序和函数内被支持。语法如同在嵌入的SQL中。光标当前是不敏感的，只读的及不滚动的。不敏感意为服务器可以活不可以复制它的结果表。

光标必须在声明处理程序之前被声明，并且变量和条件必须在声明光标或处理程序之前被声明。

例如:

```
CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE a CHAR(16);
  DECLARE b,c INT;
  DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
  DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;

  OPEN cur1;
  OPEN cur2;

  REPEAT
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
```

```
    IF NOT done THEN
       IF b < c THEN
          INSERT INTO test.t3 VALUES (a,b);
       ELSE
          INSERT INTO test.t3 VALUES (a,c);
       END IF;
    END IF;
  UNTIL done END REPEAT;

  CLOSE cur1;
  CLOSE cur2;
END
```

20.2.11.1.声明光标

```
DECLARE cursor_name CURSOR FOR select_statement
```

这个语句声明一个光标。也可以在子程序中定义多个光标，但是一个块中的每一个光标必须有唯一的名字。

SELECT语句不能有INTO子句。

20.2.11.2. 光标OPEN语句

```
OPEN cursor_name
```

这个语句打开先前声明的光标。

20.2.11.3. 光标FETCH语句

```
FETCH cursor_name INTO var_name [, var_name] ...
```

这个语句用指定的打开光标读取下一行（如果有下一行的话），并且前进光标指针。

20.2.11.4. 光标CLOSE语句

```
CLOSE cursor_name
```

这个语句关闭先前打开的光标。

如果未被明确地关闭，光标在它被声明的复合语句的末尾被关闭。

20.2.12. 流程控制构造

20.2.12.1. IF语句

20.2.12.2. CASE语句

20.2.12.3. LOOP语句

20.2.12.4. LEAVE语句

20.2.12.5. ITERATE语句

20.2.12.6. REPEAT语句

20.2.12.7. WHILE语句

IF, CASE, LOOP, WHILE, ITERATE, 及 LEAVE 构造被完全实现。

这些构造可能每个包含要么一个单独语句，要么是使用BEGIN ... END复合语句的一块语句。构造可以被嵌套。

目前还不支持FOR循环。

20.2.12.1. IF语句

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF
```

IF实现了一个基本的条件构造。如果*search_condition*求值为真，相应的SQL语句列表被执行。如果没有*search_condition*匹配，在ELSE子句里的语句列表被执行。*statement_list*可以包括一个或多个语句。

请注意，也有一个IF() 函数，它不同于这里描述的IF语句。请参阅12.2节，“控制流程函数”。

20.2.12.2. CASE语句

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

Or:

```
CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

存储程序的CASE语句实现一个复杂的条件构造。如果*search_condition* 求值为真，相应的SQL被执行。如果没有搜索条件匹配，在ELSE子句里的语句被执行。

注意：这里介绍的用在 存储程序里的CASE语句与12.2节，“控制流程函数”里描述的SQL CASE表达式的CASE语句有轻微不同。这里的CASE语句不能有ELSE NULL子句，并且用END CASE替代END来终止。

20.2.12.3. LOOP语句

```
[begin_label:] LOOP
```

```
    statement_list
END LOOP [end_label]
```

LOOP允许某特定语句或语句群的重复执行，实现一个简单的循环构造。在循环内的语句一直重复直循环被退出，退出通常伴随着一个LEAVE 语句。

LOOP语句可以被标注。除非*begin_label*存在，否则*end_label*不能被给出，并且如果两者都出现，它们必须是同样的。

20.2.12.4. LEAVE语句

```
LEAVE label
```

这个语句被用来退出任何被标注的流程控制构造。它和BEGIN ... END或循环一起被使用。

20.2.12.5. ITERATE语句

```
ITERATE label
```

ITERATE只可以出现在LOOP, REPEAT, 和WHILE语句内。ITERATE意思为:“再次循环。”

例如:

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN ITERATE label1; END IF;
    LEAVE label1;
  END LOOP label1;
  SET @x = p1;
END
```

20.2.12.6. REPEAT语句

```
[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]
```

REPEAT语句内的语句或语句群被重复，直至*search_condition* 为真。

REPEAT 语句可以被标注。 除非*begin_label*也存在，*end_label*才能被用，如果两者都存在，它们必须是一样的。

例如:

```
mysql> delimiter //


mysql> CREATE PROCEDURE dorepeat(p1 INT)
```

```
    -> BEGIN
    ->   SET @x = 0;
    ->   REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL dorepeat(1000)//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
+------+
| @x   |
+------+
| 1001 |
+------+
1 row in set (0.00 sec)
```

20.2.12.7. WHILE语句

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

WHILE语句内的语句或语句群被重复，直至*search_condition* 为真。

WHILE语句可以被标注。 除非*begin_label*也存在，*end_label*才能被用，如果两者都存在，它们必须是一样的。

例如:

```
CREATE PROCEDURE dowhile()
BEGIN
  DECLARE v1 INT DEFAULT 5;

  WHILE v1 > 0 DO
    ...
    SET v1 = v1 - 1;
  END WHILE;
END
```

20.3. 存储程序 、 函数、触发程序及复制：常见问题

MySQL 5.1存储程序和函数对复制起作用吗？

是的，在存储程序和函数中被执行标准行为被从主MySQL服务器复制到从服务器。有少数限制，它们在20.4节，“存储子程序和 触发程序二进制日志功能”中详述。

- 在主服务器上创建的存储程序和函数可以被复制到从服务器上么？

  是的，通过一般DDL语句执行的存储程序和函数，其在主服务器上的创建被复制到从服务器，所以目标将存在两个服务器上。对存储程序和函数的ALTER 和DROP语句也被复制。

- 行为如何在已复制的存储程序和函数里发生？

  MySQL纪录每个发生在存储程序和函数里的DML事件，并复制这些单独的行为到从服务器。执行存储程序和函数的切实调用不被复制。

- 对一起使用存储程序，函数和复制有什么特别的安全要求么？

  是的，因为一个从服务器有权限来执行任何读自主服务器的二进制日志的语句，指定的安全约束因与复制一起使用的存储程序和函数而存在。如果复制或二进制日志大体上是激活的（为point-in-time恢复的目的），那么MySQL DBA 有两个安全选项可选：

  - 任何想创建存储程序的用户必须被赋予SUPER权限。
  - 作为选择，一个DBA可以设置log_bin_trust_routine_creators系统变量为1，它将会允许有标准CREATE ROUTINE权限的人来创建一个存储程序和函数。

- 对复制存储程序和函数的行为有什么限制？

  嵌入到存储程序中的不确定（随机）或时基行不能适当地复制。随机产生的结果，仅因其本性，是你可预测的和不能被确实克隆的。因此，复制到从服务器的随机行为将不会镜像那些产生在主服务器上的。注意， 声明存储程序或函数为DETERMINISTIC或者在log_bin_trust_routine_creators中设置系统变量为0将会允许随即值操作被调用。

  此外，时基行为不能在从服务器上重新产生，因为在存储程序中通过对复制使用的二进制日志来计时这样的时基行为是不可重新产生的，因为该二进制日志仅纪录DML事件且不包括计时约束。

  最后，在大型DML行为（如大批插入）中非交互表发生错误，该非交互表可能经历复制，在复制版的非交互表中主服务器可以被部分地从DML行为更新。但是因为发生的那个错误，对从服务器没有更新。 对函数的DML行为，工作区将被用IGNORE关键词来执行，以便于在主服务器上导致错误的更新被忽略，并且不会导致错误的更新被复制到从服务器。

- 上述的限制会影响MySQL作 point-in-time恢复的能力吗？

  影响复制的同一限制会影响point-in-time恢复。

- MySQL要做什么来改正前述的限制呢？

  将来发行的MySQL预期有一个功能去选择复制该如何被处理：

  - 基于语句的复制（当前实现）。
  - 行级别复制（它将解决所有早先描述的限制）。

- 触发程序对复制起作用么？

  MySQL 5.1中的触发程序和复制象在大多数其它数据库引擎中一样工作，在那些引擎中，通过触发程序在主服务器上执行的行为不被复制到从服务器。取而代之的是，位于主MySQL服务器的表中的 触发程序需

要在那些存在于任何MySQL从服务器上的表内被创建，以便于触发程序可以也可以在从服务器上被激活。

- 一个行为如何通过从主服务器上复制到从服务器上的触发程序来执行呢？

  首先，主服务器上的触发程序必须在从服务器上重建。一旦重建了，复制流程就象其它参与到复制中的标准DML语句一样工作。例如：考虑一个已经插入触发程序AFTER的EMP表，它位于主MySQL服务器上。同样的EMP表和AFTER插入 触发程序也存在于从服务器上。复制流程可能是：

1. 对EMP做一个INSERT语句。
2. EMP上的AFTER触发程序激活。
3. INSERT语句被写进二进制日志。
4. 从服务器上的复制拾起INSERT语句给EMP表，并在从服务器上执行它。
5. 位于从服务器EMP上的AFTER触发程序激活。

20.4. 存储子程序和触发程序的二进制日志功能

，这一节介绍MySQL 5.1如何考虑二进制日志功能来处理存储子程序（程序和函数） 。这一节也适用于触发程序。

二进制日志包含修改数据库内容的SQL语句的信息。这个信息以描述修改的事件的形式保存起来。

二进制日志有两个重要目的：

· 复制的基础是主服务器发送包含在二进制日志里的事件到从服务器，从服务器执行这些事件来造成与对主服务器造成的同样的数据改变，请参阅6.2节，“复制概述”。

· 特定的数据恢复操作许要使用二进制日志。备份的文件被恢复之后，备份后纪录的二进制日志里的事件被重新执行。这些事件把数据库带从备份点的日子带到当前。请参阅5.9.2.2节，“使用备份恢复”。

MySQL中，以存储子程序的二进制日志功能引发了很多问题，这些在下面讨论中列出，作为参考信息。

除了要另外注意的之外，这些谈论假设你已经通过用--log-bin选项启动服务器允许了二进制日志功能。（如果二进制日志功能不被允许，复制将不可能，为数据恢复的二进制日志也不存在。）请参阅5.11.3节，“二进制日志”。

对存储子程序语句的二进制日志功能的特征在下面列表中描述。一些条目指出你应该注意到的问题。但是在一些情况下，有你可以更改的妇五七设置或你可以用来处理它们的工作区。

· CREATE PROCEDURE, CREATE FUNCTION, ALTER PROCEDURE,和ALTER FUNCTION 语句被写进二进制日志，CALL, DROP PROCEDURE, 和DROP FUNCTION 也一样。

尽管如此，对复制有一个安全暗示：要创建一个子程序，用户必须有CREATE ROUTINE权限,但有这个权限的用户不能写一个 子程序在从服务器上执行任何操作。因为在从服务器上的SQL线程用完全权限来运行。例如，如果主服务器和从服务器分别有服务器ID值1和2，在主服务器上的用户可能创建并调用如下一个 程序：

```
mysql> delimiter //

mysql> CREATE PROCEDURE mysp ()
```

```
    -> BEGIN
    ->   IF @@server_id=2 THEN DROP DATABASE accounting; END IF;
    -> END;
    -> //
mysql> delimiter ;
mysql> CALL mysp();
```

CREATE PROCEDURE和CALL语句将被写进二进制日志，所以从服务器将执行它们。因为从SQL线程有完全权限，它将移除accounting数据库。

要使允许二进制日志功能的服务器避免这个危险，MySQL 5.1已经要求 存储程序和函数的创建者除了通常需要的CREATE ROUTINE的权限外，还必须有SUPER 权限。类似地，要使用ALTER PROCEDURE或ALTER FUNCTION，除了ALTER ROUTINE权限外你必须有SUPER权限。没有SUPER权限，将会发生一个错误：

```
ERROR 1419 (HY000): You do not have the SUPER privilege and
binary logging is enabled (you *might* want to use the less safe
log_bin_trust_routine_creators variable)
```

你可能不想强制要求子程序创建者必须有SUPER权限。例如，你系统上所有有CREATE ROUTINE权限的用户可能是有经验的应用程序开发者。要禁止掉对SUPER权限的要求，设置log_bin_trust_routine_creators 全局系统变量为1。默认地，这个变量值为0，但你可以象这样改变这样：

```
mysql> SET GLOBAL log_bin_trust_routine_creators = 1;
```

你也可以在启动服务器之时用--log-bin-trust-routine-creators选项来设置允许这个变量。

如果二进制日志功能不被允许，log_bin_trust_routine_creators 没有被用上，子程序创建需要SUPER权限。

· 一个执行更新的非确定子程序是不可重复的，它能有两个不如意的影响：

o 它会使得从服务器不同于主服务器。

- 恢复的数据与原始数据不同。

要解决这些问题，MySQL强制做下面要求：在主服务器上，除非子程序被声明为确定性的或者不更改数据，否则创建或者替换子程序将被拒绝。这意味着当你创建一个子程序的时候，你必须要么声明它是确定性的，要么它不改变数据。两套子程序特征在这里适用：

- DETERMINISTIC和NOT DETERMINISTIC指出一个子程序是否对给定的输入总是产生同样的结果。如果没有给定任一特征，默认是NOT DETERMINISTIC，所以你必须明确指定DETERMINISTIC来 声明一个 子程序是确定性的。

使用NOW() 函数（或它的同义）或者RAND() 函数不是必要地使也一个子程序非确定性。对NOW()而言，二进制日志包括时间戳并正确复制。RAND()只要在一个 子程序内被调用一次也可以正确复制。（你可以认为子程序执行时间戳和随机数种子作为毫无疑问地输入，它们在主服务器和从服务器上是一样的。）

- CONTAINS SQL, NO SQL, READS SQL DATA, 和 MODIFIES SQL数据提供子程序是读还是写数据的信息。无论NO SQL 还是READS SQL DATA i都指出，子程序没有改变数据，但你必须明白地指明这些中的一个，因为如果任何这些特征没有被给出， 默认的特征是CONTAINS SQL。

默认地，要一个CREATE PROCEDURE 或 CREATE FUNCTION 语句被接受，DETERMINISTIC 或 NO SQL与READS SQL DATA 中的一个必须明白地指定，否则会产生如下错误：

```
ERROR 1418 (HY000): This routine has none of DETERMINISTIC, NO SQL,
```

```
or READS SQL DATA in its declaration and binary logging is enabled
(you *might* want to use the less safe log_bin_trust_routine_creators
variable)
```

如果设置log_bin_trust_routine_creators 为1, 移除对子程序必须是确定的或不修改数据的要求。

注意，子程序本性的评估是基于创建者的"诚实度"：MySQL不检查声明为确定性的子程序是否不含产生非确定性结果的语句。

- 如果子程序返回无错，CALL语句被写进二进制日志，否则就不写。当一个子程序修改数据失败了，你会得到这样的警告：

```
ERROR 1417 (HY000): A routine failed and has neither NO SQL nor
READS SQL DATA in its declaration and binary logging is enabled; if
non-transactional tables were updated, the binary log will miss their
changes
```

这个记日志行为潜在地导致问题.如果一个子程序部分地修改一个非交互表（比如一个MyISAM表able）并且返回一个错误，二进制日志将反映这些变化。要防止这种情况，你应该在 子程序中使用交互表并且在交互动作内修改表。

在一个子程序内，如果你在INSERT, DELETE, 或者UPDATE里使用IGNORE关键词来忽略错误，可能发生一个部分更新，但没有错误产生。这样的语句被记录日志，且正常复制。

- 如果一个存储函数在一个如SELECT这样不修改数据的语句内被调用，即使函数本身更改数据，函数的执行也将不被写进二进制日志里。这个记录日志的行为潜在地导致问题。假设函数myfunc()如下定义：

```
CREATE FUNCTION myfunc () RETURNS INT
BEGIN
  INSERT INTO t (i) VALUES(1);
  RETURN 0;
END;
```

按照上面定义，下面的语句修改表t，因为myfunc()修改表t, 但是语句不被写进二进制日志，因为它是一个SELECT语句：

```
SELECT myfunc();
```

对这个问题的工作区将调用在做更新的语句里做更新的函数。注意，虽然DO语句有时为了其估算表达式的副效应而被执行，DO在这里不是一个工作区，因为它不被写进二进制日志。

- 在一个子程序内执行的语句不被写进二进制日志。假如你发布下列语句：

```
CREATE PROCEDURE mysp INSERT INTO t VALUES(1);
CALL mysp;
```

对于这个例子来说，CREATE PROCEDURE 和CALL语句出现在二进制日志里，但INSERT语句并未出现。

- 在从服务器上，当决定复制哪个来自主服务器的事件时，下列限制被应用：--replicate-*-table规则不适用于CALL语句或子程序内的语句：在这些情况下，总是返回"复制！"

触发程序类似于存储函数，所以前述的评论也适用于触发程序，除了下列情况： CREATE TRIGGER没有可选

的DETERMINISTIC特征，所以触发程序被假定为总是确定性的。然而，这个假设在一些情况下是非法的。比如，UUID()函数是非确定性的（不能复制）。你应该小心在 触发程序中使用这个函数。

触发程序目前不能更新表，但是在将来会支持。因为这个原因，如果你没有SUPER权限且log_bin_trust_routine_creators 被设为0，得到的错误信息类似于存储子程序与CREATE TRIGGER产生的错误信息。

在本节中叙述的问题来自发生在SQL语句级别的二进制日志记录的事实。未来发行的MySQL期望能实现行级的二进制日志记录，记录发生在更 细致的级别并且指出哪个改变作为执行SQL的结果对单个记录而做。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第21章：触发程序

目录

21.1. CREATE TRIGGER语法
21.2. DROP TRIGGER语法
21.3. 使用触发程序

MySQL 5.1包含对触发程序的支持。触发程序是与表有关的命名数据库对象，当表上出现特定事件时，将激活该对象。例如，下述语句将创建1个表和1个INSERT触发程序。触发程序将插入表中某一列的值加在一起：

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));

mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account

    -> FOR EACH ROW SET @sum = @sum + NEW.amount;
```

在本章中，介绍了创建和撤销触发程序的语法，并给出了一些使用它们的示例。关于对触发程序使用的限制，请参见附录I：特性限制。

关于触发程序的二进制日志功能，请参见20.4节，“存储子程序和触发程序的二进制日志功能”。

21.1. CREATE TRIGGER语法

```
CREATE TRIGGER trigger_name trigger_time trigger_event

    ON tbl_name FOR EACH ROW trigger_stmt
```

触发程序是与表有关的命名数据库对象，当表上出现特定事件时，将激活该对象。

触发程序与命名为*tbl_name*的表相关。*tbl_name*必须引用永久性表。不能将触发程序与TEMPORARY表或视图关联起来。

*trigger_time*是触发程序的动作时间。它可以是BEFORE或AFTER，以指明触发程序是在激活它的语句之前或之后触发。

*trigger_event*指明了激活触发程序的语句的类型。*trigger_event*可以是下述值之一：

- INSERT：将新行插入表时激活触发程序，例如，通过INSERT、LOAD DATA和REPLACE语句。
- UPDATE：更改某一行时激活触发程序，例如，通过UPDATE语句。
- DELETE：从表中删除某一行时激活触发程序，例如，通过DELETE和REPLACE语句。

请注意，*trigger_event*与以表操作方式激活触发程序的SQL语句并不很类似，这点很重要。例如，关于INSERT的BEFORE触发程序不仅能被INSERT语句激活，也能被LOAD DATA语句激活。

可能会造成混淆的例子之一是INSERT INTO .. ON DUPLICATE UPDATE ...语法：BEFORE INSERT触发程序对于每一行将激活，后跟AFTER INSERT触发程序，或BEFORE UPDATE和AFTER UPDATE触发程序，具体情况取决于行上是否有重复键。

对于具有相同触发程序动作时间和事件的给定表，不能有两个触发程序。例如，对于某一表，不能有两个BEFORE UPDATE触发程序。但可以有1个BEFORE UPDATE触发程序和1个BEFORE INSERT触发程序，或1个BEFORE UPDATE触发程序和1个AFTER UPDATE触发程序。

*trigger_stmt*是当触发程序激活时执行的语句。如果你打算执行多个语句，可使用BEGIN ... END复合语句结构。这样，就能使用存储子程序中允许的相同语句。请参见20.2.7节，“BEGIN ... END复合语句”。

在MySQL 5.1中，可以编写包含按名称对表进行直接引用的触发程序，如下例中所示的名为testref的触发程序：

```
CREATE TABLE test1(a1 INT);
CREATE TABLE test2(a2 INT);
CREATE TABLE test3(a3 INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
CREATE TABLE test4(
  a4 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  b4 INT DEFAULT 0
);

DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON test1
  FOR EACH ROW BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
  END
|

DELIMITER ;

INSERT INTO test3 (a3) VALUES
  (NULL), (NULL), (NULL), (NULL), (NULL),
  (NULL), (NULL), (NULL), (NULL), (NULL);

INSERT INTO test4 (a4) VALUES
  (0), (0), (0), (0), (0), (0), (0), (0), (0), (0);
```

如果将下述值插入表test1，如下所示：

```
mysql> INSERT INTO test1 VALUES
    -> (1), (3), (1), (7), (1), (8), (4), (4);
Query OK, 8 rows affected (0.01 sec)
Records: 8  Duplicates: 0  Warnings: 0
```

那么4个表中的数据如下：

```
mysql> SELECT * FROM test1;
+------+
```

```
| a1   |
+------+
|    1 |
|    3 |
|    1 |
|    7 |
|    1 |
|    8 |
|    4 |
|    4 |
+------+
8 rows in set (0.00 sec)

mysql> SELECT * FROM test2;
+------+
| a2   |
+------+
|    1 |
|    3 |
|    1 |
|    7 |
|    1 |
|    8 |
|    4 |
|    4 |
+------+
8 rows in set (0.00 sec)

mysql> SELECT * FROM test3;
+----+
| a3 |
+----+
|  2 |
|  5 |
|  6 |
|  9 |
| 10 |
+----+
5 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM test4;
+----+------+
| a4 | b4   |
+----+------+
|  1 |    3 |
|  2 |    0 |
|  3 |    1 |
|  4 |    2 |
|  5 |    0 |
|  6 |    0 |
|  7 |    1 |
|  8 |    1 |
|  9 |    0 |
| 10 |    0 |
+----+------+
10 rows in set (0.00 sec)
```

使用别名OLD和NEW，能够引用与触发程序相关的表中的列。OLD.*col_name*在更新或删除它之前，引用已有行中的1列。NEW.*col_name*在更新它之后引用将要插入的新行的1列或已有行的1列。

激活触发程序时，对于触发程序引用的所有OLD和NEW列，需要具有SELECT权限，对于作为SET赋值目标的所有NEW列，需要具有UPDATE权限。

注释：目前，触发程序不会被级联的 外键动作激活。该限制将会被尽早放宽。

CREATE TRIGGER语句需要SUPER权限。

21.2. DROP TRIGGER语法

```
DROP TRIGGER [schema_name.]trigger_name
```

舍弃触发程序。方案名称（schema_name）是可选的。如果省略了schema（方案），将从当前方案中舍弃触发程序。

注释：从MySQL 5.0.10之前的MySQL版本升级到5.0.10或更高版本时（包括所有的MySQL 5.1版本），必须在升级之前舍弃所有的触发程序，并在随后重新创建它们，否则，在升级之后DROP TRIGGER不工作。关于推荐的升级步骤，请参见2.10.1节，“从5.0版升级”。

DROP TRIGGER语句需要SUPER权限。

21.3. 使用触发程序

在本节中，介绍了在MySQL 5.1中使用触发程序的方法，并介绍了在使用触发程序方面的限制。

触发程序是与表有关的命名数据库对象，当表上出现特定事件时，将激活该对象。在某些触发程序的用法中，可用于检查插入到表中的值，或对更新涉及的值进行计算。

触发程序与表相关，当对表执行INSERT、DELETE或UPDATE语句时，将激活触发程序。可以将触发程序设置为在执行语句之前或之后激活。例如，可以在从表中删除每一行之前，或在更新了每一行后激活触发程序。

要想创建触发程序或舍弃触发程序，可使用CREATE TRIGGER或DROP TRIGGER语句。关于这些语句的语法，请参见21.1节，“CREATE TRIGGER语法”和21.2节，“DROP TRIGGER语法”。

下面给出了1个简单的示例，在该示例中，针对INSERT语句，将触发程序和表关联了起来。其作用相当于累加器，能够将插入表中某一列的值加起来。

在下面的语句中，创建了1个表，并为表创建了1个触发程序：

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));

mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account

    -> FOR EACH ROW SET @sum = @sum + NEW.amount;
```

CREATE TRIGGER语句创建了与账户表相关的、名为ins_sum的触发程序。它还包括一些子句，这些子句指定了触发程序激活时间、触发程序事件、以及激活触发程序时作些什么：

- 关键字BEFORE指明了触发程序的动作时间。在本例中，应在将每一行插入表之前激活触发程序。这类允许的其他关键字是AFTER。

- 关键字INSERT指明了激活触发程序的事件。在本例中，INSERT语句将导致触发程序的激活。你也可以为DELETE和UPDATE语句创建触发程序。

- 跟在FOR EACH ROW后面的语句定义了每次激活触发程序时将执行的程序，对于受触发语句影响的每一行执行一次。在本例中，触发的语句是简单的SET语句，负责将插入amount列的值加起来。该语句将列引用为NEW.amount，意思是“将要插入到新行的amount列的值”。

要想使用触发程序，将累加器变量设置为0，执行INSERT语句，然后查看变量的值：

```
mysql> SET @sum = 0;

mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);

mysql> SELECT @sum AS 'Total amount inserted';

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

| Total amount inserted |

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

| 1852.48               |

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

在本例中，执行了INSERT语句后，@sum的值是14.98 + 1937.50 – 100，或1852.48。

要想销毁触发程序，可使用DROP TRIGGER语句。如果触发程序不在默认的方案中，必须指定方案名称：

```
mysql> DROP TRIGGER test.ins_sum;
```

触发程序名称存在于方案的名称空间内，这意味着，在1个方案中，所有的触发程序必须具有唯一的名称。位于不同方案中的触发程序可以具有相同的名称。

在1个方案中，所有的触发程序名称必须是唯一的，除了该要求外，对于能够创建的触发程序的类型还存在其他限制。尤其是，对于具有相同触发时间和触发事件的表，不能有2个触发程序。例如，不能为某一表定义2个BEFORE INSERT触发程序或2个AFTER UPDATE触发程序。这几乎不是有意义的限制，这是因为，通过在FOR EACH ROW之后使用BEGIN ... END复合语句结构，能够定义执行多条语句的触发程序。请参见本节后

面给出的示例。

此外，激活触发程序时，对触发程序执行的语句也存在一些限制：

- 触发程序不能调用将数据返回客户端的存储程序，也不能使用采用CALL语句的动态SQL（允许存储程序通过参数将数据返回触发程序）。

- 触发程序不能使用以显式或隐式方式开始或结束事务的语句，如START TRANSACTION、COMMIT或ROLLBACK。

使用OLD和NEW关键字，能够访问受触发程序影响的行中的列（OLD和NEW不区分大小写）。在INSERT触发程序中，仅能使用NEW.*col_name*，没有旧行。在DELETE触发程序中，仅能使用OLD.*col_name*，没有新行。在UPDATE触发程序中，可以使用OLD.*col_name*来引用更新前的某一行的列，也能使用NEW.*col_name*来引用更新后的行中的列。

用OLD命名的列是只读的。你可以引用它，但不能更改它。对于用NEW命名的列，如果具有SELECT权限，可引用它。在BEFORE触发程序中，如果你具有UPDATE权限，可使用"SET NEW.*col_name* = *value*"更改它的值。这意味着，你可以使用触发程序来更改将要插入到新行中的值，或用于更新行的值。

在BEFORE触发程序中，AUTO_INCREMENT列的NEW值为0，不是实际插入新记录时将自动生成的序列号。

OLD和NEW是对触发程序的MySQL扩展。

通过使用BEGIN ... END结构，能够定义执行多条语句的触发程序。在BEGIN块中，还能使用存储子程序中允许的其他语法，如条件和循环等。但是，正如存储子程序那样，定义执行多条语句的触发程序时，如果使用mysql程序来输入触发程序，需要重新定义语句分隔符，以便能够在触发程序定义中使用字符"**;**"。在下面的示例中，演示了这些要点。在该示例中，定义了1个UPDATE触发程序，用于检查更新每一行时将使用的新值，并更改值，使之位于0～100的范围内。它必须是BEFORE触发程序，这是因为，需要在将值用于更新行之前对其进行检查：

```
mysql> delimiter //

mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account

    -> FOR EACH ROW

    -> BEGIN

    ->     IF NEW.amount < 0 THEN

    ->         SET NEW.amount = 0;

    ->     ELSEIF NEW.amount > 100 THEN

    ->         SET NEW.amount = 100;

    ->     END IF;

    -> END;//

mysql> delimiter ;
```

较为简单的方法是，单独定义存储程序，然后使用简单的CALL语句从触发程序调用存储程序。如果你打算从数个触发程序内部调用相同的子程序，该方法也很有帮助。

在触发程序的执行过程中，MySQL处理错误的方式如下：

- 如果BEFORE触发程序失败，不执行相应行上的操作。

- 仅当BEFORE触发程序（如果有的话）和行操作均已成功执行，才执行AFTER触发程序。

- 如果在BEFORE或AFTER触发程序的执行过程中出现错误，将导致调用触发程序的整个语句的失败。
- 对于事务性表，如果触发程序失败（以及由此导致的整个语句的失败），该语句所执行的所有更改将回滚。对于非事务性表，不能执行这类回滚，因而，即使语句失败，失败之前所作的任何更改依然有效。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第22章：视图

目录

22.1. ALTER VIEW语法
22.2. CREATE VIEW语法
22.3. DROP VIEW语法
22.4. SHOW CREATE VIEW语法

在5.1版MySQL服务器中提供了视图功能（包括可更新视图）。

本章讨论了下述主题:

- 使用CREATE VIEW或ALTER VIEW创建或更改视图。
- 使用DROP VIEW销毁视图。
- 使用SHOW CREATE VIEW显示视图元数据。

关于使用视图方面的限制，请参见附录I：特性限制。

如果你已从不支持视图的较旧版本升级到MySQL 5.1，要想使用视图，应升级授权表，使之包含与视图有关的权限。请参见2.10.2节，“升级授权表”。

22.1. ALTER VIEW语法

```
ALTER [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

该语句用于更改已有视图的定义。其语法与CREATE VIEW类似。请参见22.2节，“CREATE VIEW语法”。该语句需要具有针对视图的CREATE VIEW和DROP权限，也需要针对SELECT语句中引用的每一列的某些权限。

22.2. CREATE VIEW语法

```
CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

该语句能创建新的视图，如果给定了OR REPLACE子句，该语句还能替换已有的视图。*select_statement*是一种SELECT语句，它给出了视图的定义。该语句可从基表或其他视图进行选择。

该语句要求具有针对视图的CREATE VIEW权限，以及针对由SELECT语句选择的每一列上的某些权限。对于在SELECT语句中其他地方使用的列，必须具有SELECT权限。如果还有OR REPLACE子句，必须在视图上具有DROP权限。

视图属于数据库。在默认情况下，将在当前数据库创建新视图。要想在给定数据库中明确创建视图，创建时，

应将名称指定为*db_name.view_name*。

```
mysql> CREATE VIEW test.v AS SELECT * FROM t;
```

表和视图共享数据库中相同的名称空间，因此，数据库不能包含具有相同名称的表和视图。

视图必须具有唯一的列名，不得有重复，就像基表那样。默认情况下，由SELECT语句检索的列名将用作视图列名。要想为视图列定义明确的名称，可使用可选的*column_list*子句，列出由逗号隔开的ID。*column_list*中的名称数目必须等于SELECT语句检索的列数。

SELECT语句检索的列可以是对表列的简单引用。也可以是使用函数、常量值、操作符等的表达式。

对于SELECT语句中不合格的表或视图，将根据默认的数据库进行解释。通过用恰当的数据库名称限定表或视图名，视图能够引用表或其他数据库中的视图。

能够使用多种SELECT语句创建视图。视图能够引用基表或其他视图。它能使用联合、UNION和子查询。SELECT甚至不需引用任何表。在下面的示例中，定义了从另一表选择两列的视图，并给出了根据这些列计算的表达式：

```
mysql> CREATE TABLE t (qty INT, price INT);

mysql> INSERT INTO t VALUES(3, 50);

mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;

mysql> SELECT * FROM v;

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

| qty  | price | value |

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

|    3 |    50 |   150 |

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

视图定义服从下述限制：

- SELECT语句不能包含FROM子句中的子查询。
- SELECT语句不能引用系统或用户变量。
- SELECT语句不能引用预处理语句参数。
- 在存储子程序内，定义不能引用子程序参数或局部变量。
- 在定义中引用的表或视图必须存在。但是，创建了视图后，能够舍弃定义引用的表或视图。要想检查视图定义是否存在这类问题，可使用CHECK TABLE语句。
- 在定义中不能引用TEMPORARY表，不能创建TEMPORARY视图。
- 在视图定义中命名的表必须已存在。
- 不能将触发程序与视图关联在一起。

在视图定义中允许使用ORDER BY，但是，如果从特定视图进行了选择，而该视图使用了具有自己ORDER BY的语句，它将被忽略。

对于定义中的其他选项或子句，它们将被增加到引用视图的语句的选项或子句中，但效果未定义。例如，如果在视图定义中包含LIMIT子句，而且从特定视图进行了选择，而该视图使用了具有自己LIMIT子句的语句，那么

对使用哪个LIMIT未作定义。相同的原理也适用于其他选项，如跟在SELECT关键字后的ALL、DISTINCT或SQL_SMALL_RESULT，并适用于其他子句，如INTO、FOR UPDATE、LOCK IN SHARE MODE、以及PROCEDURE。

如果创建了视图，并通过更改系统变量更改了查询处理环境，会影响从视图获得的结果：

```
mysql> CREATE VIEW v AS SELECT CHARSET(CHAR(65)), COLLATION(CHAR(65));
Query OK, 0 rows affected (0.00 sec)

mysql> SET NAMES 'latin1';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM v;
+-------------------+---------------------+
| CHARSET(CHAR(65)) | COLLATION(CHAR(65)) |
+-------------------+---------------------+
| latin1            | latin1_swedish_ci   |
+-------------------+---------------------+
1 row in set (0.00 sec)

mysql> SET NAMES 'utf8';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM v;
+-------------------+---------------------+
| CHARSET(CHAR(65)) | COLLATION(CHAR(65)) |
+-------------------+---------------------+
| utf8              | utf8_general_ci     |
+-------------------+---------------------+
1 row in set (0.00 sec)
```

可选的ALGORITHM子句是对标准SQL的MySQL扩展。ALGORITHM可取三个值：MERGE、TEMPTABLE或UNDEFINED。如果没有ALGORITHM子句，默认算法是UNDEFINED（未定义的）。算法会影响MySQL处理视图的方式。

对于MERGE，会将引用视图的语句的文本与视图定义合并起来，使得视图定义的某一部分取代语句的对应部分。

对于TEMPTABLE，视图的结果将被置于临时表中，然后使用它执行语句。

对于UNDEFINED，MySQL将选择所要使用的算法。如果可能，它倾向于MERGE而不是TEMPTABLE，这是因为MERGE通常更有效，而且如果使用了临时表，视图是不可更新的。

明确选择TEMPTABLE的1个原因在于，创建临时表之后、并在完成语句处理之前，能够释放基表上的锁定。与MERGE算法相比，锁定释放的速度更快，这样，使用视图的其他客户端不会被屏蔽过长时间。

视图算法可以是UNDEFINED，有三种方式：

- 在CREATE VIEW语句中没有ALGORITHM子句。
- CREATE VIEW语句有1个显式ALGORITHM = UNDEFINED子句。
- 为仅能用临时表处理的视图指定ALGORITHM = MERGE。在这种情况下，MySQL将生成告警，并将算法设置为UNDEFINED。

正如前面所介绍的那样，通过将视图定义中的对应部分合并到引用视图的语句中，对MERGE进行处理。在下面的示例中，简要介绍了MERGE的工作方式。在该示例中，假定有1个具有下述定义的视图v_merge：

```
CREATE ALGORITHM = MERGE VIEW v_merge (vc1, vc2) AS

SELECT c1, c2 FROM t WHERE c3 > 100;
```

示例1：假定发出了下述语句：

```
SELECT * FROM v_merge;
```

MySQL以下述方式处理语句：

- v_merge成为t
- *成为vc1、vc2，与c1、c2对应
- 增加视图WHERE子句

所产生的将执行的语句为：

```
SELECT c1, c2 FROM t WHERE c3 > 100;
```

示例2：假定发出了下述语句：

```
SELECT * FROM v_merge WHERE vc1 < 100;
```

该语句的处理方式与前面介绍的类似，但vc1 < 100变为c1 < 100，并使用AND连接词将视图的WHERE子句添加到语句的WHERE子句中（增加了圆括号以确保以正确的优先顺序执行子句部分）。所得的将要执行的语句变为：

```
SELECT c1, c2 FROM t WHERE (c3 > 100) AND (c1 < 100);
```

事实上，将要执行的语句是具有下述形式的WHERE子句：

```
WHERE (select WHERE) AND (view WHERE)
```

MERGE算法要求视图中的行和基表中的行具有一对一的关系。如果不具有该关系。必须使用临时表取而代之。如果视图包含下述结构中的任何一种，将失去一对一的关系：

- 聚合函数（SUM(), MIN(), MAX(), COUNT()等）。
- DISTINCT
- GROUP BY
- HAVING
- UNION或UNION ALL

- 仅引用文字值（在该情况下，没有基本表）。

某些视图是可更新的。也就是说，可以在诸如UPDATE、DELETE或INSERT等语句中使用它们，以更新基表的内容。对于可更新的视图，在视图中的行和基表中的行之间必须具有一对一的关系。还有一些特定的其他结构，这类结构会使得视图不可更新。更具体地讲，如果视图包含下述结构中的任何一种，那么它就是不可更新的：

- 聚合函数（SUM(), MIN(), MAX(), COUNT()等）。
- DISTINCT
- GROUP BY
- HAVING
- UNION或UNION ALL
- 位于选择列表中的子查询
- Join
- FROM子句中的不可更新视图
- WHERE子句中的子查询，引用FROM子句中的表。
- 仅引用文字值（在该情况下，没有要更新的基本表）。
- ALGORITHM = TEMPTABLE（使用临时表总会使视图成为不可更新的）。

关于可插入性（可用INSERT语句更新），如果它也满足关于视图列的下述额外要求，可更新的视图也是可插入的：

- 不得有重复的视图列名称。
- 视图必须包含没有默认值的基表中的所有列。
- 视图列必须是简单的列引用而不是导出列。导出列不是简单的列引用，而是从表达式导出的。下面给出了一些导出列示例：

```
3.14159
col1 + 3
UPPER(col2)
col3 / col4
(subquery)
```

混合了简单列引用和导出列的视图是不可插入的，但是，如果仅更新非导出列，视图是可更新的。考虑下述视图：

```
CREATE VIEW v AS SELECT col1, 1 AS col2 FROM t;
```

该视图是不可插入的，这是因为col2是从表达式导出的。但是，如果更新时不更新col2，它是可更新的。这类更新是允许的：

```
UPDATE v SET col1 = 0;
```

下述更新是不允许的，原因在于，它试图更新导出列：

```
UPDATE v SET col2 = 0;
```

在某些情况下，能够更新多表视图，假定它能使用MERGE算法进行处理。为此，视图必须使用内部联合（而不是外部联合或UNION）。此外，仅能更新视图定义中的单个表，因此，SET子句必须仅命名视图中某一表的列。即使从理论上讲也是可更新的，不允许使用UNION ALL的视图，这是因为，在实施中将使用临时表来处理它们。

对于多表可更新视图，如果是将其插入单个表中，INSERT能够工作。不支持DELETE。

对于可更新视图，可给定WITH CHECK OPTION子句来防止插入或更新行，除非作用在行上的*select_statement*中的WHERE子句为"真"。

在关于可更新视图的WITH CHECK OPTION子句中，当视图是根据另一个视图定义的时，LOCAL和CASCADED关键字决定了检查测试的范围。LOCAL关键字对CHECK OPTION进行了限制，使其仅作用在定义的视图上，CASCADED会对将进行评估的基表进行检查。如果未给定任一关键字，默认值为CASCADED。请考虑下述表和视图集合的定义：

```
mysql> CREATE TABLE t1 (a INT);

mysql> CREATE VIEW v1 AS SELECT * FROM t1 WHERE a < 2

    -> WITH CHECK OPTION;

mysql> CREATE VIEW v2 AS SELECT * FROM v1 WHERE a > 0

    -> WITH LOCAL CHECK OPTION;

mysql> CREATE VIEW v3 AS SELECT * FROM v1 WHERE a > 0

    -> WITH CASCADED CHECK OPTION;
```

这里，视图v2和v3是根据另一视图v1定义的。v2具有LOCAL检查选项，因此，仅会针对v2检查对插入项进行测试。v3具有CASCADED检查选项，因此，不仅会针对它自己的检查对插入项进行测试，也会针对基本视图的检查对插入项进行测试。在下面的语句中，介绍了这些差异：

```
ql> INSERT INTO v2 VALUES (2);

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO v3 VALUES (2);

ERROR 1369 (HY000): CHECK OPTION failed 'test.v3'
```

视图的可更新性可能会受到系统变量updatable_views_with_limit的值的影响。请参见5.3.3节，"服务器系统变量"。

INFORMATION_SCHEMA包含1个VIEWS表，从该表可获取关于视图对象的信息。请参见23.1.15节，"INFORMATION_SCHEMA VIEWS表"。

22.3. DROP VIEW语法

```
DROP VIEW [IF EXISTS]

    view_name [, view_name] ...

    [RESTRICT | CASCADE]
```

DROP VIEW能够删除1个或多个视图。必须在每个视图上拥有DROP权限。

可以使用关键字IF EXISTS来防止因不存在的视图而出错。给定了该子句时，将为每个不存在的视图生

成NOTE。请参见13.5.4.22节，“SHOW WARNINGS语法”。

如果给定了RESTRICT和CASCADE，将解析并忽略它们。

22.4. SHOW CREATE VIEW语法

```
SHOW CREATE VIEW view_name
```

该语句给出了1个创建给定视图的CREATE VIEW语句。

```
mysql> SHOW CREATE VIEW v;
+------+--------------------------------------------------+
| View | Create View                                      |
+------+--------------------------------------------------+
| v    | CREATE VIEW `test`.`v` AS select 1 AS `a`,2 AS `b` |
+------+--------------------------------------------------+
```

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第23章：INFORMATION_SCHEMA信息数据库

目录

23.1. INFORMATION_SCHEMA表

23.1.1. INFORMATION_SCHEMA SCHEMATA表
23.1.2. INFORMATION_SCHEMA TABLES表
23.1.3. INFORMATION_SCHEMA COLUMNS表
23.1.4. INFORMATION_SCHEMA STATISTICS表
23.1.5. INFORMATION_SCHEMA USER_PRIVILEGES表
23.1.6. INFORMATION_SCHEMA SCHEMA_PRIVILEGES表
23.1.7. INFORMATION_SCHEMA TABLE_PRIVILEGES表
23.1.8. INFORMATION_SCHEMA COLUMN_PRIVILEGES表
23.1.9. INFORMATION_SCHEMA CHARACTER_SETS表
23.1.10. INFORMATION_SCHEMA COLLATIONS表
23.1.11. INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY表
23.1.12. INFORMATION_SCHEMA TABLE_CONSTRAINTS表
23.1.13. INFORMATION_SCHEMA KEY_COLUMN_USAGE表
23.1.14. INFORMATION_SCHEMA ROUTINES表
23.1.15. INFORMATION_SCHEMA VIEWS表
23.1.16. INFORMATION_SCHEMA TRIGGERS表
23.1.17. 其他INFORMATION_SCHEMA表

23.2. SHOW语句的扩展

INFORMATION_SCHEMA提供了访问数据库元数据的方式。

元数据是关于数据的数据，如数据库名或表名，列的数据类型，或访问权限等。有些时候用于表述该信息的其他术语包括"数据词典"和"系统目录"。

例如：

```
mysql> SELECT table_name, table_type, engine
-> FROM information_schema.tables
-> WHERE table_schema = 'db5'
-> ORDER BY table_name DESC;
```

table_name（表名）	table_type（表类型）	引擎
v56	VIEW（视图）	NULL
v3	VIEW（视图）	NULL
v2	VIEW（视图）	NULL
v	VIEW（视图）	NULL
tables	BASE TABLE（基本表）	MyISAM
t7	BASE TABLE（基本表）	MyISAM
t3	BASE TABLE（基本表）	MyISAM
t2	BASE TABLE（基本表）	MyISAM
t	BASE TABLE（基本表）	MyISAM
pk	BASE TABLE（基本表）	InnoDB
loop	BASE TABLE（基本表）	MyISAM
kurs	BASE TABLE（基本表）	MyISAM
k	BASE TABLE（基本表）	MyISAM
into	BASE TABLE（基本表）	MyISAM
goto	BASE TABLE（基本表）	MyISAM

fk2	BASE TABLE（基本表）	InnoDB
fk	BASE TABLE（基本表）	InnoDB

集合中含17行（0.01秒）。

解释：该语句请求按逆向字母顺序列出数据库db5中的所有表，但仅显示三种信息：表名，表类型，以及表引擎。

INFORMATION_SCHEMA是信息数据库，其中保存着关于MySQL服务器所维护的所有其他数据库的信息。在INFORMATION_SCHEMA中，有数个只读表。它们实际上是视图，而不是基本表，因此，你将无法看到与之相关的任何文件。

每位MySQL用户均有权访问这些表，但仅限于表中的特定行，在这类行中含有用户具有恰当访问权限的对象。

SELECT的优点

SELECT ... FROM INFORMATION_SCHEMA语句的目的在于提供一种更为一致的方式，以访问MySQL所支持的各种SHOW语句（SHOW DATABASES、SHOW TABLES等等）提供的信息。与SHOW相比，使用SELECT有多项优点"

- 符合Codd规则。也就是说，所有访问均是在表上进行的。
- 不需要了解新语句的语法。由于他们已知道SELECT的工作方式，仅需了解对象名即可。
- 实现人无需操心增加关键词方面的事宜。
- 有数百万种可能的输出变化，而不是一种。这样，就为对元数据有不同需求的应用程序提供了更高的灵活性。
- 由于其他DBMS也采用了这类方式，移植更为容易。

然而，由于SHOW在MySQL的雇员和用户中十分流行，如果SHOW消失，可能会导致混乱，因此传统的语法方式无法给出消除SHOW的足够理由。事实上，在MySQL 5.1中，还对SHOW进行了多项增强。关于这方面的介绍，请参见23.2节，"SHOW语句的扩展"。

标准

在MySQL中，INFORMATION_SCHEMA表结构的实施遵从"ANSI/ISO SQL:2003标准，第11部分纲要"。我们的目的在于，获得与SQL:2003核心特性F021"基本信息方案"的近似兼容。

SQL服务器2000（也遵从该标准）的用户可能已注意到它们高度的相似性。但是，MySQL略去了与我们的实施方式不相关的众多列，并添加了一些MySQL特有的列。其中一种列就是INFORMATION_SCHEMA.TABLES表中的引擎列。

尽管其他DBMS使用了不同的名称，如syscat或系统，但标准名称是INFORMATION_SCHEMA。

事实上，尽管不需要生成名为INFORMATION_SCHEMA的文件，我们仍提供了名为INFORMATION_SCHEMA的新数据库。可以使用USE语句将INFORMATION_SCHEMA选择为默认数据库，但访问该数据库中所含表的唯一方式是使用SELECT语句。不能在其中插入内容，不能更新它们，也不能删除其中的内容。

权限

当前权限（SHOW）要求和SELCET权限要求不存在差别。在任何一种情况下，要想查看关于它的信息，需要对某类对象拥有特定权限。

23.1. INFORMATION_SCHEMA表

23.1.1. INFORMATION_SCHEMA SCHEMATA表
23.1.2. INFORMATION_SCHEMA TABLES表
23.1.3. INFORMATION_SCHEMA COLUMNS表
23.1.4. INFORMATION_SCHEMA STATISTICS表
23.1.5. INFORMATION_SCHEMA USER_PRIVILEGES表

23.1.6. INFORMATION_SCHEMA SCHEMA_PRIVILEGES表
23.1.7. INFORMATION_SCHEMA TABLE_PRIVILEGES表
23.1.8. INFORMATION_SCHEMA COLUMN_PRIVILEGES表
23.1.9. INFORMATION_SCHEMA CHARACTER_SETS表
23.1.10. INFORMATION_SCHEMA COLLATIONS表
23.1.11. INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY表
23.1.12. INFORMATION_SCHEMA TABLE_CONSTRAINTS表
23.1.13. INFORMATION_SCHEMA KEY_COLUMN_USAGE表
23.1.14. INFORMATION_SCHEMA ROUTINES表
23.1.15. INFORMATION_SCHEMA VIEWS表
23.1.16. INFORMATION_SCHEMA TRIGGERS表
23.1.17. 其他INFORMATION_SCHEMA表

下述章节说明

在下面的章节中，我们选择了INFORMATION_SCHEMA中的表和列。对于每一列，有三类信息：

- "标准名称"：指明了列的标准SQL名称。
- "SHOW名称"：指明了最近SHOW语句中的等效字段名，如果有的话。
- "注释"给出了适用的附加信息。

为了避免使用标准或DB2、SQL服务器或Oracle中保留的名称，我们更改了标注为"SQL扩展"的列名。（例如，在TABLES表中，我们将COLLATION改为TABLE_COLLATION）。请参见本文末尾处给出的保留字列表。http://www.dbazine.com/gulutzan5.shtml。

字符列（例如TABLES.TABLE_NAME）定义通常是VARCHAR(*N*) CHARACTER SET utf8，其中，N至少为64。

在每一部分中，指明了等效于从INFORMATION_SCHEMA中检索信息的SELECT语句的SHOW语句，或者不存在这类语句。

注释：目前，有一些丢失的列和一些混乱的列。我们正在着手解决该问题，并随着变化情况更新文档。

23.1.1. INFORMATION_SCHEMA SCHEMATA表

该方案是数据库，因此SCHEMATA表提供了关于数据库的信息。

标准名称	**SHOW**名称	注释
CATALOG_NAME	-	NULL
SCHEMA_NAME		Database
DEFAULT_CHARACTER_SET_NAME		
DEFAULT_COLLATION_NAME		
SQL_PATH		NULL

注释：SQL_PATH列的之总为NULL。

下述语句是等效的：

```
SELECT SCHEMA_NAME AS `Database
FROM INFORMATION_SCHEMA.SCHEMATA
[WHERE SCHEMA_NAME LIKE 'wild']
```

```
SHOW DATABASES
[LIKE 'wild']
```

23.1.2. INFORMATION_SCHEMA TABLES表

TABLES表给出了关于数据库中的表的信息。

标准名称	**SHOW**名称	注释
TABLE_CATALOG		NULL
TABLE_SCHEMA	Table_...	
TABLE_NAME	Table_...	
TABLE_TYPE		
ENGINE	Engine	MySQL扩展
VERSION	Version	MySQL扩展
ROW_FORMAT	Row_format	MySQL扩展
TABLE_ROWS	Rows	MySQL扩展
AVG_ROW_LENGTH	Avg_row_length	MySQL扩展
DATA_LENGTH	Data_length	MySQL扩展
MAX_DATA_LENGTH	Max_data_length	MySQL扩展
INDEX_LENGTH	Index_length	MySQL扩展
DATA_FREE	Data_free	MySQL扩展
AUTO_INCREMENT	Auto_increment	MySQL扩展
CREATE_TIME	Create_time	MySQL扩展
UPDATE_TIME	Update_time	MySQL扩展
CHECK_TIME	Check_time	MySQL扩展
TABLE_COLLATION	Collation	MySQL扩展
CHECKSUM	Checksum	MySQL扩展
CREATE_OPTIONS	Create_options	MySQL扩展
TABLE_COMMENT	Comment	MySQL扩展

注释:

- TABLE_SCHEMA和TABLE_NAME是SHOW显示中的单个字段，例如Table_in_db1。
- TABLE_TYPE（表类型）应是BASE TABLE（基本表）或VIEW（视图）。如果表是临时性的，TABLE_TYPE = TEMPORARY。（没有临时视图，因此，因此不存在歧义）。
- 如果表位于INFORMATION_SCHEMA数据库中，TABLE_ROWS列为NULL。对于InnoDB表，在SQL优化中，行计数仅是大概估计值。
- 没有关于表默认字符集的任何信息。TABLE_COLLATION处于关闭状态，原因在于校对名称以字符集名称开头。

下述语句是等效的:

```
SELECT table_name FROM INFORMATION_SCHEMA.TABLES
  [WHERE table_schema = 'db_name']
```

```
  [WHERE|AND table_name LIKE 'wild']


SHOW TABLES
  [FROM db_name]
  [LIKE 'wild']
```

23.1.3. INFORMATION_SCHEMA COLUMNS表

COLUMNS表给出了表中的列信息。

标准名称	**SHOW**名称	注释
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME	Field	
ORDINAL_POSITION		参见注释
COLUMN_DEFAULT	Default	
IS_NULLABLE	Null	
DATA_TYPE	Type	
CHARACTER_MAXIMUM_LENGTH	Type	
CHARACTER_OCTET_LENGTH		
NUMERIC_PRECISION	Type	
NUMERIC_SCALE	Type	
CHARACTER_SET_NAME		
COLLATION_NAME	Collation	
COLUMN_TYPE	Type	MySQL扩展
COLUMN_KEY	Key	MySQL扩展
EXTRA	Extra	MySQL扩展
COLUMN_COMMENT	Comment	MySQL扩展

注释:

· 在SHOW中，类型显示包括来自数个不同COLUMNS列的值。

· ORDINAL_POSITION有必要，这是因为，你可能会在某一天需要ORDER BY ORDINAL_POSITION（按ORDINAL_POSITION排序）。不同于SHOW，SELECT没有自动排序功能。

· CHARACTER_OCTET_LENGTH应与CHARACTER_MAXIMUM_LENGTH相同，但多字节字符集除外。

· CHARACTER_SET_NAME可由Collation（校对）导出。例如，如果给出了“SHOW FULL COLUMNS FROM t”，在Collation（校对）列中将见到latin1_swedish_ci的值，字符集由第1个下划线前的名称指明。latin1.

下述语句是等效的:

```
SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT
  FROM INFORMATION_SCHEMA.COLUMNS
```

```
  WHERE table_name = 'tbl_name'
  [AND table_schema = 'db_name']
  [AND column_name LIKE 'wild']


SHOW COLUMNS
  FROM tbl_name
  [FROM db_name]
  [LIKE wild]
```

23.1.4. INFORMATION_SCHEMA STATISTICS表

STATISTICS表给出了关于表索引的信息。

标准名称	**SHOW**名称	注释
TABLE_CATALOG		NULL
TABLE_SCHEMA		=数据库
TABLE_NAME	Table	
NON_UNIQUE	Non_unique	
INDEX_SCHEMA		=数据库
INDEX_NAME	Key_name	
SEQ_IN_INDEX	Seq_in_index	
COLUMN_NAME	Column_name	
COLLATION	Collation	
CARDINALITY	Cardinality	
SUB_PART	Sub_part	MySQL扩展
PACKED	Packed	MySQL扩展
NULLABLE	Null	MySQL扩展
INDEX_TYPE	Index_type	MySQL扩展
COMMENT	Comment	MySQL扩展

注释:

- 没有关于这些索引的标准表。上面的列表与SQL服务器2000中sp_statistics返回的值类似。不同之处在于用CATALOG替换了QUALIFIER，并用SCHEMA替换了OWNER。

显而易见，前述表和SHOW INDEX的输出均是由相同的父对象导出的。因此，相关性已关闭。

下述语句是等效的:

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
  WHERE table_name = 'tbl_name'
  [AND table_schema = 'db_name']


SHOW INDEX
```

```
FROM tbl_name
[FROM db_name]
```

23.1.5. INFORMATION_SCHEMA USER_PRIVILEGES表

USER_PRIVILEGES（用户权限）表给出了关于全程权限的信息。该信息源自mysql.user授权表。

标准名称	**SHOW**名称	注释
GRANTEE		例如“user'@'host”
TABLE_CATALOG		NULL
PRIVILEGE_TYPE		
IS_GRANTABLE		

注释:

- 这是一个非标准表。其值来自mysql.user表。

23.1.6. INFORMATION_SCHEMA SCHEMA_PRIVILEGES表

SCHEMA_PRIVILEGES（方案权限）表给出了关于方案（数据库）权限的信息。该信息来自mysql.db授权表。

标准名称	**SHOW**名称	注释
GRANTEE		例如“user'@'host”
TABLE_CATALOG		NULL
TABLE_SCHEMA		
PRIVILEGE_TYPE		
IS_GRANTABLE		

注释:

- 这是一个非标准表。其值来自mysql.db表。

23.1.7. INFORMATION_SCHEMA TABLE_PRIVILEGES表

TABLE_PRIVILEGES（表权限）表给出了关于表权限的信息。该信息源自mysql.tables_priv授权表。

标准名称	**SHOW**名称	注释
GRANTEE		例如“user'@'host”
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
PRIVILEGE_TYPE		
IS_GRANTABLE		

下述语句不等效:

```
SELECT ... FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES
```

```
SHOW GRANTS ...
```

PRIVILEGE_TYPE可以包含这些值之一（仅能一个）：SELECT、INSERT、UPDATE、REFERENCES、ALTER、INDEX、DROP、CREATE VIEW。

23.1.8. INFORMATION_SCHEMA COLUMN_PRIVILEGES表

COLUMN_PRIVILEGES（列权限）表给出了关于列权限的信息。该信息源自mysql.columns_priv授权表。

标准名称	**SHOW**名称	注释
GRANTEE		例如“user'@'host”
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME		
PRIVILEGE_TYPE		
IS_GRANTABLE		

注释：

- 在SHOW FULL COLUMNS（显示完整列）的输出中，权限值位于一个字段并采用小写形式，例如select、insert、update、references。在COLUMN_PRIVILEGES中，每种权限占一行，并为大写形式。
- PRIVILEGE_TYPE可以包含这些值之一（仅能一个）：SELECT, INSERT, UPDATE, REFERENCES.
- 如果用户有GRANT OPTION权限，那么IS_GRANTABLE应为YES。否则，IS_GRANTABLE应为NO。在输出中，不会将GRANT OPTION作为单独权限列出。

下述语句不等效：

```
SELECT ... FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES
```

```
SHOW GRANTS ...
```

23.1.9. INFORMATION_SCHEMA CHARACTER_SETS表

CHARACTER_SETS（字符集）表提供了关于可用字符集的信息。

标准名称	**SHOW**名称	注释
CHARACTER_SET_NAME	Charset	
DEFAULT_COLLATE_NAME	Default collation	
DESCRIPION	Description	MySQL扩展
MAXLEN	Maxlen	MySQL扩展

注释：

- 我们增加了两个非标准列，分别对应于SHOW CHARACTER SET输出的Description（描述）和Maxlen（最大长度）列。

下述语句是等效的:

```
SELECT * FROM INFORMATION_SCHEMA.CHARACTER_SETS
  [WHERE name LIKE 'wild']

SHOW CHARACTER SET
  [LIKE 'wild']
```

23.1.10. INFORMATION_SCHEMA COLLATIONS表

COLLATIONS表提供了关于各字符集的对照信息。

标准名称	**SHOW**名称	注释
COLLATION_NAME	Collation	

注释:

- 我们增加了5个非标准列，分别对应于SHOW COLLATION输出的Charset、Id、Default、Compiled和Sortlen列。

下述语句是等效的:

```
SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLLATIONS
  [WHERE collation_name LIKE 'wild']

SHOW COLLATION
  [LIKE 'wild']
```

23.1.11. INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY表

COLLATION_CHARACTER_SET_APPLICABILITY表指明了可用于校对的字符集。这些列等效于SHOW COLLATION的前两个显示字段。

标准名称	**SHOW**名称	注释
COLLATION_NAME	Collation	
CHARACTER_SET_NAME	Charset	

23.1.12. INFORMATION_SCHEMA TABLE_CONSTRAINTS表

TABLE_CONSTRAINTS表描述了存在约束的表。

标准名称	**SHOW**名称	注释
CONSTRAINT_CATALOG		NULL
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_SCHEMA		

TABLE_NAME		
CONSTRAINT_TYPE		

注释：

· CONSTRAINT_TYPE的值可以是UNIQUE（唯一）、PRIMARY KEY（主键）或FOREIGN KEY（外键）。

· 当Non_unique字段为0时，UNIQUE和PRIMARY KEY信息与SHOW INDEX输出的Key_name字段中给出的信息基本相同。

· CONSTRAINT_TYPE列可包含下述值之一：UNIQUE、PRIMARY KEY、FOREIGN KEY、CHECK。这是一个CHAR（非ENUM）列。在我们支持CHECK前，CHECK值不可用。

23.1.13. INFORMATION_SCHEMA KEY_COLUMN_USAGE表

KEY_COLUMN_USAGE表描述了具有约束的键列。

标准名称	**SHOW**名称	注释
CONSTRAINT_CATALOG		NULL
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_CATALOG		
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME		
ORDINAL_POSITION		
POSITION_IN_UNIQUE_CONSTRAINT		
REFERENCED_TABLE_SCHEMA		
REFERENCED_TABLE_NAME		
REFERENCED_COLUMN_NAME		

注释：

· 如果约束为外键，这就是外键列，而不是外键引用的列。

· ORDINAL_POSITION的值是列在约束中的位置，而不是列在表中的位置。列位置采用从1开始的数值编号。

· 对于"唯一"和"主键"约束，POSITION_IN_UNIQUE_CONSTRAINT的值为NULL。对于"外键"约束，它是所引用表内键中的顺序位置。

例如，假定有两个具有下述定义的表t1和t3：

```
CREATE TABLE t1
(
s1 INT,
s2 INT,
s3 INT,
PRIMARY KEY(s3)
) ENGINE=InnoDB;
```

```
CREATE TABLE t3
(
s1 INT,
s2 INT,
s3 INT,
KEY(s1),
CONSTRAINT CO FOREIGN KEY (s2) REFERENCES t1(s3)
) ENGINE=InnoDB;
```

对于这两个表，KEY_COLUMN_USAGE表有两行：

- 一行含有CONSTRAINT_NAME='PRIMARY', TABLE_NAME='t1', COLUMN_NAME='s3', ORDINAL_POSITION=1, POSITION_IN_UNIQUE_CONSTRAINT=NULL。
- 另一行含有CONSTRAINT_NAME='CO', TABLE_NAME='t3', COLUMN_NAME='s2', ORDINAL_POSITION=1, POSITION_IN_UNIQUE_CONSTRAINT=1。

23.1.14. INFORMATION_SCHEMA ROUTINES表

ROUTINES表提供了关于存储子程序（存储程序和函数）的信息。此时，ROUTINES表不包含自定义函数（UDF）。

名为“mysql.proc name”的列指明了对应于INFORMATION_SCHEMA.ROUTINES表的mysql.proc表列，如果有的话。

标准名称	**mysql.proc**名	注释
SPECIFIC_NAME	specific_name	
ROUTINE_CATALOG		NULL
ROUTINE_SCHEMA	db	
ROUTINE_NAME	name	
ROUTINE_TYPE	type	{PROCEDURE|FUNCTION}
DTD_IDENTIFIER		（数据类型描述符）
ROUTINE_BODY		SQL
ROUTINE_DEFINITION	body	
EXTERNAL_NAME		NULL
EXTERNAL_LANGUAGE	language	NULL
PARAMETER_STYLE		SQL
IS_DETERMINISTIC	is_deterministic	
SQL_DATA_ACCESS	sql_data_access	
SQL_PATH		NULL
SECURITY_TYPE	security_type	
CREATED	created	
LAST_ALTERED	modified	
SQL_MODE	sql_mode	MySQL扩展
ROUTINE_COMMENT	comment	MySQL扩展
DEFINER	definer	MySQL扩展

注释：

- MySQL计算EXTERNAL_LANGUAGE，因此：
  - 如果mysql.proc.language='SQL'，那么EXTERNAL_LANGUAGE为NULL。
  - 否则，EXTERNAL_LANGUAGE为mysql.proc.language中的值。然而，由于尚没有外部语言，因此该值总为NULL。

23.1.15. INFORMATION_SCHEMA VIEWS表

VIEWS表给出了关于数据库中的视图的信息。

标准名称	**SHOW**名称	注释
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
VIEW_DEFINITION		
CHECK_OPTION		
IS_UPDATABLE		
DEFINER		
SECURITY_TYPE		

注释：

- 有一种新的权限SHOW VIEW，如果没有它，将无法看到VIEWS表。
- VIEW_DEFINITION列含有你在SHOW CREATE VIEW所生成的Create Table字段中见到的大多数信息。跳过SELECT前的单词，并跳过具有CHECK OPTION（检查选项）的单词。例如，如果初始语句是：

```
CREATE VIEW v AS
  SELECT s2,s1 FROM t
  WHERE s1 > 5
  ORDER BY s1
  WITH CHECK OPTION;
```

那么视图定义为：

```
SELECT s2,s1 FROM t WHERE s1 > 5 ORDER BY s1
```

- CHECK_OPTION列的值总为NONE。
- 如果视图是可更新的，IS_UPDATABLE列的值为YES，如果视图是不可更新的，IS_UPDATABLE列的值为NO。
- DEFINER列指明了定义视图的人。SECURITY_TYPE的值为DEFINER或INVOKER。

23.1.16. INFORMATION_SCHEMA TRIGGERS表

TRIGGERS表提供了关于触发程序的信息。

必须有SUPER权限才能查看该表。

标准名称	**SHOW**名称	注释
TRIGGER_CATALOG		NULL
TRIGGER_SCHEMA		
TRIGGER_NAME	Trigger	
EVENT_MANIPULATION	Event	
EVENT_OBJECT_CATALOG		NULL
EVENT_OBJECT_SCHEMA		
EVENT_OBJECT_TABLE	Table	
ACTION_ORDER		0
ACTION_CONDITION		NULL
ACTION_STATEMENT	Statement	
ACTION_ORIENTATION		ROW
ACTION_TIMING	Timing	
ACTION_REFERENCE_OLD_TABLE		NULL
ACTION_REFERENCE_NEW_TABLE		NULL
ACTION_REFERENCE_OLD_ROW		OLD
ACTION_REFERENCE_NEW_ROW		NEW
CREATED		NULL (0)
SQL_MODE		

注释:

- TRIGGER_SCHEMA和TRIGGER_NAME列中分别含有相应数据库的名称以及触发程序的名称，在该数据库中，含有该触发程序。
- EVENT_MANIPULATION列含有下述值之一：INSERT、DELETE、或UPDATE。
- 正如第21章：触发程序中指出的那样，每个触发程序均与一个表准确相关。EVENT_OBJECT_SCHEMA和EVENT_OBJECT_TABLE列包含相应的数据库和表名，在该数据库中，含有该表。
- ACTION_ORDER语句含有触发程序动作（在相同表上所有类似触发程序列表中）的顺序位置。目前该值总为0，这是因为在相同表上具有相同EVENT_MANIPULATION和ACTION_TIMING的触发程序不能超过1个。
- ACTION_STATEMENT列含有激活了触发程序时将要执行的语句。这与SHOW TRIGGERS输出的Statement（语句）列中显示的文本相同。注意，该文本采用了UTF-8编码方式。
- ACTION_ORIENTATION列总含有值“ROW”。
- ACTION_TIMING列含有下述两种值之一：“BEFORE”或“AFTER”。
- 列ACTION_REFERENCE_OLD_ROW和ACTION_REFERENCE_NEW_ROW分别含有旧的和新的列标识符。这意味着ACTION_REFERENCE_OLD_ROW总含有值“OLD”，ACTION_REFERENCE_NEW_ROW总含有值“NEW”。
- SQL_MODE列显示了创建触发程序时有效的服务器SQL模式（无论当前的服务器SQL模式为何，只要激活了触发程序，它将保持有效）。该列的可能取值范围与sql_mode系统变量的取值范围相同。请参见5.3.2节，“SQL服务器模式”。
- 在下述列中，目前总含有NULL：TRIGGER_CATALOG，EVENT_OBJECT_CATALOG，ACTION_CONDITION，ACTION_REFERENCE_OLD_TABLE，ACTION_REFERENCE_NEW_TABLE和CREATED。

例如，使用21.3节，“使用触发程序”中定义的触发程序ins_sum。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.TRIGGERS\G
*************************** 1. row ***************************
 TRIGGER_CATALOG: NULL
TRIGGER_SCHEMA: test
 TRIGGER_NAME: ins_sum
  EVENT_MANIPULATION: INSERT
EVENT_OBJECT_CATALOG: NULL
 EVENT_OBJECT_SCHEMA: test
  EVENT_OBJECT_TABLE: account
 ACTION_ORDER: 0
ACTION_CONDITION: NULL
ACTION_STATEMENT:  SET @sum = @sum + NEW.amount
  ACTION_ORIENTATION: ROW
 ACTION_TIMING: BEFORE
ACTION_REFERENCE_OLD_TABLE: NULL
ACTION_REFERENCE_NEW_TABLE: NULL
  ACTION_REFERENCE_OLD_ROW: OLD
  ACTION_REFERENCE_NEW_ROW: NEW
CREATED: NULL
1 row in set (1.54 sec)
```

另请参见13.5.4.20节，“SHOW TRIGGERS语法”。

23.1.17. 其他INFORMATION_SCHEMA表

我们打算实施附加的INFORMATION_SCHEMA表。尤其是，我们确认了对INFORMATION_SCHEMA.PARAMETERS和INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS的需求。

23.2. SHOW语句的扩展

某些SHOW语句的扩展伴随着INFORMATION_SCHEMA的实施：

- SHOW可用于获取关于INFORMATION_SCHEMA本身结构的信息。
- 一些SHOW语句允许使用WHERE子句，这样，在指定需要显示的行时，可更为灵活。

INFORMATION_SCHEMA是一种信息数据库，因此，在SHOW DATABASES的输出中，包含其名称。与此类似，SHOW TABLES可与INFORMATION_SCHEMA一起使用，以获取表清单。

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA;
+---------------------------------------+
```

```
| Tables_in_information_schema|
+-------------------------------------+
| SCHEMATA|
| TABLES|
| COLUMNS |
| CHARACTER_SETS |
| COLLATIONS  |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| ROUTINES|
| STATISTICS  |
| VIEWS |
| TRIGGERS|
| USER_PRIVILEGES|
| SCHEMA_PRIVILEGES  |
| TABLE_PRIVILEGES|
| COLUMN_PRIVILEGES  |
| TABLE_CONSTRAINTS  |
| KEY_COLUMN_USAGE|
+-------------------------------------+
```

SHOW COLUMNS和DESCRIBE能够显示单独INFORMATION_SCHEMA表中的列信息。

扩展了一些SHOW语句，允许使用WHERE子句：

```
SHOW CHARACTER SET
SHOW COLLATION
SHOW COLUMNS
SHOW DATABASES
SHOW FUNCTION STATUS
SHOW KEYS
SHOW OPEN TABLES
SHOW PROCEDURE STATUS
SHOW STATUS
SHOW TABLE STATUS
SHOW TABLES
SHOW VARIABLES
```

如果有WHERE子句的话，将根据SHOW语句显示的列名进行计算。例如，SHOW COLLATION语句可产生这些输出列：

例如，SHOW CHARACTER SET语句可产生这些输出列：

```
mysql> SHOW CHARACTER SET;
```

Charset	描述	默认校对	最大长度
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	cp1252 West European	latin1_swedish_ci	1
latin2	ISO 8859-2 Central European	latin2_general_ci	1

要想与SHOW CHARACTER SET一起使用WHERE子句，应引用这些列名称。例如，在下面的语句中，给出了用于默认校对且含有字符串"japanese"的字符集的信息：

```
mysql> SHOW CHARACTER SET WHERE `Default collation` LIKE '%japanese%';
```

Charset	描述	默认校对	最大长度
ujis	EUC-JP Japanese	ujis_japanese_ci	3
sjis	Shift-JIS Japanese	sjis_japanese_ci	2
cp932	SJIS for Windows Japanese	cp932_japanese_ci	2
eucjpms	UJIS for Windows Japanese	eucjpms_japanese_ci	3

该语句显示了多字节字符集。

```
mysql> SHOW CHARACTER SET WHERE Maxlen > 1;
```

Charset	描述	默认校对	最大长度
big5	Big5 Traditional Chinese	big5_chinese_ci	2
ujis	EUC-JP Japanese	ujis_japanese_ci	3
sjis	Shift-JIS Japanese	sjis_japanese_ci	2
euckr	EUC-KR Korean	euckr_korean_ci	2
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci	2
gbk	GBK Simplified Chinese	gbk_chinese_ci	2
utf8	UTF-8 Unicode	utf8_general_ci	3
ucs2	UCS-2 Unicode	ucs2_general_ci	2
cp932	SJIS for Windows Japanese	cp932_japanese_ci	2
eucjpms	UJIS for Windows Japanese	eucjpms_japanese_ci	3

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第24章：精度数学

目录

24.1. 数值的类型
24.2. DECIMAL数据类型更改
24.3. 表达式处理
24.4. 四舍五入
24.5. 精度数学示例

MySQL 5.1提供了对精度数学的支持，也就是说，数值处理功能，它能给出极其精确的结果，并能对无效值进行高度控制。精度数学基于下述两种特性：

· SQL模式，控制服务器接受或拒绝无效值的严格程度（请参见5.3.2节，“SQL服务器模式”）。

· 用于定点算法的MySQL库。

对于数值操作，这些特性具有数种隐含意义：

· 精确计算：对于准确值数值，计算不会引入浮点错误。相反，将使用准确的精度。例如，对于数值.0001，会将其当作准确值予以处理，而不是近似值，将其加10000次可获得准确的结果1，而不是近似于1的值。

· 定义良好的四舍五入特性：对于准确值数值，ROUND()的结果取决于其参量，而不是环境因素，如底层C库的工作方式等。

· 平台无关性：对准确数值的操作在不同平台上（如Unix和Windows）是相同的。

· 对无效值处理的控制：能够检测到溢出和除0情况，并会将其当作错误加以处理。例如，能够将对于某列来说过大的值当作错误对待，而不是对该值进行截短使之位于列数据类型的范围内。同样，也会将除0当作错误，而不是会获得NULL结果的操作。至于选择那种方式，它是由系统变量sql_mode的设置决定的（请参见5.3.2节，“SQL服务器模式”）。

这类特性的一个重要结果是，MySQL 5.1提供了与标准SQL的高度兼容性。

在下面的讨论中，介绍了精度数学的数种工作方式（包括与早期应用程序的可能的不兼容性）。在最后，给出了一些示例，演示了MySQL 5.1是如何精确处理数值操作的。

24.1. 数值的类型

对于准确值操作，精度数学的范围包括准确值的数据类型（DECIMAL和整数类型）以及准确值数值文字。对于近似值数据类型和数值文字，仍会将其当作浮点数值予以处理。

准确值数值文字具有整数部分和小数部分，或两者。它们可以是有符号的。例如：1、.2、3.4、-5、-6.78、+9.10。

近似值数值文字采用科学计数法表示，包含尾数和指数。任意部分或两者均可以是带符号的。例如，1.2E3、1.2E-3、-1.2E3、-1.2E-3。

对于看上去类似的数值，不需要均为准确值或近似值。例如，2.34是准确值（定点）数值，而2.34E0是近似值（浮点）数值。

DECIMAL数据类型是定点类型，其计算是准确的。在MySQL中，DECIMAL类型有多个同义

词：NUMERIC、DEC、FIXED。整数类型也是准确值类型。

FLOAT和DOUBLE数据类型是浮点类型，其计算是近似的。在MySQL中，与FLOAT或DOUBLE同义的类型是DOUBLE PRECISION和REAL。

24.2. DECIMAL数据类型更改

本节讨论了MySQL 5.1中DECIMAL数据类型（以及其同义类型）的特性，尤其是下述方面：

- 数字的最大数。
- 存储格式。
- 存储要求。
- 对DECIMAL列上界 的非标准MySQL扩展。

在本节中，对于为较早MySQL版本编写的应用程序，在相应的地方指出了可能的不兼容性。

DECIMAL列的声明语法是DECIMAL(*M*,*D*)。在MySQL 5.1中，参量的取值范围如下：

- *M*是数字的最大数（精度）。其范围为1～65（在较旧的MySQL版本中，允许的范围是1～254）。
- *D*是小数点右侧数字的数目（标度）。其范围是0～30，但不得超过*M*。

对于M，最大值65意味着，对DECIMAL值的计算能精确到65位数字。这种65位数字的精度限制也适用于准确值数值文字，因此，这类文字值的最大范围不同于以前的范围（在较早的MySQL版本中，十进制值能达到254位。不过，采用的是浮点计算，因而是近似计算而不是准确计算）。

在MySQL 5.1中，采用二进制格式保存DECIMAL列的值，将9个十进制数字打包在4字节中。对于每个值的整数部分和小数部分，其存储要求是分别确定的。每9个数字需要4字节，任何剩余的数字将占用4字节的一部分。例如，DECIMAL(18,9)列在小数点的每一侧均有9位数字，因此，整数部分和小数部分均需要4字节。DECIMAL(20,10)列在小数点的每一侧均有10位数字。对于每一部分，9位数字需要4字节，剩余的1位数字需要1字节。

在下表中，给出了关于剩余数字的存储要求：

剩余的数字	字节数
0	0
1	1
2	1
3	2
4	2
5	3
6	3
7	4
8	4
9	4

与某些较早的MySQL版本不同，在MySQL 5.1中，DECIMAL列不保存前导"+"字符或前导"0"数字。如果将+0003.1插入DECIMAL(5,1)列，将保存为3.1。为了适应该变化，必须更改依赖于早期行为的应用程序。

在MySQL 5.1中，DECIMAL列不允许保存大于列定义中隐含范围的值。例如，DECIMAL(3,0)列支持的范围为-999～999。对于DECIMAL(*M*,*D*)列，小数点左侧最多允许*M* –*D*位数字（它与依赖于早期MySQL版本的应用程序不兼容，允许保存额外数字代替"+"号）。

SQL标准要求，NUMERIC(*M*,*D*)的精度必须准确为*M*位数字。对于DECIMAL(*M*,*D*)，标准要求的精度至少为*M*位数字，但允许更多。在MySQL中，DECIMAL(*M*,*D*)和NUMERIC(*M*,*D*)是相同的，两者的精度均准确为*M*位数字。

对于依赖DECIMAL数据类型早期处理方式的应用程序，关于移植这类应用程序的更多信息，请参见*MySQL 5.0*参考手册。

24.3. 表达式处理

对于精度数学，只要可能，就会使用给定的准确值数值。例如，在比较中所用的数值与给定的值准确相同，无任何变化。在严格的SQL模式下，对于插入具有准确数据类型（DECIMAL或整数）的列的INSERT操作，如果值在列的允许范围内，将插入具有准确值的数值。检索时，所获得的值与插入的值应是相同（如果未采用严格模式，允许INSERT执行截短操作）。

对数值表达式的处理取决于表达式包含的值的类型：

- 如果存在任何近似值，表达式也是近似的，并将使用浮点算法进行评估。

- 如果不存在近似值，表达式仅包含准确值。如果任一准确值包含小数部分（小数点后面的值），将使用DECIMAL准确算法来计算表达式，其精度为65位数字。术语"准确"受二进制表述方面的限制。例如，1.0/3.0在十进制表述中可近似为.333...，但并不是准确数值，因此(1.0/3.0)*3.0不会被计算为准确的1.0。

- 另外，表达式仅包含整数值。表达式是准确的，并将使用整数算法进行计算，其精度与BIGINT的相同（64比特）。

如果数值表达式包含任何字符串，会将其转换为双精度浮点值，表达式是近似的。

数值列中的插入操作受SQL模式的影响，它是由sql_mode系统变量控制的（请参见1.8.2节，"选择SQL模式"）。下面介绍了严格模式（由STRICT_ALL_TABLES或STRICT_TRANS_TABLES模式值选择）和RROR_FOR_DIVISION_BY_ZERO。要想打开所有限制，可简单地使用TRADITIONAL模式，它包含严格模式和ERROR_FOR_DIVISION_BY_ZERO：

```
mysql> SET SQL_MODE='TRADITIONAL';
```

如果将数值插入具有准确类型的列（DECIMAL或整数），如果值在列允许的范围内，将以准确值形式插入数值。

如果数值在其小数部分有过多位，将执行四舍五入操作并给出告警。关于四舍五入的具体介绍，请参见四舍五入一节。

如果数值在其整数部分有过多位，数值过大，并将按下述方式处理：

- 如果未启用严格模式，该数值将被截短为最近的合法值，并发出警告。

- 如果启用了严格模式，将给出溢出错误。

不检测下溢，因而下溢处理是不确定的。

默认情况下，除0操作会导致NULL结果，不产生告警。启用了ERROR_FOR_DIVISION_BY_ZERO SQL模式后，MySQL会以不同方式处理除0问题：

- 如果未启用严格模式，发出警告。
- 如果启用了严格模式，将禁止包含除0操作的插入和更新，并给出错误。

换句话讲，对于包含执行除0操作的表达式的插入和更新，将被当作错误对待，但除了严格模式外还需要ERROR_FOR_DIVISION_BY_ZERO。

假定下述语句：

```
INSERT INTO t SET i = 1/0;
```

对于严格模式和ERROR_FOR_DIVISION_BY_ZERO模式的组合，情况如下：

sql_mode值	结果
'' (Default)	无告警，无错误：i被设置为NULL。
strict	无告警，无错误：i被设置为NULL。
ERROR_FOR_DIVISION_BY_ZERO	告警，无错误：i被设置为NULL。
strict,ERROR_FOR_DIVISION_BY_ZERO	错误条件，不插入任何行。

将字符串插入数值列时，如果字符串具有非数值内容，将按下述方式将字符串转换为数值：

- 对于未以数值开始的字符串，在严格模式下，不能将其作为数值使用，并会产生错误，在其他情况下，给出警告。包括空字符串。
- 对于以数值开始的字符串，可以进行转换，但尾随的非数值部分将被截去。在严格模式下会导致错误，在其他情况下，给出警告。

24.4. 四舍五入

本节讨论了精度数学的四舍五入特性，ROUND()函数，以及插入DECIMAL列时的四舍五入特性。

ROUND()函数的行为取决于其参量是准确的还是近似的：

- 对于准确值数值，ROUND()采用"半值向上舍入"规则：如果小数部分的值为.5或更大，如果是正数，向上取下一个整数，如果是负数，向下取下一个整数（换句话讲，以0为界限执行舍入）。如果小数部分的值小于.5，如果是正数，向下取下一个整数，如果是负数，向上取下一个整数。
- 对于近似值数值，结果取决于C库函数。在很多系统上，它意味着ROUND()将使用"舍入至最近的偶数"规则：具有任何小数部分的值均将被舍入为最近的偶数。

在下面的示例中，介绍了舍入操作对准确值和近似值的不同处理方式：

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+------------+--------------+
| ROUND(2.5) | ROUND(25E-1) |
+------------+--------------+
| 3          |            2 |
+------------+--------------+
```

对于向DECIMAL列的插入操作，目标是准确的数据类型，无论要插入的值是准确的还是近似的，将采用"半值向上舍入"规则：

```
mysql> CREATE TABLE t (d DECIMAL(10,0));
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t VALUES(2.5),(2.5E0);
Query OK, 2 rows affected, 2 warnings (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 2

mysql> SELECT d FROM t;
+------+
| d    |
+------+
| 3    |
| 3    |
+------+
```

24.5. 精度数学示例

本节给出了一些示例，介绍了MySQL 5.1中的精度数学查询结果。

示例**1**。可能时，将使用给定的准确值：

```
mysql> SELECT .1 + .2 = .3;

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

| .1 + .2 = .3 |

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

|            1 |

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

但是，对于浮点值，结果是不准确的：

```
mysql> SELECT .1E0 + .2E0 = .3E0;

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

| .1E0 + .2E0 = .3E0 |

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

|                  0 |

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

查看准确值和近似值处理差异的另一个方法是，增加1个小的数值，并多次累加。请考虑下述存储程序，它将.0001加到变量上1000次。

```
CREATE PROCEDURE p ()

BEGIN

  DECLARE i INT DEFAULT 0;

  DECLARE d DECIMAL(10,4) DEFAULT 0;

  DECLARE f FLOAT DEFAULT 0;

  WHILE i < 10000 DO

    SET d = d + .0001;

    SET f = f + .0001E0;

    SET i = i + 1;

  END WHILE;

  SELECT d, f;
```

```
END;
```

从逻辑上讲，d和f的合计应为1，但仅对decimal计算来说是这样。浮点计算会引入小的误差：

```
+--------+-----------------+
| d      | f               |
+--------+-----------------+
| 1.0000 | 0.99999999999991 |
+--------+-----------------+
```

示例**2**。乘法是按照标准SQL所要求的标度执行。也就是说，对于具有标度S1和S2的两个数值*X1*和*X2*，结果的标度为*S1 + S2*：

```
mysql> SELECT .01 * .01;
+-----------+
| .01 * .01 |
+-----------+
| 0.0001    |
+-----------+
```

示例**3**：四舍五入定义良好：

在MySQL 5.1中，四舍五入操作（例如，使用ROUND()函数）独立于底层C库函数的实施，这意味着，在不同平台上结果是一致的。

在MySQL 5.1中，对于DECIMAL列和准确值数值，采用了"半值向上舍入"规则。对于小数部分等于或大于0.5的值，以0为分界舍入至最近的整数，如下所示：

```
mysql> SELECT ROUND(2.5), ROUND(-2.5);
+------------+-------------+
| ROUND(2.5) | ROUND(-2.5) |
+------------+-------------+
| 3          | -3          |
+------------+-------------+
```

但是，对于浮点值的舍入采用C库，在很多系统上，使用"舍入至最近的偶数"规则。在这类系统上，具有任何小数部分的值均将被舍入为最近的偶数：

```
mysql> SELECT ROUND(2.5E0), ROUND(-2.5E0);
+--------------+---------------+
| ROUND(2.5E0) | ROUND(-2.5E0) |
+--------------+---------------+
|            2 |            -2 |
+--------------+---------------+
```

示例**4**。在严格模式下，插入过大的值会导致溢出和错误，而不是截短至合法值。

当MySQL未运行在严格模式下时，将截短至合法值：

```
mysql> SET SQL_MODE='';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 128;
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> SELECT i FROM t;
+------+
| i    |
+------+
|  127 |
+------+
1 row in set (0.00 sec)
```

但是，如果严格模式起作用，将出现溢出状况：

```
mysql> SET SQL_MODE='TRADITIONAL';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.01 sec)

mysql> SET sql_mode='STRICT_ALL_TABLES';
Query OK, 0 rows affected (0.10 sec)

mysql> INSERT INTO t SET i = 128;
ERROR 1264 (22003): Out of range value adjusted for column 'i' at row 1

mysql> SELECT i FROM t;
Empty set (0.00 sec)
```

示例**5**。在严格模式下并具有ERROR_FOR_DIVISION_BY_ZERO设置时，除0会导致错误，而不是产生NULL结果。

在非严格模式下，除0将得出NULL结果：

```
mysql> SET SQL_MODE='';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET i = 1 / 0;
Query OK, 1 row affected (0.06 sec)

mysql> SELECT i FROM t;
+------+
| i    |
+------+
| NULL |
+------+
1 row in set (0.01 sec)
```

但是，如果恰当的SQL模式处于有效状态，除0将导致错误：

```
mysql> SET SQL_MODE='TRADITIONAL';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> SET sql_mode='STRICT_ALL_TABLES,ERROR_FOR_DIVISION_BY_ZERO';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 1 / 0;
ERROR 1365 (22012): Division by 0

mysql> SELECT i FROM t;
Empty set (0.01 sec)
```

示例**6**。在MySQL 4中（引入精度数学之前），准确值和近似值文字均会被转换为双精度浮点值：

```
mysql> SELECT VERSION();
+-----------------+
| VERSION()       |
+-----------------+
| 4.0.25-standard |
+-----------------+
1 row in set (0.00 sec)
```

```
mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;

mysql> DESCRIBE t;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | double(3,1) |      |     | 0.0     |       |
| b     | double      |      |     | 0       |       |
+-------+-------------+------+-----+---------+-------+
```

在MySQL 5.1中，近似值文字仍会被转换为浮点值，但准确值文字将被当作DECIMAL处理：

```
mysql> SELECT VERSION();
+-----------------+
| VERSION()       |
+-----------------+
| 5.1.2-alpha-log |
+-----------------+
1 row in set (0.00 sec)

mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;

mysql> DESCRIBE t;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| a     | decimal(2,1) | NO   |     | 0.0     |       |
| b     | double       | NO   |     | 0       |       |
+-------+--------------+------+-----+---------+-------+
```

示例**7**。如果聚合函数的参量是准确的数值类型，其结果也是准确的数值类型，标度至少为参量的标度。

考虑下述语句：

```
mysql> CREATE TABLE t (i INT, d DECIMAL, f FLOAT);
mysql> INSERT INTO t VALUES(1,1,1);
mysql> CREATE TABLE y SELECT AVG(i), AVG(d), AVG(f) FROM t;
```

在MySQL 4.0或4.1（在MySQL中引入精度数学之前）中的结果：

```
mysql> DESCRIBE y;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
```

```
+--------+--------------+------+-----+---------+-------+
| AVG(i) | double(17,4) | YES  |     | NULL    |       |
| AVG(d) | double(17,4) | YES  |     | NULL    |       |
| AVG(f) | double       | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
```

无论参量类型是什么，结果为double。

在MySQL 5.1中的结果：

```
mysql> DESCRIBE y;
+--------+---------------+------+-----+---------+-------+
| Field  | Type          | Null | Key | Default | Extra |
+--------+---------------+------+-----+---------+-------+
| AVG(i) | decimal(14,4) | YES  |     | NULL    |       |
| AVG(d) | decimal(14,4) | YES  |     | NULL    |       |
| AVG(f) | double        | YES  |     | NULL    |       |
+--------+---------------+------+-----+---------+-------+
```

仅对浮点参量，其结果为double。对于准确类型参量，结果也为准确类型。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第25章：API和库

目录

25.1. libmysqld，嵌入式MySQL服务器库

25.1.1. 嵌入式MySQL服务器库概述
25.1.2. 使用libmysqld编译程序
25.1.3. 使用嵌入式MySQL服务器时的限制
25.1.4. 与嵌入式服务器一起使用的选项
25.1.5. 嵌入式服务器中尚需完成的事项(TODO)
25.1.6. 嵌入式服务器示例
25.1.7. 嵌入式服务器的许可

25.2. MySQL C API

25.2.1. C API数据类型
25.2.2. C API函数概述
25.2.3. C API函数描述
25.2.4. C API预处理语句
25.2.5. C API预处理语句的数据类型
25.2.6. C API预处理语句函数概述
25.2.7. C API预处理语句函数描述
25.2.8. C API预处理语句方面的问题
25.2.9. 多查询执行的C API处理
25.2.10. 日期和时间值的C API处理
25.2.11. C API线程函数介绍
25.2.12. C API嵌入式服务器函数介绍
25.2.13. 使用C API时的常见问题
25.2.14. 创建客户端程序
25.2.15. 如何生成线程式客户端

25.3. MySQL PHP API

25.3.1. 使用MySQL和PHP的常见问题

25.4. MySQL Perl API
25.5. MySQL C++ API

25.5.1. Borland C++

25.6. MySQL Python API
25.7. MySQL Tcl API
25.8. MySQL Eiffel Wrapper
25.9. MySQL程序开发实用工具

25.9.1. msql2mysql：转换mSQL程序以用于MySQL
25.9.2. mysql_config：获取编译客户端的编译选项

本章介绍了MySQL可使用的API，从哪里获得它们，以及如何使用它们。 详细介绍C API，这是因为它是由MySQL团队开发的，而且它也是大多数其他API的基础。 本章还介绍了libmysqld库（嵌入式服务器），以及对应用程序开发人员有用的一些程序。

25.1. libmysqld，嵌入式MySQL服务器库

25.1.1. 嵌入式MySQL服务器库概述
25.1.2. 使用libmysqld编译程序
25.1.3. 使用嵌入式MySQL服务器时的限制
25.1.4. 与嵌入式服务器一起使用的选项
25.1.5. 嵌入式服务器中尚需完成的事项(TODO)
25.1.6. 嵌入式服务器示例
25.1.7. 嵌入式服务器的许可

25.1.1. 嵌入式MySQL服务器库概述

使用嵌入式MySQL服务器库，能够在客户端应用程序中使用具备全部特性的MySQL服务器。 主要优点在于，增加了速度，并使得嵌入式应用程序的管理更简单。

嵌入式服务器库是以MySQL的客户端／服务器版本为基础的，采用C/C++语言编写。 其结果是嵌入式服务器也是用C/C++语言编写的。 在其他语言中，嵌入式服务器不可用。

API与嵌入式MySQL版本和客户端／服务器版本等效。 要想更改旧的线程式应用程序以使用嵌入式库，正常情况下，仅需添加对下述函数的调用即可。

函数	何时调用
mysql_server_init()	应在调用任何其他MySQL函数之前调用，最好是在main()函数中调用。
mysql_server_end()	应在程序退出前调用。
mysql_thread_init()	应在你所创建的、用于访问MySQL的每个线程中调用。
mysql_thread_end()	应在调用pthread_exit()之前调用。

随后，必须将你的代码与libmysqld.a链接起来，而不是libmysqlclient.a。

在libmysqlclient.a中还包含mysql_server_*xxx*()函数，使用这类函数，通过将应用程序链接到恰当的库，即可在嵌入式版本和客户端／服务器版本之间切换。 请参见25.2.12.1节，“mysql_server_init()”。

嵌入式服务器和独立服务器之间的一项差别在于，对于嵌入式服务器，默认情况下，连接鉴定是禁止的。 对于嵌入式服务器，要想使用鉴定功能，可在激活“**configure**”以配置MySQL分发版时使用“--with-embedded-privilege-control”选项。

25.1.2. 使用libmysqld编译程序

要想获得libmysqld库，应使用“--with-embedded-server”选项配置MySQL。 请参见2.8.2节，“典型配置选项”。

将你的程序与libmysqld链接时，还必须包含系统的pthread库以及MySQL服务器使用的一些库。 执行“**mysql_config --libmysqld-libs**”，可获得库的完整列表。

对于线程程序的编译和链接，必须使用正确的标志，即使你未在代码中直接调用任何线程函数也同样。

要想编译C程序以包含必要文件，并将MySQL服务器库嵌入到程序的编译版本中，可使用GNU C编译器（gcc）。 编译器需要知道各种文件的位置，并需了解如何编译程序的指令。 在下面的示例中，介绍了如何从命令行编译程序的方法：

```
gcc mysql_test.c -o mysql_test -lz \
`/usr/local/mysql/bin/mysql_config --include --libmysqld-libs`
```

在gcc命令后紧跟着未编译C程序文件的名称。 接下来，给定的“-o”选项指明，它后面的文件名是编译器将输出文件的名称，即编译后的程序。 在下一行的代码中，通知编译器获取包含文件和库的位置，以及在其上进行编译的系统的其他设置。 由于“**mysql_config**”存在的问题，在此添加了“-lz”选项（压缩）。 “**mysql_config**”部分包含在backticks中，而不是单引号内。

25.1.3. 使用嵌入式MySQL服务器时的限制

嵌入式服务器存在下述限制：

- 不支持ISAM表。 （这样做的主要目的是为了使库更小）。
- 没有自定义函数（UDF）。
- 没有对核心转储的堆栈跟踪。
- 没有内部RAID支持。 （由于大多数当前操作系统均支持大文件，通常情况下不需要它）。
- 不能将其设置为“主”或“从”（无复制）。
- 在内存较低的系统上，可能无法使用很大的结果集。

- 不能使用套接字或TCP/IP从外部进程连接到嵌入式服务器。 但是，你可以连接到中间应用程序，随后，该中间应用程序可代表远程客户端或外部进程连接到嵌入式服务器。

通过编辑"mysql_embed.h"包含文件并重新编译MySQL，可更改某些限制。

25.1.4. 与嵌入式服务器一起使用的选项

对于任何能够与**mysqld**服务器端口监督程序一起给定的选项，也可以与嵌入式服务器库一起使用。在数组中，可将服务器选项作为参量指定给用于初始化服务器的mysql_server_init()。也能在诸如my.cnf的选项文件中给定它们。要想为C程序指定选项文件，请使用"--defaults-file"选项作为函数mysql_server_init()的第2个参量的元素之一。关于mysql_server_init()函数的更多信息，请参见25.2.12.1节，"mysql_server_init()"。

使用选项文件，能够简化客户端／服务器应用程序和嵌入了MySQL的应用程序之间的切换。将常用选项置于[server]组。它们可被两种MySQL版本读取。客户端／服务器选项应被放在[mysqld]部分。将嵌入式MySQL服务器库的选项放在[embedded]部分。将与应用程序相关的选项放在标记为[ApplicationName_SERVER]的部分。请参见4.3.2节，"使用选项文件"。

25.1.5. 嵌入式服务器中尚需完成的事项(TODO)

- 我们将提供一些选项以省去MySQL的某些部分，从而使库变得更小。
- 仍有很多速度优化工作需要完成。
- 错误将被写入stderr。我们将增加1个选项为它们指定文件名。
- 使用嵌入式版本时，需要更改InnoDB，使之不再冗长。如果你的数据库不含InnoDB表，要想抑制相关消息，可为组[libmysqd_server]下的选项文件增加"--skip-innodb"选项，或在用**mysql_server_init()**初始化服务器时添加该选项。

25.1.6. 嵌入式服务器示例

在Linux或FreeBSD系统上，无需更改就能使用下面这两个示例程序。对于其他操作系统，需要进行小的修改，主要是文件路径。设计这两个示例的目的在于，为你提供足够的细节信息，以便理解问题，它们是实际应用程序的必要组成部份。第1个示例十分直观。第2个示例采用了一些错误检查功能，略为复杂。在第1个示例的后面，给出了用于编译程序的命令行条目。在第2个示例的后面，给出了GNUmake文件，该文件可用于编译。

示例：**1**

test1_libmysqld.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include "mysql.h"

MYSQL *mysql;
MYSQL_RES *results;
MYSQL_ROW record;

static char *server_options[] = { "mysql_test", "--defaults-file=my.cnf" };
int num_elements = sizeof(server_options)/ sizeof(char *);

static char *server_groups[] = { "libmysqld_server", "libmysqld_client" };

int main(void)
{
```

```
   mysql_server_init(num_elements, server_options, server_groups);
   mysql = mysql_init(NULL);
   mysql_options(mysql, MYSQL_READ_DEFAULT_GROUP, "libmysqld_client");
   mysql_options(mysql, MYSQL_OPT_USE_EMBEDDED_CONNECTION, NULL);

   mysql_real_connect(mysql, NULL,NULL,NULL, "database1", 0,NULL,0);

   mysql_query(mysql, "SELECT column1, column2 FROM table1");

   results = mysql_store_result(mysql);

   while((record = mysql_fetch_row(results))) {
      printf("%s - %s \n", record[0], record[1]);
   }

   mysql_free_result(results);
   mysql_close(mysql);
   mysql_server_end();

   return 0;
}
```

下面给出了编译上述程序的命令行命令:

```
gcc test1_libmysqld.c -o test1_libmysqld -lz \
 `/usr/local/mysql/bin/mysql_config --include --libmysqld-libs`
```

示例: **2**

要想检验该示例，创建一个与MySQL源目录同级的test2_libmysqld目录。将test2_libmysqld.c源文件和GNUmakefile保存到该目录，并在test2_libmysqld目录下运行GNU make。

test2_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[] = {
  "test2_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
   * initializes the server using groups = {
   *   "server", "embedded", NULL
   *  }.
   *
   * In your $HOME/.my.cnf file, you probably want to put:

[test2_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 does not accept the
   * client's options, and vice versa.
   */
  mysql_options(db, MYSQL_READ_DEFAULT_GROUP, "test2_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-5.1/include
#lib      := $(HOME)/mysql-5.1/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
```

25.1.7. 嵌入式服务器的许可

我们鼓励所有人在GPL或兼容许可的旗帜下通过发布代码来推广免费软件。对于有能力完成该类事项的人员，也可以选择从MySQL AB购买MySQL的商用许可。详情请参见http://www.mysql.com/company/legal/licensing/。

25.2. MySQL C API

25.2.1. C API数据类型
25.2.2. C API函数概述
25.2.3. C API函数描述
25.2.4. C API预处理语句
25.2.5. C API预处理语句的数据类型
25.2.6. C API预处理语句函数概述
25.2.7. C API预处理语句函数描述
25.2.8. C API预处理语句方面的问题
25.2.9. 多查询执行的C API处理
25.2.10. 日期和时间值的C API处理
25.2.11. C API线程函数介绍
25.2.12. C API嵌入式服务器函数介绍
25.2.13. 使用C API时的常见问题
25.2.14. 创建客户端程序
25.2.15. 如何生成线程式客户端

C API代码是与MySQL一起提供的。它包含在mysqlclient库中，并允许C程序访问数据库。

MySQL源码分发版的很多客户端是用C语言编写的。如果你正在寻找能演示如何使用C API的示例，可参看这些客户端程序。你可以在MySQL源码分发版的客户端目录下找到它们。

大多数其他客户端API（除了Connector/J和Connector/NET）采用mysqlclient库来与MySQL服务器进行通信。这意味着（例如），你可以利用很多相同环境变量（与其他客户端程序使用的环境变量相同）带来的好处，这是因为它们是从库中引用的。关于这些变量的详细清单，请参见第8章：客户端和实用工具程序。

客户端具有最大的通信缓冲区大小。初始分配的缓冲区大小（16KB）将自动增加到最大（最大为16MB）。由于缓冲区大小将按需增加，简单地增加默认的最大限制，从其本身来说不会增加资源使用。该大小检查主要是检查错误查询和通信信息包。

通信缓冲区必须足够大，足以包含1条SQL语句（用于客户端-服务器通信）以及1行返回的数据（用于服务器-客户端通信）。每

个线程的通信缓冲区将动态增加，以处理直至最大限制的任何查询或行。例如，如果BLOB值包含高达16MB的数据，那么通信缓冲区的大小限制至少为16MB（在服务器和客户端）。客户端的默认最大值为16MB，但服务器的默认最大值为1MB。也可以在启动服务器时，通过更改max_allowed_packet参数的值增加它。请参见7.5.2节，“调节服务器参数”。

每次查询后，MySQL服务器会将通信缓冲区的大小降至net_buffer_length字节。对于客户端，不会降低与连接相关缓冲区大小，直至连接关闭为止，此时，客户端内存将被收回。

关于使用线程的编程方法，请参见25.2.15节，“如何生成线程式客户端”。关于在相同程序创建包含“服务器”和“客户端”的独立应用程序的更多信息（不与外部MySQL服务器通信），请参见25.1节，“libmysqld，嵌入式MySQL服务器库”。

25.2.1. C API数据类型

- MYSQL

  该结构代表1个数据库连接的句柄。几乎所有的MySQL函数均使用它。不应尝试拷贝MYSQL结构。不保证这类拷贝结果会有用。

- MYSQL_RES

  该结构代表返回行的查询结果（SELECT, SHOW, DESCRIBE, EXPLAIN）。在本节的剩余部分，将查询返回的信息称为“结果集”。

- MYSQL_ROW

  这是1行数据的“类型安全”表示。它目前是按照计数字节字符串的数组实施的。（如果字段值可能包含二进制数据，不能将其当作由Null终结的字符串对待，这是因为这类值可能会包含Null字节）。行是通过调用mysql_fetch_row()获得的。

- MYSQL_FIELD

  该结构包含关于字段的信息，如字段名、类型和大小。这里详细介绍了其成员。通过重复调用mysql_fetch_field()，可为每个字段获得MYSQL_FIELD结构。字段值不是该结构的组成部份，它们包含在MYSQL_ROW结构中。

- MYSQL_FIELD_OFFSET

  这是MySQL字段列表偏移量的“类型安全”表示（由mysql_field_seek()使用）。偏移量是行内的字段编号，从0开始。

- my_ulonglong

  用于行数以及mysql_affected_rows()、mysql_num_rows()和mysql_insert_id()的类型。该类型提供的范围为0～1.84e19。

  在某些系统上，不能打印类型my_ulonglong的值。要想打印这类值，请将其转换为无符号长整数类型并使用%lu打印格式，例如：

  ```
  printf ("Number of rows: %lu\n", (unsigned long) mysql_num_rows(result));
  ```

下面列出了MYSQL_FIELD结构包含的成员：

· char * name

字段名称，由Null终结的字符串。如果用AS子句为该字段指定了别名，名称的值也是别名。

· char * org_name

段名称，由Null终结的字符串。忽略别名。

· char * table

包含该字段的表的名称，如果该字段不是计算出的字段的话。对于计算出的字段，表值为空的字符串。如果用AS子句为该表指定了别名，表的值也是别名。

· char * org_table

表的名称，由Null终结的字符串。忽略别名。

- char * db

字段源自的数据的名称，由Null终结的字符串。如果该字段是计算出的字段，db为空的字符串。

- char * catalog

catalog名称。该值总是"def"。

- char * def

该字段的默认值，由Null终结的字符串。仅当使用mysql_list_fields()时才设置它。

- unsigned long length

字段的宽度，如表定义中所指定的那样。

- unsigned long max_length

用于结果集的字段的最大宽度（对于实际位于结果集中的行，最长字段值的长度）。如果使用mysql_store_result()或mysql_list_fields()，它将包含字段的最大长度。如果使用mysql_use_result()，该变量的值为0。

- unsigned int name_length

名称的长度。

- unsigned int org_name_length

org_name的长度。

- unsigned int table_length

表的长度。

- unsigned int org_table_length

org_table的长度。

- unsigned int db_length

db的长度。

- unsigned int catalog_length

catalog的长度。

- unsigned int def_length

def的长度。

- unsigned int flags

用于字段的不同"位标志"。标志的值可以有0个或多个下述位集合:

标志值	标志描述
NOT_NULL_FLAG	字段不能为NULL
PRI_KEY_FLAG	字段是主键的组成部分
UNIQUE_KEY_FLAG	字段是唯一键的组成部分
MULTIPLE_KEY_FLAG	字段是非唯一键的组成部分
UNSIGNED_FLAG	字段具有UNSIGNED属性
ZEROFILL_FLAG	字段具有ZEROFILL属性
BINARY_FLAG	字段具有BINARY属性
AUTO_INCREMENT_FLAG	字段具有AUTO_INCREMENT属性
ENUM_FLAG	字段是ENUM（不再重视）

SET_FLAG	字段是 SET（不再重视）
BLOB_FLAG	字段是BLOB或TEXT（不再重视）
TIMESTAMP_FLAG	字段是TIMESTAMP（不再重视）

不再重视BLOB_FLAG、ENUM_FLAG、SET_FLAG和TIMESTAMP_FLAG标志，原因在于，它们指出了字段的类型，而不是类型的属性。更可取的方式是使用MYSQL_TYPE_BLOB、MYSQL_TYPE_ENUM、MYSQL_TYPE_SET或MYSQL_TYPE_TIMESTAMP测试field->type。

在下面的示例中，介绍了标志值的典型用法：

```
if (field->flags & NOT_NULL_FLAG)
    printf("Field can't be null\n");
```

可以使用下述方面的宏来定义标志值的布尔状态：

标志状态	描述
IS_NOT_NULL(flags)	如果该字段定义为NOT NULL，为"真"。
IS_PRI_KEY(flags)	如果该字段是主键，为"真"。
IS_BLOB(flags)	如果该字段是BLOB或TEXT，为"真"（不再重视，用测试field->type取而代之）。

- unsigned int decimals

用于数值字段的十进制数数目。

- unsigned int charset_nr

用于字段的字符集编号。

- enum enum_field_types type

字段的类型。类型值可以是下标所列的MYSQL_TYPE_符号之一：

类型值	类型描述
MYSQL_TYPE_TINY	TINYINT字段
MYSQL_TYPE_SHORT	SMALLINT字段
MYSQL_TYPE_LONG	INTEGER字段
MYSQL_TYPE_INT24	MEDIUMINT字段
MYSQL_TYPE_LONGLONG	BIGINT字段
MYSQL_TYPE_DECIMAL	DECIMAL或NUMERIC字段
MYSQL_TYPE_NEWDECIMAL	精度数学DECIMAL或NUMERIC
MYSQL_TYPE_FLOAT	FLOAT字段
MYSQL_TYPE_DOUBLE	DOUBLE或REAL字段
MYSQL_TYPE_BIT	BIT字段
MYSQL_TYPE_TIMESTAMP	TIMESTAMP字段
MYSQL_TYPE_DATE	DATE字段
MYSQL_TYPE_TIME	TIME字段
MYSQL_TYPE_DATETIME	DATETIME字段
MYSQL_TYPE_YEAR	YEAR字段
MYSQL_TYPE_STRING	CHAR字段
MYSQL_TYPE_VAR_STRING	VARCHAR字段
MYSQL_TYPE_BLOB	BLOB或TEXT字段（使用max_length来确定最大长度）
MYSQL_TYPE_SET	SET字段
MYSQL_TYPE_ENUM	ENUM字段
MYSQL_TYPE_GEOMETRY	Spatial字段
MYSQL_TYPE_NULL	NULL-type字段
MYSQL_TYPE_CHAR	不再重视，用MYSQL_TYPE_TINY取代

可以使用IS_NUM()宏来测试字段是否具有数值类型。将类型值传递给IS_NUM()，如果字段为数值类型，会将其评估为"真"：

```
if (IS_NUM(field->type))
    printf("Field is numeric\n");
```

25.2.2. C API函数概述

这里归纳了C API可使用的函数，并在下一节详细介绍了它们。请参见25.2.3节，"C API函数描述"。

函数	描述
mysql_affected_rows()	返回上次UPDATE、DELETE或INSERT查询更改／删除／插入的行数。
mysql_autocommit()	切换 autocommit模式，ON/OFF
mysql_change_user()	更改打开连接上的用户和数据库。
mysql_charset_name()	返回用于连接的默认字符集的名称。
mysql_close()	关闭服务器连接。
mysql_commit()	提交事务。
mysql_connect()	连接到MySQL服务器。该函数已不再被重视，使用mysql_real_connect()取代。
mysql_create_db()	创建数据库。该函数已不再被重视，使用SQL语句CREATE DATABASE取而代之。
mysql_data_seek()	在查询结果集中查找属性行编号。
mysql_debug()	用给定的字符串执行DBUG_PUSH。
mysql_drop_db()	撤销数据库。该函数已不再被重视，使用SQL语句DROP DATABASE取而代之。
mysql_dump_debug_info()	让服务器将调试信息写入日志。
mysql_eof()	确定是否读取了结果集的最后一行。该函数已不再被重视，可以使用mysql_errno()或mysql_error()取而代之。
mysql_errno()	返回上次调用的MySQL函数的错误编号。
mysql_error()	返回上次调用的MySQL函数的错误消息。
mysql_escape_string()	为了用在SQL语句中，对特殊字符进行转义处理。
mysql_fetch_field()	返回下一个表字段的类型。
mysql_fetch_field_direct()	给定字段编号，返回表字段的类型。
mysql_fetch_fields()	返回所有字段结构的数组。
mysql_fetch_lengths()	返回当前行中所有列的长度。
mysql_fetch_row()	从结果集中获取下一行
mysql_field_seek()	将列光标置于指定的列。
mysql_field_count()	返回上次执行语句的结果列的数目。
mysql_field_tell()	返回上次mysql_fetch_field()所使用字段光标的位置。
mysql_free_result()	释放结果集使用的内存。
mysql_get_client_info()	以字符串形式返回客户端版本信息。
mysql_get_client_version()	以整数形式返回客户端版本信息。
mysql_get_host_info()	返回描述连接的字符串。
mysql_get_server_version()	以整数形式返回服务器的版本号。
mysql_get_proto_info()	返回连接所使用的协议版本。
mysql_get_server_info()	返回服务器的版本号。
mysql_info()	返回关于最近所执行查询的信息。
mysql_init()	获取或初始化MYSQL结构。
mysql_insert_id()	返回上一个查询为AUTO_INCREMENT列生成的ID。
mysql_kill()	杀死给定的线程。
mysql_library_end()	最终确定MySQL C API库。
mysql_library_init()	初始化MySQL C API库。
mysql_list_dbs()	返回与简单正则表达式匹配的数据库名称。
mysql_list_fields()	返回与简单正则表达式匹配的字段名称。
mysql_list_processes()	返回当前服务器线程的列表。

mysql_list_tables()	返回与简单正则表达式匹配的表名。
mysql_more_results()	检查是否还存在其他结果。
mysql_next_result()	在多语句执行过程中返回/初始化下一个结果。
mysql_num_fields()	返回结果集中的列数。
mysql_num_rows()	返回结果集中的行数。
mysql_options()	为mysql_connect()设置连接选项。
mysql_ping()	检查与服务器的连接是否工作，如有必要重新连接。
mysql_query()	执行指定为"以Null终结的字符串"的SQL查询。
mysql_real_connect()	连接到MySQL服务器。
mysql_real_escape_string()	考虑到连接的当前字符集，为了在SQL语句中使用，对字符串中的特殊字符进行转义处理。
mysql_real_query()	执行指定为计数字符串的SQL查询。
mysql_refresh()	刷新或复位表和高速缓冲。
mysql_reload()	通知服务器再次加载授权表。
mysql_rollback()	回滚事务。
mysql_row_seek()	使用从mysql_row_tell()返回的值，查找结果集中的行偏移。
mysql_row_tell()	返回行光标位置。
mysql_select_db()	选择数据库。
mysql_server_end()	最终确定嵌入式服务器库。
mysql_server_init()	初始化嵌入式服务器库。
mysql_set_server_option()	为连接设置选项（如多语句）。
mysql_sqlstate()	返回关于上一个错误的SQLSTATE错误代码。
mysql_shutdown()	关闭数据库服务器。
mysql_stat()	以字符串形式返回服务器状态。
mysql_store_result()	检索完整的结果集至客户端。
mysql_thread_id()	返回当前线程ID。
mysql_thread_safe()	如果客户端已编译为线程安全的，返回1。
mysql_use_result()	初始化逐行的结果集检索。
mysql_warning_count()	返回上一个SQL语句的告警数。

与MySQL交互时，应用程序应使用该一般性原则：

1. 通过调用mysql_library_init()，初始化MySQL库。库可以是mysqlclient C客户端库，或mysqld嵌入式服务器库，具体情况取决于应用程序是否与"-libmysqlclient"或"-libmysqld"标志链接。

2. 通过调用mysql_init()初始化连接处理程序，并通过调用mysql_real_connect()连接到服务器。

3. 发出SQL语句并处理其结果。（在下面的讨论中，详细介绍了使用它的方法）。

4. 通过调用mysql_close()，关闭与MySQL服务器的连接。

5. 通过调用mysql_library_end()，结束MySQL库的使用。

调用mysql_library_init()和mysql_library_end()的目的在于，为MySQL库提供恰当的初始化和结束处理。对于与客户端库链接的应用程序，它们提供了改进的内存管理功能。如果不调用mysql_library_end()，内存块仍将保持分配状态（这不会增加应用程序使用的内存量，但某些内存泄漏检测器将抗议它）。对于与嵌入式服务器链接的应用程序，这些调用会启动并停止服务器。

mysql_library_init()和mysql_library_end()实际上是#define符号，这类符号使得它们等效于mysql_server_init()和mysql_server_end()，但其名称更清楚地指明，无论应用程序使用的是mysqlclient或mysqld库，启动或结束MySQL库时，应调用它们。对于早期的MySQL版本，可调用mysql_server_init()和mysql_server_end()取而代之。

如果愿意，可省略对mysql_library_init()的调用，这是因为，必要时，mysql_init()会自动调用它。

要想连接到服务器，可调用mysql_init()来初始化连接处理程序，然后用该处理程序（以及其他信息，如主机名、用户名和密码）调用mysql_real_connect()。建立连接后，在低于5.0.3版的API中，mysql_real_connect()会将再连接标志（MYSQL结构的一部分）设置为1，或在较新的版本中，将其设置为0。对于该标志，值"1"指明，如果因连接丢失而无法执行语句，放弃之前，会尝试再次连接到服务器。从MySQL 5.0.13开始，可以在mysql_options()上使用MYSQL_OPT_RECONNECT选项，以控制再连接行为。完成连接后，调用mysql_close()中止它。

当连接处于活动状态时，客户端或许会使用mysql_query()或mysql_real_query()向服务器发出SQL查询。两者的差别在于，mysql_query()预期的查询为指定的、由Null终结的字符串，而mysql_real_query()预期的是计数字符串。如果字符串包含二进制数据（其中可能包含Null字节），就必须使用mysql_real_query()。

对于每个非SELECT查询（例如INSERT、UPDATE、DELETE），通过调用mysql_affected_rows()，可发现有多少行已被改变（影响）。

对于SELECT查询，能够检索作为结果集的行。注意，某些语句因其返回行，类似与SELECT。包括SHOW、DESCRIBE和EXPLAIN。应按照对待SELECT语句的方式处理它们。

客户端处理结果集的方式有两种。一种方式是，通过调用mysql_store_result()，一次性地检索整个结果集。该函数能从服务器获得查询返回的所有行，并将它们保存在客户端。第二种方式是针对客户端的，通过调用mysql_use_result()，对"按行"结果集检索进行初始化处理。该函数能初始化检索结果，但不能从服务器获得任何实际行。

在这两种情况下，均能通过调用mysql_fetch_row()访问行。通过mysql_store_result()，mysql_fetch_row()能够访问以前从服务器获得的行。通过mysql_use_result()，mysql_fetch_row()能够实际地检索来自服务器的行。通过调用mysql_fetch_lengths()，能获得关于各行中数据大小的信息。

完成结果集操作后，请调用mysql_free_result()释放结果集使用的内存。

这两种检索机制是互补的。客户端程序应选择最能满足其要求的方法。实际上，客户端最常使用的是mysql_store_result()。

mysql_store_result()的1个优点在于，由于将行全部提取到了客户端上，你不仅能连续访问行，还能使用mysql_data_seek()或mysql_row_seek()在结果集中向前或向后移动，以更改结果集内当前行的位置。通过调用mysql_num_rows()，还能发现有多少行。另一方面，对于大的结果集，mysql_store_result()所需的内存可能会很大，你很可能遇到内存溢出状况。

mysql_use_result()的1个优点在于，客户端所需的用于结果集的内存较少，原因在于，一次它仅维护一行（由于分配开销较低，mysql_use_result()能更快）。它的缺点在于，你必须快速处理每一行以避免妨碍服务器，你不能随机访问结果集中的行（只能连续访问行），你不知道结果集中有多少行，直至全部检索了它们为止。不仅如此，即使在检索过程中你判定已找到所寻找的信息，也必须检索所有的行。

通过API，客户端能够恰当地对查询作出响应（仅在必要时检索行），而无需知道查询是否是SELECT查询。可以在每次mysql_query()或mysql_real_query()后，通过调用mysql_store_result()完成该操作。如果结果集调用成功，查询为SELECT，而且能够读取行。如果结果集调用失败，可调用mysql_field_count()来判断结果是否的确是所预期的。如果mysql_field_count()返回0，查询不返回数据（表明它是INSERT、UPDATE、DELETE等），而且不返回行。如果mysql_field_count()是非0值，查询应返回行，但没有返回行。这表明查询是失败了的SELECT。关于如何实现该操作的示例，请参见关于mysql_field_count()的介绍。

无论是mysql_store_result()还是mysql_use_result()，均允许你获取关于构成结果集的字段的信息（字段数目，它们的名称和类型等）。通过重复调用mysql_fetch_field()，可以按顺序访问行内的字段信息，或者，通过调用mysql_fetch_field_direct()，能够在行内按字段编号访问字段信息。通过调用mysql_field_seek()，可以改变当前字段的光标位置。对字段光标的设置将影响后续的mysql_fetch_field()调用。此外，你也能通过调用mysql_fetch_fields()，一次性地获得关于字段的所有信息。

为了检测和通报错误，MySQL提供了使用mysql_errno()和mysql_error()函数访问错误信息的机制。它们能返回关于最近调用的函数的错误代码或错误消息，最近调用的函数可能成功也可能失败，这样，你就能判断错误是在何时出现的，以及错误是什么。

25.2.3. C API函数描述

25.2.3.1. mysql_affected_rows()
25.2.3.2. mysql_autocommit()
25.2.3.3. mysql_change_user()
25.2.3.4. mysql_character_set_name()
25.2.3.5. mysql_close()
25.2.3.6. mysql_commit()
25.2.3.7. mysql_connect()
25.2.3.8. mysql_create_db()
25.2.3.9. mysql_data_seek()
25.2.3.10. mysql_debug()
25.2.3.11. mysql_drop_db()
25.2.3.12. mysql_dump_debug_info()

25.2.3.13. mysql_eof()
25.2.3.14. mysql_errno()
25.2.3.15. mysql_error()
25.2.3.16. mysql_escape_string()
25.2.3.17. mysql_fetch_field()
25.2.3.18. mysql_fetch_field_direct()
25.2.3.19. mysql_fetch_fields()
25.2.3.20. mysql_fetch_lengths()
25.2.3.21. mysql_fetch_row()
25.2.3.22. mysql_field_count()
25.2.3.23. mysql_field_seek()
25.2.3.24. mysql_field_tell()
25.2.3.25. mysql_free_result()
25.2.3.26. mysql_get_character_set_info()
25.2.3.27. mysql_get_client_info()
25.2.3.28. mysql_get_client_version()
25.2.3.29. mysql_get_host_info()
25.2.3.30. mysql_get_proto_info()
25.2.3.31. mysql_get_server_info()
25.2.3.32. mysql_get_server_version()
25.2.3.33. mysql_hex_string()
25.2.3.34. mysql_info()
25.2.3.35. mysql_init()
25.2.3.36. mysql_insert_id()
25.2.3.37. mysql_kill()
25.2.3.38. mysql_library_end()
25.2.3.39. mysql_library_init()
25.2.3.40. mysql_list_dbs()
25.2.3.41. mysql_list_fields()
25.2.3.42. mysql_list_processes()
25.2.3.43. mysql_list_tables()
25.2.3.44. mysql_more_results()
25.2.3.45. mysql_next_result()
25.2.3.46. mysql_num_fields()
25.2.3.47. mysql_num_rows()
25.2.3.48. mysql_options()
25.2.3.49. mysql_ping()
25.2.3.50. mysql_query()
25.2.3.51. mysql_real_connect()
25.2.3.52. mysql_real_escape_string()
25.2.3.53. mysql_real_query()
25.2.3.54. mysql_refresh()
25.2.3.55. mysql_reload()
25.2.3.56. mysql_rollback()
25.2.3.57. mysql_row_seek()
25.2.3.58. mysql_row_tell()
25.2.3.59. mysql_select_db()
25.2.3.60. mysql_set_character_set()
25.2.3.61. mysql_set_server_option()
25.2.3.62. mysql_shutdown()
25.2.3.63. mysql_sqlstate()
25.2.3.64. mysql_ssl_set()
25.2.3.65. mysql_stat()
25.2.3.66. mysql_store_result()
25.2.3.67. mysql_thread_id()
25.2.3.68. mysql_use_result()
25.2.3.69. mysql_warning_count()

在本节所作的介绍中，按照C编程语言，为NULL的参数或返回值表示NULL，而不是MySQL Null值。

返回值的函数通常会返回指针或整数。除非作了其他规定，返回指针的函数将返回非Null值，以指明成功，或返回NULL值以指明出错。返回整数的函数将返回0以指明成功，或返回非0值以指明出错。注意，非0值仅表明这点。除非在函数描述中作了其他说明，不要对非0值进行测试：

```
if (result)                    /* correct */
```

```
    ... error ...


if (result < 0)               /* incorrect */
    ... error ...


if (result == -1)             /* incorrect */
    ... error ...
```

当函数返回错误时，在函数描述的"错误"部分将列出可能的错误类型。通过调用mysql_errno()可发现出现的错误是什么。通过调用mysql_error()，可获得错误的字符串表示。

25.2.3.1. mysql_affected_rows()

my_ulonglong mysql_affected_rows(MYSQL *mysql)

描述

返回上次UPDATE更改的行数，上次DELETE删除的行数，或上次INSERT语句插入的行数。对于UPDATE、DELETE或INSERT语句，可在mysql_query()后立刻调用。对于SELECT语句，mysql_affected_rows()的工作方式与mysql_num_rows()类似。

返回值

大于0的整数表明受影响或检索的行数。"0"表示UPDATE语句未更新记录，在查询中没有与WHERE匹配的行，或未执行查询。"-1"表示查询返回错误，或者，对于SELECT查询，在调用mysql_store_result()之前调用了mysql_affected_rows()。由于mysql_affected_rows()返回无符号值，通过比较返回值和"(my_ulonglong)-1"或等效的"(my_ulonglong)~0"，检查是否为"-1"。

错误

无。

示例:

```
mysql_query(&mysql,"UPDATE products SET cost=cost*1.25 WHERE group=10");
printf("%ld products updated",(long) mysql_affected_rows(&mysql));
```

如果在连接至**mysqld**时指定了标志CLIENT_FOUND_ROWS，对于UPDATE语句，mysql_affected_rows()将返回WHERE语句匹配的行数。

注意，使用REPLACE命令时，如果新行替代了旧行，mysql_affected_rows()返回2。这是因为，在该情况下，删除了重复行后插入了1行。

如果使用"INSERT ... ON DUPLICATE KEY UPDATE"来插入行，如果行是作为新行插入的，mysql_affected_rows()返回1，如果是更新了已有的行，返回2。

25.2.3.2. mysql_autocommit()

my_bool mysql_autocommit(MYSQL *mysql, my_bool mode)

描述

如果模式为"1"，启用autocommit模式；如果模式为"0"，禁止autocommit模式。

返回值

如果成功，返回0，如果出现错误，返回非0值。

错误

无。

25.2.3.3. mysql_change_user()

my_bool mysql_change_user(MYSQL *mysql, const char *user, const char *password, const char *db)

描述

更改用户，并使由db指定的数据库成为由mysql指定的连接上的默认数据库（当前数据库）。在后续查询中，对于不包含显式数据库区分符的表引用，该数据库是默认数据库。

如果不能确定已连接的用户或用户不具有使用数据库的权限，mysql_change_user()将失败。在这种情况下，不会改变用户和数据库。

如果不打算拥有默认数据库，可将db参数设置为NULL。

该命令总是会执行活动事务的ROLLBACK操作，关闭所有的临时表，解锁所有的锁定表，并复位状态，就像进行了新连接那样。即使未更改用户，也会出现该情况。

返回值

0表示成功，非0值表示出现错误。

错误

与从mysql_real_connect()获得的相同。

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中丢失了与服务器的连接。

· CR_UNKNOWN_ERROR

出现未知错误。

· ER_UNKNOWN_COM_ERROR

MySQL服务器未实施该命令（或许是较低版本的服务器）。

· ER_ACCESS_DENIED_ERROR

用户或密码错误。

· ER_BAD_DB_ERROR

数据库不存在。

· ER_DBACCESS_DENIED_ERROR

用户没有访问数据库的权限。

· ER_WRONG_DB_NAME

数据库名称过长。

示例:

```
if (mysql_change_user(&mysql, "user", "password", "new_database"))
{
   fprintf(stderr, "Failed to change user.  Error: %s\n",
           mysql_error(&mysql));
}
```

25.2.3.4. mysql_character_set_name()

const char *mysql_character_set_name(MYSQL *mysql)

描述

为当前连接返回默认的字符集。

返回值

默认字符集。

错误

无。

25.2.3.5. mysql_close()

void mysql_close(MYSQL *mysql)

描述

关闭前面打开的连接。如果句柄是由mysql_init()或mysql_connect()自动分配的，mysql_close()还将解除分配由mysql指向的连接句柄。

返回值

无。

错误

无。

25.2.3.6. mysql_commit()

my_bool mysql_commit(MYSQL *mysql)

描述

提交当前事务。

该函数的动作受completion_type系统变量的值控制。尤其是，如果completion_type的值为2，终结事务并关闭客户端连接后，服务器将执行释放操作。客户端程序应调用mysql_close()，从客户端一侧关闭连接。

返回值

如果成功，返回0，如果出现错误，返回非0值。

错误

无。

25.2.3.7. mysql_connect()

MYSQL *mysql_connect(MYSQL *mysql, const char *host, const char *user, const char *passwd)

描述

该函数已过时。最好使用mysql_real_connect()取而代之。

mysql_connect()试图建立与运行在主机上的MySQL数据库引擎的连接。在能够执行任何其他API函数之前，mysql_connect()必须成功完成，但mysql_get_client_info()例外。

这些参数的意义与mysql_real_connect()的对应参数的意义相同，差别在于连接参数可以为NULL。在这种情况下，C API将自动为连接结构分配内存，并当调用mysql_close()时释放分配的内存。该方法的缺点是，如果连接失败，你无法检索错误消息。要想从mysql_errno()或mysql_error()获得错误消息，必须提供有效的MYSQL指针。

返回值

与mysql_real_connect()的相同。

错误

与mysql_real_connect()的相同。

25.2.3.8. mysql_create_db()

int mysql_create_db(MYSQL *mysql, const char *db)

描述

创建由db参数命名的数据库。

该函数已过时。最好使用mysql_query()来发出SQL CREATE DATABASE语句。

返回值

如果数据库已成功创建，返回0，如果出现错误，返回非0值。

错误

- CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

- CR_SERVER_GONE_ERROR

MySQL服务器不可用。

- CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

- CR_UNKNOWN_ERROR

出现未知错误。

示例:

```
if(mysql_create_db(&mysql, "my_database"))
{
   fprintf(stderr, "Failed to create new database.  Error: %s\n",
           mysql_error(&mysql));
}
```

25.2.3.9. mysql_data_seek()

void mysql_data_seek(MYSQL_RES *result, my_ulonglong offset)

描述

在查询结果集中寻找任意行。偏移值为行号，范围从0到mysql_num_rows(result)-1。

该函数要求结果集结构包含查询的所有结果，因此，so mysql_data_seek()仅应与mysql_store_result()联合使用，而不是与mysql_use_result()。

返回值

无。

错误

无。

25.2.3.10. mysql_debug()

void mysql_debug(const char *debug)

描述

用给定的字符串执行DBUG_PUSH。mysql_debug()采用Fred Fish调试库。要想使用该函数，必须编译客户端库，使之支持调试功能。请参见E.1节，“调试MySQL服务器”。请参见E.2节，“调试MySQL客户端”。

返回值

无。

错误

无。

示例:

这里给出的调用将使客户端库在客户端机器的/tmp/client.trace中生成1个跟踪文件。

```
mysql_debug("d:t:O,/tmp/client.trace");
```

25.2.3.11. mysql_drop_db()

int mysql_drop_db(MYSQL *mysql, const char *db)

描述

撤销由db参数命名数据库。

该函数已过时。最好使用mysql_query()来发出SQL DROP DATABASE语句

返回值

如果成功撤销了数据库，返回0。如果出现错误，返回非0值。

错误

- CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

示例:

```
if(mysql_drop_db(&mysql, "my_database"))
  fprintf(stderr, "Failed to drop the database: Error: %s\n",
          mysql_error(&mysql));
```

25.2.3.12. mysql_dump_debug_info()

int mysql_dump_debug_info(MYSQL *mysql)

描述

指示服务器将一些调试信息写入日志。要想使之工作，已连接的用户必须具有SUPER权限。

返回值

如果命令成功，返回0。如果出现错误，返回非0值。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.13. mysql_eof()

my_bool mysql_eof(MYSQL_RES *result)

描述

该函数已过时。应使用mysql_errno()或mysql_error()取而代之。

mysql_eof()确定是否已读取了结果集的最后1行。

如果通过成功调用mysql_store_result()获得了结果集，客户端将在1次操作中收到整个结果集。在该情况下，从mysql_fetch_row()返回的NULL总表示已到达结果集末尾，而且没必要调用mysql_eof()。与mysql_store_result()一起使用时，mysql_eof()总返回"真"。

另一方面，如果你使用mysql_use_result()来初始化结果集检索，当重复调用mysql_fetch_row()时，将逐个地从服务器获取结果集的行。由于在该过程中，可能出现连接上的错误，从mysql_fetch_row()返回的NULL值不一定表示已正常地抵达结果集末尾。在该情况下，可以使用mysql_eof()来判定出现了什么情况。如果抵达结果集末尾，mysql_eof()返回非0值，如果出现错误，返回0。

从历史的角度上看，mysql_eof()在日期上早于标准的MySQL错误函数mysql_errno()和mysql_error()。由于这类错误函数提供了相同的信息，它们优先于已过时的mysql_eof()。事实上，它们提供了更多信息，这是因为，mysql_eof()仅返回布尔值，错误函数能够在出现错误时指明错误的原因。

返回值

如果未出现错误，返回0。如果抵达结果集的末尾，返回非0值。

错误

无。

示例:

在下面的示例中，介绍了使用mysql_eof()的方法:

```
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));
}
```

但是，你也能使用标准的MySQL错误函数实现相同的结果:

```
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));
}
```

25.2.3.14. mysql_errno()

unsigned int mysql_errno(MYSQL *mysql)

描述

对于由mysql指定的连接，mysql_errno()返回最近调用的API函数的错误代码，该函数调用可能成功也可能失败。“0”返回值表示未出现错误。在MySQL errmsg.h头文件中，列出了客户端错误消息编号。在附录B：错误代码和消息中，也列出了这些错误。

注意，如果成功，某些函数，如mysql_fetch_row()等，不会设置mysql_errno()。

经验规则是，如果成功，所有向服务器请求信息的函数均会复位mysql_errno()。

返回值

如果失败，返回上次mysql_*xxx*()调用的错误代码。"0"表示未出现错误。

错误

无。

25.2.3.15. mysql_error()

const char *mysql_error(MYSQL *mysql)

描述

对于由mysql指定的连接，对于失败的最近调用的API函数，mysql_error()返回包含错误消息的、由Null终结的字符串。如果该函数未失败，mysql_error()的返回值可能是以前的错误，或指明无错误的空字符串。

经验规则是，如果成功，所有向服务器请求信息的函数均会复位mysql_error()。

对于复位mysql_errno()的函数，下述两个测试是等效的：

```
if(mysql_errno(&mysql))
{
    // an error occurred
}

if(mysql_error(&mysql)[0] != '\0')
{
    // an error occurred
}
```

通过重新编译MySQL客户端库，可以更改客户端错误消息的语言。目前，能够选择数种语言显示错误消息，请参见5.10.2节，"设置错误消息语言"。

返回值

返回描述错误的、由Null终结的字符串。如果未出现错误，返回空字符串。

错误

无。

25.2.3.16. mysql_escape_string()

应使用mysql_real_escape_string()取而代之！

该函数与mysql_real_escape_string()等同，但mysql_real_escape_string()会将连接处理程序作为其第1个参量，并按照当前字符集对字符串进行转义处理。mysql_escape_string()不采用连接参量，而且不考虑当前字符集设置。

25.2.3.17. mysql_fetch_field()

MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result)

描述

返回采用MYSQL_FIELD结构的结果集的列。重复调用该函数，以检索关于结果集中所有列的信息。未剩余字段时，mysql_fetch_field()返回NULL。

每次执行新的SELECT查询时，将复位mysql_fetch_field()，以返回关于第1个字段的信息。调用mysql_field_seek()也会影响mysql_fetch_field()返回的字段。

如果调用了mysql_query()以在表上执行SELECT，但未调用mysql_store_result()，如果调用了mysql_fetch_field()以请求BLOB字段的长度，MySQL将返回默认的Blob长度（8KB）。之所以选择8KB是因为MySQL不知道BLOB的最大长度。应在日后使其成为可配置的。一旦检索了结果集，field->max_length将包含特定查询中该列的最大值的长度。

返回值

当前列的MYSQL_FIELD结构。如果未剩余任何列，返回NULL。

错误

无。

示例:

```
MYSQL_FIELD *field;


while((field = mysql_fetch_field(result)))
{
    printf("field name %s\n", field->name);
}
```

25.2.3.18. mysql_fetch_field_direct()

MYSQL_FIELD *mysql_fetch_field_direct(MYSQL_RES *result, unsigned int fieldnr)

描述

给定结果集内某1列的字段编号fieldnr，以MYSQL_FIELD结构形式返回列的字段定义。可以使用该函数检索任意列的定义。Fieldnr的值应在从0到mysql_num_fields(result)-1的范围内。

返回值

对于指定列，返回MYSQL_FIELD结构。

错误

无。

示例:

```
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);
}
```

25.2.3.19. mysql_fetch_fields()

MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)

描述

对于结果集，返回所有MYSQL_FIELD结构的数组。每个结构提供了结果集中1列的字段定义。

返回值

关于结果集所有列的MYSQL_FIELD结构的数组。

错误

无。

示例:

```
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);
}
```

25.2.3.20. mysql_fetch_lengths()

unsigned long *mysql_fetch_lengths(MYSQL_RES *result)

描述

返回结果集内当前行的列的长度。如果打算复制字段值，该长度信息有助于优化，这是因为，你能避免调用strlen()。此外，如果结果集包含二进制数据，必须使用该函数来确定数据的大小，原因在于，对于包含Null字符的任何字段，strlen()将返回错误的结果。

对于空列以及包含NULL值的列，其长度为0。要想了解区分这两类情况的方法，请参见关于mysql_fetch_row()的介绍。

返回值

无符号长整数的数组表示各列的大小（不包括任何终结NULL字符）。如果出现错误，返回NULL。

错误

mysql_fetch_lengths()仅对结果集的当前行有效。如果在调用mysql_fetch_row()之前或检索了结果集中的所有行后调用了它，将返回NULL。

示例:

```
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]);
    }
}
```

25.2.3.21. mysql_fetch_row()

MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)

描述

检索结果集的下一行。在mysql_store_result()之后使用时，如果没有要检索的行，mysql_fetch_row()返回NULL。在mysql_use_result()之后使用时，如果没有要检索的行或出现了错误，mysql_fetch_row()返回NULL。

行内值的数目由mysql_num_fields(result)给出。如果行中保存了调用mysql_fetch_row()返回的值，将按照row[0]到row[mysql_num_fields(result)-1]，访问这些值的指针。行中的NULL值由NULL指针指明。

可以通过调用mysql_fetch_lengths()来获得行中字段值的长度。对于空字段以及包含NULL的字段，长度为0。通过检查字段值的指针，能够区分它们。如果指针为NULL，字段为NULL，否则字段为空。

返回值

下一行的MYSQL_ROW结构。如果没有更多要检索的行或出现了错误，返回NULL。

错误

注意，在对mysql_fetch_row()的两次调用之间，不会复位错误。

- CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

- CR_UNKNOWN_ERROR

出现未知错误。

示例:

```
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");
}
```

25.2.3.22. mysql_field_count()

unsigned int mysql_field_count(MYSQL *mysql)

描述

返回作用在连接上的最近查询的列数。

该函数的正常使用是在mysql_store_result()返回NULL（因而没有结果集指针）时。在这种情况下，可调用mysql_field_count()来判定mysql_store_result()是否应生成非空结果。这样，客户端就能采取恰当的动作，而无需知道查询是否是SELECT（或类似SELECT的）语句。在这里给出的示例中，演示了完成它的方法。

请参见25.2.13.1节，“为什么在mysql_query()返回成功后，mysql_store_result()有时会返回NULL”.

返回值

表示结果集中列数的无符号整数。

错误

无。

示例:

```
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));
        }
    }
}
```

另一种可选的方法是，用mysql_errno(&mysql)替换mysql_field_count(&mysql)调用。在该情况下，无论语句是否是SELECT，你将直接从mysql_store_result()查找错误，而不是从mysql_field_count()的值进行推断。

25.2.3.23. mysql_field_seek()

MYSQL_FIELD_OFFSET mysql_field_seek(MYSQL_RES *result, MYSQL_FIELD_OFFSET offset)

描述

将字段光标设置到给定的偏移处。对mysql_fetch_field()的下一次调用将检索与该偏移相关的列定义。

要想查找行的开始，请传递值为0的偏移量。

返回值

字段光标的前一个值。

错误

无。

25.2.3.24. mysql_field_tell()

MYSQL_FIELD_OFFSET mysql_field_tell(MYSQL_RES *result)

描述

返回上一个mysql_fetch_field()所使用的字段光标的定义。该值可用作mysql_field_seek()的参量。

返回值

字段光标的当前偏移量。

错误

无。

25.2.3.25. mysql_free_result()

void mysql_free_result(MYSQL_RES *result)

描述

释放由mysql_store_result()、mysql_use_result()、mysql_list_dbs()等为结果集分配的内存。完成对结果集的操作后，必须调用mysql_free_result()释放结果集使用的内存。

释放完成后，不要尝试访问结果集。

返回值

无。

错误

无。

25.2.3.26. mysql_get_character_set_info()

void mysql_get_character_set_info(MYSQL *mysql, MY_CHARSET_INFO *cs)

描述

该函数提供了关于默认客户端字符集的信息。可以使用mysql_set_character_set()函数更改默认的字符集。

该函数是在MySQL 5.0.10中增加的。

示例:

```
if (!mysql_set_character_set(&mysql, "utf8"))
{
    MY_CHARSET_INFO cs;
    mysql_get_character_set_info(&mysql, &cs);
    printf("character set information:\n");
    printf("character set name: %s\n", cs.name);
    printf("collation name: %s\n", cs.csname);
    printf("comment: %s\n", cs.comment);
    printf("directory: %s\n", cs.dir);
    printf("multi byte character min. length: %d\n", cs.mbminlen);
    printf("multi byte character max. length: %d\n", cs.mbmaxlen);
}
```

25.2.3.27. mysql_get_client_info()

char *mysql_get_client_info(void)

描述

返回表示客户端库版本的字符串。

返回值

表示MySQL客户端库版本的字符串。

错误

无。

25.2.3.28. mysql_get_client_version()

unsigned long mysql_get_client_version(void)

描述

返回表示客户端库版本的整数。该值的格式是XYYZZ，其中X是主版本号，YY是发布级别，ZZ是发布级别内的版本号。例如，值40102表示客户端库的版本是4.1.2。

返回值

表示MySQL客户端库版本的整数。

错误

无。

25.2.3.29. mysql_get_host_info()

char *mysql_get_host_info(MYSQL *mysql)

描述

返回描述了所使用连接类型的字符串，包括服务器主机名。

返回值

代表服务器主机名和连接类型的字符串。

错误

无。

25.2.3.30. mysql_get_proto_info()

unsigned int mysql_get_proto_info(MYSQL *mysql)

描述

返回当前连接所使用的协议版本。

返回值

代表当前连接所使用协议版本的无符号整数。

错误

无。

25.2.3.31. mysql_get_server_info()

char *mysql_get_server_info(MYSQL *mysql)

描述

返回代表服务器版本号的字符串。

返回值

代表服务器版本号的字符串。

错误

无。

25.2.3.32. mysql_get_server_version()

unsigned long mysql_get_server_version(MYSQL *mysql)

描述

以整数形式返回服务器的版本号。

返回值

表示MySQL服务器版本的数值，格式如下：

```
major_version*10000 + minor_version *100 + sub_version
```

例如，对于5.0.12，返回500012。

在客户端程序中，为了快速确定某些与版本相关的服务器功能是否存在，该函数很有用。

错误

无。

25.2.3.33. mysql_hex_string()

unsigned long mysql_hex_string(char *to, const char *from, unsigned long length)

描述

该函数用于创建可用在SQL语句中的合法SQL字符串。请参见9.1.1节，“字符串”。

该字符串从形式上编码为十六进制格式，每个字符编码为2个十六进制数。结果被置入其中，并添加1个终结Null字节。

“from”所指向的字符串必须是长度字节“long”。必须为“to”分配缓冲区，缓冲区至少为length*2+1字节长。
当mysql_hex_string()返回时，“to”的内容为由Null终结的字符串。返回值是编码字符串的长度，不包括终结用Null字符。

可采用0x*value*或X'*value*'格式将返回值置于SQL语句中。但是，返回值不包括0x或X'...'。调用者必须提供所希望的格式是何种。

示例：

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
end = strmov(end,"0x");
end += mysql_hex_string(end,"What's this",11);
end = strmov(end,",0x");
end += mysql_hex_string(end,"binary data: \0\r\n",16);
*end++ = ')';

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
   fprintf(stderr, "Failed to insert row, Error: %s\n",
           mysql_error(&mysql));
}
```

示例中所使用的strmov()函数包含在mysqlclient库中，它的工作方式类似于strcpy()，但返回指向第1个参数终结Null的指针。

返回值

置于“to”中的值的长度，不包括终结用Null字符。

错误

无。

25.2.3.34. mysql_info()

char *mysql_info(MYSQL *mysql)

描述

检索字符串，该字符串提供了关于最近执行查询的信息，但仅对这里列出的语句有效。对于其他语句，mysql_info()返回NULL。字符串的格式取决于查询的类型，如本节所述。数值仅是说明性的，字符串包含与查询相适应的值。

· INSERT INTO ... SELECT ...

字符串格式：记录，100；副本，0；警告，0

· INSERT INTO ... VALUES (...),(...),(...)...

字符串格式：记录，3；副本，0；警告，0

· LOAD DATA INFILE ...

字符串格式：记录，1；删除，0；跳过，0；警告，0

· ALTER TABLE

字符串格式：记录，3；副本，0；警告，0

· UPDATE

字符串格式：匹配行，40；更改，40；警告，0

注意，mysql_info()为INSERT ... VALUES返回非NULL值，INSERT ... VALUES仅用于多行形式的语句（也就是说，仅当指定了多个值列表时）。

返回值

字符串，它表示最近所执行查询的额外信息。如果该查询无可用信息，返回NULL。

错误

无。

25.2.3.35. mysql_init()

MYSQL *mysql_init(MYSQL *mysql)

描述

分配或初始化与mysql_real_connect()相适应的MYSQL对象。如果mysql是NULL指针，该函数将分配、初始化、并返回新对象。否则，将初始化对象，并返回对象的地址。如果mysql_init()分配了新的对象，当调用mysql_close()来关闭连接时。将释放该对象。

返回值

初始化的MYSQL*句柄。如果无足够内存以分配新的对象，返回NULL。

错误

在内存不足的情况下，返回NULL。

25.2.3.36. mysql_insert_id()

my_ulonglong mysql_insert_id(MYSQL *mysql)

描述

返回由以前的INSERT或UPDATE语句为AUTO_INCREMENT列生成的值。在包含AUTO_INCREMENT字段的表中执行了INSERT语句后，应使用该函数。

更准确地讲，将在下述条件下更新mysql_insert_id()：

· 将值保存到AUTO_INCREMENT列中的INSERT语句。无论值是通过在列中存储特殊值NULL或0自动生成的，还是确切的非特殊值，都成立。

· 在有多行INSERT语句的情况下，mysql_insert_id()返回第1个自动生成的AUTO_INCREMENT值，如果未生成这类值，将返回插入在AUTO_INCREMENT列中的最后1个确切值。

· 通过将LAST_INSERT_ID(*expr*)插入到任意列中以生成AUTO_INCREMENT值的INSERT语句。

· 通过更新任意列至LAST_INSERT_ID(*expr*)以生成AUTO_INCREMENT值的INSERT语句。

· mysql_insert_id()的值不受诸如SELECT等返回结果集的语句的影响。

· 如果前面的语句返回了错误，mysql_insert_id()的值将是不确定的。

注意，如果前面的语句未使用AUTO_INCREMENT，mysql_insert_id()返回0。如果需要保存值，在生成值的语句后，务必立刻调用mysql_insert_id()。

mysql_insert_id()的值仅受在当前客户端连接内发出的语句的影响。不受由其他客户端发出的语句的影响。

请参见12.9.3节，“信息函数”。

此外还应注意，SQL LAST_INSERT_ID()函数的值总包含最近生成的AUTO_INCREMENT值，而且在语句之间不会被复位，原因在于该函数的值是在服务器中维护的。另一个区别是，如果设置了AUTO_INCREMENT列来指定非特殊值，不会更新LAST_INSERT_ID()。

LAST_INSERT_ID()不同于mysql_insert_id()的原因在于，LAST_INSERT_ID()在脚本中很容易使用，而mysql_insert_id()则试图提供关于在AUTO_INCREMENT列中出现情况的更准确信息。

返回值

在前面的讨论中予以了介绍。

错误

无。

25.2.3.37. mysql_kill()

int mysql_kill(MYSQL *mysql, unsigned long pid)

描述

请求服务器杀死由pid指定的线程。

返回值

0表示成功，非0值表示出现错误。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

- CR_SERVER_GONE_ERROR

MySQL服务器不可用。

- CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

- CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.38. mysql_library_end()

void mysql_library_end(void)

描述

它是mysql_server_end()函数的同义词。

关于具体的用法，请参见25.2.2节，“C API函数概述”。

25.2.3.39. mysql_library_init()

int mysql_library_init(int argc, char **argv, char **groups)

描述

这是mysql_server_init()函数的同义词。

关于具体的用法，请参见25.2.2节，“C API函数概述”。

25.2.3.40. mysql_list_dbs()

MYSQL_RES *mysql_list_dbs(MYSQL *mysql, const char *wild)

描述

返回由服务器上的数据库名称组成的结果集，该服务器与由通配符参数指定的简单正则表达式匹配。通配符参数可以包含通配符“%”或“_”，也可以是NULL指针，以便与所有的数据库匹配。调用mysql_list_dbs()的方法类似于执行查询SHOW database [LIKE wild]。

必须用mysql_free_result()释放结果集。

返回值

成功后返回MYSQL_RES结果集。如果出现错误，返回NULL。

错误

- CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

- CR_OUT_OF_MEMORY

内存溢出。

- CR_SERVER_GONE_ERROR

MySQL服务器不可用。

- CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

- CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.41. mysql_list_fields()

MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const char *wild)

描述

返回由给定表中的字段名称组成的结果集，给定表与由通配符参数指定的简单正则表达式匹配。通配符参数可以包含通配符"%"或"_"，也可以是NULL指针，以便与所有的字段匹配。调用mysql_list_fields()的方法类似于执行查询SHOW COLUMNS FROM *tbl_name* [LIKE *wild*]。

注意，建议使用SHOW COLUMNS FROM *tbl_name*，而不是mysql_list_fields()。

必须用mysql_free_result()释放结果集。

返回值

如果成功，返回MYSQL_RES结果集。如果出现错误，返回NULL。

错误

- CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

- CR_SERVER_GONE_ERROR

MySQL服务器不可用。

- CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

- CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.42. mysql_list_processes()

MYSQL_RES *mysql_list_processes(MYSQL *mysql)

描述

返回描述当前服务器线程的结果集。该类信息与**mysqladmin processlist**或SHOW PROCESSLIST查询给出的信息相同。

必须用mysql_free_result()释放结果集。

返回值

如果成功，返回MYSQL_RES结果集。如果出现错误，返回NULL。

错误

- CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.43. mysql_list_tables()

MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)

描述

返回由当前数据库内的表名组成的结果集，当前数据库与由通配符参数指定的简单正则表达式匹配。通配符参数可以包含通配符"%"或"_"，也可以是NULL指针，以便与所有的表匹配。调用mysql_list_tables()的方法类似于执行查询HOW tables [LIKE *wild*]。

必须用mysql_free_result()释放结果集。

返回值

如果成功，返回MYSQL_RES结果集。 如果出现错误，返回NULL。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.44. mysql_more_results()

my_bool mysql_more_results(MYSQL *mysql)

描述

如果当前执行的查询存在多个结果，返回"真"，而且应用程序必须调用mysql_next_result()来获取结果。

返回值

如果存在多个结果，返回"真"（1），如果不存在多个结果，返回"假"（0）。

在大多数情况下，可调用mysql_next_result()来测试是否存在多个结果，如果存在多个结果，对检索进行初始化操作。

请参见25.2.9节，"多查询执行的C API处理"。请参见25.2.3.45节，"mysql_next_result()"。

错误

无。

25.2.3.45. mysql_next_result()

int mysql_next_result(MYSQL *mysql)

描述

如果存在多个查询结果，mysql_next_result()将读取下一个查询结果，并将状态返回给应用程序。

如果前面的查询返回了结果集，必须为其调用mysql_free_result()。

调用了mysql_next_result()后，连接状态就像你已为下一查询调用了mysql_real_query()或mysql_query()时的一样。这意味着你能调用mysql_store_result()、mysql_warning_count()、mysql_affected_rows()等等。

如果mysql_next_result()返回错误，将不执行任何其他语句，也不会获取任何更多的结果，

请参见25.2.9节，“多查询执行的C API处理”。

返回值

返回值	描述
0	成功并有多个结果。
-1	成功但没有多个结果。
>0	出错

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。例如，没有为前面的结果集调用mysql_use_result()。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.46. mysql_num_fields()

unsigned int mysql_num_fields(MYSQL_RES *result)

要想传递MYSQL*参量取而代之，请使用无符号整数mysql_field_count(MYSQL *mysql)。

描述

返回结果集中的行数。

注意，你可以从指向结果集的指针或指向连接句柄的指针获得行数。如果mysql_store_result()或mysql_use_result()返回NULL，应使用连接句柄（因而没有结果集指针）。在该情况下，可调用mysql_field_count()来判断mysql_store_result()是否生成了非空结果。这样，客户端程序就能采取恰当的行动，而不需要知道查询是否是SELECT语句（或类似SELECT的语句）。在下面的示例中，介绍了执行该操作的方式。

请参见25.2.13.1节，“为什么在mysql_query()返回成功后，mysql_store_result()有时会返回NULL”。

返回值

表示结果集中行数的无符号整数。

错误

无。

示例:

```
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);
        }
    }
}
```

另一种可选方式是（如果你知道你的查询应返回结果集），使用检查“mysql_field_count(&mysql) is = 0”来替换mysql_errno(&mysql)调用。仅当出错时才应使用它。

25.2.3.47. mysql_num_rows()

my_ulonglong mysql_num_rows(MYSQL_RES *result)

描述

返回结果集中的行数。

mysql_num_rows()的使用取决于是否采用了mysql_store_result()或mysql_use_result()来返回结果集。如果使用了mysql_store_result()，可以立刻调用mysql_num_rows()。如果使用了mysql_use_result()，mysql_num_rows()不返回正确的值，直至检索了结果集中的所有行为止。

返回值

结果集中的行数。

错误

无。

25.2.3.48. mysql_options()

int mysql_options(MYSQL *mysql, enum mysql_option option, const char *arg)

描述

可用于设置额外的连接选项，并影响连接的行为。可多次调用该函数来设置数个选项。

应在mysql_init()之后、以及mysql_connect()或mysql_real_connect()之前调用mysql_options()。

选项参量指的是你打算设置的选项。Arg参量是选项的值。如果选项是整数，那么arg应指向整数的值。

可能的选项值:

选项	参量类型	功能
MYSQL_INIT_COMMAND	char *	连接到MySQL服务器时将执行的命令。再次连接时将自动地再次执行。
MYSQL_OPT_COMPRESS	未使用	使用压缩客户端／服务器协议
MYSQL_OPT_CONNECT_TIMEOUT	unsigned int *	以秒为单位的连接超时。
MYSQL_OPT_GUESS_CONNECTION	未使用	对于与libmysqld链接的应用程序，允许库"猜测"是否使用嵌入式服务器或远程服务器。"猜测"表示，如果设置了主机名但不是本地主机，将使用远程服务器。该行为是默认行为。 可使用MYSQL_OPT_USE_EMBEDDED_CONNECTION和MYSQL_OPT_USE_REMOTE_CONNECTION覆盖它。对于与libmysqlclient链接的应用程序，该选项将被忽略。
MYSQL_OPT_LOCAL_INFILE	指向单元的可选指针	如果未给定指针，或指针指向"unsigned int != 0"，将允许命令LOAD LOCAL INFILE。
MYSQL_OPT_NAMED_PIPE	未使用	使用命名管道连接到NT平台上的MySQL服务器。
MYSQL_OPT_PROTOCOL	unsigned int *	要使用的协议类型。应是mysql.h中定义的mysql_protocol_type的枚举值之一。
MYSQL_OPT_READ_TIMEOUT	unsigned int *	从服务器读取信息的超时（目前仅在Windows平台的TCP/IP连接上有效）。
MYSQL_OPT_RECONNECT	my_bool *	如果发现连接丢失，启动或禁止与服务器的自动再连接。从MySQL 5.0.3开始，默认情况下禁止再连接，这是5.0.13中的新选项，提供了一种以显式方式设置再连接行为的方法。
MYSQL_OPT_SET_CLIENT_IP	char *	对于与libmysqld链接的应用程序（具备鉴定支持特性的已编译libmysqld），它意味着，出于鉴定目的，用户将被视为从指定的IP地址（指定

		为字符串）进行连接。对于与libmysqlclient链接的应用程序，，该选项将被忽略。
MYSQL_OPT_USE_EMBEDDED_CONNECTION	未使用	对于与libmysqld链接的应用程序，对于连接来说，它将强制使用嵌入式服务器。对于与libmysqlclient链接的应用程序，，该选项将被忽略。
MYSQL_OPT_USE_REMOTE_CONNECTION	未使用	对于与libmysqld链接的应用程序，对于连接来说，它将强制使用远程服务器。对于与libmysqlclient链接的应用程序，，该选项将被忽略。
MYSQL_OPT_USE_RESULT	未使用	不使用该选项。
MYSQL_OPT_WRITE_TIMEOUT	unsigned int *	写入服务器的超时（目前仅在Windows平台的TCP/IP连接上有效）。
MYSQL_READ_DEFAULT_FILE	char *	从命名选项文件而不是从my.cnf读取选项。
MYSQL_READ_DEFAULT_GROUP	char *	从my.cnf或用MYSQL_READ_DEFAULT_FILE指定的文件中的命名组读取选项。
MYSQL_REPORT_DATA_TRUNCATION	my_bool *	通过MYSQL_BIND.error，对于预处理语句，允许或禁止通报数据截断错误（默认为禁止）。
MYSQL_SECURE_AUTH	my_bool*	是否连接到不支持密码混编功能的服务器，在MySQL 4.1.1和更高版本中，使用了密码混编功能。
MYSQL_SET_CHARSET_DIR	char*	指向包含字符集定义文件的目录的路径名。
MYSQL_SET_CHARSET_NAME	char*	用作默认字符集的字符集的名称。
MYSQL_SHARED_MEMORY_BASE_NAME	char*	命名为与服务器进行通信的共享内存对象。应与你打算连接的mysqld服务器使用的选项"-shared-memory-base-name"相同。

注意，如果使用了MYSQL_READ_DEFAULT_FILE或MYSQL_READ_DEFAULT_GROUP，总会读取客户端组。

选项文件中指定的组可能包含下述选项：

选项	描述
connect-timeout	以秒为单位的连接超时。在Linux平台上，该超时也用作等待服务器首次回应的时间。
compress	使用压缩客户端／服务器协议。
database	如果在连接命令中未指定数据库，连接到该数据库。
debug	调试选项。
disable-local-infile	禁止使用LOAD DATA LOCAL。
host	默认主机名。
init-command	连接到MySQL服务器时将执行的命令。再次连接时将自动地再次执行。
interactive-timeout	等同于将CLIENT_INTERACTIVE指定为mysql_real_connect()。请参见25.2.3.51节，"mysql_real_connect()"。
local-infile[=(0|1)]	如果无参量或参量!= 0，那么将允许使用LOAD DATA LOCAL。
max_allowed_packet	客户端能够从服务器读取的最大信息包。
multi-results	允许多语句执行或存储程序的多个结果集。
multi-statements	允许客户端在1个字符串内发送多条语句。（由";"隔开）。
password	默认密码。
pipe	使用命名管道连接到NT平台上的MySQL服务器。
protocol={TCP | SOCKET | PIPE | MEMORY}	连接到服务器时将使用的协议。
port	默认端口号。
return-found-rows	通知mysql_info()返回发现的行，而不是使用UPDATE时更新的行。
shared-memory-base-name=*name*	共享内存名称，用于连接到服务器（默认为"MYSQL"）。
socket	默认的套接字文件。
user	默认用户。

注意，“timeout”（超时）已被“connect-timeout”（连接超时）取代，但为了保持向后兼容，在MySQL 5.1.2-alpha中仍支持“timeout”（超时）。

关于选项文件的更多信息，请参见4.3.2节，“使用选项文件”。

返回值

成功时返回0。如果使用了未知选项，返回非0值。

示例：

```
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));
}
```

该代码请求客户端使用压缩客户端／服务器协议，并从my.cnf文件的obdc部分读取额外选项。

25.2.3.49. mysql_ping()

int mysql_ping(MYSQL *mysql)

描述

检查与服务器的连接是否工作。如果连接丢失，将自动尝试再连接。

该函数可被闲置了较长时间的客户端使用，用以检查服务器是否已关闭了连接，并在必要时再次连接。

返回值

如果与服务器的连接有效返回0。如果出现错误，返回非0值。返回的非0值不表示MySQL服务器本身是否已关闭，连接可能因其他原因终端，如网络问题等。

错误

- CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

- CR_SERVER_GONE_ERROR

MySQL服务器不可用。

- CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.50. mysql_query()

int mysql_query(MYSQL *mysql, const char *query)

描述

执行由"Null终结的字符串"查询指向的SQL查询。正常情况下，字符串必须包含1条SQL语句，而且不应为语句添加终结分号（';'）或"\g"。如果允许多语句执行，字符串可包含多条由分号隔开的语句。请参见25.2.9节，"多查询执行的C API处理"。

mysql_query()不能用于包含二进制数据的查询，应使用mysql_real_query()取而代之（二进制数据可能包含字符'\0'，mysql_query()会将该字符解释为查询字符串结束）。

如果希望了解查询是否应返回结果集，可使用mysql_field_count()进行检查。请参见25.2.3.22节，"mysql_field_count()"。

返回值

如果查询成功，返回0。如果出现错误，返回非0值。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.51. 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)

描述

mysql_real_connect()尝试与运行在主机上的MySQL数据库引擎建立连接。在你能够执行需要有效MySQL连接句柄结构的任何其他API函数之前，mysql_real_connect()必须成功完成。

参数的指定方式如下：

· 第1个参数应是已有MYSQL结构的地址。调用mysql_real_connect()之前，必须调用mysql_init()来初始化MYSQL结构。通过mysql_options()调用，可更改多种连接选项。请参见25.2.3.48节，"mysql_options()"。

· "host"的值必须是主机名或IP地址。如果"host"是NULL或字符串"localhost"，连接将被视为与本地主机的连接。如果操作系统支持套接字（Unix）或命名管道（Windows），将使用它们而不是TCP/IP连接到服务器。

· "user"参数包含用户的MySQL登录ID。如果"user"是NULL或空字符串""，用户将被视为当前用户。在UNIX环境下，它是当前的登录名。在Windows ODBC下，必须明确指定当前用户名。请参见26.1.9.2节，"在Windows上配置MyODBC DSN"。

· "passwd"参数包含用户的密码。如果"passwd"是NULL，仅会对该用户的（拥有1个空密码字段的）用户表中的条目进行匹配检查。这样，数据库管理员就能按特定的方式设置MySQL权限系统，根据用户是否拥有指定的密码，用户将获得不同的权限。

注释：调用mysql_real_connect()之前，不要尝试加密密码，密码加密将由客户端API自动处理。

· "db"是数据库名称。如果db为NULL，连接会将默认的数据库设为该值。

· 如果"port"不是0，其值将用作TCP/IP连接的端口号。注意，"host"参数决定了连接的类型。

· 如果unix_socket不是NULL，该字符串描述了应使用的套接字或命名管道。注意，"host"参数决定了连接的类型。

· client_flag的值通常为0，但是，也能将其设置为下述标志的组合，以允许特定功能：

标志名称	标志描述
CLIENT_COMPRESS	使用压缩协议。
CLIENT_FOUND_ROWS	返回发现的行数（匹配的），而不是受影响的行数。
CLIENT_IGNORE_SPACE	允许在函数名后使用空格。使所有的函数名成为保留字。
CLIENT_INTERACTIVE	关闭连接之前，允许interactive_timeout（取代了wait_timeout）秒的不活动时间。客户端的会话wait_timeout变量被设为会话interactive_timeout变量的值。
CLIENT_LOCAL_FILES	允许LOAD DATA LOCAL处理功能。
CLIENT_MULTI_STATEMENTS	通知服务器，客户端可能在单个字符串内发送多条语句（由';'隔开）。如果未设置该标志，将禁止多语句执行。
CLIENT_MULTI_RESULTS	通知服务器，客户端能够处理来自多语句执行或存储程序的多个结果集。如果设置了CLIENT_MULTI_STATEMENTS，将自动设置它。
CLIENT_NO_SCHEMA	禁止*db_name.tbl_name.col_name*语法。它用于ODBC。如果使用了该语法，它会使分析程序生成错误，在捕获某些ODBC程序中的缺陷时，它很有用。
CLIENT_ODBC	客户端是ODBC客户端。它将**mysqld**变得更为ODBC友好。
CLIENT_SSL	使用SSL（加密协议）。该选项不应由应用程序设置，它是在客户端库内部设置的。

对于某些参数，能够从选项文件获得取值，而不是取得mysql_real_connect()调用中的确切值。为此，在调用mysql_real_connect()之前，应与MYSQL_READ_DEFAULT_FILE或MYSQL_READ_DEFAULT_GROUP选项一起调用mysql_options()。随后，在mysql_real_connect()调用中，为准备从选项文件读取值的每个参数指定"无值"值：

- 对于host，指定NULL值或空字符串("")。
- 对于user，指定NULL值或空字符串。
- 对于passwd，指定NULL值。（对于密码，mysql_real_connect()调用中的空字符串的值不能被选项文件中的字符串覆盖，这是因为，空字符串明确指明MySQL账户必须有空密码）。
- 对于db，指定NULL值或空字符串
- 对于port，指定"0"值。
- 对于unix_socket，指定NULL值。

对于某一参数，如果在选项文件中未发现值，将使用它的默认值，如本节前面介绍的那样。

返回值

如果连接成功，返回MYSQL*连接句柄。如果连接失败，返回NULL。对于成功的连接，返回值与第1个参数的值相同。

错误

- CR_CONN_HOST_ERROR

无法连接到MySQL服务器。

- CR_CONNECTION_ERROR

无法连接到本地MySQL服务器。

- CR_IPSOCK_ERROR

无法创建IP套接字。

- CR_OUT_OF_MEMORY

内存溢出。

- CR_SOCKET_CREATE_ERROR

无法创建Unix套接字。

· CR_UNKNOWN_HOST

无法找到主机名的IP地址。

· CR_VERSION_ERROR

协议不匹配，起因于：试图连接到具有特定客户端库（该客户端库使用了不同的协议版本）的服务器。如果使用很早的客户端库来建立与较新的服务器（未使用"--old-protocol"选项开始的）的连接，就会出现该情况。

· CR_NAMEDPIPEOPEN_ERROR

无法在Windows平台下创建命名管道。

· CR_NAMEDPIPEWAIT_ERROR

在Windows平台下等待命名管道失败。

· CR_NAMEDPIPESETSTATE_ERROR

在Windows平台下获取管道处理程序失败。

· CR_SERVER_LOST

如果connect_timeout > 0，而且在连接服务器时所用时间长于connect_timeout秒，或在执行init-command时服务器消失。

示例：

```
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));
}
```

通过使用mysql_options()，MySQL库将读取my.cnf文件的[client]和[your_prog_name]部分，以确保程序工作，即使某人以某种非标准的方式设置MySQL也同样。

注意，一旦建立了连接，mysql_real_connect()将设置再连接标志（MYSQL结构的组成部份）的值，在低于5.0.3版的API中，将其设为"1"，在较新的版本中，将其设为"0"。对于该标志，值"1"表示，如果因连接丢失而无法执行语句，放弃前，将尝试再次连接到服务器。从MySQL 5.0.13开始，可以对mysql_options()使用MYSQL_OPT_RECONNECT选项，对再连接行为进行控制。

25.2.3.52. mysql_real_escape_string()

unsigned long mysql_real_escape_string(MYSQL *mysql, char *to, const char *from, unsigned long length)

注意，mysql必须是有效的开放式连接。之所以需要它是因为，转义功能取决于服务器使用的字符集。

描述

该函数用于创建可在SQL语句中使用的合法SQL字符串。请参见9.1.1节，"字符串"。

按照连接的当前字符集，将"from"中的字符串编码为转义SQL字符串。将结果置于"to"中，并添加1个终结用NULL字节。编码的字符为NUL (ASCII 0)、'\n'、'\r'、'\'、'''、'"'、以及Control-Z（请参见9.1节，"文字值"）。（严格地讲，MySQL仅需要反斜杠和引号字符，用于引用转义查询中的字符串。该函数能引用其他字符，从而使得它们在日志文件中具有更好的可读性）。

"from"指向的字符串必须是长度字节"long"。必须为"to"缓冲区分配至少length*2+1字节。在最坏的情况下，每个字符或许需要使用2个字节进行编码，而且还需要终结Null字节。当mysql_real_escape_string()返回时，"to"的内容是由Null终结的字符串。

返回值是编码字符串的长度，不包括终结用Null字符。

如果需要更改连接的字符集，应使用mysql_set_character_set()函数，而不是执行SET NAMES (或SET CHARACTER SET)语句。mysql_set_character_set()的工作方式类似于SET NAMES，但它还能影响mysql_real_escape_string()所使用的字符集，而SET NAMES则不能。

示例:

```
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));
}
```

该示例中使用的strmov()函数包含在mysqlclient库中，工作方式与strcpy()类似，但会返回指向第1个参数终结用Null的指针。

返回值

置于"to"中的值的长度，不包括终结用Null字符。

错误

无。

25.2.3.53. mysql_real_query()

int mysql_real_query(MYSQL *mysql, const char *query, unsigned long length)

描述

执行由"query"指向的SQL查询，它应是字符串长度字节"long"。正常情况下，字符串必须包含1条SQL语句，而且不应为语句添加终结分号（';'）或"\g"。如果允许多语句执行，字符串可包含由分号隔开的多条语句。请参见25.2.9节，"多查询执行的C API处理"。

对于包含二进制数据的查询，必须使用mysql_real_query()而不是mysql_query()，这是因为，二进制数据可能会包含'\0'字符。此外，mysql_real_query()比mysql_query()快，这是因为它不会在查询字符串上调用strlen()。

如果希望知道查询是否应返回结果集，可使用mysql_field_count()进行检查25.2.3.22节，"mysql_field_count()"。

返回值

如果查询成功，返回0。如果出现错误，返回非0值。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.54. mysql_refresh()

int mysql_refresh(MYSQL *mysql, unsigned int options)

描述

该函数用于刷新表或高速缓冲，或复位复制服务器信息。连接的用户必须具有RELOAD权限。

"options"参量是一种位掩码，由下述值的任意组合构成。能够以"Or"（或）方式将多个值组合在一起，用一次调用执行多项操作。

· REFRESH_GRANT

刷新授权表，与FLUSH PRIVILEGES类似。

· REFRESH_LOG

刷新日志，与FLUSH LOGS类似。

· REFRESH_TABLES

刷新表高速缓冲，与FLUSH TABLES类似。

· REFRESH_HOSTS

刷新主机高速缓冲，与FLUSH HOSTS类似。

· REFRESH_STATUS

复位状态变量，与FLUSH STATUS类似。

· REFRESH_THREADS

刷新线程高速缓冲。

· REFRESH_SLAVE

在从复制服务器上，复位主服务器信息，并重新启动从服务器，与RESET SLAVE类似。

· REFRESH_MASTER

在主复制服务器上，删除二进制日志索引中列出的二进制日志文件，并截短索引文件，与RESET MASTER类似。

返回值

0表示成功，非0值表示出现错误。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.55. mysql_reload()

int mysql_reload(MYSQL *mysql)

描述

请求MySQL服务器重新加载授权表。连接的用户必须具有RELOAD权限。

该函数已过时。最好使用mysql_query()来发出SQL FLUSH PRIVILEGES语句。

返回值

0表示成功，非0值表示出现错误。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.56. mysql_rollback()

my_bool mysql_rollback(MYSQL *mysql)

描述

回滚当前事务。

该函数的动作取决于completion_type系统变量的值。尤其是，如果completion_type的值为“2”，终结事务后，服务器将执行释放操作，并关闭客户端连接。客户端程序应调用mysql_close()，从客户端一侧关闭连接。

返回值

如果成功，返回0，如果出现错误，返回非0值。

错误

无。

25.2.3.57. mysql_row_seek()

MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET offset)

描述

将行光标置于查询结果集中的任意行。“offset”值是行偏移量，它应是从mysql_row_tell()或mysql_row_seek()返回的值。该值不是行编号，如果你打算按编号查找结果集中的行，请使用mysql_data_seek()。

该函数要求在结果集的结构中包含查询的全部结果，因此，mysql_row_seek()仅应与mysql_store_result()一起使用，而不是与mysql_use_result()。

返回值

行光标的前一个值。该值可传递给对mysql_row_seek()的后续调用。

错误

无。

25.2.3.58. mysql_row_tell()

MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result)

描述

对于上一个mysql_fetch_row()，返回行光标的当前位置。该值可用作mysql_row_seek()的参量。

仅应在mysql_store_result()之后，而不是mysql_use_result()之后使用mysql_row_tell()。

返回值

行光标的当前偏移量。

错误

无。

25.2.3.59. mysql_select_db()

int mysql_select_db(MYSQL *mysql, const char *db)

描述

使由db指定的数据库成为由mysql指定的连接上的默认数据库（当前数据库）。在后续查询中，该数据库将是未包含明确数据库区分符的表引用的默认数据库。

除非已连接的用户具有使用数据库的权限，否则mysql_select_db()将失败。

返回值

0表示成功，非0值表示出现错误。

错误

- CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.60. mysql_set_character_set()

int mysql_set_character_set(MYSQL *mysql, char *csname)

描述

该函数用于为当前连接设置默认的字符集。字符串csname指定了1个有效的字符集名称。连接校对成为字符集的默认校对。该函数的工作方式与SET NAMES语句类似，但它还能设置mysql->charset的值，从而影响了由mysql_real_escape_string()设置的字符集。

该函数是在MySQL 5.0.7中增加的。

返回值

0表示成功，非0值表示出现错误。

示例:

```
MYSQL mysql;


mysql_init(&mysql);
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));
}


if (!mysql_set_charset_name(&mysql, "utf8"))
{
    printf("New client character set: %s\n", mysql_character_set_name(&mysql));
}
```

25.2.3.61. mysql_set_server_option()

int mysql_set_server_option(MYSQL *mysql, enum enum_mysql_set_option option)

描述

允许或禁止连接的选项。选项可以取下述值之一:

MYSQL_OPTION_MULTI_STATEMENTS_ON	允许多语句支持。
MYSQL_OPTION_MULTI_STATEMENTS_OFF	禁止多语句支持。

返回值

0表示成功，非0值表示出现错误。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· ER_UNKNOWN_COM_ERROR

服务器不支持mysql_set_server_option()（当服务器版本低于4.1.1时），或服务器不支持试图设置的选项。

25.2.3.62. mysql_shutdown()

int mysql_shutdown(MYSQL *mysql, enum enum_shutdown_level shutdown_level)

描述

请求数据库服务器关闭。已连接的用户必须具有SHUTDOWN权限。MySQL 5.1服务器仅支持1种关闭类型，shutdown_level必须等效于SHUTDOWN_DEFAULT。设计规划了额外的关闭级别，以便能够选择所需的级别。对于用旧版本libmysqlclient头文件编译并调用mysql_shutdown()的动态链接可执行程序，需要与旧版的libmysqlclient动态库一起使用。

在5.5节，“MySQL服务器关机进程”中，介绍了关机进程。

返回值

0表示成功，非0值表示出现错误。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.63. mysql_sqlstate()

const char *mysql_sqlstate(MYSQL *mysql)

描述

返回由Null终结的字符串，该字符串包含关于上次错误的SQLSTATE错误代码。错误代码包含5个字符。'00000'表示无错误。其值由ANSI SQL和ODBC指定。关于可能取值的列表，请参见附录B：错误代码和消息。

注意，并非所有的MySQL错误均会被映射到SQLSTATE错误代码。值'HY000'（一般错误）用于未映射的错误。

返回值

包含SQLSTATE错误码的、由Null终结的字符串。

另请参见：

请参见25.2.3.14节，“mysql_errno()”。请参见25.2.3.15节，“mysql_error()”。请参见25.2.7.26节，“mysql_stmt_sqlstate()”。

25.2.3.64. mysql_ssl_set()

int mysql_ssl_set(MYSQL *mysql, const char *key, const char *cert, const char *ca, const char *capath, const char *cipher)

描述

使用mysql_ssl_set()，可采用SSL建立安全连接。必须在mysql_real_connect()之前调用它。

除非在客户端库中允许了OpenSSL支持，否则mysql_ssl_set()不作任何事。

Mysql是从mysql_init()返回的连接处理程序。其他参数的指定如下：

- key是key文件的路径名。
- cert是证书文件的路径名。
- ca是证书授权文件的路径名。
- capath是指向目录的路径名，该目录中包含以pem格式给出的受信任SSL CA证书。
- cipher是允许密码的列表，用于SSL加密。

对于任何未使用的SSL参数，可为其给定NULL。

返回值

该函数总返回0。如果SSL设置不正确，当你尝试连接时，mysql_real_connect()将返回错误。

25.2.3.65. mysql_stat()

char *mysql_stat(MYSQL *mysql)

描述

返回包含特定信息的字符串，该信息与**mysqladmin status**命令提供的信息类似。包括以秒为单位的正常运行时间，以及运行线程的数目，问题数，再加载次数，以及打开的表数目。

返回值

描述服务器状态的字符集。如果出现错误，返回NULL。

错误

- CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

- CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.66. mysql_store_result()

MYSQL_RES *mysql_store_result(MYSQL *mysql)

描述

对于成功检索了数据的每个查询（SELECT、SHOW、DESCRIBE、EXPLAIN、CHECK TABLE等），必须调用mysql_store_result()或mysql_use_result()。

对于其他查询，不需要调用mysql_store_result()或mysql_use_result()，但是如果在任何情况下均调用了mysql_store_result()，它也不会导致任何伤害或性能降低。通过检查mysql_store_result()是否返回0，可检测查询是否没有结果集（以后会更多）。

如果希望了解查询是否应返回结果集，可使用mysql_field_count()进行检查。请参见25.2.3.22节，“mysql_field_count()”。

mysql_store_result()将查询的全部结果读取到客户端，分配1个MYSQL_RES结构，并将结果置于该结构中。

如果查询未返回结果集，mysql_store_result()将返回Null指针（例如，如果查询是INSERT语句）。

如果读取结果集失败，mysql_store_result()还会返回Null指针。通过检查mysql_error()是否返回非空字符串，mysql_errno()是否返回非0值，或mysql_field_count()是否返回0，可以检查是否出现了错误。

如果未返回行，将返回空的结果集。（空结果集设置不同于作为返回值的空指针）。

一旦调用了mysql_store_result()并获得了不是Null指针的结果，可调用mysql_num_rows()来找出结果集中的行数。

可以调用mysql_fetch_row()来获取结果集中的行，或调用mysql_row_seek()和mysql_row_tell()来获取或设置结果集中的当前行位置。

一旦完成了对结果集的操作，必须调用mysql_free_result()。

请参见25.2.13.1节，“为什么在mysql_query()返回成功后，mysql_store_result()有时会返回NULL”.

返回值

具有多个结果的MYSQL_RES结果集合。如果出现错误，返回NULL。

错误

如果成功，mysql_store_result()将复位mysql_error()和mysql_errno()。

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_OUT_OF_MEMORY

内存溢出。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.67. mysql_thread_id()

unsigned long mysql_thread_id(MYSQL *mysql)

描述

返回当前连接的线程ID。该值可用作mysql_kill()的参量以杀死线程。

如果连接丢失，并使用mysql_ping()进行了再连接，线程ID将改变。这意味着你不应获取线程ID并保存它供以后使用。应在需要时获取它。

返回值

当前连接的线程ID。

错误

无。

25.2.3.68. mysql_use_result()

MYSQL_RES *mysql_use_result(MYSQL *mysql)

描述

对于成功检索数据的每个查询（SELECT、SHOW、DESCRIBE、EXPLAIN），必须调用mysql_store_result()或mysql_use_result()。

mysql_use_result()将初始化结果集检索，但并不像mysql_store_result()那样将结果集实际读取到客户端。它必须通过对mysql_fetch_row()的调用，对每一行分别进行检索。这将直接从服务器读取结果，而不会将其保存在临时表或本地缓冲区内，与mysql_store_result()相比，速度更快而且使用的内存也更少。客户端仅为当前行和通信缓冲区分配内存，分配的内存可增加到max_allowed_packet字节。

另一方面，如果你正在客户端一侧为各行进行大量的处理操作，或者将输出发送到了用户可能会键入"^S"（停止滚动）的屏幕，就不应使用mysql_use_result()。这会绑定服务器，并阻止其他线程更新任何表（数据从这类表获得）。

使用mysql_use_result()时，必须执行mysql_fetch_row()，直至返回NULL值，否则，未获取的行将作为下一个检索的一部分返回。C API给出命令不同步错误，如果忘记了执行该操作，将不能运行该命令。

不应与从mysql_use_result()返回的结果一起使用mysql_data_seek()、mysql_row_seek()、mysql_row_tell()、mysql_num_rows()或mysql_affected_rows()，也不应发出其他查询，直至mysql_use_result()完成为止。（但是，提取了所有行后，mysql_num_rows()将准确返回提取的行数）。

一旦完成了对结果集的操作，必须调用mysql_free_result()。

使用libmysqld嵌入式服务器时，由于在调用mysql_free_result()之前，内存使用将随着每个检索的行增加，内存效益将基本丧失。

返回值

MYSQL_RES结果结构。如果出现错误，返回NULL。

错误

如果成功，mysql_use_result()将复位mysql_error()和mysql_errno()。

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_OUT_OF_MEMORY

内存溢出。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.3.69. mysql_warning_count()

unsigned int mysql_warning_count(MYSQL *mysql)

错误

返回执行前一个SQL语句期间生成的告警数目。

返回值

告警计数。

错误

无。

25.2.4. C API预处理语句

MySQL客户端／服务器协议提供了预处理语句。该功能采用了由mysql_stmt_init()初始化函数返回的MYSQL_STMT语句处理程序数据结构。对于多次执行的语句，预处理执行是一种有效的方式。首先对语句进行解析，为执行作好准备。接下来，在以后使用初始化函数返回的语句句柄执行一次或多次。

对于多次执行的语句，预处理执行比直接执行快，主要原因在于，仅对查询执行一次解析操作。在直接执行的情况下，每次执行语句时，均将进行查询。此外，由于每次执行预处理语句时仅需发送参数的数据，从而减少了网络通信量。

预处理语句的另一个优点是，它采用了二进制协议，从而使得客户端和服务器之间的数据传输更有效率。

下述语句可用作预处理语句：CREATE TABLE、DELETE、DO、INSERT、REPLACE、SELECT、SET、UPDATE、以及大多数SHOW语句。在MySQL 5.1中，不支持其他语句。

25.2.5. C API预处理语句的数据类型

预处理语句主要使用MYSQL_STMT和MYSQL_BIND数据结构。第3种结构MYSQL_TIME用于传输暂时性数据。

· MYSQL_STMT

该结构表示预处理语句。通过调用mysql_stmt_init()创建语句，返回语句句柄，即指向MYSQL_STMT的指针。该句柄用户所有后续的与语句有关的函数，直至使用mysql_stmt_close()关闭了它为止。

MYSQL_STMT结构没有供应用程序使用的参数。此外，不应尝试复制MYSQL_STMT结构。不保证这类复制物会有用。

多个语句句柄能够与单个连接关联起来。对句柄数目的限制取决于系统资源。

· MYSQL_BIND

该结构用于语句输入（发送给服务器的数据值）和输出（从服务器返回的结果值）。对于输入，它与mysql_stmt_bind_param()一起使用，用于将参数数据值绑定到缓冲区上，以供mysql_stmt_execute()使用。对于输出，它与mysql_stmt_bind_result()一起使用，用于绑定结果缓冲区，以便用于with mysql_stmt_fetch()以获取行。

MYSQL_BIND结构包含下述供应用程序使用的成员。每个成员用于输入和输出，但在某些时候，也能用于不同的目的，具体情况取决于数据传输的方向。

o enum enum_field_types buffer_type

缓冲的类型。在本节后面列出了允许的buffer_type值。对于输入，buffer_type指明了与语句参数捆绑的值类型。对于输出，它指明了你希望从结果缓冲收到的值类型。

o void *buffer

对于输入，这是指向存储语句参数数据值的缓冲的指针。对于输出，它是指向返回结果集列值的缓冲的指针。对于数值列类型，缓冲应指向恰当的C类型变量（如果将该变量与具有UNSIGNED属性的列关联起来，变量unsigned C类型。通过使用is_unsigned成员，指明变量是signed或unsigned类型，详情请参见本节后面的介绍）。对于日期和时间列类型，缓冲应指向MYSQL_TIME结构。对于字符和二进制字符串列类型，缓冲应指向字符缓冲区。

o unsigned long buffer_length

*buffer的实际大小，单位为字节。它指明了可保存在缓冲区内的最大数据。对于字符和二进制C数据，buffer_length值指定了与mysql_stmt_bind_param()一起使用时的*buffer长度，或与mysql_stmt_bind_result()一起使用时能够提取到缓冲区内的最大数据。

o unsigned long *length

指向unsigned long变量的指针，该变量指明了存储在*buffer中数据的实际字节数。“length”用于字符或二进制C数据。对于输入参数数据绑定，“length”指向unsigned long变量，该变量指明了存储在*buffer中参数值的长度，供mysql_stmt_execute()使用。对于输出值绑定，mysql_stmt_fetch()会将返回的列值保存到“length”指向的变量中。

对于数值和临时数据类型，“length”将被忽略，原因在于，数据值的长度是由buffer_type值决定的。

o my_bool *is_null

该成员指向my_bool变量，如果值为NULL，该变量为“真”，如果值为非Null，该变量为“假”。对于输入，将*is_null设置为“真”，指明以语句参数的形式传递NULL值。对于输出，如果从语句返回的结果集列值为NULL，当获取了行后，该值将被设为“真”。

“is_null”是指向布尔类型的指针，而不是布尔标量，以便能以下述方式使用它：

§ 如果数据值总是NULL，使用MYSQL_TYPE_NULL绑定列。

§ 如果数据值总是NOT NULL，设置is_null = (my_bool*) 0。

§ 在所有其他情况下，应将is_null设置为my_bool变量的地址，并在各次执行之间恰当地更改变量的值，以指明数据值是NULL或NOT NULL。

o my_bool is_unsigned

该成员用于整数类型。（对应于MYSQL_TYPE_TINY、MYSQL_TYPE_SHORT、MYSQL_TYPE_LONG、以及MYSQL_TYPE_LONGLONG类型的代码）。对于无符号类型，应将“is_unsigned”设置为“真”，对于带符号类型，应将其设置为“假”。

o my_bool error

对于输出，该成员用于通报数据截短错误。必须通过调用带有MYSQL_REPORT_DATA_TRUNCATION选项的mysql_options()，启用截短通报功能。允许该功能后，mysql_stmt_fetch()返回MYSQL_DATA_TRUNCATED，而且对于出现截短情况的参数，在MYSQL_BIND结构中，错误标志为“真”。截短指明丢失了符号或有效位数，或字符串过长以至于无法容纳在1列中。

要想使用MYSQL_BIND结构，应将其内容置为0以便初始化它，然后对其进行设置，恰当地描述它。例如，要想声明并初始化三个MYSQL_BIND结构的数组，可使用下述代码：

```
MYSQL_BIND    bind[3];

memset(bind, 0, sizeof(bind));
```

· MYSQL_TIME

该结构用于将DATE、TIME、DATETIME和TIMESTAMP数据直接发送到服务器，或从服务器直接接收这类数据。将MYSQL_BIND结构的buffer_type成员设置为临时值之一，并将buffer成员设置为指向MYSQL_TIME结构，即可实现该点。

MYSQL_TIME结构包含下述成员：

o unsigned int year

年份

o unsigned int month

月份

o unsigned int day

天

o unsigned int hour

小时

o unsigned int minute

分钟

o unsigned int second

秒

o my_bool neg

布尔标志，用于指明时间是否为负数。

o unsigned long second_part

秒的分数部分。该成员目前不使用。

仅使用施加在给定临时类型值上的MYSQL_TIME结构的部分：用于DATE、DATETIME和TIMESTAMP的年、月、日部分。用于TIME、DATETIME和TIMESTAMP值的小时、分钟、秒部分。请参见25.2.10节，“日期和时间值的C API处理”。

在下面的表格中，给出了可在MYSQL_BIND结构的buffer_type成员中指定的允许值。在该表中，还给出了与每个buffer_type值最接近的对应SQL类型，对于数值和临时类型，给出了对应的C类型。

buffer_type值	**SQL**类型	**C**类型
MYSQL_TYPE_TINY	TINYINT	char
MYSQL_TYPE_SHORT	SMALLINT	short int
MYSQL_TYPE_LONG	INT	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	

隐式类型转换可沿两个方向执行。

25.2.6. C API预处理语句函数概述

在此归纳了预处理语句处理功能可使用的函数，并在后面的章节中详细介绍了它。请参见25.2.7节，“C API预处理语句函数描述”。

函数	描述
mysql_stmt_affected_rows()	返回由预处理语句UPDATE、DELETE或INSERT变更、删除或插入的行数目。
mysql_stmt_attr_get()	获取预处理语句属性的值。
mysql_stmt_attr_set()	设置预处理语句的属性。
mysql_stmt_bind_param()	将应用程序数据缓冲与预处理SQL语句中的参数标记符关联起来。
mysql_stmt_bind_result()	将应用程序数据缓冲与结果集中的列关联起来。
mysql_stmt_close()	释放预处理语句使用的内存。
mysql_stmt_data_seek()	寻找语句结果集中的任意行编号。
mysql_stmt_errno()	返回上次语句执行的错误编号。
mysql_stmt_error()	返回上次语句执行的错误消息。
mysql_stmt_execute()	执行预处理语句。
mysql_stmt_fetch()	从结果集获取数据的下一行，并返回所有绑定列的数据。
mysql_stmt_fetch_column()	获取结果集当前行中某列的数据。
mysql_stmt_field_count()	对于最近的语句，返回结果行的数目。
mysql_stmt_free_result()	释放分配给语句句柄的资源。
mysql_stmt_init()	为MYSQL_STMT结构分配内存并初始化它。
mysql_stmt_insert_id()	对于预处理语句的AUTO_INCREMENT列，返回生成的ID。
mysql_stmt_num_rows()	从语句缓冲结果集返回总行数。
mysql_stmt_param_count()	返回预处理SQL语句中的参数数目。
mysql_stmt_param_metadata()	返回结果集的参数元数据。
mysql_stmt_prepare()	为执行操作准备SQL字符串。
mysql_stmt_reset()	复位服务器中的语句缓冲区。
mysql_stmt_result_metadata()	以结果集形式返回预处理语句元数据。
mysql_stmt_row_seek()	使用从mysql_stmt_row_tell()返回的值，查找语句结果集中的行偏移。
mysql_stmt_row_tell()	返回语句行光标位置。
mysql_stmt_send_long_data()	将程序块中的长数据发送到服务器。
mysql_stmt_sqlstate()	返回关于上次语句执行的SQLSTATE错误代码。
mysql_stmt_store_result()	将完整的结果集检索到客户端。

调用mysql_stmt_init()以创建语句句柄，然后调用mysql_stmt_prepare准备语句，调用mysql_stmt_bind_param()提供参数数据，并调用mysql_stmt_execute()执行语句。通过更改mysql_stmt_bind_param()提供的相应缓冲区中的参数值，可重复执行mysql_stmt_execute()。

如果语句是SELECT或任何其他能生成结果集的语句，mysql_stmt_prepare()也会通过mysql_stmt_result_metadata()以MYSQL_RES结果集的形式返回结果集元数据信息。

你可以使用mysql_stmt_bind_result()提供结果缓冲，以便mysql_stmt_fetch()能自动将数据返回给这些缓冲。这是一种按行获取方式。

此外，你也能使用mysql_stmt_send_long_data()将程序块中的文本或二进制数据发送到服务器。请参见25.2.7.25节，“mysql_stmt_send_long_data()”。

完成语句执行后，必须使用mysql_stmt_close()关闭语句句柄，以便与之相关的所有资源均能被释放。

如果通过调用mysql_stmt_result_metadata()获得了SELECT语句的结果集元数据，也应使用mysql_free_result()释放元数据。

执行步骤

要想准备和执行语句，应用程序必须采取下述步骤：

1. 用msyql_stmt_init()创建预处理语句句柄。要想在服务器上准备预处理语句，可调用mysql_stmt_prepare()，并为其传递包含SQL语句的字符串。

2. 如果语句生成了结果集，调用mysql_stmt_result_metadata()以获得结果集元数据。虽然与包含查询返回列的结果集不同，该元数据本身也采用了结果集的形式。元数据结果集指明了结果中包含多少列，并包含每一列的信息。

3. 使用mysql_stmt_bind_param()设置任何参数的值。必须设置所有参数。否则，语句执行将返回错误，或生成无法预料的结果。

4. 调用mysql_stmt_execute()执行语句。

5. 如果语句生成了结果集，捆绑数据缓冲，通过调用mysql_stmt_bind_result()，检索行值。

6. 通过重复调用mysql_stmt_fetch()，按行将数据提取到缓冲区，直至未发现更多行为止。

7. 通过更改参数值并再次执行语句，重复步骤3到步骤6。

调用mysql_stmt_prepare()时，MySQL客户端／服务器协议将执行下述动作：

· 服务器解析语句，并通过赋值语句ID将OK状态发回客户端。此外，如果它是面向结果集的语句，还将发送总的参数数目，列计数和元数据。在此调用过程中，服务器将检查语句的所有语法和语义。

· 客户端采用该语句ID用于进一步操作，以便服务器能从其语句池中识别语句。

调用mysql_stmt_execute()时，MySQL客户端／服务器协议将执行下述动作：

· 客户端使用语句句柄，并将参数数据发送到服务器。

· 服务器使用由客户端提供的ID来识别语句，用新提供的数据替换参数标记符，并执行语句。如果语句生成了结果集，服务器将数据发回客户端。否则，服务器会将发送OK状态，以及总的变更、删除和插入行数。

调用mysql_stmt_fetch()时，MySQL客户端／服务器协议将执行下述动作：

· 客户端按行从信息包读取数据，并通过执行必要的转换操作将其放入应用程序数据缓冲中。如果应用程序的缓冲类型与服务器返回的字段类型相同，转换十分简明。

如果出现了错误，可分别使用mysql_stmt_errno()、mysql_stmt_error()和mysql_stmt_sqlstate()获取语句错误代码、错误消息和SQLSTATE值。

预处理语句日志功能

对于与mysql_stmt_prepare()和mysql_stmt_execute() C API函数一起执行的预处理语句，服务器会将"准备"和"执行"行写入一般查询日志，以便你能了解语句是在何时准备和执行的。

假定按下述方式准备和执行了语句：

1. 调用mysql_stmt_prepare()以准备语句字符串"SELECT ?"。

2. 调用mysql_stmt_bind_param()将值"3"绑定到预处理语句中的参数。

3. 调用mysql_stmt_execute()，执行预处理语句。

上述调用的结果是，服务器将下述行写入一般查询日志：

```
Prepare  [1] SELECT ?
Execute  [1] SELECT 3
```

日志中的每个"准备"和"执行"行均具有[*n*]语句ID标识，这样，你就能跟踪已记录的预处理语句。*N*是正整数。对于客户端，如

果同时有多个活动的预处理语句，*n*可能会大于1。替换了"?"参数的数据值后，每个"执行"行将显示一条预处理语句。

版本说明：在MySQL 4.1.10之前，显示的"准备"行无[*n*]标识。在MySQL 4.1.10之前，不显示"执行"行。

25.2.7. C API预处理语句函数描述

25.2.7.1. mysql_stmt_affected_rows()
25.2.7.2. mysql_stmt_attr_get()
25.2.7.3. mysql_stmt_attr_set()
25.2.7.4. mysql_stmt_bind_param()
25.2.7.5. mysql_stmt_bind_result()
25.2.7.6. mysql_stmt_close()
25.2.7.7. mysql_stmt_data_seek()
25.2.7.8. mysql_stmt_errno()
25.2.7.9. mysql_stmt_error()
25.2.7.10. mysql_stmt_execute()
25.2.7.11. mysql_stmt_fetch()
25.2.7.12. mysql_stmt_fetch_column()
25.2.7.13. mysql_stmt_field_count()
25.2.7.14. mysql_stmt_free_result()
25.2.7.15. mysql_stmt_init()
25.2.7.16. mysql_stmt_insert_id()
25.2.7.17. mysql_stmt_num_rows()
25.2.7.18. mysql_stmt_param_count()
25.2.7.19. mysql_stmt_param_metadata()
25.2.7.20. mysql_stmt_prepare()
25.2.7.21. mysql_stmt_reset()
25.2.7.22. mysql_stmt_result_metadata()
25.2.7.23. mysql_stmt_row_seek()
25.2.7.24. mysql_stmt_row_tell()
25.2.7.25. mysql_stmt_send_long_data()
25.2.7.26. mysql_stmt_sqlstate()
25.2.7.27. mysql_stmt_store_result()

为了准备和执行查询，请使用下述部分详细介绍的函数。

注意，与MYSQL_STMT结构一起使用的所有函数均以前缀mysql_stmt_开始。

要想创建MYSQL_STMT句柄，请使用mysql_stmt_init()函数。

25.2.7.1. mysql_stmt_affected_rows()

my_ulonglong mysql_stmt_affected_rows(MYSQL_STMT *stmt)

描述

返回上次执行语句更改、删除或插入的总行数。对于UPDATE、DELETE或INSERT语句，可在mysql_stmt_execute()之后立刻调用它们。对于SELECT语句，mysql_stmt_affected_rows()的工作方式类似于mysql_num_rows()。

返回值

大于0的整数指明了受影响或检索的行数。对于UPDATE语句，"0"表明未更新任何记录，在查询中没有与WHERE子句匹配的行，或尚未执行任何查询。"-1"表明返回了错误，或对SELECT查询，在调用mysql_stmt_store_result()之前调用了mysql_stmt_affected_rows()。由于mysql_stmt_affected_rows()返回无符号值，可通过比较返回值和"(my_ulonglong)-1"（或等效的"(my_ulonglong)~0"），检查"-1"。

关于返回值的额外信息，请参见25.2.3.1节，"mysql_affected_rows()"。

错误

无。

示例：

关于mysql_stmt_affected_rows()的用法，请参阅25.2.7.10节，“mysql_stmt_execute()”中给出的示例。

25.2.7.2. mysql_stmt_attr_get()

int mysql_stmt_attr_get(MYSQL_STMT *stmt, enum enum_stmt_attr_type option, void *arg)

描述

可用于获得语句属性的当前值。

“option”参量是希望获取的选项，“arg”应指向包含选项值的变量。如果“option”是整数，那么“arg”应指向整数的值。

关于选项和选项类型的清单，请参见25.2.7.3节，“mysql_stmt_attr_set()”。

返回值

如果OK，返回0。如果选项未知，返回非0值。

错误

无。

25.2.7.3. mysql_stmt_attr_set()

int mysql_stmt_attr_set(MYSQL_STMT *stmt, enum enum_stmt_attr_type option, const void *arg)

描述

可用于影响预处理语句的行为。可多次调用该函数来设置多个选项。

“option”参量是希望设置的选项，“arg”参量是选项的值。如果“option”是整数，那么“arg”应指向整数的值。

可能的选项值:

选项	参量类型	功能
STMT_ATTR_UPDATE_MAX_LENGTH	my_bool *	如果设为1：更新mysql_stmt_store_result()中的元数据MYSQL_FIELD->max_length。
STMT_ATTR_CURSOR_TYPE	unsigned long *	调用mysql_stmt_execute()时，语句将打开的光标类型。*arg可以是CURSOR_TYPE_NO_CURSOR（默认值）或CURSOR_TYPE_READ_ONLY。
STMT_ATTR_PREFETCH_ROWS	unsigned long *	使用光标时，一次从服务器获取的行数。*arg的范围从1到unsigned long的最大值。默认值为1。

如果与CURSOR_TYPE_READ_ONLY一起使用了STMT_ATTR_CURSOR_TYPE选项，当调用了mysql_stmt_execute()时，将为语句打开光标。如果存在由前一个mysql_stmt_execute()调用打开的光标，在打开新的光标前，将关闭该光标。此外，为再执行而准备语句之前，mysql_stmt_reset()还将关闭任何打开的光标。mysql_stmt_free_result()将关闭任何打开的光标。

如果为预处理语句打开了光标，没必要调用mysql_stmt_store_result()，这是因为，该函数会导致在客户端一侧对结果集进行缓冲处理。

在MySQL 5.0.2中增加了STMT_ATTR_CURSOR_TYPE选项。在MySQL 5.0.6中，增加了STMT_ATTR_PREFETCH_ROWS选项。

返回值

如果OK，返回0。如果选项未知，返回非0值。

错误

无。

示例:

在下述示例中，为预处理语句打开了1个光标，并将每次获取的行数设为5：

```
MYSQL_STMT *stmt;
int rc;
unsigned long type;
unsigned long prefetch_rows = 5;

stmt = mysql_stmt_init(mysql);
type = (unsigned long) CURSOR_TYPE_READ_ONLY;
rc = mysql_stmt_attr_set(stmt, STMT_ATTR_CURSOR_TYPE, (void*) &type);
/* ... check return value ... */
rc = mysql_stmt_attr_set(stmt, STMT_ATTR_PREFETCH_ROWS,
                                        (void*) &prefetch_rows);
/* ... check return value ... */
```

25.2.7.4. mysql_stmt_bind_param()

my_bool mysql_stmt_bind_param(MYSQL_STMT *stmt, MYSQL_BIND *bind)

描述

mysql_stmt_bind_param()用于为SQL语句中的参数标记符绑定数据，以传递给mysql_stmt_prepare()。它使用MYSQL_BIND结构来提供数据。“bind”是MYSQL_BIND结构的某一数组的地址。按照客户端库的预期，对于查询中出现的每个“?”参数标记符，数组中均包含1个元素。

假定你准备了下述语句：

```
INSERT INTO mytbl VALUES(?,?,?)
```

绑定参数时，MYSQL_BIND结构的数组包含3个元素，并能声明如下：

```
MYSQL_BIND bind[3];
```

在25.2.5节，“C API预处理语句的数据类型”中，介绍了应设置的每个MYSQL_BIND元素的成员。

返回值

如果绑定成功，返回0。如果出现错误，返回非0值。

错误

- CR_INVALID_BUFFER_USE

指明“bind”（绑定）是否将提供程序块中的长数据，以及缓冲类型是否为非字符串或二进制类型。

- CR_UNSUPPORTED_PARAM_TYPE

不支持该转换。或许buffer_type值是非法的，或不是所支持的类型之一。

- CR_OUT_OF_MEMORY

内存溢出。

- CR_UNKNOWN_ERROR

出现未知错误。

示例：

关于mysql_stmt_bind_param()的用法，请参见25.2.7.10节，“mysql_stmt_execute()”给出的示例。

25.2.7.5. mysql_stmt_bind_result()

my_bool mysql_stmt_bind_result(MYSQL_STMT *stmt, MYSQL_BIND *bind)

描述

mysql_stmt_bind_result()用于将结果集中的列与数据缓冲和长度缓冲关联（绑定）起来。当调用mysql_stmt_fetch()以获取数据时，MySQL客户端／服务器协议会将绑定列的数据置于指定的缓冲区内。

调用mysql_stmt_fetch()之前，必须将所有列绑定到缓冲。"bind"是MYSQL_BIND结构某一数组的地址。按照客户端库的预期，对于结果集中的每一列，数组应包含相应的元素。如果未将列绑定到MYSQL_BIND结构，mysql_stmt_fetch()将简单地忽略数据获取操作。缓冲区应足够大，足以容纳数据值，这是因为协议不返回成块的数据值。

可以在任何时候绑定或再绑定列，即使已部分检索了结果集后也同样。新的绑定将在下一次调用mysql_stmt_fetch()时起作用。假定某一应用程序绑定了结果集中的列，并调用了mysql_stmt_fetch()。客户端／服务器协议将返回绑定缓冲区中的数据。接下来，假定应用程序将多个列绑定到不同的缓冲。该协议不会将数据置于新绑定的缓冲区，直至下次调用mysql_stmt_fetch()为止。

要想绑定列，应用程序将调用mysql_stmt_bind_result()，并传递类型、地址、以及长度缓冲的地址。在25.2.5节，"C API预处理语句的数据类型"中，介绍了应设置的各MYSQL_BIND元素的成员。

返回值

如果绑定成功，返回0。如果出现错误，返回非0值。

错误

- CR_UNSUPPORTED_PARAM_TYPE

不支持该转换。或许buffer_type值是非法的，或不是所支持的类型之一。

- CR_OUT_OF_MEMORY

内存溢出。

- CR_UNKNOWN_ERROR

出现未知错误。

示例:

关于mysql_stmt_bind_result()的用法，请参见25.2.7.11节，"mysql_stmt_fetch()"中给出的示例。

25.2.7.6. mysql_stmt_close()

my_bool mysql_stmt_close(MYSQL_STMT *)

描述

关闭预处理语句。此外，mysql_stmt_close()还会取消由"stmt"指向的语句句柄分配。

如果当前语句已挂起或未读取结果，该函数将取消它们，以便能执行下一个查询，

返回值

如果成功释放了语句，返回0。如果出现错误，返回非0值。

错误

- CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_UNKNOWN_ERROR

出现未知错误。

示例:

关于mysql_stmt_close()的用法，请参见25.2.7.10节，“mysql_stmt_execute()”中给出的示例。

25.2.7.7. mysql_stmt_data_seek()

void mysql_stmt_data_seek(MYSQL_STMT *stmt, my_ulonglong offset)

描述

查找语句结果集中的任意行。偏移量为行编号，应位于从0到mysql_stmt_num_rows(stmt)-1的范围内。

该函数要求语句结果集结构包含上次执行查询的全部结果，这样，mysql_stmt_data_seek()就能与mysql_stmt_store_result()一起使用。

返回值

无。

错误

无。

25.2.7.8. mysql_stmt_errno()

unsigned int mysql_stmt_errno(MYSQL_STMT *stmt)

描述

对于由stmt指定的语句，mysql_stmt_errno()将返回最近调用的语句API函数的错误代码，该函数或成功或失败。“0”返回值表示未出现错误。在MySQL errmsg.h头文件中列出了客户端错误消息编号。在mysqld_error.h中，列出了服务器错误消息。此外，在附录B：错误代码和消息中，也列出了错误消息。

返回值

错误代码值。如果未出现错误，返回0。

错误

无。

25.2.7.9. mysql_stmt_error()

const char *mysql_stmt_error(MYSQL_STMT *stmt)

描述

对于由stmt指定的语句，mysql_stmt_error()返回由Null终结的字符串，该字符串包含最近调用的语句API函数的错误消息，该函数或成功或失败。如果未出现错误，返回空字符串("")。这意味着下述两个测试是等效的:

```
if (mysql_stmt_errno(stmt))
{
  // an error occurred
```

```
}

if (mysql_stmt_error(stmt)[0])
{
  // an error occurred
}
```

通过重新编译MySQL客户端库，可更改客户端错误消息的语言。目前，能够选择数种语言之一显示错误消息。

返回值

描述了错误的字符串。如果未出现错误，返回空字符串。

错误

无。

25.2.7.10. mysql_stmt_execute()

int mysql_stmt_execute(MYSQL_STMT *stmt)

描述

mysql_stmt_execute()执行与语句句柄相关的预处理查询。在该调用期间，将当前绑定的参数标记符的值发送到服务器，服务器用新提供的数据替换标记符。

如果语句是UPDATE、DELETE或INSERT，通过调用mysql_stmt_affected_rows()，可发现更改、删除或插入的总行数。如果这是诸如SELECT等能生成结果集的语句，调用任何其他能导致查询处理的函数之前，必须调用mysql_stmt_fetch()来获取数据。关于如何获取结果的更多信息，请参见25.2.7.11节，“mysql_stmt_fetch()”。

对于生成结果集的语句，执行语句之前，可通过调用mysql_stmt_attr_set()，请求mysql_stmt_execute()为语句打开光标。如果多次执行某一语句，在打开新的光标前，mysql_stmt_execute()将关闭任何已打开的光标。

返回值

如果执行成功，返回0。如果出现错误，返回非0值。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_OUT_OF_MEMORY

内存溢出。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

示例:

在下面的示例中，介绍了使用mysql_stmt_init()、mysql_stmt_prepare()、mysql_stmt_param_count()、mysql_stmt_bind_param()、mysql_stmt_execute()、以及mysql_stmt_affected_rows()创建和填充表的方法。假定mysql变量具有有效的连接句柄。

```
#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; the server */
/*  sets it to the current date and time) */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
```

```
if (mysql_stmt_prepare(stmt, INSERT_SAMPLE, strlen(INSERT_SAMPLE)))
{
  fprintf(stderr, " mysql_stmt_prepare(), INSERT failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
fprintf(stdout, " prepare, INSERT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_stmt_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 */

memset(bind, 0, sizeof(bind));

/* 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_stmt_bind_param(stmt, bind))
{
```

```
  fprintf(stderr, " mysql_stmt_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_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_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): %lu\n",
                (unsigned long) 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_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_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): %lu\n",
                (unsigned long) 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);
}
```

注释：关于使用预处理语句函数的完整示例，请参见文件tests/mysql_client_test.c。该文件可从MySQL源码分发版获得，或从BitKeeper源码仓库获得。

25.2.7.11. mysql_stmt_fetch()

int mysql_stmt_fetch(MYSQL_STMT *stmt)

描述

mysql_stmt_fetch()返回结果集中的下一行。仅能当结果集存在时调用它，也就是说，调用了能创建结果集的mysql_stmt_execute()之后，或当mysql_stmt_execute()对整个结果集即行缓冲处理后调用了mysql_stmt_store_result()。

使用mysql_stmt_bind_result()绑定的缓冲，mysql_stmt_fetch()返回行数据。对于当前列集合中的所有列，它将返回缓冲内的数据，并将长度返回到长度指针。

调用mysql_stmt_fetch()之前，应用程序必须绑定所有列。

如果获取的数据值是NULL值，对应MYSQL_BIND结构的*is_null值将包含TRUE (1)。否则，将根据应用程序指定的缓冲类型，在*buffer和*length内返回数据及其长度。每个数值类型和临时类型都有固定的长度，请参见下面的表格。字符串类型的长度取决于由data_length指明的实际数据值的长度。

类型	长度
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

返回值

返回值	描述
0	成功，数据被提取到应用程序数据缓冲区。
1	出现错误。通过调用mysql_stmt_errno()和mysql_stmt_error()，可获取错误代码和错误消息。
MYSQL_NO_DATA	不存在行／数据。
MYSQL_DATA_TRUNCATED	出现数据截短。

不返回MYSQL_DATA_TRUNCATED，除非用mysql_options()启用了截短通报功能。返回该值时，为了确定截短的参数是哪个，可检查MYSQL_BIND参数结构的错误成员。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_OUT_OF_MEMORY

内存溢出。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

· CR_UNSUPPORTED_PARAM_TYPE

缓冲类型为MYSQL_TYPE_DATE、MYSQL_TYPE_TIME、MYSQL_TYPE_DATETIME、或MYSQL_TYPE_TIMESTAMP，但数据类型不是DATE、TIME、DATETIME、或TIMESTAMP。

· 从mysql_stmt_bind_result()返回所有其他不支持的转换错误。

示例:

在下面的示例中，介绍了使用mysql_stmt_result_metadata()、mysql_stmt_bind_result()和mysql_stmt_fetch()从表中获取数据的方法。（在本示例中，将检索在25.2.7.10节，“mysql_stmt_execute()”一节的示例中插入的两行内容）。假定mysql变量具有有效的连接句柄。

```
#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_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(stmt, SELECT_SAMPLE, strlen(SELECT_SAMPLE)))
{
  fprintf(stderr, " mysql_stmt_prepare(), SELECT failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
fprintf(stdout, " prepare, SELECT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_stmt_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_stmt_result_metadata(stmt);
if (!prepare_meta_result)
{
  fprintf(stderr,
         " mysql_stmt_result_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_stmt_execute(stmt))
  {
    fprintf(stderr, " mysql_stmt_execute(), failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
  }

  /* Bind the result buffers for all 4 columns before fetching them */

  memset(bind, 0, sizeof(bind));

  /* 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_stmt_bind_result(stmt, bind))
  {
    fprintf(stderr, " mysql_stmt_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_stmt_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);
}
```

25.2.7.12. mysql_stmt_fetch_column()

int mysql_stmt_fetch_column(MYSQL_STMT *stmt, MYSQL_BIND *bind, unsigned int column, unsigned long offset)

描述

从当前结果集行获取1列。“bind”提供了应将数据置于其中的缓冲。其设置方法应与设置mysql_stmt_bind_result()的相同。“column”指明了将获取哪个列。第1列编号为0。“offset”是数据值内的偏移量，将从该处开始检索数据。可将其用于获取碎片形式的数据值。值开始部分的偏移量为0。

返回值

如果成功获取了值，返回0。如果出现错误，返回非0值。

错误

- CR_INVALID_PARAMETER_NO

Invalid column number.

- CR_NO_DATA

已抵达结果集的末尾。

25.2.7.13. mysql_stmt_field_count()

unsigned int mysql_stmt_field_count(MYSQL_STMT *stmt)

描述

为语句处理程序返回关于最近语句的行数。对于诸如INSERT或DELETE等不生成结果集的语句，该值为0。

通过调用mysql_stmt_prepare()准备好了语句后，可调用mysql_stmt_field_count()。

返回值

表示结果集中行数的无符号整数。

错误

无。

25.2.7.14. mysql_stmt_free_result()

my_bool mysql_stmt_free_result(MYSQL_STMT *stmt)

描述

释放与执行预处理语句生成的结果集有关的内存。对于该语句，如果存在打开的光标，mysql_stmt_free_result()将关闭它。

返回值

如果成功释放了结果集，返回0。如果出现错误，返回非0值。

错误

25.2.7.15. mysql_stmt_init()

MYSQL_STMT *mysql_stmt_init(MYSQL *mysql)

描述

创建MYSQL_STMT句柄。对于该句柄，应使用mysql_stmt_close(MYSQL_STMT *)释放。

返回值

成功时，返回指向MYSQL_STMT结构的指针。如果内存溢出，返回NULL。

错误

- CR_OUT_OF_MEMORY

内存溢出。

25.2.7.16. mysql_stmt_insert_id()

my_ulonglong mysql_stmt_insert_id(MYSQL_STMT *stmt)

描述

返回预处理INSERT或UPDATE语句为AUTO_INCREMENT列生成的值。在包含AUTO_INCREMENT字段的表上执行了预处理INSERT语句后，使用该函数。

更多信息，请参见25.2.3.36节，“mysql_insert_id()”。

返回值

为在执行预处理语句期间自动生成或明确设置的AUTO_INCREMENT列返回值，或由LAST_INSERT_ID(*expr*)函数生成的值。如果语句未设置AUTO_INCREMENT值，返回值不确定。

错误

无。

25.2.7.17. mysql_stmt_num_rows()

my_ulonglong mysql_stmt_num_rows(MYSQL_STMT *stmt)

描述

返回结果集中的行数。

mysql_stmt_num_rows()的用法取决于是否使用了mysql_stmt_store_result()来对语句句柄中的全部结果集进行了缓冲处理。

如果使用了mysql_stmt_store_result()，可立刻调用mysql_stmt_num_rows()。

返回值

结果集中的行数。

错误

无。

25.2.7.18. mysql_stmt_param_count()

unsigned long mysql_stmt_param_count(MYSQL_STMT *stmt)

描述

返回预处理语句中参数标记符的数目。

返回值

表示语句中参数数目的无符号长整数。

错误

无。

示例:

关于mysql_stmt_param_count()的用法，请参见25.2.7.10节，“mysql_stmt_execute()”中给出的示例。

25.2.7.19. mysql_stmt_param_metadata()

MYSQL_RES *mysql_stmt_param_metadata(MYSQL_STMT *stmt)

该函数目前不做任何事。

描述

返回值

错误

25.2.7.20. mysql_stmt_prepare()

int mysql_stmt_prepare(MYSQL_STMT *stmt, const char *query, unsigned long length)

描述

给定mysql_stmt_init()返回的语句句柄，准备字符串查询指向的SQL语句，并返回状态值。字符串长度应由“length”参量给出。字符串必须包含1条SQL语句。不应为语句添加终结用分号(‘;’)或\g。

通过将问号字符“?”嵌入到SQL字符串的恰当位置，应用程序可包含SQL语句中的一个或多个参数标记符。

标记符仅在SQL语句中的特定位置时才是合法的。例如，它可以在INSERT语句的VALUES()列表中（为行指定列值），或与WHERE子句中某列的比较部分（用以指定比较值）。但是，对于ID（例如表名或列名），不允许使用它们，不允许指定二进制操作符（如等于号“=”）的操作数。后一个限制是有必要的，原因在于，无法确定参数类型。一般而言，参数仅在DML（数据操作语言）语句中才是合法的，在DDL（数据定义语言）语句中不合法。

执行语句之前，必须使用mysql_stmt_bind_param()，将参数标记符与应用程序变量绑定在一起。

返回值

如果成功处理了语句，返回0。如果出现错误，返回非0值。

错误

- CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

- CR_OUT_OF_MEMORY

内存溢出。

- CR_SERVER_GONE_ERROR

MySQL服务器不可用。

- CR_SERVER_LOST

查询过程中，与服务器的连接丢失。

- CR_UNKNOWN_ERROR

出现未知错误。

如果准备操作失败（即mysql_stmt_prepare()返回非0值），可通过调用mysql_stmt_error()获取错误消息。

示例:

关于mysql_stmt_prepare()的用法，请参见25.2.7.10节，“mysql_stmt_execute()”中给出的示例。

25.2.7.21. mysql_stmt_reset()

my_bool mysql_stmt_reset(MYSQL_STMT *stmt)

描述

在客户端和服务器上，将预处理语句复位为完成准备后的状态。主要用于复位用mysql_stmt_send_long_data()发出的数据。对于语句，任何已打开的光标将被关闭。

要想重新准备用于另一查询的语句，可使用mysql_stmt_prepare()。

返回值

如果语句成功复位，返回0。如果出现错误，返回非0值。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.7.22. mysql_stmt_result_metadata()

MYSQL_RES *mysql_stmt_result_metadata(MYSQL_STMT *stmt)

描述

如果传递给mysql_stmt_prepare()的语句能够成生结果集，mysql_stmt_result_metadata()将以指针的形式返回结果集元数据，该指针指向MYSQL_RES结构，可用于处理元信息，如总的字段数以及单独的字段信息。该结果集指针可作为参量传递给任何基于字段且用于处理结果集元数据的API函数，如：

· 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()

完成操作后，应释放结果集结构，可通过将其传递给mysql_free_result()完成。它与释放通过mysql_store_result()调用获得的结果集的方法类似。

mysql_stmt_result_metadata()返回的结果集仅包含元数据。不含任何行结果。与mysql_stmt_fetch()一起使用语句句柄，可获取行。

返回值

MYSQL_RES结果结构。如果不存在关于预处理查询的任何元信息，返回NULL。

错误

· CR_OUT_OF_MEMORY

内存溢出。

· CR_UNKNOWN_ERROR

出现未知错误。

示例:

关于mysql_stmt_result_metadata()的用法，请参见25.2.7.11节，“mysql_stmt_fetch()”中给出的示例。

25.2.7.23. mysql_stmt_row_seek()

MYSQL_ROW_OFFSET mysql_stmt_row_seek(MYSQL_STMT *stmt, MYSQL_ROW_OFFSET offset)

描述

将行光标设置到语句结果集中的任意行。“offset”值是行偏移的值，行偏移应是从mysql_stmt_row_tell()或mysql_stmt_row_seek()返回的值。该值不是行编号，如果打算按编号查找结果集中的行，可使用mysql_stmt_data_seek()取而代之。

该函数要求结果集结构包含查询的全部结果，以便mysql_stmt_row_seek()能够仅与mysql_stmt_store_result()一起使用。

返回值

行光标的前一个值。可以将该值换递给后续的mysql_stmt_row_seek()调用。

错误

无。

25.2.7.24. mysql_stmt_row_tell()

MYSQL_ROW_OFFSET mysql_stmt_row_tell(MYSQL_STMT *stmt)

描述

返回针对前一个mysql_stmt_fetch()的行光标的当前位置。该值可用作mysql_stmt_row_seek()的参量。

仅应在mysql_stmt_store_result()之后使用mysql_stmt_row_tell()。

返回值

行光标的当前偏移量。

错误

无。

25.2.7.25. mysql_stmt_send_long_data()

my_bool mysql_stmt_send_long_data(MYSQL_STMT *stmt, unsigned int parameter_number, const char *data, unsigned long length)

描述

允许应用程序分段地（分块）将参数数据发送到服务器。可以多次调用该函数，以便发送关于某一列的字符或二进制数据的不同部分，列必须是TEXT或BLOB数据类型之一。

“parameter_number”指明了与数据关联的参数。参数从0开始编号。“data”是指向包含将要发送的数据的缓冲区的指针，“length”指明了缓冲区内的字节数。

注释：自上一个mysql_stmt_execute()或mysql_stmt_reset()后，对于与mysql_stmt_send_long_data()一起使用的所有参数，下一个mysql_stmt_execute()调用将忽略绑定缓冲。

如果希望复位／忽略已发送的数据，可使用mysql_stmt_reset()。请参见25.2.7.21节，“mysql_stmt_reset()”。

返回值

如果成功地将数据发送到服务器，返回0。如果出现错误，返回非0值。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_OUT_OF_MEMORY

内存溢出。

· CR_UNKNOWN_ERROR

出现未知错误。

示例：

在下面的示例中，介绍了以信息块形式为TEXT列发送数据的方法。它会将数据值“MySQL，最流行的开放源码数据库”插入到text_column列中。假定mysql变量具有有效的连接句柄。

```
#define INSERT_QUERY "INSERT INTO test_long_data(text_column) VALUES(?)"


MYSQL_BIND bind[1];
long       length;


smtt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_QUERY, strlen(INSERT_QUERY)))
{
  fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  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_stmt_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_stmt_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_stmt_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_stmt_execute(stmt))
{
  fprintf(stderr, "\n mysql_stmt_execute failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}
```

25.2.7.26. mysql_stmt_sqlstate()

const char *mysql_stmt_sqlstate(MYSQL_STMT *stmt)

描述

对于由stmt指定的语句，mysql_stmt_sqlstate()返回由Null终结的字符串，该字符串包含针对最近调用预处理语句API函数的SQLSTATE错误代码，该函数或成功或失败。错误代码由5个字符构成。"00000"表示"无错误"。这些值由ANSI SQL和ODBC指定。关于可能值的列表，请参见附录B：错误代码和消息。

注意，并非所有的MySQL错误均会被映射到SQLSTATE代码。值"HY000"（一般错误）用于未映射的错误。

返回值

包含SQLSTATE错误代码、由Null终结的字符串。

25.2.7.27. mysql_stmt_store_result()

int mysql_stmt_store_result(MYSQL_STMT *stmt)

描述

对于成功生成结果集的所有语句（SELECT、SHOW、DESCRIBE、EXPLAIN），而且仅当你打算对客户端的全部结果集进行缓冲处理时，必须调用mysql_stmt_store_result()，以便后续的mysql_stmt_fetch()调用能返回缓冲数据。

对于其他语句，没有必要调用mysql_stmt_store_result()，但如果调用了它，也不会造成任何伤害或导致任何性能问题。通过检查mysql_stmt_result_metadata()是否返回NULL，可检测语句是否生成了结果集。更多信息，请参见25.2.7.22节，“mysql_stmt_result_metadata()”。

注释：默认情况下，对于mysql_stmt_store_result()中的所有列，MySQL不计算MYSQL_FIELD->max_length，这是因为，计算它会显著降低mysql_stmt_store_result()的性能，而且大多数应用程序不需要max_length。如果打算更新max_length，可通过调用mysql_stmt_attr_set(MYSQL_STMT, STMT_ATTR_UPDATE_MAX_LENGTH, &flag)启用它。请参见25.2.7.3节，“mysql_stmt_attr_set()”。

返回值

如果成功完成了对结果的缓冲处理，返回0。如果出现错误，返回非0值。

错误

· CR_COMMANDS_OUT_OF_SYNC

以不恰当的顺序执行了命令。

· CR_OUT_OF_MEMORY

内存溢出。

· CR_SERVER_GONE_ERROR

MySQL服务器不可用。

· CR_SERVER_LOST

在查询过程中，与服务器的连接丢失。

· CR_UNKNOWN_ERROR

出现未知错误。

25.2.8. C API预处理语句方面的问题

下面列出了一些目前已知的与预处理语句有关的问题：

· TIME、TIMESTAMP和DATETIME不支持秒部分，例如来自DATE_FORMAT()的秒部分。

· 将整数转换为字符串时，在某些情况下，当MySQL不打印前导0时，可与预处理语句一起使用ZEROFILL。例如，与MIN(number-with-zerofill)一起。

· 将浮点数转换为客户端中的字符串时，被转换值最右侧的位可能会与原始值的有所不同。

· 预处理语句不使用查询高速缓冲，即使当查询不含任何占位符时也同样。。请参见5.13.1节，“查询高速缓冲如何工作”。

25.2.9. 多查询执行的C API处理

MySQL 5.1支持在单个查询字符串中指定的多语句的执行。要想与给定的连接一起使用该功能，打开连接时，必须将标志参数中的CLIENT_MULTI_STATEMENTS选项指定给mysql_real_connect()。也可以通过调用mysql_set_server_option(MYSQL_OPTION_MULTI_STATEMENTS_ON)，为已有的连接设置它。

在默认情况下，mysql_query()和mysql_real_query()仅返回第1个查询的状态，并能使用mysql_more_results()和mysql_next_result()对后续查询的状态进行处理。

```
/* 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));
```

多语句功能可与mysql_query()或mysql_real_query()一起使用。它不能与预处理语句接口一起使用。按照定义，预处理语句仅能与包含单个语句的字符串一起使用。

25.2.10. 日期和时间值的C API处理

二进制协议允许你使用MYSQL_TIME结构发送和接受日期和时间值（DATE、TIME、DATETIME和TIMESTAMP）。在25.2.5节，“C API预处理语句的数据类型”中，介绍了该结构的成员。

要想发送临时数据值，可使用mysql_stmt_prepare()创建预处理语句。然后，在调用mysql_stmt_execute()执行语句之前，可采用下述步骤设置每个临时参数：

1. 在与数据值相关的MYSQL_BIND结构中，将buffer_type成员设置为相应的类型，该类型指明了发送的临时值类型。对于DATE、TIME、DATETIME或TIMESTAMP值，将buffer_type分别设置为MYSQL_TYPE_DATE、MYSQL_TYPE_TIME、MYSQL_TYPE_DATETIME或MYSQL_TYPE_TIMESTAMP。

2. 将MYSQL_BIND结构的缓冲成员设置为用于传递临时值的MYSQL_TIME结构的地址。

3. 填充MYSQL_TIME结构的成员，使之与打算传递的临时支的类型相符。

使用mysql_stmt_bind_param()将参数数据绑定到语句。然后可调用mysql_stmt_execute()。

要想检索临时值，可采用类似的步骤，但应将buffer_type成员设置为打算接受的值的类型，并将缓冲成员设为应将返回值置于其中的MYSQL_TIME结构的地址。调用mysql_stmt_execute()之后，并在获取结果之前，使用mysql_bind_results()将缓冲绑定到语句上。

下面给出了一个插入DATE、TIME和TIMESTAMP数据的简单示例。假定mysql变量具有有效的连接句柄。

```
  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_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(mysql, query, strlen(query)))
{
  fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

/* set up 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_stmt_bind_param(stmt, bind);

/* supply the data to be sent in the ts structure */
ts.year= 2002;
ts.month= 02;
ts.day= 03;

ts.hour= 10;
ts.minute= 45;
ts.second= 20;

mysql_stmt_execute(stmt);
..
```

25.2.11. C API线程函数介绍

25.2.11.1. my_init()
25.2.11.2. mysql_thread_init()
25.2.11.3. mysql_thread_end()
25.2.11.4. mysql_thread_safe()

当你打算创建线程客户端时，需要使用下述函数。请参见25.2.15节，“如何生成线程式客户端”。

25.2.11.1. my_init()

void my_init(void)

描述

调用任何MySQL函数之前，需要在程序中调用该函数。它将初始化MySQL所需的某些全局变量。如果你正在使用线程安全客户端库，它还能为该线程调用mysql_thread_init()。

通过mysql_init()、mysql_library_init()、mysql_server_init()和mysql_connect()，可自动调用该函数。

返回值

无。

25.2.11.2. mysql_thread_init()

my_bool mysql_thread_init(void)

描述

对于每个创建的线程，需要调用该函数来初始化与线程相关的变量。

它可由my_init()和mysql_connect()自动调用。

返回值

如果成功，返回0，如果出现错误，返回非0值。

25.2.11.3. mysql_thread_end()

void mysql_thread_end(void)

描述

调用pthread_exit()来释放mysql_thread_init()分配的内存之前，需要调用该函数。

注意，该函数不会被客户端库自动调用。必须明确调用它以避免内存泄漏。

返回值

无。

25.2.11.4. mysql_thread_safe()

unsigned int mysql_thread_safe(void)

描述

该函数指明了客户端是否编译为线程安全的。

返回值

如果客户端是线程安全的，返回1，否则返回0。

25.2.12. C API嵌入式服务器函数介绍

25.2.12.1. mysql_server_init()
25.2.12.2. mysql_server_end()

如果希望允许应用程序链接到嵌入式MySQL服务器库，必须使用mysql_server_init()和mysql_server_end()函数。请参见25.1节，“libmysqld，嵌入式MySQL服务器库”。

但是，要想提供改进的内存管理，即使是对与“-lmysqlclient”而不是与“-lmysqld”链接的程序，也应包含启用和结束库使用的调用。mysql_library_init()和mysql_library_end()函数可用于该目的。它们实际上是使其等效于mysql_server_init()和mysql_server_end()的#define符号，但它们的名称更清楚地指明，无论应用程序使用的是libmysqlclient或libmysqld，开始使用或结束MySQL C API库的使用时，应调用它们。关于更多信息，请参见25.2.2节，“C API函数概述”。

25.2.12.1. mysql_server_init()

int mysql_server_init(int argc, char **argv, char **groups)

描述

调用任何其他MySQL函数之前，必须在使用嵌入式服务器的程序中调用该函数。它将启动服务器，并初始化服务器使用的任何子系统（mysys、InnoDB等）。如果未调用该函数，对mysql_init()的下一次调用将执行mysql_server_init()。如果你正在使用与MySQL一起提供的DBUG软件包，应在调用了my_init()之后调用它。

对于main()的参量，argc和argv是类似的参量。argv的第1个元素将被忽略（典型情况下，它包含程序名）。为了方便起见，如果没有针对服务器的命令行参量，argc可以是0。mysql_server_init()将复制参量，以便能够在调用之后安全地摧毁argv或groups。

如果打算连接到外部服务器而不启动嵌入式服务器，应为argc指定负值。

"groups"中以Null终结的字符串列表选择了选项文件中的活动"groups"。请参见4.3.2节，"使用选项文件"。为了方便起见，groups可以是NULL，在该情况下，[server]和[embedded]组是活动的。

示例：

```
#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) {
  if (mysql_server_init(sizeof(server_args) / sizeof(char *),
                        server_args, server_groups))
    exit(1);

  /* Use any MySQL API functions here */

  mysql_server_end();

  return EXIT_SUCCESS;
}
```

返回值

如果OK，返回0。如果出现错误，返回1。

25.2.12.2. mysql_server_end()

void mysql_server_end(void)

描述

在所有其他MySQL函数后，在程序中必须调用该函数一次。它将关闭嵌入式服务器。

返回值

无。

25.2.13. 使用C API时的常见问题

25.2.13.1. 为什么在mysql_query()返回成功后，mysql_store_result()有时会返回NULL
25.2.13.2. What Results You Can Get from a Query
25.2.13.3. 如何获得上次插入行的唯一ID
25.2.13.4. 与C API有关的问题

25.2.13.1. 为什么在mysql_query()返回成功后，mysql_store_result()有时会返回NULL

成功调用mysql_query()后，mysql_store_result()能够返回NULL。出现该情况时，表明出现了下述条件之一：

- 出现了malloc()故障（例如，如果结果集过大）。
- 无法读取数据（在连接上出现了错误）。
- 查询未返回数据（例如，它是INSERT、UPDATE或DELETE）。

通过调用mysql_field_count()，始终能检查语句是否应生成非空结果。如果mysql_field_count()返回0，结果为空，而且上一个查询是未返回值的语句（例如INSERT或DELETE）。如果mysql_field_count()返回非0值，语句应生成非空结果。关于这方面的示例，请参见mysql_field_count()函数介绍。

通过调用mysql_error()或mysql_errno()，可测试是否出现了错误。

25.2.13.2. What Results You Can Get from a Query

除了查询返回的结果集外，还能获取下述信息：

- 执行INSERT、UPDATE或DELETE时，mysql_affected_rows()返回上次查询影响的行数。

对于快速在创建，请使用TRUNCATE TABLE。

- mysql_num_rows()返回结果集中的行数。使用mysql_store_result()，一旦mysql_store_result()返回，就能调用mysql_num_rows()。使用mysql_use_result()，仅当用mysql_fetch_row()获取了所有行后，才能调用mysql_num_rows()。
- mysql_insert_id()返回上次查询生成的ID，该查询使用AUTO_INCREMENT索引将行插入到表内。请参见25.2.3.36节，“mysql_insert_id()”。
- 某些查询（LOAD DATA INFILE ...、INSERT INTO ... SELECT ...、UPDATE）将返回额外信息。结果由mysql_info()返回。关于它返回的字符串格式，请参见关于mysql_info()的介绍。如果没有额外信息，mysql_info()将返回NULL指针。

25.2.13.3. 如何获得上次插入行的唯一ID

如果将记录插入包含AUTO_INCREMENT列的表中，通过调用mysql_insert_id()函数，可获取保存在该列中的值。

通过执行下述代码，可从C应用程序检查某一值是否保存在AUTO_INCREMENT列中（假定该语句已成功执行）。它能确定查询是否是具有AUTO_INCREMENT索引的INSERT：

```
if ((result = mysql_store_result(&mysql)) == 0 &&
    mysql_field_count(&mysql) == 0 &&
    mysql_insert_id(&mysql) != 0)
{
    used_id = mysql_insert_id(&mysql);
}
```

关于更多信息，请参见25.2.3.36节，“mysql_insert_id()”。

生成新的AUTO_INCREMENT值时，也能与mysql_query()一起通过执行SELECT LAST_INSERT_ID()语句获得它，并从该语句返回的结果集检索该值。

对于LAST_INSERT_ID()，最近生成的ID是在服务器上按连接维护的。它不会被另一个客户端改变。即使用non-magic值（即非Null非0值）更新了另一个AUTO_INCREMENT列，也不会更改它。

如果打算使用从某一表生成的ID，并将其插入到第2个表中，可使用如下所示的SQL语句：

```
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
```

注意，mysql_insert_id()返回保存在AUTO_INCREMENT列中的值，无论该值是因存储NULL或0而自动生成的，或是明确指定的，均如此。LAST_INSERT_ID()仅返回自动生成的AUTO_INCREMENT值。如果你保存了除NULL或0之外的确切值，不会影响LAST_INSERT_ID()返回的值。

25.2.13.4. 与C API有关的问题

与C API链接时，在某些系统上可能出现下述错误：

```
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
```

如果在你的系统上出现了该情况，必须在编译/链接行的末尾增加"-lm"，通过该方式包含数学库。

25.2.14. 创建客户端程序

如果你编译了自己编写的MySQL客户端，或编译了从第三方获取的MySQL客户端，必须在链接命令中使用"-lmysqlclient -lz"选项链接它们。你或许还应指定"-L"选项，通知链接程序到哪里找到库。例如，如果将库安装到了/usr/local/mysql/lib，可在链接命令中使用sr/local/mysql/lib -lmysqlclient –lz。

对于使用MySQL头文件的客户端，编译它们时还须指定"-I"选项（例如，-I/usr/local/mysql/include），以便编译器能找到头文件。

为了使在Unix平台上编译MySQL程序变得简单，提供了**mysql_config**脚本。请参见25.9.2节，"mysql_config：获取编译客户端的编译选项"。

你也可以使用它来编译MySQL客户端，如下所述：

```
CFG=/usr/local/mysql/bin/mysql_config
sh -c "gcc -o progname `$CFG --cflags` progname.c `$CFG --libs`"
```

需要使用"sh –c"，使得shell不将**mysql_config**的输出当作1个词对待。

25.2.15. 如何生成线程式客户端

客户端库总是线程安全的。最大的问题在于从套接字读取的net.c中的子程序并不是中断安全的。或许你可能希望用自己的告警中断对服务器的长时间读取，以此来解决问题。如果为SIGPIPE中断安装了中断处理程序，套接字处理功能应是线程安全的。

为了避免连接中断时放弃程序，MySQL将在首次调用mysql_server_init()、mysql_init()或mysql_connect()时屏蔽SIGPIPE。如果你打算使用自己的SIGPIPE处理程序，首先应调用mysql_server_init()，然后安装你的处理程序，

在较旧的发布在我方网站上（http://www.mysql.com/）的二进制版本中，未用线程安全的选项对客户端库进行正常编译（默认情况下，编译的Windows二进制版本是线程安全的）。较新的二进制分发版应是正常的和线程安全的客户端库。

为了获得能从其他线程中断客户端的线程式客户端，并在与MySQL服务器通信时设置超时，应使用"-lmysys"、"-lmystrings"和"-ldbug"库，以及服务器使用的net_serv.o代码。

如果你不需要使用中断或超时，可编译线程安全客户端库（mysqlclient_r）并使用它。。请参见25.2节，"MySQL C API"。在该

情况下，不必担心net_serv.o对象文件或其他MySQL库。

使用线程式客户端并打算使用超时或中断时，可更好地利用thr_alarm.c文件中的子程序。如果你正在使用来自mysys库的子程序，唯一需要记住的事是首先调用my_init()！请参见25.2.11节，“C API线程函数介绍”。

对于除mysql_real_connect()外的所有函数，在默认情况下它们均是线程安全的。在下面的说明中，介绍了编译线程安全客户端库的方法，以及以线程安全方式使用它的方法。（下面关于mysql_real_connect()的说明实际上也适用于mysql_connect()，但由于mysql_connect()已不再被重视，总应尽量使用mysql_real_connect()）。

要想使mysql_real_connect()成为线程安全的，必须用下述命令再次编译客户端库：

```
shell> ./configure --enable-thread-safe-client
```

它创建了线程安全客户端库libmysqlclient_r。（假定你的操作系统有线程安全的gethostbyname_r()函数）。按照连接，该库是线程安全的。可遵循下述警告，使两个线程共享相同的连接：

- 在相同的连接上，两个线程不能同时将查询发送到MySQL服务器。尤其是，必须确保在mysql_query()和mysql_store_result()之间，没有使用相同连接的其他线程。
- 很多线程均能访问由mysql_store_result()检索的不同结果集。
- 如果使用了mysql_use_result，务必确保无其他线程正在使用相同的连接，直至关闭了结果集为止。然而，对于线程式客户端，最好是共享相同的连接以使用mysql_store_result()。
- 如果打算在相同的连接上使用多个线程，必须在mysql_query()和mysql_store_result()调用组合上拥有互斥锁。一旦mysql_store_result()准备就绪，可释放锁定，其他线程可在相同的连接上执行查询。
- 如果使用POSIX线程进行编程，可使用pthread_mutex_lock()和pthread_mutex_unlock()来建立并释放互斥锁。

如果你有1个调用MySQL函数的线程，而该函数未创建与MySQL数据库的连接，就需了解下述事宜：

调用mysql_init()或mysql_connect()时，MySQL会为调试库使用的线程创建与线程相关的变量（尤其）。

在线程调用mysql_init()或mysql_connect()之前，如果调用了MySQL函数，该线程将没有所需的线程类变量，而且你很可能或早或晚结束于内核转储。

要想使这些操作平稳工作，需要采取下述措施：

1. 如果程序在调用mysql_real_connect()之前需要调用任何其他MySQL函数，请在启动程序时调用my_init()。
2. 调用任何MySQL函数之前，在线程处理程序中调用mysql_thread_init()。
3. 在线程中，调用pthread_exit()之前请调用mysql_thread_end()。这样，就能释放MySQL线程类变量使用的内存。

将客户端链接到libmysqlclient_r时，如果存在未定义的符号，可能会出错。在大多数情况下，其原因在于，未将线程库包含在link/compile行上。

25.3. MySQL PHP API

25.3.1. 使用MySQL和PHP的常见问题

PHP是一种服务器端、HTML嵌入式脚本处理语言，可使用该语言创建动态网页。它可用于大多数操作系统和Web服务器，也能访问大多数常见数据库，包括MySQL。PHP可以作为单独程序运行，也能编译为模块，用于Apache Web服务器。

PHP实际上提供了两种不同的MySQL API扩展：

- mysql：适用于PHP版本4和5，该扩展用于MySQL 4.1之前的MySQL版本。该扩展不支持MySQL 5.1中采用的、改进的鉴定协议，也不支持与预处理语句或多语句。如果打算与MySQL 5.1一起使用该扩展，应配置MySQL服务器，以使用“**--old-passwords**”选项（请参见A.2.3节，“客户端不支持鉴定协议”）。在PHP网站的文档中记录了该该扩展http://php.net/mysql。
- mysqli是“MySQL, Improved”的缩写，该扩展仅适用于PHP 5。它能用于MySQL 4.1.1和更高版本。该扩展完全支持MySQL 5.1中采用的鉴定协议，也支持预处理语句和多语句API。此外，该扩展还提供了先进的、面向对象的编程接口。在http://php.net/mysqli上，可找到关于mysqli扩展的文档。在http://www.zend.com/php5/articles/php5-mysqli.php处，给出

了一篇有用的文章。

PHP分发版和文档均能从PHP网站获得。

25.3.1. 使用MySQL和PHP的常见问题

· 错误：超出了最大执行时间，这是一种PHP限制，如果需要，进入文件php.ini，并设置最大执行时间（开始为30秒）。此外，还可以将每脚本允许使用的RAM增加一倍，从8MB变为16MB，这也是个不错的主意。

· 致命错误：在...中调用了不支持或未定义的mysql_connect()函数，这意味着，你的PHP版本不支持MySQL。你可以编译动态MySQL模块并将其加载到PHP，或使用内置的MySQL支持重新编译PHP。在PHP手册中，详细介绍了该进程。

· 错误：对'uncompress'的未定义引用，这意味着所编译的客户端库支持压缩客户端／服务器协议。更正方法是，用"-lmysqlclient"进行链接时，在最后添加"-lz"。

· 错误：客户端不支持鉴定协议，与MySQL 4.1.1和更高版本一起使用较旧的mysql扩展时常会遇到该问题。可能的解决方案是：降级到MySQL 4.0，转向PHP 5和较新的mysqli扩展，或用"--old-passwords"配置MySQL服务器（更多信息，请参见A.2.3节，"客户端不支持鉴定协议"）。

25.4. MySQL Perl API

Perl DBI模块为数据库访问提供了一个通用接口。能够编写无需更改就能与不同的数据库引擎一起工作的DBI脚本。要想使用DBI，必须安装DBI模块，并为打算访问的每种服务器安装数据库驱动程序（DBD）模块。对于MySQL，该驱动程序是DBD::mysql模块。

Perl DBI是推荐的Perl接口。它取代了旧的名为mysqlperl的接口，mysqlperl已过时。

关于Perl DBI支持的安装说明，请参见2.13节，"Perl安装注意事项"。

DBI信息能够在命令行上提供，也能以在线方式提供，或采用印刷形式：

· 一旦安装了DBI和DBD::mysql模块，可使用perldoc 命令在命令行上获取关于它们的信息：

```
· shell> perldoc DBI
· shell> perldoc DBI::FAQ
· shell> perldoc DBD::mysql
```

也可以使用pod2man、pod2html等将这类信息转换为其他格式。

· 关于Perl DBI的在线信息，请访问DBI网站，http://dbi.perl.org/。该站点还提供了1个一般性DBI邮件列表。MySQL AB提供了1个专门针对DBD::mysql 的邮件列表，请参见1.7.1.1 "MySQL邮件列表"。

· 至于印刷版信息，官方的DBI书籍是编程*Perl DBI*（Alligator Descartes和Tim Bunce，O'Reilly & Associates, 2000）。关于该书的信息，请访问DBI网站http://dbi.perl.org/。

关于与MySQL一起使用DBI的专门信息，请参见针对*Web*的*MySQL*和*Perl*（Paul DuBois, New Riders, 2001）。该书的网站是http://www.kitebird.com/mysql-perl/。

25.5. MySQL C++ API

25.5.1. Borland C++

MySQL++是用于C++的MySQL API。Warren Young负责该项目。要想了解更多信息，请访问http://www.mysql.com/products/mysql++/。

25.5.1. Borland C++

可以使用Borland C++ 5.02编译MySQL Windows源码（Windows源码仅包括用于Microsoft VC++的项目，对于Borland C++，你将不得不自己编制项目文件）。

使用Borland C++时的1个已知问题是，它采用了不同于VC++的结构对齐方式。这意味着，如果你打算与Borland C++一起使用默认的libmysql.dll库（它是使用VC++编译的），将会遇到问题。为了避免该问题，仅应调用将Null作为参量的mysql_init()，而不是预先分配MYSQL结构。

25.6. MySQL Python API

MySQLdb为Python提供了MySQL支持，它符合Python DB API版本2.0的要求，可在http://sourceforge.net/projects/mysql-python/上找到它。

25.7. MySQL Tcl API

MySQLtcl是一种简单的API，用于从Tcl编程语言访问MySQL数据库服务器。可在http://www.xdobry.de/mysqltcl/上找到它。

25.8. MySQL Eiffel Wrapper

Eiffel MySQL是一种与MySQL数据库服务器的接口，它采用的是Eiffel编程语言，由Michael Ravits编写。可在http://efsa.sourceforge.net/archive/ravits/mysql.htm上找到它。

25.9. MySQL程序开发实用工具

25.9.1. msql2mysql：转换mSQL程序以用于MySQL
25.9.2. mysql_config：获取编译客户端的编译选项

在本节中，介绍了开发MySQL程序时可能会有用的一些实用工具。

- msql2mysql

1种shell脚本，用于将mSQL程序转换为MySQL程序。它不能处理所有情况，但能为转换提供良好的开端。

- mysql_config

1种shell脚本，能生成编译MySQL程序时所需的选项值。

25.9.1. msql2mysql：转换mSQL程序以用于MySQL

最初，开发的MySQL C API很类似为mSQL数据库系统开发的API。正因为如此，通过更改C API函数的名称，通常能相对容易地转换mSQL程序，使之用于MySQL。

msql2mysql实用工具用于将mSQL C API函数调用转换为其MySQL对等物。**msql2mysql**能够转换位于恰当位置输入文件，在执行具体转换之前复制原件。例如，可采用下述方式使用**msql2mysql**：

```
shell> cp client-prog.c client-prog.c.orig

shell> msql2mysql client-prog.c

client-prog.c converted
```

然后，检查client-prog.c，并执行可能需要的后期转换修订。

msql2mysql使用**replace**实用工具来替换函数名。请参见8.14节，“replace：字符串替换实用工具”。

25.9.2. mysql_config：获取编译客户端的编译选项

mysql_config提供了关于编译MySQL客户端以及将其连接到MySQL的有用信息。

mysql_config支持下述选项：

- --cflags

编译器标志，用于查找包含文件，以及编译libmysqlclient库时所要使用的关键编译器标志和定义。

- --include

编译器选项，用于查找MySQL包含文件（注意，正常情况下应使用"—cflags"而不是该选项）。

- --libmysqld-libs, ---embedded

与MySQL嵌入式服务器进行链接所需的库和选项。

- --libs

与MySQL客户端库进行链接所需的库和选项。

- --libs_r

与线程安全MySQL客户端库进行链接所需的库和选项。

- --port

默认的TCP/IP端口号，配置MySQL时定义。

- --socket

默认的Unix套接字文件，配置MySQL时定义。

- --version

版本号以及MySQL分发版的版本。

如果未使用任何选项调用了**mysql_config**，将显示它所支持的所有选项的列表，以及它们的值：

```
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]
```

可以在命令行上使用**mysql_config**，并包含针对特定选项的值。例如，要想编译MySQL客户端程序，可使用**mysql_config**，如下例所示：

```
shell> CFG=/usr/local/mysql/bin/mysql_config
shell> sh -c "gcc -o progname `$CFG --cflags` progname.c `$CFG --libs`"
```

以这种方式使用**mysql_config**时，务必在字符(``')内调用它。这样，就能通知shell执行它，并将其输出代入到环境命令中。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第26章：连接器

目录

26.1. MySQL Connector/ODBC

26.1.1. MyODBC介绍
26.1.2. 关于ODBC和MyODBC的一般信息
26.1.3. 如何安装MyODBC
26.1.4. 在Windows平台上从二进制版本安装MyODBC
26.1.5. I在Unix平台上从二进制版本安装MyODBC
26.1.6. 在Windows平台上从源码版本安装MyODBC
26.1.7. 在Unix平台上从源码版本安装MyODBC
26.1.8. 从BitKeeper开发源码树安装MyODBC
26.1.9. MyODBC配置
26.1.10. 与MyODBC连接相关的事宜
26.1.11. MyODBC和Microsoft Access
26.1.12. MyODBC和Microsoft VBA及ASP
26.1.13. MyODBC和第三方ODBC工具
26.1.14. MyODBC通用功能
26.1.15. 基本的MyODBC应用步骤
26.1.16. MyODBC API引用
26.1.17. MyODBC数据类型
26.1.18. MyODBC错误代码
26.1.19. MyODBC与VB：ADO、DAO和RDO
26.1.20. MyODBC与Microsoft.NET
26.1.21. 感谢

26.2. MySQL Connector/NET

26.2.1. 前言
26.2.2. 下载并安装MySQL Connector/NET
26.2.3. Connector/NET体系结构
26.2.4. 使用MySQL Connector/NET
26.2.5. MySQL Connector/NET变更史

26.3. MySQL Connector/J

26.3.1. 基本的JDBC概念
26.3.2. 安装 Connector/J
26.3.3. JDBC引用
26.3.4. 与J2EE和其他Java框架一起使用 Connector/J
26.3.5. 诊断 Connector/J方面的问题
26.3.6. Changelog

26.4. MySQL Connector/MXJ

26.4.1. 前言
26.4.2. 支持平台：
26.4.3. Junit测试要求
26.4.4. 运行Junit测试
26.4.5. 作为JDBC驱动程序的一部分运行
26.4.6. 在Java对象中运行
26.4.7. MysqldResource API
26.4.8. 在JMX代理(custom)中运行

26.4.9. 部署在标准的JMX代理环境下 (JBoss)
26.4.10. 安装

在本章中，介绍了MySQL连接器，所谓连接器，是为客户端程序提供与MySQL服务器连接性的驱动程序。

26.1. MySQL Connector/ODBC

26.1.1. MyODBC介绍
26.1.2. 关于ODBC和MyODBC的一般信息
26.1.3. 如何安装MyODBC
26.1.4. 在Windows平台上从二进制版本安装MyODBC
26.1.5. I在Unix平台上从二进制版本安装MyODBC
26.1.6. 在Windows平台上从源码版本安装MyODBC
26.1.7. 在Unix平台上从源码版本安装MyODBC
26.1.8. 从BitKeeper开发源码树安装MyODBC
26.1.9. MyODBC配置
26.1.10. 与MyODBC连接相关的事宜
26.1.11. MyODBC和Microsoft Access
26.1.12. MyODBC和Microsoft VBA及ASP
26.1.13. MyODBC和第三方ODBC工具
26.1.14. MyODBC通用功能
26.1.15. 基本的MyODBC应用步骤
26.1.16. MyODBC API引用
26.1.17. MyODBC数据类型
26.1.18. MyODBC错误代码
26.1.19. MyODBC与VB：ADO、DAO和RDO
26.1.20. MyODBC与Microsoft.NET
26.1.21. 感谢

通过MySQL Connector/ODBC（MyODBC驱动程序系列），MySQL为ODBC提供了支持。这是针对MyODBC驱动程序中Connector/ODBC产品系列的参考，它提供了对MySQL数据库系统的ODBC 3.5x兼容访问。介绍了安装MyODBC和使用MyODBC的方式。此外，在本章中还介绍了能够与MyODBC一起工作的公用程序信息，并回答了一些关于MyODBC的常见问题。

本参考适用于MyODBC 3.51。对于相应的版本，你可以找到旧的二进制版或源码版MyODBC手册。

这是关于MySQL ODBC驱动程序的参考手册，而不是通用ODBC参考。关于ODBC的更多信息，请参阅http://www.microsoft.com/data/。

对于本参考的应用程序开发部分，假定用户在C语言方面有着良好的实践知识，对DBMS有一般了解，最后，还应熟悉MySQL。关于MySQL功能及其语法的更多信息，请参阅http://dev.mysql.com/doc/。

如果你的问题未能在本文档中得到解答，请发送电子邮件至myodbc@lists.mysql.com。

26.1.1. MyODBC介绍

26.1.1.1. 什么是ODBC？
26.1.1.2. 什么是Connector/ODBC？
26.1.1.3. 什么是MyODBC 2.50？
26.1.1.4. 什么是MyODBC 2.50？
26.1.1.5. 从哪获取MyODBC
26.1.1.6. 支持的平台
26.1.1.7. MyODBC邮件列表
26.1.1.8. MyODBC论坛
26.1.1.9. 如何通报MyODBC问题或缺陷
26.1.1.10. 如何提交MyODBC补丁

26.1.1.1. 什么是ODBC？

ODBC（开放式数据库连接性）为客户端程序提供了访问众多数据库或数据源的一种方式。ODBC是标准化的API，允许与SQL数据库服务器进行连接。它是根据SQL Access Group的规范开发的，它定义了一套函数调用、错误代码和数据类型，可将其用于开发独立于数据库的应用程序。通常情况下，当需要数据库独立或需要同时访问不同的数据源时，将用到ODBC。

关于ODBC的更多信息，请参阅http://www.microsoft.com/data/。

26.1.1.2. 什么是Connector/ODBC？

Connector/ODBC是描述MySQL ODBC驱动程序MySQL AB产品系列的名称。它们也称为MyODBC驱动程序。

26.1.1.3. 什么是MyODBC 2.50？

MyODBC 2.50是MySQL AB的32位ODBC驱动程序，它基于ODBC 2.50规范层次0（具有层次1和层次2的特性）。这是开放源码市场最流行的ODBC驱动程序之一，很多用户都使用它来访问MySQL提供的功能。

26.1.1.4. 什么是MyODBC 2.50？

MyODBC 3.51是一种32位ODBC驱动程序，也称为MySQL ODBC 3.51驱动程序。与已有的MyODBC 2.50驱动程序相比，该版本有所增强。它支持ODBC 3.5x规范层次1（全部核心API +层次2特性），以便能够为访问MySQL提供所有的ODBC功能。

26.1.1.5. 从哪获取MyODBC

MySQL AB依GPL（通用公共许可）发布其所有产品。你可以从MySQL AB的网站获取最新的MyODBC二进制版和源码版：http://dev.mysql.com/downloads/。

关于MyODBC的更多信息，请访问http://www.mysql.com/products/myodbc/。

关于许可的更多信息，请访问http://www.mysql.com/company/legal/licensing/。

26.1.1.6. 支持的平台

MyODBC可用于MySQL支持的所有主要平台，如：

- Windows 95, 98, Me, NT, 2000, XP和2003
- 所有Unix操作系统
  - AIX
  - Amiga
  - BSDI
  - DEC
  - FreeBSD
  - HP-UX 10, 11
  - Linux
  - Mac OS X Server

- Mac OS X
- NetBSD
- OpenBSD
- OS/2
- SGI Irix
- Solaris
- SunOS
- SCO OpenServer
- SCO UnixWare
- Tru64 Unix

对于特定平台，如果无法下载二进制版本，可通过下载驱动程序源码自行创建驱动程序。你也可以为MySQL贡献二进制代码，方式是发送邮件至myodbc@lists.mysql.com，这样其他用户就能使用你贡献的内容。

26.1.1.7. MyODBC邮件列表

MySQL AB通过其邮件列表为用户社区提供帮助。对于与MyODBC有关的事宜，可使用myodbc@lists.mysql.com邮件列表，从有经验的用户处获得帮助。

关于订阅MySQL邮件列表或浏览列表档案的更多信息，请访问http://lists.mysql.com/。

其中，关注程度最高的是论坛MySQL连接器部分的ODBC论坛。

26.1.1.8. MyODBC论坛

通过MySQL论坛（位于http://forums.mysql.com），可获得有经验用户的支持和帮助。

26.1.1.9. 如何通报MyODBC问题或缺陷

如果遇到与MyODBC有关的困难或问题，首先应使用ODBC管理器和MyODBC生成一份日志文件（请求来自ODBC ADMIN的日志时获得的日志文件）。关于完成该步骤的方式，请参见26.1.9.7节，“获取ODBC跟踪文件”。

检查MyODBC跟踪文件，找出可能出错的地方。通过在myodbc.log文件中搜索字符串“>mysql_real_query”，可确定已执行的语句。

此外，你还应尝试从**mysql**客户端程序或admndemo执行语句。这样，就能帮助你确定错误的出处，MyODBC或MySQL。

如果你发现了不正确的事项，请将相关行（最多40行）发送给MyODBC邮件列表。请参见1.7.1.1节，“MySQL邮件列表”。请勿发送整个MyODBC或ODBC日志文件！

如果你无法找出错误之所在，最后的选择是，以tar或zip格式创建包含MyODBC跟踪文件、ODBC日志文件和README文件（阐明问题）的档案。你可以将该档案文件发送至ftp://ftp.mysql.com/pub/mysql/upload/。只有位于MySQL AB的我们才能访问你上传的文件，而且我们会十分谨慎地对待这类数据。

如果你创建了仍出现问题的程序，请将该程序也包含在档案文件中。

如果程序能够与某些其他SQL服务器一起工作，档案中还应包含在这类其他SQL服务器下工作的ODBC日志文件。

请记住，你提供给我们的信息越多，我们更正问题的机会就越大。

26.1.1.10. 如何提交MyODBC补丁

你可以通过电子邮件，就已有代码或问题发送补丁或提出更好的解决方案：myodbc@lists.mysql.com。

26.1.2. 关于ODBC和MyODBC的一般信息

26.1.2.1. ODBC介绍
26.1.2.2. MyODBC体系结构
26.1.2.3. ODBC驱动管理器
26.1.2.4. MySQL ODBC驱动程序的类型

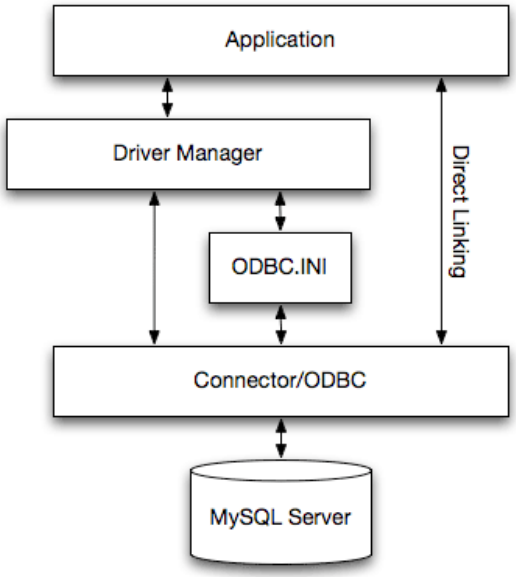
26.1.2.1. ODBC介绍

开放式数据库连接性（ODBC）是广泛接受的用于数据库访问的应用程序编程接口（API）。它基于针对数据库API的CLI（调用层接口）规范（来自X/Open和ISO/IEC），并采用了结构化查询语言（SQL）作为其数据库访问语言。

在26.1.16节，“MyODBC API引用”中，概要介绍了MyODBC支持的ODBC功能。关于ODBC的更多信息，请参阅http://www.microsoft.com/data/。

26.1.2.2. MyODBC体系结构

MyODBC体系结构建立在5个组件上，如下图所示：



- 应用程序：

应用程序指的是通过调用ODBC API来访问MySQL服务器上数据的程序。应用程序使用标准的ODBC调用与驱动管理器进行通信。应用程序不关心数据的存储位置，存储方式，甚至不关心为访问数据而进行的系统配置方式。它仅需要知道数据源名（DSN）。

对于所有的应用程序，无论它们使用OBDC的方式是什么，很多任务是共同的。这些任务包括：

o 选择MySQL服务器，并与之连接。

- 提交将要执行的SQL语句。
- 检索结果（如果有的话）。
- 处理错误。
- 提交或回滚包含SQL语句的事务。
- 断开与MySQL服务器的连接。

由于大多数数据访问工作是使用SQL完成，对于使用OBDC的应用程序来说，其主要任务是提交SQL语句，并检索由这些语句生成的结果。

· 驱动管理器：

驱动管理器是用于管理应用程序和驱动程序间通信的库。它负责执行下述任务：

- 解析数据源名（DSN）。
- 加载和卸载驱动程序。
- 处理ODBC调用，或将其传递给驱动程序。

· **MyODBC**驱动程序：

MyODBC驱动程序是用于实施ODBC API所提供功能的库。它负责处理ODBC函数调用，将SQL请求提交给MySQL服务器，并将结果返回给应用程序。如有必要，驱动程序会更改应用程序的请求，以便该请求符合MySQL支持的语法。

· **ODBC.INI**：

ODBC.INI是ODBC配置文件，其中保存了连接到服务器所需的驱动信息和数据库信息。驱动管理器将使用它来确定加载哪个驱动程序（使用数据源名）。驱动程序将根据指定的DSN使用它来读取连接参数。更多信息，请参见26.1.9节，“MyODBC配置”。

· **MySQL**服务器：

MySQL服务器是数据源。MySQL是：

- 一种数据库管理系统（DBMS）
- 一种关联数据库管理系统（RDBMS）
- 开放源码软件

26.1.2.3. ODBC驱动管理器

ODBC驱动管理器是用于管理ODBC应用程序和驱动程序间通信的库。其主要功能包括：

· 解析数据源名（DSN）。

· 加载和卸载驱动程序。

· 处理ODBC函数调用，或将其传递给驱动程序。

下面给出了一些常用的驱动程序：

· Microsoft Windows ODBC驱动管理器（odbc32.dll），http://www.microsoft.com/data/

· unixODBC Unix驱动管理器（libodbc.so），http://www.unixodbc.org。

- iODBC ODBC Unix驱动管理器（libiodbc.so），http://www.iodbc.org。

从2.1.2版开始，UnixODBC也提供MyODBC 3.51。

26.1.2.4. MySQL ODBC驱动程序的类型

MySQL AB支持两种通过ODBC API访问MySQL功能的开放源码ODBC驱动程序： MyODBC (MyODBC 2.50)和MySQL ODBC 3.51驱动(MyODBC 3.51)。

注释： 从本节起，我们将这两类驱动程序统称为MyODBC。但当存在差异时，我们将使用它们的本名。

26.1.3. 如何安装MyODBC

MyODBC能够工作在Windows 9x, Me, NT, 2000, XP和2003，以及大多数Unix平台上。

MyODBC是开放源码软件。你可以在网站http://dev.mysql.com/downloads/connector/odbc/上找到它的最新版本。请注意，2.50.x版采用的是LGPL许可，而3.51.x版采用的是GPL许可。

如果使用MyODBC时出现了问题，而且你的程序还能与OLEDB一起工作，应尝试使用OLEDB驱动程序。

正常情况下，在Windows机器上仅需安装MyODBC。仅当你拥有运行在Unix机器上的程序（如ColdFusion），而且该程序将使用ODBC来访问数据库时，才需安装用于Unix的MyODBC。

如果你打算在Unix机器上安装MyODBC，还需要1个ODBC管理器。MyODBC能够与大多数Unix ODBC管理器一起工作。

- 要想使用ODBC应用程序（不支持MySQL的应用程序），建立从Windows平台到Unix平台的连接，首先必须在Windows机器上安装MyODBC。
- 用户和Windows机器必须具有访问位于Unix机器上的MySQL服务器的权限。这可通过GRANT命令设置。请参见13.5.1.3节，“GRANT和REVOKE语法”。
- 必须创建ODBC DSN条目，方式如下：

1. 打开Windows机器上的控制面板。
2. 双击ODBC数据源32位图标。
3. 点击选项卡“用户DSN”。
4. 点击“添加”按钮。
5. 在“创建新数据源”屏幕上选择MySQL，并点击“完成”按钮。
6. 显示MySQL驱动程序的默认配置屏幕。请参见26.1.9.2节，“在Windows上配置MyODBC DSN”。

- 启动应用程序，并使用在ODBC管理器中指定的DSN选择ODBC驱动程序。

注意，在MySQL屏幕上还显示了其他选项，如果遇到问题，可尝试这些选项（如跟踪、连接时不提示等）。

26.1.4. 在Windows平台上从二进制版本安装MyODBC

要想在Windows平台上安装MyODBC，应从下述站点下载恰当的分发文件，http://dev.mysql.com/downloads/connector/odbc/，解压该文件，并执行**MyODBC-***VERSION***.exe**文件。

在Windows平台上，安装较旧的MyODBC 2.50驱动时，可能会遇到下述错误：

拷贝C:\WINDOWS\SYSTEM\MFC30.DLL时出现错误。

重启Windows，并再次安装（在运行任何使用ODBC的应用程序之前）。

问题在于其他程序正使用ODBC。由于Windows的设计方式，在这种情况下，你可能无法使用Microsoft的ODBC设置程序安装新的ODBC驱动。在大多数情况下，可以通过连续按"忽略"键拷贝剩余的MyODBC文件，最终安装应仍能工作。如不然，解决方案是在"安全模式"下重新启动计算机。在重启的过程中，在机器启动Windows前按F8，选择"安全模式"，安装MyODBC，然后在正常模式下重新启动计算机。

26.1.5. I在Unix平台上从二进制版本安装MyODBC

26.1.5.1. 从RPM分发版安装MyODBC
26.1.5.2. 从二进制Tarball分发版安装MyODBC

26.1.5.1. 从RPM分发版安装MyODBC

要想使用RPM分发版在Linux平台上安装或升级MyODBC，可简单地下载最新MyODBC的RPM分发版，并按照下面介绍的方式操作。使用**su root**成为根用户，然后安装RPM文件。

如果是首次安装：

```
shell> su root
shell> rpm -ivh MyODBC-3.51.01.i386-1.rpm
```

如果驱动程序已存在，可按照下述方式升级它：

```
shell> su root
shell> rpm -Uvh MyODBC-3.51.01.i386-1.rpm
```

如果存在关于MySQL客户端库libmysqlclient的任何依存错误，可使用"-nodeps"选项简单地忽略它，然后确保MySQL客户端共享库位于路径中或通过LD_LIBRARY_PATH进行了设置。

这样，就会将驱动程序库和相关文件分别安装到/usr/local/lib和/usr/share/doc/MyODBC目录下。请转至26.1.9.3节，"在Unix平台上配置MyODBC DSN"。

要想卸载驱动程序，请首先成为根用户，然后执行**rpm**命令：

```
shell> su root
shell> rpm -e MyODBC
```

26.1.5.2. 从二进制Tarball分发版安装MyODBC

要想从tarball分发版（.tar.gz文件）安装驱动程序，请下载针对你所使用操作系统的最新版驱动程序，然后按照下述步骤操作：

```
shell> su root
shell> gunzip MyODBC-3.51.01-i686-pc-linux.tar.gz
shell> tar xvf MyODBC-3.51.01-i686-pc-linux.tar
shell> cd MyODBC-3.51.01-i686-pc-linux
```

请阅读INSTALL-BINARY文件中的安装说明，并执行下述命令：

```
shell> cp libmyodbc* /usr/local/lib
shell> cp odbc.ini /usr/local/etc
shell> export ODBCINI=/usr/local/etc/odbc.ini
```

然后，请跳至26.1.9.3节，"在Unix平台上配置MyODBC DSN"为MyODBC配置DSN。更多信息，请参见与发布版一起提供的INSTALL-BINARY文件。

26.1.6. 在Windows平台上从源码版本安装MyODBC

26.1.6.1. 要求
26.1.6.2. 构建MyODBC 3.51
26.1.6.3. 测试
26.1.6.4. 构建MyODBC 2.50

26.1.6.1. 要求

- MDAC, Microsoft Data Access SDK：http://www.microsoft.com/data/。
- MySQL客户端库以及MySQL 4.0.0或更高版本的包含文件。（最好是MySQL 4.0.16或更高版本）。应满足上述要求，这是因为MyODBC需要用到该版本以上的库才提供的新调用和结构。要想获得客户端库和包含文件，请访问http://dev.mysql.com/downloads/。

26.1.6.2. 构建MyODBC 3.51

MyODBC 3.51源码分发版包括使用**nmake**的Makefiles。在分发版中，你可以找到用于创建发布版的Makefile，以及用于创建驱动库和DLL调试版的Makefile_debug。

要想创建驱动程序，请采取下述步骤：

1. 下载并将源码展开到文件夹，然后将位置切换到该文件夹。在下述命令中，假定文件夹为myodbc3-src：

```
C:\> cd myodbc3-src
```

3. 编辑Makefile，为MySQL客户端库和头文件指定正确的路径。然后使用下述命令创建并安装发布版。

```
C:\> nmake -f Makefile
C:\> nmake -f Makefile install
```

nmake -f Makefile用于创建驱动程序的发布版并将二进制码放入名为Release的子目录下。

nmake -f Makefile install用于将驱动程序DLL和库（myodbc3.dll, myodbc3.lib）安装（拷贝）到系统目录下。

6. 要想创建调试版，请使用Makefile_Debug而不是Makefile，如下所示：

```
C:\> nmake -f Makefile_debug
C:\> nmake -f Makefile_debug install
```

9. 使用下述命令，可清除并重新创建驱动程序：

```
C:\> nmake -f Makefile clean
C:\> nmake -f Makefile install
```

注释：

- 确保在Makefiles中指定了正确的MySQL客户端库和头文件路径（设置MYSQL_LIB_PATH和MYSQL_INCLUDE_PATH变量）。默认的头文件路径是C:\mysql\include。对于发布版DLL，默认的库路径是C:\mysql\lib\opt，对于调试版，默认路径是C:\mysql\lib\debug。
- 关于**nmake**的完整用法，请参见http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vcce4/html/evgrfRunningNMAKE.asp。
- 如果你正在使用BitKeeper树来进行编译，所有的针对Windows的Makefiles均将被命名为Win_Makefile*。

26.1.6.3. 测试

将驱动程序库拷贝／安装到系统目录后，可使用示例子目录下提供的示例测试这些库是否已正确创建：

```
C:\> cd samples
C:\> nmake -f Makefile all
```

26.1.6.4. 构建MyODBC 2.50

MyODBC 2.50源码分发版包含VC工作空间文件。通过在Microsoft Visual Studio 6.0中加载这些文件（.dsp和.dsw），可使用它们直接创建驱动程序。

26.1.7. 在Unix平台上从源码版本安装MyODBC

26.1.7.1. 要求
26.1.7.2. 典型配置选项
26.1.7.3. 线程安全客户端
26.1.7.4. 共享或静态选项
26.1.7.5. 启用调试信息
26.1.7.6. 允许文档功能
26.1.7.7. 创建和编译
26.1.7.8. 创建共享库
26.1.7.9. 安装驱动库
26.1.7.10. 在Unix平台上测试MyODBC
26.1.7.11. Mac OS X注意事项
26.1.7.12. HP-UX注意事项
26.1.7.13. AIX注意事项

26.1.7.1. 要求

- MySQL客户端库以及MySQL 4.0.0或更高版本的包含文件。（最好是MySQL 4.0.16或更高版本）。应满足上述要求，这是因为MyODBC需要用到该版本以上的库才提供的新调用和结构。要想获得客户端库和包含文件，请访问http://dev.mysql.com/downloads/。
- 必须使用"--enable-thread-safe-client"选项配置MySQL库。Libmysqlclient是作为共享库安装的。
- 必须安装下述Unix ODBC驱动管理器之一：
  - iodbc 3.0或更高版本（http://www.iodbc.org）
  - unixodbc Alpha 3或更高版本（http://www.unixodbc.org）
- 如果使用了未编译在MySQL客户端库中的字符集（默认字符集为： latin1 big5 czech euc_kr gb2312 gbk sjis tis620 ujis），就需要从字符集目录下将mysql字符定义安装到*SHAREDIR*中（默认情况下位于/usr/local/mysql/share/mysql/charsets）。如果在相同机器上安装了MySQL，它们应位于恰当位置。

一旦完成了所有所需文件的安装，将源码文件解包到单独目录下，并按照下面给出的说明进行操作。

26.1.7.2. 典型配置选项

使用**configure**脚本，能够对你所创建MyODBC的配置方式进行多种控制。典型情况下，可在"**configure**"命令行使用选项完成该配置操作。也可以使用环境变量来影响配置。要想了解"**configure**"命令支持的选项列表和环境变量，可运行下述命令：

```
shell> ./configure --help
```

下面介绍了一些常用的"**configure**"选项。

1. 要想编译MyODBC，须使用"--with-mysql-path=*DIR*"选项来提供MySQL客户端库文件和包含文件路径，其中，"*DIR*"是MySQL的安装目录。

可通过运行"*DIR*/bin/mysql_config"来确定MySQL编译选项。

2. 为ODBC驱动管理器（iodbc或unixobc）提供标准的头文件和库文件路径。

· 如果你正在使用iodbc，而且iodbc未安装在其默认位置（/usr/local），可能需要使用"--with-iodbc=*DIR*"选项，其中，"*DIR*"是iodbc的安装目录。

如果iodbc头文件未位于*DIR*/include目录下，可使用"--with-iodbc-includes=*INCDIR*"选项指定它们的位置。

上面所述也适用于库文件。如果库文件未位于*DIR*/lib目录下，可使用"--with-iodbc-libs=*LIBDIR*"选项。

· 如果你正在使用unixODBC，可使用"--with-unixODBC=*DIR*"选项（区分大小写），让**configure**寻找unixODBC而不是默认的iodbc，其中，"*DIR*"是unixODBC的安装目录。

如果unixODBC头文件和库文件未位于目录*DIR*/include和*DIR*/lib下，可使用"--with-unixODBC-includes=*INCDIR*"和"--with-unixODBC-libs=*LIBDIR*"选项。

3. 或许你也希望指定不同于"/usr/local"的安装前缀。例如，要想将MyODBC驱动安装到"/usr/local/odbc/lib"目录下，可使用"--prefix=/usr/local/odbc"选项。

最终的配置命令应与下面给出的相似：

```
shell> ./configure --prefix=/usr/local \
         --with-iodbc=/usr/local \
         --with-mysql-path=/usr/local/mysql
```

26.1.7.3. 线程安全客户端

为了将驱动程序与MySQL线程安全客户端库libmysqlclient_r.so或libmysqlclient_r.a链接起来，必须指定下述**configure**选项：

```
--enable-thread-safe
```

也可以使用下述选项禁止它：

```
--disable-thread-safe
```

使用该选项，能够通过mysql线程安全客户端库libmysqlclient_r.so（扩展名与操作系统有关）的链接，创建驱动程序线程安全库libmyodbc3_r.so。

在配置线程安全选项时，如果出现了配置错误，应检查config.log，检查错误是否是因系统中缺少线程库而导致的，如果是，使用LIBS选项提供一个，即

```
LIBS="-lpthread" ./configure ..
```

26.1.7.4. 共享或静态选项

可以使用下述选项启用或禁止共享和静态选项：

```
--enable-shared[=yes/no]
--disable-shared
--enable-static[=yes/no]
--disable-static
```

26.1.7.5. 启用调试信息

默认情况下，所有的二进制分发版均会被创建为非调试版（采用"--without-debug"进行配置）。

要想启用调试信息，请使用源码分发版创建驱动程序，并在运行"**configure**"时使用"--with-debug"选项。

26.1.7.6. 允许文档功能

该选项仅能用于BK克隆树，而不是一般的源码分发版。

默认情况下，驱动程序是使用"--without-docs"创建的。如果希望在正常创建过程中观察文档信息，可使用下述选项进行配置：

```
--with-docs
```

26.1.7.7. 创建和编译

要想创建驱动程序库，仅需执行"**make**"，该命令能完成所有事项。

```
shell> make
```

如果出现错误，更正后，继续执行创建进程。如果无法创建，请发送详细的电子邮件至myodbc@lists.mysql.com，以获取进一步帮助。

26.1.7.8. 创建共享库

在大多数平台上，默认情况下，MySQL不会创建或支持".so"（共享）客户端库，这是因为，创建共享库在过去造成过很多问题。

在这种情况下，你可以下载MySQL分发版，并使用以下选项进行配置：

```
--without-server --enable-shared
```

要想创建共享驱动程序库，必须为"**configure**"指定"--enable-shared"选项。默认情况下，"**configure**"不启用该选项。

如果使用"--disable-shared"选项进行了配置操作，可使用下述命令，从静态库创建".so"文件。

```
shell> cd MyODBC-3.51.01
shell> make
shell> cd driver
shell> CC=/usr/bin/gcc \
     $CC -bundle -flat_namespace -undefined error \
         -o .libs/libmyodbc3-3.51.01.so \
         catalog.o connect.o cursor.o dll.o error.o execute.o \
         handle.o info.o misc.o myodbc3.o options.o prepare.o \
         results.o transact.o utility.o \
         -L/usr/local/mysql/lib/mysql/ \
         -L/usr/local/iodbc/lib/ \
         -lz -lc -lmysqlclient -liodbcinst
```

如果你正在使用unixODBC而不是iODBC，务必将"-liodbcinst"更改为"-lodbcinst"，并相应地配置库路径。

这样，就创建了libmyodbc3-3.51.01.so文件，并将其放在".libs"目录下。将该文件拷贝到MyODBC库目录下（/usr/local/lib，或使用"--prefix"提供的安装目录下的"lib"目录）。

```
shell> cd .libs

shell> cp libmyodbc3-3.51.01.so /usr/local/lib

shell> cd /usr/local/lib

shell> ln -s libmyodbc3-3.51.01.so libmyodbc3.so
```

要想创建线程安全驱动程序库：

```
shell> CC=/usr/bin/gcc \
     $CC -bundle -flat_namespace -undefined error
      -o .libs/libmyodbc3_r-3.51.01.so
      catalog.o connect.o cursor.o dll.o error.o execute.o
      handle.o info.o misc.o myodbc3.o options.o prepare.o
      results.o transact.o utility.o
      -L/usr/local/mysql/lib/mysql/
      -L/usr/local/iodbc/lib/
      -lz -lc -lmysqlclient_r -liodbcinst
```

26.1.7.9. 安装驱动库

要想安装驱动程序库，请执行下述命令：

```
shell> make install
```

该命令将安装下述库集合之一：

对于MyODBC 3.51：

- libmyodbc3.so
- libmyodbc3-3.51.01.so，其中，3.51.01是驱动程序的版本
- libmyodbc3.a

对于线程安全MyODBC 3.51：

- libmyodbc3_r.so
- libmyodbc3-3_r.51.01.so
- libmyodbc3_r.a

对于MyODBC 2.5.0：

- libmyodbc.so
- libmyodbc-2.50.39.so，其中，2.50.39是驱动程序的版本
- libmyodbc.a

关于创建进程的更多信息，请参阅与源码分发版一起提供的INSTALL文件。注意，如果你试图使用Sun的"make"，可能会以错误结束。从另一方面来说，GNU **gmake**在所有平台上均能良好工作。

26.1.7.10. 在Unix平台上测试MyODBC

要想与你创建的库一起运行分发版中提供的示例，可执行：

```
shell> make test
```

首先，务必在odbc.ini中配置DSN 'myodbc3'，并将环境变量ODBCINI指向正确的odbc.ini文件；同时MySQL服务器应处于运行状态。在驱动分发版中，可找到一个示例用odbc.ini文件。

你甚至可以更改示例/运行示例脚本，以命令行参数的形式将所需的DSN、UID和PASSWORD值传递给示例。

26.1.7.11. Mac OS X注意事项

要想在Mac OS X (Darwin)环境下创建驱动程序，可使用下述**configure**示例：

```
shell> ./configure --prefix=/usr/local
           --with-unixODBC=/usr/local
           --with-mysql-path=/usr/local/mysql
           --disable-shared
           --enable-gui=no
           --host=powerpc-apple
```

该命令假定unixODBC和MySQL均安装在默认位置。如不然，请进行相应配置。

在 Mac OS X环境下，"--enable-shared"选项将默认创建".dylib"文件。你也可以采用下述方式创建".so"文件：

```
shell> make
shell> cd driver
shell> CC=/usr/bin/gcc \
     $CC -bundle -flat_namespace -undefined error
         -o .libs/libmyodbc3-3.51.01.so *.o
         -L/usr/local/mysql/lib/
         -L/usr/local/iodbc/lib
         -liodbcinst -lmysqlclient -lz -lc
```

要想创建线程安全驱动程序库：

```
shell> CC=/usr/bin/gcc \
     $CC -bundle -flat_namespace -undefined error
     -o .libs/libmyodbc3-3.51.01.so *.o
     -L/usr/local/mysql/lib/
     -L/usr/local/iodbc/lib
     -liodbcinst -lmysqlclienti_r -lz -lc -lpthread
```

如果你正在使用unixODBC而不是iODBC，务必将"-liodbcinst"更改为"-lodbcinst"，并相应地配置库路径。

在Apple的GCC版本中，**cc**和**gcc**实际上均是**gcc3**的符号链接。

将该库拷贝到$prefix/lib目录下，并将symlink拷贝到libmyodbc3.so。

可以使用下述命令交叉检验输出的共享库属性：

```
shell> otool -LD .libs/libmyodbc3-3.51.01.so
```

26.1.7.12. HP-UX注意事项

要想在HP-UX 10.x或11.x环境下创建驱动程序，可使用下述**configure**示例：

如果使用**cc**：

```
shell> CC="cc" \
     CFLAGS="+z" \
     LDFLAGS="-Wl,+b:-Wl,+s" \
     ./configure --prefix=/usr/local
           --with-unixodbc=/usr/local
           --with-mysql-path=/usr/local/mysql/lib/mysql
           --enable-shared
           --enable-thread-safe
```

如果使用**gcc**：

```
shell> CC="gcc" \
     LDFLAGS="-Wl,+b:-Wl,+s" \
```

```
./configure --prefix=/usr/local
      --with-unixodbc=/usr/local
      --with-mysql-path=/usr/local/mysql
      --enable-shared
      --enable-thread-safe
```

一旦创建了驱动程序，使用“**chatr .libs/libmyodbc3.sl**”交叉检查其属性，查看是否需要使用SHLIB_PATH环境变量的MySQL客户端库。对于静态版，忽略所有的共享库选项，并使用“--disable-shared”选项运行“**configure**”。

26.1.7.13. AIX注意事项

要想在AIX环境下创建驱动程序，可使用下述**configure**示例：

```
shell> ./configure --prefix=/usr/local
      --with-unixodbc=/usr/local
      --with-mysql-path=/usr/local/mysql
      --disable-shared
      --enable-thread-safe
```

注释： 关于在不同平台上创建和设置静态和共享库方式的更多信息，请参见跨平台使用静态和共享库。

26.1.8. 从BitKeeper开发源码树安装MyODBC

注释： 如果你对协助我们测试新的代码感兴趣，应阅读本节的内容。

要想获得我方的最新开发源码树，请：

1. 参见2.8.3节，“从开发源码树安装”，关于如何下载和安装BitKeeper的说明。
2. 安装完BitKeeper后，首先进入打算在其中工作的目录，然后，如果打算克隆MyODBC 3.51分支，请使用该命令：

```
3.          shell> bk clone bk://mysql.bkbits.net/myodbc3 myodbc-3.51
```

在前面的示例中，源码树是在myodbc-3.51/中设置的，或在当前目录的myodbc3/子目录下设置的（默认）。如果你位于防火墙后，而且仅能启动HTTP连接，也可以通过HTTP使用BitKeeper。如果要求使用代理服务器，可简单地设置环境变量http_proxy，使之指向代理服务器：

```
shell> export http_proxy="http://your.proxy.server:8080/"
```

执行克隆操作时，用http://替换bk://。例如：

```
shell> bk clone http://mysql.bkbits.net/myodbc3 myodbc-3.51
```

首次下载源码树时需要一段时间，具体情况取决于连接速度，请耐心等候。

4. 要想运行下一组命令，需要GNU autoconf 2.52（或更新版本），automake 1.4，libtool 1.4，以及m4。

```
5.          shell> cd myodbc-3.51
6.          shell> bk -r edit
7.          shell> aclocal; autoheader; autoconf;  automake;
8.          shell> ./configure  # Add your favorite options here
9.          shell> make
```

关于如何创建的更多信息，请参阅位于相同目录下的INSTALL文件。在Windows平台下，创建驱动程序时，请使用Windows Makefiles WIN-Makefile和WIN-Makefile_debug，更多信息，请参见26.1.6节，“在Windows平台上从源码版本安装MyODBC”。

10.完成创建后，运行**make install**，将MyODBC 3.51驱动程序安装到你的系统上。

11.如果进入了**make**阶段，但并未编译分发版本，请将其通报给myodbc@lists.mysql.com。

12.启动了**bk clone**操作获得源码树后，应定期运行**bk pull**进行更新。

13.可以使用“**bk sccstool**”检查树的变更史。如果你发现了有趣的差异，并对代码存在一问，请立刻发送电子邮件至myodbc@lists.mysql.com。

此外，如果你认为有更好的主意，请发送电子邮件至相同的地址并附上补丁。更改了源码后，使用“**bk diffs**”可生成补丁。如果你没有时间就你的观点编写代码，可发送描述性信息。

14.BitKeeper具有一个可通过**bk helptool**访问的帮助工具。

通过浏览http://mysql.bkbits.net:8080/myodbc3，也能在线浏览变化集、注释和源代码。

26.1.9. MyODBC配置

26.1.9.1. 什么是数据源名？
26.1.9.2. 在Windows上配置MyODBC DSN
26.1.9.3. 在Unix平台上配置MyODBC DSN
26.1.9.4. 连接参数
26.1.9.5. 没有预定义DSN下的连接
26.1.9.6. 建立从系统A到系统B的远程连接
26.1.9.7. 获取ODBC跟踪文件
26.1.9.8. 用MyODBC测试的应用程序
26.1.9.9. 已知的能与MyODBC一起工作的程序

本节介绍了配置MyODBC的方法，包括DSN创建，以及驱动程序在连接字符串中作为输入参数的不同参数。此外，还介绍了创建ODBC跟踪文件的方法。

26.1.9.1. 什么是数据源名？

“数据源”是提供数据的地点。数据源必须有稳定的标识符，即数据源名。使用数据源名，MySQL可访问初始化信息。通过初始化信息，MySQL能够了解去哪里访问数据库，以及在开始访问时使用什么设置。

事实上，数据源就是数据的路径。在不同的情况下，它可能有着不同的内容，但是在典型情况下，它指明了正在运行的MySQL服务器（例如，通过网络地址或服务器名），连接时该服务器的默认数据库，以及必要的连接信息（如端口）。MySQL驱动程序（以及Windows系统上的ODBC驱动管理器）将使用数据源进行连接。对于该目的，名为Microsoft ODBC数据源管理器的管理工具可能十分有用。

有两处可能保存初始化信息的位置： Windows注册表（Windows系统），或DSN文件（任何系统）。

如果信息位于Windows注册表中，它称为“机器数据源”。它可以是“用户数据源”，在这种情况下，只有一位用户能看到它。它也可以是“系统数据源”，在这种情况下，计算机上的所有用户均能访问它，如果用户是通过Microsoft Windows NT服务连接在一起的话，与该计算机相连的所有用户均能访问它。运行ODBC数据管理程序时，可以选择是否使用“用户”或“系统”，它们位于不同的选项卡上。

如果信息位于DSN文件中，它称为“文件数据源”。这是一种文本文件。其优点在于： （a）它适合于任何类型的计算机，而不仅仅是使用Windows操作系统的计算机；（b）其内容的拷贝或传输相对容易。

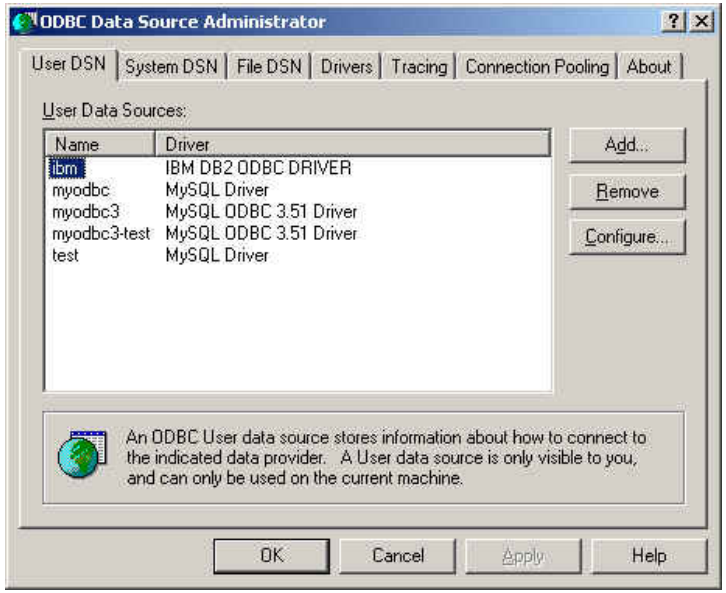
26.1.9.2. 在Windows上配置MyODBC DSN

要想在Windows平台上添加和配置新的MyODBC数据源，请使用ODBC数据源管理器。ODBC管理器能够更新数据源连接信息。添加了数据源时，ODBC管理器能够更新注册信息。

要想从控制面板打开ODBC管理器：

1. 点击"开始"，将指针指向"设置"，然后点击"控制面板"。
2. 在运行Microsoft Windows 2000或更新版本的计算机上，双击"管理工具"，然后双击"数据源"（ODBC）。在运行旧版本Windows的计算机上，双击32位ODBC或ODBC。

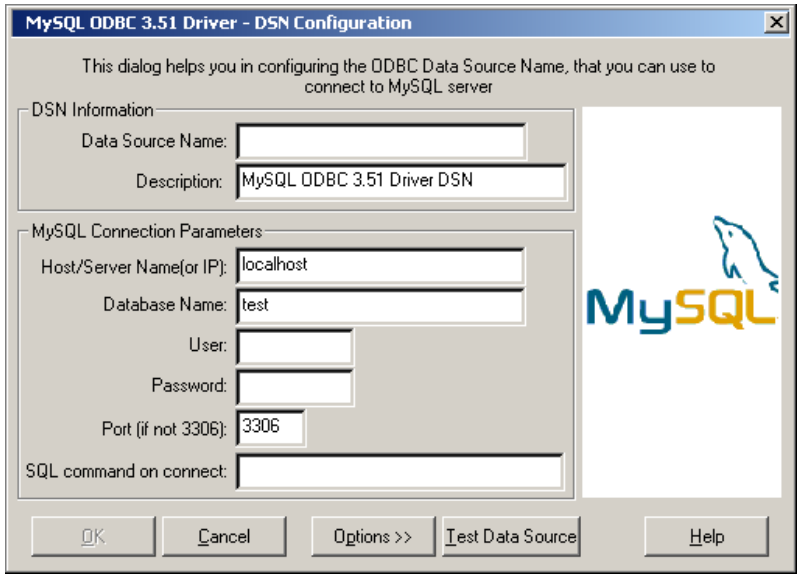
打开ODBC数据源管理器对话框，如下图所示：



点击"帮助"以了解ODBC数据源管理器对话框各选项卡的详细信息。

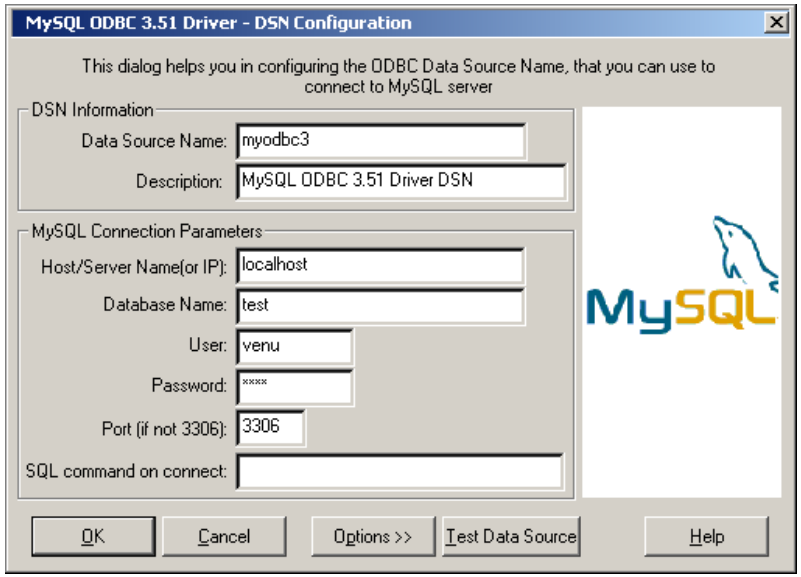
要想在Windows平台上添加数据源：

1. 打开ODBC数据源管理器。
2. 在ODBC数据源管理器对话框中，点击"添加"。打开"创建新数据源"对话框。
3. 选择MySQL ODBC 3.51驱动程序，然后点击"完成"。打开"MySQL ODBC 3.51驱动程序-DSN配置"对话框，如下图所示：



4. 在"数据源名"框中，输入打算访问的数据源的名称。它可以是你选择的任何有效名称。
5. 在"描述"框中，输入DSn所需的描述信息。
6. 在"主机"或"服务器名"（或IP）框中，输入准备访问的MySQL服务器主机的名称。默认情况下为localhost（本地主机）。
7. 在"数据库名"框中，输入准备用作默认数据库的MySQL数据库名称。
8. 在"用户"框中，输入你的MySQL用户名（数据库用户ID）。
9. 在"密码"框中输入密码。
10. 在"端口"框中，如果端口不是默认端口，输入端口号。
11. 在"SQL命令"框中，可输入建立连接后自动执行的SQL语句。

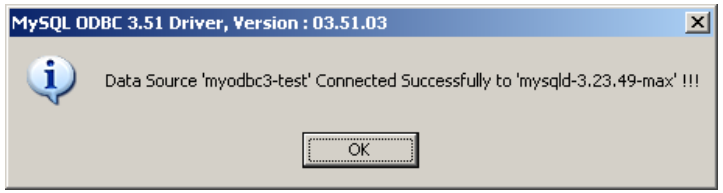
最后，对话框与下图显示的类似:



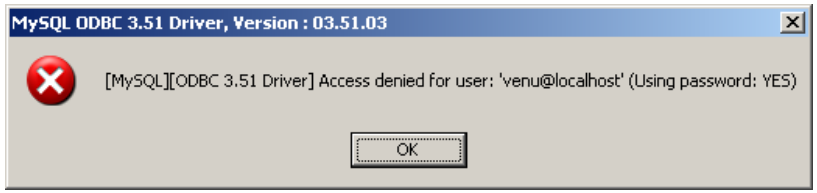
点击"OK"添加该数据源。

注释： 点击"OK"后，将打开"数据源"对话框，ODBC管理器将更新注册信息。连接到该数据源时，你所输入的用户名和连接字符串将成为该数据源的默认连接值。

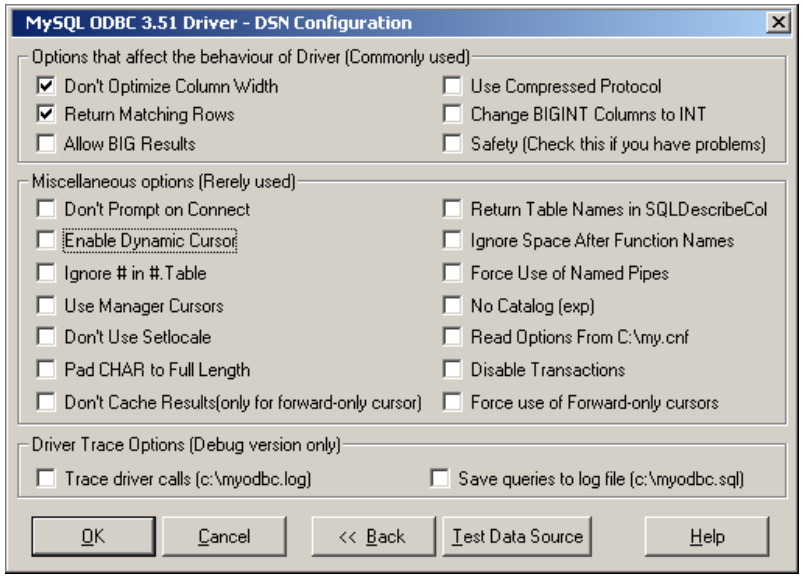
你也可以使用"测试数据源"按钮，测试你的设置是否适合于连接到服务器。该特性仅对MyODBC 3.51驱动程序有效。成功完成测试后，将显示下述窗口：



如果测试失败，将显示错误消息。



DNS配置对话框也有一个"选项"按钮。如果选择了它，将打开下述选项对话框，显示控制驱动程序的行为。关于这些选项的含义，请参见26.1.9.4节，"连接参数"。



注释： 在"驱动程序跟踪"选项下列出的选项已被禁止（灰色），除非你使用的是驱动DLL的调试版本。

要想在Windows平台上更改数据源：

1. 打开ODBC数据源管理器。点击恰当的选项卡"DSN"。
2. 选择打算更改的MySQL数据源，然后点击"配置"。打开"MySQL ODBC 3.51驱动程序-DSN配置"对话框。
3. 更改适用的数据源字段，然后点击"OK"。

更改完该对话框中的信息后，ODBC管理器将更新注册信息。

26.1.9.3. 在Unix平台上配置MyODBC DSN

在Unix平台上，可以直接在odbc.ini文件中配置DSN条目。这里给出了1个典型的odbc.ini文件，在该文件中，分别将myodbc和myodbc3配置为MyODBC 2.50和MyODBC 3.51的DSN名称：

```
;

;  odbc.ini对MyODBC和MyODBC 3.51驱动程序的配置

;

[ODBC Data Sources]
myodbc      = MyODBC 2.50 Driver DSN
myodbc3     = MyODBC 3.51 Driver DSN

[myodbc]
Driver       = /usr/local/lib/libmyodbc.so
Description  = MyODBC 2.50 Driver DSN
SERVER       = localhost
PORT         =
USER         = root
Password     =
Database     = test
OPTION       = 3
SOCKET       =

[myodbc3]
Driver       = /usr/local/lib/libmyodbc3.so
Description  = MyODBC 3.51 Driver DSN
SERVER       = localhost
```

```
PORT          =
USER          = root
Password      =
Database      = test
OPTION        = 3
SOCKET        =

[Default]
Driver        = /usr/local/lib/libmyodbc3.so
Description   = MyODBC 3.51 Driver DSN
SERVER        = localhost
PORT          =
USER          = root
Password      =
Database      = test
OPTION        = 3
SOCKET        =
```

关于可提供连接参数的清单，请参见26.1.9.4节，“连接参数”。

注释： 如果你正在使用unixODBC，可使用下述工具设置DSN：

- ODBCConfig GUI tool(HOWTO: ODBCConfig)
- odbcinst

在某些情况下使用unixODBC，可能会出现下述错误：

`Data source name not found and no default driver specified`（数据源名不存在，未指定默认驱动程序）

如果出现该情况，请确认ODBCINI和ODBCSYSINI环境变量指向正确的odbc.ini文件。例如，如果你的odbc.ini文件位于目录“/usr/local/etc”下，可将环境变量设为：

```
export ODBCINI=/usr/local/etc/odbc.ini
export ODBCSYSINI=/usr/local/etc
```

26.1.9.4. 连接参数

你可以在ODBC.INI文件的[Data Source Name]（数据源名）部分、或通过SQLDriverConnect() call的InConnectionString参量为MyODBC指定下述参数。

参数	默认值	注释
user	ODBC (on Windows)	用于链接至MySQL的用户名。
server	localhost	MySQL服务器的主机名。
database		默认数据库。
option	0	指定MyODBC工作方式的选项。参见下面。
port	3306	如果服务器不是本地主机将要使用的TCP/IP端口。
stmt		连接至MySQL时将要执行的语句。
password		服务器上用户账户的密码。
socket		当服务器是本地主机是将要连接的Unix套接字文件或Windows命名管道。

选项参量用于通知MyODBC：客户端不是100% ODBC兼容的。在Windows平台下，正常情况下，应通过切换连接屏幕上的复选框选择选项，但也能在选项参量中选择它们。下述选项是按照它们在MyODBC连接屏幕上显示的顺序排列的：

值	描述
1	客户端无法处理，MyODBC返回列的实际宽度。
2	客户端无法处理，MyODBC返回受影响行的真值。如果设置了该标志，MySQL将返回“发现的行”取而代之。MySQL的版本必须是3.21.14或更高版本，该功能才能生效。
4	在c:\myodbc.log中生成调试日志。它与将MYSQL_DEBUG=d:t:O,c::\myodbc.log放到AUTOEXEC.BAT中的效果相同（在Unix平台下，该文件是/tmp/myodbc.log）。
8	不为结果和参数设置任何信息报限制。

16	即使驱动程序可能会给出提示，对出现的问题不予提示。
32	允许或禁止动态光标支持。（在MyODBC 2.50中不允许）。
64	在db_name.tbl_name.col_name中忽略数据库名的使用。
128	强制使用ODBC管理器光标（实验性）。
256	禁止使用扩展取数据（实验性）。
512	将CHAR列填充为全列宽。
1024	SQLDescribeCol()返回完全合格的列名。
2048	使用压缩客户端／服务器协议。
4096	通知服务器忽略函数名之后和"("之前的空格（PowerBuilder要求这样）。这会使所有的函数名成为关键字。
8192	用命名管道链接至运行在NT环境下的**mysqld**服务器。
16384	将LONGLONG列更改为INT列（某些应用程序不能处理LONGLONG列）。
32768	从SQLTables返回作为Table_qualifier和Table_owner的用户（实验性）。
65536	从my.cnf的[client]和[odbc]组读取参数。
131072	增加一些额外检查（不应需要之，但...）。
262144	禁止事务。
524288	允许将查询记录到c:\myodbc.sql(/tmp/myodbc.sql)文件。（仅在调试模式下才能启用）。
1048576	不要驱动中的结果进行缓冲处理，而应从服务器读取"mysql_use_result()"。仅对正向光标才能起作用。当你不希望缓冲处理整个结果集时，对于大表处理，该选项十分重要。
2097152	强制使用正向光标类型。在应用程序设置了默认静态／动态光标类型的情况下，如果希望驱动程序使用非缓冲结果集，那么该选项能够保证正向光标的行为。

要想选择多个选项，可将它们的值加在一起。例如，将选项设置为12（4＋8），就能获得调试功能，但没有信息包限制。

默认的myodbc3.dll是为优化性能而编译的。如果希望调试MyODBC 3.51（例如，启用跟踪功能），应使用myodbc3d.dll。要想安装该文件，请拷贝myodbc3d.dll，使之覆盖已安装的myodbc3.dll文件。一旦完成了调试操作，务必恢复至驱动DLL的发布版本，这是因为调试版本可能会导致性能问题。注意，在MyODBC 3.51.07至3.51.11中未包含myodbc3d.dll。如果你正在使用这些版本中的一个，应从之前的版本（例如3.51.06）拷贝该DLL文件。

对于MyODBC 2.50，采用了myodbc.dll和myodbcd.dll取而代之。

在下面的表各中，给出了针对各种配置的推荐选项值：

配置	选项值
Microsoft Access	3
Microsoft Visual Basic	3
具有很多行的大表	2049
驱动跟踪生成（调试模式）	4
查询日志生成（调试模式）	524288
生成驱动跟踪和查询日志（调试模式）	524292
具有非缓冲结果的大表	3145731

26.1.9.5. 没有预定义DSN下的连接

是。通过指定DRIVER名称字段，可使用SQLDriverConnect连接到MySQL服务器。下面给出了使用DSN-Less连接的MyODBC连接字符串：

对于**MyODBC 2.50**：

```
ConnectionString = "DRIVER={MySQL};\
                    SERVER=localhost;\
                    DATABASE=test;\
```

```
                    USER=venu;\
                    PASSWORD=venu;\
                    OPTION=3;"
```

对于**MyODBC 3.51**：

```
ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};\
                    SERVER=localhost;\
                    DATABASE=test;\
                    USER=venu;\
                    PASSWORD=venu;\
                    OPTION=3;"
```

如果你使用的编程语言会将后跟空格的反斜杠转换为空格，最好将连接字符串指定为单个长字符串，或使用不会在其中添加空格的多个字符串串接。例如：

```
ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};"
                    "SERVER=localhost;"
                    "DATABASE=test;"
                    "USER=venu;"
                    "PASSWORD=venu;"
                    "OPTION=3;"
```

关于可提供连接参数的清单，请参见26.1.9.4节，“连接参数”。

26.1.9.6. 建立从系统A到系统B的远程连接

如果你打算使用myuser和mypassword作为用户名和密码从系统B连接到系统A，可参考下面给出的简单步骤。

在系统A上，执行下述步骤：

1. 启动MySQL服务器。
2. 使用GRANT建立用户名为myuser的账户，该账户可使用密码myuser从系统B建立连接。

```
3.            GRANT ALL ON *.* to 'myuser'@'B' IDENTIFIED BY 'mypassword';
```

4. GRANT语句为用户myuser授予了使用密码mypassword从系统B进行连接的所有权限。要想执行该语句，必须在系统A上拥有根用户权限，或是具有恰当权限的另一用户。关于MySQL权限的更多信息，请参见5.8节，“MySQL用户账户管理”。

在系统B上，执行下述步骤：

1. 使用下述连接参数配置MyODBC DSN：

```
2.          DSN            = remote_test
3.          SERVER or HOST = A (or IP address of system A)
4.          DATABASE       = test (The default database or an appropriate one)
5.          USER           = myuser
6.          PASSWORD       = mypassword
```

关于建立DSN-less连接的更多信息，请参见26.1.9.5节，“没有预定义DSN下的连接”。

7. 使用Ping命令或其它方式检查是否能从系统B访问系统A。如果无法访问系统A，请检查网络或Internet连接，或与你的系统管理员联系。

8. 尝试使用DSN=remote_test进行连接。如果失败，请跟踪查询MyODBC日志，并根据日志给出的错误信息采取进一步的步骤。如果需要进一步帮助，请发送详细的电子邮件至myodbc@lists.mysql.com。

在下述站点，你可以找到关于如何完成该操作的简单示例：http://www.phphelp.com/tutorial/using-myodbc-to-connect-to-a-remote-database.html.

26.1.9.7. 获取ODBC跟踪文件

如果遇到与MyODBC有关的困难或问题，首先应使用ODBC管理器和MyODBC生成一份日志文件（请求来自ODBC ADMIN的日志时获得的日志文件）。

要想通过驱动管理器获得ODBC跟踪文件，可采取下述步骤：

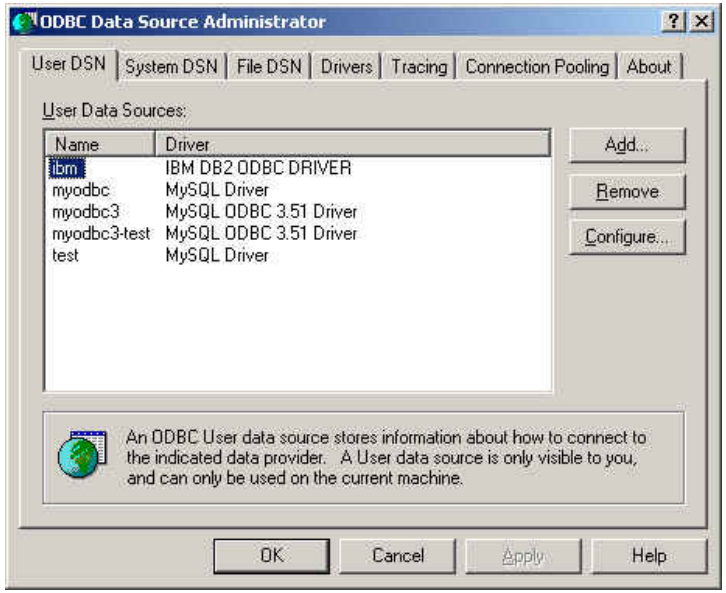
· 打开ODBC数据源管理器：

1. 点击"开始"，将指针指向"设置"，然后点击"控制面板"。

2. 在运行Microsoft Windows 2000、XP或2003的计算机上，双击"管理工具"，然后双击"数据源"（ODBC），如下图所示。

在运行早期Microsoft Windows版本的计算机上，双击"控制面板"中的32位ODBC或ODBC。

3. 打开ODBC数据源管理器对话框，如下图所示：

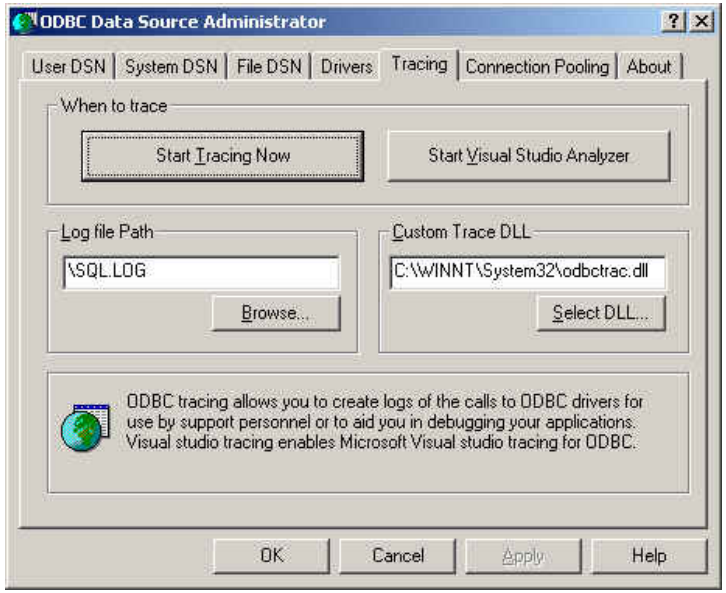


4. 点击"帮助"以了解ODBC数据源管理器对话框各选项卡的详细信息。

· 启用跟踪选项 对于Windows和Unix平台，该步骤不同。

要想在Windows平台上启用跟踪选项：

1. 通过"ODBC数据源管理器"对话框的"跟踪"选项卡，可对跟踪ODBC函数的方式进行配置。
2. 从"跟踪"选项卡激活了跟踪功能后，驱动管理器会对后续运行的所有应用程序的ODBC函数调用进行跟踪。
3. 激活跟踪功能前所运行应用程序的ODBC函数调用不会被记录。ODBC函数调用将被记录在你指定的日志文件中。
4. 点击"现在停止跟踪"后，跟踪功能将停止。请记住，启动跟踪功能后，日志文件将不断增大，而且跟踪功能会影响所有ODBC应用程序的性能。



要想在Unix平台上启用跟踪选项：

5. 在Unix平台上，需要在ODBC.INI文件中明确设置跟踪选项。

使用TraceFile和odbc.ini中的Trace（跟踪）参数打开或关闭跟踪功能，如下所示：

```
TraceFile  = /tmp/odbc.trace
Trace      = 1
```

TraceFile指明了跟踪文件的名称和完整路径，将Trace（跟踪）设为ON或OFF。也可以使用"1"或"Yes"表示ON，以及"0"或"No"表示OFF。如果正在使用unixODBC的**ODBCConfig**，然后遵照HOWTO-ODBCConfig中介绍的关于跟踪unixODBC调用的指示说明。

要想生成MyODBC日志，可采取下述步骤：

6. 确保你所使用的是驱动程序调试DLL（对于MyODBC 3.51，它是myodbc3d.dll而不是myodbc3.dll，对于MyODBC 2.50，它是myodbcd.dll）。

最简单的方法是从MyODBC 3.51分发版找到myodbc3d.dll（或myodbcd.dll），并用其覆盖myodbc3.dll（或myodbc.dll），该文件通常位于C:\windows\system32或C:\winnt\system32目录下。注意，完成测试后，你或许希望恢复旧的myodbc.dll文件，这是因为它比myodbc3d.dll（或myodbcd.dll）快很多，因此，请保存原始DLL的备份。

7. 在"MyODBC连接／配置"屏幕上启用"跟踪MyODBC"选项。日志将被写入文件C:\myodbc.log。当你返回上述屏幕时，如果你设置的跟踪选项未被记住，表明你正在使用的是myodbcd.dll驱动（参见前面的介绍）。在Linux平台上，或你使用的是DSN-Less连接，需在连接字符串中提供"OPTION=4"。

8. 启动应用程序，并尝试着使其出现问题。然后检查MyODBC跟踪文件，找出可能出错的地方。

如果发现某些事项出错，请发送电子邮件至myodbc@lists.mysql.com（或support@mysql.com，如果有与MySQL AB签订的支持合同），简要描述出现的问题，并提供下述额外

信息：

- MyODBC版本
- ODBC驱动管理器的类型和版本
- MySQL服务器的版本
- 驱动管理器的ODBC跟踪
- 来自MyODBC驱动的MyODBC日志文件
- 简单的可复制示例

请记住，你提供给我们的信息越多，我们更正问题的机会就越大。

此外，在提供缺陷信息前，请检查MyODBC邮件列表（http://lists.mysql.com/）。

26.1.9.8. 用MyODBC测试的应用程序

使用下述应用程序测试了MyODBC：

- MS Access 95, 97, 2000, and 2002
- C++-Builder, Borland Builder 4
- Centura Team Developer (formerly Gupta SQL/Windows)
- ColdFusion (on Solaris and NT with service pack 5), How-to: MySQL and Coldfusion. Troubleshooting Data Sources and Database Connectivity for UnixPlatforms.
- Crystal Reports
- DataJunction
- Delphi
- ERwin
- MS Excel
- iHTML
- FileMaker Pro
- FoxPro
- Notes 4.5/4.6
- MS Visio Enterprise 2000
- Vision
- Visual Objects
- Visual Interdev
- SBSS

- Perl DBD-ODBC
- Paradox
- Powerbuilder
- Powerdesigner 32-bit
- MS Visual C++
- Visual Basic
- ODBC.NET through CSharp(C#), VB and C++
- Data Architect(http://thekompany.com/products/dataarchitect/)
- SQLExpress for Xbase++(http://www.SQLExpress.net)
- Open Office (http://www.openoffice.org) How-to: MySQL + OpenOffice. How-to: OpenOffice + MyODBC + unixODBC.
- Star Office (http://wwws.sun.com/software/star/staroffice/6.0/index.html)
- G2-ODBC bridge (http://www.gensym.com)
- Sambar Server (http://www.sambarserver.info) How-to: MyODBC + SambarServer + MySQL.

如果你知道能够与MyODBC一起工作的其他应用程序，请以电子邮件的方式指明它：myodbc@lists.mysql.com。

26.1.9.9. 已知的能与MyODBC一起工作的程序

大多数程序均能与MyODBC一起工作，对上面所列的每一程序，我们自己进行了测试，或得到用户的确认。很多介绍中均给出了你可能会遇到问题的描述。

· 程序

注释

· Access

要想使Access工作：

o 如果你正在使用Access 2000，应从下述地址获取并安装最新的（2.6版或更高）Microsoft MDAC（Microsoft数据访问组件），http://www.microsoft.com/data/。它更正了Access在将数据导出至MySQL时存在的一个缺陷，未指定表名和列名。另一种解决该缺陷的方法是，升级到MyODBC 2.50.33和MySQL 3.23.x, 它们共同提供了避免该问题的一种方式。

此外，你还应获取并应用Microsoft Jet 4.0 Service Pack 5 (SP5)，可在下述地址找到它：http://support.microsoft.com/default.aspx?scid=kb;EN-US;q239114。它修正了某些情况下在Access中列被标注为"#DELETED#"的问题。

注释： 如果你正使用MySQL 3.22，必须安装MDAC补丁，并使用MyODBC 2.50.32或2.50.34或更高版本以解决该问题。

o 对于所有版本的Access，应启用"MyODBC返回匹配行"选项。对于Access 2.0，还应额外启用"模拟ODBC 1.0"选项。

o 在希望能够更新的所有表中，均应有时间戳。为了获得最大的可移植性，在列声明中不要使用长度规范。也就是说，应使用TIMESTAMP，而不是TIMESTAMP(*n*), *n* < 14。

o 在表中应有1个主键。如不然，新的或更新的行可能会显示为"#DELETED#"。

o 仅应使用DOUBLE浮点字段。与单精度浮点进行比较时，Access将失败。其征兆是新的或更新的行可能会显示为"#DELETED#"，或无法找到或更新行。

- 如果你正使用MyODBC来链接到有BIGINT列的表，结果会显示为"#DELETED"。排除它的解决方案是：
  - 有1个以TIMESTAMP作为数据类型的虚拟列。
  - 在"ODBC DSN管理器"的连接对话框中选择"将BIGINT列更改为INT"选项。
  - 删除与Access的表链接，并重新创建它。

旧记录仍将显示为"#DELETED#"，但新增／更新的记录会恰当显示。

- 添加了TIMESTAMP列后，另一位用户更改了数据，如果错误依旧出现，下述技巧或许有所帮助：

不要使用表数据表视图。取而代之的是，从你希望使用的表创建一个表单，并使用表单数据表视图。应将TIMESTAM列的DefaultValue属性设置为NOW()。在视图中隐藏TIMESTAMP列或许是个好主意，这样就不会使你的用户感到迷惑。

- 在某些情况下，Access可能会生成MySQL无法理解的SQL语句。可通过在Access菜单中选择"Query|SQLSpecific|Pass-Through"来更正该问题。
- 在NT平台上，Access会将BLOB列通报为OLE OBJECTS（OLE对象）。如果你打算用MEMO列取而代之，应使用ALTER TABLE将BLOB列更改为TEXT。
- Access无法在任何时候均恰当处理DATE列。如果遇到这类问题，请将列更改为DATETIME。
- 如果在Access中存在定义为BYTE的列，Access会视图将其导出为TINYINT而不是TINYINT UNSIGNED。如果列中的值大于127，将出现问题。

· ADO

使用ADO API和MyODBC进行编码时，需要注意某些不被MySQL服务器支持的默认属性。例如，对于RecordCount属性，如果将CursorLocation属性用作adUseServer，将返回结果"-1"。要想获得正确的值，需要将该属性设置为adUseClient，如下面给出的VB代码示例所示：

```
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
```

另一种处理方式是，对类似查询使用SELECT COUNT(*)语句以获取正确的行计数。

· 主动服务器页（ASP）

应选择"返回匹配行"选项。

· BDE应用程序

要想使这类应用程序工作，应选择"不优化列宽度并返回匹配行"选项。

· Borland Builder 4

开始查询时，可使用Active属性或Open方法。注意，Active将通过自动发出"SELECT * FROM ..."查询开始。如果表很大，这不是什么好事。

· ColdFusion（在Unix平台上）

下述信息取自ColdFusion文档：

使用下述信息来配置用于Linux的ColdFusion服务器，以便使用针对MySQL数据源的unixODBC驱动和MyODBC。Allaire已证明，MyODBC 2.50.26能够与MySQL 3.22.27以及用于Linux的ColdFusion一起工作。（任何较新的版本也应能正确工作）。你可以在网站http://dev.mysql.com/downloads/connector/odbc/上下载MyODBC。

通过ColdFusion 4.5.1版，可以使用"ColdFusion管理器"来添加MySQL数据源。但是，驱动程序未包含在ColdFusion 4.5.1版中。在MySQL驱动程序出现在ODBC数据源下拉列表之前，必须创建MyODBC驱动程序，并将其拷贝到/opt/coldfusion/lib/libmyodbc.so。

在Contrib目录下包含程序mydsn-*xxx*.zip，使用它，对于Coldfusion应用程序，可创建并删除用于MyODBC驱动的DSN注册文件。

· DataJunction

应对其进行更改，使之输出VARCHAR而不是ENUM，因为其导出ENUM的方式会造成MySQL问题。

· Excel

工作。一些提示：

o 如果遇到日期方面的问题，请使用CONCAT()函数，将其选择为字符串。例如：

```
o                    SELECT CONCAT(rise_time), CONCAT(set_time)
o                      FROM sunrise_sunset;
```

采用该方式以字符串提取的值应能被Excel97正确识别为时间值。

在本例中，CONCAT()的目的是让ODBC认为列是"字符串类型"。如果没有CONCAT()，ODBC会将列视为时间类型，Excel无法理解它。

注意，Excel存在1个缺陷，这是因为它会自动将字符串转换为时间。如果源是文本文件，不存在问题，但当源是通报各列准确类型的ODBC连接时，将出现问题。

· Word

要想将数据从MySQL提取到Word/Excel文档，需要使用MyODBC驱动程序以及"Microsoft查询帮助"插件。

例如，用含有两列文本的表创建1个数据库：

o 使用**mysql**客户端命令行工具插入行。

o 使用ODBC管理器创建1个DSN文件，例如，针对刚创建数据库的"my"。

o 打开Word应用程序。

o 创建1个新的空白文档。

o 在数据库工具栏上，按"插入数据库"按钮。

o 按"获取数据"按钮。

o 在"获取数据"屏幕右侧，按"Ms Query"按钮。

- 在"Ms Query"中使用"my DSN"文件创建1个新数据源。
- 选择新查询。
- 选择打算使用的列。
- 如果愿意，创建1个过滤器。
- 如果愿意，创建1个分类。
- 选择"将数据返回到Microsoft Word"。
- 点击"完成"。
- 点击"插入数据"并选择记录。
- 点击OK，在你的Word文档中将看到插入的行。

· odbcadmin

ODBC的测试程序。

· Delphi

必须使用BDE 3.2版或更新的版本。连接到MySQL时，选择"不优化列宽度"选项。

此外，这里给出了一些可能有用的Delphi代码，这些代码可设置为MyODBC设置ODBC条目和BDE条目。BDE条目要求用到"BDE别名编辑器"，它位于靠近你的"Delphi Super Page"上，可自由拖动。（下述内容由Bryan Brunton <bryan@flesherfab.com>提供）：

```
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

用BDE 3.0版进行了测试。目前已知的唯一问题是，更改表方案时，查询字段不更新。然而，BDE看上去不会识别主键，它仅是名为PRIMARY的索引，尽管这谈不上是问题。

- Vision

应选择"返回匹配行"选项。

- Visual Basic

要想更新表，必须为表定义主键。

带有ADO的Visual Basic不能处理大整数。这意味着某些查询（如SHOW PROCESSLIST等）不会正确工作。更正方法是，在ODBC连接字符串中使用OPTION=16384，或在MyODBC连接屏幕上选择"将BIGINT列更改为INT"选项。或许，你也希望选择"返回匹配行"选项。

- VisualInterDev

如果在结果中有BIGINT，可能会出现错误"[Microsoft][ODBC Driver Manager]驱动程序不支持该参数"。请在MyODBC连接屏幕上选择"将BIGINT列更改为INT"选项。

- Visual Objects

应选择"不优化列宽度"选项。

- MS Visio Enterprise 2000

通过MyODBC（2.50.37或更高版本），通过连接MS Vision Enterprise 2000和MySQL，并使用Visio的逆向工程师功能，我们建立了数据库模型，使用它来检索关于DB的信息（Visio显示了所有的列定义、主键、索引等）。此外，我们还通过指定Visio中的新表进行了测试，并通过MyODBC将其导出至MySQL。

26.1.10. 与MyODBC连接相关的事宜

26.1.10.1. 配置MyODBC DSN时，出现不能加载翻译器或设置库错误
26.1.10.2. 连接时，出现拒绝访问错误
26.1.10.3. INFO：关于ODBC连接池

在本节中，回答了与MyODBC连接有关的问题。

26.1.10.1. 配置MyODBC DSN时，出现不能加载翻译器或设置库错误

更多信息，请参见MS知识库文章(Q260558)。此外，请确认在你的系统目录下有最新的有效ctl3d32.dll文件。

26.1.10.2. 连接时，出现拒绝访问错误

请参见5.7.8节，“拒绝访问错误的原因”。

26.1.10.3. INFO：关于ODBC连接池

关于连接池方面的信息，请参阅下述文档： http://support.microsoft.com/default.aspx?scid=kb;EN-US;q169470。

26.1.11. MyODBC和Microsoft Access

26.1.11.1. 如何设置Microsoft Access，使之能够与使用MyODBC的MySQL一起工作？
26.1.11.2. 如何将表或查询从Access导出到MySQL？
26.1.11.3. 如何导入MySQL数据库表或将其链接到Access？
26.1.11.4. 链接表的结构或位置已改变，我能看到链接表中的这些变化吗？
26.1.11.5. 当我在链接表中插入记录或更新其中的记录时，遇到“#DELETED#”
26.1.11.6. 如何处理写冲突或行位置错误？
26.1.11.7. 无论何时，当我从Access 97导出表时，出现陌生的语法错误
26.1.11.8. 编辑记录时，Access返回“另一用户更改了你修改的记录”
26.1.11.9. 如何在Access中俘获ODBC登录错误消息？
26.1.11.10. 如何优化Access与MyODBC一起工作的性能？
26.1.11.11. 我有很长的表，MyODBC访问这些Access表的最佳配置是什么？
26.1.11.12. 如何为ODBC连接设置QueryTimeout值？
26.1.11.13. INFO：用于Access和MySQL间导入/导出的工具

在本节中，回答了与MyODBC和Microsoft Access有关的问题。

26.1.11.1. 如何设置Microsoft Access，使之能够与使用MyODBC的MySQL一起工作？

要想使Microsoft Access能够与MyODBC一起工作，在你的客户端PC上必须完成下述操作。

1. 如果你正在使用Access 2000，应从下述地址获取并安装最新的（2.6版或更高）Microsoft MDAC（Microsoft数据访问组件），http://www.microsoft.com/data/。它更正了Access在将数据导出至MySQL时存在的一个缺陷，未指定表名和列名。另一种解决该缺陷的方法是，升级到MyODBC 2.50.33和MySQL 3.23.x, 它们共同提供了避免该问题的一种方式。

此外，你还应获取并应用Microsoft Jet 4.0 Service Pack 5 (SP5)，可在下述地址找到它：http://support.microsoft.com/default.aspx?scid=kb;EN-US;q239114。它修正了某些情况下在Access中列被标注为“#DELETED#”的问题。

注释： 如果你正使用MySQL 3.22，必须安装MDAC补丁，并使用MyODBC 2.50.32或2.50.34或更高版本以解决该问题。

2. 安装最新版MySQL，http://dev.mysql.com/downloads/。

3. 安装最新版MyODBC 3.51或2.50，http://dev.mysql.com/downloads/connector/odbc/。

4. 对于所有版本的Access，应启用“MyODBC返回匹配行”选项。

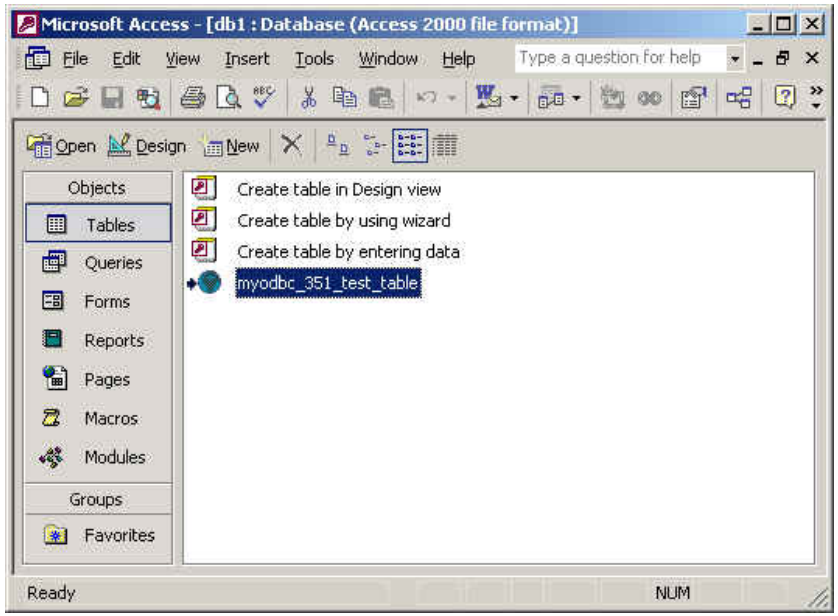
5. 通过MyODBC，将Access用作MySQL服务器的前端程序。

26.1.11.2. 如何将表或查询从Access导出到MySQL？

除非已安装了MyODBC，否则不能将表或查询导出到MySQL。

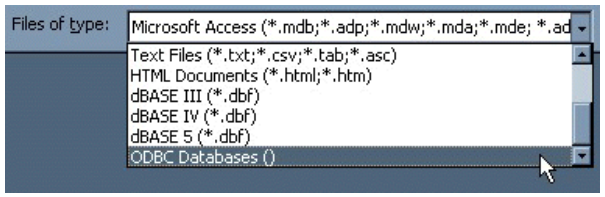
要想将表从Access导入MySQL，请遵循下述说明：

1. 打开Access数据库或Access项目时，出现"数据库"窗口。其中显示了用于创建新数据库对象和打开已有对象的快捷方式。



2. 点击打算导出的表名或查询名，然后在"文件"菜单中选择"导出"。

3. 在"导出对象类型对象名至"对话框中，在"另存为类型"框中，选择"ODBC数据库()"，如下图所示：



4. 在"导出"对话框中，输入文件名（或使用建议的文件名），然后选择OK。

5. 显示"选择数据源"对话框，其中列出了为计算机上已安装的各ODBC驱动定义的数据源。点击"文件数据源"或"机器数据源"选项卡，然后双击打算导出至的MyODBC或MyODBC 3.51数据源。关于为MyODBC定义新数据源的方法，请参见26.1.9.2节，"在Windows上配置MyODBC DSN"。

Microsoft Access通过该数据源连接至MySQL服务器，并导出新的表和／或数据。

26.1.11.3. 如何导入**MySQL**数据库表或将其链接到**Access**？

除非已安装了MyODBC，否则不能将表或查询导出到MySQL数据库。

要想将表从**MySQL**导入或链接到**Access**，请采取下述步骤：

1. 打开数据库，或切换到"数据库"窗口以打开数据库。

2. 要想导入表，在"文件"菜单上，将鼠标指针指向"获取外部数据"，然后点击"导入"。要想链接表，在"文件"菜单上，将鼠标指针指向"获取外部数据"，然后点击"链接表"。

3. 在"导入"（或"链接"）对话框中，在"文件类型"框中选择"ODBC Databases ()"。在"选择数据源"对话框中，列出了定义的数据源。显示"选择数据源"对话框，其中列出了为安装在计算机上的任何ODBC驱动定义的数据源。点击"文件数据源"或"机器数据源"选项卡，然后双击打算导出至的MyODBC或MyODBC 3.51数据源。关于为MyODBC或MyODBC 3.51驱动定义新数据源的方法，请参见26.1.9.2节，"在Windows上配置MyODBC DSN"。

4. 如果所选的数据源要求登录，请输入登录ID和密码（可能还需要额外信息），然后点击OK。

5. Microsoft Access通过ODBC数据源连接到MySQL服务器，并显示可导入或链接的表清单。

6. 点击希望导入或链接的每个表，然后点击OK。如果你正在链接1个表，但它没有唯一识别各条记录的索引，Microsoft Access将显示链接表中的字段列表。点击能唯一标识各记录的字段或字段组合，然后点击OK。

26.1.11.4. 链接表的结构或位置已改变，我能看到链接表中的这些变化吗？

是。当链接表的结构或位置发生变化时，可采取下述步骤查看或刷新链接。"链接表管理器"列出了当前链接的所有表的路径。

要想查看或刷新链接：

1. 打开包含表链接的数据库。

2. 在"工具"菜单上，指向"加载项"（在Access 2000或更新版本中为"数据库实用工具"），然后点击"链接表管理器"。

3. 选中打算刷新链接的表的复选框。

4. 点击OK，刷新链接。

Microsoft Access将确认成功的刷新操作，或者，如果未找到表，将显示"选择<table name>新位置"对话框，在该对话框中，可指定表的新位置。如果你所选择的数个表已被移至你所指定的新位置，链接表管理器将针对所有所选的表搜索该位置，并一次性地更新所有链接。

要想更改链接表集合的路径：

1. 打开包含表链接的数据库。

2. 在"工具"菜单上，指向"加载项"（在Access 2000或更新版本中为"数据库实用工具"），然后点击"链接表管理器"。

3. 选中"对新位置始终提示"复选框。

4. 选中打算更改链接的表的复选框，然后点击OK。

5. 在"选择<table name>新位置"对话框中，指定新位置，点击"打开"，然后点击OK。

26.1.11.5. 当我在链接表中插入记录或更新其中的记录时，遇到**"#DELETED#"**

如果在Access中插入或更新的记录显示为"#DELETED#"：

· 如果你正在使用Access 2000，应从下述地址获取并安装最新的（2.6版或更高）Microsoft MDAC（Microsoft数据访问组件），http://www.microsoft.com/data/。它更正了Access在将数据导出至MySQL时存在的一个缺陷，未指定表名和列名。另一种解决该缺陷的方法是，升级到MyODBC 2.50.33和MySQL 3.23.x, 它们共同提供了避免该问题的一种方式。

此外，你还应获取并应用Microsoft Jet 4.0 Service Pack 5 (SP5)，可在下述地址找到它：http://support.microsoft.com/default.aspx?scid=kb;EN-US;q239114。它修正了某些情况下在Access中列被标注为"#DELETED#"的问题。

注释： 如果你正使用MySQL 3.22，必须安装MDAC补丁，并使用MyODBC 2.50.32或2.50.34或更高版本以解决该问题。

· 对于所有版本的Access，应启用"MyODBC返回匹配行"选项。对于Access 2.0，还应额外启用"模拟ODBC 1.0"选项。

- 在希望能够更新的所有表中，均应有时间戳。为了获得最大的可移植性，在列声明中不要使用长度规范。也就是说，应使用TIMESTAMP，而不是TIMESTAMP(*n*), *n* < 14。
- 在表中应有1个主键。如不然，新的或更新的行可能会显示为"#DELETED#"。
- 仅应使用DOUBLE浮点字段。与单精度浮点进行比较时，Access将失败。其征兆是新的或更新的行可能会显示为"#DELETED#"，或无法找到或更新行。
- 如果你正使用MyODBC来链接到有BIGINT列的表，结果会显示为"#DELETED"。排除它的解决方案是：
  - 有1个以TIMESTAMP作为数据类型的虚拟列。
  - 在"ODBC DSN管理器"的连接对话框中选择"将BIGINT列更改为INT"选项。
  - 删除与Access的表链接，并重新创建它。

旧记录仍将显示为"#DELETED#"，但新增／更新的记录会恰当显示。

26.1.11.6. 如何处理写冲突或行位置错误？

如果看到下述错误，请在"DSN配置"对话框中选择"返回匹配行"选项，或将连接参数指定为"OPTION=2"。

写冲突。另一用户更改了你的数据。

无法找到需更新行的位置。自上次读取操作以来，某些值可能已被改变。

26.1.11.7. 无论何时，当我从**Access 97**导出表时，出现陌生的语法错误

对于Access 97，这是件奇怪的事宜，但在Access 2000或2002中并未出现。将MyODBC升级至MyODBC 3.51.02或以上，可以克服该问题。

26.1.11.8. 编辑记录时，**Access**返回"另一用户更改了你修改的记录"

对于某些程序，可能会出现该错误： 另一用户更改了你所修改的记录。在大多数情况下，可通过下述措施解决该问题：

- 如果主键不存在，为表添加1个主键。
- 如果时间戳不存在，添加1个时间戳列。
- 仅应使用DOUBLE浮点字段。与单精度浮点值比较时，某些程序会出错。

如果这些措施未能解决问题，首先应从ODBC管理器生成1个日志文件（请求来自ODBC ADMIN的日志时获得的日志文件），以及1个MyODBC日志，使用它们找出出错的原因。具体介绍，请参见26.1.9.7节，"获取ODBC跟踪文件"。

26.1.11.9. 如何在**Access**中俘获**ODBC**登录错误消息？

请阅读"如何在Access中俘获ODBC登录错误消息"，http://support.microsoft.com/support/kb/articles/Q124/9/01.asp?LN=EN-US&SD=gn&FR=0%3CP%3E。

26.1.11.10. 如何优化**Access**与**MyODBC**一起工作的性能？

- 优化客户端／服务器性能
- 关于转换应用程序以使用ODBCDirect的技巧
- 关于在所附的SQL表上优化查询的技巧

26.1.11.11. 我有很长的表，MyODBC访问这些Access表的最佳配置是什么？

如果在Access中有很大（长）的表，可能需要相当长的时间才能打开。或者，也可能是虚拟内存较低的情况下运行，最终导致ODBC查询失败错误，并无法打开表。为了解决该问题，可选择下述选项：

- 返回匹配行（2）
- 允许BIG结果（8）

这样可将值增加到10（OPTION=10）。

26.1.11.12. 如何为ODBC连接设置QueryTimeout值？

请参阅"为ODBC连接设置QueryTimeout值"，http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B153756。

26.1.11.13. INFO：用于Access和MySQL间导入/导出的工具

关于可用工具列表，请参阅http://www.mysql.com/portal/software/convertors/一节。

26.1.12. MyODBC和Microsoft VBA及ASP

26.1.12.1. 为什么SELECT COUNT(*) FROM tbl_name返回错误？
26.1.12.2. 无论何时，当我使用AppendChunk()或GetChunk() ADO方法时，遇到错误"多步操作导致错误，请检查每个状态值"
26.1.12.3. 在ADO中如何发现受特定SQL语句影响的总行数？
26.1.12.4. 在Visual Basic中如何处理Blob数据？
26.1.12.5. 如何将Visual Basic的数据类型映射到MySQL类型？
26.1.12.6. 示例：VB与ADO、DAO和RDO
26.1.12.7. ASP和MySQL以及MyODBC
26.1.12.8. INFO：关于ActiveX数据对象（ADO）的常见问题

在本节中，回答了与Microsoft Visual Basic(ADO, DAO & RDO)和ASP一起使用MyODBC有关的问题。

26.1.12.1. 为什么SELECT COUNT(*) FROM tbl_name返回错误？

这是因为COUNT(*)表达式返回了1个BIGINT，ADO不理解这个大值的含义。选择"将BIGINT列更改为INT"选项（选项值16384）。

26.1.12.2. 无论何时，当我使用AppendChunk()或GetChunk() ADO方法时，遇到错误"多步操作导致错误，请检查每个状态值"

将光标位置指定为adUseServer时，ADO的GetChunk()和AppendChunk()方法不能按预期的方式工作。从另一方面上讲，可使用adUseClient克服该问题。

在http://www.dwam.net/iishelp/ado/docs/adomth02_4.htm上给出了一个简单示例。

26.1.12.3. 在ADO中如何发现受特定SQL语句影响的总行数？

在ADO执行方法中使用RecordsAffected属性。关于使用执行方法的更多信息，请参见http://msdn.microsoft.com/library/default.asp?url=/library/en-us/ado270/htm/mdmthcnnexecute.asp。

26.1.12.4. 在Visual Basic中如何处理Blob数据？

下面给出了Mike Hillyer（m.hillyer@telusplanet.net）写的一篇好文章，其中解释了如何在ADO中通过MyODBC插入数据和／或从Blob列获取数据的方法。MySQL BLOB列和Visual Basic 6。

26.1.12.5. 如何将Visual Basic的数据类型映射到MySQL类型？

下面给出了Mike Hillyer（m.hillyer@telusplanet.net）的另一篇好文章。如何将Visual basic数据类型映射为MySQL类型。

26.1.12.6. 示例：VB与ADO、DAO和RDO

下面给出了ADO、DAO和RDO与VB一起使用的用法示例：

- ADO示例： 26.1.19节，“MyODBC与VB：ADO、DAO和RDO”
- DAO示例： 26.1.19节，“MyODBC与VB：ADO、DAO和RDO”
- RDO示例： 26.1.19节，“MyODBC与VB：ADO、DAO和RDO”

如果你有其他好的例子，或关于ADO/DAO/RDO的基本知识，请将详情发送至myodbc@lists.mysql.com。

26.1.12.7. ASP和MySQL以及MyODBC

关于如何使用MyODBC通过ASP访问MySQL的更多信息，请参阅下述文章：

- 使用MyODBC通过ASP访问你的MySQL数据库
- ASP and MySQL at DWAM.NT

在http://support.microsoft.com/default.aspx?scid=/Support/ActiveServer/faq/data/adofaq.asp中，给出了关于ASP的常见问题清单。

26.1.12.8. INFO：关于ActiveX数据对象（ADO）的常见问题

更多信息，请参见ActiveX数据对象（ADO）常见问题。

26.1.13. MyODBC和第三方ODBC工具

26.1.13.1. 如何将数据从MySQL提取到MS-Word/Excel文档？
26.1.13.2. 使用MyODBC将表从MS DTS导出到MySQL时出现语法错误
26.1.13.3. 如何在Solaris平台上配置MySQL+MyODBC+unixODBC+ColdFusion

在本节中，回答了与MyODBC和各种ODBC相关工具有关的问题，如Microsoft Word、Excel和ColdFusion。

26.1.13.1. 如何将数据从MySQL提取到MS-Word/Excel文档？

要想将数据从MySQL提取到Word/Excel文档，需要使用MyODBC驱动程序以及“Microsoft查询帮助”插件。

例如，用含有两列文本的表创建1个数据库：

- 使用**mysql**客户端命令行工具插入行。
- 使用ODBC管理器创建1个DSN文件，例如，针对刚创建数据库的“my”。
- 打开Word应用程序。
- 创建1个新的空白文档。
- 在数据库工具栏上，按“插入数据库”按钮。

- 按"获取数据"按钮。
- 在"获取数据"屏幕右侧，按"Ms Query"按钮。
- 在"Ms Query"中使用"my DSN"文件创建1个新数据源。
- 选择新查询。
- 选择打算使用的列。
- 如果愿意，创建1个过滤器。
- 如果愿意，创建1个分类。
- 选择"将数据返回到Microsoft Word"。
- 点击"完成"。
- 点击"插入数据"并选择记录。
- 点击OK，在你的Word文档中将看到插入的行。

26.1.13.2. 使用MyODBC将表从MS DTS导出到MySQL时出现语法错误

这与当表由TEXT或VARCHAR数据类型构成时Access 97遇到的问题类似。通过将MyODBC驱动升级到3.51.02或更高版本，即可排除该错误。

26.1.13.3. 如何在Solaris平台上配置MySQL+MyODBC+unixODBC+ColdFusion

请参见MySQL ColdFusion unixODBC MyODBC和Solaris：如何成功

26.1.14. MyODBC通用功能

26.1.14.1. 如何在ODBC中获取AUTO_INCREMENT列的值
26.1.14.2. MyODBC支持动态光标类型吗？
26.1.14.3. 导致事务无法启用错误的原因是什么？
26.1.14.4. 导致无法找到光标错误的原因是什么？
26.1.14.5. 我能与MyODBC 3.51一起使用MyODBC 2.50应用程序吗？
26.1.14.6. 我能使用MyODBC从.NET环境访问MySQL吗？
26.1.14.7. MyODBC的性能为什么很差，对于相对较小的查询也会导致大量的磁盘动作？

在本节中，回答了与MyODBC一般功能有关的问题。

26.1.14.1. 如何在ODBC中获取AUTO_INCREMENT列的值

一个常见问题是，如何获取从INSERT语句自动生成的ID的值。使用ODBC，你可以作与以下示例类似的任何事（假定"auto"为AUTO_INCREMENT字段）：

```
INSERT INTO tbl (auto,text) VALUES(NULL,'text');
SELECT LAST_INSERT_ID();
```

或者，如果你仅打算将ID插入到另一表中，你可以：

```
INSERT INTO tbl (auto,text) VALUES(NULL,'text');
INSERT INTO tbl2 (id,text) VALUES(LAST_INSERT_ID(),'text');
```

请参见25.2.13.3节，“如何获得上次插入行的唯一ID”。

为了使某些ODBC应用程序（至少是Delphi和Access）获得更好的性能，可使用下述查询来找到新插入的行：

```
SELECT * FROM tbl WHERE auto IS NULL;
```

26.1.14.2. MyODBC支持动态光标类型吗？

是。MyODBC 3.51支持动态光标类型以及正向和静态特性。

由于性能方面的原因，在默认情况下，驱动程序不支持该特性。你可以启用该特性，方法是，将连接选项标志指定为“OPTION=32”，或在DSN配置中选中“启用动态光标”选项。

26.1.14.3. 导致事务无法启用错误的原因是什么？

当应用程序发出事务调用，但底层MySQL服务器不支持事务或禁止事务时，驱动程序将返回该错误。

为了避免该问题，必须使用启用了InnoDB或BDB存储引擎（或两者）的服务器，并使用这类表。从4.0版以后，默认情况下，MySQL服务器均支持InnoDB。在BDB可用的平台上，MySQL-Max服务器也支持BDB。

此外，如果你的服务器支持事务表类型（InnoDB和BDB），请确保在DSN配置中未设置“禁止事务”选项。

26.1.14.4. 导致无法找到光标错误的原因是什么？

这是因为应用程序正在使用旧的MyODBC 2.50版本，不能通过SQLSetCursorName明确设置光标名称。更正它的方法是升级到MyODBC 3.51版。

26.1.14.5. 我能与MyODBC 3.51一起使用MyODBC 2.50应用程序吗？

是。如果你发现不能与MyODBC 3.51一起工作，但能与MyODBC 2.50一起工作的任何事项，请发送电子邮件至myodbc@lists.mysql.com。

26.1.14.6. 我能使用MyODBC从.NET环境访问MySQL吗？

是。你可以使用**odbc.net**，通过MyODBC连接到MySQL。这里给出了一些从VC.NET和VB.NET连接到MySQL的基本示例。

- 请参见26.1.20.1节，“ODBC.NET: CSHARP(C#)”
- 请参见26.1.20.2节，“ODBC.NET: VB”

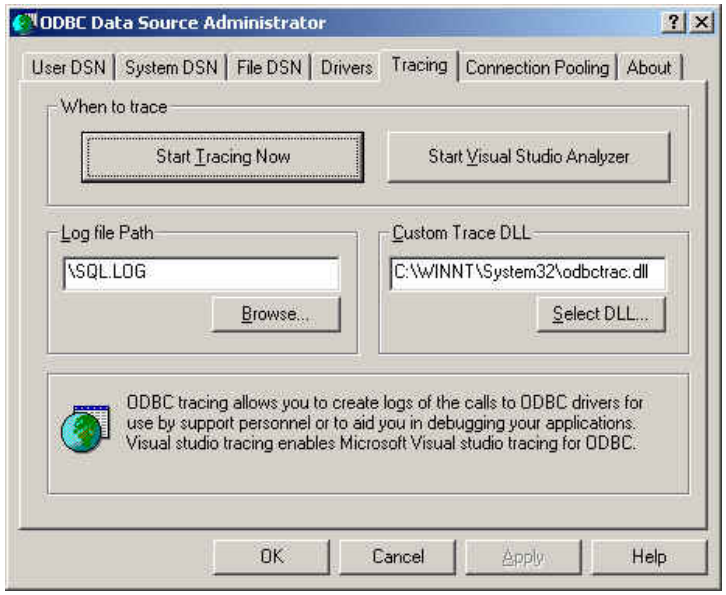
这里给出了**Venu**（MyODBC开发人员）撰写的另一篇好文章http://www.mysql.com/newsletter/2002-07/a0000000035.html，其中，给出了所有的MySQL .NET接口以及一些有用的例子。

注意： 在与MyODBC一起使用ODBC.NET的过程中，在获取空字符串的同时（长度为0），将给出SQL_NO_DATA异常。从站点http://support.microsoft.com/default.aspx?scid=kb;EN-US;q319243，可获取针对它的补丁。

26.1.14.7. MyODBC的性能为什么很差，对于相对较小的查询也会导致大量的磁盘动作？

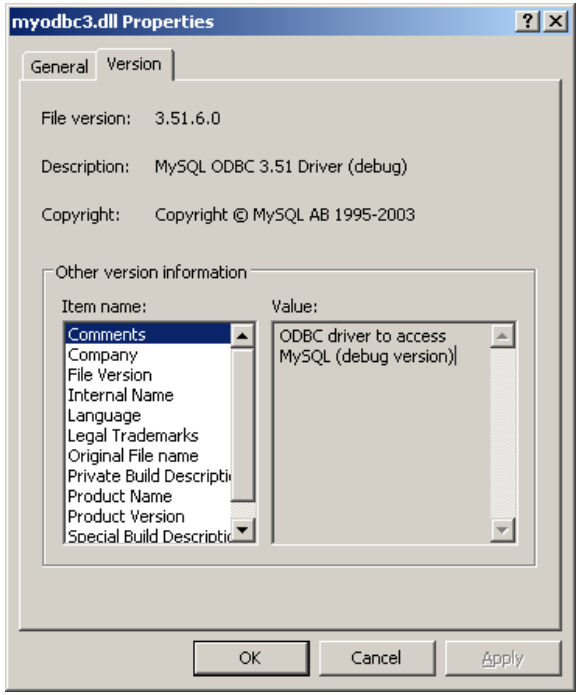
MyODBC比其他ODBC驱动程序快很多。缓慢可能是因未使用下述选项造成的：

- 打开“ODBC跟踪”选项。遵循这里给出的指示说明，交叉检查是否未启用该选项。

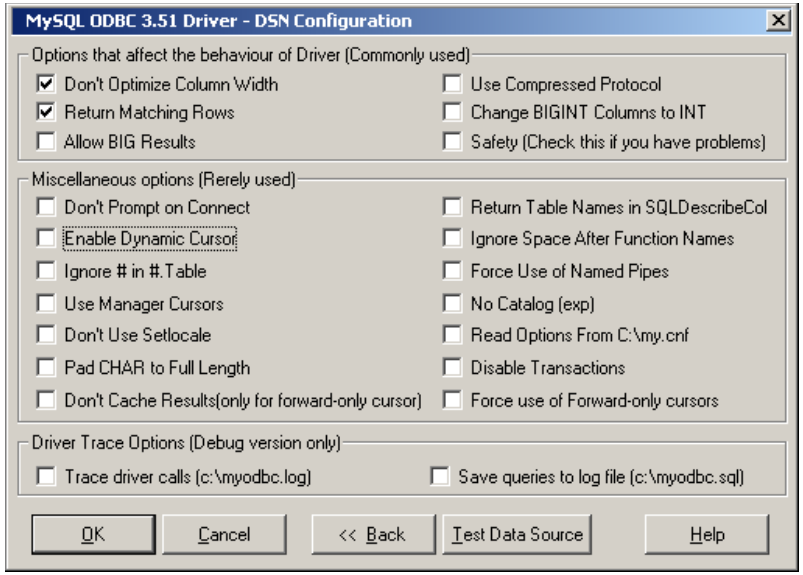


如上图所示，“ODBC数据源管理器”“跟踪”选项卡的“何时跟踪”选项应始终指向“现在开始跟踪”，而不是“现在停止跟踪”。

- 使用了驱动程序的调试版本。如果你正在使用驱动DLL的调试版本，也会使查询处理变慢。你可以执行交叉检查，通过驱动DLL属性（在系统目录下，右击驱动DLL并点击“属性”）的“注释”区，检查DLL是否是调试版或发布版，如下图所示：



- 启用了“驱动跟踪和查询日志”。即使你打算使用驱动程序的调试版（在生产环境下总应使用发布版），也应确保禁止了“驱动跟踪和查询日志”选项（OPTION=4,524288），如下图所示：

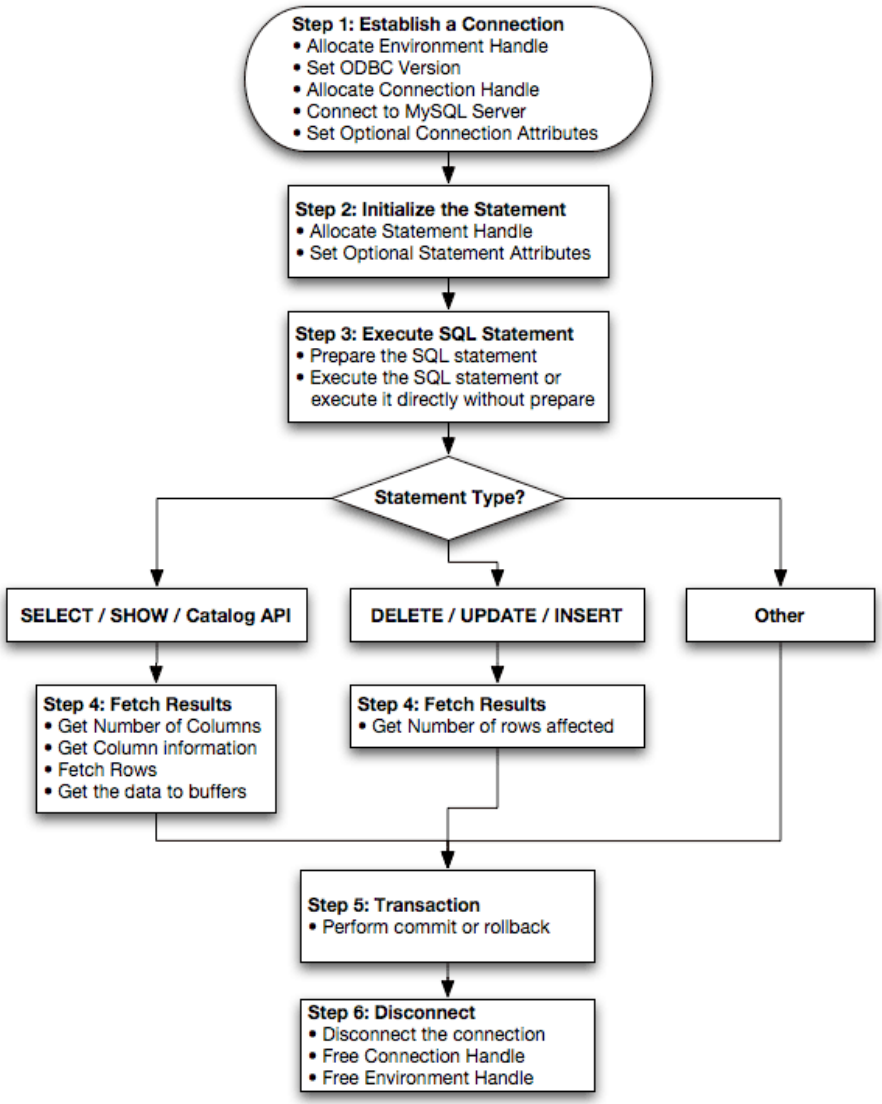


26.1.15. 基本的MyODBC应用步骤

从MyODBC应用程序执行的与MySQL服务器的交互包含以下操作：

- 配置MyODBC DSN。
- 连接到MySQL服务器。
- 初始化操作。
- 执行SQL语句。
- 检索结果。
- 执行事务。
- 断开与服务器的连接。

大多数应用程序均使用了这些步骤的某些变体。在下图中，给出了基本的应用步骤：



26.1.16. MyODBC API引用

在本节中，概要介绍了按功能分类的ODBC子程序。

关于全部ODBC API参考，请参见ODBC程序员参考，http://msdn.microsoft.com/library/en-us/odbc/htm/odbcabout_this_manual.asp。

应用程序可以调用SQLGetInfo函数来获得关于MyODBC的一致性信息。为了获得驱动程序对特定函数的支持信息，应用程序可调用SQLGetFunctions。

注释： 为了向后兼容，MyODBC 3.51驱动程序支持所有已不使用的函数。

在下面的表各中，按任务分组列出了MyODBC API调用：

连接到数据源：

函数名	**MyODBC**	**MyODBC**	一致性	目的
	2.50	**3.51**		
SQLAllocHandle	No	Yes	ISO 92	获取环境、连接、语句或描述符句柄。
SQLConnect	Yes	Yes	ISO 92	按数据源名、用户ID和密码连接到特定驱动程序。
SQLDriverConnect	Yes	Yes	ODBC	通过连接字符串，或驱动管理器和驱动显示对话框发出的请求，连接到特定驱动程序。
SQLAllocEnv	Yes	Yes	Deprecated	获得驱动程序分配的环境句柄。
SQLAllocConnect	Yes	Yes	Deprecated	获取连接句柄。

获取关于驱动程序和数据源的信息:

函数名	**MyODBC**	**MyODBC**	一致性	目的
	2.50	**3.51**		
SQLDataSources	No	No	ISO 92	返回可用数据源的列表，由驱动管理器处理。
SQLDrivers	No	No	ODBC	返回已安装驱动程序和器属性的列表，由驱动管理器处理。
SQLGetInfo	Yes	Yes	ISO 92	返回关于特定驱动程序和数据源的信息。
SQLGetFunctions	Yes	Yes	ISO 92	返回支持的驱动函数。
SQLGetTypeInfo	Yes	Yes	ISO 92	返回关于所支持数据类型的信息。

设置并检索驱动属性:

函数名	**MyODBC**	**MyODBC**	一致性	目的
	2.50	**3.51**		
SQLSetConnectAttr	No	Yes	ISO 92	设置连接属性。
SQLGetConnectAttr	No	Yes	ISO 92	返回连接属性的值。
SQLSetConnectOption	Yes	Yes	Deprecated	设置连接选项。
SQLGetConnectOption	Yes	Yes	Deprecated	返回连接选项的值。
SQLSetEnvAttr	No	Yes	ISO 92	设置环境属性。
SQLGetEnvAttr	No	Yes	ISO 92	返回环境属性的值。
SQLSetStmtAttr	No	Yes	ISO 92	设置语句属性。
SQLGetStmtAttr	No	Yes	ISO 92	返回语句属性的值。
SQLSetStmtOption	Yes	Yes	Deprecated	设置语句选项。
SQLGetStmtOption	Yes	Yes	Deprecated	返回语句选项的值。

准备**SQL**请求:

函数名	**MyODBC**	**MyODBC**	一致性	目的
	2.50	**3.51**		
SQLAllocStmt	Yes	Yes	Deprecated	分配语句句柄。
SQLPrepare	Yes	Yes	ISO 92	准备随后执行的SQL语句。
SQLBindParameter	Yes	Yes	ODBC	为SQL语句中的参数分配存储器。
SQLGetCursorName	Yes	Yes	ISO 92	返回与语句句柄相关的光标名。
SQLSetCursorName	Yes	Yes	ISO 92	指定光标名。
SQLSetScrollOptions	Yes	Yes	ODBC	设置控制光标行为的选项。

提交请求:

函数名	**MyODBC**	**MyODBC**	一致性	目的
	2.50	**3.51**		
SQLExecute	Yes	Yes	ISO 92	执行准备好的语句。
SQLExecDirect	Yes	Yes	ISO 92	执行语句。
SQLNativeSql	Yes	Yes	ODBC	返回由驱动程序翻译的SQL语句的文本。
SQLDescribeParam	Yes	Yes	ODBC	返回语句中特定参数的描述。
SQLNumParams	Yes	Yes	ISO 92	返回语句中的参数数目。
SQLParamData	Yes	Yes	ISO 92	与SQLPutData一起使用，以便在执行时提供参数。（对于长数据值很有用）。
SQLPutData	Yes	Yes	ISO 92	发送某一参数数据值的部分或全部。（对于长数据值很有用）。

检索结果以及关于结果的信息:

函数名	**MyODBC**	**MyODBC**	一致性	目的
	2.50	**3.51**		
SQLRowCount	Yes	Yes	ISO 92	返回插入、更新或删除请求影响的行数。
SQLNumResultCols	Yes	Yes	ISO 92	返回结果集中的列数。
SQLDescribeCol	Yes	Yes	ISO 92	描述结果集中的列。
SQLColAttribute	No	Yes	ISO 92	描述结果集中的某1列的属性。
SQLColAttributes	Yes	Yes	Deprecated	描述结果集中的某1列的多个属性。
SQLFetch	Yes	Yes	ISO 92	返回多个结果行。
SQLFetchScroll	No	Yes	ISO 92	返回可滚动结果行。
SQLExtendedFetch	Yes	Yes	Deprecated	返回可滚动结果行。
SQLSetPos	Yes	Yes	ODBC	将光标定为在获取的数据块中，允许应用程序更新行集合中的数据，或更新或删除结果集中的数据。
SQLBulkOperations	No	Yes	ODBC	执行批量插入和批量书签操作，包括更新、删除和按书签获取。

检索错误和诊断信息:

函数名	**MyODBC**	**MyODBC**	一致性	目的
	2.50	**3.51**		
SQLError	Yes	Yes	Deprecated	返回额外的错误或状态信息。
SQLGetDiagField	Yes	Yes	ISO 92	返回额外的诊断信息（诊断性数据结构的单个字段）。
SQLGetDiagRec	Yes	Yes	ISO 92	返回额外的诊断信息（诊断性数据结构的多个字段）。

获取关于数据源的系统表（目录函数）条目的信息:

函数名	**MyODBC**	**MyODBC**	一致性	目的
	2.50	**3.51**		
SQLColumnPrivileges	Yes	Yes	ODBC	返回关于一个或多个表的列和相关属性的列表。
SQLColumns	Yes	Yes	X/Open	返回指定表中列名的列表。
SQLForeignKeys	Yes	Yes	ODBC	在指定表中如果存在外键，返回构成外键的列名列表。
SQLPrimaryKeys	Yes	Yes	ODBC	返回构成某1表的主键的列名列表。
SQLSpecialColumns	Yes	Yes	X/Open	返回关于最佳列集合的信息，该列集合唯一地指明了指定表中的行，或当某1事务更新了行中的任何值时自动更新的列。
SQLStatistics	Yes	Yes	ISO 92	返回关于单个表的统计信息，以及与表相关的索引列表。
SQLTablePrivileges	Yes	Yes	ODBC	返回表列表，以及与各表相关的权限。
SQLTables	Yes	Yes	X/Open	返回存储在特定数据源内的表名列表。

执行事务:

函数名	**MyODBC**	**MyODBC**	一致性	目的
	2.50	**3.51**		
SQLTransact	Yes	Yes	Deprecated	提交或回滚事务。
SQLEndTran	No	Yes	ISO 92	提交或回滚事务。

中止语句:

函数名	**MyODBC**	**MyODBC**	一致性	目的
	2.50	**3.51**		
SQLFreeStmt	Yes	Yes	ISO 92	结束语句处理，舍弃未决结果，并释放与语句句柄相关的所有资源（可选）。
SQLCloseCursor	Yes	Yes	ISO 92	关闭在语句句柄上打开的指针。
SQLCancel	Yes	Yes	ISO 92	取消SQL语句。

中止连接:

函数名	**MyODBC**	**MyODBC**	一致性	目的
	2.50	**3.51**		
SQLDisconnect	Yes	Yes	ISO 92	关闭连接。
SQLFreeHandle	No	Yes	ISO 92	释放环境、连接、语句或描述符句柄。
SQLFreeConnect	Yes	Yes	Deprecated	释放连接句柄。
SQLFreeEnv	Yes	Yes	Deprecated	释放连接句柄。

26.1.17. MyODBC数据类型

在下表中，介绍了驱动程序将服务器数据类型映射为默认SQL和C数据类型的方法:

值	**SQL**类型	**C**类型
bit	SQL_BIT	SQL_C_BIT
tinyint	SQL_TINYINT	SQL_C_STINYINT
tinyint unsigned	SQL_TINYINT	SQL_C_UTINYINT
bigint	SQL_BIGINT	SQL_C_SBIGINT
bigint unsigned	SQL_BIGINT	SQL_C_UBIGINT
long varbinary	SQL_LONGVARBINARY	SQL_C_BINARY
blob	SQL_LONGVARBINARY	SQL_C_BINARY
longblob	SQL_LONGVARBINARY	SQL_C_BINARY
tinyblob	SQL_LONGVARBINARY	SQL_C_BINARY
mediumblob	SQL_LONGVARBINARY	SQL_C_BINARY
long varchar	SQL_LONGVARCHAR	SQL_C_CHAR
text	SQL_LONGVARCHAR	SQL_C_CHAR
mediumtext	SQL_LONGVARCHAR	SQL_C_CHAR
char	SQL_CHAR	SQL_C_CHAR
numeric	SQL_NUMERIC	SQL_C_CHAR
decimal	SQL_DECIMAL	SQL_C_CHAR
integer	SQL_INTEGER	SQL_C_SLONG
integer unsigned	SQL_INTEGER	SQL_C_ULONG
int	SQL_INTEGER	SQL_C_SLONG
int unsigned	SQL_INTEGER	SQL_C_ULONG

mediumint	SQL_INTEGER	SQL_C_SLONG
mediumint unsigned	SQL_INTEGER	SQL_C_ULONG
smallint	SQL_SMALLINT	SQL_C_SSHORT
smallint unsigned	SQL_SMALLINT	SQL_C_USHORT
real	SQL_FLOAT	SQL_C_DOUBLE
double	SQL_FLOAT	SQL_C_DOUBLE
float	SQL_REAL	SQL_C_FLOAT
double precision	SQL_DOUBLE	SQL_C_DOUBLE
date	SQL_DATE	SQL_C_DATE
time	SQL_TIME	SQL_C_TIME
year	SQL_SMALLINT	SQL_C_SHORT
datetime	SQL_TIMESTAMP	SQL_C_TIMESTAMP
timestamp	SQL_TIMESTAMP	SQL_C_TIMESTAMP
text	SQL_VARCHAR	SQL_C_CHAR
varchar	SQL_VARCHAR	SQL_C_CHAR
enum	SQL_VARCHAR	SQL_C_CHAR
set	SQL_VARCHAR	SQL_C_CHAR
bit	SQL_CHAR	SQL_C_CHAR
bool	SQL_CHAR	SQL_C_CHAR

26.1.18. MyODBC错误代码

在下表中，列出了驱动程序返回的除服务器错误之外的错误代码列表：

本机代码	**SQLSTATE 2**	**SQLSTATE 3**	错误消息
500	01000	01000	一般警告
501	01004	01004	字符串数据，右截
502	01S02	01S02	选项值被更改
503	01S03	01S03	未更新／删除行
504	01S04	01S04	更新／删除了1个以上的行
505	01S06	01S06	在结果集合返回第1个行集合之前视图获取数据。
506	07001	07002	对于所有参数，未使用SQLBindParameter。
507	07005	07005	精制语句不符合光标规范
508	07009	07009	无效的描述符索引。
509	08002	08002	连接名正在使用。
510	08003	08003	连接不存在。
511	24000	24000	无效的光标状态。
512	25000	25000	无效的事务状态。
513	25S01	25S01	事务状态未知。
514	34000	34000	无效光标名。
515	S1000	HY000	一般的驱动程序定义错误。
516	S1001	HY001	内存分配错误。
517	S1002	HY002	无效的列编号。
518	S1003	HY003	无效的应用缓冲类型。
519	S1004	HY004	无效的SQL数据类型。
520	S1009	HY009	空指针的无效使用。

521	S1010	HY010	函数顺序错误。
522	S1011	HY011	现在无法设置属性。
523	S1012	HY012	无效的事务操作码。
524	S1013	HY013	内存管理错误。
525	S1015	HY015	无可用的光标名。
526	S1024	HY024	无效的属性值。
527	S1090	HY090	无效字符串或缓冲长度。
528	S1091	HY091	无效的描述符字段标识符。
529	S1092	HY092	无效的属性／选项标识符。
530	S1093	HY093	无效的参数编号。
531	S1095	HY095	函数类型超出范围。
532	S1106	HY106	获取类型超出范围。
533	S1117	HY117	行值超出范围。
534	S1109	HY109	无效的光标位置。
535	S1C00	HYC00	可选特性未实施。
0	21S01	21S01	列计数与值计数不匹配。
0	23000	23000	完整性约束违反。
0	42000	42000	语法错误或访问冲突。
0	42S02	42S02	未发现基本表或视图。
0	42S12	42S12	未发现索引。
0	42S21	42S21	列已存在。
0	42S22	42S22	未发现列。
0	08S01	08S01	通信链接失败。

26.1.19. MyODBC与VB：ADO、DAO和RDO

26.1.19.1. ADO: rs.addNew, rs.delete和rs.update
26.1.19.2. DAO: rs.addNew, rs.update和滚动
26.1.19.3. RDO: rs.addNew和rs.update

在本节中，给出了MySQL ODBC 3.51驱动程序与ADO、DAO和RDO一起使用的一些简单示例。

26.1.19.1. ADO: rs.addNew, rs.delete和rs.update

在下面的ADO（ActiveX数据对象）示例中，创建了表my_ado，并演示了rs.addNew、rs.delete和rs.update的用法。

```
Private Sub myodbc_ado_Click()

Dim conn As ADODB.Connection
Dim rs As ADODB.Recordset
Dim fld As ADODB.Field
Dim sql As String

'connect to MySQL server using MySQL ODBC 3.51 Driver（使用MySQL ODBC 3.51驱动程序连接到MySQL服务器）
Set conn = New ADODB.Connection
conn.ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};"_
```

```
                        & "SERVER=localhost;"_
                        & " DATABASE=test;"_
                        & "UID=venu;PWD=venu; OPTION=3"

conn.Open

'create table（创建表）
conn.Execute "DROP TABLE IF EXISTS my_ado"
conn.Execute "CREATE TABLE my_ado(id int not null primary key, name varchar(20)," _
                              & "txt text, dt date, tm time, ts timestamp)"

'direct insert（直接插入）
conn.Execute "INSERT INTO my_ado(id,name,txt) values(1,100,'venu')"
conn.Execute "INSERT INTO my_ado(id,name,txt) values(2,200,'MySQL')"
conn.Execute "INSERT INTO my_ado(id,name,txt) values(3,300,'Delete')"

Set rs = New ADODB.Recordset
rs.CursorLocation = adUseServer

'fetch the initial table ..（获取初始表…）
rs.Open "SELECT * FROM my_ado", conn
  Debug.Print rs.RecordCount
  rs.MoveFirst
  Debug.Print String(50, "-") & "Initial my_ado Result Set " & String(50, "-")
  For Each fld In rs.Fields
    Debug.Print fld.Name,
    Next
    Debug.Print

    Do Until rs.EOF
    For Each fld In rs.Fields
    Debug.Print fld.Value,
    Next
    rs.MoveNext
    Debug.Print
  Loop
rs.Close

'rs insert（rs插入）
```

```
rs.Open "select * from my_ado", conn, adOpenDynamic, adLockOptimistic
rs.AddNew
rs!Name = "Monty"
rs!txt = "Insert row"
rs.Update
rs.Close

'rs update (rs更新)
rs.Open "SELECT * FROM my_ado"
rs!Name = "update"
rs!txt = "updated-row"
rs.Update
rs.Close

'rs update second time.. (rs更新第2次…)
rs.Open "SELECT * FROM my_ado"
rs!Name = "update"
rs!txt = "updated-second-time"
rs.Update
rs.Close

'rs delete (rs删除)
rs.Open "SELECT * FROM my_ado"
rs.MoveNext
rs.MoveNext
rs.Delete
rs.Close

'fetch the updated table .. (获取更新的表…)
rs.Open "SELECT * FROM my_ado", conn
  Debug.Print rs.RecordCount
  rs.MoveFirst
  Debug.Print String(50, "-") & "Updated my_ado Result Set " & String(50, "-")
  For Each fld In rs.Fields
    Debug.Print fld.Name,
    Next
    Debug.Print

    Do Until rs.EOF
    For Each fld In rs.Fields
```

```
    Debug.Print fld.Value,
    Next
    rs.MoveNext
    Debug.Print
  Loop
rs.Close
conn.Close
End Sub
```

26.1.19.2. DAO: rs.addNew, rs.update和滚动

在下面的DAO（数据访问对象）示例中，创建了表my_dao，并演示了rs.addNew、rs.update、以及结果集滚动的用法。

```
Private Sub myodbc_dao_Click()

Dim ws As Workspace
Dim conn As Connection
Dim queryDef As queryDef
Dim str As String

'connect to MySQL server using MySQL ODBC 3.51 Driver（使用MySQL ODBC 3.51驱动程序连接到MySQL）
Set ws = DBEngine.CreateWorkspace("", "venu", "venu", dbUseODBC)
str = "odbc;DRIVER={MySQL ODBC 3.51 Driver};"_
                      & "SERVER=localhost;"_
                      & " DATABASE=test;"_
                      & "UID=venu;PWD=venu; OPTION=3"
Set conn = ws.OpenConnection("test", dbDriverNoPrompt, False, str)

'Create table my_dao（创建表my_dao）
Set queryDef = conn.CreateQueryDef("", "drop table if exists my_dao")
queryDef.Execute

Set queryDef = conn.CreateQueryDef("", "create table my_dao(Id INT AUTO_INCREMENT PRIMARY KEY, " _
                                                      & "Ts TIMESTAMP(14) NOT NULL, Name varchar(20), Id2 INT)")
queryDef.Execute

'Insert new records using rs.addNew（使用rs.addNew插入新记录）
Set rs = conn.OpenRecordset("my_dao")
Dim i As Integer

  For i = 10 To 15
```

```
  rs.AddNew
  rs!Name = "insert record" & i
  rs!Id2 = i
  rs.Update
  Next i
          rs.Close

'rs update..（rs更新）
Set rs = conn.OpenRecordset("my_dao")
rs.Edit
rs!Name = "updated-string"
rs.Update
rs.Close

'fetch the table back...（向后获取表…）
Set rs = conn.OpenRecordset("my_dao", dbOpenDynamic)
str = "Results:"
rs.MoveFirst
While Not rs.EOF
str = " " & rs!Id & " , " & rs!Name & ", " & rs!Ts & ", " & rs!Id2
Debug.Print "DATA:" & str
rs.MoveNext
Wend

'rs Scrolling（rs滚动）
rs.MoveFirst
str = " FIRST ROW: " & rs!Id & " , " & rs!Name & ", " & rs!Ts & ", " & rs!Id2
Debug.Print str

rs.MoveLast
str = " LAST ROW: " & rs!Id & " , " & rs!Name & ", " & rs!Ts & ", " & rs!Id2
Debug.Print str

rs.MovePrevious
str = " LAST-1 ROW: " & rs!Id & " , " & rs!Name & ", " & rs!Ts & ", " & rs!Id2
Debug.Print str

'free all resources（释放所有资源）
rs.Close
```

```
queryDef.Close
conn.Close
ws.Close

End Sub
```

26.1.19.3. RDO: rs.addNew和rs.update

在下面的RDO（远程数据对象）示例中，创建了表my_rdo，并演示了rs.addNew和rs.update的用法。

```
Dim rs As rdoResultset
  Dim cn As New rdoConnection
  Dim cl As rdoColumn
  Dim SQL As String

  'cn.Connect = "DSN=test;"
  cn.Connect = "DRIVER={MySQL ODBC 3.51 Driver};"_
                    & "SERVER=localhost;"_
                    & " DATABASE=test;"_
                    & "UID=venu;PWD=venu; OPTION=3"

  cn.CursorDriver = rdUseOdbc
  cn.EstablishConnection rdDriverPrompt



  'drop table my_rdo（舍弃表my_rdo）
  SQL = "drop table if exists my_rdo"
  cn.Execute SQL, rdExecDirect

  'create table my_rdo（创建表my_rdo）
  SQL = "create table my_rdo(id int, name varchar(20))"
  cn.Execute SQL, rdExecDirect

  'insert — direct（插入，直接）
  SQL = "insert into my_rdo values (100,'venu')"
  cn.Execute SQL, rdExecDirect

  SQL = "insert into my_rdo values (200,'MySQL')"
  cn.Execute SQL, rdExecDirect

  'rs insert（rs插入）
```

```
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
rs.AddNew
rs!id = 300
rs!Name = "Insert1"
rs.Update
rs.Close

'rs insert（rs插入）
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
rs.AddNew
rs!id = 400
rs!Name = "Insert 2"
rs.Update
rs.Close

'rs update（rs更新）
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
rs.Edit
rs!id = 999
rs!Name = "updated"
rs.Update
rs.Close

'fetch back...
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
Do Until rs.EOF
For Each cl In rs.rdoColumns
            Debug.Print cl.Value,
  Next
  rs.MoveNext
  Debug.Print
          Loop
Debug.Print "Row count="; rs.RowCount

'close（关闭）
rs.Close
```

```
  cn.Close


End Sub
```

26.1.20. MyODBC与Microsoft.NET

26.1.20.1. ODBC.NET: CSHARP(C#)
26.1.20.2. ODBC.NET: VB

本节包含一些简单示例，介绍了MyODBC驱动程序与ODBC.NET一起使用的用法。

26.1.20.1. ODBC.NET: CSHARP(C#)

在下面的简单示例中创建了表my_odbc_net，并介绍了它在C#中的使用。

```
/**
* @sample    : mycon.cs
* @purpose   : Demo sample for ODBC.NET using MyODBC
* @author    : Venu, <venu@mysql.com>
*
* (C) Copyright MySQL AB, 1995-2004
*
**/

/* build command
*
*  csc /t:exe
*      /out:mycon.exe mycon.cs
*      /r:Microsoft.Data.Odbc.dll
*/

using Console = System.Console;
using Microsoft.Data.Odbc;

namespace myodbc3
{
class mycon
{
  static void Main(string[] args)
  {
    try
    {
      //Connection string for MyODBC 2.50
      /*string MyConString = "DRIVER={MySQL};" +
                           "SERVER=localhost;" +
                           "DATABASE=test;" +
                           "UID=venu;" +
                           "PASSWORD=venu;" +
                           "OPTION=3";
      */
      //Connection string for MyODBC 3.51
      string MyConString = "DRIVER={MySQL ODBC 3.51 Driver};" +
                           "SERVER=localhost;" +
                           "DATABASE=test;" +
                           "UID=venu;" +
                           "PASSWORD=venu;" +
                           "OPTION=3";

      //Connect to MySQL using MyODBC
      OdbcConnection MyConnection = new OdbcConnection(MyConString);
      MyConnection.Open();

      Console.WriteLine("\n !!! success, connected successfully !!!\n");

      //Display connection information
      Console.WriteLine("Connection Information:");
      Console.WriteLine("\tConnection String:" + MyConnection.ConnectionString);
      Console.WriteLine("\tConnection Timeout:" + MyConnection.ConnectionTimeout);
      Console.WriteLine("\tDatabase:" + MyConnection.Database);
      Console.WriteLine("\tDataSource:" + MyConnection.DataSource);
      Console.WriteLine("\tDriver:" + MyConnection.Driver);
      Console.WriteLine("\tServerVersion:" + MyConnection.ServerVersion);
```

```
        //Create a sample table
        OdbcCommand MyCommand = new OdbcCommand("DROP TABLE IF EXISTS my_odbc_net",MyConnection);
        MyCommand.ExecuteNonQuery();
        MyCommand.CommandText = "CREATE TABLE my_odbc_net(id int, name varchar(20), idb bigint)";
        MyCommand.ExecuteNonQuery();

        //Insert
        MyCommand.CommandText = "INSERT INTO my_odbc_net VALUES(10,'venu', 300)";
        Console.WriteLine("INSERT, Total rows affected:" + MyCommand.ExecuteNonQuery());;

        //Insert
        MyCommand.CommandText = "INSERT INTO my_odbc_net VALUES(20,'mysql',400)";
        Console.WriteLine("INSERT, Total rows affected:" + MyCommand.ExecuteNonQuery());

        //Insert
        MyCommand.CommandText = "INSERT INTO my_odbc_net VALUES(20,'mysql',500)";
        Console.WriteLine("INSERT, Total rows affected:" + MyCommand.ExecuteNonQuery());

        //Update
        MyCommand.CommandText = "UPDATE my_odbc_net SET id=999 WHERE id=20";
        Console.WriteLine("Update, Total rows affected:" + MyCommand.ExecuteNonQuery());

        //COUNT(*)
        MyCommand.CommandText = "SELECT COUNT(*) as TRows FROM my_odbc_net";
        Console.WriteLine("Total Rows:" + MyCommand.ExecuteScalar());

        //Fetch
        MyCommand.CommandText = "SELECT * FROM my_odbc_net";
        OdbcDataReader MyDataReader;
        MyDataReader =  MyCommand.ExecuteReader();
        while (MyDataReader.Read())
        {
         if(string.Compare(MyConnection.Driver,"myodbc3.dll") == 0) {
           Console.WriteLine("Data:" + MyDataReader.GetInt32(0) + " " +
                                       MyDataReader.GetString(1) + " " +
                                       MyDataReader.GetInt64(2)); //Supported only by MyODBC 3.51
         }
         else {
           Console.WriteLine("Data:" + MyDataReader.GetInt32(0) + " " +
                                       MyDataReader.GetString(1) + " " +
                                       MyDataReader.GetInt32(2)); //BIGINTs not supported by MyODBC
         }
        }

        //Close all resources
        MyDataReader.Close();
        MyConnection.Close();
      }
      catch (OdbcException MyOdbcException)//Catch any ODBC exception ..
      {
        for (int i=0; i < MyOdbcException.Errors.Count; i++)
        {
          Console.Write("ERROR #" + i + "\n" +
            "Message: " + MyOdbcException.Errors[i].Message + "\n" +
            "Native: " + MyOdbcException.Errors[i].NativeError.ToString() + "\n" +
            "Source: " + MyOdbcException.Errors[i].Source + "\n" +
            "SQL: " + MyOdbcException.Errors[i].SQLState + "\n");
        }
      }
    }
  }
}
```

26.1.20.2. ODBC.NET: VB

在下面的简单示例中创建了表my_vb_net，并介绍了它在VB中的用法。

```
' @sample    : myvb.vb
' @purpose   : Demo sample for ODBC.NET using MyODBC
' @author    : Venu, <venu@mysql.com>
'
' (C) Copyright MySQL AB, 1995-2004
'
'

'
' build command
'
```

```
' vbc /target:exe
'     /out:myvb.exe
'     /r:Microsoft.Data.Odbc.dll
'     /r:System.dll
'     /r:System.Data.dll
'

Imports Microsoft.Data.Odbc
Imports System

Module myvb
  Sub Main()
      Try

          'MyODBC 3.51 connection string
          Dim MyConString As String = "DRIVER={MySQL ODBC 3.51 Driver};" & _
                         "SERVER=localhost;" & _
                         "DATABASE=test;" & _
                         "UID=venu;" & _
                         "PASSWORD=venu;" & _
                         "OPTION=3;"

          'Connection
          Dim MyConnection As New OdbcConnection(MyConString)
          MyConnection.Open()

          Console.WriteLine ("Connection State::" & MyConnection.State.ToString)

          'Drop
          Console.WriteLine ("Dropping table")
          Dim MyCommand As New OdbcCommand()
          MyCommand.Connection = MyConnection
          MyCommand.CommandText = "DROP TABLE IF EXISTS my_vb_net"
          MyCommand.ExecuteNonQuery()

          'Create
          Console.WriteLine ("Creating....")
          MyCommand.CommandText = "CREATE TABLE my_vb_net(id int, name varchar(30))"
          MyCommand.ExecuteNonQuery()

          'Insert
          MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(10,'venu')"
          Console.WriteLine("INSERT, Total rows affected:" & MyCommand.ExecuteNonQuery())

          'Insert
          MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(20,'mysql')"
          Console.WriteLine("INSERT, Total rows affected:" & MyCommand.ExecuteNonQuery())

          'Insert
          MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(20,'mysql')"
          Console.WriteLine("INSERT, Total rows affected:" & MyCommand.ExecuteNonQuery())

          'Insert
          MyCommand.CommandText = "INSERT INTO my_vb_net(id) VALUES(30)"
          Console.WriteLine("INSERT, Total rows affected:" & MyCommand.ExecuteNonQuery())

          'Update
          MyCommand.CommandText = "UPDATE my_vb_net SET id=999 WHERE id=20"
          Console.WriteLine("Update, Total rows affected:" & MyCommand.ExecuteNonQuery())

          'COUNT(*)
          MyCommand.CommandText = "SELECT COUNT(*) as TRows FROM my_vb_net"
          Console.WriteLine("Total Rows:" & MyCommand.ExecuteScalar())

          'Select
          Console.WriteLine ("Select * FROM my_vb_net")
          MyCommand.CommandText = "SELECT * FROM my_vb_net"
          Dim MyDataReader As OdbcDataReader
          MyDataReader = MyCommand.ExecuteReader
          While MyDataReader.Read
              If MyDataReader("name") Is DBNull.Value Then
                  Console.WriteLine ("id = " & CStr(MyDataReader("id")) & "  name = " & _
                    "NULL")
              Else
                  Console.WriteLine ("id = " & CStr(MyDataReader("id")) & "  name = " & _
                                     CStr(MyDataReader("name")))
              End If
          End While

      'Catch ODBC Exception
      Catch MyOdbcException As OdbcException
          Dim i As Integer
```

```
            Console.WriteLine (MyOdbcException.ToString)

        'Catch program exception
        Catch MyException As Exception
            Console.WriteLine (MyException.ToString)
    End Try
    End Sub
End Module
```

26.1.21. 感谢

下面列出了一些MySQL AB公司负责MyODBC和MyODBC 3.51驱动程序开发的人员。

- Micheal (Monty) Widenius
- Venu Anuganti
- Peter Harvey

26.2. MySQL Connector/NET

26.2.1. 前言
26.2.2. 下载并安装MySQL Connector/NET
26.2.3. Connector/NET体系结构
26.2.4. 使用MySQL Connector/NET
26.2.5. MySQL Connector/NET变更史

26.2.1. 前言

使用MySQL Connector/NET，开发人员能够方便地创建需要安全和高性能数据连接（与MySQL）的.NET应用程序。它实施了所需的ADO.NET接口，并将其集成到了ADO.NET aware工具中。开发人员能够使用他们选择的.NET语言创建应用程序。MySQL Connector/NET是用100％纯C#语言编写的可完全管理的ADO.NET驱动程序。

MySQL Connector/NET包括对下述事宜的完整支持：

- MySQL 5.0特性（存储程序等）。
- MySQL 4.1特性（服务器端的精制语句、Unicode、以及共享内存访问等）。
- 大信息包支持，可发送和接收高达2GB的行和BLOB。
- 协议压缩，允许压缩客户端和服务器之间的数据流。
- 支持使用CP/IP套接字、命名管道、以及Windows共享内存的连接。
- 支持使用CP/IP套接字、或Unix套接字的连接。
- 支持由Novell开发的开放源码Mono框架。
- 可完全管理，不利用MySQL客户端库。

MySQL Connector/NET的开发人员高度尊重用户在软件开发过程中提供的帮助。如果你发现MySQL Connector/NET缺少对你来说很重要的某些特性，或者如果你发现了缺陷，请使用我们的MySQL缺陷系统请求该特性或通报问题。

通过http://forums.mysql.com上的论坛以及http://lists.mysql.com上的邮件列表，可找到针对MySQL Connector/NET的社区支持信息。MySQL AB公司提供付费支持，更多信息请参见http://www.mysql.com/support/。

本文档的目的是作为MySQL Connector/NET的用户指南，而不是语法参考。如果你打算了解详细的语法信息，请阅读MySQL Connector/NET分发版中提供的Documentation.chm文件。

26.2.2. 下载并安装MySQL Connector/NET

MySQL Connector/NET能够运行在任何支持.NET框架的平台上。.NET框架主要被最近的Microsoft Windows版本支持，通过由Novell开发的Mono框架，在Linux上也支持它（请参见http://www.mono-project.com）。

MySQL Connector/NET可通过使用Windows Installer (.msi)安装软件包进行安装，使用该软件包，可在任何Windows操作系统上安装MySQL Connector/NET。MSI软件包包含在名为mysql-connector-net-*version*.zip的压缩文件中，其中，“*version*”（版本）指明了MySQL Connector/NET的版本。

可从下述网站下载MySQL Connector/NET：http://dev.mysql.com/downloads/connector/net/1.0.html。

随着Windows XP的发布，Windows Installer（安装器）引擎也予以了更新，对于使用旧版本的用户，可参阅该Microsoft知识库文章以了解升级至最新版本的更多信息。

要想安装MySQL Connector/NET，请右击MSI文件并选择“安装”。在安装器提示你完成安装参数选择后，安装将自动开始。对于大多数用户，建议采用典型安装。

如果在运行安装器时遇到问题，可下载不带安装器的ZIP文件。该文件名为mysql-connector-net-*version*-noinstall.zip。使用ZIP程序，将其解压至你所选择的目录。

除非作了其他选择，否则MySQL Connector/NET将被安装到“C:\Program Files\MySQL\MySQL Connector Net *X.X.X*”，其中，“*X.X.X*”是你所安装的MySQL Connector/NET的版本号。新安装不会覆盖已有的MySQL Connector/NET版本。

26.2.3. Connector/NET体系结构

26.2.3.1. MySqlCommand类
26.2.3.2. MySqlCommandBuilder类
26.2.3.3. MySqlConnection类
26.2.3.4. MySqlDataAdapter类
26.2.3.5. MySqlDataReader类
26.2.3.6. MySqlException类
26.2.3.7. MySqlHelper类
26.2.3.8. MySqlTransaction类

MySQL Connector/NET包含数个类，这些类可用于连接到数据库，执行查询和语句，并管理查询结果。

下面介绍了MySQL Connector/NET的主要类：

- MySqlCommand：代表对MySQL数据库进行执行操作的SQL语句。
- MySqlCommandBuilder：自动生成单个表的命令，用于协调对DataSet所作的更改和相关的MySQL数据库。
- MySqlConnection：代表与MySQL服务器数据库的开放式连接。
- MySqlDataAdapter：代表一组数据命令和数据库连接，用于填充数据库和更新MySQL数据库。
- MySqlDataReader：提供了从MySQL数据库读取行的“仅正向”流的一种方式。
- MySqlException：当MySQL返回错误时抛出的异常。
- MySqlHelper：助手类，能使工作变的更简单。
- MySqlTransaction：代表将在MySQL数据库中进行的SQL事务。

在后续段落中，将分别介绍这些对象。这些章节的目的是概要介绍MySQL Connector/NET的主要类，而不是语法参考。如果你打算了解详细的语法信息，请阅读MySQL Connector/NET分发版中提供的Documentation.chm文件。

26.2.3.1. MySqlCommand类

26.2.3.1.1. 属性
26.2.3.1.2. 方法
26.2.3.1.3. 用法

MySqlCommand类代表对MySQL数据库进行执行操作的SQL语句。

注释：在以前的版本中，采用符号"@"来标识SQL中的参数。它与MySQL用户变量不兼容，因此，现采用符号"?"来定位SQL中的参数。为了支持早期代码，也可以在连接字符串中设置"old syntax=yes"。如果进行了这类设置，请注意，如果无法定义希望在SQL中使用的参数（定义失败），不会给出异常提示。

26.2.3.1.1. 属性

可用属性如下：

- CommandText：获取或设置将在数据源上执行的SQL语句。
- CommandTimeout：获取或设置中止执行命令并生成错误之前应等待的时间。
- CommandType：获取或设置值，该值指明了解释CommandText的方式。可能的值包括StoredProcedure、TableDirect和Text。
- Connection：获取或设置该MySqlCommand实例使用的MySqlConnection。
- IsPrepared：如果该命令已准备好，为"真"，否则为"假"。
- Parameters：获取MySqlParameterCollection。
- Transaction：获取或设置MySqlTransaction，MySqlCommand将在其中执行。
- UpdatedRowSource：当DbDataAdapter的Update方法使用它时，用于获取或设置命令结果作用在DataRow上的方式。

26.2.3.1.2. 方法

可用方法如下：

- Cancel：尝试取消MySqlCommand的执行。不支持该操作。
- Clone：创建该MySqlCommand对象的克隆对象。包括CommandText、Connection和Transaction属性，以及整个参数列表。
- CreateParameter：创建MySqlParameter对象的新实例。
- Dispose：处理该MySqlCommand实例。
- ExecuteNonQuery：根据连接情况执行SQL语句，并返回受影响的行数。
- ExecuteReader：将CommandText发送给Connection，并创建MySqlDataReader。
- ExecuteScalar：执行查询，并返回查询操作所返回的结果集中第1行的第1列。多余的列或行将被忽略。
- Prepare：在MySQL服务器的1个实例上创建命令的预制版本。

26.2.3.1.3. 用法

26.2.3.1.3.1. VB.NET
26.2.3.1.3.2. C#

在下面的示例中，创建了1个MySqlCommand和1个MySqlConnection。打开了MySqlConnection，并将其设置为用于MySqlCommand的连接。随后，调用ExecuteNonQuery，并
ExecuteNonQuery 1 1 1 SQL INSERT

关闭连接。为了完成该任务，为 传递了 个连接字符串和 个查询字符串（查询字符串是 条 语句）。

26.2.3.1.3.1. VB.NET

在下例中，介绍了在VB.NET中使用MySqlCommand类的方法：

```
Public Sub InsertRow(myConnectionString As String)
    ' If the connection string is null, use a default.
    If myConnectionString = "" Then
        myConnectionString = "Database=Test;Data Source=localhost;User Id=username;Password=pass"
    End If
    Dim myConnection As New MySqlConnection(myConnectionString)
    Dim myInsertQuery As String = "INSERT INTO Orders (id, customerId, amount) Values(1001, 23, 30.66)"
    Dim myCommand As New MySqlCommand(myInsertQuery)
    myCommand.Connection = myConnection
    myConnection.Open()
    myCommand.ExecuteNonQuery()
    myCommand.Connection.Close()
End Sub
```

26.2.3.1.3.2. C#

在下例中，介绍了在C#中使用MySqlCommand类的方法：

```
public void InsertRow(string myConnectionString)
{
    // If the connection string is null, use a default.
    if(myConnectionString == "")
    {
        myConnectionString = "Database=Test;Data Source=localhost;User Id=username;Password=pass";
    }
    MySqlConnection myConnection = new MySqlConnection(myConnectionString);
    string myInsertQuery = "INSERT INTO Orders (id, customerId, amount) Values(1001, 23, 30.66)";
    MySqlCommand myCommand = new MySqlCommand(myInsertQuery);
    myCommand.Connection = myConnection;
    myConnection.Open();
    myCommand.ExecuteNonQuery();
    myCommand.Connection.Close();
}
```

26.2.3.2. MySqlCommandBuilder类

26.2.3.2.1. 属性
26.2.3.2.2. 方法
26.2.3.2.3. 用法

MySqlDataAdapter不会自动生成所需的SQL语句（用于协调对DataSet所作的更改和相关的MySQL实例）。但是，如果设置了MySqlDataAdapter的SelectCommand属性，可以创建MySqlCommandBuilder对象来自动生成针对单个表更新的SQL语句。随后，MySqlCommandBuilder将生成你未设置的任何附加的SQL语句。

一旦你设置了DataAdapter属性，MySqlCommandBuilder会将自己注册为针对OnRowUpdating事件的监听程序。一次只能将1个MySqlDataAdapter或MySqlCommandBuilder对象关联起来。

为了生成INSERT、UPDATE或DELETE语句，MySqlCommandBuilder使用了SelectCommand属性来自动检索所需的元数据集合。如果在检索完元数据后更改了SelectCommand（例如首次更新后），应调用RefreshSchema方法来更新元数据。

SelectCommand也必须返回至少1个主键或唯一列。如果未显示任何返回信息，将生成InvalidOperation异常，而且不会生成命令。

MySqlCommandBuilder还会使用SelectCommand引用的Connection、CommandTimeout和Transaction属性。如果更改了这些属性中的任何1个，或者，如果替换了SelectCommand本身，用户应调用RefreshSchema。如不然，InsertCommand、UpdateCommand和DeleteCommand属性将保持它们以前的值。

如果调用了Dispose，MySqlCommandBuilder将解除与MySqlDataAdapter的关联，已生成的命令将不再使用。

26.2.3.2.1. 属性

可用属性如下：

- DataAdapter：MySqlCommandBuilder将自己注册为针对RowUpdating事件的监听程序，RowUpdating事件是由在该属性中指定的MySqlDataAdapter生成的。创建了新的MySqlCommandBuilder实例时，将释放任何已有的与MySqlDataAdapter关联的MySqlCommandBuilder。

- QuotePrefix, QuoteSuffix：MySQL中的数据库对象能够包含特殊字符，如空格等，这会使得正常的SQL字符串无法解析。使用QuotePrefix和QuoteSuffix属性，MySqlCommandBuilder能够创建处理该问题的SQL命令。

26.2.3.2.2. 方法

可用方法如下：

- DeriveParameters：从MySqlCommand指定的存储程序中检索参数信息，并填充所指定MySqlCommand对象的参数集。目前不支持该方法，这是因为MySQL中未提供存储程序。

- GetDeleteCommand：获取用于在数据库上执行删除操作所需的、自动生成的MySqlCommand对象。

- GetInsertCommand：获取用于在数据库上执行插入操作所需的、自动生成的MySqlCommand对象。

- GetUpdateCommand：获取用于在数据库上执行更新操作所需的、自动生成的MySqlCommand对象。

- RefreshSchema：刷新用于生成INSERT、UPDATE或DELETE语句的数据库方案信息。

26.2.3.2.3. 用法

26.2.3.2.3.1. VB.NET
26.2.3.2.3.2. C#

在下面给出的示例中，使用了MySqlCommand、MySqlDataAdapter和MySqlConnection，用于从数据源选择行。为该示例传递了1个初始化的DataSet、1个连接字符串、1个查询字符串（是SQL SELECT语句）、以及1个作为数据库表名称的字符串。随后，该示例创建了1个MySqlCommandBuilder。

26.2.3.2.3.1. VB.NET

在下例中，介绍了在VB.NET中使用MySqlCommandBuilder类的方法：

```
Public Shared Function SelectRows(myConnection As String, mySelectQuery As String, myTableName As String) As DataSet
      Dim myConn As New MySqlConnection(myConnection)
      Dim myDataAdapter As New MySqlDataAdapter()
      myDataAdapter.SelectCommand = New MySqlCommand(mySelectQuery, myConn)
      Dim cb As SqlCommandBuilder = New MySqlCommandBuilder(myDataAdapter)

      myConn.Open()

      Dim ds As DataSet = New DataSet
      myDataAdapter.Fill(ds, myTableName)

      ' Code to modify data in DataSet here

      ' Without the MySqlCommandBuilder this line would fail.
      myDataAdapter.Update(ds, myTableName)

      myConn.Close()
  End Function 'SelectRows
```

26.2.3.2.3.2. C#

在下例中，介绍了在C#中使用MySqlCommandBuilder类的方法：

```
public static DataSet SelectRows(string myConnection, string mySelectQuery, string myTableName)
{
```

```
        MySqlConnection myConn = new MySqlConnection(myConnection);
        MySqlDataAdapter myDataAdapter = new MySqlDataAdapter();
        myDataAdapter.SelectCommand = new MySqlCommand(mySelectQuery, myConn);
        MySqlCommandBuilder cb = new MySqlCommandBuilder(myDataAdapter);

        myConn.Open();

        DataSet ds = new DataSet();
        myDataAdapter.Fill(ds, myTableName);

        //code to modify data in DataSet here

        //Without the MySqlCommandBuilder this line would fail
        myDataAdapter.Update(ds, myTableName);

        myConn.Close();

        return ds;
    }
```

26.2.3.3. MySqlConnection类

26.2.3.3.1. 属性
26.2.3.3.2. 方法
26.2.3.3.3. 用法

MySqlConnection对象代表与MySQL服务器数据源的会话。创建MySqlConnection实例时，所有属性均将被设置为它们的初始值。关于这些值的列表，请参见MySqlConnection构造函数。

如果MySqlConnection超出范围，不会被关闭。因此，必须通过调用Close或Dispose明确地关闭连接。

26.2.3.3.1. 属性

可用属性如下：

- ConnectionString：设置或获取用于连接至MySQL服务器数据库的字符串。
- ConnectionTimeout：获取在中止尝试并生成错误之前为建立连接所需的等待时间。
- Database：获取当前数据库的名称或打开连接后将使用的数据库的名称。
- DataSource：获取将要连接的MySQL服务器的名称。
- ServerThread：返回该连接所使用的服务器线程的ID。
- ServerVersion：获取包含客户端与之相连的MySQL服务器版本的字符串。
- State：获取连接的当前连接的状态。
- UseConnection：与服务器进行通信时，指明该连接是否将使用压缩特性。

26.2.3.3.2. 方法

可用方法如下：

- BeginTransaction：开始数据库事务。
- ChangeDatabase：针对打开的MySqlConnection，更改当前数据库。
- Close：关闭与数据库的连接。这是关闭任何打开连接的首选方法。

- CreateCommand：创建并返回与MySqlConnection相关的MySqlCommand对象。
- Dispose：释放MySqlConnection使用的资源。
- Open：用ConnectionString指定的属性设置打开数据库连接。
- Ping：对MySQL服务器执行Ping操作。

26.2.3.3.3. 用法

26.2.3.3.3.1. VB.NET
26.2.3.3.3.2. C#

在下面的示例中，创建了1个MySqlCommand和1个MySqlConnection。打开了MySqlConnection，并将其设置为用于MySqlCommand的连接。随后，调用ExecuteNonQuery，并关闭连接。为了完成该任务，为ExecuteNonQuery传递了1个连接字符串和1个查询字符串（查询字符串是1条SQL INSERT语句）。

26.2.3.3.3.1. VB.NET

在下例中，介绍了在VB.NET中使用MySqlConnection类的方法：

```
Public Sub InsertRow(myConnectionString As String)
    ' If the connection string is null, use a default.
    If myConnectionString = "" Then
        myConnectionString = "Database=Test;Data Source=localhost;User Id=username;Password=pass"
    End If
    Dim myConnection As New MySqlConnection(myConnectionString)
    Dim myInsertQuery As String = "INSERT INTO Orders (id, customerId, amount) Values(1001, 23, 30.66)"
    Dim myCommand As New MySqlCommand(myInsertQuery)
    myCommand.Connection = myConnection
    myConnection.Open()
    myCommand.ExecuteNonQuery()
    myCommand.Connection.Close()
End Sub
```

26.2.3.3.3.2. C#

在下例中，介绍了在C#中使用MySqlConnection类的方法：

```
public void InsertRow(string myConnectionString)
{
    // If the connection string is null, use a default.
    if(myConnectionString == "")
    {
        myConnectionString = "Database=Test;Data Source=localhost;User Id=username;Password=pass";
    }
    MySqlConnection myConnection = new MySqlConnection(myConnectionString);
    string myInsertQuery = "INSERT INTO Orders (id, customerId, amount) Values(1001, 23, 30.66)";
    MySqlCommand myCommand = new MySqlCommand(myInsertQuery);
    myCommand.Connection = myConnection;
    myConnection.Open();
    myCommand.ExecuteNonQuery();
    myCommand.Connection.Close();
}
```

26.2.3.4. MySqlDataAdapter类

26.2.3.4.1. 属性
26.2.3.4.2. 方法
26.2.3.4.3. 用法

MySQLDataAdapter起着DataSet和MySQL之间桥梁的作用，用于检索和保存数据。MySQLDataAdapter通过映射Fill（填充）和Update（更新）提供了该桥，Fill能改变DataSet中

的数据以便与数据源中的数据匹配，Update能改变数据源中的数据以便与DataSet中的数据匹配（通过对数据源使用恰当的SQL语句）。

当MySQLDataAdapter填充DataSet时，如果表或列不存在，它将为返回的数据创建必要的表和列。但是，在隐式创建的方案中不包括主键信息，除非将MissingSchemaAction属性设为AddWithKey。在使用FillSchema用数据填充它之前，也能让MySQLDataAdapter创建DataSet方案，包含主键信息。

MySQLDataAdapter用于MySqlConnection和MySqlCommand的连接，用以提升连接至MySQL数据库时的性能。

MySQLDataAdapter还包括SelectCommand、InsertCommand、DeleteCommand、UpdateCommand和TableMappings属性，用于简化数据的加载和更新操作。

26.2.3.4.1. 属性

可用属性如下：

- AcceptChangesDuringFill：获取或设置值，该值指明了在任何填充操作过程中，在将DataRow添加到DataTable后，是否要在DataRow上调用AcceptChanges。
- ContinueUpdateOnError：获取或设置值，该值指定了在行更新过程中出现错误时是否要生成异常项。
- DeleteCommand：获取或设置用于将记录从数据集中删除的SQL语句或存储程序。
- InsertCommand：获取或设置用于在数据集中插入记录的SQL语句或存储程序。
- MissingMappingAction：确定当进入的数据不含匹配表或列时需要采取的动作。
- MissingSchemaAction：确定当已有的DataSet方案与进入数据不匹配时需要采取的动作。
- SelectCommand：获取或设置用于在数据源中选择记录的SQL语句或存储程序。
- TableMappings：获取提供了源表和DataTable之间主映射的集合。
- UpdateCommand：获取或设置用于在数据源中更新记录的SQL语句或存储程序。

26.2.3.4.2. 方法

可用方法如下：

- Fill：使用DataSet名称添加或刷新DataSet中的行，以便与数据源中的行匹配，并创建名为"Table"的DataTable。
- FillSchema：将名为"Table"的DataTable添加到指定的DataSet，并配置方案，以便与基于指定SchemaType的数据源中的表匹配。
- GetFillParameters：执行SQL SELECT语句时，按用户获取参数集。
- Update：为指定DataSet中的各插入行、更新行或删除行分别调用INSERT、UPDATE或DELETE语句。

26.2.3.4.3. 用法

26.2.3.4.3.1. VB.NET
26.2.3.4.3.2. C#

在下面的示例中，创建了1个MySqlCommand和1个MySqlConnection。打开MySqlConnection，并将其设置为用于MySqlCommand的连接。随后，调用ExecuteNonQuery，并关闭连接。为了完成该任务，为ExecuteNonQuery传递了1个连接字符串和1个查询字符串（查询字符串是1条SQL INSERT语句）。

26.2.3.4.3.1. VB.NET

在下例中，介绍了在VB.NET中使用MySqlDataAdapter类的方法：

```
Public Function SelectRows(dataSet As DataSet, connection As String, query As String) As DataSet
    Dim conn As New MySqlConnection(connection)
    Dim adapter As New MySqlDataAdapter()
    adapter.SelectCommand = new MySqlCommand(query, conn)
    adapter.Fill(dataset)
    Return dataset
End Function
```

26.2.3.4.3.2. C#

在下例中，介绍了在C#中使用MySqlDataAdapter类的方法：

```
public DataSet SelectRows(DataSet dataset,string connection,string query)
{
    MySqlConnection conn = new MySqlConnection(connection);
    MySqlDataAdapter adapter = new MySqlDataAdapter();
    adapter.SelectCommand = new MySqlCommand(query, conn);
    adapter.Fill(dataset);
    return dataset;
}
```

26.2.3.5. MySqlDataReader类

26.2.3.5.1. 属性
26.2.3.5.2. 方法
26.2.3.5.3. 用法

MySqlDataReader类提供了从MySQL数据库读取行的"仅正向"流的一种方式。

要想创建MySQLDataReader，必须调用MySqlCommand对象的ExecuteReader方法，而不是直接使用构造函数。

使用MySqlDataReader的同时，相关的MySqlConnection将忙于MySqlDataReader。除了关闭它之外，不能在MySqlConnection上执行任何操作。该情况将一直持续到调用了MySqlDataReader的"Close"方法为止。

关闭了MySqlDataReader后，你只能调用IsClosed和RecordsAffected属性。尽管在MySqlDataReader存在同时能够访问RecordsAffected属性，但在返回RecordsAffected的值之前总应调用"Close"，以确保准确的返回值。

为了获得最佳性能，MySqlDataReader将避免创建不必要的对象或执行不必要的数据拷贝。其结果是，对诸如GetValue等方法的多个调用会返回对相同对象的引用。如果你准备更改由诸如GetValue等方法返回的对象的基本值，请仔细小心。

26.2.3.5.1. 属性

可用属性如下：

- Depth：获取指明当前行嵌套深度的值。目前并不支持方法，总会返回0。
- FieldCount：获取当前行中的列数。
- HasRows：获取值，该值指明了MySqlDataReader是否包含1行或多行。
- IsClosed：获取值，该值指明了和苏剧阅读器是否已关闭。
- Item：以固有格式获取列的值。在C#，该属性是MySqlDataReader类的索引属性。
- RecordsAffected：获取隐执行SQL语句而更改、插入、或删除的行数。

26.2.3.5.2. 方法

可用方法如下：

- Close：关闭MySqlDataReader对象。
- GetBoolean：获取指定列的布尔值。
- GetByte：以字节形式获取指定列的值。
- GetBytes：读取从指定列偏移至缓冲的字节流，数组从给定的缓冲偏移位置开始。
- GetChar：以单字符形式获取指定列的值。
- GetChars：读取从指定列偏移至缓冲的字符流，数组从给定的缓冲偏移位置开始。
- GetDataTypeName：获取源数据类型的名称。
- GetDateTime：以DateTime对象形式获取指定列的值。
- GetDecimal：以DateTime对象形式获取指定列的值。
- GetDouble：以双精度浮点数的形式获取指定列的值。
- GetFieldType：获取作为对象数据类型的类型。
- GetFloat：以单精度浮点数的形式获取指定列的值。
- GetGuid：以GUID的形式获取指定列的值。
- GetInt16：以16位带符号整数的形式获取指定列的值。
- GetInt32：以32位带符号整数的形式获取指定列的值。
- GetInt64：以64位带符号整数的形式获取指定列的值。
- GetMySqlDateTime：以MySqlDateTime对象的形式获取指定列的值。
- GetName：获取指定列的名称。
- GetOrdinal：给定列名，获取列的顺序。
- GetSchemaTable：返回描述了MySqlDataReader的列元数据的DataTable。
- GetString：以String对象的形式获取指定列的值。
- GetTimeSpan：以TimeSpan对象的形式获取指定列的值。
- GetUInt16：以16位无符号整数的形式获取指定列的值。
- GetUInt32：以32位无符号整数的形式获取指定列的值。
- GetUInt64：以64位无符号整数的形式获取指定列的值。
- GetValue：以固有格式获取指定列的值。
- GetValues：获取当前行集合中的所有属性列。
- IsDBNull：获取值，该值指明了列中是否包含不存在或丢失的值。
- NextResult：读取批SQL语句的结果时，使数据阅读器跳到下一个结果。

- Read：使MySqlDataReader跳到下一条记录。

26.2.3.5.3. 用法

26.2.3.5.3.1. VB.NET
26.2.3.5.3.2. C#

在下面的示例中，创建了1个MySqlConnection，1个MySqlCommand和1个MySqlDataReader。该示例读取数据，并将数据输出到控制台。最后，本例关闭了MySqlDataReader，然后关闭了MySqlConnection。

26.2.3.5.3.1. VB.NET

在下例中，介绍了在VB.NET中使用MySqlDataReader类的方法：

```
Public Sub ReadMyData(myConnString As String)
    Dim mySelectQuery As String = "SELECT OrderID, CustomerID FROM Orders"
    Dim myConnection As New MySqlConnection(myConnString)
    Dim myCommand As New MySqlCommand(mySelectQuery, myConnection)
    myConnection.Open()
    Dim myReader As MySqlDataReader
    myReader = myCommand.ExecuteReader()
    ' Always call Read before accessing data.
    While myReader.Read()
        Console.WriteLine((myReader.GetInt32(0) & ", " & myReader.GetString(1)))
    End While
    ' always call Close when done reading.
    myReader.Close()
    ' Close the connection when done with it.
    myConnection.Close()
End Sub 'ReadMyData
```

26.2.3.5.3.2. C#

在下例中，介绍了在C#中使用MySqlDataReader类的方法：

```
public void ReadMyData(string myConnString) {
    string mySelectQuery = "SELECT OrderID, CustomerID FROM Orders";
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery,myConnection);
    myConnection.Open();
    MySqlDataReader myReader;
    myReader = myCommand.ExecuteReader();
    // Always call Read before accessing data.
    while (myReader.Read()) {
       Console.WriteLine(myReader.GetInt32(0) + ", " + myReader.GetString(1));
    }
    // always call Close when done reading.
    myReader.Close();
    // Close the connection when done with it.
    myConnection.Close();
 }
```

26.2.3.6. MySqlException类

26.2.3.6.1. 属性
26.2.3.6.2. 方法
26.2.3.6.3. 用法

当MySql数据提供方遇到服务器生成的错误时将创建该类。

抛出异常时，打开的连接不会自动关闭。如果客户端应用程序判定该异常是致命的，应关闭任何打开的MySqlDataReader对象或MySqlConnection对象。

26.2.3.6.1. 属性

可用属性如下：

- HelpLink：获取或设置指向与该异常相关的帮助文件的链接。
- InnerException：获取导致当前异常的异常实例。
- IsFatal：如果该异常是致命的，为"真"，并关闭连接，如果不是致命的，为"假"。
- Message：获取描述当前异常的消息。
- Number：获取指明错误类型的编号。
- Source：获取或设置导致错误的应用程序或对象的名称。
- StackTrace：获取抛出当前异常时在调用堆栈上帧的字符串表征。
- TargetSite：获取抛出当前异常的方法。

26.2.3.6.2. 方法

MySqlException类没有相应的方法。

26.2.3.6.3. 用法

26.2.3.6.3.1. VB.NET
26.2.3.6.3.2. C#

在下述示例中，因丢失了服务器而生成了MySqlException，然后显示异常。

26.2.3.6.3.1. VB.NET

该示例介绍在VB.NET下使用MySqlException类的方法。

```
Public Sub ShowException()
     Dim mySelectQuery As String = "SELECT column1 FROM table1"
     Dim myConnection As New MySqlConnection ("Data Source=localhost;Database=Sample;")
     Dim myCommand As New MySqlCommand(mySelectQuery, myConnection)

     Try
         myCommand.Connection.Open()
     Catch e As MySqlException
        MessageBox.Show( e.Message )
     End Try
 End Sub
```

26.2.3.6.3.2. C#

该示例介绍在C#下使用MySqlException类的方法。

```
public void ShowException()
{
   string mySelectQuery = "SELECT column1 FROM table1";
   MySqlConnection myConnection =
      new MySqlConnection("Data Source=localhost;Database=Sample;");
   MySqlCommand myCommand = new MySqlCommand(mySelectQuery,myConnection);

   try
   {
      myCommand.Connection.Open();
```

```
    }
    catch (MySqlException e)
    {
        MessageBox.Show( e.Message );
    }
}
```

26.2.3.7. MySqlHelper类

26.2.3.7.1. 属性
26.2.3.7.2. 方法

助手类，能使与提供方（Provider）一起进行的工作变的更简单。开发人员可以使用该类提供的方法自动执行共同任务。

26.2.3.7.1. 属性

MySqlHelper类没有相应的属性。

26.2.3.7.2. 方法

可用方法如下：

- ExecuteDataRow：执行单个SQL语句并返回结果集的第1行。在该方法的执行过程中，将创建、打开并关闭1个新的MySqlConnection对象。
- ExecuteDataset：执行单个SQL命令并返回DataSet中的结果集。在该方法的执行过程中，将创建、打开并关闭1个新的MySqlConnection对象。
- ExecuteNonQuery：在MySQL数据库上执行单个命令。调用该方法时，将认为MySqlConnection已打开，方法执行完后，MySqlConnection仍保持打开状态。
- ExecuteReader：Overloaded：在MySQL数据库上执行单个命令。
- ExecuteScalar：在MySQL数据库上执行单个命令。
- UpdateDataSet：用来自给定DataSet的数据更新给定表。

26.2.3.8. MySqlTransaction类

26.2.3.8.1. 属性
26.2.3.8.2. 方法
26.2.3.8.3. 用法

代表将在MySQL数据库中进行的SQL事务。

26.2.3.8.1. 属性

可用属性如下：

- Connection：获取与事务相关的MySqlConnection对象，如果事务不再有效，获取空引用（在Visual Basic中为Nothing）。
- IsolationLevel：为该事务指定IsolationLevel。

26.2.3.8.2. 方法

可用方法如下：

- Commit：提交数据库事务。

- Rollback：从挂起状态回滚事务。

26.2.3.8.3. 用法

26.2.3.8.3.1. VB.NET
26.2.3.8.3.2. C#

在下面的示例中，创建了1个MySqlConnection和1个MySqlTransaction。此外，在示例中还介绍了如何使用BeginTransaction、Commit和Rollback方法。

26.2.3.8.3.1. VB.NET

在下例中，介绍了在VB.NET中使用MySqlTransaction类的方法：

```
Public Sub RunTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySqlTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()
    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
      myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (100, 'Description')"
      myCommand.ExecuteNonQuery()
      myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (101, 'Description')"
      myCommand.ExecuteNonQuery()
      myTrans.Commit()
      Console.WriteLine("Both records are written to database.")
    Catch e As Exception
      Try
        myTrans.Rollback()
      Catch ex As MySqlException
        If Not myTrans.Connection Is Nothing Then
          Console.WriteLine("An exception of type " & ex.GetType().ToString() & _
                            " was encountered while attempting to roll back the transaction.")
        End If
      End Try

      Console.WriteLine("An exception of type " & e.GetType().ToString() & _
                        "was encountered while inserting the data.")
      Console.WriteLine("Neither record was written to database.")
    Finally
      myConnection.Close()
    End Try
End Sub 'RunTransaction
```

26.2.3.8.3.2. C#

在下例中，介绍了在C#中使用MySqlTransaction类的方法：

```
public void RunTransaction(string myConnString)
 {
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();

    MySqlCommand myCommand = myConnection.CreateCommand();
    MySqlTransaction myTrans;

    // Start a local transaction
    myTrans = myConnection.BeginTransaction();
    // Must assign both transaction object and connection
    // to Command object for a pending local transaction
    myCommand.Connection = myConnection;
    myCommand.Transaction = myTrans;
```

```
    try
    {
      myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (100, 'Description')";
      myCommand.ExecuteNonQuery();
      myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (101, 'Description')";
      myCommand.ExecuteNonQuery();
      myTrans.Commit();
      Console.WriteLine("Both records are written to database.");
    }
    catch(Exception e)
    {
      try
      {
        myTrans.Rollback();
      }
      catch (MySqlException ex)
      {
        if (myTrans.Connection != null)
        {
          Console.WriteLine("An exception of type " + ex.GetType() +
                            " was encountered while attempting to roll back the transaction.");
        }
      }

      Console.WriteLine("An exception of type " + e.GetType() +
                        " was encountered while inserting the data.");
      Console.WriteLine("Neither record was written to database.");
    }
    finally
    {
      myConnection.Close();
    }
}
```

26.2.4. 使用MySQL Connector/NET

26.2.4.1. 前言
26.2.4.2. 使用MySQL Connector/NET连接到MySQL
26.2.4.3. 与预处理语句一起使用MySQL Connector/NET
26.2.4.4. 用MySQL Connector/NET访问存储程序
26.2.4.5. 用Connector/NET处理BLOB数据
26.2.4.6. 与Crystal Reports一起使用MySQL Connector/NET
26.2.4.7. 在MySQL Connector/NET中处理日期和时间信息

26.2.4.1. 前言

在本节中，介绍的Connector/NET的一些常用方式，包括BLOB处理，日期处理，以及与诸如Crystal Reports等常见工具一起使用Connector/NET的方法。

26.2.4.2. 使用MySQL Connector/NET连接到MySQL

26.2.4.2.1. 前言
26.2.4.2.2. 创建连接字符串
26.2.4.2.3. 打开连接
26.2.4.2.4. 处理连接错误

26.2.4.2.1. 前言

.NET应用程序和MySQL服务器之间的所有交互均是通过MySqlConnection对象传送的。在应用程序能够与服务器进行交互之前，必须获取、配置、并打开MySqlConnection对象。

即使在使用MySqlHelper类时，MySqlConnection对象也会被Helper类创建。

在本节中，介绍了使用MySqlConnection对象连接到MySQL的方法。

26.2.4.2.2. 创建连接字符串

MySqlConnection对象是使用连接字符串配置的。1个连接字符串包含服务器键／值对，由分号隔开。每个键／值对由等号连接。

下面给出了1个简单的连接字符串示例：

```
Server=127.0.0.1;Uid=root;Pwd=12345;Database=test;
```

在本例中，对MySqlConnection对象进行了配置，使用用户名"root"和密码"12345"与位于127.0.0.1的MySQL服务器相连。所有语句的默认数据库为测试数据库。

典型的选项如下（关于选项的完整清单，请参见API文档）：

- Server：将要连接的MySQL实例的名称或网络地址。默认为本地主机。别名包括Host, Data Source, DataSource, Address, Addr和Network Address。
- Uid：连接时使用的MySQL用户账户。别名包括User Id, Username和User name。
- Pwd：MySQL账户的密码。也可以使用别名密码。
- Database：所有语句作用于的默认数据库。默认为mysql。也可以使用别名Initial Catalog。
- Port：MySQL用于监听连接的端口。默认为3306。将该值指定为"-1"将使用命名管道连接。

26.2.4.2.3. 打开连接

一旦创建了连接字符串，可使用它打开与MySQL服务器的连接。

下述代码用于创建MySqlConnection对象，指定连接字符串，并打开连接。

[VB]

```
Dim conn As New MySql.Data.MySqlClient.MySqlConnection
Dim myConnectionString as String

myConnectionString = "server=127.0.0.1;" _
            & "uid=root;"_
            & "pwd=12345;"_
            & "database=test;"

Try
  conn.ConnectionString = myConnectionString
  conn.Open()

Catch ex As MySql.Data.MySqlClient.MySqlException
  MessageBox.Show(ex.Message)
End Try
```

[C#]

```
MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;

myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection();
    conn.ConnectionString = myConnectionString;
    conn.Open();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message);
```

```
}
```

你也可以将连接字符串传递给MySqlConnection类的构造函数：

[VB]

```
Dim myConnectionString as String

myConnectionString = "server=127.0.0.1;" _
              & "uid=root;"
              & "pwd=12345;"_
              & "database=test;"

Try
    Dim conn As New MySql.Data.MySqlClient.MySqlConnection(myConnectionString)
    conn.Open()
Catch ex As MySql.Data.MySqlClient.MySqlException
   MessageBox.Show(ex.Message)
End Try
```

[C#]

```
MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;

myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection(myConnectionString);
    conn.Open();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message);
}
```

一旦打开了连接，其他MySQL Connector/NET类也能使用该连接与MySQL服务器进行通信。

26.2.4.2.4. 处理连接错误

由于与外部服务器的连接不可预测，应为你的.NET应用程序添加错误处理功能，这点很重要。出现连接错误时，MySqlConnection类将返回1个MySqlException对象。该对象有两个在处理错误时十分有用的属性：

- Message：描述当前异常的消息。
- Number：MySQL错误编号。

处理错误时，可根据错误编号了解应用程序的响应。进行连接时最常见的两个错误编号如下：

- 0: 无法连接到服务器。
- 1045: 无效的用户名和／或密码。

在下面的代码中，介绍了根据实际错误改编应用程序的方法：

[VB]

```
Dim myConnectionString as String

myConnectionString = "server=127.0.0.1;" _
          & "uid=root;"
          & "pwd=12345;"_
          & "database=test;"

Try
    Dim conn As New MySql.Data.MySqlClient.MySqlConnection(myConnectionString)
```

```
    conn.Open()
Catch ex As MySql.Data.MySqlClient.MySqlException
    Select Case ex.Number
        Case 0
            MessageBox.Show("Cannot connect to server. Contact administrator")
        Case 1045
            MessageBox.Show("Invalid username/password, please try again")
    End Select
End Try
```

[C#]

```
MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;

myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection(myConnectionString);
    conn.Open();
}
    catch (MySql.Data.MySqlClient.MySqlException ex)
{
    switch (ex.Number)
    {
        case 0:
            MessageBox.Show("Cannot connect to server.  Contact administrator");
        case 1045:
            MessageBox.Show("Invalid username/password, please try again");
    }
}
```

26.2.4.3. 与预处理语句一起使用MySQL Connector/NET

26.2.4.3.1. 前言
26.2.4.3.2. 在MySQL Connector/NET中准备语句

26.2.4.3.1. 前言

从MySQL 4.1开始，能够与MySQL Connector/NET一起使用预处理语句。使用预处理语句能够现住改善多次执行的查询的性能。

对于多次执行的语句，预处理执行的速度快于直接执行，这是因为只需进行1次解析操作。在直接执行的情况下，每次执行时均将进行解析操作。预处理执行还能降低网络通信量，这是因为对于预处理语句的每次执行，仅需发送用于参数的数据。

预处理语句的另一优点是，它能使用二进制协议，这使得客户端和服务器间的数据传输更有效率。

26.2.4.3.2. 在MySQL Connector/NET中准备语句

为了准备好语句，需创建1个命令对象，并为查询设置.CommandText属性。

输入语句后，调用MySqlCommand对象的.Prepare方法。完成语句的准备后，为查询中的每个元素添加参数。

输入查询并输入参数后，使用.ExecuteNonQuery()、.ExecuteScalar()、或.ExecuteReader方法执行语句。

对于后续的执行操作，仅需更改参数值并再次调用执行方法，无需设置.CommandText属性或重新定义参数。

[VB]

```
Dim conn As New MySqlConnection

Dim cmd As New MySqlCommand
```

```
conn.ConnectionString = strConnection

Try
   conn.Open()
   cmd.Connection = conn

   cmd.CommandText = "INSERT INTO myTable VALUES(NULL, ?number, ?text)"
   cmd.Prepare()

   cmd.Parameters.Add("?number", 1)
   cmd.Parameters.Add("?text", "One")

   For i = 1 To 1000
       cmd.Parameters("?number").Value = i
       cmd.Parameters("?text").Value = "A string value"

       cmd.ExecuteNonQuery()
     Next
Catch ex As MySqlException
    MessageBox.Show("Error " & ex.Number & " has occurred: " & ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

[C#]

```
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();

conn.ConnectionString = strConnection;

try
{
    conn.Open();
    cmd.Connection = conn;

    cmd.CommandText = "INSERT INTO myTable VALUES(NULL, ?number, ?text)";
    cmd.Prepare();
```

```
    cmd.Parameters.Add("?number", 1);
    cmd.Parameters.Add("?text", "One");

    for (int i=1; i <= 1000; i++)
    {
        cmd.Parameters["?number"].Value = i;
        cmd.Parameters["?text"].Value = "A string value";

        cmd.ExecuteNonQuery();
    }
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

26.2.4.4. 用MySQL Connector/NET访问存储程序

26.2.4.4.1. 前言
26.2.4.4.2. 从MySQL Connector/NET创建存储程序
26.2.4.4.3. 从MySQL Connector/NET调用存储程序

26.2.4.4.1. 前言

随着MySQL版本5的发布，MySQL服务器目前支持存储程序，它采用了SQL 2003存储程序的语法。

存储程序指的是能够保存在服务器上的一组SQL语句。 一旦完成了该操作，客户端无需再次发出单独语句，而仅需引用存储程序取而代之。

在下述情况下，存储程序尤其有用：

- 多个客户端应用程序是采用不同语言编写的或工作在不同平台上，但需执行相同的数据库操作。
- 安全性极其重要时。例如，对于所有共同操作，银行采用了存储程序。这样，就能提供一致且安全的环境，而且这类存储程序能够保证每次操作均具有恰当登录。在这类设置下，应用程序和用户无法直接访问数据库表，但能执行特定的存储程序。

MySQL Connector/NET支持通过MySqlCommand对象的存储程序调用。使用MySqlCommand.Parameters集，能够将数据传入和传出MySQL存储程序。

在本节中，未深度介绍创建存储程序方面的信息，要想了解这类信息，请参见MySQL参考手册的存储程序。

在MySQL Connector/NET安装的Samples目录下，可找到1个相应的示例，该示例演示了与MySQL Connector/NET一起使用存储程序的方法。

26.2.4.4.2. 从MySQL Connector/NET创建存储程序

可使用多种工具创建MySQL中的存储程序。首先，可使用**mysql**命令行客户端创建存储程序。其次，可使用MySQL Query Browser GUI客户端创建存储程序。最后，可使用MySqlCommand对象的.ExecuteNonQuery方法创建存储程序。

[VB]

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;"_
    & "pwd=12345;"_
    & "database=test"

Try
    conn.Open()
    cmd.Connection = conn

    cmd.CommandText = "CREATE PROCEDURE add_emp(" _
        & "IN fname VARCHAR(20), IN lname VARCHAR(20), IN bday DATETIME, OUT empno INT) " _
        & "BEGIN INSERT INTO emp(first_name, last_name, birthdate) " _
        & "VALUES(fname, lname, DATE(bday)); SET empno = LAST_INSERT_ID(); END"

    cmd.ExecuteNonQuery()
Catch ex As MySqlException
    MessageBox.Show("Error " & ex.Number & " has occurred: " & ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

[C#]

```
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn.Open();
    cmd.Connection = conn;

    cmd.CommandText = "CREATE PROCEDURE add_emp(" +
        "IN fname VARCHAR(20), IN lname VARCHAR(20), IN bday DATETIME, OUT empno INT) " +
        "BEGIN INSERT INTO emp(first_name, last_name, birthdate) " +
        "VALUES(fname, lname, DATE(bday)); SET empno = LAST_INSERT_ID(); END";

    cmd.ExecuteNonQuery();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
    "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

请注意，不同于命令行和GUI客户端，在MySQL Connector/NET中创建存储程序时不需要指定特殊的定界符。

26.2.4.4.3. 从**MySQL Connector/NET**调用存储程序

要想使用MySQL Connector/NET来调用存储程序，应创建1个MySqlCommand对象，并将存储程序名作为.CommandText属性传递。将.CommandType属性设置为CommandType.StoredProcedure。

命名了存储程序后，为存储程序中的每个参数创建1个MySqlCommand参数。用参数名和包含值的对象定义IN参数，用参数名和预计将返回的数据类型定义OUT参数。对于所有参数，均需定义参数方向。

定义完参数后，使用MySqlCommand.ExecuteNonQuery()方法调用存储程序。

[VB]

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;"_
    & "pwd=12345;"_
```

```
        & "database=test"

    Try
        conn.Open()
        cmd.Connection = conn

        cmd.CommandText = "add_emp"
        cmd.CommandType = CommandType.StoredProcedure

        cmd.Parameters.Add("?lname", 'Jones')
        cmd.Parameters("?lname").Direction = ParameterDirection.Input

        cmd.Parameters.Add("?fname", 'Tom')
        cmd.Parameters("?fname").Direction = ParameterDirection.Input

        cmd.Parameters.Add("?bday", #12/13/1977 2:17:36 PM#)
        cmd.Parameters("?bday").Direction = ParameterDirection.Input

        cmd.Parameters.Add("?empno", MySqlDbType.Int32)
        cmd.Parameters("?empno").Direction = ParameterDirection.Output

        cmd.ExecuteNonQuery()

        MessageBox.Show(cmd.Parameters("?empno").Value)
    Catch ex As MySqlException
        MessageBox.Show("Error " & ex.Number & " has occurred: " & ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Try
```

[C#]

```
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn.Open();
    cmd.Connection = conn;

    cmd.CommandText = "add_emp";
    cmd.CommandType = CommandType.StoredProcedure;

    cmd.Parameters.Add("?lname", "Jones");
    cmd.Parameters("?lname").Direction = ParameterDirection.Input;

    cmd.Parameters.Add("?fname", "Tom");
    cmd.Parameters("?fname").Direction = ParameterDirection.Input;

    cmd.Parameters.Add("?bday", DateTime.Parse("12/13/1977 2:17:36 PM"));
    cmd.Parameters("?bday").Direction = ParameterDirection.Input;

    cmd.Parameters.Add("?empno", MySqlDbType.Int32);
    cmd.Parameters("?empno").Direction = ParameterDirection.Output;

    cmd.ExecuteNonQuery();

    MessageBox.Show(cmd.Parameters("?empno").Value);
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
      "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

一旦调用了存储程序，可使用MySqlConnector.Parameters集的.Value属性检索输出参数的值。

26.2.4.5. 用Connector/NET处理BLOB数据

26.2.4.5.1. 前言
26.2.4.5.2. 准备MySQL服务器
26.2.4.5.3. 将文件写入数据库

26.2.4.5.4. 将BLOB从数据库读取到磁盘上的文件

26.2.4.5.1. 前言

MySQL的1种用途是在BLOB列中保存二进制数据。MySQL支持4种不同的BLOB数据类型：TINYBLOB, BLOB, MEDIUMBLOB和LONGBLOB。

可使用Connector/NET访问保存在BLOB列中的数据，并能使用客户端代码对这类数据进行操作。使用Connector/NET和BLOB数据时，无特殊要求。

在本节中，给出了数个简单的代码示例，在MySQL Connector/NET安装的Samples目录下，可找到1个完整的示例应用程序。

26.2.4.5.2. 准备MySQL服务器

与BLOB数据一起使用MySQL的第1步是配置服务器。首先，让我们从创建要访问的表开始。在我的文件表中，通常有4列：1个具有恰当大小的AUTO_INCREMENT列（UNSIGNED SMALLINT），用于保存识别文件的主键；1个VARCHAR列，用于保存文件名；1个UNSIGNED MEDIUMINT列，用于保存文件的大小；以及1个用于保存文件本身的MEDIUMBLOB列。对于本例，我将使用下述表定义：

```
CREATE TABLE file(

file_id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,

file_name VARCHAR(64) NOT NULL,

file_size MEDIUMINT UNSIGNED NOT NULL,

file MEDIUMBLOB NOT NULL);
```

完成表的创建后，或许需要更改max_allowed_packet系统变量。该变量决定了能够发送给MySQL服务器的信息包（即单个行）大小。默认情况下，服务器能够接受来自客户端应用程序的信息包最大为1MB。如果不打算超过1MB，情况良好。如果打算在文件传输中超出1MB，必须增加该数值。

可以使用"MySQL系统管理员的启动变量"屏幕更改max_allowed_packet选项。在"联网"选项卡的"内存"部分，恰当调整"允许的最大值"选项。完成值的调整后，点击"应用更改"按钮，并使用"MySQL管理员"的"服务控制"屏幕重新启动服务器。也可以在my.cnf文件中直接调整该值（添加1行，max_allowed_packet=xxM），或在MySQL中使用SET max_allowed_packet=xxM。

设置max_allowed_packet时应保守些，这是因为传输BLOB数据需要一段时间。恰当地设置该值，使之与预期使用相符，并在必要时增大该值。

26.2.4.5.3. 将文件写入数据库

要想将文件写入数据库，需要将文件转换为字节数组，然后将字节数组用作INSERT查询的参数。

在下述代码中，使用FileStream对象打开了1个文件，将其读入至字节数组，然后将其插入到文件表中：

[VB]

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

Dim SQL As String

Dim FileSize As UInt32
Dim rawData() As Byte
Dim fs As FileStream

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

Try
    fs = New FileStream("c:\image.png", FileMode.Open, FileAccess.Read)
    FileSize = fs.Length

    rawData = New Byte(FileSize) {}
    fs.Read(rawData, 0, FileSize)
    fs.Close()
```

```
    conn.Open()

    SQL = "INSERT INTO file VALUES(NULL, ?FileName, ?FileSize, ?File)"

    cmd.Connection = conn
    cmd.CommandText = SQL
    cmd.Parameters.Add("?FileName", strFileName)
    cmd.Parameters.Add("?FileSize", FileSize)
    cmd.Parameters.Add("?File", rawData)

    cmd.ExecuteNonQuery()

    MessageBox.Show("File Inserted into database successfully!", _
    "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk)

    conn.Close()
Catch ex As Exception
    MessageBox.Show("There was an error: " & ex.Message, "Error", _
        MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

[C#]

```
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();

string SQL;
UInt32 FileSize;
byte[] rawData;
FileStream fs;

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    fs = new FileStream(@"c:\image.png", FileMode.Open, FileAccess.Read);
    FileSize = fs.Length;

    rawData = new byte[FileSize];
    fs.Read(rawData, 0, FileSize);
    fs.Close();

    conn.Open();

    SQL = "INSERT INTO file VALUES(NULL, ?FileName, ?FileSize, ?File)";

    cmd.Connection = conn;
    cmd.CommandText = SQL;
    cmd.Parameters.Add("?FileName", strFileName);
    cmd.Parameters.Add("?FileSize", FileSize);
    cmd.Parameters.Add("?File", rawData);

    cmd.ExecuteNonQuery();

    MessageBox.Show("File Inserted into database successfully!",
        "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

    conn.Close();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

FileStream对象的“Read”方法可用于将文件加载到字节数组中，该字节数组的大小是根据FileStream对象的“Length”属性确定的。

将字节数组指定为MySqlCommand对象的参数后，调用ExecuteNonQuery方法，并将BLOB插入到文件表中。

26.2.4.5.4. 将BLOB从数据库读取到磁盘上的文件

一旦将文件加载到了文件表中，就能使用MySqlDataReader类来检索它。

在下述代码中，从文件表提取了1行，然后将数据装载到要写入至磁盘的FileStream对象。

[VB]

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myData As MySqlDataReader
Dim SQL As String
Dim rawData() As Byte
Dim FileSize As UInt32
Dim fs As FileStream

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

SQL = "SELECT file_name, file_size, file FROM file"

Try
    conn.Open()

    cmd.Connection = conn
    cmd.CommandText = SQL

    myData = cmd.ExecuteReader

    If Not myData.HasRows Then Throw New Exception("There are no BLOBs to save")

    myData.Read()

    FileSize = myData.GetUInt32(myData.GetOrdinal("file_size"))
    rawData = New Byte(FileSize) {}

    myData.GetBytes(myData.GetOrdinal("file"), 0, rawData, 0, FileSize)

    fs = New FileStream("C:\newfile.png", FileMode.OpenOrCreate, FileAccess.Write)
    fs.Write(rawData, 0, FileSize)
    fs.Close()

    MessageBox.Show("File successfully written to disk!", "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk)

    myData.Close()
    conn.Close()
Catch ex As Exception
    MessageBox.Show("There was an error: " & ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

[C#]

```
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataReader myData;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();

string SQL;
UInt32 FileSize;
byte[] rawData;
FileStream fs;

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

SQL = "SELECT file_name, file_size, file FROM file";

try
{
    conn.Open();

    cmd.Connection = conn;
    cmd.CommandText = SQL;

    myData = cmd.ExecuteReader();
```

```
    if (! myData.HasRows)
        throw new Exception("There are no BLOBs to save");

    myData.Read();

    FileSize = myData.GetUInt32(myData.GetOrdinal("file_size"));
    rawData = new byte[FileSize];

    myData.GetBytes(myData.GetOrdinal("file"), 0, rawData, 0, FileSize);

    fs = new FileStream(@"C:\newfile.png", FileMode.OpenOrCreate, FileAccess.Write);
    fs.Write(rawData, 0, FileSize);
    fs.Close();

    MessageBox.Show("File successfully written to disk!",
        "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

    myData.Close();
    conn.Close();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

连接后，文件表的内容将被加载到MySqlDataReader对象中。使用MySqlDataReader的GetBytes方法将BLOB加载到字节数组，然后使用FileStream对象将字节数据写入磁盘。

MySqlDataReader的GetOrdinal方法可用于确定命名列的整数索引。如果SELECT查询的列顺序发生变化，使用GetOrdinal方法能够防止错误。

26.2.4.6. 与Crystal Reports一起使用MySQL Connector/NET

26.2.4.6.1. 前言
26.2.4.6.2. 创建数据源
26.2.4.6.3. 创建报告
26.2.4.6.4. 显示报告

26.2.4.6.1. 前言

Crystal Reports是Windows应用程序开发人员用于通报文档生成的常用工具。在本节中，介绍了Crystal Reports XI与MySQL和Connector/NET一起使用的方法。

在MySQL Connector/NET安装的Samples目录的CrystalDemo子目录下，可找到完整的示例应用程序。

26.2.4.6.2. 创建数据源

在Crystal Reports中创建报告时，在设计报告时，有两个用于访问MySQL数据的选项。

第1个选项是，设计报告时，使用Connector/ODBC作为ADO数据源。你能够浏览数据库，并使用拖放式操作选择表和字段以创建报告。该方法的缺点是，必须在应用程序中执行额外操作以生成与报告预期的数据集匹配的数据集。

第2个选项是在VB.NET中创建数据集，并将其保存为XML格式。随后，该XML文件可被用于设计报告。在应用程序中显示报告时，它的表现相当良好，但设计时的通用性较差，这是因为在创建数据集时，必须选择所有的相关列。如果忘记选择了某一列，在能够将列添加到报告前，必须重新创建数据集。

使用下述代码，可根据查询操作创建数据集，并将其写入磁盘。

[VB]

```
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
```

```
        & "pwd=12345;" _
        & "database=world"

    Try
        conn.Open()
        cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " _
            & "country.name, country.population, country.continent " _
            & "FROM country, city ORDER BY country.continent, country.name"
        cmd.Connection = conn

        myAdapter.SelectCommand = cmd
        myAdapter.Fill(myData)

        myData.WriteXml("C:\dataset.xml", XmlWriteMode.WriteSchema)
    Catch ex As Exception
        MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Try
```

[C#]

```
DataSet myData = new DataSet();
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataAdapter myAdapter;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
myAdapter = new MySql.Data.MySqlClient.MySqlDataAdapter();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
  "pwd=12345;database=test;";

try
{
  cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " +
  "country.name, country.population, country.continent " +
  "FROM country, city ORDER BY country.continent, country.name";
  cmd.Connection = conn;

  myAdapter.SelectCommand = cmd;
  myAdapter.Fill(myData);

  myData.WriteXml(@"C:\dataset.xml", XmlWriteMode.WriteSchema);
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
  MessageBox.Show(ex.Message, "Report could not be created",
  MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

设计报告时，可将该代码生成的XML文件用作ADO.NET XML数据源。

如果你选择使用Connector/ODBC来设计报告，可从dev.mysql.com下载它。

26.2.4.6.3. 创建报告

对于大多数应用目的，标准的报告向导应能帮助你完成报告的最初创建。要想启动向导，打开Crystal Reports并从“文件”菜单选择“New > Standard Report”选项。

向导首先要求你提供数据源。如果你正使用Connector/ODBC作为数据源，选择数据源时，请使用OLE DB (ADO)树的“用于ODBC的OLEDB provider”选项，，而不是来自ODBC (RDO)的对应选项。如果你使用的是已保存的数据集，请选择ADO.NET (XML)选项，并浏览你保存的数据集。

在报告的创建过程中，剩余部分将由向导自动完成。

创建完报告后，选择“文件”菜单中的“Report Options...”菜单项。取消对“Save Data With Report”（与报告一起保存数据）选项的选择。这样，就能防止保存的数据干扰应用程序中的数据加载操作。

26.2.4.6.4. 显示报告

要想显示报告，首先用报告所需的数据填充数据集，然后加载报告，并将其与绑定到数据集。最后，将报告传递给crViewer控制，以便向用户显示它。

在显示报告的项目中，需要下述引用：

- CrytalDecisions.CrystalReports.Engine
- CrystalDecisions.ReportSource
- CrystalDecisions.Shared
- CrystalDecisions.Windows.Forms

在下述代码中，假定你使用数据集（用创建数据源中给出的代码保存的数据集）创建了报告，并在名为"myViewer"的表单上有1个crViewer控件。

[VB]

```
Imports CrystalDecisions.CrystalReports.Engine
Imports System.Data
Imports MySql.Data.MySqlClient

Dim myReport As New ReportDocument
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter

conn.ConnectionString = _
    "server=127.0.0.1;" _
    & "uid=root;"
    & "pwd=12345;"_
    & "database=test"

Try
    conn.Open()

    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " _
        & "country.name, country.population, country.continent " _
        & "FROM country, city ORDER BY country.continent, country.name"
    cmd.Connection = conn

    myAdapter.SelectCommand = cmd
    myAdapter.Fill(myData)

    myReport.Load(".\world_report.rpt")
    myReport.SetDataSource(myData)
    myViewer.ReportSource = myReport
Catch ex As Exception
    MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

[C#]

```
using CrystalDecisions.CrystalReports.Engine;
using System.Data;
using MySql.Data.MySqlClient;

ReportDocument myReport = new ReportDocument();
DataSet myData = new DataSet();
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataAdapter myAdapter;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
myAdapter = new MySql.Data.MySqlClient.MySqlDataAdapter();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " +
        "country.name, country.population, country.continent " +
        "FROM country, city ORDER BY country.continent, country.name";
    cmd.Connection = conn;

    myAdapter.SelectCommand = cmd;
```

```
    myAdapter.Fill(myData);

    myReport.Load(@".\world_report.rpt");
    myReport.SetDataSource(myData);
    myViewer.ReportSource = myReport;
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message, "Report could not be created",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

使用相同的查询（用于生成前面保存的数据集），可生成新的数据集。一旦填充了数据集，可使用ReportDocument加载报告文件，并将其与数据集绑定在一起。ReportDocument是作为crViewer的ReportSource而传递的。

使用Connector/ODBC从单个表创建报告时，采用了相同的方法。数据集替换报告中使用的表，并恰当显示报告。

如果报告是使用Connector/ODBC从多个表创建的，在我们的应用程序中必须创建具有多个表的数据集。这样，就能用数据集中的报告替换报告数据源中的各个表。

在我们的MySqlCommand对象中提供多条SELECT语句，通过该方式，用多个表填充数据集。这些SELECT语句基于SQL查询，如数据库菜单"Show SQL Query"选项中的"Crystal Reports"中显示的那样。假定有下述查询：

```
SELECT `country`.`Name`, `country`.`Continent`, `country`.`Population`, `city`.`Name`, `city`.`Population`

FROM `world`.`country` `country` LEFT OUTER JOIN `world`.`city` `city` ON `country`.`Code`=`city`.`CountryCode`

ORDER BY `country`.`Continent`, `country`.`Name`, `city`.`Name`
```

该查询将被转换为两条SELECT查询，并以下述代码显示：

[VB]

```
Imports CrystalDecisions.CrystalReports.Engine
Imports System.Data
Imports MySql.Data.MySqlClient

Dim myReport As New ReportDocument
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=world"

Try
    conn.Open()
    cmd.CommandText = "SELECT name, population, countrycode FROM city ORDER BY countrycode, name; " _
        & "SELECT name, population, code, continent FROM country ORDER BY continent, name"
    cmd.Connection = conn

    myAdapter.SelectCommand = cmd
    myAdapter.Fill(myData)

    myReport.Load(".\world_report.rpt")
    myReport.Database.Tables(0).SetDataSource(myData.Tables(0))
    myReport.Database.Tables(1).SetDataSource(myData.Tables(1))
    myViewer.ReportSource = myReport
Catch ex As Exception
    MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

[C#]

```
using CrystalDecisions.CrystalReports.Engine;
using System.Data;
using MySql.Data.MySqlClient;

ReportDocument myReport = new ReportDocument();
DataSet myData = new DataSet();
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
```

```
MySql.Data.MySqlClient.MySqlDataAdapter myAdapter;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
myAdapter = new MySql.Data.MySqlClient.MySqlDataAdapter();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    cmd.CommandText = "SELECT name, population, countrycode FROM city ORDER " +
        "BY countrycode, name; SELECT name, population, code, continent FROM " +
        "country ORDER BY continent, name";
    cmd.Connection = conn;

    myAdapter.SelectCommand = cmd;
    myAdapter.Fill(myData);

    myReport.Load(@".\world_report.rpt");
    myReport.Database.Tables(0).SetDataSource(myData.Tables(0));
    myReport.Database.Tables(1).SetDataSource(myData.Tables(1));
    myViewer.ReportSource = myReport;
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message, "Report could not be created",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

应将SELECT语句按字母顺序排列，这点很重要，原因在于，这是报告希望其源表所具有的顺序。对于报告中的每个表，均需要一条SetDataSource语句。

该方法会导致性能问题，这是因为Crystal Reports必须在客户端一侧将表绑定在一起，与使用以前保存的数据集相比，速度较慢。

26.2.4.7. 在MySQL Connector/NET中处理日期和时间信息

26.2.4.7.1. 前言
26.2.4.7.2. 使用无效日期时的问题
26.2.4.7.3. 限制无效日期
26.2.4.7.4. 处理无效日期
26.2.4.7.5. 处理NULL日期

26.2.4.7.1. 前言

MySQL和.NET语言处理日期和时间信息的方式是不同的，MySQL允许使用无法由.NET数据类型表示的日期，如“0000-00-00 00:00:00”。如果处理步当，该差异会导致问题。

在本节中，介绍了使用MySQL Connector/NET时恰当处理日期和时间信息的方法。

26.2.4.7.2. 使用无效日期时的问题

对于使用无效日期的开发人员来说，数据处理方面的差异会导致问题。无效的MySQL日期无法被加载到.NET DateTime对象中，包括NULL日期。

由于该原因，不能用MySqlDataAdapter类的Fill方法填充.NET DataSet对象，这是因为无效日期会导致System.ArgumentOutOfRangeException异常。

26.2.4.7.3. 限制无效日期

对日期问题的最佳解决方案是，限制用户输入无效日期。这即可在客户端上进行，也可在服务器端进行。

在客户端上限制无效日期十分简单，即总使用.NET DateTime类来处理日期。DateTime类仅允许有效日期，从而确保了数据库中的值也是有效的。该方法的缺点是，在使用.NET和非.NET代码操作数据库的混合环境下不能使用它，这是因为各应用程序必须执行自己的日期验证。

MySQL 5.0.2和更高版本的用户可使用新的传统SQL模式来限制无效日期值。关于使用传统SQL模式的更多信息，请参见http://dev.mysql.com/doc/mysql/en/server-sql-mode.html。

26.2.4.7.4. 处理无效日期

强烈建议在你的.NET应用程序中应避免使用无效日期，尽管如此，也能tongguo MySqlDateTime数据类型使用无效日期。

MySqlDateTime数据类型支持MySQL服务器支持的相同日期值。MySQL Connector/NET的默认行为是，对有效的日期值返回1个.NET DateTime对象，对无效日期值返回错误。可以更改该默认方式，使MySQL Connector/NET为无效日期返回MySqlDateTime对象。

要想使MySQL Connector/NET为无效日期返回MySqlDateTime对象，可在连接字符串中添加下行：

```
Allow Zero Datetime=True
```

请注意，使用MySqlDateTime类仍会产生问题。下面介绍了一些已知问题：

1. 无效日期的数据绑定仍会导致错误（零日期0000-00-00看上去不存在该问题）。
2. ToString方法返回按标准MySQL格式进行格式处理的日期（例如，2005-02-23 08:50:25）。这与.NET DateTime类的ToString行为不同。
3. MySqlDateTime类支持NULL日期，但.NET DateTime类不支持NULL日期。如果未首先检查NULL，在试图将MySQLDateTime转换为DateTime时，会导致错误。

由于存在上述已知事宜，最佳建议仍是，在你的应用程序中仅使用有效日期。

26.2.4.7.5. 处理NULL日期

.NET DateTime数据类型不能处理NULL值。同样，在查询中为DateTime变量赋值时，必须首先检查值是否是NULL。

使用MySqlDataReader时，在赋值前，应使用.IsDBNull方法检查值是否为NULL：

[VB]

```
If Not myReader.IsDBNull(myReader.GetOrdinal("mytime")) Then
    myTime = myReader.GetDateTime(myReader.GetOrdinal("mytime"))
Else
    myTime = DateTime.MinValue
End If
```

[C#]

```
if (! myReader.IsDBNull(myReader.GetOrdinal("mytime")))
    myTime = myReader.GetDateTime(myReader.GetOrdinal("mytime"));
else
    myTime = DateTime.MinValue;
```

NULL值能够在数据集中使用，也能将其绑定以构成控件，无需特殊处理。

26.2.5. MySQL Connector/NET变更史

26.2.5.1. 版本2.0.0
26.2.5.2. 版本1.0.7
26.2.5.3. 版本1.0.6
26.2.5.4. 版本1.0.5
26.2.5.5. 版本1.0.4 1-20-05
26.2.5.6. 版本1.0.3-gamma 12-10-04
26.2.5.7. 版本1.0.2-gamma 04-11-15
26.2.5.8. 版本1.0.1-beta2 04-10-27
26.2.5.9. 版本1.0.0 04-09-01

26.2.5.10. 版本0.9.0 04-08-30
26.2.5.11. 版本0.76
26.2.5.12. 版本0.75
26.2.5.13. 版本0.74
26.2.5.14. 版本0.71
26.2.5.15. 版本0.70
26.2.5.16. 版本0.68
26.2.5.17. 版本0.65
26.2.5.18. 版本0.60
26.2.5.19. 版本0.50

26.2.5.1. 版本2.0.0

- 更正了在未填充Connection.Database的情况下使用存储程序时出现的异常 (Bug #11450) 。
- 特定的残缺查询将触发"连接必须是有效和打开的"错误消息 (Bug #11490) 。

26.2.5.2. 版本1.0.7

- 调用其某一参数含有特殊字符（如"@"）的存储程序将产生异常。注意，必须启用ANSI_QUOTES才会使之成为可能 (Bug #13753) 。
- 如果语句包含对相同参数的多个引用，无法对其进行预处理 (Bug #13541) 。
- Ping()方法不更新Connection对象的State属性 (Bug #13658) 。

26.2.5.3. 版本1.0.6

- Nant构建序列有问题(Bug #12978)
- 如果传递的第1个值是NULL，参数的串行化操作失败 (Bug #13276) 。
- 含下述字符的字段名将导致错误：()%<>/ (Bug #13036) 。
- MySQL Connector/NET 1.0.5安装程序不能同时安装MySQL Connector/NET 1.0.4. (Bug #12835)。
- 在Mono上MySQL Connector/NET 1.0.5不能连接 (Bug #13345) 。

26.2.5.4. 版本1.0.5

- 连接字符串中有多个主机时，MySQL Connector/NET无法与列表中的1个主机相连 (Bug #12628) 。
- MySQL Connector/NET将新的十进制数据类型解释为字节数组 (Bug #11294) 。
- 不支持cp1250字符集 (Bug #11621) 。
- 当.NET线程池无可用的工作线程时，连接可能失败 (Bug #10637) 。
- 十进制参数导致语法错误 (Bug #11550, Bug #10486, Bug #10152)。
- 如果存储程序不含参数，调用存储程序将导致异常 (Bug #11542) 。
- 特定的残缺查询将触发"连接必须是有效和打开的"错误消息 (Bug #11490) 。
- 除了默认数据库外，MySqlCommandBuilder类不能处理引用了其他数据库中表的查询 (Bug #8382) 。

- MySQL Connector/NET无法特定的局部设置一起正常工作 (WL#8228)。
- 未填充Connection.Database时使用存储程序导致异常 (Bug #11450) 。
- 读取TIMESTAMP列时产生异常 (Bug #7951) 。
- 用换行符隔开参数时，无法识别参数 (Bug #9722) 。
- 在初始连接上未设置连接字符串时，调用MySqlConnection.clone将导致错误 (Bug #10281) 。
- 增加了对从MySQL Connector/NET调用存储函数的支持 (Bug #10644) 。
- MySQL Connector/NET不能连接到MySQL 4.1.14. (Bug #12771)。
- 用设计器添加了MySqlConnection对象时，无法设置ConnectionString属性 (Bug #12551, Bug #8724)。

26.2.5.5. 版本1.0.4 1-20-05

- Bug #7243：调用准备导致异常[已更正]。
- 更正了与预处理语句有关的另一个小问题。
- Bug #7258：MySqlCommand.Connection返回IDbConnection [已更正]。
- Bug #7345：MySqlAdapter.Fill方法抛出错误消息：需要非负数值[已更正]。
- Bug #7478：MySqlCommand中的克隆方法缺陷[已更正]。
- Bug #7612：当字段为NULL时，MySqlDataReader.GetString(index)返回了非Null值[已更正]。
- Bug #7755：如果列是无符号类型，MySqlReader.GetInt32抛出异常[已更正]。
- Bug #7704：GetBytes不再工作[已更正]。
- Bug #7724：引用字符"\222"在EscapeString中未被引用[已更正]。
- 更正了命名管道不能与某些Blob功能一起工作的问题。
- 更正了与共享内存连接有关的问题。
- Bug #7436：与多个结果集有关的问题... [已更正]。
- 在API参考文档中增加了多个主题。

26.2.5.6. 版本1.0.3-gamma 12-10-04

- 使MySQL成为默认的命名管道名称。
- 现在，连接时SHOW COLLATION可用于检索完整的字符集ID列表。
- 更正了无效字符集索引：200 (Bug #6547) 。
- 安装器现在包含了多个选项，可安装至GAC中，并创建"开始"菜单项。
- Bug #6863：MySqlCommand参数中的Int64支持[已更正]。

- 对于连接，现在无需在连接字符串上给出数据库。
- Bug #6770：MySqlDataReader.GetChar(int i)抛出IndexOutOfRange异常[已更正]。
- 更正了因具有不同行数的多个结果集而导致的问题。
- Bug #6983：再次抛出异常时异常堆栈跟踪丢失[已更正]。
- 更正了与使用预处理语句检测Null值有关的主要问题。
- Bug #6902：解析存储程序参数时的错误[已更正]。
- Bug #6668：存储程序的整数输出参数返回为字符串[已更正]。
- Bug #7032：按文本分类的数据表中的MySqlDateTime，无数据 [已更正]。
- Bug #7133：使用inout参数时的无效查询字符串[已更正]。
- Bug #6831：与MySQL 4.0一起时，测试失败，原因在于表名的大小写敏感性[已更正]。
- Bug #7132：插入DateTime导致System.InvalidCastException的抛出[已更正]。
- Bug #6879：使用DATE_ADD-function时的InvalidCast[已更正]。
- Bug #6634：1个打开的连接被主机系统关闭[已更正]。
- 为MySqlConnection添加了ServerThread属性以显示服务器线程ID。
- 为MySqlConnection增加了Ping方法。
- 将测试包的名称更改为MySql.Data.Tests.dll。

26.2.5.7. 版本1.0.2-gamma 04-11-15

- 更正了与MySqlBinary有关的问题，其中，无法使用字符串值更新扩展的文本列。
- 更正了使用定制安装时忽略的安装目录问题(Bug #6329)。
- 更正了设置命令文本将命令留在预处理状态的问题。
- 更正了MySqlParameter双类型处理问题（字符串parameterName，对象值）(Bug #6428)。
- 更正了填充数据集时返回零日期"0000-00-00"错误(Bug #6429)。
- 更正了调用存储程序可能会导致"Illegal mix of collations"（非法校对组合）的问题。
- 增加了charset连接字符串选项。
- 更正了#HY000"Illegal mix of collations"（非法校对组合）(latin1_swedish_ci,IMPLICIT)和(utf8_general_ (Bug #6322)问题。
- 增加了TableEditor CS和VB示例。
- 更正了关于UCS-2的Charset-map问题(Bug #6541)。
- 更新了安装器，包含了新的示例。
- 更正规了Long插入耗时很长的问题(Bug #5453)。

- 更正了对象无法被处理的问题(Bug #6649)。
- 提供方正将服务器指定的字符集用作默认字符集。

26.2.5.8. 版本1.0.1-beta2 04-10-27

- 更正了MySqlParameter(string, object)构造函数中的可能缺陷BUG #5602。
- 更正了BUG #5458，在longtext列上调用GetChars将抛出异常。
- 更正了BUG #5474，无法运行存储程序来填充mysqlcommand.parameters。
- 更正了BUG #5469，设置DbType时抛出NullReferenceException。
- 更正了在关闭套接字之前连接器无法发出CMD_QUIT的问题。
- 更正了BUG #5392，MySqlCommand在字符串文本内容中发现作为参数的"?"。
- 更正了与ConnectionInternal有关的问题，其中，1个键可能会被添加多次。
- 当服务器版本为4.1.2或更高时，CP1252仅用于Latin1。
- 更正了BUG #5388，如果1行为NULL，DataReader通报所有行均为NULL。
- 虚拟化了驱动子系统，以便未来版本能轻易地支持客户端或嵌入式服务器。
- 再次使用字段缓冲，以减少内存分配并增加速度。
- 更正了相应的问题，使用接口时使用旧语法将导致问题。
- 对于写入流操作，使用PacketWriter取代Packet。
- 在CompressedStream中再分解压缩代码，以清理NativeDriver。
- 增加了用于在预处理命令上重置命令文本的测试范例。
- 更正了给定Null值时MySqlParameterCollection.Add()将抛出不明异常的问题(Bug #5621)。
- 更正了MySqlCommand()中的构造函数初始化问题(Bug #5613)。
- 更正了解析";"字符的问题(Bug #5876)。
- 更正了在DbType设置器中丢失引用的问题(Bug #5897)。
- 更正了使用YEAR数据类型时的System.OverflowException问题(Bug #6036)。
- 增加了聚合函数测试（实际上不是缺陷）。
- 更正了浮点参数(double, numeric, single, decimal)的序列化问题(Bug #5900)。
- IsNullable错误(Bug #5796)。
- 更正了不遵守连接字符串上给出的连接寿命的问题。
- 更正了不遵守Min Pool Size（最小池大小）的问题。
- MySqlDataReader "show tables from ..." ... (Bug #5256)

更正了 和 （从 显示表）行为 。

- 实施了SequentialAccess。
- 更正了发现第1个0后MySqlDateTime在所有subseq.records上设置IsZero属性的问题(Bug #6006)。
- 更正了无法正确显示中文字的问题(Bug #5288)。
- 还更正了俄文字符支持。
- 更正了Method TokenizeSql()仅将有限的有效字符用于参数的问题(Bug #6217)。
- 更正了丢失resx文件的NET Connector源(Bug #6216)。
- 更正了与检索／更新查询一起使用是会导致问题的DBNull值 (Bug #5798) 。
- 更正了仍有另一个"未设置给对象实例的对象引用"(Bug #5496)。
- 更正了PacketReader中的问题，其中，会试图在EnsureCapacity中分配错误的缓冲大小。
- 更正了GetBoolean返回错误值的问题(Bug #6227)。
- 更正了带有GetString(index)的DataReader一起读取BLOB时的IndexOutOfBounds问题(Bug #6230)。

26.2.5.9. 版本1.0.0 04-09-01

- 更正了BUG# 3889，不能正确支持Thai编码。
- 更新了很多测试范例。
- 更正了与使用压缩有关的问题。
- 将贝塔1版本的版本号扩充为1.0.0。
- 增加了用于安装器的COPYING.rtf文件。
- 删除了所有的XML注释警告（以后将更好地清理它们）。
- 删除了一些对ByteFX的最近引用。

26.2.5.10. 版本0.9.0 04-08-30

- 为预处理语句增加了测试定位器。
- 目前，所有类型的类均实施了SerializeBinary方法，用于将其数据发送给PacketWriter。
- 增加了PacketWriter类，允许将来的低内存大对象处理。
- 更正了运行预处理语句和存储程序中存在的很多小缺陷。
- 更改了多条命令，使得在执行带有特定参数（采用旧语法模式）的存储程序时不再抛出异常。
- SingleRow现在工作正常，即使在存在限制的情况下也同样。
- GetBytes目前仅作用在二进制列上。

- Logger现在能够截短长的SQL命令，从而使得blob列不会"撑爆"日志。
- 主机和数据库目前的默认值为"'"，除非作了其他设置。
- 更正了BUG# 5214，忽略了连接超时。
- 增加了测试范例，针对bug# 5051：GetSchema不能正确工作。
- 更正了当列为关键字时GetSchema为IsUnique返回"假"的问题。
- MySqlDataReader GetXXX方法目前采用了字段级MySqlValue对象，不执行转换。
- 更正了BUG# 5097：DataReader为时间列返回NULL。
- A增减了针对LOAD DATA LOCAL INFILE的测试范例。
- 增加了replacetext custom nant任务。
- 增加了CommandBuilderTest定位器。
- 为CommandBuilder增加了Last One Wins（最后一个胜出）特性。
- 更正了持续性安全信息问题。
- 更正了GetBool，使得1, true, "true"和"yes"均可表示trueWL# 2024，从而使得参数标志成为可配置的。
- 增加了"old syntax"连接字符串参数，允许使用"@"参数标记符。
- 更正了Bug #4658，MySqlCommandBuilder。
- 更正了Bug #4864，如果"Persist Security Info"（持续性安全信息）为假，ByteFX.MySqlClient将对密码进行缓冲处理。
- 在所有的源文件中更新了许可标志，以包含FLOSS异常。
- 针对目前所有的MySql类型，增加了新的.Types名称空间和具体实施。
- 增加了作为MySqlField子类的MySqlField41。
- 更改了很多类，使之能够使用新的.Types类型。
- 将enum int类型更改为Int32，将short类型更改为Int16，并将bigint类型更改为Int64。
- 增加了伪类型UInt16、UInt32和UInt64，允许创建无符号参数。
- 现在，从连接池拉出连接时，连接将被复位。
- 在驱动程序中再次分解了auth代码，使得其即能用于auth，也能用于reset。
- 在PoolingTests.cs中增加了UserReset测试。
- 现在，使用COM_CHANGE_USER从池中拉出连接时，连接将被复位。
- 实现了SingleResultSet行为。
- 实现了对unicode的支持。
- 为utf-8和ucs-2增加了字符集映射。

- 更正了Bug #4520，使用bytefx .net mysql驱动时，时间字段溢出。
- 在数据类型测试定位器中修改了时间测试，以便能够检查"hours > 24"的时间跨度。
- 更正了Bug #4505，在ByteFx.Data.MySqlClient.MySqlParameter中带有反斜杠转义的错误字符串。
- 为参数测试范例TestQuoting增加了代码，以测试反斜线符号。
- 更正了Bug #4486，与multi-word列名一起工作时，mysqlcommandbuilder失败。
- 更正了TokenizeSql中的缺陷，其中，下划线将中止获取参数名中的字符。
- 为列名空间增加了测试范例。
- 更正了bug# 4324，MySqlDataReader.GetBytes不能正确工作。
- 为DataReader测试定位器增加了GetBytes()测试范例。
- 现在，能够将InternalConnection.Configure中的所有服务器变量读入到Hashtable。
- 目前使用字符串[]，用于CharSetMap中的索引映射。
- 为SQL中的carriage返回增加了CRInSQL测试范例。
- 在Driver.ctor中，将maxPacketSize设为默认值。
- 更正了bug #4442，在参数上设置MySqlDbType的操作不设置一般类型。
- 删除了过时的列类型Long和LongLong。
- 更正了bug# 4071，在连接字符串上使用"use pipe"时，抛出溢出异常。
- 将关键字"use pipe"更改为"pipe name"或"pipe"。
- 允许从单个查询读取多个结果集。
- 为ServerStatusFlags enum增加了标志属性。
- 将ServerStatus enum的名称更改为ServerStatusFlags。
- 更正了BUG #4386，插入的数据行未正确更新。
- 更正了bug #4074，错误处理表明创建了表。
- 将Packet.ReadLenInteger更改为ReadPackedLong，并增加了packet.ReadPackedInteger，它总读取用2、3、4组装的整数。
- 增加了syntax.cs测试定位器，以测试各种SQL语法缺陷。
- 更正了bug# 4149，对时间值的不当处理。现在，值"00:00:00"不再被当作Null。
- 将所有的测试包文件移到了TestSuite文件夹。
- 更正了空列会将结果信息包指针向后移的问题。
- 增加了新的nant创建脚本。
- 更正了BUG #3917，清除表名，以便能在下一GenerateSchema执行期间恰当地重新生成它。

- 更正了bug #3915，GetValues总返回0，而且总是试图复制所有字段，而不是根据所传入数组的大小。
- 实施了共享内存访问协议。
- 实施了针对的MySQL 4.1的预处理语句。
- 实施了针对MySQL 5.0的存储程序。
- 将MySqlInternalConnection重新命名为InternalConnection。
- SQL现在被解释为字符，更正了与其他语言有关的问题。
- 增加了日志功能，并允许批连接字符串选项。
- 更正了bug #3888，设置DataAdapter属性时未设置RowUpdating事件。
- 更正了字符集映射中存在的缺陷。
- 实施了4.1鉴定。
- 改善了驱动中的open/auth代码。
- 改善了在连接过程中连接位的设置方式。
- 现在，在初始的握手阶段，将数据库名传递给了服务器。
- 将客户端的名称空间更改为MySql.Data.MySqlClient。
- 将客户端的装配名称更改为MySql.Data.dll。
- 将所有源文件中的许可文本更改为了GPL。
- 增加了MySqlClient.build Nant文件。
- 删除了mono批处理文件。
- 将一些未使用的文件移到了notused文件夹，从而使得nant创建文件能够使用通配符。
- 实施了共享内存访问。
- 对代码结构进行了较大修补。
- 现在，预处理语句能够在MySql 4.1.1和更高版本中使用。
- 对4.0、4.1.0和4.1.1完成了auth实施。
- 将名称空间从MySQL.Data.MySQLClient更改为MySql.Data.MySqlClient。
- 更正了CharSetMapping中存在的缺陷，其中，它试图将文本名称用作ints。
- 将名称空间更改为MySQL.Data.MySQLClient。
- 集成了来自UC2004的auth变动。
- 更正了在读取数据之前和值后、在datareader上调用任何GetXXX方法时不能抛出恰当异常的缺陷（感谢Luca Morelli morelli.luca@iol.it）。
- 在parameter.cs中增加了TimeSpan代码，以便能恰当地将timespan对象处理为mysql时间格式（感谢Gianluca Colombo g.colombo@alfi.it）。

- 为参数序列化代码增加了TimeStamp。防止DataAdatper不正常的更新（感谢MIchael King）。
- 更正了MySqlHelper.cs中的拼写错误（感谢Patrick Kristiansen）。

26.2.5.11. 版本0.76

- 驱动程序现在能使用握手协议中给定的字符集编号创建编码。
- 更改了命令编辑器，使之指向MySqlClient.Design。
- 更正了Version.isAtLeast中的缺陷。
- 更改了DBConnectionString，使之能够支持对MySqlConnectionString所作的更改。
- 删除了SqlCommandEditor和DataAdapterPreviewDialog。
- 在很多地方采用了新的Long返回值。
- 集成了新的CompressedStream类。
- 更改了ConnectionString并增加了多项属性，从而使之能够在MySqlClient.Design中使用。
- 更改了packet.cs，以支持ReadLenInteger中的较新长度。
- 更改了其他类，以使用MySqlConnectionString的新属性和字段。
- 现在，ConnectionInternal能够使用PING命令查看服务器是否可用。
- 将工具箱位图移到了resource/下。
- 更改field.cs，允许值直接来自行缓冲器。
- 进行了相应的更改，以使用新的driver.Send语法。
- 使用了新的信息包排队系统。
- 开始着手进行"损坏的"压缩信息包处理。
- 更正了StreamCreator中的缺陷，无法连接到主机将导致无限循环（感谢Kevin Casella）。
- 改善了connectstring处理。
- 将设计器移到了Pro产品中。
- 从command.cs删除了一些旧的、被注释掉的代码。
- 更正了与压缩有关的1个问题。
- 更正了连接对象，打开连接前抛出的异常不会使连接保持在连接状态（感谢Chris Cline）。
- 增加了GUID支持。
- 更正了序列混乱缺陷（感谢Mark Reay）。

26.2.5.12. 版本0.75

- 现在，可将Enum值作为参数值加以支持（感谢Philipp Sumi）。
- 支持Year数据类型。
- 更正了压缩问题。
- 更正了以TimeSpan作为值的参数无法恰当序列化的缺陷。
- 更正了默认ctor不能设置默认连接字符串值的缺陷。
- 为一些新成员增加了一些XML注释。
- 着手更正／改善压缩处理事宜。
- 改善了ConnectionString处理功能，使之能够与SqlClient设定的标准更好地匹配。
- 如果用户名未包含在连接字符串中，将抛出MySqlException。
- 如果在连接字符串中未指定，本地主机将用作默认主机。
- 如果在连接打开的同时试图设置连接字符串，将抛出异常。
- 对ConnectionString文档进行了小的修改。
- 删除了MultiHostStream和MySqlStream。采用Common/StreamCreator取而代之。
- 增加了对"Use Pipe"连接字符串值的支持。
- 增加了Platform类，以便能更容易地访问平台的实用工具功能。
- 更正了小的连接池缺陷，即，在IsAlive失败后，不能创建新的连接。
- 增加了Platform.cs和StreamCreator.cs。
- 更正了Field.cs，以便能恰当处理4.1版分格的时间戳。
- 将Common.Version更改为Common.DBVersion，以避免名称冲突。
- 更正了field.cs，从而使得文本列能返回正确的字段类型（感谢beni27@gmx.net）。
- 增加了MySqlError类，以提供对错误代码的一些引用（感谢Geert Veenstra）。

26.2.5.13. 版本0.74

- 增加了Unix套接字支持（感谢Mohammad DAMT [md@mt.web.id]）。
- 没有可用数据时，仅调用Thread.Sleep。
- 该井了参数数据中引用字符的转义特性。
- 删除了parameter.cs中易造成误解的注释。
- 更正了连接池缺陷。
- 再次更正了相同的连接池缺陷！！ ;-)
- 更正了ConnectionSTring编辑器对话框（感谢marco p (pomarc)）。

- 现在，在连接字符串中支持UserId（感谢Jeff Neeley）。
- 创建非输入参数时抛出异常（感谢Ryan Gregg）。
- 增加了更多文档。
- 提供了新的MultiHostStream能力。诚挚感谢Dan Guisinger对此的贡献。是他首次提供了在连接字符串上支持多台机器的代码和观念。
- 增加了大量文档。仍有很多文档需要增加。
- 更正了与0.73有关的速度事宜。
- 更改了MySqlDataStream中的Thread.Sleep(0)，以便在不需要等待时优化性能（感谢Todd German）。
- 预先将idlepools填充到了MinPoolSize。
- 个高质量MySqlPool死锁条件以及愚蠢的缺陷，其中，CreateNewPooledConnection从不为连接池添加新连接。此外，还更正了MySqlStream.ReadBytes和ReadByte，从而不再使用并非始终正确的TicksPerSecond。（感谢Matthew J. Peddlesden）。
- 修正了精度和标度（感谢Matthew J. Peddlesden）。
- 为流读取方法增加了Thread.Sleep(1)，使之对CPU更友好（感谢Sean McGinnis）。
- 更正了ExecuteReader有时会返回Null的问题（感谢Lloyd Dupont）。
- 更正了与Null字段处理有关的主要缺陷（感谢Naucki）。
- 封装了针对max_allowed_packet的查询，以及Try Catch中的字符集（并设置为默认）。
- 更正了套接字未能恰当关闭的问题（感谢Steve）。
- 更正了ExecuteNonQuery不能始终返回正确值的问题。
- 更正了InternalConnection，不使用@@session.max_allowed_packet，而是使用@@max_allowed_packet。（感谢Miguel）。
- 增加了很多新XML文档行。
- 更正了SQL解析功能，不发送控查询（感谢Rory）。
- 更正了阅读器在关闭时不能unpeeking信息包的问题。
- 更正了不能处理用户变量的问题（感谢Sami Vaaraniemi）。
- 更正了MySqlPool中的循环检查功能（感谢Steve M. Brown）。
- 更正了ParameterCollection.Add方法，以与SqlClient匹配（感谢Joshua Mouch）。
- 更正了ConnectionSTring解析功能，以处理布尔类型的NO和YES，以及非小写值（感谢Naucki）。
- 增加了InternalConnection类，修改了连接池功能。
- 实现了Persist Security Info（持续性安全信息）。
- 为项目增加了security.cs和version.cs。
- 更正了Parameter.cs中的DateTime处理功能（感谢Burkhard Perkens-Golomb）。

- 更正了某些类型抛出cast异常的参数序列化问题。
- 更正了DataReader，转换所有的返回值以防止抛弃错误（感谢Keith Murray）。
- 为Command.ExecuteReader增加了代码，如果初始SQL命令抛出异常，将返回Null（感谢Burkhard Perkens-Golomb）。
- 构造了与重组一起引入ExecuteScalar缺陷。
- 进行了重新构造，允许LOCAL DATA INFILE，以及更好的信息包排序。
- 更正了与重组有关的数个缺陷。
- 完成了前期工作，支持Mysql 4.1中更安全的密码。不再支持4.1版中的旧密码。
- 正确处理系统参数后显示参数（Adam M. (adammil)）。
- 现在，可将字符串直接赋给blob字段（Adam M.）。
- 更正了浮点参数（感谢Pent）。
- 改善了参数ctor和ParameterCollection.Add方法，以更好地匹配SqlClient（感谢Joshua Mouch）。
- 更正了Connection.CreateCommand以返回MySqlCommand类型。
- 更正了连接字符串设计器的对话框问题（感谢Abraham Guyt）。
- 更正了与发送命令无法总是读取响应信息包有关的问题（感谢Joshua Mouch）。
- 更正了某些Blob类型无法被处理的参数序列化问题（感谢Sean McGinnis）。
- 从DataReader代码中删除了伪MessageBox.show（感谢Joshua Mouch）。
- 更正了split sql代码中的丑陋缺陷（感谢所有人！ :-))

26.2.5.14. 版本0.71

- 更正了MySqlStream中的缺陷，即可能会读取过多数据（感谢Peter Belbin）。
- 实现了HasRows（感谢Nash Pherson）。
- 更正了大于256列的表会导致异常的问题（感谢Joshua Kessler）。
- 更正了以“;”结束的SQL语句会导致问题的缺陷（感谢Shane Krueger）。
- 更正了驱动中的缺陷，即，错误消息被截去1个字符（感谢Shane Krueger）。
- 使得MySqlException成为可序列化的（感谢Mathias Hasselmann）。

26.2.5.15. 版本0.70

- 更新了一些字符代码页，使之更加准确。
- 更正了阅读器能够在已有打开阅读器的连接上打开的问题。
- 发布了0.70。

- 将测试移至单独的MySqlClientTests下。
- 更正了驱动程序序列混乱的愚蠢问题（感谢Peter Belbin）。
- 增加了一些管道测试。
- 将默认最大池大小增加到50。
- 与Mono 0-24一起进行了编译。
- 更正了连接和数据阅读器处理问题。
- 为参数序列化增加了字符串数据类型处理功能。
- 更正了抛出异常后在驱动程序中出现的顺序问题（感谢Burkhard Perkens-Golomb）。
- 增加了对CommandBehavior.SingleRow到DataReader的支持。
- 更正了命令sql的处理功能，以便能更好地处理引用（感谢Theo Spears）。
- 更正了double、single和decimal值的解析问题，以解释非英文分隔符。如果你正使用硬编码sql，仍须使用正确的语法，但是，如果你使用参数，代码将转换浮点类型，以便在进出服务器的过程中恰当地在内部使用"."。[感谢匿名人]。
- 增加了MySqlStream类，以简化超时和驱动编码。
- 更正了DataReader，以便在相关连接关闭时恰当地关闭它。[感谢smishra]。
- 使得客户端更兼容SqlClient，在连接能够用于运行另一命令前关闭DataReaders。
- 改进了字段中的DBNull.Value处理功能。
- 增加了数个单元测试。
- 更正了MySqlException，以便能调用基本类：-o
- 改进了驱动编码。
- 更正了NextResult在最后1个结果集上返回"假"的缺陷。
- 为MySQL增加了多个测试。
- 通过等化无符号32bit值和Int64，以及无符号16bit值和Int32等，改进了抛弃问题。
- 为MySqlParameter增加了新的ctor（名称、类型、大小、srccol）。
- 更正了MySqlDataReader中存在的问题，即，在返回字段计数前，不能检查空的字段列表。
- 开始增加了MySqlClient单元测试（增加了MySqlClient/Tests文件夹以及一些测试范例）。
- 更正了连接字符串处理中的一些问题。
- 将INIT_DB移到MySqlPool。可或许会在此移动它，这是在协商的准备过程中。
- 更正了CommandBuilder中存在的缺陷，该缺陷会阻止插入正确出现。
- 改写了一些内部构件，从而使得Command的所有三种执行方法均能正确工作。
-

更正了在基准测试过程中发现的一些小问题。

- CoonectionPooling的首次截除工作恰当。保留了"min pool size"和"max pool size"。
- 进行处理，允许返回多个结果集。
- 现在，字符集的处理更为智能化。启动时，驱动程序查询MySQL，寻找默认的字符集。随后，如果能够加载代码页，该字符集将用于转换。如不然，将使用当前操作系统的默认代码页。
- 增加了代码，以便将推断的类型保存在名称，以及参数的值ctor中。
- 此外，如果使用Value属性更改了空参数的值，还能推断类型。
- 转换了所有的文件以使用恰当的Camel范例。现在，在所有文件中，MySQL是MySql。PgSQL现在是PgSql。
- 为PgSql代码增加了属性，以防止设计器显示它。
- 为参数对象增加了MySQLDbType属性，并为从DbType到MySQLDbType的转换增加了恰当的转换代码。
- 从MySQLParameter.cs中删除了从未使用的ObjectToString方法。
- 更正了ParameterCollection中的Add(..)方法，不必使用Add(name, value)取而代之。
- 更正了ParameterCollection中的IndexOf和Contains，使之清楚保存参数名时不需要@。
- 更正了Command.ConvertSQLToBytes，仅允许能够构出现在MySQL变量名中的字符。
- 更正了DataReader和字段，从而使得Blob字段能够从Field.cs读取其数据，而且GetBytes工作正确。
- 为MySQLCommand的CommandText属性增加了简单的构造器编辑器。
- 更正了CommandBuilder和Parameter序列化，指明在参数名称中不保存@。
- 从Field.cs删除了MySQLFieldType enum，现使用MySQLDbType enum。
- 为数个类增加了Designer属性，防止了使用VS.Net时的设计器视图。
- 更正了ConnectionString设计器中的初始目录类型。
- 删除了与（名称、类型、值）冲突的3种MySQLParameter参数ctor。
- 更改了MySQLParameter，现在能够保存paramName而无需前导@（这修正了使用设计器是的Null插入问题）。
- 更改了用于MySQLParameter的TypeConverter，以便能够与所有属性一起使用ctor。

26.2.5.16. 版本0.68

- 更正了驱动程序中的顺序问题。
- 增加了DbParametersEditor，使得参数编辑更像SqlClient。
- 更正了Command类，以便能够使用设计器编辑参数。
- 更新了连接字符串设计器，支持使用压缩标志。
- 更正了字符串编码功能，从而使得欧洲字符（如ä）能够正确工作。

- 创建了基本类，以帮助创建新的数据Provider。
- 在连接字符串中增加了对UID关键字的支持。
- 字段、参数和命令现在都能使用DBNull.Value，而不是null。
- 使用DBNull.Value的CommandBuilder。
- 未出现auto_insert字段时，CommandBuilder现在能正确创建插入命令。
- 现在，字段使用typeof关键字来返回System.Types（性能）。

26.2.5.17. 版本0.65

- 目前实现了MySQLCommandBuilder。
- 目前实现了事务支持（并非所有的表类型均支持它）。
- 更正了GetSchemaTable，不再使用xsd（对于Mono）。
- 驱动程序先能兼容Mono！！
- 现在支持TIME数据类型。
- 需要更多工作以改善Timestamp数据类型处理。
- 更改了所有类的特征以匹配对应的SqlClient类。

26.2.5.18. 版本0.60

- 采用SharpZipLib的协议压缩（www.icsharpcode.net）。
- Windows平台上的命名管道现工作正常。
- 完成了更多工作，改善了Timestamp数据类型处理。
- 在DataReader上实现了Ienumerable，以使DataGrid能恰当工作。

26.2.5.19. 版本0.50

- 通过删除网络同步代码中的缺陷，大幅度提高了速度。
- 驱动程序不再对数据行进行缓冲处理（更兼容ADO.Net）。
- 更正了与TIMESTAMP和DATETIME字段有关的转换缺陷。

26.3. MySQL Connector/J

26.3.1. 基本的JDBC概念
26.3.2. 安装 Connector/J
26.3.3. JDBC引用
26.3.4. 与J2EE和其他Java框架一起使用 Connector/J
26.3.5. 诊断 Connector/J方面的问题
26.3.6. Changelog

通过JDBC驱动，MySQL提供了与使用Java编程语言开发的客户端应用程序的连通性，该驱动称为MySQL Connector/J。

MySQL Connector/J是一种JDBC-3.0"类型4"驱动，这意味着它是一种纯Java程序，实施了3.0版JDBC规范，并能使用MySQL协议与MySQL服务器直接通信。

本文档是为初级JDBC开发人员准备和安排的。如果你已有了使用JDBC方面的经验，可直接从安装 Connector/J开始。

尽管JDBC本身很有用，但我们希望，如果你在阅读完本手册的前几节后尚未熟悉JDBC，除了最平常的问题外应避免全面使用"裸"JDBC，应考虑使用流行的架构，如Hibernate、Spring的JDBC模板或Ibatis SQL Maps等，使用它们来完成大多数重复性工作，以及在某些时侯需要用到JDBC的繁重任务。

本节不是作为完整的JDBC教程而设计的。如果需要了解使用JDBC方面的更多信息，或许会对下述在线教程感兴趣，与这里提供的信息相比，它们介绍的更为详细和更具深度。

- JDBC基础，Sun公司提供的教程，涵盖了JDBC的基本主题。。
- JDBC简明课程，Sun和JGuru提供了更深的教程。

26.3.1. 基本的JDBC概念

26.3.1.1. 使用DriverManager接口连接到MySQL
26.3.1.2. 使用语句以执行SQL
26.3.1.3. 使用CallableStatements以执行存储程序
26.3.1.4. 检索AUTO_INCREMENT列的值

在本节中，介绍一些一般性的JDBC背景知识。

26.3.1.1. 使用DriverManager接口连接到MySQL

在应用服务器外使用JDBC时，DriverManager类将用于管理连接的建立。

需要告诉DriverManager应与哪个JDBC驱动建立连接。完成该任务的最简单方法是：在实施了java.sql.Driver接口的类上使用Class.forName()。对于MySQL Connector/J，该类的名称是com.mysql.jdbc.Driver。采用该方法，可使用外部配置文件来提供连接到数据库时将使用的驱动类名和驱动参数。

在下面的Java代码中，介绍了在应用程序的main()方法中注册MySQL Connector/J的方式：

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

// Notice, do not import com.mysql.jdbc.*
// or you will have problems!（注意，不要导入com.mysql.jdbc.*，否则// 将出现问题！）

public class LoadDriver {
    public static void main(String[] args) {
        try {
            // The newInstance() call is a work around for some
            // broken Java implementations

            Class.forName("com.mysql.jdbc.Driver").newInstance();
        } catch (Exception ex) {
            // handle the error
```

```
        }
}
```

在DriverManager中注册了驱动后，通过调用DriverManager.getConnection()，能够获得与特殊数据库相连的连接实例。

示例**26.1**：从**DriverManager**获得连接

在本示例中，介绍了从DriverManager获得连接实例的方法。对于getConnection()方法，有一些不同的特性。关于如何使用它们的更多信息，请参阅与JDK一起提供的API文档。

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

    ... try {
            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost/test?user=monty&password=greatsqldb");

            // Do something with the Connection

           ....
        } catch (SQLException ex) {
            // handle any errors
            System.out.println("SQLException: " + ex.getMessage());
            System.out.println("SQLState: " + ex.getSQLState());
            System.out.println("VendorError: " + ex.getErrorCode());
        }
```

一旦建立了连接，它可被用于创建语句和PreparedStatements，并检索关于数据库的元数据。在下面数节内，给出了进一步的解释。

26.3.1.2. 使用语句以执行SQL

使用语句，可执行基本的SQL查询，并通过下面介绍的ResultSet类检索结果。

要想创建语句实例，应通过前面介绍的DriverManager.getConnection()或DataSource.getConnection()方法之一，在检索的连接对象上调用createStatement()方法。

一旦拥有了语句实例，可以与希望使用的SQL一起通过调用executeQuery(String)方法执行SELECT查询。

要想更新数据库中的数据，可使用executeUpdate(String SQL)方法。该方法将返回受更新语句影响的行数。

如果你事先不清楚SQL语句是SELECT或UPDATE/INSERT，应使用execute(String SQL)方法。如果SQL查询是SELECT，本方法将返回"真"，如果SQL查询是UPDATE/INSERT/DELETE，本方法将返回"假"。如果是SELECT查询，能够通过调用getResultSet()方法检索结果。如果是UPDATE/INSERT/DELETE查询，能够通过在语句实例上调用getUpdateCount()检索受影响的行计数。

示例**26.2**：使用**java.sql.Statement**执行**SELECT**查询

```
// assume conn is an already created JDBC connection
Statement stmt = null;
ResultSet rs = null;

try {
    stmt = conn.createStatement();
    rs = stmt.executeQuery("SELECT foo FROM bar");

    // or alternatively, if you don't know ahead of time that
```

```
        // the query will be a SELECT...

        if (stmt.execute("SELECT foo FROM bar")) {
            rs = stmt.getResultSet();
        }

        // Now do something with the ResultSet ....
    } finally {
        // it is a good idea to release
        // resources in a finally{} block
        // in reverse-order of their creation
        // if they are no-longer needed

        if (rs != null) {
            try {
                rs.close();
            } catch (SQLException sqlEx) { // ignore }

            rs = null;
        }

        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException sqlEx) { // ignore }

            stmt = null;
        }
    }
```

26.3.1.3. 使用CallableStatements以执行存储程序

从MySQL服务器5.0版开始，与Connector/J 3.1.1或更新版本一起使用时，可完全实现java.sql.CallableStatement接口，但getParameterMetaData()方法例外。

在MySQL参考手册的"http://www.mysql.com/doc/en/Stored_Procedures.html"一节中，介绍了MySQL存储程序的语法。

通过JDBC的CallableStatement接口，Connector/J指明了存储程序的功能。

在下面的示例中，给出了1个存储程序，它返回增量为1的inOutParam的值，并通过inputParam传递了作为ResultSet的字符串。

示例**26.3.** 存储程序示例

```
CREATE PROCEDURE demoSp(IN inputParam VARCHAR(255), INOUT inOutParam INT)
BEGIN
    DECLARE z INT;
    SET z = inOutParam + 1;
    SET inOutParam = z;

    SELECT inputParam;

    SELECT CONCAT('zyxw', inputParam);
END
```

要想与Connector/J一起使用demoSp，可采取下述步骤：

1. 使用Connection.prepareCall()准备可调用语句。

注意，必须使用JDBC转义语法，而且必须使用包含占位符的圆括号：

示例**26.4.** 使用**Connection.prepareCall()**

导入java.sql.CallableStatement:

```
...

    //
    // Prepare a call to the stored procedure 'demoSp'
    // with two parameters
    //
    // Notice the use of JDBC-escape syntax ({call ...})
    //

    CallableStatement cStmt = conn.prepareCall("{call demoSp(?, ?)}");





    cStmt.setString(1, "abcdefg");
```

注释:

Connection.prepareCall()是一种开销很大的方法，原因在于驱动程序执行的支持输出参数的元数据检索。出于性能方面的原因，应在你的代码中再次使用CallableStatement实例，通过该方式，使对Connection.prepareCall()的不必要调用降至最低。

2. 注册输出参数（如果有的话）

为了检索输出参数的值（创建存储程序时指定为OUT或INOUT的参数），JDBC要求在CallableStatement接口中使用各种registerOutputParameter()方法来执行语句之前指定它们:

示例**26.5.** 注册输出参数

导入java.sql.Types:

```
...
    //
    // Connector/J supports both named and indexed
    // output parameters. You can register output
    // parameters using either method, as well
    // as retrieve output parameters using either
    // method, regardless of what method was
    // used to register them.
    //
    // The following examples show how to use
    // the various methods of registering
    // output parameters (you should of course
    // use only one registration per parameter).
```

```
//

//
// Registers the second parameter as output
//

cStmt.registerOutParameter(2);

//
// Registers the second parameter as output, and
// uses the type 'INTEGER' for values returned from
// getObject()
//

cStmt.registerOutParameter(2, Types.INTEGER);

//
// Registers the named parameter 'inOutParam'
//

cStmt.registerOutParameter("inOutParam");

//
// Registers the named parameter 'inOutParam', and
// uses the type 'INTEGER' for values returned from
// getObject()
//

cStmt.registerOutParameter("inOutParam", Types.INTEGER);
```

...

3. 设置输入参数（如果有的话）

输入以及输入／输出参数是作为PreparedStatement对象而设置的。但是，CallableStatement也支持按名称设置参数：

示例**26.6.** 设置**CallableStatement**输入参数

...

```
//
// Set a parameter by index
```

```
    //

    cStmt.setString(1, "abcdefg");

    //
    // Alternatively, set a parameter using
    // the parameter name
    //

    cStmt.setString("inputParameter", "abcdefg");

    //
    // Set the 'in/out' parameter using an index
    //

    cStmt.setInt(2, 1);

    //
    // Alternatively, set the 'in/out' parameter
    // by name
    //

    cStmt.setInt("inOutParam", 1);

...
```

4. 执行CallableStatement，并检索任何结果集或输出参数。

尽管CallableStatement支持调用任何语句执行方法（executeUpdate()，executeQuery()或execute()），最灵活的方法是调用execute()，这是因为，采用该方法，你无需事先知道存储程序是否将返回结果集：

示例**26.7.** 检索结果和输出参数值

```
...

    boolean hadResults = cStmt.execute();

    //
    // Process all returned result sets
    //

    while (hadResults) {
        ResultSet rs = cStmt.getResultSet();
```

```
        // process result set
        ...

        hadResults = cStmt.getMoreResults();
    }

    //
    // Retrieve output parameters
    //
    // Connector/J supports both index-based and
    // name-based retrieval
    //

    int outputValue = cStmt.getInt(1); // index-based

    outputValue = cStmt.getInt("inOutParam"); // name-based

...
```

26.3.1.4. 检索AUTO_INCREMENT列的值

在JDBC API 3.0版之前，没有从支持"自动增量"或ID列的数据库中检索关键值的标准方法。对于针对MySQL的早期JDBC驱动，总是不得不在语句接口上使用特定的MySQL方法，或在向拥有AUTO_INCREMENT关键字的表发出"INSERT后"发出"SELECT LAST_INSERT_ID()"。特殊的MySQL方法调用是不可移植的，而且对于发出"SELECT"以获取AUTO_INCREMENT关键字的值来说，需要对数据库执行另一往返操作，其效率不高。在下面的代码片段中，介绍了检索AUTO_INCREMENT值的三种不同方式。首先，介绍了JDBC-3.0新方法"getGeneratedKeys()"的使用方式，当你需要检索AUTO_INCREMENT关键字并访问JDBC-3.0，它是目前的首选方法。在第2个示例中，介绍了使用标准"SELECT LAST_INSERT_ID()"查询检索相同值的方法。在最后一个示例中，介绍了使用"insertRow()"方法时，用可更新结果集检索AUTO_INCREMENT值的方式。

示例**26.8.** 使用**Statement.getGeneratedKeys()**检索**AUTO_INCREMENT**列的值

```
Statement stmt = null;
ResultSet rs = null;

try {

 //
 // Create a Statement instance that we can use for
 // 'normal' result sets assuming you have a
 // Connection 'conn' to a MySQL database already
 // available

 stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
                             java.sql.ResultSet.CONCUR_UPDATABLE);

 //
 // Issue the DDL queries for the table for this example
 //

 stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
 stmt.executeUpdate(
         "CREATE TABLE autoIncTutorial ("
         + "priKey INT NOT NULL AUTO_INCREMENT, "
         + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

 //
 // Insert one row that will generate an AUTO INCREMENT
```

```
        // key in the 'priKey' field
        //

        stmt.executeUpdate(
                "INSERT INTO autoIncTutorial (dataField) "
                + "values ('Can I Get the Auto Increment Field?')",
                Statement.RETURN_GENERATED_KEYS);

        //
        // Example of using Statement.getGeneratedKeys()
        // to retrieve the value of an auto-increment
        // value
        //

        int autoIncKeyFromApi = -1;

        rs = stmt.getGeneratedKeys();

        if (rs.next()) {
            autoIncKeyFromApi = rs.getInt(1);
        } else {

            // throw an exception from here
        }

        rs.close();

        rs = null;

        System.out.println("Key returned from getGeneratedKeys():"
            + autoIncKeyFromApi);
    } finally {

        if (rs != null) {
            try {
                rs.close();
            } catch (SQLException ex) {
                // ignore
            }
        }

        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException ex) {
                // ignore
            }
        }
    }
```

示例**26.9.** 使用**SELECT LAST_INSERT_ID()**检索**AUTO_INCREMENT**列的值

```
    Statement stmt = null;
    ResultSet rs = null;

    try {

        //
        // Create a Statement instance that we can use for
        // 'normal' result sets.

        stmt = conn.createStatement();

        //
        // Issue the DDL queries for the table for this example
        //

        stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
        stmt.executeUpdate(
                "CREATE TABLE autoIncTutorial ("
                + "priKey INT NOT NULL AUTO_INCREMENT, "
                + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

        //
        // Insert one row that will generate an AUTO INCREMENT
        // key in the 'priKey' field
        //

        stmt.executeUpdate(
```

```
            "INSERT INTO autoIncTutorial (dataField) "
            + "values ('Can I Get the Auto Increment Field?')");

    //
    // Use the MySQL LAST_INSERT_ID()
    // function to do the same thing as getGeneratedKeys()
    //

    int autoIncKeyFromFunc = -1;
    rs = stmt.executeQuery("SELECT LAST_INSERT_ID()");

    if (rs.next()) {
        autoIncKeyFromFunc = rs.getInt(1);
    } else {
        // throw an exception from here
    }

    rs.close();

    System.out.println("Key returned from " + "'SELECT LAST_INSERT_ID()': "
        + autoIncKeyFromFunc);

} finally {

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}
```

示例**26.10.** 在可更新的**ResultSets**中检索**AUTO_INCREMENT**列的值

```
Statement stmt = null;
ResultSet rs = null;

try {

    //
    // Create a Statement instance that we can use for
    // 'normal' result sets as well as an 'updatable'
    // one, assuming you have a Connection 'conn' to
    // a MySQL database already available
    //

    stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
                                java.sql.ResultSet.CONCUR_UPDATABLE);

    //
    // Issue the DDL queries for the table for this example
    //

    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
    stmt.executeUpdate(
            "CREATE TABLE autoIncTutorial ("
            + "priKey INT NOT NULL AUTO_INCREMENT, "
            + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

    //
    // Example of retrieving an AUTO INCREMENT key
    // from an updatable result set
    //

    rs = stmt.executeQuery("SELECT priKey, dataField "
       + "FROM autoIncTutorial");

    rs.moveToInsertRow();
```

```
        rs.updateString("dataField", "AUTO INCREMENT here?");
        rs.insertRow();

        //
        // the driver adds rows at the end
        //

        rs.last();

        //
        // We should now be on the row we just inserted
        //

        int autoIncKeyFromRS = rs.getInt("priKey");

        rs.close();

        rs = null;

        System.out.println("Key returned for inserted row: "
            + autoIncKeyFromRS);

    } finally {

        if (rs != null) {
            try {
                rs.close();
            } catch (SQLException ex) {
                // ignore
            }
        }

        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException ex) {
                // ignore
            }
        }
    }
```

运行上面的示例代码时，将获得下述输出：从getGeneratedKeys():返回的键， 从"SELECT LAST_INSERT_ID():"返回1个键， 从插入的行返回1个键。请注意，有些时候，使用"SELECT LAST_INSERT_ID()"查询十分复杂，原因在于函数值与连接相关。因此，如果在相同连接上存在其他查询，函数值将被覆盖。另一方面，"getGeneratedKeys()"方法是由语句实例确定的，因此，即使在相同连接上存在其他查询也能使用它，但在相同语句实例上存在其他查询时则不能使用。

26.3.2. 安装 Connector/J

26.3.2.1. 所需的软件版本
26.3.2.2. 升级旧版本

请按照下述说明安装Connector/J。

26.3.2.1. 所需的软件版本

26.3.2.1.1. 支持的Java版本
26.3.2.1.2. MySQL服务器版本指南
26.3.2.1.3. 安装驱动程序并配置CLASSPATH

26.3.2.1.1. 支持的Java版本

MySQL Connector/J支持Java-2 JVMs，包括JDK-1.2.x、JDK-1.3.x、JDK-1.4.x和JDK-1.5.x，并需要JDK-1.4.x或更新的版本进行编译（而不是运行）。MySQL Connector/J不支持JDK-1.1.x或JDK-1.0.x。

由于实现了java.sql.Savepoint，Connector/J 3.1.0和更新版本不会运行在早于1.4版的JDK上，除非关闭了类验证器（-Xverify:none），这是因为，类验证器将试图加载用于java.sql.Savepoint的类定义，除非使用了savepoint功能，否则驱动程序不会访问类验证器。

早于1.4.x版的JVM上，不能使用Connector/J 3.1.0或更高版本提供的新缓冲功能，这是因为该功能依赖在JDK-1.4.0中首次提供的java.util.LinkedHashMap。

26.3.2.1.2. MySQL服务器版本指南

MySQL Connector/J支持所有著名的MySQL服务器版本。某些特性（外键，可更新结果集）需要更新的MySQL版本才能工作。

与MySQL服务器4.1版或更高版本建立连接时，最好使用MySQL Connector/J 3.1版，这是因为它全面支持较新版本的服务器提供的特性，包括Unicode字符、视图、存储程序和服务器端预处理语句。

尽管3.0版Connector/J能够与MySQL服务器4.1或更高版本建立连接，但由于实现了Unicode字符和新的鉴定机制，将无法更新Connector/J 3.0以支持当前和未来服务器版本中提供的新特性。

26.3.2.1.3. 安装驱动程序并配置CLASSPATH

MySQL Connector/J是以.zip或.tar.gz形式分发的，其中包含源码、类文件、以及仅为"二进制".jar的类文件（名为mysql-connector-java-[version]-bin.jar），从Connector/J 3.1.8开始，驱动程序的"调试版"位于名为"mysql-connector-java-[version]-bin-g.jar"的文件中。

从Connector/J 3.1.9开始，我们不再单独提供.class文件，仅在与驱动程序一起提供的JAR文件中提供它们。

不应使用驱动程序的"调试版"，除非是在向MySQL AB通报问题或缺陷时需要用到它，这是因为"调试版"不是为生产环境下的运行而设计的，如果使用它，会对性能造成负面影响。二进制代码的调试取决于Aspect/J运行时库，该库位于与Connector/J分发版一起提供的src/lib/aspectjrt.jar文件中。

需要使用恰当的GUI或命令行使用工具来解开分发文件（例如，用于.zip文件的WinZip，以及用于.tar.gz文件的"tar"）。由于在分发版中可能存在长文件名，我们采用了GNU tar档案格式。需要使用GNU tar（或能理解GNU tar档案格式的其他应用程序）来解开分发版的.tar.gz文件。

一旦解包了分发版档案文件，可以将mysql-connector-java-[version]-bin.jar放在你的类路径中，或是在你的CLASSPATH环境变量中添加它的完整路径，或是在启动JVM（Java虚拟机）时用命令行开关"-cp"直接指定它，通过该方式安装驱动。

如果你打算用JDBC DriverManager来使用驱动，可使用"com.mysql.jdbc.Driver"，将其用作实施了"java.sql.Driver"类。

示例**26.11.** 设置**Unix**环境下的**CLASSPATH**

在Unix环境下，下述命令用于"csh"：

```
$ setenv CLASSPATH /path/to/mysql-connector-java-[version]-bin.jar:$CLASSPATH
```

可以将上述命令添加到恰当的、用于登录shell的启动文件中，从而使得所有的Java应用程序均能使用MySQL Connector/J。

如果希望与诸如Tomcat或Jboss等应用服务器一起使用MySQL Connector/J，应仔细阅读供应商提供的文档，以了解如何配置第三方类库的更多信息，这是因为大多数应用服务器均会忽略CLASSPATH环境变量。在"与J2EE和其他Java框架一起使用 Connector/J"一节中，给出了针对一些J2EE应用服务器的配置示例，但是，对于特定的应用服务器，JDBC连接池配置信息的权威信息源是该应用服务器的文档。

如果你准备开发小服务程序和／或JSP，而且你的应用服务器是J2EE兼容的，可以将驱动的.jar文件放到webapp的WEB-INF/lib子目录下，在J2EE Web应用程序中，这是第三方类库的标准位置。

如果你的J2EE应用服务器支持或要求，也可以使用com.mysql.jdbc.jdbc2.optional可选软件包中的MysqlDataSource或MysqlConnectionPoolDataSource类。多种MysqlDataSource类均支持下述参数（通过标准的"Set"存取器）：

- user
- password
- serverName（参见前面关于故障切换主机的章节）
- databaseName

- port

26.3.2.2. 升级旧版本

26.3.2.2.1. 从MySQL Connector/J 3.0升级到3.1
26.3.2.2.2. 升级到MySQL服务器4.1版或更新版本时的JDBC事宜

MySQL AB试图使升级进程尽可能简单，但是，对于任何软件来说，某些时侯需要在新版本中进行一些更改才能支持新的特性，改进已有的功能，或与新标准相符。

在本节中，介绍了打算从Connector/J的一个版本升级到另一版本（或考虑到JDBC的功能，升级到MySQL服务器的新版本）的用户应了解的信息。

26.3.2.2.1. 从MySQL Connector/J 3.0升级到3.1

设计Connector/J 3.1时，尽量使它能向后兼容Connector/J 3.0。大的变化被单独作为MySQL-4.1和更新版中的新功能，包括Unicode字符集、服务器端预处理语句、由服务器返回的错误信息中的SQLState代码、以及各种性能增强特性（可通过配置属性启用或禁止）。

- Unicode字符集：关于MySQL新特性的更多信息，请参见下一节，以及服务器手册中的"字符集"一节。如果有些事项配置不当，通常会显示错误，同时给出错误消息，如"非法校对组合"。

- 服务器端预处理语句：Connector/J 3.1将自动检测服务器端预处理语句，并在可用时自动使用它们（MySQL服务器4.1.0版或更新）。

从3.1.7版开始，驱动程序能通过各种Connection.prepareStatement()变体扫描SQL，以判断它是否是能够在服务器端支持的语句类型，如果不被服务器端支持，会将其作为客户端的模拟预处理语句进行处理。也可以通过在JDBC URL中传递"emulateUnsupportedPstmts=false"禁止该特性。

如果应用程序遇到与服务器端预处理语句有关的问题，可将其回复为旧的客户端模拟预处理语句代码，在早于4.1.0版的MySQL服务器中仍使用该代码，连接属性如下：

useServerPrepStmts=false

- 具有全0组分的Datetimes（0000-00-00 ...）：在Java中，无法可靠地表示这些值。从结果集读取它们时，Connector/J 3.0.x总是会将其转换为NULL。

默认情况下，遇到这类值时，Connector/J 3.1将抛出异常，这是因为，根据JDBC和SQL标准，这是最正确的行为方式。可以使用"zeroDateTimeBehavior"配置属性改变该行为。允许的值包括："exception"，（默认值），用代码为"S1009"的SQLState抛出SQLException；"convertToNull"，返回NULL而不是数据；以及"round"，对日期进行舍入处理，使之成为最接近的值，即"0001-01-01"。

从Connector/J 3.1.7开始，能够使用"noDatetimeStringSync=true"（默认值为"假"），将ResultSet.getString()与该行为分离开，从而能够以字符串的形式提取未被改变的全0值。请注意，这也会阻止使用任何时区转换功能，因此，驱动程序将禁止同时启用noDatetimeStringSync和useTimezone。

- 新SQLState代码：Connector/J 3.1采用MySQL返回的SQL:1999 SQLState代码（如果支持的话），它不同于Connector/J 3.0使用的"传统"X/Open状态码。如果连接到了版本低于MySQL-4.1.0（能够将SQLStates作为错误代码组成部分返回的最早版本）的MySQL服务器，驱动程序将使用内置的映射功能。你也可以使用下述配置选项，采用旧的映射。

useSqlStateCodes=false

- 在BLOB列上调用ResultSet.getString()将返回代表它的字节[]数组的地址，而不是BLOB的字符串形式。BLOB没有字符集，因此，在不造成数据丢失或损坏的情况下，不能将它们转换为java.lang.Strings。

要想以BLOB方式将字符串保存在MySQL中，可使用一种TEXT类型，驱动程序会将其当作java.sql.Clob对待。

- 从Connector/J 3.1.8开始，驱动的"调试版"（在名为"mysql-connector-java-[version]-bin-g.jar"的文件中与正常的"二进制"jar文件"名为mysql-connector-java-[version]-bin.jar"一起提供。

从Connector/J 3.1.9开始，我们不再单独提供.class文件，仅在与驱动程序一起提供的JAR文件中提供它们。

不应使用驱动程序的"调试版"，除非是在向MySQL AB通报问题或缺陷时需要用到它，这是因为"调试版"不是为生产环境下的运行而设计的，如果使用它，会对性能造成负面影响。二进制代码的调试取决于Aspect/J运行时库，该库位于与Connector/J分发版一起提供的src/lib/aspectjrt.jar文件中。

26.3.2.2.2. 升级到MySQL服务器4.1版或更新版本时的JDBC事宜

- 使用*UTF-8*字符编码：在4.1版MySQL服务器之前，服务器不支持*UTF-8*字符编码，但JDBC驱动能使用它，从而允许在服务器上的latin1中保存多个字符集。

从MySQL-4.1版开始，该功能被放弃。如果你有依赖该功能的应用程序，而且无法升级它们以使用MySQL服务器4.1版或更高版本中支持的正是Unicode字符集，应在连接URL中增加下述属性：

useOldUTF8Behavior=true

- 服务器端预处理语句：Connector/J 3.1将自动检测服务器端预处理语句，并在可用时自动使用它们（MySQL服务器4.1.0版或更新）。如果应用程序遇到与服务器端预处理语句有关的问题，可将其回复为旧的客户端模拟预处理语句代码，在早于4.1.0版的MySQL服务器中仍使用该代码，连接属性如下：

useServerPrepStmts=false

26.3.3. JDBC引用

26.3.3.1. Driver/Datasource类名，URL语法，以及Connector/J的配置属性
26.3.3.2. JDBC API实施说明
26.3.3.3. Java，JDBC和MySQL类型
26.3.3.4. 使用字符集和Unicode
26.3.3.5. 使用SSL进行安全连接
26.3.3.6. 使用主／从复制和ReplicationConnection

26.3.3.1. Driver/Datasource类名，URL语法，以及Connector/J的配置属性

在MySQL Connector/J中实现了java.sql.Driver的类名是"com.mysql.jdbc.Driver"。"org.gjt.mm.mysql.Driver"类名任是可用的，以保持与MM.MySQL的向后兼容性。注册驱动程序，或配置软件以使用MySQL Connector/J时，应使用该类名。

用于MySQL Connector/J的JDBC URL格式如下，方括号"[,]"的项为可选项：

```
jdbc:mysql://[host][,failoverhost...][:port]/[database][?propertyName1][=propertyValue1][&propertyName2][=propertyValue2]...
```

如果未指定主机名，默认为"127.0.0.1"。如果未指定端口，默认为"3306"，它是MySQL服务器的默认端口号。

```
jdbc:mysql://[host:port],[host:port].../[database][?propertyName1][=propertyValue1][&propertyName2][=propertyValue2]...
```

如果未指定数据库，将使用无"当前"数据库进行连接。在这种情况下，需要在连接实例上调用"setCatalog()"方法，或在SQL中使用数据库名指定完整的表名（即"SELECT dbname.tablename.colname FROM dbname.tablename...")。不指定连接时使用的数据库，该选项通常仅在创建用于处理多个数据库的工具时才有用，例如GUI数据库管理器。

MySQL Connector/J支持故障切换功能。这样，就允许驱动程序切换至"从"主机上，并仍能执行只读查询。仅当连接处于autoCommit(true)状态时，才会出现故障切换，这是因为当事务正在进行时，无法可靠地保证故障切换。在事务/连接结束后，大多数应用服务器和连接池均会将autoCommit设置为"真"。

故障切换功能具有下述行为方式：

如果URL属性"autoReconnect"为"假"：故障切换仅会在连接初始化过程中出现，当驱动程序判断第1台主机再次可用时，将返回。

如果URL属性"autoReconnect"为"真"：当驱动程序判断连接失败时（在任意查询之前），将出现故障切换，而且当驱动程序判断第1台主机再次可用时（发出queriesBeforeRetryMaster查询之后），将返回第1台主机。

在任何一种情况下，当你连接到经过故障切换的服务器时，会将连接设置为只读状态，因此，对于会更改数据的查询来说，将抛出异常（MySQL服务器不会处理该查询）。

配置属性定义了Connector/J与MySQL服务器进行连接的方式。除非作了其他说明，否则可以为DataSource对象或Connection对象设置属性。

可采用下述方式的一种设置Configuration（配置）属性：

- 在java.sql.DataSource的MySQL实施实例上使用set*()方法（它是使用java.sql.DataSource实施实例时的首选方法）：
  - com.mysql.jdbc.jdbc2.optional.MysqlDataSource
  - com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource
- 作为传递给DriverManager.getConnection()或Driver.connect()的java.util.Properties实例中的 键/值对。
- 作为URL中的JDBC URL参数，以传递给java.sql.DriverManager.getConnection()、java.sql.Driver.connect()、或javax.sql.DataSource的setURL()方法的MySQL实施实例。

注释：

如果你用来配置JDBC URL的方法是基于XML的，需要使用XML字符“&”来隔开配置参数，“&”是XML的保留字符。

在下面的表各中，列出了这些属性：

表**26.1.** 连接属性

属性名	定义	要求?	默认值	版本
Connection/Authentication（连接／鉴定）				
user	连接的用户	No		全部
password	连接时使用的密码。	No		全部
socketFactory	驱动程序用于创建与服务器套接字连接的类的名称。该类必须实现了接口“com.mysql.jdbc.SocketFactory”，并有公共无参量构造函数。	No	com.mysql.jdbc.StandardSocketFactory	3.0.3
connectTimeout	套接字连接的超时（单位为毫秒），0表示无超时。仅对JDK-1.4或更新版本有效。默认值为“0”。	No	0	3.0.1
socketTimeout	网络套接字连接的超时（默认值0表示无超时）。	No	0	3.0.1
useConfigs	在解析URL属性或应用用户指定的属性之前，加载由逗号“,”分隔的配置属性列表。在文档的“配置”部分中解释了这些配置。	No		3.1.5
interactiveClient	设置CLIENT_INTERACTIVE标志，根据INTERACTIVE_TIMEOUT而不是WAIT_TIMEOUT向MySQL通报超时连接。	No	false	3.1.0
propertiesTransform	com.mysql.jdbc.ConnectionPropertiesTransform的1个实施实例，在尝试连接之前，驱动程序将使用它来更改传递给驱动的URL属性。	No		3.1.4
useCompression	与服务器进行通信时采用zlib压缩（真／假）？ 默认值为“假”。	No	false	3.0.17
High Availability and Clustering（高可用性和簇集）				
autoReconnect	驱动程序是否应尝试再次建立失效的和／或死连接？ 如果允许，对于在失效或死连接上发出的查询（属于当前事务），驱动程序将抛出异常，但在新事务的连接上发出下一个查询时，将尝试再连接。不推荐使用该特性，这是因为，当应用程序不能恰当处理SQLExceptions时，它会造成与会话状态和数据一致性有关的副作用，设计它的目的仅用于下述情况，即，当你无法配置应用程序来恰当处理因死连接和／或无效连接导致的SQLExceptions时。作为可选方式，可将MySQL服务器变量“wait_timeout”设置为较高的值，而不是默认的8小时。	No	false	1.1
autoReconnectForPools	使用适合于连接池的再连接策略（默认值为“假”）。	No	false	3.1.3
failOverReadOnly	在autoReconnect模式下出现故障切换时，是否应将连接设置为“只读”?	No	true	3.0.12
reconnectAtTxEnd	如果将autoReconnect设置为“真”，在每次事务结束后驱动程序是否应尝试再连接?	No	false	3.0.10
roundRobinLoadBalance	启用了autoReconnect而且failoverReadonly为“假”时，是否应按照循环方式挑选要连接的主机?	No	false	3.1.2
queriesBeforeRetryMaster	出现故障切换（使用多主机故障切换）并返回主机之前发出的查询数。无论首先满足了哪个条件，“queriesBeforeRetryMaster”或“secondsBeforeRetryMaster”，均会再次与主机进行连接。默认值为“50”。	No	50	3.0.2

secondsBeforeRetryMaster	出现故障切换后，在尝试再次连接到主服务器之前，驱动程序应等待的时间？ 无论首先满足了哪个条件，"queriesBeforeRetryMaster"或"secondsBeforeRetryMaster"，均会再次与主机进行连接。单位为秒，默认值为30。	No	30	3.0.2
enableDeprecatedAutoreconnect	自3.2版开始，自动再连接功能受到冷落，在3.3版中将删除该功能。将该属性设置为"真"可禁止检查配置的特性。	No	false	3.2.1
Security（安全）				
allowMultiQueries	在一条语句中，允许使用";"来分隔多条查询（真／假，默认值为"假"）。	No	false	3.1.1
useSSL	与服务器进行通信时使用SSL（真／假），默认值为"假"。	No	false	3.0.2
requireSSL	要求SSL连接，useSSL=true？ 默认值为"假"。	No	false	3.1.0
allowUrlInLocalInfile	驱动程序在是"LOAD DATA LOCAL INFILE"语句中否允许URL？	No	false	3.1.4
paranoid	采取措施，防止在错误信息中泄漏敏感信息，并可可能时清除保存敏感数据的数据结构？ 默认值为"假"。	No	false	3.0.1
Performance Extensions（性能扩展）				
metadataCacheSize	如果将cacheResultSetMetaData设置为"真"，对cacheResultSetMetadata的查询次数（默认值为50）。	No	50	3.1.1
prepStmtCacheSize	如果允许预处理语句缓冲功能，应缓冲处理多少条预处理语句？	No	25	3.0.10
prepStmtCacheSqlLimit	如果允许预处理语句缓冲功能，驱动程序将执行解析缓冲处理的最大SQL是什么？	No	256	3.0.10
maintainTimeStats	驱动程序是否应维持各种内部定时器，以允许空闲时间计算，以及与服务器的连接失败时允许提供更详细的错误消息？ 将该属性设置为"假"，对于每次查询，至少能减少两次对System.getCurrentTimeMillis()的调用。	No	true	3.1.9
blobSendChunkSize	组块，当通过ServerPreparedStatements发送BLOB/CLOB时使用。	No	1048576	3.1.9
cacheCallableStmts	驱动程序是否应对CallableStatements的解析过程执行缓冲处理。	No	false	3.1.2
cachePrepStmts	驱动程序是否应对客户端预处理语句的PreparedStatements的解析过程执行缓冲处理，是否应检查服务器端预处理语句的适用性以及服务器端预处理语句本身？	No	false	3.0.10
cacheResultSetMetadata	驱动程序是否应对用于Statements和PreparedStatements的ResultSetMetaData执行缓冲处理？ 要求 JDK-1.4+，真／假，默认为"假"。	No	false	3.1.1
cacheServerConfiguration	驱动程序是否应根据每条URL对"HOW VARIABLES"和"SHOW COLLATION"的结果执行缓冲处理？	No	false	3.1.5
dontTrackOpenResources	JDBC规范要求驱动程序自动跟踪和关闭资源，但是，如果你的应用程序不能明确调用作用在语句或结果集上的close()，可能会导致内存泄漏。将该属性设置为"真"，可放宽该限制，对于某些应用程序，会提供更高的内存效率。	No	false	3.1.7
dynamicCalendars	需要时，驱动程序是否应检索默认日历，或根据连接／会话对其进行缓冲处理？	No	false	3.1.5
elideSetAutoCommits	如果使用MySQL-4.1或更高版本，当服务器的状态与Connection.setAutoCommit(boolean)请求的状态不匹配时，驱动程序是否仅应发出"set autocommit=n"查询？	No	false	3.1.3
holdResultsOpenOverStatementClose	驱动程序是否应按照JDBC规范的要求关闭Statement.close()上的结果集？	No	false	3.1.7
locatorFetchBufferSize	如果将"emulateLocators"配置为"真"，当获取关于getBinaryInputStream的BLOB数据时，缓冲区的大小应是多少？	No	1048576	3.2.1
useFastIntParsing	是否使用内部"String->Integer"转换子程序来避免创建过多对象？	No	true	3.1.4
useLocalSessionState	驱动程序是否应引用autocommit的内部值，以及由Connection.setAutoCommit()和Connection.setTransactionIsolation()设置的事务隔离，而不是查询数据库？	No	false	3.1.7
useNewIO	驱动程序是否应将java.nio.* interfaces用于网络通信（真／假），默认为"假"。	No	false	3.1.0
useReadAheadInput	从服务器读取数据时，是否使用较新的、优化的非成组缓冲输入流？	No	true	3.1.5
Debuging/Profiling（调试/仿形）				
logger	实现了com.mysql.jdbc.log.Log的类的名称，com.mysql.jdbc.log.Log用于记录消息（默认为"com.mysql.jdbc.log.StandardLogger"，它会将日志记录到STDERR）。	No	com.mysql.jdbc.log.StandardLogger	3.1.1
profileSQL	跟踪查询以及它们对已配制记录器的执行/获取次数（真／假），默认为"假"。	No	false	3.1.0

reportMetricsIntervalMillis	如果允许"gatherPerfMetrics"，记录它们的频率是多少（单位毫秒）？	No	30000	3.1.2
maxQuerySizeToLog	调试或仿形时，控制将记录的查询的最大长度／大小。	No	2048	3.1.3
packetDebugBufferSize	当"enablePacketDebug"为"真"时，需要保留的最大信息包数目。	No	20	3.1.3
slowQueryThresholdMillis	如果允许"logSlowQueries"，在将查询记录为"慢"之前的查询时间是多少（毫秒）？	No	2000	3.1.2
useUsageAdvisor	驱动程序是否应发出"使用情况"警告，就DBC和MySQL Connector/J的恰当和高效使用给出建议（真／假，默认为"假"）？	No	false	3.1.1
autoGenerateTestcaseScript	驱动程序是否应将正在执行的SQL（包括服务器端预处理语句）转储到STDERR？	No	false	3.1.9
dumpQueriesOnException	驱动程序是否应将发送至服务器的查询内容转储到SQLExceptions中？	No	false	3.1.3
enablePacketDebug	允许时，将保留"packetDebugBufferSize"信息包的环形缓冲区，并当在驱动程序代码的关键区域抛出异常时进行转储。	No	false	3.1.3
explainSlowQueries	如果允许了"logSlowQueries"，驱动程序是否应在服务器上自动发出"EXPLAIN"，并以WARN级别将结果发送给配置好的日志？	No	false	3.1.2
logSlowQueries	是否要记录时间长于"slowQueryThresholdMillis"的查询？	No	false	3.1.2
traceProtocol	是否应记录跟踪级网络协议？	No	false	3.1.2
Miscellaneous（其他）				
useUnicode	处理字符串时，驱动程序是否应使用Unicode字符编码？ 仅应在驱动程序无法确定字符集映射，或你正在强制驱动程序使用MySQL不是固有支持的字符集时（如UTF-8）才应使用。真／假，默认为"真"。	No	false	1.1g
characterEncoding	如果"useUnicode"被设置为"真"，处理字符串时，驱动程序应使用什么字符编码？ 默认为"autodetect"。	No		1.1g
characterSetResults	字符集，用于通知服务器以何种字符集返回结果。	No		3.0.13
connectionCollation	如果设置了它，将通知服务器通过"set collation_connection"使用该校对。	No		3.0.13
sessionVariables	以逗号隔开的"名称／值"对列表，当驱动程序建立了连接后，以"SET SESSION ..."的方式将其发送给服务器。	No		3.1.8
allowNanAndInf	驱动程序是否应在PreparedStatement.setDouble()中允许NaN或+/- INF值？	No	false	3.1.5
autoDeserialize	驱动程序是否应自动检测并串并转换保存在BLOB字段中的对象？	No	false	3.1.5
capitalizeTypeNames	是否将DatabaseMetaData中的类型名转换为大写？ 通常仅在使用WebObjects时有用，真／假。默认为"假"。	No	false	2.0.7
clobberStreamingResults	这会使"流式"结果集被自动关闭，如果在所有数据尚未从服务器中读取完之前，执行了另一查询，正在从服务器流出的任何未完成数据均将丢失。	No	false	3.0.9
continueBatchOnError	如果一条语句失败，驱动程序是否应继续处理批命令？ JDBC规范允许任何一种方式（默认为"真"）。	No	true	3.0.3
createDatabaseIfNotExist	如果不存在，创建URL中给定的数据库。假定用户具有创建数据库的权限。	No	false	3.1.9
emptyStringsConvertToZero	驱动程序是否应允许从空字符串字段到数值"0"的转换？	No	true	3.1.8
emulateLocators	N/A	No	false	3.1.0
emulateUnsupportedPstmts	驱动程序是否应检测不被服务器支持的预处理语句，并用客户端模拟版替换它们？	No	true	3.1.7
ignoreNonTxTables	是否忽略关于回退的非事务表？ 默认值为"假"。	No	false	3.0.9
jdbcCompliantTruncation	连接到支持告警的服务器时（MySQL 4.1.0和更高版本），当按照JDBC的要求截短数据时，驱动程序是否应抛出java.sql.DataTruncation异常？	No	true	3.1.2
maxRows	返回的最大行数（0，默认值表示返回所有行）。	No	-1	all versions
noDatetimeStringSync	不保证ResultSet.getDatetimeType().toString().equals(ResultSet.getString()。	No	false	3.1.7
nullCatalogMeansCurrent	当DatabaseMetadataMethods请求"目录"参数时，值"Null"是否意味着使用当前目录？ 它不兼容JDBC，但符合驱动程序早期版本的传统行为。	No	true	3.1.8
nullNamePatternMatchesAll	接受*pattern参数的DatabaseMetaData方法是否应将null按对待"％"的相同方式处理（不兼容JDBC，但驱动程序的早期版本能接受与规范的这类偏离）。	No	true	3.1.8
pedantic	严格遵守JDBC规范。	No	false	3.0.0

relaxAutoCommit	如果驱动程序所连接的MySQL服务器的版本不支持事务，仍允许调用commit()、rollback()和setAutoCommit()？真／假，默认为"假"。	No	false	2.0.13
retainStatementAfterResultSetClose	调用ResultSet.close()后，驱动程序是否应将语句引用保存在结果集中？ 在JDBC-4.0后，与JDBC不兼容。	No	false	3.1.11
rollbackOnPooledClose	当连接池中的逻辑连接关闭时，驱动程序是否应发出rollback()？	No	true	3.0.15
runningCTS13	允许在Sun与JDBC兼容的testsuite 1.3版中处理缺陷。	No	false	3.1.7
serverTimezone	覆盖时区的检测/映射。当服务器的时区为映射到Java时区时使用。	No		3.0.2
strictFloatingPoint	仅在兼容性测试的早期版本中使用。	No	false	3.0.0
strictUpdates	驱动程序是否应对可更新结果集进行严格检查（选择所有的主键）？真／假，默认为"真"。	No	true	3.0.4
tinyInt1isBit	驱动程序是否应将数据类型TINYINT(1)当作BIT类型对待？创建表时，服务器会执行BIT -> TINYINT(1)操作。	No	true	3.0.16
transformedBitIsBoolean	如果驱动程序将TINYINT(1)转换为不同的类型，为了与MySQL-5.0兼容，驱动程序是否应使用BOOLEAN取代BIT？这是因为MySQL-5.0具有BIT类型。	No	false	3.1.9
ultraDevHack	由于UltraDev已损坏，并为所有语句发出了prepareCall()，需要时，是否要为prepareCall()创建PreparedStatements? 真／假，默认值为"假"。	No	false	2.0.3
useHostsInPrivileges	在DatabaseMetaData.getColumn/TablePrivileges()中为用户添加"@hostname"。真／假，默认为"真"。	No	true	3.0.2
useOldUTF8Behavior	与4.0和更早版本的服务器进行通信时，使用UTF-8。	No	false	3.1.6
useOnlyServerErrorMessages	对服务器返回的错误消息，不事先设定"标准的"SQLState错误消息。	No	true	3.0.15
useServerPrepStmts	如果服务器支持，是否使用服务器端预处理语句？ 默认值为"真"。	No	true	3.1.0
useSqlStateCodes	使用SQL标准状态码取代"传统的"X/Open/SQL状态码，真／假，默认为"真"。	No	true	3.1.3
useStreamLengthsInPrepStmts	是否采用PreparedStatement/ResultSet.setXXXStream()方法调用中的流长度参数？真／假，默认为"真"。	No	true	3.0.2
useTimezone	是否在客户端和服务器时区间转换时间／日期类型（真／假，默认为"假"）？	No	false	3.0.2
useUnbufferedInput	不使用BufferedInputStream来从服务器读取数据。	No	true	3.0.11
yearIsDateType	JDBC驱动程序是否应将MySQL类型"YEAR"当作java.sql.Date或SHORT对待？	No	true	3.1.9
zeroDateTimeBehavior	当驱动程序遇到全由0组成的DATETIME值时，应出现什么？MySQL使用它来表示无效日期。有效值是"exception"、"round"和"convertToNull"。	No	exception	3.1.4

通过"socketFactory"属性，使用NamedPipeSocketFactory，在Windows NT/2000/XP平台上，通过命名管道，Connector/J也支持对MySQL的访问。如果不使用namedPipePath属性，将使用\\.\pipe\MySQL的默认值。如果使用NamedPipeSocketFactory，将忽略JDBC url中的主机名和端口号。

在URL中添加下述属性可启用NamedPipeSocketFactory：

socketFactory=com.mysql.jdbc.NamedPipeSocketFactory

命名管道仅能当连接到位于相同物理机器上的MySQL时才能正常工作，该机器上应使用了JDBC驱动程序。在简单的性能测试中，命名管道的访问速度比标准的TCP/IP访问块30～50％。

使用com.mysql.jdbc.NamedPipeSocketFactory或com.mysql.jdbc.StandardSocketFactory中的示例代码，可创建你自己的套接字代理。

26.3.3.2. JDBC API实施说明

MySQL Connector/J通过了Sun JDBC兼容测试套件公共版中的所有测试。但是，在很多场合下，对于应如何实施特定的功能，JDBC规范并未给出明确的规定，或者说，该规范允许有一定的实施范围。

在本节中，就特定实施方案将如何影响MySQL Connector/J的使用方式，给出了接口层面上的详细介绍。

· Blob

Blob实施不允许"原地"调整（它们是"副本"，正如DatabaseMetaData.locatorsUpdateCopies()方法所指明的那样）。因此，应使用对应的PreparedStatement.setBlob()或ResultSet.updateBlob()（对于可更新结果集）方法，将变化保存到数据库中。

自Connector/J version 3.1.0开始，通过在JDBC URL中添加属性"emulateLocators=true"，能够使用定位器模拟Blob。随后，必须使用带有列值的列别名，在你编写的用于检索Blob的SELECT中，将列值设为Blob列的世纪名称。SELECT还必须仅引用1个表，该表必须有1个主键，而且SELECT必须涵盖构成主键的所有列。随后，驱动程序将延期加载实际的Blob数据，直至检索了Blob并在其上调用了检索方法为止（getInputStream(), getBytes(),等）。

· CallableStatement

自Connector/J 3.1.1开始，当通过CallableStatement接口连接到MySQL 5.0或更高版本时，可支持存储程序。目前，不支持CallableStatement的getParameterMetaData()方法。

· Clob

Clob实施不允许"原地"调整（它们是"副本"，正如DatabaseMetaData.locatorsUpdateCopies()方法所指明的那样）。因此，应使用PreparedStatement.setClob()方法将变更保存到数据库中。JDBC API没有ResultSet.updateClob()方法。

· Connection

与MM.MySQL的早期版本不同，"isClosed()"不会对服务器即行Ping操作以确定服务器是否有效。按照JDBC规范，如果在连接上调用了"closed()"，它仅返回"真"。如果需要确定连接是否依然有效，应发出简单查询，如"SELECT 1"。如果连接不再有效，驱动程序将抛出异常。

· DatabaseMetaData

对于外键信息（getImported/ExportedKeys()和getCrossReference()），仅在"InnoDB"类性的表中可用。但是，驱动程序会使用"SHOW CREATE TABLE"来检索该信息，因此，当其他表类型支持外键时，驱动程序也能支持它们。

· Driver

· PreparedStatement

PreparedStatements是由驱动程序实现的，这是应为MySQL未提供预处理语句功能。出于该原因，驱动程序不实施getParameterMetaData()或getMetaData()，这是因为，它要求驱动程序在客户端上具有完整的SQL语法分析程序。

从3.1.0版MySQL Connector/J开始，当服务器支持时，将使用服务器端预处理语句和"二进制编码"的结果集。

使用带有"large"参数（这类参数是通过setBinaryStream()、setAsciiStream()、setUnicodeStream()、setBlob()或setClob()设置的）的服务器端预处理语句时应谨慎。如果打算再次执行已将任何"large"参数更改为非"large"参数的语句，需要调用clearParameters()，并再次设置所有参数。其原因如下：

o 设置了参数时，驱动程序会将"large"数据"out-of-band"发送给服务器端的预处理语句（执行预处理语句之前）。

o 一旦完成，将关闭用于读取客户端上数据的流（根据JDBC规范），而且不能再次读取流。

o 如果参数从"large"变为非"large"，驱动程序必须复位预处理语句的服务器端状态，以便允许已更改的参数区带以前的"large"值。这将删除已发送给服务器的所有"large"数据，因而需要通过setBinaryStream()、setAsciiStream()、setUnicodeStream()、setBlob()或setClob()方法再次发送数据。

因而，如果你打算将参数类型更改为非"large"类型，必须调用clearParameters()，并在重新执行预处理语句之前再次设置预处理语句的所有参数。

· ResultSet

在默认情况下，ResultSets（结果集）是可完全检索的，并被保存在内存中。对于大多数情况，这是最有效的操作方式，而且还应归因于更容易实施的MySQL网络协议设计。如果你正在处理具有大量行或大数据的ResultSets，而且无法在JVM内为所需内存分配大量空间，可以通知驱动程序以"流"方式返回结果，一次一行。

要想允许该功能，需要以下述方式创建1个语句实例：

```
stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
              java.sql.ResultSet.CONCUR_READ_ONLY);
stmt.setFetchSize(Integer.MIN_VALUE);
```

正向、只读结果集，以及Integer.MIN_VALUE的组合用于指示驱动程序以"流"方式按行处理结果集。此后，对于该语句创建的结果集，将按行检索。

对于该方式，有一些需注意的事项。能够在连接上发出任何其他查询之前，应读取结果集中的所有行（或关闭结果集），否则将抛出异常。

能够释放这些锁定语句（无论它们是MyISAM表级锁定，还是某些其他存储引擎如InnoDB中的行级锁定）的最早时刻是完成语句时。

如果语句在事务的范围内，当事务完成后将释放锁定（它意味着语句需首先完成）。与大多数其他数据库一样，在读取了语句上所有的未决结果集或关闭了语句的活动结果集之前，语句不会结束。

因此，如果正在使用"流式"结果，如果希望保持对特定表的同时访问，而这些表被生成结果集的语句所引用，就应尽快地处理"流式"结果。

- ResultSetMetaData

仅当使用MySQL服务器4.0或更高版本时，"isAutoIncrement()"方法才能正确工作。

- Statement

使用版本低于3.2.1的JDBC驱动程序，而且所连接的服务器版本低于5.0.3时，除了像前面介绍的那样切换结果集外，"setFetchSize()"方法不起作用。

MySQL不支持SQL光标，而且JDBC驱动程序也不能模拟它们，因此"setCursorName()"没有效果。

26.3.3.3. Java，JDBC和MySQL类型

MySQL Connector/J在处理MySQL数据类型和Java数据类型的转换处理方面十分灵活。

尽管可能会出现舍入、溢出或精度损失，当在通常情况下，能够将任何MySQL数据类型转换为java.lang.String，并能将任何数值类型转换为Java数值类型。

从Connector/J 3.1.0开始，按照JDBC规范的要求，JDBC驱动程序将发出警告或抛出DataTruncation异常，除非通过使用"jdbcCompliantTruncation"属性并将其设为"假"，对连接进行相应配置取消了前述要求。

在下面的表格中，列出能可靠工作的转换：

表**26.2.** 转换表

下述**MySQL**数据类型	总能转换为下述**Java**类型
CHAR, VARCHAR, BLOB, TEXT, ENUM, and SET	java.lang.String, java.io.InputStream, java.io.Reader, java.sql.Blob, java.sql.Clob
FLOAT, REAL, DOUBLE PRECISION, NUMERIC, DECIMAL, TINYINT, SMALLINT, MEDIUMINT, INTEGER, BIGINT	java.lang.String, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Double, java.math.BigDecimal 注释： 与希望转换的MySQL数据类型相比，如果选择了精度较低的Java数值类型，可能会出现舍入、溢出或精度损失。
DATE, TIME, DATETIME, TIMESTAMP	java.lang.String, java.sql.Date, java.sql.Timestamp

在MySQL类型和Java类型之间，ResultSet.getObject()方法采用了下述类型转换方式，在可能的情况下遵从JDBC规范：

表**26.3.** 用于**ResultSet.getObject()**的**MySQL**类型和**Java**类型

MySQL类型名称	以**Java**类返回
BIT(1) (new in MySQL-5.0)	java.lang.Boolean
BIT(> 1) (new in MySQL-5.0)	byte[]
TINYINT	java.lang.Boolean，如果将配置属性"tinyInt1isBit"设为"真"（默认值），并将存储大小设为"1"；或java.lang.Integer，如果不是的话。
BOOL , BOOLEAN	请参见上面的TINYINT，它们目前是TINYINT(1)的别名。
SMALLINT[(M)] [UNSIGNED]	java.lang.Integer（无论是否为UNSIGNED）
MEDIUMINT[(M)] [UNSIGNED]	java.lang.Integer（无论是否为UNSIGNED）
INT,INTEGER[(M)] [UNSIGNED]	java.lang.Integer，如果是UNSIGNED，java.lang.Long
BIGINT[(M)] [UNSIGNED]	java.lang.Long，如果是UNSIGNED，java.math.BigInteger
FLOAT[(M,D)]	java.lang.Float
DOUBLE[(M,B)]	java.lang.Double
DECIMAL[(M[,D])]	java.math.BigDecimal
DATE	java.sql.Date
DATETIME	java.sql.Timestamp
TIMESTAMP[(M)]	java.sql.Timestamp
TIME	java.sql.Time
YEAR[(2|4)]	java.sql.Date（日期设为2月1日晚上2点）
CHAR(M)	java.lang.String（除非列的字符集是BINARY），然后返回字节[]
VARCHAR(M) [BINARY]	java.lang.String（除非列的字符集是BINARY），然后返回字节[]
BINARY(M)	byte[]
VARBINARY(M)	byte[]
TINYBLOB	byte[]
TINYTEXT	java.lang.String
BLOB	byte[]
TEXT	java.lang.String
MEDIUMBLOB	byte[]
MEDIUMTEXT	java.lang.String
LONGBLOB	byte[]
LONGTEXT	java.lang.String
ENUM('value1','value2',...)	java.lang.String
SET('value1','value2',...)	java.lang.String

26.3.3.4. 使用字符集和Unicode

对于从JDBC驱动程序发往服务器的所有字符串，均将自动地从固有放热Java Unicode形式转换为客户端字符编码，包括通过Statement.execute()、Statement.executeUpdate()和Statement.executeQuery()发出的所有查询，以及除了用setBytes()、setBinaryStream()、setAsiiStream()、setUnicodeStream()和setBlob()排除的参试之外的所有PreparedStatement和CallableStatement参数。

在MySQL服务器4.1之前，Connector/J支持每连接单一字符编码，能够从服务器配置自动检测到它，也能由用户通过使用*useUnicode*和*characterEncoding*属性配置它。

从MySQL服务器4.1版起，Connector/J支持客户端和服务器之间的但以字符编码，以及针对结果集中从服务器返回至客户端的数据的任意数目字符编码。

连接时将自动检测客户端和服务器之间的字符编码。对于由驱动程序使用的编码来说，它是在服务器上通过使用配置变量"*character_set*"（低于4.1.0的服务器版本）和"*character_set_server*"（4.1.0和更高的服务器版本）指定的。更多信息，请参见MySQL服务器手册中的http://www.mysql.com/doc/en/Charset-server.html一节。

要想覆盖客户端上的自动检测编码功能，可在用于连接到服务器的URL中使用"*characterEncoding*"属性。

在客户端上指定字符编码时，应使用Java风格名称。在下面的表格中，列出了用于MySQL字符集的Java风格名称：

表**26.4. MySQL**对**Java**编码名称的翻译

MySQL字符集名称	**Java**风格字符编码名称
usa7	US-ASCII
big5	Big5
gbk	GBK
sjis	SJIS
gb2312	EUC_CN
ujis	EUC_JP
euc_kr	EUC_KR
latin1	ISO8859_1
latin1_de	ISO8859_1
german1	ISO8859_1
danish	ISO8859_1
latin2	ISO8859_2
czech	ISO8859_2
hungarian	ISO8859_2
croat	ISO8859_2
greek	ISO8859_7
hebrew	ISO8859_8
latin5	ISO8859_9
latvian	ISO8859_13
latvian1	ISO8859_13
estonia	ISO8859_13
dos	Cp437
pclatin2	Cp852
cp866	Cp866
koi8_ru	KOI8_R
tis620	TIS620
win1250	Cp1250
win1250ch	Cp1250
win1251	Cp1251
cp1251	Cp1251
win1251ukr	Cp1251
cp1257	Cp1257
macroman	MacRoman
macce	MacCentralEurope
utf8	UTF-8
ucs2	UnicodeBig

警告

不要用Connector/J发出查询“set names”，这是因为驱动程序不会检测已变化的字符集，而是会继续使用在初始连接设置中检测到的字符集。

为了允许从客户端发出的多个字符集，应使用"UTF-8"编码，方式是，将utf8配置为默认的服务器字符集，或通过"*characterEncoding*"属性配置JDBC驱动程序以使用"UTF-8"。

26.3.3.5. 使用SSL进行安全连接

MySQL Connector/J中的SSL能够对JDBC驱动程序和服务器之间传输的所有数据进行加密（除了初始握手数据）。启用SSL会导致性能损失，体现在查询时间将增加35～50％，具体情况取决于查询的大小以及返回的数据量。

要想使SSL支持能够工作，必须满足下述要求：

- 包含JSSE（Java安全套接字扩展）的JDK，如JDK-1.4.1或更高版本。SSL目前不能与能够为其添加JSSE的JDK一起工作，如JDK-1.2.x或JDK-1.3.x，原因在于下述JSSE缺陷：http://developer.java.sun.com/developer/bugParade/bugs/4273544.html
- 支持SSL并已编译和配置了该功能的MySQL服务器，如MySQL-4.0.4和更高版本，请参见：http://www.mysql.com/doc/en/Secure_connections.html
- 客户端证书（在本节稍后介绍）。

首先，需要将MySQL服务器CA证书导入到Java truststore。在MySQL源码分发版的"SSL"子目录下给出了1个示例MySQL服务器CA证书。SSL将使用它来确定是否与安全MySQL服务器进行通信。

要想使用Java的"keytool"在当前目录下创建truststore，并导入服务器的CA证书（"cacert.pem"），可采取下述方式（假定"keytool"位于路径中。它位于JDK或JRE的"bin"子目录下）：

```
shell> keytool -import -alias mysqlServerCACert -file cacert.pem -keystore truststore
```

Keytool将给出下述响应信息：

```
Enter keystore password:  *********
Owner: EMAILADDRESS=walrus@example.com, CN=Walrus, O=MySQL AB, L=Orenburg, ST=Some
-State, C=RU
Issuer: EMAILADDRESS=walrus@example.com, CN=Walrus, O=MySQL AB, L=Orenburg, ST=Som
e-State, C=RU
Serial number: 0
Valid from: Fri Aug 02 16:55:53 CDT 2002 until: Sat Aug 02 16:55:53 CDT 2003
Certificate fingerprints:
         MD5:  61:91:A0:F2:03:07:61:7A:81:38:66:DA:19:C4:8D:AB
         SHA1: 25:77:41:05:D5:AD:99:8C:14:8C:CA:68:9C:2F:B8:89:C3:34:4D:6C
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

随后，需要生成客户端证书，以便MySQL服务器知道它正与安全客户端进行通信：

```
 shell> keytool -genkey -keyalg rsa -alias mysqlClientCertificate -keystore keystore
```

Keytool将给出下述提示信息，并在当目录下创建名为"keystore"的密钥存储器。

你应使用与具体情况相适应的新作出响应：

```
Enter keystore password:  *********
What is your first and last name?
```

```
  [Unknown]:  Matthews
What is the name of your organizational unit?
  [Unknown]:  Software Development
What is the name of your organization?
  [Unknown]:  MySQL AB
What is the name of your City or Locality?
  [Unknown]:  Flossmoor
What is the name of your State or Province?
  [Unknown]:  IL
What is the two-letter country code for this unit?
  [Unknown]:  US
Is <CN=Matthews, OU=Software Development, O=MySQL AB,
 L=Flossmoor, ST=IL, C=US> correct?
  [no]:  y

输入<mysqlClientCertificate>的密码
        如果与keystore的密码相同，按回车）：
```

最后，要想使JSSE能够使用你生成的keystore和truststore，启动JVM时，需要设置下述系统属性，用你所创建的keystore文件完整路径替换“path_to_keystore_file”，用你所创建的truststore文件完整路径替换“path_to_truststore_file”，并为每个属性使用恰当的密码值。

```
-Djavax.net.ssl.keyStore=path_to_keystore_file
-Djavax.net.ssl.keyStorePassword=*********
-Djavax.net.ssl.trustStore=path_to_truststore_file
-Djavax.net.ssl.trustStorePassword=*********
```

此外，还需要在用于MySQL Connector/J的连接参数中将“useSSL”设置为“真”，方法是，在URL中添加“useSSL=true”，或在准备传递给DriverManager.getConnection()的java.util.Properties实例中将“useSSL”设置为“真”。

你可以打开JSSE调试功能能够，测试SSL是否工作（详情如下），并查找下述关键事件：

```
...
 *** ClientHello, v3.1
 RandomCookie:  GMT: 1018531834 bytes = { 199, 148, 180, 215, 74, 12, 54, 244, 0, 168, 55, 103, 215, 64, 16, 138, 225, 190, 132, 153, 2, 217, 219, 239,
202, 19, 121, 78 }
 Session ID:  {}
 Cipher Suites:  { 0, 5, 0, 4, 0, 9, 0, 10, 0, 18, 0, 19, 0, 3, 0, 17 }
 Compression Methods:  { 0 }
 ***
 [write] MD5 and SHA1 hashes:  len = 59
 0000: 01 00 00 37 03 01 3D B6   90 FA C7 94 B4 D7 4A 0C  ...7..=.......J.
 0010: 36 F4 00 A8 37 67 D7 40   10 8A E1 BE 84 99 02 D9  6...7g.@........
 0020: DB EF CA 13 79 4E 00 00   10 00 05 00 04 00 09 00  ....yN..........
 0030: 0A 00 12 00 13 00 03 00   11 01 00                 ...........
```

```
main, WRITE:  SSL v3.1 Handshake, length = 59

main, READ:  SSL v3.1 Handshake, length = 74

*** ServerHello, v3.1

RandomCookie:  GMT: 1018577560 bytes = { 116, 50, 4, 103, 25, 100, 58, 202, 79, 185, 178, 100, 215, 66, 254, 21, 83, 187, 190, 42, 170, 3, 132, 110, 82,
148, 160, 92 }

Session ID:  {163, 227, 84, 53, 81, 127, 252, 254, 178, 179, 68, 63, 182, 158, 30, 11, 150, 79, 170, 76, 255, 92, 15, 226, 24, 17, 177, 219, 158, 177,
187, 143}

Cipher Suite:  { 0, 5 }

Compression Method: 0

***

%% Created:  [Session-1, SSL_RSA_WITH_RC4_128_SHA]

** SSL_RSA_WITH_RC4_128_SHA

[read] MD5 and SHA1 hashes:  len = 74

0000: 02 00 00 46 03 01 3D B6   43 98 74 32 04 67 19 64  ...F..=.C.t2.g.d

0010: 3A CA 4F B9 B2 64 D7 42   FE 15 53 BB BE 2A AA 03  :.O..d.B..S..*..

0020: 84 6E 52 94 A0 5C 20 A3   E3 54 35 51 7F FC FE B2  .nR..\ ..T5Q....

0030: B3 44 3F B6 9E 1E 0B 96   4F AA 4C FF 5C 0F E2 18  .D?.....O.L.\...

0040: 11 B1 DB 9E B1 BB 8F 00   05 00                    ..........

main, READ:  SSL v3.1 Handshake, length = 1712

...
```

设置了下述系统属性时，JSSE可提供调试功能（为STDOUT）：-Djavax.net.debug=all。它用于设定要使用的keystores和truststores，以及在SSL握手和证书交换过程中将出现什么。当你尝试进行SSL连接时，如果打算确定不能工作的部分，该设置十分有用。

26.3.3.6. 使用主／从复制和ReplicationConnection

从Connector/J 3.1.7开始，我们提供了1个驱动程序变体，它能自动发出读／写主服务器的查询，或根据Connection.getReadOnly()的状态对从主机进行故障切换或循环负载平衡设置。

应用程序发出信号，通过调用Connection.setReadOnly(true)指明事务为只读的，该具有"复制意识"的连接将使用从连接之一，从连接是采用了循环方案的负载平衡per-vm（给定连接与从连接密切相关，除非在服务中删除了从连接）。如果你有1项写事务，或1项对时间敏感的读事务（记住，在MySQL中，复制是以异步方式进行的），请调用Connection.setReadOnly(false)，将连接设置为非只读的，驱动程序会确保进一步的调用均将被发送到主MySQL服务器。驱动程序负责传播autocommit的当前状态，隔离级别，以及用于完成该负载平衡功能的所有连接之间的目录。

要想启用该该功能，在配置应用服务器的连接池时，或为独立应用程序创建JDBC驱动实例时，请使用"com.mysql.jdbc.ReplicationDriver"类。由于它能接受与标准MySQL JDBC驱动相同的URL格式，ReplicationDriver目前不能与基于java.sql.DriverManager的连接一起使用，除非它是用DriverManager注册的唯一MySQL JDBC驱动程序。

下面给出了一个简短的简单示例，介绍了如何在独立应用程序中使用ReplicationDriver的方法。

```
import java.sql.Connection;

import java.sql.ResultSet;

import java.util.Properties;


import com.mysql.jdbc.ReplicationDriver;


public class ReplicationDriverDemo {
```

```
public static void main(String[] args) throws Exception {
    ReplicationDriver driver = new ReplicationDriver();

    Properties props = new Properties();

    // We want this for failover on the slaves
    props.put("autoReconnect", "true");

    // We want to load balance between the slaves
    props.put("roundRobinLoadBalance", "true");

    props.put("user", "foo");
    props.put("password", "bar");

    //
    // Looks like a normal MySQL JDBC url, with a comma-separated list
    // of hosts, the first being the 'master', the rest being any number
    // of slaves that the driver will load balance against
    //

    Connection conn =
        driver.connect("jdbc:mysql://master,slave1,slave2,slave3/test",
            props);

    //
    // Perform read/write work on the master
    // by setting the read-only flag to "false"
    //

    conn.setReadOnly(false);
    conn.setAutoCommit(false);
    conn.createStatement().executeUpdate("UPDATE some_table ....");
    conn.commit();

    //
    // Now, do a query from a slave, the driver automatically picks one
    // from the list
    //
```

```
        conn.setReadOnly(true);


        ResultSet rs = conn.createStatement().executeQuery("SELECT a,b,c FROM some_other_table");


         .......
    }
}
```

26.3.4. 与J2EE和其他Java框架一起使用 Connector/J

26.3.4.1. 一般J2EE概念
26.3.4.2. 与Tomcat一起使用Connector/J
26.3.4.3. 与JBoss一起使用Connector/J

本节介绍了在数种不同情况下使用Connector/J的方法。

26.3.4.1. 一般J2EE概念

26.3.4.1.1. 理解连接池

在本节中，介绍了与Connector/J使用有关的J2EE概念的基本知识。

26.3.4.1.1. 理解连接池

连接池是创建和管理多个连接的一种技术，这些连接可被需要使用它们的任何线程使用。连接池技术基于下述事实：对于大多数应用程序，当它们正在处理通常需要数毫秒完成的事务时，仅需要能够访问JDBC连接的1个线程。未处理事务时，连接处于闲置状态。使用连接池，允许其他线程使用闲置连接来执行有用的任务。

事实上，当某一线程需要用JDBC在MySQL或其他数据库上执行操作时，需要用到由连接池提供的连接。使用连接完成线程后，线程会将连接返回给连接池，以便该连接能够被其他需要使用连接的线程使用。

从连接池"借出"连接时，该连接仅供请求它的线程使用。从编程观点看，其效果等同于每次需要JDBC连接时调用DriverManager.getConnection()，但是，采用连接池技术，可通过使用新的或已有的连接结束线程。

连接池技术能显著增加Java应用程序的性能，同时还能降低资源使用率。连接池技术的主要优点包括：

- 缩短了连接创建时间

与其他数据库相比，MySQL提供了快速的连接设置功能，连接时间通常不是问题，但创建新的JDBC连接仍会导致联网操作和一定的IDBC驱动开销，如果这类连接是"循环"使用的，使用该方式，可避免这类不利因素。

- 简化的编程模型

使用连接池技术时，每个单独线程能够像创建了自己的JDBC连接那样进行操作，从而允许使用直接的JDBC编程技术。

- 受控的资源使用

如果不使用连接池技术，而是在每次需要时为线程创建新的连接，那么应用程序的资源使用将十分浪费，而且在负载较重的情况下会导致无法预期的结果。

注意，与MySQL的每个连接均会在客户端和服务器端造成一定的开销（每寸、CPU、关联转换等）。每个连接均会对应用程序和MySQL服务器的可用资源带来一定的限制。无论连接是否执行任何有用的任务，仍将使用这些资源中的相当一部分。

连接池能够使性能最大化，同时还能将资源利用控制在一定的水平之下，如果超过该水平，应用程序将崩溃而不仅仅是变慢。

幸运的是，Sun公司通过JDBC-2.0"可选"接口，完成了JDBC中连接池概念的标准化实施，所有主要应用服务器均实施了能够与MySQL Connector/J一起良好工作的这类API。

通常，你可以在应用服务器的配置文件中配置连接池，并通过Java命名和目录接口（JNDI）访问它。在下面的代码中，介绍了在J2E应用服务器上运行的应用程序中使用连接池的方法：

示例**26.12.** 与**J2EE**应用服务器一起使用连接池

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

import javax.naming.InitialContext;
import javax.sql.DataSource;


public class MyServletJspOrEjb {

    public void doSomething() throws Exception {
        /*
         * Create a JNDI Initial context to be able to
         *  lookup  the DataSource
         *
         * In production-level code, this should be cached as
         * an instance or static variable, as it can
         * be quite expensive to create a JNDI context.
         *
         * Note: This code only works when you are using servlets
         * or EJBs in a J2EE application server. If you are
         * using connection pooling in standalone Java code, you
         * will have to create/configure datasources using whatever
         * mechanisms your particular connection pooling library
         * provides.
         */

        InitialContext ctx = new InitialContext();

         /*
          * Lookup the DataSource, which will be backed by a pool
          * that the application server provides. DataSource instances
          * are also a good candidate for caching as an instance
          * variable, as JNDI lookups can be expensive as well.
          */

        DataSource ds = (DataSource)ctx.lookup("java:comp/env/jdbc/MySQLDB");

        /*
         * The following code is what would actually be in your
         * Servlet, JSP or EJB 'service' method...where you need
         * to work with a JDBC connection.
         */

        Connection conn = null;
        Statement stmt = null;

        try {
            conn = ds.getConnection();

            /*
             * Now, use normal JDBC programming to work with
             * MySQL, making sure to close each resource when you're
             * finished with it, which allows the connection pool
             * resources to be recovered as quickly as possible
             */

            stmt = conn.createStatement();
            stmt.execute("SOME SQL QUERY");

            stmt.close();
            stmt = null;

            conn.close();
            conn = null;
        } finally {
            /*
             * close any jdbc instances here that weren't
             * explicitly closed during normal code path, so
             * that we don't 'leak' resources...
             */
```

```
            if (stmt != null) {
                try {
                    stmt.close();
                } catch (sqlexception sqlex) {
                    // ignore -- as we can't do anything about it here
                }

                stmt = null;
            }

            if (conn != null) {
                try {
                    conn.close();
                } catch (sqlexception sqlex) {
                    // ignore -- as we can't do anything about it here
                }

                conn = null;
            }
        }
    }
}
```

如上例所示，获得JNDI InitialContext并查找到数据库后，其余代码与过去在JDBC编程中使用的类似。

使用连接池时需要牢记的最重要事项是，无论在代码中出现了什么（异常、控制流等），连接以及由连接创建的任何部分（语句、结果集等）均应被关闭，以便能再次使用它们。如不然，它们将纠缠在一起，在最好的情况下，意味着它们所代表的MySQL服务器资源（缓冲区、锁定、套接字等）可能会捆绑一段时间，在最坏的情况下，可能会导致永久捆绑。

连接池的最佳大小是什么？

与所有其他配置经验规则一样，回答是"它取决于具体情况"。尽管最佳大小取决与预期的负载和平均的数据库事务时间，最佳的连接池大小小于你的预期。例如，如果使用的是Sun公司的Java Petstore Blueprint应用程序，对于包含15～20个连接的连接池，使用MySQL和Tomcat，在可接受的相应时间下，可服务于中等程度的负载（600个并发用户）。

要想确定用于应用程序的连接池大小，应使用诸如Apache Jmeter或The Grinder等工具创建负载测试脚本，并对应用程序进行负载测试。

确定出发点的一种简单方法是，将连接池的最大连接数配置为"无限"，运行负载测试，并测量最大的并发连接数。随后，应进行反向操作，确定出使应用程序具有最佳性能的连接池的最小和最大值。

26.3.4.2. 与Tomcat一起使用Connector/J

下述内容基于关于Tomcat-5.x的指示说明，http://jakarta.apache.org/tomcat/tomcat-5.0-doc/jndi-datasource-examples-howto.html，在编写本文档时它是最新的。

首先安装与Connector/J in $CATALINA_HOME/common/lib一起提供的.jar文件，以便它能用于已安装的所有应用程序。

其次，在定义Web应用程序的Context（场景）内，通过为$CATALINA_HOME/conf/server.xml增加声明资源，配置JNDI DataSource：

```
<Context ....>

  ...

  <Resource name="jdbc/MySQLDB"
              auth="Container"
              type="javax.sql.DataSource"/>

  <!-- The name you used above, must match _exactly_ here!

       The connection pool will be bound into JNDI with the name
       "java:/comp/env/jdbc/MySQLDB"
  -->

  <ResourceParams name="jdbc/MySQLDB">
    <parameter>
      <name>factory</name>
      <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
    </parameter>

    <!-- Don't set this any higher than max_connections on your
         MySQL server, usually this should be a 10 or a few 10's
```

```
      of connections, not hundreds or thousands -->

  <parameter>
    <name>maxActive</name>
    <value>10</value>
  </parameter>

  <!-- You don't want to many idle connections hanging around
       if you can avoid it, only enough to soak up a spike in
       the load -->

  <parameter>
    <name>maxIdle</name>
    <value>5</value>
  </parameter>

  <!-- Don't use autoReconnect=true, it's going away eventually
       and it's a crutch for older connection pools that couldn't
       test connections. You need to decide if your application is
       supposed to deal with SQLExceptions (hint, it should), and
       how much of a performance penalty you're willing to pay
       to ensure 'freshness' of the connection -->

  <parameter>
    <name>validationQuery</name>
    <value>SELECT 1</value>
  </parameter>

 <!-- The most conservative approach is to test connections
      before they're given to your application. For most applications
      this is okay, the query used above is very small and takes
      no real server resources to process, other than the time used
      to traverse the network.

      If you have a high-load application you'll need to rely on
      something else. -->

  <parameter>
    <name>testOnBorrow</name>
    <value>true</value>
  </parameter>

 <!-- Otherwise, or in addition to testOnBorrow, you can test
      while connections are sitting idle -->

  <parameter>
    <name>testWhileIdle</name>
    <value>true</value>
  </parameter>

  <!-- You have to set this value, otherwise even though
       you've asked connections to be tested while idle,
       the idle evicter thread will never run -->

  <parameter>
    <name>timeBetweenEvictionRunsMillis</name>
    <value>10000</value>
  </parameter>

  <!-- Don't allow connections to hang out idle too long,
       never longer than what wait_timeout is set to on the
       server...A few minutes or even fraction of a minute
       is sometimes okay here, it depends on your application
       and how much spikey load it will see -->

  <parameter>
    <name>minEvictableIdleTimeMillis</name>
    <value>60000</value>
  </parameter>

  <!-- Username and password used when connecting to MySQL -->

  <parameter>
   <name>username</name>
   <value>someuser</value>
  </parameter>

  <parameter>
   <name>password</name>
   <value>somepass</value>
  </parameter>
```

```
    <!-- Class name for the Connector/J driver -->

    <parameter>
       <name>driverClassName</name>
       <value>com.mysql.jdbc.Driver</value>
    </parameter>

    <!-- The JDBC connection url for connecting to MySQL, notice
         that if you want to pass any other MySQL-specific parameters
         you should pass them here in the URL, setting them using the
         parameter tags above will have no effect, you will also
         need to use &amp; to separate parameter values as the
         ampersand is a reserved character in XML -->

    <parameter>
      <name>url</name>
      <value>jdbc:mysql://localhost:3306/test</value>
    </parameter>

  </ResourceParams>
</Context>
```

一般而言，应遵循关于相应Tomcat版本的安装说明，这是因为，在Tomcat中配置数据源的方式会随时变化，很不幸，如果在XML文件中使用了错误的语法，很可能会以异常结束，如下所示：

```
Error: java.sql.SQLException: Cannot load JDBC driver class 'null ' SQL
state: null
```

26.3.4.3. 与JBoss一起使用Connector/J

下述说明适用于JBoss-4.x。要想使应用服务器能够使用JDBC驱动类，请将与Connector/J一起提供的.jar文件拷贝到用于服务器配置的lib目录下（通常称为"默认"）。随后，在相同的配置目录下，在名为"deploy"的子目录下，创建以"-ds.xml"结尾的数据源配置文件，用于通知Jboss将该文件当作JDBC数据源。该文件应包含下述内容：

```
<datasources>
    <local-tx-datasource>
        <!-- This connection pool will be bound into JNDI with the name
             "java:/MySQLDB" -->

        <jndi-name>MySQLDB</jndi-name>
        <connection-url>jdbc:mysql://localhost:3306/dbname</connection-url>
        <driver-class>com.mysql.jdbc.Driver</driver-class>
        <user-name>user</user-name>
        <password>pass</password>

        <min-pool-size>5</min-pool-size>

        <!-- Don't set this any higher than max_connections on your
         MySQL server, usually this should be a 10 or a few 10's
         of connections, not hundreds or thousands -->

        <max-pool-size>20</max-pool-size>

        <!-- Don't allow connections to hang out idle too long,
         never longer than what wait_timeout is set to on the
         server...A few minutes is usually okay here,
         it depends on your application
         and how much spikey load it will see -->

        <idle-timeout-minutes>5</idle-timeout-minutes>

        <!-- If you're using Connector/J 3.1.8 or newer, you can use
             our implementation of these to increase the robustness
             of the connection pool. -->

        <exception-sorter-class-name>com.mysql.jdbc.integration.jboss.ExtendedMysqlExceptionSorter</exception-sorter-class-name>
        <valid-connection-checker-class-name>com.mysql.jdbc.integration.jboss.MysqlValidConnectionChecker</valid-connection-checker-class-name>

    </local-tx-datasource>
</datasources>
```

26.3.5. 诊断 Connector/J方面的问题

26.3.5.1. 常见问题和解决方案
26.3.5.2. 如何通报缺陷和问题

在本节中，介绍了如何解决使用Connector/J时遇到的问题。

26.3.5.1. 常见问题和解决方案

对于MySQL Connector/J用户，会遇到一些常见的共同问题。在本节中，介绍了它们的症状和相应的解决方法。 关于更进一步的信息，请参见"支持"一节。

27.3.5.1.1:

问题:

当我尝试用MySQL Connector/J连接到数据库时，遇到下述异常:

```
SQLException: Server configuration denies access to data source

SQLState: 08001

VendorError: 0
```

出现了什么问题？ 使用MySQL命令行客户端时，连接良好。

回答:

MySQL Connector/J必须使用TCP/IP套接字来连接MySQL，原因在于Java不支持Unix Domain套接字。因此，当MySQL Connector/J连接到MySQL时，MySQL服务器的安全管理器将使用其授权表判断是否允许连接。必须添加授权才能允许该操作。下面给出了一个执行该操作的示例（但并非最安全的）。

从mysql命令行客户端以能够授权的用户身份登录，并发出下述命令:

```
GRANT ALL PRIVILEGES ON [dbname].* to

               '[user]'@'[hostname]' identified by

               '[password]'
```

用你的数据库名称替换[dbname]，用用户名替换[user]，用MySQL Connector/J将连接的主机替换[hostname]，并用打算使用的密码替换[password]。注意，对于从本地主机进行连接的主机名部分，RedHat Linux将失败。在这种情况下，对于[hostname]值，需要使用"localhost.localdomain"。随后，发出FLUSH PRIVILEGES命令。

注释:

除非添加了"--host"标志，并为主机使用了不同于"localhost"的其他设置，否则将无法使用mysql命令行客户端测试连通性。如果使用了特殊的主机名"localhost"，mysql命令行客户端将使用Unix域套接字。如果正在测试与"localhost"的连通性，请使用"127.0.0.1"作为主机名。

警告

如果你不了解"GRANT"命令是干什么的，或不了解该命令的工作方式，在尝试更改权限之前，请阅读MySQL手册中的 一般安全事宜以及MySQL访问权限体系一节。

如果在MySQL中不恰当地更改了权限和许可，可能会使服务器不会具有最佳的安全性能。

27.3.5.1.2:

问题:

我的应用程序抛出SQLException"无恰当的驱动程序"。为什么会出现该情况？

回答:

出现了两种情况之一。或是1驱动程序未位于你的CLASSPATH中（请参见前面的"安装部分"），或是URL格式不正确（请参见用MySQL Connector/J开发应用程序）。

27.3.5.1.3:

问题:

当我试图在Java程序或应用程序中使用MySQL Connector/J时，遇到类似下面的异常:

```
SQLException: 无法连接到host:3306上的MySQL服务器。
在你尝试连接的机器／端口上是否有正在运行的MySQL服务器?


(java.security.AccessControlException)
SQLState: 08S01
VendorError: 0
```

回答:

或许是因为你正在运行Applet，你的MySQL服务器是采用"--skip-networking"选项集安装的；或许是因为MySQL服务器位于防火墙之后。

Applet仅能使网络连接返回运行Web服务器的机器，该Web服务器提供了用于Applet的.class文件。这意味着，要想使其工作，MySQL必须运行在相同的机器上（或必须使某类端口重定向）。这还意味着，你无法通过你的本地文件系统来测试Java程序，你必须将它们放在Web服务器上。

MySQL Connector/J仅能使用TCP/IP与MySQL进行通信，这是因为Java不支持Unix域套接字。如果MySQL是用"--skip-networking"标志启动的，或采用了防火墙，TCP/IP与MySQL的通信可能会受到影响。

如果MySQL是用"--skip-networking"选项集启动的（例如MySQL服务器的Debian Linux包即用于该目的），需要在文件/etc/mysql/my.cnf或/etc/my.cnf中将其注释掉。当然，my.cnf文件也可能位于MySQl服务器的"data"目录下或其他地方（取决于系统中MySQL的编译方式）。MySQL AB创建的二进制文件总会在查找/etc/my.cnf和[datadir]/my.cnf。如果为MySQL服务器部署了防火墙，需要对防火墙进行配置，允许从运行Java代码的主机在MySQL监听的端口上（默认为3306）建立与MySQL服务器的TCP/IP连接。

27.3.5.1.4:

问题:

I我的小服务程序／应用程序白天工作良好，但在晚上却停止工作。

回答:

不工作时间超过8小时后，MySQL关闭了连接。你或许需要使用能处理失效连接的连接池，或使用"autoReconnect"参数（请参见用MySQL Connector/J开发应用程序）。

此外，你应在应用程序中俘获 SQLException并处理它们，而不是在应用程序退出前一直传播它们，这是1个好的编程习惯。在查询处理过程中遇到网络连通性方面的问题时，MySQL Connector/J会将SQLState（参见APIDOCS中的java.sql.SQLException.getSQLState()）设置为"08S01"。随后，应用程序代码将尝试再次连接到MySQL。

在下面的示例（simplistic）中，给出了能够处理这类异常的代码:

示例**26.13.** 重试逻辑的事务示例

```
public void doBusinessOp() throws SQLException {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        //
        // How many times do you want to retry the transaction
        // (or at least _getting_ a connection)?
        //
        int retryCount = 5;

        boolean transactionCompleted = false;

        do {
            try {
                conn = getConnection(); // assume getting this from a
```

```
                                // javax.sql.DataSource, or the
                                // java.sql.DriverManager

        conn.setAutoCommit(false);

        //
        // Okay, at this point, the 'retry-ability' of the
        // transaction really depends on your application logic,
        // whether or not you're using autocommit (in this case
        // not), and whether you're using transacational storage
        // engines
        //
        // For this example, we'll assume that it's _not_ safe
        // to retry the entire transaction, so we set retry count
        // to 0 at this point
        //
        // If you were using exclusively transaction-safe tables,
        // or your application could recover from a connection going
        // bad in the middle of an operation, then you would not
        // touch 'retryCount' here, and just let the loop repeat
        // until retryCount == 0.
        //
        retryCount = 0;

        stmt = conn.createStatement();

        String query = "SELECT foo FROM bar ORDER BY baz";

        rs = stmt.executeQuery(query);

        while (rs.next()) {
        }

        rs.close();
        rs = null;

        stmt.close();
        stmt = null;

        conn.commit();
        conn.close();
        conn = null;

        transactionCompleted = true;
    } catch (SQLException sqlEx) {

        //
        // The two SQL states that are 'retry-able' are 08S01
        // for a communications error, and 41000 for deadlock.
        //
        // Only retry if the error was due to a stale connection,
        // communications problem or deadlock
        //

        String sqlState = sqlEx.getSQLState();

        if ("08S01".equals(sqlState) || "41000".equals(sqlState)) {
            retryCount--;
        } else {
            retryCount = 0;
        }
    } finally {
        if (rs != null) {
            try {
                rs.close();
            } catch (SQLException sqlEx) {
                // You'd probably want to log this . . .
            }
        }

        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException sqlEx) {
                // You'd probably want to log this as well . . .
            }
        }

        if (conn != null) {
            try {
                //
                // If we got here, and conn is not null, the
```

```
                        // transaction should be rolled back, as not
                        // all work has been done

                        try {
                            conn.rollback();
                        } finally {
                            conn.close();
                        }
                    } catch (SQLException sqlEx) {
                        //
                        // If we got an exception here, something
                        // pretty serious is going on, so we better
                        // pass it up the stack, rather than just
                        // logging it. . .

                        throw sqlEx;
                    }
                }
            }
        } while (!transactionCompleted && (retryCount > 0));
    }
```

27.3.5.1.5:

问题:

我正尝试使用JDBC-2.0可更新结果集，但遇到异常，说我的结果集不可更新。

回答:

由于MySQL没有行ID，MySQL Connector/J仅能更新来自查询且位于有至少一个主键的表上的结果集，查询必须选择所有的主键，而且查询即能作用在1个表上（即不存在联合）。在JDBC规范中给出了这方面的介绍。

26.3.5.2. 如何通报缺陷和问题

通报缺陷的正常地址是http://bugs.mysql.com/，它也是我方缺陷数据库的地址。这是1个公共数据库，任何人都能浏览它并进行相应的搜索。如果你已登录到系统，也应能输入新的报告。

如果发现MySQL中存在敏感的安全缺陷，请发送电子邮件至security@mysql.com。

编写良好的缺陷报告需要耐心，但在第1时间正确地完成它不仅能节省我们的时间，也能节省你自己的时间。良好的缺陷报告应包含对缺陷的完整测试情况，以便我们你能够在下个版本中更正该缺陷。

本节介绍的内容用于帮助你正确地编写报告，从避免将你的时间浪费在对我们帮助不大或没有帮助的事上，

如果有1份可重复的缺陷报告，请将其提交到缺陷数据库，http://bugs.mysql.com/。

对于任何我们能再现的缺陷，在下一个MySQL版本中修正它的机会很大。

要想通报其他问题，请使用MySQL邮件列表。

请注意，我们可能会对包含过多信息的消息作出响应，但不太会对包含过少信息的消息作出回应。人们常会省略掉一些事实，因为他们认为自己知道了故障的原因，并想当然地认为这类细节无关紧要。

良好的原则是：如果你对陈述某事犹豫不定，请陈述之。如果我们要求你提供初始报告中缺少的信息，在报告中编写多行信息源比等候回复要快，麻烦也更小。

在缺陷报告，最常犯的错误包括：（a）未包含所使用Connector/J或MySQL的版本号，以及（b）未完全描述安装了Connector/J的平台（包括JVM版本，平台类型，以及所安装MySQL本身的版本号）。

这是高度相关的信息，如果没有它，99％的缺陷报告无用。我们遇到这类问题，“为何它对我没用”？ 随后，我们发现在该MySQL版本中，所请求的特性尚未实施，或在较新的MySQL版本中已更正了报告中描述的缺陷。

有些时候，错误与平台相关，在这类情况下，如果不知道操作系统和平台的版本号，我们几乎不可能更正任何问题。

如果可能，你应创建1份可重复的、不含任何第三方类的独立测试案例。

为了是该进程流线化，我们与Connector/J一起提供了用于测试的基本类，名为com.mysql.jdbc.util.BaseBugReport。要想使用该类为Connector/J创建1个测试案例，你应应创建自己的从com.mysql.jdbc.util.BaseBugReport继承的类，并覆盖方法setUp()、tearDown()和runTest()。

在setUp()方法中，创建用于创建表的代码，并用演示缺陷所需的数据填充表。

在runTest ()方法中，使用在"setUp"方法中创建的表和数据，创建用于演示缺陷的代码。

在tearDown()方法中，撤销在setUp()方法中创建的任何表。

对于上述三种方法中的任何一种，应使用getConnection ()各种变体中的一种创建与MySQL的JDBC连接。

- getConnection()：提供了与在getUrl()中指定的JDBC URL的连接。如果连接已存在，返回该连接，否则将创建新的连接。
- getNewConnection()：如果需要为缺陷报告获得新的连接（即包含1个以上的连接），应使用它。
- getConnection(String url)：使用给定的URL返回连接。
- getConnection(String url, Properties props)：使用给定的URL和属性返回连接。

如果需要使用不同于"jdbc:mysql:///test"的JDBC URL，还应覆盖方法getUrl()。

在演示你所预计行为的测试案例中（相对于你观察到的世纪行为，这是你填充错误报告的最可能原因），使用assertTrue(boolean expression)和assertTrue(String failureMessage, boolean expression)方法创建必须满足的条件。

最后，创建用于创建测试案例实例的main ()方法，并调用run方法：

```
public static void main(String[] args) throws Exception {
      new MyBugReport().run();
 }
```

完成了测试案例并证实它能演示你所通报的缺陷后，请将该案例与缺陷报告一起上传到http://bugs.mysql.com/。

26.3.6. Changelog

```
# Changelog
# $Id: CHANGES,v 1.38.4.206 2005/05/12 15:25:54 mmatthews Exp $


05-17-05：版本3.2.1-alpha


    - 现已不再重视Autoreconnect功能（即autoReconnect=true）。
      如果尝试并使用它将抛出异常，使用"enableDeprecatedAutoreconnect=true"可继续使用autoReconnect。但是，在Connector/J 3.3中将删除该项特性，请参见手册中关于不需要使用autoReconnect的相应解决方案。


    - 现在，驱动程序将检查是否设置了服务器变量"init_connect"，如果设置了该变量，将检查autocommit（自动提交）设置，并应用它。


    - 如果连接的服务器版本在5.0.x以上，而且Statement.setFetchSize( > 0)，驱动程序将尝试使用服务器预处理语句，并使用结果集"cursors"获取语句。


    - ServerPreparedStatements现在能正确地将BLOB/CLOB数据以"流方式"发送至服务器。你可以使用属性"blobSendChunkSize"配置程序块大小的阈值（默认值为1MB）。
```

- 支持sql模式NO_BACKSLASH_ESCAPES以及非服务器端预处理语句。

12-23-04：版本3.2.0-alpha

-更正了DatabaseMetaData.supportsCatalogIn*()错误的返回值。

-使用ServerPreparedStatements以及MySQL 5.0或更高版本时，支持基于“cursor”的结果集。结果集需为“仅正向”结果集，并需启用针对该项特性的非零获取大小。

- 重新分解了预处理语句的where逻辑，服务器端预处理语句保持有效。

10-07-05：版本3.1.11-stable

- 更正了BUG#11629：当字符编码是“utf8”时控制台上的伪“!”。

-更正了为丢失“;”（用于“plain”语句）的测试案例生成的语句。

- 更正了BUG#11663：为服务器端预处理语句生成的不正确的testcase脚本。

-更正了因修补BUG#11552而导致的回归，对于BUG#11552，当整数处于正号类型的范围内时，对于无符号整数，该缺陷将导致驱动程序返回错误值。

-将源代码移到了svn repo。

-更正了BUG#11797：转义标志不考虑用于转义用途的层套式单引号。

-使用服务器端预处理语句时，不识别GEOMETRY类型。

-更正了BUG#11879：ReplicationConnection不会切换至从连接，抛出“目录不能为空”异常。

-更正了BUG#12218，主连接和具有复制连接的从连接之间共享的属性。

- 更正了BUG#10630。如果语句已关闭，Statement.getWarnings()无法与NPE一起工作。

-需要时，在PreparedStatement.ParseInfo()中，即能从SQL获取char[]。

-更正了BUG#12104，服务器端预处理语句不能处理Geometry类型。

-更正了BUG#11614，使用多字节字符编码时，StringUtils.getBytes()不工作，在“_characters_”中指定了长度。

- BUG#11798 Pstmt.setObject(...., Types.BOOLEAN)

更正了 , 抛出异常。

-更正了BUG#11976，maxPerformance.properties拼错“elideSetAutoCommits”。

-更正了BUG#11575，对于在Window平台上的服务器，DBMD.storesLower/Mixed/UpperIdentifiers()通报不正确的值。

-更正了BUG#11190，在ResultSet.moveToCurrentRow()之前调用了ResultSet.moveToInsertRow()时，ResultSet.moveToCurrentRow()不起作用。

-更正了BUG#11115，使用服务器端预处理语句和.setBytes()时，VARBINARY数据发生崩溃。

-更正了BUG#12229，服务器端预处理语句挂起explainSlowQueries。

-更正了BUG#11498，转义处理器不考虑用双引号分隔的字符串。

-对于服务器端预处理语句，增加了对更改流参数的限制。只要在执行前设置了“_all_”流参数，可不必调用.clearParameters()。（由于客户端／服务器协议的限制，预处理语句不能复位服务器端的“_individual_ stream”数据）。

-修改了Field类，*Buffer和MysqlIO，能够识别大于Integer.MAX_VALUE的字段长度。

-更新了DBMD.supportsCorrelatedQueries()，当版本高于4.1时返回“真”，更新了supportsGroupByUnrelated()使之返回“真”，并更新了getResultSetHoldability()使之返回HOLD_CURSORS_OVER_COMMIT。

-更正了BUG#12541，DatabaseMetaData.getIndexInfo()中的catalog（目录）参量处理，它也意味着对DatabaseMetaData中下述方法的更改：

- getBestRowIdentifier()
- getColumns()
- getCrossReference()
- getExportedKeys()
- getImportedKeys()
- getIndexInfo()
- getPrimaryKeys()
- getProcedures() (and thus indirectly getProcedureColumns())
- getTables()

上述所有方法中的“catalog”参量现具有下述行为特征：

-如果指定为Null，表示不会使用catalog来过滤结果（因此将搜索所有数据库），除非在JDBC URL属性中设置了“nullCatalogMeansCurrent=true”。

-指定为“”表示当前catalog，尽管它不是十分兼容JDBC规范，但它是为传统用户保留的。

- catalog API

指定 ，使之按 文档中阐明的方式工作。

–使得jdbc2.optional软件包中的“封装”连接能够使用Connection.clientPrepare()（使用ConnectionPoolDataSource实例创建连接）。

–为客户端增加了Connection.isMasterConnection()，以确定多主机主／从连接是否连接至列表中的第1个主机。

–更正了BUG#12753，URL属性用于“=”的标志会导致“sessionVariables=....”无法正确地参数化。

–更正了BUG#11781，当DatabaseMetaData方法使用该信息时，所引用的外键信息不能被正确解析。

–考虑到流缓冲区大小和信息包报头，“sendBlobChunkSize”属性现在与“max_allowed_packet”密切结合在一起，当max_allowed_packet的大小类似于默认的“sendBlobChunkSize”时（为1MB），能避免PacketTooBigExceptions。

–CallableStatement.clearParameters()现能清除与INOUT/OUTPUT参数和INPUT参数相关的属性。

–更正了BUG#12417，Connection.prepareCall()是区分大小写的数据库名称（在Windows系统上）。

–更正了BUG#12752，对于版本高于4.0.x的服务器，Cp1251不正确地映射至win1251。

–更正了BUG#12970，使用DatabaseMetaData.getColumns()时，java.sql.Types.OTHER返回BINARY和VARBINARY列。

–引用参数约束列表前，ServerPreparedStatement.getBinding()现在将检查语句是否已关闭，以避免抛出NullPointerException。

–更正了BUG#13277，无论何时，当调用需要连接引用的方法时，来自Statement.getGeneratedKeys()的ResultSetMetaData将导致NullPointerExceptions的抛出。

–自5.0起，Field类ResultSetMetaData.getColumnClassName()和ResultSet.getObject(int)的反向移植出现了变化，更正了VARCHAR BINARY/VARBINARY和相关类型有关的行为。

–更正了NullPointerException，当参数为NULL时，在很多DatabaseMetaDataMethods中将“catalog”参数转换为byte[]时出现的异常（对于结果集）。（根据JDBC规范，从技术角度上不允许“null”，但从历史观点上，我们允许使用它）。

–从5.0起，VAR[BINARY|CHAR] [BINARY]类型检测的反向移植。

–即使无法打开本地文件，也能读取MysqlIO.sendFileToServer()中响应，否则，下一个发出的查询将失败，这是因为，它正在读取对发送至服务器的空LOAD DATA INFILE信息包的响应。

–避开了BUG#13374，已关闭结果集上的ResultSet.getStatement()返回NULL（按照JDBC 4.0规范，但不具有向后兼容性）。将连接属性“retainStatementAfterResultSetClose”设为“真”，以便能够在通过.getStatement()关闭了结果集后检索结果集的语句（默认为“假”以便与JDBC兼容，并降低使用JDBC的代码名泄漏语句实例的机会）。

–更正了BUG#13453，URL配置参数不允许在它们的值中使用“&”或“=”。现在JDBC驱动程序能够对配置参数进行相应地解析，就像它们是使用application/x-www-form-urlencoded格式（在java.net.URLDecoder中指定，http://java.sun.com/j2se/1.5.0/docs/api/java/net/URLDecoder.html）进行编码那样。

如果在配置属性中出现字符“%”，现在必须用“%25”表示，它是使用“application/x-www-form-urlencoded”编码使“%”的已编码形式。

-配置属性“sessionVariables”现在允许你指定以“@”符号开始的变量。

-更正了BUG#13043，为低于4.1.0版的服务器允许了“gatherPerfMetrics”时，如果查询未使用任何表，结果集的构造函数将抛出NullPointerException。

06-23-05：版本3.1.10-stable

-更正了因无指定数据库而导致的MysqlIO.changeDatabaseTo()中的异常。

-首次实现了用于PreparedStatement.getParameterMetadata()的ParameterMetadata。仅能与CallableStatements一起完全发挥作用，这是因为当前的服务器端预处理语句会将所有参数返回为VARCHAR类型。

06-22-05：版本3.1.9-stable

-彻底检查了字符集配置，现在所有相关项均能位于属性文件中。

-如果在用于Windows-31J的服务器上可用，驱动程序能正确使用CP932，以及CP932和MS932 java编码名称，否则将求助于仅为近似的SJIS。目前，仅在MySQL-5.0.3和更高版本上（以及MySQL-4.1.12/13，取决于何时反向移植字符集），才能可靠地支持任何CP932变体。

-更正了BUG#9064，com.mysql.jdbc.PreparedStatement.ParseInfo对toCharArray()的不必要调用。

-更正了Bug#10144，如果serverPrepare()失败，ServerPreparedStatement中的内存泄漏。

-将清单文件实际写入正确位置，以便其终止于二进制jar文件。

-增加了“createDatabaseIfNotExist”属性（默认为“假”），该属性会使驱动程序请求服务器创建在URL中指定的数据库（如果该数据库不存在的话）。必须拥有恰当的数据库创建权限才能执行该任务。

-更正了BUG#10156，对于ResultSet.getInt()，无符号SMALLINT被当作带符号类型，更正了所有的UNSIGNED整数和服务器端预处理语句，以及用于UNSIGNED TINYINT的ResultSet.getObject()。

-更正了BUG#10155，解析客户端预处理语句时不能识别双引号。

-使得enableStreamingResults()在com.mysql.jdbc.jdbc2.optional.StatementWrapper上可见。

-使得ServerPreparedStatement.asSql()能正确工作，自动解释功能可以与服务器端预处理语句一起工作。

-使得兼容JDBC2的软件包成为公共包，以便允许访问厂家扩展。

-整理了剖析工具事件的记录功能，移动了代码以将剖析工具事件转储为com.mysql.jdbc.log.LogUtils的字符串，以便第三方能够使用它。

- DatabaseMetaData.supportsMultipleOpenResults()现在返回“真”。DBMD刚刚丢失某一内容时，驱动程序在一段时间内支持它。

-更正了BUG#10310，驱动程序不支持用来调用存储程序的{?=CALL(...)}。其中包括，为DatabaseMetaData.getProcedures()和getProcedureColumns()增加了函数检索支持。

-更正了BUG#10485，用ResultSet.getString()检索YEAR(2)时抛出SQLException。现在，驱动程序总是将YEAR类型当作java.sql.Dates对待，并为getString()返回正确值。另外，可以将“yearIsDateType”连接属性设置为“假”，并将值作为SHORT进行处理。

-当“tinyInt1isBit=true”时（默认），使用新的配置属性“transformedBitIsBoolean”（默认为“假”），对于TINYINT(1)列，返回的数据类型可在Types.BOOLEAN和Types.BIT之间切换。如果设为“假”（默认），对于TINYINT(1)列，DatabaseMetaData.getColumns()和ResultSetMetaData.getColumnType()将返回Types.BOOLEAN。如果为“真”，将返回Types.BOOLEAN。无论该配置属性是如何设置的，如果允许了“tinyInt1isBit”，具有TINYINT(1)类型的列将作为ResultSet.getObject(..)的java.lang.Boolean实例返回，而且ResultSetMetaData.getColumnClassName()将返回“java.lang.Boolean”。

-更正了BUG#10496，与cp932或eucjpms一起使用“characterSetResults”属性时将抛出SQLException。

-重组了目录格局，源码现位于“src”文件夹下，创建时不填充父目录，输出位于“./build”下，分发版位于“./dist”下。

-这类支持/缺陷跟踪特性，将“autoGenerateTestcaseScript”设置为“真”时，该特性能够为STDERR生成.sql测试脚本。

-更正了BUG#10850，使用服务器端预处理语句时，不会将“0长度”流发送给服务器。

-现在，如果设置了“cachePrepStmts=true”，也会使连接进行高速缓冲操作，检查驱动程序的执行情况，以判断预处理语句是否能够在服务器端工作，并能在连接的生命期内对服务器端预处理语句进行高速缓冲处理。与以往一样，参数“prepStmtCacheSize”负责控制这些高速缓冲区的大小。

-进行了尝试，以更为优美的方式处理OutOfMemoryErrors。尽管所能作的事情不是很多，但在大多数情况下它们能关闭所遇到的连接，这样，进一步的操作不会进入状态不明的连接。出现OOM时，连接上的任何进一步操作均将失败，同时抛出“连接已关闭”异常，还将列出作为隐式连接关闭事件原因的OOM异常。

-如果未要求，执行服务器端预处理语句时不发送COM_RESET_STMT。

-驱动程序将检测是否正在运行MySQL-5.0.7或更高版本，而且不扫描正在处理的语句中的“LIMIT ?[,?]”，这是因为目前服务器支持这类查询。

-更正了BUG#11115，使用服务器端预处理语句和ResultSet.getBytes()时，VARBINARY数据发生崩溃

-Connection.setCatalog()想在能够识别“useLocalSessionState”配置属性，将该属性设为“真”时，如果所请求的catalog与当前catalog相同，会阻止驱动程序将“USE ...”发送给服务器。

-增加了下述配置捆包，通过“useConfigs”配置属性使用1个或多个：

* maxPerformance：考虑后果时的最大性能
* solarisMaxPerformance：Solaris的需性能，在可能的情况下避免系统调用。
* 3-0-Compat：与Connector/J 3.0.x功能兼容。

-增加了“"maintainTimeStats”配置属性（默认为“真”），用于通知驱动程序是否应跟踪上次查询时间，以及上次成功将信息包发送到服务器的时间。如果将其设置为“假”，将删除每查询的两次系统调用。

-更正了BUG#11259，autoReconnect的ping操作会导致连接启动时的异常。

-更正了BUG#11360，Connector/J将查询两次转储到SQLException。

-更正了PreparedStatement.setClob()，不接受Null作为参数。

-更正了BUG#11411，生产包不包括JBoss集成类。

-删除了使用“usage advisor”时无意义的“开销昂贵的类型转换”告警。

04-14-05：版本3.1.8-stable

-更正了DatabaseMetaData.getTables()，未要求时，以请求表的类型之一返回视图。

-在5.0.3和更高版本的MySQL中，增加了对新精度数学DECIMAL类型的支持。

-更正了ResultSet.getTime()，作用于服务器端预处理语句的Null值上时抛出NPE。 -使Connection.ping()成为public方法。 -更正了Bug#8868，DATE_FORMAT()查询从getObject()返回BLOB。 - ServerPreparedStatements现在能正确地将BLOB/CLOB数据以“流方式”发送至服务器。你可以使用属性“blobSendChunkSize”配置程序块大小的阈值（默认值为1MB）。

-生成预处理语句时，BlobFromLocator现在能使用正确的ID引用。

-使用连接属性“sessionVariables”的逗号分隔列表方式传递它们，能够在连接时预先设置服务器端会话变量。

-为使用“autoReconnect=true”的用户更正了ping()中的回归问题。

-更正了BUG#9040，PreparedStatement.addBatch()不能与服务器端预处理语句和流式BINARY数据一起工作。

-更正了BUG#8800，对于运行于区分大小写的文件系统上的服务器，DBMD.supportsMixedCase*Identifiers()返回错误值。

-更正了BUG#9206，对于characterSetResults配置属性，不能使用“UTF-8”。

-更正了BUG#9236，连续BUG#8868，在查询中使用了多个函数，查询本应返回非字符串类型，但当临时表解析它们时，返回类型突然变为难以理解的二进制字符串（针对服务器限制采取了相应的避规措施）。此外，还更正了类型为CHAR(n) CHARACTER SET BINARY的字段，使其能够为RSMD.getColumnClassName()和ResultSet.getObject()返回正确／匹配的类。

-更正了BUG#8792，对于“仅正向/只读”结果集（我们支持该方式），DBMD.supportsResultSetConcurrency()未返回“真”。

-更正了BUG#8803，访问时，DBMD.getBestRowIdentifier()的“DATA_TYPE”列导致ArrayIndexOutOfBoundsException（事实上，未返回任何值）。

-检查了将char/varchar列数据转换为数值时的空字符串('')，如果配置属性“emptyStringsConvertToZero”被设为“假”，抛出异常（为了与3.0版的向后兼容，默认情况下，将其设为“真”，但在3.2版中，最可能的默认设置为“假”）。

-更正了BUG#9320，在特定条件下，当未使用服务器端预处理语句时，PreparedStatement.getMetaData()在数据库中插入了空白行。

-Connection.canHandleAsPreparedStatement()现在将尽“最大努力”来识别带有占位符的LIMIT子句，目的在于，对于服务器目前不能将其当作服务器端预处理语句予以处理的语句，为其生成处理措施时，获得较少的错误肯定结果。

-更正了build.xml，如果log4j不可用，不编译log4j日志功能。

-增加了对c3p0连接池的（http://c3p0.sf.net/）验证/连接检验器接口的支持，如果服务器可用，该检验器接口将使用轻型“COM_PING”调用。要想使用它，请配置c3p0连接持的“connectionTesterClassName”属性，以使用“com.mysql.jdbc.integration.c3p0.MysqlConnectionTester”。

-更好的检测引用字符串的内外LIMIT，以便驱动程序能更加正确地判断是否可以在服务器上处理预处理语句。

-更正了BUG#9319，当驱动程序试图确定参数计数/类型时，分不清不同数据库中具有相同名称的存储程序。

-为ResultSet和Statement实施增加了最终确定器，以便与JDBC规范兼容，该规范要求，如果它们未明确关闭，在无用单元收集阶段应关闭这些资源。

-更正了BUG#9682，对于具有DECIMAL参数和存储要求的存储程序，如果包含“,”，将失败。

- PreparedStatement.setObject(int, Object, int type, int scale)现在能使用针对BigDecimal实例的标度值。

-更正了BUG#9704，当已有结果集是.close()d时，Statement.getMoreResults()可能抛出NPE。

-性能度量特性现能收集关于SELECT中引用的表数量的信息。

-现在能够自动配置日志系统。如果用户通过URL属性“logger”或系统属性“com.mysql.jdbc.logger”设置了值，那么将使用用户的设置，否则将采用下述步骤自动检测设置：

如果Log4j可用，将使用它，接下来是JDK1.4日志功能，再接下来是STDERR日志功能。

-更正了BUG#9778，即使数据库版本不支持视图，如果请求了视图，DBMD.getTables()不应返回表。

-更正了驱动程序，当在服务器端预处理语句返回的结果集上调用了ResultSet.getBoolean()时，对于“-1”不返回“真”。

-为.jar文件增加了Manifest.MF文件以及实施信息。

-在Field.isOpaqueBinary()中提供了更多测试，以识别晦涩的二进制字段（即具有CHAR(n)和CHARACTER SET BINARY类型的字段），这类字段来自各种标量函数和聚合函数（返回字符串）的输出。

-更正了BUG#9917，出于传统方面的考虑，即使它与JDBC不兼容，也应接受DBMD方法中用于catalog（使用当前值）的Null。将连接属性“nullCatalogMeansCurrent”设置为“假”可禁止它（在C/J 3.2.x中，它是默认设置）。

-更正了BUG#9769，出于传统方面的考虑，即使它与JDBC不兼容，也应接受DBMD中用于名称（“%”）的Null。将连接属性“nullNamePatternMatchesAll”设置为“假”可禁止它（在C/J 3.2.x中，它是默认设置）。

02-18-05：版本3.1.7-stable

-更正了BUG#7686，Timestamp关键字列数据需要分离的“_binary”，用于UpdatableResultSet.refreshRow()。

-更正了BUG#7715，对于服务器端预处理语句和可更新结果集，Timestamps错误地转换为字符串。

-检测字符串表单（以前为整数）中的sql_mode变量，并恰当调整字符串的引用方法。

-增加了“holdResultsOpenOverStatementClose”属性（默认为“假”），它能将statement.close()的结果集保持打开状态，或执行相同语句（由Kevin Burton推荐）。

-更正了BUG#7952，在故障切换配置下回退至主连接时的无限递归。

-如果允许了高速缓冲，将禁止4.1.10版之前MySQL-4.1版本的多语句功能（如果以允许的话），这是因为在该配置下，服务器会返回错误的结果。

-更正了configureClientCharset()中的重复代码，该类代码将阻止useOldUTF8Behavior=true的恰当运行。

-删除了“dontUnpackBinaryResults”功能，现在，驱动程序将按原样保存来自服务器端预处理语句的结果（与服务器提供的相同，并在需要时展开）。

-更正了BUG#8096，使用服务器端预处理语句时，模拟定位器破坏了二进制数据。

-更正了ServerPreparedStatement.serverPrepare()的同步事宜，如果在多个线程间共享连接，可能会导致死锁/崩溃。

-默认情况下，驱动程序能通过各种Connection.prepareStatement()的各种变体扫描SQL，以判断它是否是能够在服务器端支持的语句类型，如果不被服务器端支持，会将其作为客户端模拟预处理语句进行处理（BUG#4718））。也可以通过在JDBC URL中传递“emulateUnsupportedPstmts=false”禁止该特性。

-从CallableStatement中删除了用作输入／输出参数的“_binary”引介词。

-对于注册为*BINARY的输出参数，总返回byte[]。

-对于PreparedStatement.setObject(n, "true", Types.BIT)，将布尔“True”的正确值发送给服务器。

-更正了与连接有关的缺陷，当语句不是服务器端预处理语句时，无法对来自prepareStatement()的语句进行高速缓冲处理。

-使用ResultSet.get(..., cal)和PreparedStatement.set(...., cal)时，如果客户端和服务器均位于GMT（格林威治标准时间）时区，选择进行时间调整的正确方向。

-增加了“dontTrackOpenResources”选项（默认为“假”以便兼容JDBC），对于具有不良特性的应用程序（例如应关闭语句时，却未关闭语句的应用程序），它有助于改善内存的使用。

-更正了BUG#8428，ResultSet.getString()不能保持存储在服务器上的格式，仅当将“noDatetimeStringSync”属性设置为“真”（默认为“假”）时，才能允许缺陷更正。

-更正了使用“usage advisor”而且结果集已关闭时ResultSet.realClose()中的NPE。

-更正了BUG#8487，不创建流式结果集的PreparedStatements。

-不将NULL传给给ResultSet.getNativeConvertToString()中的String.valueOf()，原因在于String.valueOf()会对其进行字符串处理（即返回“null”）,对于所述方法，这是错误的。

-更正了BUG#8484，当需要舍入操作以设定标度时，ResultSet.getBigDecimal()抛出异常。如果非舍入性BigDecimal.setScale()失败，驱动程序现在将选择“半向上”舍入模式。

-增加了“useLocalSessionState”配置属性，将其设置为“真”时，JDBC驱动程序将认为应用程序行为良好，并会使用java.sql.Connection提供的方法仅设置autocommit和事务隔离级别，因此，能够在大多数情况下处理这些值，而不引发数据库服务器循环。

-为连接池实施实例的语句增加了enableStreamingResults()，可检查Statement.setFetchSize()是否存在兼容规范的值。调用“Statement.setFetchSize(>=0)”可禁止关于该语句的流式结果。

-增加了对MySQL-5.0.3中BIT类型的支持。驱动程序会将BIT(1-8)当作JDBC的BIT类型对待（映射至java.lang.Boolean），这是因为当声明了“< 9 bits”)时，服务器目前不会发送能确定比特字段（bitfield）大小的足够信息。BIT(>9)将被当作VARBINARY对待，并当调用getObject()时返回byte[]。

12-23-04：版本3.1.6-stable

-更正了SocketInputStream.read()的挂起问题，当驱动程序必须直接截取结果集，而不是跟踪结果集末尾的“LIMIT n”时，与Statement.setMaxRows()和多个结果集一起使用时SocketInputStream.read()挂起。

-更正了BUG#7026，DBMD.getProcedures()不考虑catalog参数。

12-02-04：版本3.1.5-gamma

-更正了字符串常数和动态字符串之间的比较事宜，或是toUpperCase()d或是toLowerCase()d，以使用Locale.ENGLISH，将其用作英语的覆盖规则。此外，还使用StringUtils.indexOfIgnoreCase()替代了.toUpperCase().indexOf()，以避免创建很短的过渡性字符串实例。

-更正了BUG#5235，服务器端预处理语句不考虑“zeroDateTimeBehavior”属性，而且当使用ResultSet.getObject()时会导致类抛弃异常，这是因为总返回全0字符串。

-更正了批更新和服务器预处理语句有关的问题，与以前的集合相比，如果在给定的批参数集合中发生了类型变更，服务器预处理语句不会检测这类变化，从而导致服务器返回错误“Wrong arguments to mysql_stmt_execute()”。

-解决了当时间戳的字符串表示包含后缀“.”但其后没有数字时的情况。

-更正了BUG#5706，对ResultSet.getNativeString()中以前存在字符串实例的低效检测。

-不抛出针对Connection.releaseSavepoint()的异常。

-解码来自ServerPreparedStatements的日期时，默认情况下使用按会话进行的日历实例（通过设置“dynamicCalendars=true”，设置为早期的性能较低的方式）。

-增加了实验性配置属性“dontUnpackBinaryResults”，它延迟了解包二进制结果集取值的功能，直至请求了它们为止（默认情况下设为“假”）。对于某些usecase/jvm组合，它对无用信息收集器更友好。

-更正了BUG#5729，对来自服务器端预处理语句结果集的UNSIGNED BIGINT，未能正确解包。

-更正了BUG#6225，ServerSidePreparedStatement，分配了不必要的具有短生存时间的对象。

-删除了ResultSet构造函数中有害的新Throwable()，原因在于不良合并（导致任何所创建结果集永远不会使用的新对象），分析BUG#6359时发现。

-更正了在EscapeProcessor.escapeSQL()中过早创建StringBuffer的问题，而且在不需要转义功能时仍返回字符串（防止不必要的对象分配）。分析BUG#6359时发现。

-为可更新结果集中的键比较使用“null-safe-equals”。

-更正了BUG#6537，如果需要0填充，作用在Decimal上的SUM()以及服务器端预处理语句均将忽略标度（由于服务器将其转换为DOUBLE，0填充将结束，将其转换为BigDecimal类型的字符串进行解析时，丢失所有的填充“0”）。

-创建DBMD查询时使用DatabaseMetaData.getIdentifierQuoteString()。

-如果在服务器上LOAD DATA LOCAL INFILE小于“max_allowed_packet”，使用1MB的信息包发送文件。

-更正了BUG#6399，对于多字节字符集，ResultSetMetaData.getColumnDisplaySize()返回错误值。

-通过“autoDeserialize”属性（默认为“假”），能够对保存在BLOB中的java.lang.Objects的自动串并转换功能进行配置。

-修改了Field.isOpaqueBinary()以检测“CHAR(n) CHARACTER SET BINARY”，从而能够支持用于ResultSet.getObject()的固定长度二进制字段。

-使用我们自己的缓冲输入流实施方式来处理java.io.BufferedInputStream的封闭行为。使用“useReadAheadInput=false”可禁止它。

-更正了BUG#6348，当给定主机名的某一地址是IPV6时无法连接到服务器（服务器尚未与IPV6捆绑） 驱动程序现在能够查找给定主机的所有IP地址，并停在接受socket.connect()的第1个地址上。

09-04-04：版本3.1.4-beta

-更正了BUG#4510，Connector/j 3.1.3 beta不能正确处理整数（由为了支持Buffer.readInt() -> Buffer.readShort()中的无符号读取操作所作的更改而导致）。

-在DatabaseMetaData.getTables()和getTableTypes()中增加了对VIEW的支持，该类特性现已在MySQL服务器5.0.x版中提供。

-更正了BUG#4642，解包字段元数据时，ServerPreparedStatement.execute*()有时会抛出ArrayIndexOutOfBoundsException。

-优化了整数解析功能，通过“useFastIntParsing=false”属性，允许使用JDK类较慢的解析功能。

-增加了“useOnlyServerErrorMessages”属性，它会使服务器生成的异常中的消息文本仅包含由服务器发送的文本（与SQLState“标准”介绍中给出的相反，此时文本后面还包括服务器的错误消息）。默认情况下，该属性被设置为“真”。

-更正了BUG#4689，如果前面的Null已被返回，对于原语，ResultSet.wasNull()不工作。

-如果“enablePacketDebug=true”，跟踪信息包的序列号，如果收到的信息包次序混乱，抛出异常。

-更正了BUG#4482，使用预处理语句时，对于字符串，ResultSet.getObject()返回错误类型。

-两次调用MysqlPooledConnection.close()（虽然也是应用程序错误）会导致NPE。已更正。

-更正了BUG#5012，处理DECIMAL类型的返回值的ServerPreparedStatements不工作。

-更正了BUG#5032，对于来自4.1.x版预处理语句的伪位类型，ResultSet.getObject()不返回布尔类型（在针对伪位类型的getObject()中使用二进制编码的结果集模糊测试时，它提供了避免额外类型转换的快捷方式）。

-现在能够在“LOAD DATA LOCAL INFILE”语句中使用URL，而且驱动程序将使用Java的内置处理程序来检索数据，并将其发送到服务器。默认情况下不允许该特性，要想使用它，必须将“allowUrlInLocalInfile”连接属性设置为“真”。

-对于在ResultSet.get*()的数值上执行的截取操作，驱动程序将更加严格，而且当检测到截取操作时将抛出SQLException。将“jdbcCompliantTruncation”设置为“假”，可以禁止该功能（在默认情况下允许该功能，这是因为该功能是兼容JDBC所需的）。

-从结果集读取“全零”日期时间值时，增加了三种处理它们的方法，“exception”，（默认值），用代码为“S1009”的SQLState抛出SQLException；“convertToNull”，返回NULL而不是数据；以及“round”，对日期进行舍入处理，使之成为最接近的值，即“0001-01-01”。

-更正了ServerPreparedStatement，使之能够“以脱线”方式读取预处理语句元数据，尽管在目前它是取代在任何时侯均不工作的MysqlIO.clearInputStream()的占位符，之所以如此，是因为目前尚未从服务器读取数据。通过抛出ArrayIndexOutOfBoundExceptions的erverPreparedStatements，它修正了用户遇到的偶发性错误。

-加载资源捆包时使用com.mysql.jdbc.Message的类加载器，应能更正调用器的类加载器不能确定资源捆包位置时出现的偶发性错误。

07-07-04：版本3.1.3-beta

-对CallableStatements的输出参数名进行Mangle处理，使得在与用户变量名一起使用时，这类名称不会崩溃。

-增加了对CallableStatements中INPUT参数的支持。

-更正了BUG#4119，为服务器端预处理语句发送的null比特掩码不正确。

-默认情况下使用SQL的标准SQL状态，除非将“useSqlStateCodes”属性设置为“假”。

-增加了信息包调试代码（请参见“enablePacketDebug”属性文档）。

-为MySQL错误编号增加了常量（可公共访问，请参见com.mysql.jdbc.MysqlErrorNumbers），并增加了生成映射的能力，能够将厂家代码映射为驱动程序使用的SQLStates（出于文档编制目的）。

-使更多消息更为具体（正在努力）。

-更正了BUG#4311，在检索具有预处理语句和二进制协议特性的mediumint列时发生错误。

-当“useTimezone=true”时，在MySQL-4.1.3中支持新的时区变量。

-支持无符号数值作为预处理语句返回的类型。对于“bigint unsigned”类型，这也会导致ResultSet.getObject()中的内容发生变化，“bigint unsigned”类型用于返回BigDecimal实例，现在它将返回java.lang.BigInteger实例。

06-09-04：版本3.1.2-alpha

更正了为参数（如char(), varchar()）指定大小时存储程序参数的解析信息。

-通过“cacheCallableStmts”属性，允许对可调用语句进行高速缓冲处理。

-更正了下述问题：未为存储程序指定输出参数时，发出虚假查询来检索输出参数，从而导致服务器语法错误。

-更正了在CallableStatement.setOutputParameters()中没有任何参数会导致NullPointerException的问题。

-删除了MysqlIO.changeUser()中的异常捆包。

-更正了关于发送大型查询分离信息包方面的问题，也允许发送大信息包的nio功能。

-为ServerPreparedStatement增加了.toString()功能，如果你正在调试作为预处理语句（显示为服务器应处理的SQL）的查询，它应有所帮助。

-增加了“gatherPerformanceMetrics”属性，以及用于控制在何时/何处记录这类信息的多种属性（更多信息请参见相关文档）。

-调用.close()时，ServerPreparedStatements实际上不能取消对服务器端资源的分配。

-增加了“logSlowQueries”属性，以及“slowQueriesThresholdMillis”属性，以控制应在何时将查询视为“缓慢”。

-相对于registerOutParameter()中隐含的顺序，正确地将输出参数映射到prepareCall()中给定的位置，；更正了BUG#3146。

-对于版本等于高于4.1.0的服务器，能正确地检测字符集。

-整理了服务器属性的检测功能。

-对于版本等于高于4.1.2的服务器，支持用于参数元数据的占位符。

-更正了BUG#3539，getProcedures()未返回结果集中的任何进程。

-更正了BUG#3540，getProcedureColumns()不能与程序名的通配符一起使用。

-更正了BUG#3520，DBMD.getSQLStateType()返回不正确的值。

-增加了“connectionCollation”属性，如果给定字符集的默认校对不恰当，驱动程序将在连接初始过程中发出“set collation_connection=...”查询。

-更正了在MySQL-5.0.0上运行时的DatabaseMetaData.getProcedures()问题（在5.0.1和5.0.0间，“show procedure status”的输出有所不同）。

-更正了BUG#3804，getWarnings()返回SQLWarning而不是DataTruncation。

-对于版本为5.0.0或5.0.1的服务器，不要启用服务器端预处理语句，这是因为，它们与驱动程序使用的“4.1.2+”风格不兼容（驱动程序预计返回的信息并不存在，因而挂起）。

02-14-04：版本3.1.1-alpha

-更正了与不使用客户端预处理语句的UpdatableResultSets有关的缺陷。

-当MySQL不提供字符集并将JVM设置为多字节编码时，更正了将字节转换为ASCII时的字符编码事宜（通常会影响数值检索）。

-将服务器预处理语句的“未知”数据类型解包为字符串。

-为服务器预处理语句实现了长数据类型（Blob、Clob、InputStreams、Readers）。

-为MySQL-4.1和更高版本实现了Statement.getWarnings()（使用“SHOW WARNINGS”）。

-默认结果集类型更改为TYPE_FORWARD_ONLY（兼容JDBC）。

- 结果集类型和并发性的集中设置。

-再次确定了连接属性的设置方式和显示为DriverPropertyInfo的方式，以及Connection和DataSource属性。

-支持NIO。在支持NIO的平台上使用“useNIO=true”。

-支持SAVEPOINTs（MySQL >= 4.0.14或4.1.1）。

-支持“mysql_change_user()...”，请参见“com.mysql.jdbc.Connection”中的changeUser()方法。

-减少了平均查询中调用的方法数目，使之更有效率。

-自动再连接时重新处理预处理语句。任何遇到的错误均被延迟，直至首次尝试再执行经过重新处理的语句为止。

-按照JDBC规范，确保在预处理语句上执行查询前给出的警告是明确的（目前，我们支持警告功能）。

-在ConnectionProperties中，支持“旧的”profileSql大写特性。该属性已受到冷落，应尽量使用“profileSQL”。

-优化了Buffer.readLenByteArray()，以便当长度为0时返回共享的空字节数组。

-在对.execute*()的多次调用之间，允许保留PreparedStatement.setBlob()的内容。

-处理了EscapeProcessor中的0长度令牌（由可调用语句转义语法导致）。

-在UpdatableResultSet中的删除／更新／插入行操作上，检查关闭的连接。

-更正了检查UpdatableResultSet中的所有主键时对表别名的支持事宜。

-删除了useFastDates连接属性。

-对来自JNDI Refs的数据源属性进行了正确的初始化，包括明确指定的URL。

-对于版本为5.0.0或更高的MySQL，DatabaseMetaData现在能通报supportsStoredProcedures()。

-更正了Connection.prepareCall()中的堆栈溢出问题（不良合并）。

-对低于1.4版的JDK，更正了对DateTimeValue中Calendar.getTimeInMillis()的IllegalAccessError（非法访问错误）。

-更正了BUG#1673，对于非“%”列名，DatabaseMetaData.getColumns()未返回正确的列顺序信息。

-合并了数据类型映射的更正事项，从MySQL类型“FLOAT”到java.sql.Types.REAL（自3.0版起）的数据类型映射。

-检测用于RSMD.isCaseSensitive()的列校对。

-更正了与发送大于16M查询有关的问题。

-为CallableStatement增加了命名和索引式输入／输出参数支持。

MySQL-5.0.x或更高版本。

-更正了ServerPreparedStatement.setTimestamp()中的NullPointerException，以及ServerPreparedStatement.setTimestamp()、setDate()中的年月差异。

-为build.xml中的一致性和递归/单元测试增加了拥有多个数据库/JVM目标的能力。

-更正了访问ServerPreparedStatements和其结果集中某些日期时间功能时的NPE和不良年月转换问题。

-显示关闭连接的位置和原因（用于帮助调试）。

-实现了CommunicationsException，它能尝试判断与服务器失去通信的原因，并能在调用.getMessage()时显示可能的原因。

-更正了BUG#2359，二进制编码结果集中数值类型的NULL值会导致NullPointerExceptions。

-实现了Connection.prepareCall()，DatabaseMetaData.getProcedures()，以及getProcedureColumns()。

-调用clearParameters()时，通过发送COM_RESET_STMT至服务器，复位了ServerPreparedStatement中的“long binary”参数。

-自3.0版起，合并了预处理语句高速缓冲和.getMetaData()支持。

-解包来自服务器端预处理语句的结果时，在某些情况下，对于TimeUtil.fastDate/TimeCreate()的年份，会出现off-by-1900错误，更正了该错误。

-更正了BUG#2502，getTables()中的字符集转换事宜。

-实现了由语句或存储程序返回的多个结果集。

-更正了BUG#2606，服务器端预处理语句未正确返回数据类型“YEAR”。

-允许对来自服务器端预处理语句的结果集进行流处理。

-更正了BUG#2623，使用滚动结果集和服务器端预处理语句时出现类舍弃异常。

-自3.0版起，合并了无缓冲输入代码。

-更正了不能通过存取器恰当显示的ConnectionProperties，并整理了ConnectionProperties代码。

-更正了BUG#2671，在服务器端预处理语句中，在任何情况下均不能正确编码NULL字段。

-更正了将数字写入缓冲以发送预处理语句执行请求时出现的罕见缓冲区下溢问题。

-对于交付的驱动程序，使用了文档的DocBook版。

02-18-03：版本3.1.0-alpha

-增加了“requireSSL”属性。

-增加了“useServerPrepStmts”属性（默认为“假”）。当服务器版本支持并将该属性设置为“真”时（4.1和更高版本），驱动程序将使用服务器端预处理语句。目前其默认设置为“假”，除非所有的捆绑／获取功能均已实施。目前，仅为4.1版的服务器端预处理语句实现了DML预处理语句。

-跟踪打开的语句，并在调用Connection.close()时关闭所有打开的语句（JDBC兼容）。

06-23-05：版本3.0.17-ga

-更正了BUG#5874，当useTimeZone='true'而且服务器时区不同于客户端时区时，Timestamp/Time转换出现方向错误。

-更正了BUG#7081，DatabaseMetaData.getIndexInfo()忽略“唯一”参数。

-支持新的协议类型“MYSQL_TYPE_VARCHAR”。

-增加了“useOldUTF8Behavoior”配置属性，连接到MySQL-4.1或更高版本且字符编码为“utf-8”时，该属性能使JDBC驱动程序的行为方式类似于在MySQL-4.0.x和更早版本下的行为方式。

-更正了BUG#7316，调用getConnection()时，从连接池创建的语句返回实际连接而不是逻辑连接。

-更正了BUG#7033，在静态SQL字符串中，PreparedStatements不能正确编码Big5（以及其他多字节）字符集。

-更正了BUG#6966，启动了故障切换的连接（由于主连接失败）从不再次尝试主连接。

-更正了BUG#7061，PreparedStatement.fixDecimalExponent()增加额外“+”，使得MySQL服务器无法解析数值。

-更正了BUG#7686，Timestamp关键字列数据需要分离的“_binary”，用于UpdatableResultSet.refreshRow()。

-反向移植了来自Connector/J 3.1的SQLState代码映射，使用连接属性“useSqlStateCodes=true”可启用它，在本版本中默认为“假”，以免破坏传统应用程序（对于Connector/J 3.1，默认为“真”）。

-更正了BUG#7061，PreparedStatement.fixDecimalExponent()增加额外“+”，使得MySQL服务器无法解析数值。

-转义序列{fn convert(..., type)}现在支持由SQL预先设定的ODBC类型。

-更正了configureClientCharset()中的重复代码，该类代码将阻止useOldUTF8Behavior=true的恰当运行。

-通过更正行数计数器的环绕式处理程序，能正确处理大于20亿行的流式结果集。

-更正了BUG#7607，MS932、SHIFT_JIS和Windows_31J不接受针对sjis的别名。

-更正了BUG#6549（更正#7607的同时），为sjis增加了CP943别名。

-更正了BUG#8064，与预处理语句一起使用多字节字符集时，要求对二进制数据进行十六进制转义处理。

-更正了BUG#8812，DBMD.getIndexInfo()的NON_UNIQUE列返回倒置值。

-解决了服务器BUG#9098，无法将DATE/TIME/TIMESTAMP/TIMESTAMP列CURRENT_*的默认值与“字符串”值区别开，因此，插入默认值时，UpdatableResultSet.moveToInsertRow()会生成不良的SQL。

-更正了BUG#8629，将“EUCKR”作为“SET NAMES euc_kr”发送，MySQL-4.1以及更高版本不能理解该特性。

-根据服务器的版本，DatabaseMetaData.supportsSelectForUpdate()返回正确值。

-对于含别名Windows-31J、CP934、MS932的双字节字符集，对于PreparedStatement.setBytes()，采用十六进制转义特性。

-增加了对“EUC_JP_Solaris”字符编码的支持，映射到“eucjpms”的MySQL编码（从3.1版开始的反向移植）。它只能在支持eucjpms的服务器山工作，也就是说5.0.3或更高版本。

11-15-04：版本3.0.16-ga

-连接至MySQL-4.1或更高版本且再次使用连接池和／或Connection.changeUser()时，将再次发出字符集配置命令。

-更正了ResultSetMetaData.isReadOnly()，以便在连接至MySQL-4.1或更高版本时，能根据“原始的”表名和列名，检测不可写的列。

-更正了BUG#5664，当ResultSet.updateByte()位于插入行上时抛出ArrayOutOfBoundsException。

-更正了DatabaseMetaData.getTypes()，对于NUMERIC类型，它返回不正确的（非负）标度。

-更正了BUG#6198，Buffer.readString(string)中的“off-by-one”问题。

-通过“tinyInt1isBit”属性，使得能够对TINYINT(1) -> BIT/Boolean转换进行配置（为了与JDBC兼容，默认为“真”）。

-如果服务器版本大于等于4.1.1，在连接建立过程中仅设置“character_set_results”。

-更正了回归问题，其中，useUnbufferedInput默认为“假”。

-更正了BUG#6231，ResultSet.getTimestamp()作用在具有TIME的列上时失败。

09-04-04：版本3.0.15-production

-更正了BUG#4010，对于GBK，StringUtils.escapeEasternUnicodeByteStream仍被破坏。

-更正了BUG#4334，对于autoReconnect的故障切换，对任何主机均不使用端口“#”，而且不重试所有主机。（警告：需要更改SocketFactory connect()方法的特征，它目前是公共套接字连接

（String host，int portNumber，Properties props）。也能次，必须更改任何第三方套接字实施方式以支持该特征。

-当它们已被关闭并被返回到连接池时，由MysqlConnectionPoolDataSource创建的逻辑连接将发出rollback()。如果你的应用服务器/连接池已帮助你完成了该任务，可以将“rollbackOnPooledClose”属性设置为“假”以避免额外的rollback()开销。

-删除了结果集中对checkRowPos()的多余调用。

-更正了BUG#4742，在DBMD.getTypeInfo()中，“DOUBLE”映射了两次。

-增加了FLOSS许可豁免。

-更正了BUG#4808，在PooledConnection上调用两次.close()将导致NPE。

-更正了BUG#4138和BUG#4860，对于无符号列，DBMD.getColumns()返回错误的JDBC类型。这也会影响对RSMD.getColumnType()和RSMD.getColumnTypeNames()方法中所有数值类型的类型映射，以确保DBMD.getColumns()的类似类型与RSMD.getColumnType()和RSMD.getColumnTypeNames()返回的类型匹配。

-分发版命名方案中的“Production”-“GA”。

-更正了BUG#4880，对于非数值类型，RSMD.getPrecision()返回0（对于非二进制类型，应返回字符的最大长度，对于二进制类型，应返回最大字节长度）。根据服务器发回的长度（在网络协议层，服务器不区分TINYBLOB、BLOB、MEDIUMBLOB或LONGBLOB），它还更正了针对BLOB类型的RSMD.getColumnType()和RSMD.getColumnTypeName()映射。

-更正了BUG#5022，结果集应在“.close()”中释放Field[]实例。

-更正了BUG#5069，如果结果集已关闭，ResultSet.getMetaData()不应返回未正确初始化的元数据，而是应抛出SQLException。此外，在对实例级别的字段（.close()过程中取消的字段）进行操作之前，通过调用checkClosed()，它还更正了getRow()、getWarnings()和遍历方法。

-从4.1.x版服务器开始，能解析新的时区变量。

-与MySQL-4.1.x或更高版本连接时，为PreparedStatement.setBytes()和set*Stream()使用“_binary”引介词，以防止在字符集转换过程中出现错误判断。

05-28-04：版本3.0.14-production

-更正了URL解析错误。

05-27-04：版本3.0.13-production

-更正了BUG#3848，无服务器名称时，不能使用MySQLDatasource。

-更正了BUG#3920，使用MysqlConnectionPoolDataSource时出现“No Database Selected”（未选择数据库）。

-更正了BUG#3873，对于批插入，PreparedStatement.getGeneratedKeys()方法仅返回1个结果。

05-18-04：版本3.0.12-production

- 为TYPE_NAME列中的DatabaseMetaData.getColumns()输出增加了无符号属性。
- 增加了“failOverReadOnly”属性，允许最终用户配置出现故障切换时的连接状态（只读／可写）。
- 自3.1开始，反向移植了“change user”和“reset server state”功能，允许MysqlConnectionPoolDataSource的客户端复位连接池上getConnection()的服务器状态。
- 使用MySQL-4.1或更高版本时，不对SJIS/GBK/BIG5进行转义处理。
- 允许MysqlDataSource和MysqlConnectionPool数据源使用“url”参数，以便能够从内部应用服务器传递其他属性。
- 将复制键和外键错误映射到“23000”的SQLState。
- 自3.1开始，反向移植了文档编制工具。
- 返回用于结果集的创建语句，该结果集由getGeneratedKeys()创建（BUG#2957）。
- 允许作为参数将java.util.Date发送到PreparedStatement.setObject()，将其转换为Timestamp类型以保持完整精度（BUG#3103）。
- 使用setBytes()和/或setBinary/CharacterStream()时，不截取BLOB/CLOB（BUG#2670）。
- 连接时，在使用“SHOW COLLATION”的MySQL-4.1.0和更高版本上，为字段级字符集动态配置字符集映射。
- 将“binary”字符集映射到“US-ASCII”，以支持4.1.2和更高版本服务器的DATETIME字符集识别功能。
- 在初始化过程中使用“SET character_set_results”，允许将结果集的任何字符集返回到驱动程序。
- 在>= 4.1.0的MySQL上发出“SET NAMES”之前，在连接至编码查询的过程中，使用返回的charsetnr。
- 为ResultSetMetaData（getColumnCharacterEncoding()和getColumnCharacterSet()）增加了Helper（助手）方法，允许最终用户查看驱动程序认为应在列上使用的字符集。
- 在>= 4.1.0的MySQL上仅设置character_set_results。
- 更正了BUG#3511，StringUtils.escapeSJISByteStream()不能正确处理所有的东方双字节字符集。
- 将StringUtils.escapeSJISByteStream()重新命名为更贴切的escapeEasternUnicodeByteStream()。
- 更正了BUG#3554，在URL中未指定数据库将导致MalformedURL exception。

-如果使用了characterEncoding属性，自动将MySQL编码名称转换为Java编码名称。

-增加了在某些JVM上能识别的编码名称，以更正错误地将其逆向映射为MySQL编码名称的问题。

-为所有单元测试使用junit.textui.TestRunner（允许在Ant或Eclipse以外的命令行上运行它们）。

-更正了BUG#3557，UpdatableResultSet不能获取moveToInsertRow()的默认值。

-更正了BUG#3570，不一致的列类型通报。服务器仍不能正确返回*BLOBs *TEXT的所有类型，因此驱动程序也不能正确返回它们。

-更正了BUG#3520，DBMD.getSQLStateType()返回不正确的值。

-更正了PreparedStatement.setString()和东方字符编码中的递归问题。

- 增加了对StringRegressionTest 4.1-unicode的识别。

02-19-04：版本3.0.11-stable

-通过“characterEncoding”属性将编码方式强制设为“utf8”或“utf-8”时，触发“SET NAMES utf8”。以前，只能用Java风格的“utf-8”编码名称才能触发该操作。

-AutoReconnect时间的增长速度快于指数速度（BUG#2447）。

-更正了故障切换总跳至列表中最后1个主机的问题（BUG#2578）。

-增加了“useUnbufferedInput”参数，它也是目前的默认参数（因JVM事宜，http://developer.java.sun.com/developer/bugParade/bugs/4401235.html）。

-检测服务器上lower_case_table_names的“on/off”，或“1”、“2”、“3”形式。

-为ResultSetMetaData.getColumnClassName()的TINYINT和SMALLINT类型返回“java.lang.Integer”（更正了BUG#2852）。

-为ResultSetMetaData.getColumnClassName()的FLOAT类型返回“java.lang.Double”（更正了BUG#2855）。

-为ResultSetMetaData.getColumnClassName()的BINARY、VARBINARY和LONGVARBINARY类型返回“[B”而不是“java.lang.Object”（兼容JDBC）。

-在由ConnectionPoolDataSource创建的所有实例上发出连接事件。

01-13-04：版本3.0.10-stable

- PreparedStatement “ ” ID BUG#1511

在 解析中，当位于 字符串 内时，不对引用的 进行计数（更正了 ）。

-关于PacketTooLargeException的“Friendlier”异常消息（BUG#1534）。

-从3.1版开始，反向移植了对checkUpdatability()方法中别名表和UpdatableResultSets的补丁。

-更正了使用Statement.setMaxRows()时出现的ArrayIndexOutOfBounds异常（BUG#1695）。

-更正了BUG#1576，处理未正确读取的大BLOB和分离信息包。

-更正了Statement.getGeneratedKeys()和REPLACE语句的递归问题。

-更正了BUG#1630，如果结果集是不可更新的，对ResultSet.updateFoo()的后续调用将导致NPE。

-确定了4.1.1风格的auth，无密码。

-更正了BUG#1731，外键列的顺序与DatabaseMetaData.getImported/Exported/CrossReference()不一致。

-更正了BUG#1775，DatabaseMetaData.getSystemFunction()返回错误函数“VResultsSion”。

-更正了BUG#1592，未正确检查交叉数据库可更新结果集的可更新性。

-对于MySQL LONGTEXT类型，DatabaseMetaData.getColumns()应返回Types.LONGVARCHAR。

-作用在TINYINT和SMALLINT列上的ResultSet.getObject()应返回Java类型“Integer”（BUG#1913）。

-增加了“alwaysClearStream”连接属性，它会使驱动程序在每次查询前清空输入流上任何余留的数据。

-增加了更具描述性的错误消息“Server Configuration Denies Access to DataSource”（服务器配置拒绝对数据源的访问），并能从服务器上检索消息。

-如果已发生变化，Autoreconnect代码在红心连接时不设置catalog。

-实现了ResultSet.updateClob()。

-对于CHAR/VARCHAR列，ResultSetMetaData.isCaseSensitive()返回错误值。

-更正了BUG#1933，不尊重连接属性“maxRows”。

-更正了BUG#1925，在DBMD.extractForeignKeyFromCreateTable()中，创建语句的次数太多。

-更正了BUG#1914，支持转义序列{fn convert ... }

-更正了BUG#1958，当参数编号等于参数数目＋1时，ArrayIndexOutOfBounds。

-更正了BUG#2006，当SELECT查询中有多个重复列名时，ResultSet.findColumn()应使用第1个匹配的列名（兼容JDBC）。

-从PreparedStatement.setTimestamp()中删除了静态同步瓶颈。

-从SingleByteCharsetConverter的实例方法中删除了静态同步瓶颈。

-允许通过“cachePrepStmts”、“prepStmtCacheSize”和“prepStmtCacheSqlLimit”属性，对预处理语句的解析步骤进行高速缓冲处理（默认情况下禁止）。

-加快了PreparedStatements的解析操作，只要可能，尽量采用一次性方式。

-更正了在小应用程序中使用时的安全异常问题（小应用程序不能读取LOAD DATA LOCAL INFILE所需的系统属性“file.encoding”）。

-为SQLStates使用常数。

-连接至MySQL-4.1.0或更高版本时，将字符集“ko18_ru”映射到“ko18r”。

-确保Buffer.writeString()保存在“\0”之外的空间内。

-更正了“connect w/ JDK-1.4.0”上的异常“未知字符集danish”。

-更正了SQLError中的映射功能，用“41000”SQLStates通报死锁状态。

-“maxRows”属性会影响内部语句，因此，应检查为驱动程序创建的所有内部语句，并在不是内部语句的情况下将其设为0。

10-07-03：版本3.0.9-stable

-ResultSet和PreparedStatement中更快的日期处理代码（不再使用用来同步静态日历的Date方法）。

-更正了对Buffer.readString()中缓冲区末端的测试。

-更正了ResultSet.previous()行为方式，当位于结果集的第1行上时，将当前位置移到结果集之前（bugs.mysql.com BUG#496）。

-更正了当已使用setMaxRows()而且在查询中包含LIMIT子句时发出伪查询的语句和PreparedStatement。

- BUG#661 refreshRow

更正了 ，当主键值包含需要转义的值时 不工作（以双倍转义结束）。

-提取DatabaseMetaData中的外键信息时，支持InnoDB约束名，BUG#517和BUG#664（由Parwinder Sekhon提出）。

-从3.1版开始，反向移植了4.1协议变化（服务器端SQL状态，新字段信息，较大的客户端能力标志，与数据库的连接等）。

-更正了UpdatableResultSet，作用在插入行上时，为getXXX()返回值（BUG#675）。

-调用moveToInsertRow()时，能够用默认的列值加载UpdatableResultSet中的insertRow（BUG#688）。

-对于指定为NULL的默认值，DatabaseMetaData.getColumns()未返回NULL。

-将默认的语句类型／并发行更改为TYPE_FORWARD_ONLY和CONCUR_READ_ONLY（兼容规范）。

-如果MySQL不支持，不要尝试或复位再连接上的隔离级别。

-不将SQLExceptions封在RowDataDynamic内。

-如果useTimezone==true，不更改时间戳TZ两次（BUG#774）。

-更正了大的分离信息包处理中存在的回归问题（BUG#848）。

-更好地诊断“流式”结果集异常中的错误消息。

-在空结果集上抛出关于ResultSet.getXXX()的异常（在某些情况未被俘获）。

-不隐藏关于在I/O层中抛出异常的消息。

-关闭连接池时或在具有已打开连接的PooledConnection.getConnection()上，不触发连接关闭时间（BUG#884）。

-截短+/- INF（最小和最大代表值，对于MySQL中的类型），以及NaN（至0，对于setDouble/setFloat()），当服务器不支持+/- INF或NaN时，就语句发出警告。

-更正了BUG#879，当字符集为SJIS或GBK时，以及“\”出现在非转义输入中时对“\”的双转义处理。

-清空“流式”结果集未使用行的输入流时，每100行均有当前线程yield()，以免独占CPU时间。

-更正了BUG#1099，DatabaseMetaData.getColumns()弄不清字符列中的关键字“set”。

-更正了与Statement.setMaxRows()相关的死锁事宜。

-更正了CLOB.truncate()，BUG#1130。

-优化了CLOB.setChracterStream(), BUG#1131。

-使databaseName、portNumber以及serverName成为MysqlDataSourceFactory的可选参数（BUG#1246）。

-修订了BUG#1247，导致字符127损坏的ResultSet.get/setString。

-从3.1版开始，反向移植了针对4.11和更高版本的auth.更改。

-增加了com.mysql.jdbc.util.BaseBugReport，以帮助创建缺陷报告的测试范例。

-通过将“clobberStreamingResults”属性设置为“真”（默认为“假”），为“clobber”流式结果增加了属性。这会使“流式”结果集被自动关闭，如果在所有数据尚未从服务器中读取完之前，执行了另一查询，正在从服务器流出的任何未完成数据均将被舍弃。

05-23-03：版本3.0.8-stable

-允许在Driver.getPropertyInfo()中使用伪URL。

-与Statement.getGeneratedKeys()一起使用多值INSERTS时，返回所生成键的列表。

-与文件名和“LOAD DATA [LOCAL] INFILE”一起使用JVM字符集。

-更正了与Connection.cleanup()有关的无限循环。

-将Ant目标“compile-core”更改为“compile-driver”，并使测试套件编译成为单独的目标。

-更正了未获得关于Statement.executeUpdate()的集合的结果集，在某些情况下，它会影响getGeneratedKeys()和getUpdateCount()。

-字符串中的Unicode字符0xFFFF会导致驱动程序抛出ArrayOutOfBoundsException（Bug #378）。

-使用“REPLACE”语句时返回所生成键的正确数目。

-更正了在某些情况下检测服务器字符集的问题。

-更正了使用极大信息包时的行数据解码错误。

-优化了行数据解码。

-在已关闭预处理语句上执行操作使抛出异常。

-更正了SJIS编码缺陷，感谢Naoto Sato。

-优化了EscapeProcessor的使用。

-允许对Statement.close()的多次调用。

04-08-03：版本3.0.7-stable

-更正了调用错误事件类型的MysqlPooledConnection.close()。

-更正了PreparedStatement.setClob()中的StringIndexOutOfBoundsException。

- 4.1版列元数据更正。

-从Driver.connect()和Driver.acceptsUrl()中删除了同步功能。

-事务过程中的IOExceptions现在会导致关闭连接。

-更正了ResultSetMetaData.getColumnTypeName()中丢失的“YEAR”类型转换。

-对于DBMD.getPrimaryKeys()，不提取以“pri”作为主键起始的索引。

-试图在强制关闭的连接上执行操作时抛出SQLExceptions（即，通信链路失败时）。

-现在，可以使用Connection.setProfileSql(boolean)来打开或关闭仿型功能。

更正了与数据库元数据有关的字符集事宜（字符集不能正确获取集合）。

-与MySQL-4.1或更高版本连接时，现在能够为别名表/列创建可更新结果集。

-更正了文件大于“max_allowed_packet”时的“LOAD DATA LOCAL INFILE”缺陷。

-对于GBK和Big5字符集，更正了0x5c ('\')字符的转义功能。

-更正了基础字段为DATE类型时的ResultSet.getTimestamp()问题。

-保证了来自alignPacketSize()的信息包大小不超过MAX_ALLOWED_PACKET（JVM缺陷）。

-autoReconnecting（自动再连接）时不复位Connection.isReadOnly()。

02-18-03：版本3.0.6-stable

-更正了ResultSetMetaData，当catalog未知时返回""。更正了与Sun CachedRowSet有关的NullPointerExceptions。

-更正了DBMD.getTypeInfo()和DBMD.getColumns()为TEXT/BLOB类型中的精度返回不同值的问题。

-通过将“ignoreNonTxTables”设置为“真”，在回退期间（兼容性／可用性），允许忽略“non transactional tables”（非事务表）告警。

-更正了在初始连接上SQLExceptions耗尽的问题。

-更正了Statement.setMaxRows()，当其不需要时，停止发送“LIMIT”类型的查询（性能）。

-整理了语句查询/方法失配测试（即，不允许与.executeQuery()一起使用INSERT）。

-在ResultSet遍历方法中增加了更多检查，以获取何时处于关闭状态的信息。

-更正了ResultSetMetaData.isWritable()以返回正确值。

-为DBMD.nullsAreSortedAtStart增加了不同NULL分类行为的“窗口”（4.0.2至4.0.10，真，其他为“否”）。

-实现了Blob.setBytes()。仍需要将所得的Blob传回到可更新ResultSet（结果集）或PreparedStatement以保持变更，这是因为MySQL不支持“定位器”。

-从Connector/J 3.1反向移植了4.1字符集字段。

01-22-03：版本3.0.5-gamma

-更正了解包字段时对某些查询Buffer.fastSkipLenString()导致ArrayIndexOutOfBounds异常的问题。

-为Connection.getTypeMap()实现了空的TypeMap，以便某些第三方应用程序能与MySQL一起工作（IBM WebSphere 5.0连接池）。

-为DBMD.getColumns()增加了丢失的LONGTEXT类型。

-当MySQL版本支持时，Connection.getTransactionIsolation()能从数据库检索TX_ISOLATION，而不是实例变量。

-引用DatabaseMetaData.getColumns()、getPrimaryKeys()、getIndexInfo()、getBestRowIdentifier()中的表名。

- PreparedStatements setBinaryStream()

大幅度降低了 中 所需的内存。

-更正了ResultSet.isBeforeFirst()处理空结果集方面的问题。

-为外键元数据增加了“更新”选项。

01-06-03：版本3.0.4-gamma

-为Connection.setCatalog增加了对数据库名的引用ID。

-在PreparedStatement分析程序中增加了对引用ID的支持。

-对于setByte()，在PreparedStatements中，流线式字符转换和byte[]处理。

-通过与MysqlIO共享出站信息包，降低了PreparedStatements的内存占用。

-增加了“strictUpdates”属性，能够对可更新结果集的大量“正确性”检查进行控制。如果希望获得更快的可更新结果集，并使用主键通过作用在表上的SELECT创建了结果集，而且在查询中选择了所有的主键，应将其设置为“假”。

-增加了对4.0.8风格大信息包的支持。

-更正了PreparedStatement.executeBatch()参数重写的问题。

12-17-02：版本3.0.3-dev

-将SingleByteCharConverter中的charsToByte更改为非静态类型。

-更改了SingleByteCharConverter，以使用各转换器的空闲初始化特性。

-更正了Fields.java中的字符集处理功能。

-实现了Connection.nativeSQL()。

-更可靠的转义标志“-- recognize '--'”注释，并允许嵌套式转义序列（请参见testsuite.EscapeProcessingTest）。

-DBMD.getImported/ExportedKeys()现在能处理每个表上的多个外键。

-更正了对某些浮点类型ResultSetMetaData.getPrecision()返回错误值的问题。

-更正了ResultSetMetaData.getColumnTypeName()对TEXT类型返回BLOB，对BLOB类型返回TEXT的问题。

-对于4.1和更高版本服务器，更正了Buffer.isLastDataPacket()。

-增加了CLIENT_LONG_FLAG，以便能获得更多的列标志（isAutoIncrement()最重要）。

-由于上述原因，实现了ResultSetMetaData.isAutoIncrement()以使用Field.isAutoIncrement()。

-在DatabaseMetaData方法中比较表名且在服务器中允许时，优先考虑“lower_case_table_names”。

-一些MySQL-4.1协议支持（扩展字段信息）。

-使用非别名表/列名和数据库名，以完全限定UpdatableResultSet中的表和列（需要MySQL-4.1或更高版本）。

-允许用户通过“continueBatchOnError”属性（默认为“真”）更改Statement/PreparedStatement.executeBatch()的行为。

-在更多连接方法（createStatement、prepareStatement、setTransactionIsolation、setAutoCommit）中检查关闭的连接。

-更可靠地实现了可更新结果集。检查表的所有主键是否已被选择。

-如果在服务器的配置允许，“LOAD DATA LOCAL INFILE ...”现在能正常工作。不能使用“allowLoadLocalInfile”属性关闭它（请参见README）。

-在单字节字符集中，对于未知的字符转换，用“?”替代“\0”。

- NamedPipeSocketFactory现能正常工作（仅对Windows平台）。具体说明请参见README。

11-08-02：版本3.0.2-dev

-更正了可更新结果集和PreparedStatements不工作的问题。

-更正了ResultSet.setFetchDirection(FETCH_UNKNOWN)。

-更正了使用任意值并调用Statement.setFetchSize()时的问题。

-更正了ResultSet.getLong()中的不正确转换。

-实现了ResultSet.updateBlob()。

-删除了UpdatableResultSet中的重复代码（能从ResultSet继承而来，不需要各方法处理可更新性的额外代码，但我认为在将来可能需要）。

–更正了通过属性“强制”字符编码时抛出的“UnsupportedEncodingException”。

–更正了各种非ASCII字符编码问题。

–增加了驱动程序属性“useHostsInPrivileges”。默认值为“真”。它将影响是否在“DBMD.getColumn/TablePrivileges”中使用“@hostname”。

–描述方案的所有DBMD结果集列现返回NULL，以便能与用于其他数据库的其他JDBC驱动程序的行为更兼容（MySQL不支持方案）。

–增加了SSL支持。关于如何使用它的更多信息，请参见README。

–执行自动再连接或故障切换时，恰当恢复连接属性，包括autoCommit状态以及隔离级别。

–可能时，使用“SHOW CREATE TABLE”，以确定用于DatabaseMetaData的外键信息，对于返回的DELETE信息还允许级联选项。

–对于SJIS字符集，转义了字符串中的“0x5c”字符。

–更正了Clob.getSubString()中起始位置偏离1的错误。

–实现了Clob.truncate()。

–实现了Clob.setString()。

–实现了Clob.setAsciiStream()。

–实现了Clob.setCharacterStream()。

–增加了com.mysql.jdbc.MiniAdmin类，该类允许你将“shutdown”命令发送至MySQL服务器。在最终用户应用中嵌入Java和MySQL服务器时使用。

–增加了“connectTimeout”参数，允许JDK-1.4和更高版本的用户指定建立连接所需等候的最长时间。

–仅当连接处于autoCommit(false)状态时，故障切换和autoReconnect才能工作，目的在于保持事务的安全。

–增加了“queriesBeforeRetryMaster”属性，出现故障切换时而且在重新连接到主服务器之前，该属性指定了能够发出的查询数目（默认为50）。

–更正了DBMD.supportsResultSetConcurrency()，以便为ResultSet.TYPE_SCROLL_INSENSITIVE和ResultSet.CONCUR_READ_ONLY或ResultSet.CONCUR_UPDATABLE返回“真”。

–更正了ResultSet.isLast()处理空结果集方面的问题（应返回“假”）。

– PreparedStatement现在将优先考虑setBinary/Ascii/Character Stream()中的流长度，除非将连接属性“useStreamLengthsInPrepStmts”设置为“假”。

-删除了一些使用EscapeProcessor、Connection和DatabaseMetaData类中的“Strings smarter”创建的不需要的临时对象。

09-21-02：版本3.0.1-dev

-更正了ResultSet.getRow()偏差1的缺陷。

-更正了RowDataStatic.getAt()偏差1的缺陷。

-增加了有限Clob功能（ResultSet.getClob()、PreparedStatemtent.setClob()、PreparedStatement.setObject(Clob)）。

-为URL增加了socketTimeout参数。

- Connection.isClosed()不再对服务器执行“Ping”操作。

-当“getAutoCommit() == false”时Connection.close()发出rollback()。

-增加了“妄想”参数，通过删除其中的“敏感”信息清理了错误消息（即，主机名、端口、用户名等），并在可能的情况下，清理了“敏感”的数据结构。

-更正了ResultSetMetaData.isSigned()在处理TINYINT和BIGINT方面的问题。

- 现在将自动检测字符集。优化了单字节字符集转换的代码。

-实现了ResultSet.getCharacterStream()。

-为DatabaseMetaData.getTableTypes()中的表类型增加了“LOCAL TEMPORARY”。

- 整理了大块代码，以遵循Java编码惯例（时机成熟）。

07-31-02：版本3.0.0-dev

- !!! 许可变化!!! 驱动现在是GPL。如果需要非GPL许可，请与我们联系（mark@mysql.com）。

- JDBC-3.0功能包括Statement/PreparedStatement.getGeneratedKeys()和ResultSet.getURL()。

-性能增强，在大多数情况下，驱动程序快了50-100％，而且创建的临时对象更少。

-重新封装...新的驱动程序名是“com.mysql.jdbc.Driver”，但旧名称依然有效（驱动程序现由MySQL-AB提供）。

-更好地在语句和PreparedStatement中检查已关闭的连接。

-支持流式结果集（按行），请参见README，感谢Doron。

-支持大信息包（MySQL-4.0协议的新增内容），更多信息请参见README。

- JDBC兼容，除了存储程序测试外，提供了所有测试。

-更正并分类了DBMetaData中的主键名称（SF缺陷582086和582086）。

-浮点类型现为java.sql.Types.FLOAT（SF缺陷579573）。

- ResultSet.getTimestamp()现在能处理DATE类型（SF缺陷559134）。

- ResultSet.getDate/Time/Timestamp现在能识别由MySQL设置为全零的所有形式的无效值（SF缺陷586058）。

- Testsuite现在将使用Junit（可从www.junit.org获得）。

-驱动程序现在仅能与JDK-1.2或更高版本一起工作。

-增加了多主机故障切换支持（请参见README）。

-进行了一般性的源码清理。

-读取信息包时，通过控制MysqlIO类创建的过渡对象，改善了总的速度。

-改善了字符串处理和字段元数据创建的性能（例示），由Alex Twisleton-Wykeham-Fiennes提供。

05-16-02：版本2.0.14

-更多代码整理。

- PreparedStatement现在能释放.close()上的资源（SF缺陷553268）。

-如果服务器版本不支持，不使用引用ID。此外，如果服务器是以“—ansi”或“--sql-mode=ANSI_QUOTES”开始，那么“"”将用作ID引用，否则将使用“`”。

- ResultSet.getDouble()现在能更准确地使用JDK内置的代码（但较慢）。

- LogicalHandle.isClosed()调用，直至物理连接。

-增加了SQL仿型功能（到STDERR）。在JDBC url中设置“profileSql=true”。更多信息请参见README。

-更正了relaxAutoCommit参数的类型。

04-24-02：版本2.0.13

-更多代码整理。

-更正了未能正确读取的unicode字符（SF缺陷541088）。

-为PrepStmt提供了更快的Blob转义功能。

-为DataSource(s)增加了set/getPortNumber()（SF缺陷548167）。

-为MySQLXADataSource增加了setURL()（SF缺陷546019）。

-更正了PreparedStatement.toString()（SF缺陷534026）。

-实现了ResultSetMetaData.getColumnClassName()。

- 现在实施了来自JDBC-3.0的Statement.getGeneratedKeys()的初级版本（要想使之工作，需要使用JDK-1.4，我个人认为）。

- DBMetaData.getIndexInfo()，更正了不良的PAGES（SF缺陷542201）。

04-07-02：版本2.0.12

-一般性代码整理。

-为Connection和MysqlLogicalHandle增加了getIdleFor()方法。

-放松了所有类中的功能，应更正520615和520393。

-为DBMD增加了getTable/ColumnPrivileges()（更正484502）。

-为getTypeInfo()增加了新类型，更正了已有类型，感谢Al Davis和Kid Kalanon。

- PreparedStatement BIT 51870

为 增加了 类型支持（ ）。

-更正了ResultSet中的getRow()缺陷（527165）。

-更正了PreparedStatement中的ResultSet可更新性。

-更正了PreparedStatement中时区偏差1小时的缺陷（538286、528785）。

- ResultSet: 更正了可更新性（如果不可更新，将值设为NULL）。

- DataSources，更正了setUrl缺陷（511614、525565），错误的数据源类名（532816、528767）。

-为需要它们的所有DatabaseMetaData方法增加了ID引用（应更正518108）。

-增加了对YEAR类型的支持（533556）。

- ResultSet.insertRow()目前能够在大多数情况下检测auto_increment字段，并在新行中使用该值。但是，该检测不适用于多值键，原因在于MySQL协议不返回该类信息。

-实现了ResultSet.refreshRow()。

-更正了testsuite.Traversal afterLast()缺陷，感谢Igor Lastric。

01-27-02：版本2.0.11

-更正了DBMD.getImported/ExportedKeys()和getCrossReference()中丢失的DELETE_RULE值。

-Statement.java的完全同步。

-多处修改，更正了读取BLOB时的“Unexpected end of input stream”（输入流意外结束）错误。这应是最后更正。

01-24-02：版本2.0.10

-更正了MysqlIO中的虚假“Unexpected end of input stream”（输入流意外结束）错误（缺陷507456）。

-更正了与Websphere 4一起使用MysqlConnectionPoolDataSource时的“null-pointer-exceptions”（缺陷505839）。

01-13-02：版本2.0.9

- Ant创建失败，包括jar文件，已更正（缺陷487669）。

-更正了MysqlIO.readPacket()中额外的内存分配（缺陷488663）。

-实现了DatabaseMetaData.getExported/ImportedKeys()和getCrossReference()。

-在更改实例和类共享引用的方法上，实现了完全同步，驱动程序现在是完全线程安全的（如果遇到问题，请告诉我）。

-将DataSource实施移至org.gjt.mm.mysql.jdbc2.optional软件包，PooledConnectionDataSource和XADataSource的原始实施仍保留在原处（感谢Todd Wolff给出了PooledConnectionDataSource与IBM WebSphere 4一起的实施方案和测试结果）。

-增加了读取信息包时对关闭网络连接的检测（感谢Todd Lizambri）。

-更正了与转义处理其有关的错误（缺陷486265）。

-通过DatabaseMetaData，支持批更新（缺陷495101）。

-更正了PreparedStatement.setTimestamp()中偏差1小时的错误（缺陷491577）。

-从驱动程序中删除了级连支持（“||”操作符），较早的VisualAge版本似乎是使用它的唯一软件，它与逻辑“||”操作符冲突。要想使用“||”操作符作为级联，需要用“--ansi”标志启动mysqld（缺陷491680）。

-更正了PreparedStatement中的舍弃缺陷（488663）。

11-25-01：版本2.0.8

-现在支持批更新（感谢Daniel Rall的鼓励）。

- XADataSource/ConnectionPoolDataSource代码（实验性）。

- PreparedStatement.setAnyNumericType()现在能正确处理正指数（增加了“+”，因此MySQL能理解它）。

- DatabaseMetaData.getPrimaryKeys()和getBestRowIdentifier()目前在识别主键方面更加可靠（匹配，不管主键在Key_type列中的大小写、或缩写／全名）。

10-24-01：版本2.0.7

- PreparedStatement.setCharacterStream()现已实现。

-更正了处于高利用率模式（autoReconnect=true）下时的悬挂套接字问题，连接的finalizer将关闭任何在GC上悬挂的套接字。

-更正了在较新版本的MySQL上ResultSetMetaData.getPrecision()返回的值比实际值小1的问题。

-如果列值为NULL，ResultSet.getBlob()现在将返回NULL。

-如果useUnicode=true而且未设置characterEncoding，将从数据库读取字符集。（感谢Dmitry Vereshchagin）。

-从数据库读取初始事务隔离级别（如果可用的话）（感谢Dmitry Vereshchagin）。

-更正了DatabaseMetaData.supportsTransactions()，supportsTransactionIsolationLevel()，getTypeInfo() SQL_DATETIME_SUB，以及不可读取的SQL_DATA_TYPE字段。

-更正了用于生成SQL并回在某些查询中以语法错误结束的PreparedStatement。

-更正了ResultSet.isAfterLast()总返回“假”的问题。

-更正了PreparedStatement.setTimestamp()中的时区问题（感谢Erik Olofsson）。

-在URL或属性中传递了“captializeTypeNames=true”时，将类型名转换为大写（对于WebObjects，感谢Anjo Krank）。

-可更新结果集现在能正确处理字段中的NULL值。

- PreparedStatement.setDouble()现在能使用双精度值（撤销了以前所作的截取更正）。

-如果MySQL的版本高于或等于3.21.23，PreparedStatement.setBoolean()将使用1/0值。

06-16-01：版本2.0.6

-更正了PreparedStatement参数检查功能。

-更正了ResultSet.java中区分大小写的列名。

06-13-01：版本2.0.5

- 更正了ResultSet.getBlob() ArrayIndex超出范围的问题。

-更正了ResultSetMetaData.getColumnTypeName关于TEXT/BLOB的问题。

-更正了发送大BLOB查询时的ArrayIndexOutOfBounds问题（未设置最大信息包大小）。

-为Connection.setIsolationLevel()增加了ISOLATION级别支持。

-更正了所有列均未设置时在PreparedStatement.executeUpdate()上的NPE问题。

-更正了采用两位数字年份的TIMESTAMP的数据解析问题。

-为PreparedStatement.setObject()增加了Byte。

- ResultSet.getBoolean()现在能将“-1”识别为“真”。

- ResultSet具有+/-Inf/inf支持特性。

-即使并非所有列均已被设置（设为NULL），ResultSet.insertRow()现在也能工作。

- DataBaseMetaData.getCrossReference()不再使用ArrayIndexOOB。

-作用在结果集上的getObject()能正确执行TINYINT->Byte和SMALLINT->Short操作。

12-03-00：版本2.0.3

-为JDBC2实现了不带标度组分的getBigDecimal()。

-更正了与可更新结果集有关的复合键问题。

-增加了-/+INF对双精度的检测。

-更快的ASCII字符串操作。

-更正了MAX_ALLOWED_PACKET的不正确检测，因此，现在能发送大的Blob。

-更正了java.sql.Blob实施代码中的“偏差1”错误。

-增加了“ultraDevHack”URL参数，将其设置为“真”，可允许Macromedia UltraDev使用该驱动程序。

04-06-00：版本2.0.1

-更正了RSMD.isWritable()返回错误值的问题。感谢Moritz Maass。

-整理了连接确定时的异常处理功能。

-使用getObject()，具有TEXT类型的列现在会作为字符串返回。

- DatabaseMetaData.getPrimaryKeys() key seq Brian Slesinsky

现在能正确工作（写入到 ）。感谢 。

-按照JDBC规范，在PreparedStatements上，不再进行转义处理。

-更正了很多JDBC-2.0遍历、定位错误，尤其是写入空结果集的问题。感谢Ron Smits、Nick Brook、Cessar Garcia和Carlos Martinez。

-更正了使用多个主键时，与结果集中可更新性支持有关的一些问题。

02-21-00：版本2.0pre5

-更正了不良的握手问题。

01-10-00：版本2.0pre4

-更正了针对insertRow()的结果集，感谢Cesar Garcia。

-对驱动程序进行了修改，使之能够通过加载JDBC-2.0类来识别JDBC-2.0，而不是依靠JDK版本号。感谢John Baker。

-更正了结果集，以返回正确的行号。

- Statement.getUpdateCount()现在能返回匹配的行，而不是实际更新的行，它更像SQL-92。

10-29-99

-更正了Statement/PreparedStatement.getMoreResults()缺陷。感谢Noel J. Bergman。

-为PreparedStatement.setObject()增加了Short类型。感谢Jeff Crowder。

-驱动程序现在能通过查询服务器自动配置最大/首选的信息包大小。

-如果服务器支持，Autoreconnect代码将使用更快的ping命令。

-更正了从服务器读取信息包、以及为写入到服务器而分配信息包时与信息包大小有关的多种缺陷。

08-17-99：版本2.0pre

-目前是在JDK-1.2下编译的。通过核心类集合，驱动程序同时支持JDK-1.1和JDK-1.2。在运行时，通过判断你所使用的JVM版本，驱动程序将加载恰当的接口类。

-修正了在首行中全为NULL的结果集。（由Tim Endres指出）。

-更正了结果集内SQLExceptions的列编号（感谢Blas Rodriguez Somoza）。

-不再需要将数据库指定给连接。（感谢Christian Motschke）。

07-04-99：版本1.2b

-更好的文档（不断改善），doc/mm.doc/book1.html。

-对于列名模式，DBMD现在允许null（未在规范中定义），它将被更改为“%”。

- DBMD现在提供了针对getXXX()的正确类型／长度。

-修改了ResultSet.getDate()、getTime()和getTimestamp()。（由Alan Wilken提供）。

- EscapeProcessor现在能正确处理引号内的“\{ \}”和“{ or }”。（感谢Alik就如何修正它给出的观点）。

-对连接中的属性处理功能进行了修正。（由Juho Tikkala提供）。

-对于表中的NULL列，ResultSet.getObject()现在能返回NULL，而不是销毁。（感谢Ben Grosman）。

-对于MySQL不了解的类型，ResultSet.getObject()现在能返回字符串。（由Chris Perdue建议）。

-删除了不需要的DataInput/Output流，对于每次IO操作，1／2的方法调用都是不需要的。

-如果未指定字符编码，使用默认的字符编码。这是对已损坏JVM的一种避规措施，这是因为，按照规范，所有的JVM都必须支持“ISO8859_1”，但并非如此。

-更正了连接事宜，如果未明确设置字符编码，将使用平台的字符编码，而不是“ISO8859_1”。它修正了加载并不总存在的字符转换器类时存在的问题（JVM缺陷）。（感谢Fritz Elfert指出了该问题）。

-修改了MysqlIO，使之在可能的情况下再次使用信息包，而不是降低内存使用率。

-更正了与引号内“{}”有关的转义处理器缺陷。

04-14-99：版本1.2a

-更正了对非Javasoft JVM的字符集支持（感谢很多指出该问题的人员）。

-更正了ResultSet.getBoolean()，使之能够识别作为布尔标志的“y”和“n”，以及“1”和“0”。（感谢Tim Pizey）。

- ResultSet.getTimestamp() Richard Swift

更正了 ，以提供更好的性能。（感谢 ）。

- 更正了getByte()在处理数值类型方面的问题。（感谢Ray Bellis）。

- 更正了DatabaseMetaData.getTypeInfo()在处理DATE类型时存在的问题。（感谢Paul Johnston）。

-更正了用于“fn”调用的EscapeProcessor。（感谢locomotive.org的Piyush Shah）。

-更正了EscapeProcessor，如果没有转义代码，不执行额外操作。（感谢Ryan Gustafson）。

-更正了驱动程序，使之能解析“jdbc:mysql://host:port”形式的URL（感谢Richard Lobb）。

03-24-99：版本1.1i

-更正了关于PreparedStatements的Timestamps问题。

-更正了RSMD和RS中的Null指针异常。

-对于有效的类文件，与jikes一起进行了再编译（感谢ms!）。

03-08-99：版本1.1h

-更正了转义处理器，以处理不匹配的“{”和“}”（感谢Craig Coles）。

-更正了转义处理器，以创建移植性更好的（在DATETIME和TIMESTAMP类型间）表达式，以便能与BETWEEN子句一起使用。（感谢Craig Longman。

- MysqlIO.quit()现在能关闭套接字连接。在此之前，多次连接失败后，某些操作系统将耗尽文件描述符。（感谢Michael Brinkman）。

-更正了Driver.getPropertyInfo中的NullPointerException（感谢Dave Potts）。

-修正了MysqlDefs，允许字字符串形式检索所有的*text字段。（感谢Chris at Leverage）。

-更正了PreparedStatement中的setDouble，使之能用于处理大数字，防止将科学记数法发送到数据库。（感谢J.S. Ferguson）。

-更正了RSMD中的getScale()和getPrecision()。（由James Klicman贡献）。

-更正了字段为DECIMAL或NUMERIC时的getObject()（感谢Bert Hobbs）。

-传递Null表名时，DBMD.getTables()出现严重故障。已更正（感谢Richard Lobb）。

-增加了在连接过程中对“client not authorized”（客户端未被授权）错误的检查。（感谢Hannes Wallnoefer）。

02-19-99：版本1.1g

-结果集行现在是字节数组。Blob和Unicode现在能双向工作。目前实施了useUnicode和编码选项。

-修正了PreparedStatement，使用setXXXStream（不改变地发送）将二进制集合发送到MySQL服务器。

-修正了getDriverPropertyInfo()。

12-31-98：版本1.1f

-将所有的结果集字段更改为字符串，这样，应能使Unicode工作，但你的JVM必须能够在字符集间进行转换。它还能使对服务器的数据读取更快，原因在于，现在不存在从StringBuffer到String（字符串）的转换。

-更改了PreparedStatement.streamToString()，使之更有效（代码由Uwe Schaefer提供）。

- URL解析功能更可靠（对于错误，抛出SQL异常，而不是NullPointerExceptions）。

- PreparedStatement现在能通过setObject()将String转换为Time/Date值（代码由Robert Currey提供）。

-在Buffer.readInt()中，IO进程不再被挂起，该缺陷是在1.1d中当将结果集更改为全字节数组时引入的。（由Samo Login指出）。

11-03-98：版本1.1b

-修正了DatabaseMetaData，允许IBM VA和J-Builder同时工作。请告诉我它的工作机制。（感谢Jac Kersing）。

-修正了ResultSet.getBoolean()在处理NULL字符串方面的问题（感谢Barry Lagerweij）。

-开始代码整理，并进行了格式处理。开始将其分出为并行的JDBC-2.0源树。

-为MysqlIO和Buffer内的关键部分增加了“最终”限定符，允许编译器采用内联方法以提高速度。

9-29-98

-如果传递给PreparedStatement中setXXX()的对象引用是空的，将自动调用setNull()。（感谢Erik Ostrom给出的提议）。

-对于Types.OTHER对象和未知类型对象，PreparedStatement中的setObject()现在会尝试将对象的串行化表示写入到数据库。

- Util现在有了1个静态方法readObject()，结果集和列索引将以上述方式在此例示序列化的对象。

9-02-98 — 版本1.1

-消除了MysqlIO.nextRow()中的“丑陋缺陷”。更正了Buffer.isLastDataPacket()，而不是获取异常。

- Connection.getCatalog()和Connection.setCatalog()现在能够工作。

- Statement.setMaxRows()能够正常工作，也能使用属性maxRows进行设置。通过属性或URL参数，Statement.setMaxRows()能覆盖maxRows设置。

-提供了自动再连接功能。由于在每次查询前不得不Ping数据库，在默认情况下，它将被关闭。要想使用该功能，请在连接URL中传递“autoReconnect=true”。通过“maxReconnects=n”（默认为3）和“initialTimeout=n”（默认为2秒）参数，你也可以更改再连接尝试的次数和初始超时值。Timeout是超时的指数补偿类型。例如，如果初始超时设置为2秒，maxReconnects为3，那么再连接尝试之间的间隔分别是2秒，4秒和16秒。

8-24-98：版本1.0

- 更正了Buffer.java中Blob数据的处理功能。

-更正了与尺寸过小的鉴定信息包有关的缺陷。

-JDBC驱动程序现在采用LPGL。

8-14-98 -

- 更正了 Buffer.readLenString()，使之能正确读取BLOB数据。

-更正了PreparedStatement.stringToStream，使之能正确读取BLOB数据。

-更正了PreparedStatement.setDate()，使之不增加1天。（感谢Vincent Partington在上述修正项方面的贡献）。

-增加了URL参数解析功能（?user=...等等）。

8-04-98：版本0.9d

- 重大新闻！ 新的软件包名。ICE工程公司的Tim Endres着手为GNU GPL的Java软件建立新的源树。他友好地将org.gjt.mm软件包目录提供给我，因此，现在驱动程序在org.gjt.mm.mysql软件包中。目前我是合法用户。期待Tim项目的更多信息。

-现在采用了动态确定大小的信息包，向数据库发送命令时，能够减少内存使用。

-对getTypeInfo()的参数等方面进行了小的修正。

- DatabaseMetaData现已完全实现。如果这些驱动程序能与各种IDE一起工作，请告知。我已听说它们正与Jbuilder一起使用。

-在软件包中增加了JavaDoc文档。

-软件包采用.zip或.tar.gz格式提供。

7-28-98：版本0.9

-实现了getTypeInfo()。根据JDBC规范，Connection.rollback()现在能抛出SQLException。

-增加了PreparedStatement，它支持预处理语句的所有JDBC API方法，包括InputStreams。请检查该点，如有问题，请告知。

-更正了ResultSet中的1个缺陷，该缺陷会破坏仅返回1行的某些查询。

- 更正了DatabaseMetaData.getTables()、DatabaseMetaData.getColumns()和DatabaseMetaData.getCatalogs()中存在的缺陷。

-增加了语句的功能，允许executeUpdate()保存由AUTO_INCREMENT字段自动生成的ID值。一般而言，执行executeUpdate()后，将查找SQLWarnings以了解警告信息，如LAST_INSERTED_ID = 'some number'，COMMAND = 'your SQL query'。

如果在表中正使用AUTO_INCREMENT字段，并在一条语句上执行了多个executeUpdate()，务必每次都执行clearWarnings()以节省内存。

7-06-98：版本0.8

-将MysqlIO和Buffer分离为单独类。对于这两个类中的某些字段，一些ClassLoader（类加载器）会给出IllegalAccess（非法访问）错误。现在，mm.mysql能够在小应用程序和所有类加载器中正常工作。

感谢Joe Ennis jce@mail.boone.com指出该问题，并与我一起进行了修正。

7-01-98：版本0.7

-更正了getColumns()中的DatabaseMetadata problems，并更正了字段构造函数内开关语句中存在的缺陷。

感谢Costin Manolache costin@tdiinc.com指出了它们。

5-21-98：版本0.6

-在MysqlIO.java和ResultSet.java中，结合Richard Swift Richard.Swift@kanatek.ca给出的有效变更内容。

-现在，我们的驱动程序比GWE的驱动程序快15％。

- 开始着手处理DatabaseMetaData。

实现了下述方法:

```
* getTables()
* getTableTypes()
* getColumns
* getCatalogs()
```

26.4. MySQL Connector/MXJ

26.4.1. 前言
26.4.2. 支持平台:
26.4.3. Junit测试要求
26.4.4. 运行Junit测试
26.4.5. 作为JDBC驱动程序的一部分运行
26.4.6. 在Java对象中运行
26.4.7. MysqldResource API
26.4.8. 在JMX代理(custom)中运行
26.4.9. 部署在标准的JMX代理环境下 (JBoss)
26.4.10. 安装

26.4.1. 前言

MySQL Connector/MXJ是一种Java实用工具软件包，用于部署和管理MySQL数据库。可以将Connector/MXJ捆绑到已有的Java应用程序，或将其作为JMX Mbean部署。MySQL的部署和使用十分简单，就像为JDBC连接URL添加额外参数一样简单，连接建立后，连接URL将启动数据库。这样，Java开发人员能够通过降低其最终用户的安装难度，更容易地部署需要数据库的应用程序。

MySQL Connector/MXJ使得MySQL数据库像是基于java的组件。它通过下述步骤完成该任务，确定系统所运行的平台，选择恰当的二进制文件，并执行程序。也能可选地部署具有任何指定参数的个初始数据库。

作为一种JMX Mbean，MySQL Connector/MXJ需要与JMX v1.2兼容的Mbean容器，如Jboss版本4。Mbean将使用标准的JMX管理API来展示与平台相适应的参数（并允许设置参数）。

其中包含与JDBC驱动程序一起使用的说明，以及以JMX Mbean方式将其部署至Jboss的说明。

可从下述站点下载源版本和二进制版本：http://dev.mysql.com/downloads/connector/mxj/

这是一种测试板，欢迎反馈和鼓励。

如有任何问题或意见，请发送电子邮件至java@lists.mysql.com。

26.4.2. 支持平台:

- Linux, i386
- Windows NT, x86
- Windows 2000, x86
- Windows XP, x86
- Solaris 9, SPARC 32

26.4.3. Junit测试要求

要想确保你的平台是在支持范围内，最好方法是运行Junit测试。

首先应确保组件能够在平台上工作。由于"MysqldResource"类实际上是MySQL固有版本的包装器，并非所有的平台均支持它。编写本文时，对运行在i386架构上的Linux进行了测试，看上去工作良好，就像在OS X v10.3上一样。在Windows和Solaris平台上进行了有限测试。

要求：

1. JDK-1.4或更高版本（或JRE，如果不打算编译源文件或JSP的话）。
2. 通过CLASSPATH安装了并提供了MySQL Connector/J版本3.1或更高版本（http://dev.mysql.com/downloads/connector/j/ ）。
3. 用于JMX版本1.2.1的javax.management类，它们位于下述应用服务器上：
   - JBoss - 4.0rc1或更高版本
   - Apache Tomcat - 5.0或更高版本
   - Sun公司的JMX参考实施版本1.2.1，http://java.sun.com/products/JavaManagement/。
4. Junit 3.8.1（http://www.junit.org/）

如果从源码创建，除了上述所有要求外，还须满足：

1. Ant版本1.5或更高版本（可从http://ant.apache.org/ 上下载）。

26.4.4. 运行Junit测试

1. 这类测试将尝试在3336端口上启动MySQL。如果有正在运行的MySQL，可能会出现冲突，但可能性不大，原因在于MySQL的默认端口是3306。然而，也可以将"c-mxj_test_port"Java属性设置为你所选择的端口。作为可选方式，你也可以通过关闭运行在目标机器上的MySQL实例来启动测试。

默认情况下，测试结果将输出到控制台。要想获得详细输出，可以将"c-mxj_test_silent"Java属性设置为"假"。

2. 要想运行Junit测试套件，$CLASSPATH必须包含下述部分：
   - JUnit
   - JMX
   - Connector/J
   - MySQL Connector/MXJ
3. 如果你下载的文件中不含connector-mxj.jar，请解包MySQL Connector/MXJ源码档案文件。
4.
```
cd mysqldjmx
```
5.
```
ant dist
```

随后，将$TEMP/cmxj/stage/connector-mxj/connector-mxj.jar添加到CLASSPATH（类路径）。

6. 如果有junit，执行单元测试。从命令行上输入：
7.
```
java junit.textui.TestRunner com.mysql.management.AllTestsSuite
```

输出与下面给出的类似:

```
.......................................
.......................................
..........
Time: 259.438


OK (101 tests)
```

注意，在快结束时速度会变慢，请耐心等候。

26.4.5. 作为JDBC驱动程序的一部分运行

MySQL Connector/J JDBC驱动程序的1个特点是，能够在JDBC连接字符串中将"SocketFactory"指定为参数。MySQL Connector/MXJ包含1个定制的SocketFactory。首次连接时，SocketFactory将部署并启动MySQL数据库。SocketFactory也会显示1个"shutdown"方法。

要想使用它，请在JDBC连接字符串上指定"socketFactory"参数，并将其值设为"com.mysql.management.driverlaunched.ServerLauncherSocketFactory"。

在下面的示例中，有1个能创建连接的程序，执行查询，并将结果输出到System.out。MySQL数据库将作为连接进程的组成部分予以部署并启动，最后是结束部分。

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

import com.mysql.management.driverlaunched.ServerLauncherSocketFactory;

public class ConnectorMXJTestExample {
    public static void main(String[] args) throws Exception {
        String hostColonPort = "localhost:3336";

        String driver = com.mysql.jdbc.Driver.class.getName();
        String url = "jdbc:mysql://" + hostColonPort + "/" + "?"
                + "socketFactory="
                + ServerLauncherSocketFactory.class.getName();
        String userName = "root";
        String password = "";

        Class.forName(driver);
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(url, userName, password);
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT VERSION()");
            rs.next();
            String version = rs.getString(1);
            rs.close();
            stmt.close();

            System.out.println("------------------------");
            System.out.println(version);
            System.out.println("------------------------");
        } finally {
            try {
                conn.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
            ServerLauncherSocketFactory.shutdown(hostColonPort);
        }
    }
}
```

CLASSPATH connector-mxj.jar Connector/J

要想运行上述程序，在 中必须有 和 。然后键入：

```
java ConnectorMXJTestExample
```

当然，对于MySQL数据库，有很多可设置的选项。通过为每个服务器选项冠以前缀"server"，可将其作为JDBC连接字符串的部分，简单地指定这些选项。在下述示例中，我们设置了3个驱动程序参数和2个服务器参数：

```
        String url = "jdbc:mysql://" + hostColonPort + "/"
                + "?"
                + "socketFactory="
                + ServerLauncherSocketFactory.class.getName();
                + "&"
                + "cacheServerConfiguration=true"
                + "&"
                + "useLocalSessionState=true"
                + "&"
                + "server.basedir=/opt/myapp/db"
                + "&"
                + "server.datadir=/mnt/bigdisk/myapp/data";
```

26.4.6. 在Java对象中运行

有1个java应用程序并打算嵌入MySQL数据库，直接使用com.mysql.management.MysqldResource类。可以使用默认的构造函数（无参量）例示该类，或者通过在java.io.File对象（代表希望服务器解包至的目录）中传递类来例示之。也可用针对"stdout"和"stderr"（用于记录）的输出流例示它。

一旦完成例示，java.util.Map，该对象将能提供与平台以及希望使用的MySQL版本相适应的服务器选项的java.util.Map。

MysqldResource允许你使用所提供的服务器选项的java.util.Map启动MySQL，并允许你关闭数据库。在下面的示例中，给出了使用明码java对象将MySQL嵌入到应用程序的简单方法。

```
import com.mysql.management.MysqldResource;

 ...

    public void startMySQL() {
        File baseDir = new File(ourAppDir, "mysql");
        mysqldResource = new MysqldResource(baseDir);
        Map options = new HashMap();
        options.put("port", "3336");
        String threadName = "OurApp MySQL";
        mysqldResource.start(threadName, options);
    }

    public void stopMySQL() {
        if (mysqldResource != null) {
            mysqldResource.shutdown();
        }
        mysqldResource = null;
    }

    public java.sql.Connection getConnection() throws Exception {
        String db = "test";
        String url = "jdbc:mysql://localhost:3336/" + db;
        String userName = "root";
        String password = "";
        Class.forName(com.mysql.jdbc.Driver.class.getName());
        return DriverManager.getConnection(url, userName, password);
    }
```

26.4.7. MysqldResource API

构造函数：

- public MysqldResource(File baseDir, PrintStream out, PrintStream err);

允许设置安装MySQL文件的"basedir"，并设置标准输出和标准错误的输出流。

- public MysqldResource(File baseDir);

允许设置安装MySQL文件的"basedir"。标准输出和标准错误的输出将被导至System.out和System.err。

- public MysqldResource();

Basedir是java.io.tempdir的默认子目录。标准输出和标准错误的输出将被导至System.out和System.err。

MysqldResource API包含下述方法：

- void start(String threadName, Map mysqldArgs);

部署并启动MySQL。"threadName"字符串用于命名实际执行MySQL命令行命令的线程。"map"是将要传递给命令行的参量和参联值的集合。

- void shutdown();

关闭由MysqldResource对象管理的MySQL实例。

- Map getServerOptions();

返回所有选项以及MySQL数据库可用的当前选项（或默认选项，如果未运行的话）的映射。

- boolean isRunning();

如果MySQL数据库正在运行，返回"真"。

- boolean isReadyForConnections();

一旦数据库通报它已做好连接准备，返回"真"。

- void setKillDelay(int millis);

默认的"Kill Delay"是30秒。它表示发出初始关闭请求和发出"强制杀死"（如果数据库未关闭）命令之间需要等待的时间。

- void addCompletionListenser(Runnable listener);

当服务器进程完成时，允许通知应用程序。每个"listener"（监听程序）将在自己的线程中发出。

- String getVersion();

返回MySQL的版本。

- void setVersion(int MajorVersion, int minorVersion, int patchLevel);

标准分发版本仅提供了1种版本的MySQL软件包。但也能将多个版本封装在一起，并指定要使用的版本。

26.4.8. 在JMX代理(custom)中运行

如果你正在使用JMX的SUN参考实施版本，可跳过本节。或者，如果你正在部署Jboss，请跳到下一节。

我们希望在JMX代理的活动中看到MysqldDynamicMBean。在com.mysql.management.jmx.sunri软件包中，它是带有2个Mbeans的JMX代理：

1. MysqldDynamicMBean，以及
2. com.sun.jdmk.comm.HtmlAdaptorServer，它提供了用于操控JMX代理内众多元素的Web接口。

启动了这个十分简单的代理程序后，允许用Web浏览器启动并停止MySQL数据库。

1. 如前所述，完成平台测试。
- 当前JDK, JUnit, Connector/J, MySQL Connector/MXJ
- 本节需要JMX的SUN参考实施版本
- PATH, JAVA_HOME, ANT_HOME, CLASSPATH
2. 如果不是从源码创建的，跳到下一步。

rebuild with the "sunri.present"

```
ant -Dsunri.present=true dist
re-run tests:
java junit.textui.TestRunner com.mysql.management.AllTestsSuite
```

3. 从命令行启动测试代理：

```
4.    java com.mysql.management.jmx.sunri.MysqldTestAgentSunHtmlAdaptor &
```

5. 从浏览器：

```
6.          http://localhost:9092/
```

7. 在MysqldAgent下

```
8.            选择“name=mysqld”
```

9. 观察Mbean视图
10. 滚动到屏幕底部，按startMysqld按钮
11. 点击“Back to MBean View”（返回Mbean视图）
12. 滚动到屏幕底部，按stopMysqld按钮
13. 杀死运行测试代理的Java进程（jmx服务器）

26.4.9. 部署在标准的JMX代理环境下 (JBoss)

一旦确定Mbean能够在平台上工作，接下来应在标准的JMX代理内部署Mbean。其中包含部署到Jboss的说明。

1. 确保有最新版本的java开发工具箱（v1.4.x），请参见前面的介绍。
- 确保设置JAVA_HOME（Jboss要求JAVA_HOME）。
- 确保JAVA_HOME/bin位于PATH中（不需要设置CLASSPATH，也不需要以前测试中使用的任何jar文件）。
2. 确保安装了Jboss的最新版本（v4.0RC1或更高）。

```
3.            http://www.jboss.org/index.html
4.            选择“Downloads”。
```

```
5.              选择“jboss-4.0.zip”。
6.              选择1个镜像。
7.              unzip ~/dload/jboss-4.0.zip
8.              创建JBOSS_HOME环境变量，设置解包目录。
9.              仅对Unix：
10.        cd $JBOSS_HOME/bin
11.        chmod +x *.sh
```

12.将connector-mxj.jar安装（拷贝）到$JBOSS_HOME/server/default/lib。

13.将mysql-connector-java-3.1.4-beta-bin.jar安装（拷贝）到$JBOSS_HOME/server/default/lib。

14.在$JBOSS_HOME/server/default/deploy下创建mxjtest.war目录。

15. 将index.jsp安装（拷贝）到$JBOSS_HOME/server/default/deploy/mxjtest.war。

16.在$JBOSS_HOME/server/default/deploy下创建mysqld-service.xml文件。

```
17.        <?xml version="1.0" encoding="UTF-8"?>
18.         <server>
19.          <mbean code="com.mysql.management.jmx.jboss.JBossMysqldDynamicMBean"
20.             name="mysql:type=service,name=mysqld">
21.          <attribute name="datadir">/tmp/xxx_data_xxx</attribute>
22.          <attribute name="autostart">true</attribute>
23.          </mbean>
24.         </server>
```

25.启动jboss：

- 在Unix上：**$JBOSS_HOME/bin/run.sh**
- 在Windows上：**%JBOSS_HOME%\bin\run.bat**

准备就绪：Jboss在屏幕上显示大量输出。

26. 当Jboss看上去停止将信息输出到屏幕上时，打开Web浏览器：http://localhost:8080/jmx-console

27.滚动到mysql部分页面底部，选择bulleted mysqld链接。

28.观察JMX MBean View页面。MySQL应已运行。

29. （如果设置了“autostart=true”可跳过该步）。滚动到屏幕底部。按Invoke按钮停止（或启动）MySQL，观察已成功完成而且无返回值的操作。点击“Back to MBean View”（返回Mbean视图）

30. 要想确定MySQL是否正在运行，打开Web浏览器http://localhost:8080/mxjtest/，应看到：

```
SELECT 1
```

returned with a result of

```
1
```

31.按照$JBOSS_HOME/server/default/deploy/mxjtest.war/index.jsp中的介绍，能够在你的Web应用程序中使用MySQL。其中提供了供测试用的测试数据库和根用户（无密码）。创建表，插入一些行，并进行一些选择。

32. 关闭MySQL。停止Jboss时，MySQL将自动停止，或：在浏览器中，滚动到MBean View底部，并按停止服务Invoke按钮中止服务。观察已成功完成而且无返回值的操作。使用ps或任务管理器查看MySQL是否已不再运行。

对于1.0.6-beta版，能够在启动时让Mbean启动MySQL数据库。此外，我们还借鉴了Jboss生命周期扩展方法的优点，关闭Jboss时能优雅地关闭数据库。

26.4.10. 安装

如果阅读了上述部分，应已完成了这些步骤。但我们在下面列出了它们，以供快速参考。

启动了驱动程序：

1. 下载并解包Connector/MXJ，将connector-mxj.jar添加到CLASSPATH。

2. 为JDBC连接字符串添加下述参数："socketFactory=" + ServerLauncherSocketFactory.class.getName()

JBoss:

1. 下载Connector/MXJ，将connector-mxj.jar文件拷贝到目录$JBOSS_HOME/server/default/lib。

2. 下载Connector/J，将connector-mxj.jar文件拷贝到目录$JBOSS_HOME/server/default/lib。

3. 有任意属性设置在$JBOSS_HOME/server/default/deploy目录下创建Mbean服务xml文件，例如datadir和autostart。

4. 设置Web应用程序的JDBC参数，以使用：String driver = "com.mysql.jdbc.Driver"; String url = "jdbc:mysql:///test?propertiesTransform="+ "com.mysql.management.jmx.ConnectorMXJPropertiesTransform"; String user = "root"; String password = ""; Class.forName(driver); Connection conn = DriverManager.getConnection(url, user, password);

你或许希望为每个应用程序创建单独用户和数据库表空间，而不是使用根用户和测试数据库。

强烈建议定期备份，将数据库文件备份到datadir目录下。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

第27章：扩展MySQL

目录

27.1. MySQL内部控件

27.1.1. MySQL线程
27.1.2. MySQL测试套件

27.2. 为MySQL添加新函数

27.2.1. 自定义函数接口的特性
27.2.2. CREATE FUNCTION/DROP FUNCTION 语法
27.2.3. 添加新的自定义函数
27.2.4. 添加新的固有函数

27.3. 为MySQL添加新步骤

27.3.1. 步骤分析
27.3.2. 编写步骤

27.1. MySQL内部控件

27.1.1. MySQL线程
27.1.2. MySQL测试套件

本章包含许多在你处理MySQL代码时需要了解的你事情。如果你想投入到MySQL的开发中，或想要接触到最新的中间版本的代码，或者就是想了解开发的进度，请参阅2.8.3节，“从开发源代码树安装”的说明。如果你对MySQL的内部插件感兴趣，你也可以订阅我们的内部插件邮件列表。这个列表的流量相对低一些。欲知如何订阅的详情，请参阅1.7.1.1节，“MySQL邮件列表”。在MySQL AB 的所有开发人员都在内部插件列表里， 此外，我们帮助那些正在处理MySQL代码的人。请随意使用这个邮件列表来问代码有关的问题，也可用它来发送你想奉献给MySQL项目的 补丁！

27.1.1. MySQL线程

MySQL服务器创建如下线程：

- TCP/IP 连接线程处理所有连接请求，并为每一个连接创建一个新的专用线程来处理认证和SQL查询处理。
- Windows NT 平台上有一个名为管道处理程序（pipe handler）的线程，它和名为管道连接请求（pipe connect requests）的TCP/IP连接线程做同样的工作。
- 信号线程处理所有的信号，这个线程通常也处理报警和调用process_alarm() 函数来强制使得空闲时间太长的连接超时。
- 若**mysqld**是与DUSE_ALARM_THREAD线程一起编译的，这个专用线程是处理 创建的警报的。这个线程用在一些sigwait()函数有问题的系统上，或者用在你想在应用程序中使用thr_alarm()代码而不带专用信号处理线程之时。
- 若想使用flush_time=*val*选项，会创建一个专用线程以给定的时间间隔刷新所有表格。

- 每个连接都有它自己的线程。
- 每个被使用INSERT DELAYED 的不同表格都会有自己的线程。
- 若使用了master-host, 则会创建一个从属的复制线程从主线程读取并实施更新。

mysqladmin processlist 仅显示连接，INSERT DELAYED, 及复制线程

27.1.2. MySQL测试套件

27.1.2.1. 运行MySQL测试套件
27.1.2.2. 扩展MySQL测试套件
27.1.2.3. MySQL测试套件报告缺陷

包含在Unix源码和二进制分发版中的测试系统可以让用户和开发人员对MySQL代码施行回归测试。这些测试可以在Unix上进行，目前它们还不能在原生的Windows环境下进行。

当前的测试案例套件不能在MySQL中测试所有东西，但是它能发现SQL处理代码，OS/library文件中大多数明显的缺陷，并且在测试复件方面也是非常彻底的。我们的终极目标是对100%的代码进行测试。我们欢迎大家给我们的测试套件添加内容。你可能会特别想贡献出那些检查你系统里功能性危机的测试，因为这将确保未来所有发行版的MySQL会与你的应用程序一起更好地运行。

27.1.2.1. 运行MySQL测试套件

测试系统包括一个测试语言解释器(**mysqltest**)，一个运行所有测试的外壳脚本(**mysql-test-run**)，用专用语言编写的测试案例，以及它们的预期结果。在系统上编译好之后，在源代码的root下键入**make test** 或mysql-test/mysql-test-run。如果安装了一个二进制分发版， cd 到安装root (如 /usr/local/mysql), 然后键入scripts/mysql-test-run。所有测试应该都通过，假使有没通过的，若是一个MySQL里的缺陷，你可以试着找找是因为什么，并且报告这个问题。请参阅27.1.2.3节，“在MySQL测试套件里报告缺陷”。

如果你想要运行测试套件的机器上已经运行了一个 **mysqld** ，只要它不占用9306 和 9307端口，就不用停掉它。如果占用了其中的一个，以可以编辑**mysql-test-run**把主端口和（或）从端口号改为其它可用的。.

可使用下面指令运行单个测试案例 mysql-test/mysql-test-run test_name.

若一个测试未通过，你可以用--force选项来检查运行着的**mysql-test-run**看是否是别的测试未通过。

27.1.2.2. 扩展MySQL测试套件

你可以用**mysqltest** 语言编写你自己的测试案例。不幸地是，我们还没有写完相关方面完整地文档。但是，你可以查看我们现有的测试案例，并将它们作为范例。下面几点将有助于你入手:

- 测试位于 mysql-test/t/*.test
- 测试案例包括终止声明，测试案例类似于**mysql**命令行客户端的输入。 默认的声明是一个被发送到MySQL服务器的查询，除非这个声明被识别为内部命令（如 **sleep**）。
- 所有产生结果的查询，例如SELECT, SHOW, EXPLAIN等，必须在 *@/path/to/result/file*之前。那个文件必须包含期望的结果。生成结果文件的一个简单办法是在mysql-test目录运行**mysqltest -r < t/test-case-name.test** ，然后编辑生成的结果文件，如果需要，可将它们调整到想要的输出端。在那种情况下，要小心避免添加或删除任何不可见的字符，确保只改变文本和（或）删除行。如果插入一行，要确保插入的区域被一个硬标识隔开，且在行尾有一个硬标识。你可能会想要使用**od -c**来确保你的文本编辑器在编辑 步骤中没有搞乱任何东西。当你发现一个缺陷而不得不编辑**mysqltest -r**的输出时，我们真希

望你不要编辑它。

- 为和我们的设置一致，你应该把你的结果文件放在mysql-test/r 目录，并取名为test_name.result。如果测试产生不止一个结果，你应该使用诸如 test_name.a.result，test_name.b.result等这样的名字。
- 如果声明返回一个错误，你可以在声明的前一行使用--error error-number来详细说明它。错误号可能是由"，"分开的可能错误号的列表。
- 如果你正编写一个重复的测试案例，你应该在测试文件的第一行写：source include/master-slave.inc;。用connection master; 和 connection slave;来切换主案例和从案例。如果你需要对一个替换的连接做点什么，对于主连接，用connection master1;，对于从连接，用connection slave1;。
- 如果需要在一个循环里做点什么，可以用些这样的内容：

```
let $1=1000;
while ($1)
{
 # do your queries here
 dec $1;
}
```

- 在查询之间休眠，使用**sleep**命令。此命令支持几分之几秒，所以，例如你想要休眠1.3秒，你可以使用**sleep 1.3;** 命令。
- 对你的测试案例要运行带附加选项的从案例，以命令行方式把它们放在mysql-test/t/test_name-slave.opt。对于主案例，把它们放在mysql-test/t/test_name-master.opt。
- 如果对测试套件有问题，和想要献出一个测试案例，发送邮件信息到MySQL 内部插件 邮件列表。请参阅1.7.1.1节，"MySQL邮件列表"。 虽然这个列表不接受附件，你可以把相关文件通过ftp上传到：ftp://ftp.mysql.com/pub/mysql/upload/

27.1.2.3. 在MySQL测试套件中报告缺陷

如果你的MySQL的版本没有通过测试套件，你可以采取如下措施：

- 在尽可能多地找到出错之时的错误之前，不要发送缺陷报告。查找之时，请使用**mysqlbug**脚本比便我们能获取你的系统和MySQL版本信息，参阅1.7.1.3节 ，"如何报告缺陷或问题"。
- 确保包含了**mysql-test-run**的输出，以及 mysql-test/r目录下所有.reject文件的内容。
- 如果测试套件里的测试未通过，用如下命令检查一下看它自己运行时是否通过测试：

```
cd mysql-test
mysql-test-run --local test-name
```

  如果未能通过，你应该用 --with-debug 配置MySQL并使用--debug选项来运行**mysql-test-run**。如果这样也未能通过，请把追踪文件var/tmp/master.trace 上传到 ftp://ftp.mysql.com/pub/mysql/upload/ 以便我们能检查它。请记得也要包含你系统的完整描述，**mysqld** 二进制文件的版本，以及你是如何编译它的。

- 也试着带--force选项运行一下**mysql-test-run** ，看是否还有别的测试未通过。
- 如果你是自己编译的MySQL，查看我们的手册看看如何在你的平台上编译MySQL，最好用一个在http://dev.mysql.com/downloads/ 上我们已经为你编译好的二进制版本。我们所有标准的二进制版本都能通过测试套件的测试！
- 如果错误是Result length mismatch 或 Result content mismatch ，这意味测试的输出于期望的输出不匹配，这可能是在MySQL或你的**mysqld** 版本里的缺陷在某些环境下产生稍有不同的结果。

未通过的测试结果放在和结果文件同主名但扩展名为.reject的文件里。如果测试案例未通过，你应该对两个文件做diff操作。如果你不能发现它们是如何不同，用od -c 命令检查它们，也检查一下文件长度。

- 如果测试完全未通过，你应该检查mysql-test/var/log目录下的日志文件以获得有关错误的一些提示。
- 如果你是为调试而编译MySQL，试一下带--gdb和（或）--debug参数运行**mysql-test-run** 。请参阅E.1.2节，“创建跟踪文件”。

  如果你没有为调试而编译MySQL，这应该是你可能去做的。只要带－-with-debug参数运行**configure**。请参阅2.8节，“使用源码分发版安装MySQL ”。

27.2. 为MySQL添加新函数

27.2.1. 自定义函数接口的特性
27.2.2. CREATE FUNCTION/DROP FUNCTION 语法
27.2.3. 添加新的自定义函数
27.2.4. 添加新的固有函数

有两个途径来为MySQL添加新函数:

- 你可以通过自行医函数接口 (UDF)来添加函数。自定义函数被编译为目标文件，然后用CREATE FUNCTION 和DROP FUNCTION 声明动态地添入到服务器中及从服务器中移出。参阅27.2.2节，“CREATE FUNCTION/DROP FUNCTION 语法”。
- 你可以将函数添加为MySQL固有（内建）函数。固有函数被编译进**mysqld**服务器中，成为永久可用的。

每种途径都有其优点和缺点:

- 如果你编写自定义函数，你除了安装服务器本身之外还要安装目标文件。如果将你的函数编译进服务器中，你就不需要这么做了。
- 你可以给二进制版本的MySQL分发版添加UDF。固有函数需要你去修正源码分发版。.
- 如果你升级你的MySQL分发版，你可以继续使用先前安装了的UDF， 除非你升级到一个UDF接口改变了的新版本。对固有函数而言，每次升级你都必须重复一次修正。

无论你使用哪种方法去添加新函数，它们都可以被SQL声明调用，就像 ABS() 或 SOUNDEX()这样的固有函数一样。

另一个添加函数的方法时创建存储函数。这些函数时用SQL声明编写的，而不是编译目标代码。编写存储函数的语法在第20章：存储程序和函数 中描述。

下面的小节描述UDF接口的特性，给出编写UDF的指令，并讨论MySQL为防止UDF被误用而采取的安全预防措施。

给出源代码的例子来说明如何编写UDF，看一看MySQL源码分发版中提供的sql/udf_example.cc 文件。

27.2.1. 自定义函数接口的特性

MySQL自定义函数接口有如下特性和功能:

- 函数能分÷返回字符串，整数或实数。

- 你可以定义一次作用于一行的简单函数，或作用于多行的组的集合函数。
- 提供给函数的信息使得函数可以检查传递给它们的参量的数目和类型。
- 你可以让MySQL在将某参量传递给函数之前强制其为某一类型。
- 你可以表示函数返回NULL 或发生错误。

27.2.2. CREATE FUNCTION/DROP FUNCTION 语法

```
CREATE [AGGREGATE] FUNCTION function_name RETURNS {STRING|INTEGER|REAL}
       SONAME shared_library_name

DROP FUNCTION function_name
```

一个自定义函数 (UDF)就是用一个象ABS() 或 CONCAT()这样的固有（内建）函数一样作用的新函数去扩展MySQL。

function_name 是 用在SQL声明中以备调用的函数名字。RETURNS 子句说明函数返回值的类型。*shared_library_name* 是共享目标文件的基本名，共享目标文件含有实现函数的代码。该文件必须位于一个能被你系统的动态连接者搜索的目录里。

你必须有mysql 数据库的INSERT 权限才能创建一个函数，你必须有mysql 数据库的DELETE权限才能撤销一个函数。这是因为CREATE FUNCTION 往记录函数名字，类型和共享名的mysql.func系统表里添加了一行，而DROP FUNCTION则是从表中删掉这一行。如果你没有这个系统表，你应该运行**mysql_fix_privilege_tables**脚本来创建一个。请参阅2.10.2节，“升级授权表”。

一个有效的函数是一个用CREATE FUNCTION加载且没有用DROP FUNCTION移除的函数。每次服务器启动的时候会重新加载所有有效函数，除非你使用--skip-grant-tables参数启动**mysqld**。在这种情况下， 将跳过UDF的初始化，UDF不可用。

要了解编写自定义函数的说明，请参阅27.2.3节，“添加新的自定义函数”。要使得UDF机制能够起作用，必须使用C或者C++编写函数，你的系统必须支持动态加载，而且你必须是动态编译的**mysqld**（非静态）。

一个AGGREGATE函数就像一个MySQL固有的集合（总和）函数一样起作用，比如，SUM或COUNT()函数。要使得AGGREGATE 起作用，你的mysql.func表必须包括一个type列。如果你的mysql.func表没有这一 列，你应该运行**mysql_fix_privilege_tables**脚本来创建此 列。

27.2.3. 添加新的自定义函数

27.2.3.1. UDF对简单函数的调用顺序
27.2.3.2. UDF对集合函数的调用顺序
27.2.3.3. UDF参量处理
27.2.3.4. UDF返回值和错误处理
27.2.3.5. 编译和安装自定义函数
27.2.3.6. 自定义函数安全预报措施

要使得UDF机制能够起作用，必须使用C或者C++编写函数，你的系统必须支持动态加载。MySQL 源码分发版包括一个sql/udf_example.cc 文件，此文件定义了5个新函数。可以参考这个文件，看UDF是如何调用常规工作。

为了能使用UDF，你需要动态链接**mysqld**。不要配置MySQL使用--with-mysqld-ldflags=-all-static参数。如果你想使用一个需要从**mysqld** 访问符号的UDF（例如在使用default_charset_info的sql/udf_example.cc文件中的metaphone函数），你必须使用-rdynamic参数来链接程序（参阅man dlopen）。如果你计划使用UDF, 一个

经验法则就是，用with-mysqld-ldflags=-rdynamic设定MySQL，除非你有很好的理由不去这么做。

如果你使用的是预编译分发版的MySQL， 请使用MySQL-Max，其中含有一个动态链接了的服务器，它可以支持动态加载。

对于每个你想要使用在SQL声明中的函数，你应该定义相应的C （或 C++）函数。在下面的讨论中，xxx用来表示范例函数的名字，为了区分使用SQL还是C/C++，xxx()(上标）表示SQL函数调用，xxx()（下标）表示C/C++函数调用。

你为xxx()编写来实现接口的C/C++ 函数如下:

- xxx() (必有)

  主函数。 这是函数结果被计算的地方。SQL函数数据类型与C/C++函数返回类型的对应关系如下:

SQL 类型	**C/C++** 类型
STRING	char *
INTEGER	long long
REAL	double

- xxx_init() (可选)

  对xxx()的初始化函数。它可以被用来:

  - 检查传递给xxx()的参量数目。
  - 检查参量是否为必需的类型，或者，除此之外，在主函数被调用的时候告诉MySQL将参量强制为想要的类型。
  - 分配主函数需要的内存。
  - 指定结果的最大长度。
  - 指定（对于REAL 函数）小数的最多位数。
  - 指定结果是否可以为 NULL。

- xxx_deinit() （可选）

  对xxx()的去初始化函数。它释放初始化函数分配的内存。

当SQL声明调用XXX()时，MySQL调用初始化函数xxx_init()，让它执行必要的设置，比如，检查 参量或分配内存。如果xxx_init() 返回一个错误，SQL声明会退出并给出错误信息，而主函数和去初始化函数并没有被调用。否则，主函数xxx() 对每一行都被调用一次。所有行都处理完之后，调用去初始化函数xxx_deinit() 执行必要的清除。

对于象SUM()一样工作的集合函数，你也必须提供如下的函数:

- xxx_clear() （在5.1节中必须）

  对一个新组重置当前集合值为初试集合值，但不插入任何参量。

- xxx_add() （必须）

  添加参量到当前集合值。

MySQL 按下列操作来处理集合UDF:

1. 调用 xxx_init() 让集合函数分配它需要用来存储结果的内存。
2. 按照GROUP BY表达式来排序表。
3. 为每个新组中的第一行调用xxx_clear()函数。
4. 为属于同组的每一个新行调用xxx_add()函数。
5. 当组改变时或每组的最后一行被处理完之后，调用xxx()来获取集合结果。
6. 重复，以上3-步直到所有行被处理完。
7. 调用xxx_deinit() 函数去释放UDF分配的内存。.

所有函数必须时线程安全的，这不仅对主函数，对初始化和去初始化函数也一样，也包括集合函数要求的附加函数。这个要求的一个结果就是，你不能分配任何变化的全局或静态变量。如果你需要内存，你可以在xxx_init()函数分配内存，然后在xxx_deinit()函数释放掉。

27.2.3.1. UDF 对简单函数的调用顺序

下面介绍创建简单UDF时需要定义的不同函数。27.2.3节，“添加新的自定义函数”中介绍了MySQL调用这些函数的顺序。

如本节所示，应该说明主函数xxx()。注意返回值和参数会有所不同，这取决于你说明的SQL函数xxx()在CREATE FUNCTION声明中返回的是STRING，INTEGER类型还是REAL类型示:

对于STRING 型函数:

```
char *xxx(UDF_INIT *initid, UDF_ARGS *args,
          char *result, unsigned long *length,
          char *is_null, char *error);
```

对于INTEGER型函数:

```
long long xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

对于REAL型函数:

```
double xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

初始化和去初始化函数如下说明:

```
my_bool xxx_init(UDF_INIT *initid, UDF_ARGS *args, char *message);

void xxx_deinit(UDF_INIT *initid);
```

initid 参数被传递给所有的三个函数。它指向一个UDF_INIT 结构，这个结构被用来在函数之间交换信息。UDF_INIT 结构项跟随着。初始化函数应该给任何它想要改变的项赋值。（要使用项的默认值，就让它不被改变）

- my_bool maybe_null

  如果xxx() 能返回NULL，xxx_init()应maybe_null 为 1 。如果任一参量被说明了 maybe_null值，其 默认

值是1 。

- unsigned int decimals

  小数位数。默认值是传到主函数的参量里小数的最大位数。（例如，如果函数传递 1.34, 1.345, 和1.3, 那么默认值为，因为1.345 有3位小数。

- unsigned int max_length

  结果的最大长度。max_length 的默认值因函数的结果类型而异。对字符串函数，默认值是最长参量的长度。对整型函数，默认是21位。对实型函数，默认是13再加上initid->decimals指示的小数位数。（对数字函数，长度包含正负号或者小数点符）。

  如果想返回团值，你可以把max_length 设为从65KB到16MB。这个内存不会被分配，但是如果有临时数据需要存储，这个设置了的值被用来决定使用哪种 列的类型。

- char *ptr

  函数可以用作本身目的的指针。比如，函数可以用initid->ptr 来在分配了的内存内部通讯。 xxx_init() 应该分配内存，并指派给这个指针：

  ```
  initid->ptr = allocated_memory;
  ```

  在 xxx() 和 xxx_deinit()中，借用 initid->ptr 来使用或分配内存。

27.2.3.2. UDF对集合函数的调用顺序

本节介绍创建集合UDF之时需要定义的不同函数。27.2.3节，“添加新的自定义函数” 介绍了MySQL调用这些函数的顺序。

- xxx_reset()

  当MySQL在一个新组中发现第一行时调用这个函数。它对这个组重置任何内部总和变量，然后使用给定的UDF_ARGS参量作为内部总和值的第一个值。如下说明 xxx_reset() 函数：

  ```
  char *xxx_reset(UDF_INIT *initid, UDF_ARGS *args,
                  char *is_null, char *error);
  ```

  在MySQL5.1版中UDF接口不需要或不使用xxx_reset()函数，而是使用xxx_clear()函数作为替代。但是如果你想让UDF也能在老版本的服务器上运行，你也可以定义 xxx_reset() 和 xxx_clear() 函数。（如果你使用了这两个函数，xxx_reset()函数在很多情况下可以通过调用函数来内部实现，即调用xxx_clear()函数重置所有变量，然后添加UDF_ARGS参量作为组的第一个值。）

- xxx_clear()

  当MySQL需要重置总和结果时调用此函数。对每一个新组，在开始之时调用它，但是它也可以被调用来为一个没有匹配行在其中的查询重置值。如下说明xxx_clear()：

  ```
  char *xxx_clear(UDF_INIT *initid, char *is_null, char *error);
  ```

  在调用xxx_clear()之前is_null 被设置指向 CHAR(0) 。

  如果发生错误，你可以存储一个值在 error参量指向的变量中。error指向一单字节变量，而不是一个字符串缓冲区。

  xxx_clear() 是MySQL 5.1必须的。

- xxx_add()

  为同组除了第一行之外，所有的行调用这个函数。你应该用它在UDF_ARGS参量中向内部总和变量加值。.

```
char *xxx_add(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

对集合UDF而言xxx() 函数应该用与非集合UDF一样的方法来说明。请参阅27.2.3.1节，“UDF调用简单函数的顺序”。

对一个集合UDF，MySQL 在组内所有行被处理之后调用xxx()函数。这里你应该一般不会接触到它的UDF_ARGS参量，但是取而代之地根据内部总和变量返回给你值。

在xxx()中处理的返回值应该用与对非集合UDF一样的方法来操作。请参阅27.2.3.4节，“UDF返回值和错误处理”。

xxx_reset() 和 xxx_add() 函数用与非集合UDF一样的方法来处理它们的UDF_ARGS 参量。请参阅27.2.3.3节，“UDF参量处理”。

到is_null和error的指针 参量和所有到xxx_reset(), xxx_clear(), xxx_add() 和 xxx()调用一样。你可以用这个来提醒你获取一个错误或无论xxx()是否返回NULL的一个结果。你不能把一个字符串存到*error！ error指向单字节变量而不是字符串缓冲区。

*is_null 对每一个组都重置（调用xxx_clear()前）， *error 从不重置。

如果 xxx()返回时，*is_null 或 *error 被设置，MySQL返回 NULL作为组函数的结果。

27.2.3.3. UDF参量处理

args 参数指向列着结构元的 UDF_ARGS 结构:

- unsigned int arg_count

  参量个数。如果你需要你的函数带着某个数目的参量被调用，在初始化函数检查这个值，例如:

```
if (args->arg_count != 2)
{
    strcpy(message,"XXX() requires two arguments");
    return 1;
}
```

- enum Item_result *arg_type

  一个指针，对每个参量指向包含类型的一个数列。可能的类型值是STRING_RESULT, INT_RESULT 和 REAL_RESULT。

  要确信一个参量是给定类型的，并且如果不是的话就返回一个错误，请检查初始化函数中的arg_type数列。比如:

```
if (args->arg_type[0] != STRING_RESULT ||
    args->arg_type[1] != INT_RESULT)
{
    strcpy(message,"XXX() requires a string and an integer");
    return 1;
}
```

  要求你函数的参量是某一类型的另一方法是，使用初始化函数设置arg_type元素为你想要的类型。对所

有对xxx()的调用而言，这会导致MySQL强制参量为这些类型。比如，要指定投两个参量强制成字符串和整数，在xxx_init()中分别：

```
args->arg_type[0] = STRING_RESULT;
args->arg_type[1] = INT_RESULT;
```

- char **args

  args->args 与初始化函数做有关传到你函数的参量的一般情况做通讯。对于常参量i，args->args[i] 指向参量值。（看下面的说明了解如何妥善地访问这个值）。对非－常参量，args->args[i] 为 0。一个常参量为仅使用参量的表达式，如 3 或 4*7-2 或 SIN(3.14)。一个非常 参量是一个行与行不同的表达式，如，列名或带非－常参量调用的函数。

  对主函数的每次调用，args->args 包含为每个当前处理的行传递的实际参量。

  如下使用参量i的函数：

  - 给一个STRING_RESULT 型的参量作为一个字符串加一个长度，可以允许所有二进制数或任意长度的数处理。字符串内容作为args->args[i] 而字符串长度为args->lengths[i]。你不能采用null结尾的字符串。

  - 对一个INT_RESULT型的参量，你必须转换args->args[i] 为一个long long 值：

    ```
    long long int_val;
    int_val = *((long long*) args->args[i]);
    ```

  - 对一个REAL_RESULT型参量，你必须转换args->args[i]为一个双精度值：

    ```
    double    real_val;
    real_val = *((double*) args->args[i]);
    ```

- unsigned long *lengths

  对初始化函数，lengths数列表示对每个参量的最大字符串长度。你不要改变它。对主函数的每次调用，lengths包含了对当前处理行传递的任何字符串 参量的实际长度。对于INT_RESULT 或 REAL_RESULT类型的参量，lengths 仍包含参量的最大长度（对初始化函数）。

27.2.3.4. UDF返回值和错误处理

如果没有错误发生，初始化函数应该返回0，否则就返回1。如果有错误发生，xxx_init() 应该在message 参数存储一个以null结尾的错误消息。该消息被返回给客户端。消息缓冲区是 MYSQL_ERRMSG_SIZE 字符长度，但你应该试着把消息保持在少于80个字符，以便它能适合标准终端屏幕的宽度。

对于long long 和 double 类型的函数，主函数 xxx()的返回返回值是函数值。字符函数返回一个指向结果的指针，并且设置 *result 和 *length 为返回值的内容和长度。例如：

```
memcpy(result, "result string", 13);
*length = 13;
```

被传给 xxx() 函数的结果缓冲区是 255 字节长。如果你的结果适合这个长度，你就不需要担心对结果的内存分配。

如果字符串函数需要返回一个超过255字节的字符串，你必须用 malloc() 在你的 xxx_init() 函数或者 xxx() 函数里为字符串分配空间，并且在 xxx_deinit() 函数里释放此空间。你可以将已分配内存存储在 UDF_INIT 结构里的 ptr 位置以备将来 xxx() 调用。请参阅27.2.3.1节，“UDF 对简单函数的调用顺序”。

要在主函数中指明一个 NULL 的返回值，设置 *is_null 为 1 ：

```
*is_null = 1;
```

要在主函数中指明错误返回，设置 *error 为 1 ：

```
*error = 1;
```

如果 xxx() 对任意行设置 *error 为 1 ，对于任何 XXX()被调用的语句处理的当前行和随后的任意行，该函数值为 NULL （甚至都不为随后的行调用 xxx()）。

27.2.3.5. 编译和安装自定义函数

实现UDF的文件必须在运行服务器的主机上编译和安装。这个步骤在下面介绍，以包含在MySQL源码分发版里的UDF文件sql/udf_example.cc 为例。

紧接着下面的指令是对Unix的，对Windows的指令在本节稍后给出。

udf_example.cc 文件包含下列函数：

- metaphon() 返回字符串参量的一个变音位(metaphon)字符串，这有点象一个探测法（soundex）字符串，但是它英语更协调。
- myfunc_double()返回在其参量中所有字符的ASCII值的和，除以其 参量长度之和。
- myfunc_int()返回其参量长度之和。
- sequence([const int]) 返回一个序列，从给定数开始，若没有给定数则从1开始。
- lookup() 返回对应主机名的IP数。
- reverse_lookup() 返回对应一个IP数的主机名。函数可以带'xxx.xxx.xxx.xxx'形式的一个单字符串 参量调用，要么带4个数字调用。

一个可动态加载的文件应使用如下这样的命令编译为一个可共享的对象文件：

```
shell> gcc -shared -o udf_example.so udf_example.cc
```

如果你使用**gcc**，你应该能用一个更简单的命令创建udf_example.so ：

```
shell> make udf_example.so
```

通过运行MySQL源码树下sql里的如下命令，你可以容易地为你的系统决定正确的编译器选项：

```
shell> make udf_example.o
```

你应该运行一个类似于**make**所显示那样的编译命令，除了要在行尾附近删除-c选项，并在行尾加上加上 -o udf_example.so。（在某些系统上，你可能需要在命令行留着-c 选项）。

编译好一个包含有UDF的共享目标后，你必须安装它并通知MySQL。从udf_example.cc编译一个共享目标文件产生一个名字类似于udf_example.so 的文件（确切名字可能因平台而异）。把这个文件复制到 /usr/lib 这样被你系统的动态（运行时）链接器搜索到的目录下，或者 把你放共享目标文件的目录添加到链接器配置文件（如，/etc/ld.so.conf ）。

动态链接器的名字时系统特定的（如，在FreeBSD上是**ld-elf.so.1** ，在Linux上是 **ld.so**，在Mac OS X上是**dyld** ）。查看一下你系统的文档，看看链接器的名字是什么及如何配置链接器。

在许多系统上，你也可以设置环境变量LD_LIBRARY 或 LD_LIBRARY_PATH 指向你放UDF的目录。dlopen 手册

会告诉你，在你系统上用哪个变量名。你可以在**mysql.server** 或 **mysqld_safe** 启动脚本里设置这个然后重启 **mysqld**。

在一些系统上，配置动态链接器的**ldconfig**不能识别不是以lib做名字开头的共享目标。在这种情况下，你应该把udf_example.so 改名为 libudf_example.so。

在Windows系统上，你可以通过下列步骤编译自定义函数：

1. 你需要获得BitKeeper source repository for MySQL 5.1。 请参阅 2.8.3节，“从开发源树安装”。
2. 在源数据仓里的VC++Files/examples/udf_example目录下，有名为udf_example.def, udf_example.dsp, 和 udf_example.dsw 的文件。
3. 在数据仓的sql目录下，复制 udf_example.cc 文件到 VC++Files/examples/udf_example 目录，并改其名为udf_example.cpp。
4. Visual Studio VC++用打开 udf_example.dsw 文件，用它把UDF编译为一个一般项目。

共享目标文件安装完以后，为新函数信息修改 **mysqld** ，做如下声明：

```
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';
```

可以使用DROP FUNCTION删除函数：

```
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 和 DROP FUNCTION 声明更新mysql 数据库中的func 系统表。函数名，类型和共享库名存进表中。你必须有mysql 数据库的INSERT 和DELETE 权限来创建和移除函数。

你不能使用 CREATE FUNCTION 去田间一个先前已经被创建的函数。如果你需要重新安装一个函数，你可以用DROP FUNCTION移除它，然后再用CREATE FUNCTION重新安装它。你可能会需要这么做，比如你重新编译新版本的函数以便**mysqld**得到这个新版本。不然，服务器还继续使用旧的版本。

一个有效程序是已被 CREATE FUNCTION加载且没有被DROP FUNCTION移除的函数。所有有效函数在每次服务器启动时重新加载，除非你使用--skip-grant-tables选项来启动**mysqld**。这种情况下，UDF的初始化将被跳过，UDF不可用。

27.2.3.6. 自定义函数安全预防措施

MySQL 采取下列措施来防止误用自定义函数。

你必须有 INSERT 权限才能使用 CREATE FUNCTION 及有 DELETE 权限才能使用 DROP FUNCTION。这是很必要的，因为这些声明在mysql.func表里添加合删除行。

除了对应主 xxx()函数的xxx 符号，UDF应该至少定义一个符号。这些辅助符号对应 xxx_init(), xxx_deinit(), xxx_reset(), xxx_clear() 和 xxx_add() 函数。**mysqld** 也支持一个控制仅有一个xxx符号的UDF是否被加载的--allow-suspicious-udfs。这个选项 默认是关，以防止从共享目标文件而不是从这些已包含的合法UDF加载的企

图。如果你有仅含xxx符号的老版本UDF，以及不能重编译来包含辅助符号的老版本UDF，那就有必要选--allow-suspicious-udfs 选项。否则，你应该避免打开这个选项。

UDF 目标文件不能放在任意目录。它们必须位于动态链接器被配置来搜索到的一些系统目录。为强制执行这个限制并防止指定被动态链接器搜索到的目录之外的路径，MySQL在加载函数的时候检查在CREATE FUNCTION中指定的共享目标文件名，以及存在mysql.func表中的文件的路径分隔符。这防止通过直接操作mysql.func表指定非法路径名。有关UDF和运行时链接器，请参阅27.2.3.5节，“编译和安装自定义函数”。

27.2.4. 添加新的固有函数

下面介绍添加新固有函数的步骤。要注意你不能添加固有函数到二进制分发版里，因为这个步骤包含修改MySQL源代码。你必须从源码分发版自己编译MySQL。另外要注意，如果你把MySQL移植到另一个版本（比如新版本放出来的时候），你需要用新版本重复这个添加 步骤。

采取下列步骤来添加MySQL新的固有函数：

1. 在定义函数名的lex.h文件中的sql_functions[]数列里添加一行。

2. 如果函数原型是简单的（只有零个，一个，二个或三个参量），你应该在lex.h中指定 SYM(FUNC_ARG*N*)（其中*N* 是参量的个数）作为sql_functions[]数列中的第二个 参量，并添加一个在item_create.cc中创建函数目标的函数。可以看看 "ABS" 和 create_funcs_abs() 作为举例说明。

   如果函数原型是复杂的（举例，如果函数有多种参量），你应该给sql_yacc.yy添加两行。一行表示**yacc**应该定义的预处理程序记号，（这应该在文件的开始添加）。然后定义函数 参数，并添加一个带这些参数的项到simple_expr分析规则中。举一个例子，你可以检查 sql_yacc.yy 中所有出现的ATAN 看看这个定义是什么样子的。

3. 在 item_func.h中说明一个继承自Item_num_func 还是 Item_str_func的类，取决于你的函数是返回一个数还是一个字符串。

4. 在 item_func.cc中是否添加下列说明之一，取决于你是定义一个数字函数还是字符函数：

   ```
   double   Item_func_newname::val()
   longlong Item_func_newname::val_int()
   String  *Item_func_newname::Str(String *str)
   ```

   如果你从任何标准项继承了你的目标（类似于Item_num_func），你或许只要定义这些函数中的一个，然后让父目标照管别的函数。比如，Item_str_func 类定义了一个 val() 函数，它这个函数对::str()返回的值进行 atof()操作。

5. 你或许也定义了下列目标函数：

   ```
   void Item_func_newname::fix_length_and_dec()
   ```

   这个函数至少应该计算基于给定参量的max_length。 max_length 是函数可能返回字符的最大个数。如果主函数不能返回 NULL值，这个函数也应该设置 maybe_null = 0。函数可以通过检查函数的maybe_null值来检查是否有函数 参量能返回NULL值。你可以看一下Item_func_mod::fix_length_and_dec 作为典型的例子来说明这个问题。

所有函数都必须是线程安全的，换句话说就是，如果没有互斥体保护，不要在函数中使用任何全局或静态变量。

如果你想要从函数::val(), ::val_int()或::str()返回NULL，你应该设null_value为1,并返回0。

对于目标函数 ::str() 有一些需要而外考虑之处：：

- 字符串参量*str 提供一个字符串缓冲可以用来保持结果（更多关于字符串类型的信息请参阅sql_string.h文件）。
- 如果结果为NULL，::str() 函数应该返回保持这个结果的字符串或(char*) 0。
- 除非有绝对地需要，所有当前的字符串函数要避免分配内存！

27.3. 为MySQL添加新步骤

27.3.1. 步骤分析
27.3.2. 编写步骤

在MySQL中，你可以用C++定义一个步骤，在一个查询被发送到客户端之前访问和修改其中的数据。修改可以一行接一行地做，或者按照级别成组（GROUP）地做。

我们创建一个范例步骤来演示你可以做的。

此外，我们推荐你看一下mylua。通过它你可以用 LUA语言把运行时里的一个 步骤加载到**mysqld**中。

27.3.1. 步骤分析

analyse([*max_elements*,[*max_memory*]])

这个步骤在sql/sql_analyse.cc定义，这个步骤检查你查询的结果，并且返回对此结果的一个分析：

- *max_elements* (默认值 256) 是analyse注意到每 列不同值的最高数目。analyse使用此 参数来检查是否最优化的列的类型是ENUM类型。
- *max_memory* (默认值 8192) 是analyse在查找所有不同值时分配给每 列的最大内存数。i

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max_elements,[max_memory]])
```

27.3.2. 编写步骤

当前来说，相关的文档只有源码。

检查下列文件可以获得关于步骤的所有信息：

- sql/sql_analyse.cc
- sql/procedure.h
- sql/procedure.cc
- sql/sql_select.cc

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

附录A：问题和常见错误

目录

A.1. 如何确定导致问题的原因
A.2. 使用MySQL程序时的常见错误

A.2.1. 拒绝访问
A.2.2. 无法连接到[local] MySQL服务器
A.2.3. 客户端不支持鉴定协议
A.2.4. 输入密码时出现密码错误
A.2.5. 主机的host_name被屏蔽
A.2.6. 连接数过多
A.2.7. 内存溢出
A.2.8. MySQL服务器不可用
A.2.9. 信息包过大
A.2.10. 通信错误和失效连接
A.2.11. 表已满
A.2.12. 无法创建文件／写入文件
A.2.13. 命令不同步
A.2.14. 忽略用户
A.2.15. 表tbl_name不存在
A.2.16. 无法初始化字符集
A.2.17. 文件未找到

A.3. 与安装有关的事宜

A.3.1. 与MySQL客户端库的链接问题
A.3.2. 如何以普通用户身份运行MySQL
A.3.3. 与文件许可有关的问题

A.4. 与管理有关的事宜

A.4.1. 如何复位根用户密码
A.4.2. 如果MySQL依然崩溃，应作些什么
A.4.3. MySQL处理磁盘满的方式
A.4.4. MySQL将临时文件储存在哪里
A.4.5. 如何保护或更改MySQL套接字文件/tmp/mysql.sock
A.4.6. 时区问题

A.5. 与查询有关的事宜

A.5.1. 搜索中的大小写敏感性
A.5.2. 使用DATE列方面的问题
A.5.3. 与NULL值有关的问题
A.5.4. 与列别名有关的问题
A.5.5. 非事务表回滚失败
A.5.6. 从相关表删除行
A.5.7. 解决与不匹配行有关的问题
A.5.8. 与浮点比较有关的问题

A.6. 与优化器有关的事宜
A.7. 与表定义有关的事宜

A.7.1. 与ALTER TABLE有关的问题
A.7.2. 如何更改表中的列顺序
A.7.3. TEMPORARY TABLE问题

A.8. MySQL中的已知事宜

A.8.1. MySQL中的打开事宜

在本附录中，列出了一些你可能会遇到的常见问题和错误消息。并介绍了确定故障原因的方法，以及为解决问题所应采取的措施。

A.1. 如何确定导致问题的原因

当你遇到问题时，首先要做的是找出导致问题的程序和设备部件：

· 如果遇到下述征兆之一，或许是因为硬件问题（如内存、主板、CPU或硬盘）或内核问题：

1. 键盘不工作。正常情况下可通过按Caps Lock建进行检查。如果Caps Lock的点亮状态未改变，就需要更换键盘（在此之前，应尝试重启计算机，并检查与键盘相连的所有电缆）。

2. 鼠标指针不移动。

3. 机器未对远程机器的Ping命令做出应答。

4. 与MySQL无关的其他程序工作不正常。

5. 系统意外重启（有问题的用户级程序应不能使系统出现严重问题）。

在该情况下，应检查所有的电缆并运行某些诊断工具，对你的硬件进行检查！此外，还应检查是否有能够解决问题的、适用于你的操作系统的补丁、更新或服务包。请检查所有的库（如glibc）是否是最新的。

使用配备ECC内存的机器以便尽早发现问题总是个好主意。

· 如果键盘已锁定，可从另一台机器登录到你的机器，并执行kbd_mode –a，或许能恢复。

· 请检查系统的日志文件（/var/log/messages或类似物）以找出问题的原因。如果你认为问题出在MySQL中，应检查MySQL的日志文件。请参见5.11节，“MySQL日志文件”。

· 如果你不认为存在硬件问题，应尝试找出导致问题的原因。请使用**top**、**ps**、任务管理器或类似程序，以检查哪个程序占用了所有CPU时间或锁定了机器。

· 使用**top**、**df**或类似程序检查是否内存不够、磁盘空间不足、文件描述符缺乏、或其他关键资源缺少。

· 如果问题是失去控制的进程，应尝试杀死它。如果杀不死进程，或许是因为操作系统中存在缺陷。

如果在检查了所有其他可能性之后，并得出结论问题是由MySQL服务器或MySQL客户端导致的，应创建提供给我方的邮件列表或支持团队的缺陷报告。在缺陷报告中，请详细描述系统的行为，以及你认为发生了什么情况。+3.

还应阐明为什么你认为是MySQL导致了问题。请考虑本章介绍的所有情况。准确阐明当你检查系统时问题是如何出现的。对于程序和日志文件的任何输出和错误消息，请使用“复制和粘贴”方法。

尽量详细描述不工作的程序，以及你所见到的所有征兆。我们过去收到过很多仅说明“系统不工作”的缺陷报告。这不会为我们提供有助于解决问题的信息。

如果程序失败，了解下述信息总是有用的：

· 有嫌疑的程序是否出现了分段故障（是否转储内核）？

· 程序是否占用了所有可用的CPU时间？用**top.**进行检查。让程序运行一段时间，或许能简单地评估某些事是否是计算密集性的。

· 如果问题是因**mysqld**服务器导致的，使用**mysqladmin -u root ping**或**mysqladmin -u root processlist**是否能获得服务器的响应？

· 当你尝试连接到MySQL服务器（例如，mysql）时，客户端程序给出的信息是什么？客户端是否堵塞？是否获得了来自程序的任何输出？

发送缺陷报告时，请遵循1.7.1.2节，“请教问题或通报缺陷”中给出的说明。

A.2. 使用MySQL程序时的常见错误

A.2.1. 拒绝访问
A.2.2. 无法连接到[local] MySQL服务器
A.2.3. 客户端不支持鉴定协议
A.2.4. 输入密码时出现密码错误
A.2.5. 主机的host_name被屏蔽
A.2.6. 连接数过多
A.2.7. 内存溢出
A.2.8. MySQL服务器不可用
A.2.9. 信息包过大
A.2.10. 通信错误和失效连接
A.2.11. 表已满
A.2.12. 无法创建文件／写入文件
A.2.13. 命令不同步
A.2.14. 忽略用户
A.2.15. 表tbl_name不存在
A.2.16. 无法初始化字符集
A.2.17. 文件未找到

本节列出了用户运行MySQL服务器时常会遇到的一些错误。尽管问题是在你尝试运行客户端时出现的，但对很多问题的解决方案来说，需要更改MySQL服务器的配置。

A.2.1. 拒绝访问

导致拒绝访问错误的原因很多。该错误常与连接时服务器允许客户端使用的MySQL账户有关。请参见5.7.8节，“拒绝访问错误的原因”。请参见5.7.2节，“权限系统工作原理”。

A.2.2. 无法连接到[local] MySQL服务器

A.2.2.1. 在Windows上与MySQL服务器的连接失败

Unix平台上的MySQL客户端能够以两种不同的方式连接到**mysqld**服务器：通过文件系统中的文件（默认为/tmp/mysql.sock）使用Unix套接字进行连接，或通过端口号使用TCP/IP进行连接。Unix套接字文件的连接速度比TCP/IP快，但仅能在与相同计算机上的服务器相连时使用。如果未指定指定主机名或指定了特殊的主机名localhost，将使用Unix套接字。

如果MySQL服务器运行在Windows 9x或Me上，仅能通过TCP/IP进行连接。如服务器运行在Windows NT、2000、XP或2003上，而且使用--enable-named-pipe选项启动，如果在运行服务器的机器上运行客户端，也能使用命名管道进行连接。默认情况下，命名管道的名称为MySQL。如果在连接到mysqld时未给定主机名，MySQL客户端首先会尝试连接到命名管道。如果不能工作，将连接到TCP/IP端口。使用"."作为主机名，可在Windows平台上强制使用命名管道。

错误(2002)"无法连接到..."通常意味着在系统没有运行的MySQL服务器，或在连接到服务器时使用了不正确的Unix套接字文件名或TCP/IP端口号。

首先检查服务器主机上是否有名为mysqld的进程（在Unix平台上使用**ps xa | grep mysqld**，或在Windows平台上使用任务管理器）。如果没有这类进程，应启动服务器。请参见2.9.2.3节，"启动MySQL服务器以及其故障诊断和排除"。

如果**mysqld**进程正在运行，可使用下述命令检查。在你的具体设置中，端口号或Unix套接字文件名可能会有所不同。host_ip代表运行服务器的机器的IP编号。

```
shell> mysqladmin version

shell> mysqladmin variables

shell> mysqladmin -h `hostname` version variables

shell> mysqladmin -h `hostname` --port=3306 version

shell> mysqladmin -h host_ip version

shell> mysqladmin --protocol=socket --socket=/tmp/mysql.sock version
```

注意，应与主机名命令一起使用"`"而不是"'"，这会使主机名输出（当前主机名）被代入**mysqladmin**命令。如果没有主机名命令或正运行在Windows平台上，应以手动方式输入机器的主机名（无"`"符号），后跟-h选项。也可以使用TCP/IP协议用-h 127.0.0.1连接到本地主机。

下面给出了一些"无法连接到本地MySQL服务器"错误的可能原因：

1. **Mysqld**未运行。请检查操作系统的进程列表以确保**mysqld**进程正在运行。

2. 你正在具有很多TCP/IP连接的Windows平台上运行MySQL服务器。如果你的客户端经常出现错误，请参见A.2.2.1节，"在Windows上与MySQL服务器的连接失败"，以找出规避方法。

3. 你正在使用MIT-pthreads的系统上运行。如果你正在运行不具有固有线程的操作系统，**mysqld**将使用MIT-pthreads软件包。请参见2.1.1节，"MySQL支持的操作系统"。但是，并非所有的MIT-pthreads版本均支持Unix套接字文件。在不支持套接字文件的系统上，连接到服务器时，必须明确指定主机名。请使用下述命令来检查是否连接到了服务器：

4.
```
shell> mysqladmin -h `hostname` version
```

5. 某人移动了**mysqld**使用的Unix套接字文件（默认为/tmp/mysql.sock）。例如，你可能执行了将旧文件从/tmp目录删除的**cron**任务。你总能执行**mysqladmin version**来检查**mysqladmin**试图使用的Unix套接字文件是否的确存在。在该情况下，更正方式是更改**cron**任务，不删除mysql.sock文件，或将套接字文件置于其他地方。请参见A.4.5节，"如何保护或更改MySQL套接字文件/tmp/mysql.sock"。

6. 你使用--socket=/path/to/socket选项启动了**mysqld**服务器，当忘记将套接字文件的新名称通知客户端程序。如果更改了关于服务器的套接字路径，也必须通知MySQL客户端。可在运行客户端程序时使用相同的—socket选项来完成该任务。此外，你还应确保客户端具有访问文件mysql.sock的权限。要想找出套接字文件的位置，可使用：

7.
```
shell> netstat -ln | grep mysql
```

请参见A.4.5节，"如何保护或更改MySQL套接字文件/tmp/mysql.sock"。

8. 你正在使用Linux而且1个服务器线程已死亡（内核已清除）。在此情况下，在重启MySQL服务器之前，必须杀死其他**mysqld**线程（例如，使用kill或mysql_zap脚本）。请参见A.4.2节，“如果MySQL依然崩溃，应作些什么”。

9. 服务器或客户端程序不具有访问包含Unix套接字文件的目录或套接字文件本身的恰当权限。在该情况下，必须更改目录或套接字文件的访问权限，以便服务器或客户端程序能够访问它们，或用–socket选项重启**mysqld**，在该选项中指定服务器能创建、而且客户端可访问的目录下的套接字文件名。

如果遇到错误消息“无法连接到some_host上的MySQL服务器”，可尝试采取下述步骤以找出问题所在：

- 执行“telnet some_host 3306”并按两次回车键，检查服务器是否运行在该主机上（3306是默认的MySQL端口号。如果你的服务器正在监听不同的端口，请更改该值）。如果有1个MySQL服务器正在运行并监听该端口，你应收到包含服务器版本号的回应。如果遇到错误，如“telnet:无法连接到远程主机:拒绝连接”，表示在该定端口上没有运行的服务器。

- 如果服务器正运行在本地主机上，请使用Unix套接字文件，并使用**mysqladmin -h localhost variables**进行连接。验证服务器监听的TCP/IP端口号（它是port变量的值）。

- 确保你的**mysqld**服务器未用--skip-networking选项启动。如果使用了该选项，将无法使用TCP/IP连接到它。

- 检查并确认不存在屏蔽了对MySQL访问的防火墙。需要配置诸如ZoneAlarm和Windows XP个人防火墙等应用程序，以允许对MySQL服务器的外部访问。

A.2.2.1. 在Windows上与MySQL服务器的连接失败

当你在具有很多TCP/IP连接的Windows上运行MySQL服务器，并经常在客户端上遇到“无法连接到MySQL服务器”错误时，可能是因为Windows不允许足够的临时（短命）端口用于这类连接。

默认情况下，Windows允许用于使用5000个临时（短命）TCP端口。任何端口关闭后，它将在TIME_WAIT状态保持120秒。与重新初始化全新的连接相比，该状态允许以更低的开销重新使用连接。但是，在该时间逝去前，无法再次使用该端口。

对于小的可用TCP端口堆栈（5000），以及具有TIME_WAIT状态的大量在短时间内打开和关闭的TCP端口，你很可能遇到端口耗尽问题。处理该问题的方法有两种：

- 通过调查连接池以及可能的持久连接，减少快速消耗的TCP端口数。

- 调整Windows注册表中的某些设置（请参见下面）。

要点：下述步骤涉及更改**Windows** 注册表。更改注册表之前，请备份注册表，并确认你已掌握在出现问题时恢复注册表的方法。关于备份年、恢复和编辑注册表的更多信息，请请参见**Microsoft**知识库中的下述文献：**http://support.microsoft.com/kb/256986/EN-US/**。

- 启动注册表编辑器（Regedt32.exe）。

- 在注册表中确定下述键值的位置：

- `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters`

- 在“编辑”菜单上点击“添加值”，然后增加下述注册值：

- `Value Name: MaxUserPort`

- `Data Type: REG_DWORD`

- Value: 65534

它用于设置为任何用户提供的临时端口数。有效范围介于5000和65534之间（十进制）。默认值为0x1388（5000，十进制）。

- 在"编辑"菜单上点击"添加值"，然后增加下述注册值：

- Value Name: TcpTimedWaitDelay
- Data Type: REG_DWORD
- Value: 30

它用于设置关闭之前将TCP端口连接保持在TIME_WAIT状态的秒数。有效范围介于0秒和300秒之间。默认值为0x78（120秒）。

- 退出注册表编辑器。
- 重新引导机器。

注释：撤销上述设置十分简单，就像删除你创建的注册表一样。

A.2.3. 客户端不支持鉴定协议

MySQL 5.1采用了基于密码混编算法的鉴定协议，它与早期客户端（4.1之前）使用的协议不兼容。如果你将服务器升级到4.1之上，用早期的客户端进行连接可能失败，并给出下述消息：

```
shell> mysql
客户端不支持服务器请求的鉴定协议：请考虑升级MySQL客户端。
```

要想解决该问题，应使用下述方法之一：

- 升级所有的客户端程序，以使用4.1.1或更新的客户端库。
- 用4.1版之前的客户端连接到服务器时，请使用仍具有4.1版之前风格密码的账户。
- 对于需要使用4.1版之前的客户端的每位用户，将密码恢复为4.1版之前的风格。可以使用SET PASSWORD语句和OLD_PASSWORD()函数完成该任务：

```
mysql> SET PASSWORD FOR
    -> 'some_user'@'some_host' = OLD_PASSWORD('newpwd');
```

也可以使用UPDATE和FLUSH PRIVILEGES：

```
mysql> UPDATE mysql.user SET Password = OLD_PASSWORD('newpwd')
    -> WHERE Host = 'some_host' AND User = 'some_user';
mysql> FLUSH PRIVILEGES;
```

用你打算使用的密码替换前例中的"*newpwd*"。MySQL不能告诉你原来的密码是什么，因此，你需要选择新的密码。

- 通知服务器使用旧的密码混编算法：

1. 使用"--old-passwords"选项启动**mysqld**。
2. 对于已将密码更新为较长4.1格式的每个账户，为其指定具有旧格式的密码。可以使用下述查询确定这些

账户：

```
3.                  mysql> SELECT Host, User, Password FROM mysql.user
4.                      -> WHERE LENGTH(Password) > 16;
```

对于查询显示的每个账户记录，请使用Host和User值，并使用OLD_PASSWORD()函数以及SET PASSWORD或UPDATE之一指定密码，如前面所介绍的那样。

注释：在早期的PHP版本中，mysql扩展不支持MySQL 4.1.1和更高版中的鉴定协议。无论使用的PHP版本是什么，它均是正确的。如果你打算与MySQL 4.1或更高版本一起使用mysql扩展，需要使用前面介绍的选项之一，配置MySQL，以便与较早的客户端一起使用。mysqli扩展（支持"改进的MySQL"，在PHP 5中增加）与MySQL 4.1和更高版本中使用的改进的密码混编算法兼容，不需要对MySQL进行特殊配置就能使用该MySQL客户端库。关于mysqli扩展的更多信息，请参见http://php.net/mysqli。

关于密码混编和鉴定功能的额外背景知识，请参见5.7.9节，"MySQL 4.1中的密码哈希处理"。

A.2.4. 输入密码时出现密码错误

使用无下述密码值的"—password"或"-p"选项调用时，MySQL客户端程序将提示输入密码：

```
shell> mysql -u user_name -p
Enter password:
```

在某些系统上，当你在选项文件或命令行上指定时，你可能会发现密码能够工作，但是当你在"Enter password:"提示下以交互方式输入密码时，你可能会发现输入的密码不工作。当系统所提供的用于读取密码的库将密码值限定在少数字符时（典型情况下为8个），就会出现该问题。这是与系统库有关的问题，与MySQL无关。要想处理该问题，可将MySQL密码更改为由8个字符或更少字符构成的值，或将密码置于选项文件中。

A.2.5. 主机的*host_name*被屏蔽

如果遇到下述错误，表示**mysqld**已收到来来自主机"*host_name*"的很多连接请求，但该主机却在中途中断。

```
由于出现很多连接错误，主机'host_name'被屏蔽。
可使用'mysqladmin flush-hosts'解除屏蔽。
```

允许的中断连接请求的数目由max_connect_errors系统变量的值决定。当超出max_connect_errors规定的连接请求时，**mysqld**将认为某处出错（例如，某人正试图插入），并屏蔽主机的进一步连接请求，直至执行了**mysqladmin flush-hosts**命令，或发出了FLUSH HOSTS语句为止。请参见5.3.3节，"服务器系统变量"。

在默认情况下，**mysqld**会在10次连接错误后屏蔽主机。你可以通过下述方式启动服务器来调整该值：

```
shell> mysqld_safe --max_connect_errors=10000 &
```

如果在给定主机上遇到该错误，首先应核实该主机的TCP/IP连接是否正确。如果存在网络问题，增加max_connect_errors变量的值不会有任何好处。

A.2.6. 连接数过多

当你试图连接到**mysqld**服务器时遇到"过多连接"错误，这表示所有可用的连接均已被其他客户端使用。

允许的连接数由max_connections系统变量控制。默认值为100。如果需要支持更多的连接，应使用该变量的较大值重启**mysqld**。

mysqld实际上允许max_connections+1个客户端进行连接。额外的连接保留给具有SUPER权限的账户。通过为系统管理员而不是普通用户授予SUPER权限（普通用户不应具有该权限），系统管理员能够连接到服务器，并使用SHOW PROCESSLIST来诊断问题，即使已连接的无特权客户端数已达到最大值也同样。请参见13.5.4.16节，“SHOW PROCESSLIST语法”。

MySQL能支持的最大连接数取决于给定平台上线程库的质量。Linux或Solaris应能支持500-1000个并发连接，具体情况取决于RAM容量，以及客户端正在作什么。MySQL AB提供的静态Linux库能支持高达4000个连接。

A.2.7. 内存溢出

如果使用**mysql**客户端程序发出了查询，并收到下述错误之一，则表示**mysql**没有足够内存来保存全部查询结果：

```
mysql: Out of memory at line 42, 'malloc.c'

mysql: needed 8136 byte (8k), memory in use: 12481367 bytes (12189k)

错误2008: MySQL client ran out of memory
```

要想更正该问题，首先应检查查询是否正确。返回这么多行是否合理？如果不合理，更正查询并再次尝试。否则，应使用“--quick”选项调用**mysql**。这样，将使用mysql_use_result() C API函数来检索结果集，这类函数能够降低客户端上的负载（但会加重服务器上的负载）。

A.2.8. MySQL服务器不可用

在本节中，还介绍了出现查询错误期间，与丢失了服务器连接有关的事宜。

MySQL服务器不可用错误的最常见原因是服务器超时以及连接已关闭。在该情况下，通常能见到下述错误代码之一（具体的错误代码与操作系统有关）：

错误代码	描述
CR_SERVER_GONE_ERROR	客户端无法将问题发送至服务器。
CR_SERVER_LOST	写入服务器时客户端未收到错误，但也未获得问题的完整答案（或任何答案）。

在默认情况下，如果未发生任何事，8小时后服务器将关闭连接。也可以在启动**mysqld**时，通过设置wait_timeout变量更改时间限制。请参见5.3.3节，“服务器系统变量”.

如果有1个脚本，你仅需要再次发出查询，让客户端再次进行自动连接即可。其中，假定在客户端中启用了自动再连接功能（对于mysql命令行客户端，这是默认设置）。

MySQL服务器不可用错误的一些其他常见原因如下：

- 你（或db系统管理员）使用KILL语句或**mysqladmin kill**命令杀死了正在运行的线程。
- 你试图在关闭了与服务器的连接后运行查询。这表明应更正应用程序中的逻辑错误。
- 你在客户端一侧遇到TCP/IP连接超时错误。如果你使用了命令：mysql_options(..., MYSQL_OPT_READ_TIMEOUT,...)或mysql_options(..., MYSQL_OPT_WRITE_TIMEOUT,...)，就可能出现该问题。在该情况下，增加超时值可能有助于问题的解决。
- 你在服务器端遇到超时错误，而且禁止了客户端中的自动再连接功能（MYSQL结构中的再连接标志等于0）。
- 你正在使用Windows客户端，而且在发出命令之前服务器撤销了连接（或许是因为已超

过wait_timeout）。

在Windows平台上出现问题的原因，在某些情况下，将TCP/IP连接写入服务器时，MySQL未收到来自操作系统的错误，但当试图从连接读取答案时出现错误。

在该情况下，即使MYSQL结构中的再连接标志等于1，MySQL也不会执行自动再连接并再次发出查询，这是因为它不知道服务器是否收到原始查询。

对此的解决方式是：如果自上一次查询以来经过了较长时间，在连接上执行mysql_ping（正是MyODBC所作的）；或在**mysqld**服务器上将wait_timeout设置得很高，使之实际上不存在超时。

- 如果你向服务器发出了不正确或过大的查询，也会遇到这类问题。如果**mysqld**收到过大或无序的信息包，它会认为客户端出错，并关闭连接。如果需要执行较大的查询（例如，正在处理大的BLOB列），可通过设置服务器的max_allowed_packet变量，增加查询限制值，该变量的默认值为1MB。或许，你还需增加客户端上的最大信息包大小。关于设置信息包大小的更多信息，请参见A.2.9节，“信息包过大”。

- 如果你的客户端低于4.0.8而且你的服务器高于4.0.8，当你接收16MB或更大的信息包时，可能会丢失连接。

- 如果MySQL是用“--skip-networking”选项启动的，也会见到MySQL服务器不可用错误。

- 你遇到了执行查询时服务器宕机的缺陷。

通过执行**mysqladmin version**并检查服务器的正常工作时间，可检查服务器是否宕机并重启。如果客户端连接是因**mysqld**崩溃和重启而断开的，应将重点放在查找崩溃你方面。首先应再次检查发出的查询是否再次杀死了服务器。请参见A.4.2节，“如果MySQL依然崩溃，应作些什么”。

用“--log-warnings=2”选项启动mysqld，可获得关于连接的更多信息。这样，就能将某些断开连接错误记录到hostname.err文件中。请参见5.11.1节，“错误日志”。

如果你打算创建与该问题有关的缺陷报告，务必包含下述信息：

1. 指明MySQL服务器是否宕机。通过服务器错误日志可发现这方面的信息。请参见A.4.2节，“如果MySQL依然崩溃，应作些什么”。

2. 如果特定查询杀死了**mysqld**，而且在运行查询前用CHECK TABLE检查了涉及的表，你是否能提供可重复的测试范例？请参见E.1.6节，“如果出现表崩溃，请生成测试案例”。

3. 在MySQL服务器中，系统变量wait_timeout的值是什么？（**mysqladmin variables**给出了该变量的值）。

4. 你是否尝试使用“--log”选项来运行**mysqld**，以确定是否在日志中出现问题？

另请参见A.2.10节，“通信错误和失效连接”。

请参见1.7.1.2节，“请教问题或通报缺陷”。

A.2.9. 信息包过大

通信信息包是发送至MySQL服务器的单个SQL语句，或发送至客户端的单一行。

在MySQL 5.1服务器和客户端之间最大能发送的可能信息包为1GB。

当MySQL客户端或**mysqld**服务器收到大于max_allowed_packet字节的信息包时，将发出“信息包过大”错误，并关闭连接。对于某些客户端，如果通信信息包过大，在执行查询期间，了能回遇到“丢失与MySQL服务器的

连接”错误。

客户端和服务器均有自己的max_allowed_packet变量，因此，如你打算处理大的信息包，必须增加客户端和服务器上的该变量。

如果你正在使用**mysql**客户端程序，其max_allowed_packet变量的默认值为16MB。要想设置较大的值，可用下述方式启动**mysql**：

```
mysql> mysql --max_allowed_packet=32M
```

它将信息包的大小设置为32MB。

服务器的默认max_allowed_packet值为1MB。如果服务器需要处理大的查询，可增加该值（例如，如果准备处理大的BLOB列）。例如，要想将该设置为16MB，可采用下述方式启动服务器：

```
mysql> mysqld --max_allowed_packet=16M
```

也能使用选项文件来设置max_allowed_packet。要想将服务器的该变量设置为16MB，可在选项文件中增加下行内容：

```
[mysqld]
max_allowed_packet=16M
```

增加该变量的值十分安全，这是因为仅当需要时才会分配额外内存。例如，仅当你发出长查询或**mysqld**必须返回大的结果行时**mysqld**才会分配更多内存。该变量之所以取较小默认值是一种预防措施，以捕获客户端和服务器之间的错误信息包，并确保不会因偶然使用大的信息包而导致内存溢出。

如果你正是用大的BLOB值，而且未为**mysqld**授予为处理查询而访问足够内存的权限，也会遇到与大信息包有关的奇怪问题。如果怀疑出现了该情况，请尝试在**mysqld_safe**脚本开始增加**ulimit -d 256000**，并重启**mysqld**。

A.2.10. 通信错误和失效连接

对于连接问题，服务器错误日志是有用的信息源。请参见5.11.1节，“错误日志”。如果服务器是用“--log-warnings”选项启动的，在错误日志中可能会发现下述消息：

```
010301 14:38:23  Aborted connection 854 to db: 'users' user: 'josh'
```

如果“Aborted connections”（放弃连接）消息出现在错误日志中，可能的原因是：

1. 客户端程序在退出之前未调用mysql_close()。

2. 客户端的空闲时间超过wait_timeout或interactive_timeout秒，未向服务器发出任何请求。请参见5.3.3节，“服务器系统变量”。

3. 客户端在数据传输中途突然结束。

出现这类情况时，服务器将增加“Aborted_clients”（放弃客户端）状态变量。

出现下述情况时，服务器将增加“Aborted_clients”（放弃客户端）状态变量。

- 客户端不具有连接至数据库的权限。
- 客户端采用了不正确的密码。

- 连接信息包不含正确信息。
- 获取连接信息包的时间超过connect_timeout秒。请参见5.3.3节，“服务器系统变量”。

如果出现这类情况，可能表明某人正试图侵入你的服务器！

对于放弃客户端或放弃连接问题，其他可能的源应包括：

- 与Linux一起使用以太网协议，半双工或全双工。很多Linux以太网驱动均存在该缺陷。应通过FTP在客户端和服务器机器之间传输大文件来测试该缺陷。如果传输处于burst-pause-burst-pause（爆发-暂停-爆发-暂停）模式，表明你遇到了Linux双工故障。唯一的解决方法是，将网卡和Hub/交换器的双工模式切换为全双工或半双工，并对结果进行测试以确定最佳设置。
- 与线程库有关的某些问题导致读取中断。
- 配置不良的TCP/IP。
- 有问题的以太网、Hub、交换器、电缆等。仅能通过更换硬件才能恰当诊断。
- 变量max_allowed_packet过小或查询要求的内存超过为**mysqld**分配的内存。请参见A.2.9节，“信息包过大”。

另请参见A.2.8节，“MySQL服务器不可用”。

A.2.11. 表已满

表已满错误出现的方式有数种：

- 你正在使用低于3.23版的MySQL服务器，而且“内存中”临时表超过了tmp_table_size字节。要想避免该问题，可使用“-O tmp_table_size=*val*”选项以便**mysqld**增加临时表的大小，或在发出有问题的查询之前，使用SQL选项SQL_BIG_TABLES。请参见13.5.3节，“SET语法”。

也可以使用“--big-tables”选项启动**mysqld**。它与使用针对所有查询的SQL_BIG_TABLES完全相同。

自MySQL 3.23起，该问题应不再出现。如果“内存中”临时表超过tmp_table_size，服务器会自动将其转换为基于磁盘的MyISAM表。

- 你正在使用InnoDB表，并超出了InnoDB表空间。在该情况下，解决方法是增加InnoDB表空间。请参见15.2.7节，“添加和删除InnoDB数据和日志文件”。
- 你正在仅支持2GB文件的操作系统上使用ISAM或MyISAM表，数据文件或索引文件达到了该限制值。
- 你正在使用MyISAM表，而且表所需的空间超过内部指针允许的大小。如果在创建表时未指定MAX_ROWS表，MySQL将使用myisam_data_pointer_size系统变量。默认值为6字节，它足以容纳65536TB数据。请参见5.3.3节，“服务器系统变量”。

使用该语句，可检查最大数据/索引大小：

```
SHOW TABLE STATUS FROM database LIKE 'tbl_name';
```

也可以使用**myisamchk -dv /path/to/table-index-file**。

如果指针大小过小，可使用ALTER TABLE更正该问题：

```
ALTER TABLE tbl_name MAX_ROWS=1000000000 AVG_ROW_LENGTH=nnn;
```

仅应为具有BLOB或TEXT列的表指定AVG_ROW_LENGTH。在该情况下，MySQL不能仅根据行数优化所需的空间。

A.2.12. 无法创建文件／写入文件

如果对某些查询遇到下述类型的错误，它意味着MySQL不能为临时目录下的结果集创建临时文件：

```
无法创建/写入文件'\\sqla3fe_0.ism'。
```

前述错误是Windows平台上的典型消息，Unix平台上的消息与之类似。

一种更正方式是使用"--tmpdir"选项启动**mysqld**，或在选项文件的[mysqld]部分增加该选项。例如，要想指定目录C:\temp，可使用：

```
[mysqld]
tmpdir=C:/temp
```

目录C:\temp必须存在，并有足够的空间允许MySQL写入它。请参见4.3.2节，"使用选项文件"。

该错误的另一个原因可能是许可事宜。请确认MySQL服务器能够写入tmpdir目录。

此外，还用使用**perror**检查错误代码。服务器无法写入表的一个原因是文件系统已满。

```
shell> perror 28
错误代码28：磁盘上无剩余空间。
```

A.2.13. 命令不同步

如果遇到"命令不同步"错误，将无法在你的客户端代码中运行该命令，你正在以错误顺序调用客户端函数。

例如，如果你正使用mysql_use_result()，并打算在调用mysql_free_result()之前执行新查询，就会出现该问题。如果你试图执行两次查询，但并未在两次查询之间调用mysql_use_result()或mysql_store_result()，也会出现该问题。

A.2.14. 忽略用户

如果遇到下述错误，表示当启动**mysqld**时或重新加载授权表时，在用户表中发现具有非法密码的账户。

发现用户'*some_user*'@'*some_host*'密码错误：忽略用户。

作为其结果，许可系统将简单忽略账户。

在下面的介绍中，指明了可能的原因和问题的更正措施：

1. 或许，你正打算用旧的用户表运行新版本的**mysqld**。执行**mysqlshow mysql user**检查Password（密码）列是否短于16个字符，通过该方式可检查该问题。如果结果是肯定的，可运行脚本/add_long_password脚本更正该问题。

2. 账户具有旧的密码（8字符长），而且未使用"--old-protocol"选项启动**mysqld**。更新用户表中的账户，使之具有新的密码，或使用"--old-protocol"选项重启**mysqld**。

3. 在用户表中未使用PASSWORD()函数指定了密码。使用**mysql**用新密码更新用户表中的账户，务必使用PASSWORD()函数：

```
4.          mysql> UPDATE user SET Password=PASSWORD('newpwd')
5.              -> WHERE User='some_user' AND Host='some_host';
```

A.2.15. 表tbl_name不存在

如果遇到下述错误之一，通常意味着当前数据库中不存在具有给定名称的表：

表'*tbl_name*'不存在

无法找到文件：'*tbl_name*' (errno: 2)

在某些情况下，表或许存在，但未正确引用它：

- 由于MySQL使用目录和文件来保存数据库和表，如果它们位于区分文件名大小写的文件系统上，数据库和表名也区分文件大小写。
- 即使对于不区分大小写的文件系统，如Windows，在查询内对给定表的所有引用必须使用相同的大小写。

可以使用SHOW TABLES检查位于当前数据库中的表。请参见13.5.4节，“SHOW语法”。

A.2.16. 无法初始化字符集

如果存在字符集问题，可能会遇到下述错误：

MySQL连接失败：无法初始化字符集charset_name。

导致该错误的原因：

- 字符集为多字节字符集，但客户端不支持该字符集。在该情况下，需要使用“--with-charset=*charset_name*”或“--with-extra-charsets=*charset_name*”选项运行**configure**以重新编译客户端。请参见2.8.2节，“典型配置选项”。

所有的标准MySQL二进制文件均是采用“--with-extra-character-sets=complex”编译的，能够支持所有的多字节字符集。请参见5.10.1节，“数据和排序用字符集”。

- 字符集是未编译到**mysqld**中的简单字符集，而且字符集定义文件不在客户端预期的位置。

在该情况下，需要采取下述方法之一解决问题：

1. 重新编译客户端，使之支持字符集。请参见2.8.2节，“典型配置选项”。
2. 为客户端指定字符集定义文件所在的目录。对于很多客户端，可使用“--character-sets-dir”选项完成该任务。
3. 将字符集定义文件复制到客户端预期的位置。

A.2.17. 文件未找到

如果遇到“ERROR '...'未发现(errno: 23)”，“无法打开文件：... (errno: 24)”，或来自MySQL的具有errno 23或errno 24的其它错误，它表示未为MySQL服务器分配足够的文件描述符。你可以使用**perror**实用工具来了解错误编号的含义：

```
shell> perror 23
```

错误代码23：文件表溢出

```
shell> perror 24
```

错误代码24：打开文件过多

```
shell> perror 11
```

错误代码11：资源暂时不可用

这里的问题是，**mysqld**正试图同时打开过多的文件。你可以通知**mysqld**不要一次打开过多文件，或增加**mysqld**可用文件描述符的数目。

要想通知**mysqld**将一次打开的文件控制在较小的数目上，可降低table_cache系统变量的值（），从而减少表高速缓冲（默认值为64）。降低max_connections的值也能降低打开文件的数目（默认值为100）。

要想更改**mysqld**可用的文件描述符的数目，可在**mysqld_safe**上使用"--open-files-limit"选项或设置（自MySQL 3.23.30开始）open_files_limit系统变量。请参见5.3.3节，"服务器系统变量"。设置这些值的最简单方式是在选项文件中增加1个选项。请参见4.3.2节，"使用选项文件"。如果**mysqld**的版本较低，不支持设置打开文件的数目，可编辑**mysqld_safe**脚本。在脚本中有1个注释掉的行**ulimit -n 256**。你可以删除'#'字符取消对该行的注释，更改数值256，以设置**mysqld**可用的文件描述符数目。

"--open-files-limit"和**ulimit**能够增加文件描述符的数目，但最高不能超过操作系统限制的数目。此外还有1个"硬"限制，仅当以根用户身份启动**mysqld_safe**或**mysqld**时才能覆盖它（请记住，在该情况下，还需使用"--user"选项启动服务器，以便在启动后不再以根用户身份继续运行）。如果需要增加操作系统限制的对各进程可用文件描述符的数目，请参阅系统文档。

注释：如果运行**tcsh** shell，**ulimit**不工作！请求当前限制值时，**tcsh**还能通报不正确的值。在该情况下，应使用**sh**启动**mysqld_safe**。

A.3. 与安装有关的事宜

A.3.1. 与MySQL客户端库的链接问题
A.3.2. 如何以普通用户身份运行MySQL
A.3.3. 与文件许可有关的问题

A.3.1. 与MySQL客户端库的链接问题

当你链接到应用程序以使用MySQL客户端库时，可能会遇到以mysql_开始的未定义引用错误，如下所示：

```
/tmp/ccFKsdPa.o: 在函数`main'中:
/tmp/ccFKsdPa.o(.text+0xb): 对`mysql_init'的未定义引用。
/tmp/ccFKsdPa.o(.text+0x31): 对`mysql_real_connect'的未定义引用。
/tmp/ccFKsdPa.o(.text+0x57): 对`mysql_real_connect'的未定义引用。
/tmp/ccFKsdPa.o(.text+0x69): 对`mysql_error'的未定义引用。
/tmp/ccFKsdPa.o(.text+0x9a): 对`mysql_close'的未定义引用。
```

通过在链接命令后增加"-Ldir_path -lmysqlclient"选项，应能解决该问题，其中，dir_path代表客户端库所在目录的路径名。要想确定正确的目录，可尝试下述命令：

```
shell> mysql_config --libs
```

来自**mysql_config**的输出可能会指明应在链接命令上指定的其他库。

对于非压缩或压缩函数，如果遇到未定义引用错误，可在链接命令后添加-lz，并再次尝试。

对于应在系统上存在的函数（如connect），如果遇到未定义引用错误，请检查相关函数的手册页，以便确定应在链接命令上增加哪些库。

对于系统上不存在的函数，可能会遇到未定义引用错误，如下所示：

```
mf_format.o(.text+0x201): 对`__lxstat'的未定义引用。
```

它通常意味着你的MySQL客户端库是在与你的系统不100％兼容的系统上编译的。在该情况下，应下载最新的MySQL源码分发版，并自己编译MySQL。请参见2.8节，“使用源码分发版安装MySQL”。

当你试图执行MySQL程序时，可能会遇到运行时未定义引用错误。如果这类错误指明了以mysql_开始的符号，或指明未发现mysqlclient库，这意味着你的系统无法找到共享的libmysqlclient.so库。对其的更正方式是，通知系统在库所在位置搜索共享库。请使用与系统相适应的下述方法：

- 将libmysqlclient.so所在目录的路径添加到LD_LIBRARY_PATH环境变量中。
- 将libmysqlclient.so所在目录的路径添加到LD_LIBRARY环境变量中。
- 将libmysqlclient.so拷贝到可被系统搜索的目录下，如/lib，然后通过执行ldconfig更新共享库信息。

解决该问题的另一种方法是，以静态方式将你的程序与“-static”选项链接在一起，或在链接代码之前删除动态MySQL库。使用第2种方法之前，应确保没有使用动态库的其它程序。

A.3.2. 如何以普通用户身份运行MySQL

在Windows平台上，能够使用普通用户账户以Windows服务方式运行服务器。

在Unix平台上，不是任何用户都能启动并运行MySQL服务器**mysqld**的。但是，处于安全方面的原因，应避免以Unix根用户身份运行服务器。要想更改**mysqld**，使之能以正常的无特权Unix用户*user_name*身份运行，必须采取下述步骤：

如果服务器正在运行，停止它（使用**mysqladmin shutdown**）。

更改数据库目录和文件，允许*user_name*读写其中的文件（可能需要以Unix根用户身份完成这类设置）：

```
shell> chown -R user_name /path/to/mysql/datadir
```

如果未这样做，当以*user_name*身份运行时，服务器无法访问数据库或表。

如果MySQL数据目录下的目录或文件采用的是符号链接，还需跟踪这些链接，并更改它们指向的目录和文件。chown –R可帮助你跟踪符号链接。

以*user_name*身份启动服务器。如果你正在使用MySQL 3.22或更高版本，另一种可选方式是，以Unix根用户身份启动**mysqld**，并使用--user=*user_name*选项。**mysqld**启动，然后在接受任何连接前，切换至Unix用户*user_name*并以该用户身份运行。

要想在系统启动时自动以给定的用户身份启动服务器，可在服务器数据目录下的/etc/my.cnf选项文件或my.cnf选项文件的[mysqld]组中，通过增加用户选项来指定用户名。。例如：

```
[mysqld]
user=user_name
```

如果你的Unix机器本身并不安全，应在授权表中为MySQL根账户指定密码。如不然，任何在该机器上具有登录

账户的用户都能使用"--user=root"选项运行**mysql**客户端，并执行任何操作。在任何情况下均应为MySQL账户指定密码，这是个好主意，尤其是在服务器主机上存在其他登录账户时，更是如此。请参见2.9节，"安装后的设置和测试"。

A.3.3. 与文件许可有关的问题

如果遇到与文件许可有关的问题，可能数启动mysqld时UMASK环境变量设置得不正确。例如，当你创建表时，MySQL可能会发出下述错误消息：

```
ERROR: 无法找到文件: 'path/with/filename.frm' (Errcode: 13)
```

UMASK的默认值是0660。通过下述方式启动**mysqld_safe**，可改变该情况：

```
shell> UMASK=384  # = 600 in octal
shell> export UMASK
shell> mysqld_safe &
```

在默认情况下，MySQL用0700的许可创建数据库和RAID目录。你可以通过设置UMASK_DIR变量更改该行为。如果你设置了它的值，将使用组合的UMASK和UMASK_DIR值创建新目录。例如，如果你打算为所有新的目录授予组访问权限，可：

```
shell> UMASK_DIR=504  # = 770 in octal
shell> export UMASK_DIR
shell> mysqld_safe &
```

在MySQL 3.23.25和更高版本中，如果是以0开始的，MySQL将认为UMASK和UMASK_DIR的值均采用八进制形式。

请参见附录F：环境变量。

A.4. 与管理有关的事宜

A.4.1. 如何复位根用户密码
A.4.2. 如果MySQL依然崩溃，应作些什么
A.4.3. MySQL处理磁盘满的方式
A.4.4. MySQL将临时文件储存在哪里
A.4.5. 如何保护或更改MySQL套接字文件/tmp/mysql.sock
A.4.6. 时区问题

A.4.1. 如何复位根用户密码

如果你从未为MySQL设置根用户密码，服务器在以根用户身份进行连接时不需要密码。但是，建议你为每个账户设置密码。请参见5.6.1节，"通用安全指南"。

如果你以前设置了根用户密码，但却忘记了该密码，可设置新的密码。下述步骤是针对Windows平台的。在本节后面的内容中，介绍了针对Unix平台的步骤。

在Windows平台下，该步骤是：

以系统管理员身份登录到系统。

如果MySQL服务器正在运行，停止它。对于作为Windows服务运行的服务器，进入服务管理器：

开始菜单->控制面板->管理工具->服务

然后在列表中找出MySQL服务器，并停止它。

如果服务器不是作为服务而运行的，可能需要使用任务管理器来强制停止它。

创建1个文本文件，并将下述命令置于单一行中：

```
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('MyNewPassword');
```

用任意名称保存该文件。在本例中，该文件为C:\mysql-init.txt。

打开控制台窗口，进入DOS命令提示：

开始菜单->运行-> cmd

假定你已将MySQL安装到C:\mysql。如果你将MySQL安装到了另一位置，请对下述命令进行相应的调整。

在DOS命令提示符下，执行命令：

```
C:\> C:\mysql\bin\mysqld-nt --init-file=C:\mysql-init.txt
```

在服务器启动时，执行由"--init-file"选项命名的文件的内容，更改根用户密码。当服务器成功启动后，应删除C:\mysql-init.txt。

如果你使用MySQL安装向导安装了MySQL，或许需要指定"--defaults-file"选项：

```
C:\> C:\Program Files\MySQL\MySQL Server 5.1\bin\mysqld-nt.exe
         --defaults-file="C:\Program Files\MySQL\MySQL Server 5.1\my.ini"
         --init-file=C:\mysql-init.txt
```

使用服务管理器，可找到恰当的"--defaults-file"设置：

开始菜单->控制面板->管理工具->服务

在列表中找出MySQL服务，右击，并选择"属性"选项。在可执行字段的Path（路径）中包含"--defaults-file"设置。

停止MySQL服务器，然后在正常模式下重启它。如果以服务方式运行服务器，应从Windows服务窗口启动它。如果以手动方式启动了服务器，能够像正常情形下一样使用命令。

应能使用新密码进行连接。

在Unix环境下，重置根用户密码的步骤如下：

以Unix根用户身份、或以运行mysqld服务器的相同身份登录到系统。

找到包含服务器进程ID的.pid文件。该文件的准确位置和名称取决于你的分发版、主机名和配置。常见位置是/var/lib/mysql/、/var/run/mysqld/和/usr/local/mysql/data/。一般情况下，文件名的扩展名为.pid，并以mysqld或系统的主机名开始。

在下述命令中使用.pid文件的路径名，向**mysqld**进程发出正常的kill（而不是kill -9），可停止MySQL服务器：

```
shell> kill `cat /mysql-data-directory/host_name.pid`
```

注意，cat命令使用符号"`"而不是""：这会使cat的输出代入到kill命令中。

创建文本文件，并将下述命令放在文件内的1行上：

```
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('MyNewPassword');
```

用任意名称保存文件。对于本例，文件为~/mysql-init。

用特殊的"--init-file=~/mysql-init"选项重启MySQL服务器：

```
shell> mysqld_safe --init-file=~/mysql-init &
```

文件init-file的内容在服务器启动时执行，更改根用户密码。服务器成功启动后，应删除~/mysql-init。

应能使用新密码进行连接。

作为可选方式，在任何平台上，可使用**mysql**客户端设置新密码（但该方法不够安全）：

停止**mysqld**，并用"--skip-grant-tables --user=root"选项重启它（Windows用户可省略--user=root部分）。

使用下述命令连接到**mysqld**服务器：

```
shell> mysql -u root
```

在**mysql**客户端发出下述语句：

```
mysql> UPDATE mysql.user SET Password=PASSWORD('newpwd')
    ->                   WHERE User='root';
mysql> FLUSH PRIVILEGES;
```

用打算使用的实际根用户密码替换"*newpwd*"。

应能使用新密码进行连接。

A.4.2. 如果MySQL依然崩溃，应作些什么

正式发布之前，每个MySQL版本均在很多平台上进行了测试。这不表示MySQL中不存在缺陷，但是，如果存在缺陷，它们应很少，而且很难发现。如果你遇到问题，如果你尝试找出导致系统崩溃的准确原因，这始终很有帮助，这是因为，如果这样的话，快速解决问题的机会很大。

首先，应尝试找出问题是否与**mysqld**服务器有关，或是否与客户端有关。通过执行**mysqladmin version**，可检查**mysqld**服务器运行了多长时间。如果**mysqld**宕机并重启，应查看服务器的错误日志以找出原因。请参见5.11.1节，"错误日志"。

在某些系统上，在错误日志中，可发现**mysqld**宕机的堆栈跟踪，可使用resolve_stack_dump程序解决它。请参见E.1.4节，"使用堆栈跟踪"。注意，错误日志中的变量值并非始终是100％正确的。

很多服务器崩溃是因损坏的数据文件或索引文件而导致的，每次执行完SQL语句之后并在向客户端通告结果之前，MySQL将使用write()系统调用更新磁盘上的文件（如果你使用了"--delay-key-write"选项，情况并非如此，此时将写入数据文件而不是索引文件）。这意味着，即使mysqld崩溃，数据文件的内容也是安全的，这是因为操作系统能保证将未刷新的数据写入磁盘。使用"--flush"选项启动**mysqld**，这样，每次执行完SQL语句后，可强制MySQL将所有内容写入磁盘。

前述介绍表明，在正常情况下不会出现损坏的表，除非出现了下述情况之一：

在更新过程中，MySQL服务器或服务器主机被停止。

你发现了**mysqld**中存在的1个缺陷，该缺陷导致**mysqld**在更新中途中止。

在**mysqld**操作的同时，某些外部程序正在操控数据文件或索引文件，未恰当锁定表。

你正使用系统上的相同数据目录运行很多**mysqld**服务器，该系统不支持良好的文件系统锁定（通常是由lockd锁定管理器负责的），或使用"--skip-external-locking"选项运行了多个服务器。

崩溃的数据文件或索引文件，其中包含导致mysqld混乱的损坏很严重的数据。

在数据存储节点发现缺陷。这种可能性不大，但至少是可能的。在该情况下，可在修复的表副本上，通过使用ALTER TABLE，尝试将表类型更改为另一种存储引擎。

由于很难得知为什么某事会出现崩溃，首先请检查用于其他方面的事项是否崩溃。请尝试采取下述措施：

用**mysqladmin shutdown**停止**mysqld**服务器，从数据目录运行**myisamchk --silent --force */*.MYI**，检查所有的MyISAM表，并重启**mysqld**。这样，就能确保从干净的状态运行服务器。请参见第5章：数据库管理。

使用"--log"选项启动**mysqld**，并根据写入日志的信息确定是否某些特殊的查询杀死了服务器。约95%的缺陷与特定的查询有关。正常情况下，这是服务器重启前日志文件中最够数个查询中的1个。请参见5.11.2节，"通用查询日志"。如果能够用特殊查询重复杀死MySQL，即使在发出查询前检查了所有表的情况下也同样，那么你就应能确定缺陷，并应提交关于该缺陷的缺陷报告。请参见1.7.1.3节，"如何通报缺陷和问题"。

尝试提供一个测试范例，我们应能利用该范例重复问题。请参见E.1.6节，"如果出现表崩溃，请生成测试案例"。

请在mysql-test目录下并根据MySQL基准进行测试。请参见27.1.2节，"MySQL测试套件"。它们能相当良好地测试MySQL。你也可以为基准测试增加代码，以模拟你的应用程序。基准测试可在源码分发版的sql-bench目录下找到，对于二进制分发版，可在MySQL安装目录下的sql-bench目录下找到。

尝试使用fork_big.pl脚本（它位于源码分发版的测试目录下）。

如果你将MySQL配置为调试模式，如果某事出错，可更为容易地搜集关于可能错误的信息。如果将MySQL配置为调试模式，可生成1个安全的内存分配程序，可使用它发现某些错误。此外，它还提供了很多输出，这类输出与出现的问题相关。在**configure**上使用"--with-debug"或"--with-debug=full"选项重新配置MySQL，然后再编译它。请参见E.1节，"调试MySQL服务器"。

确保为你的操作系统应用了最新的补丁。

对**mysqld**使用"--skip-external-locking"选项。在某些系统上，lockd锁定管理器不能正确工作，"--skip-external-locking"选项通知**mysqld**不使用外部锁定。（这意味着，你不能在相同的数据目录上运行2个**mysqld**服务器，如果使用**myisamchk**，必须谨慎。然而，尝试将该选项用作测试也是有益的）。

当mysqld看上去正在运行但并未响应时，是否运行了**mysqladmin -u root processlist**？某些时候，即使你认为**mysqld**处于闲置状态时，实际情况并非如此。问题可能是因为所有连接均已使用，或存在某些内部锁定问题。即使在该情况下，**mysqladmin -u root processlist**通常能够进行连接，并能提供关于当前连接数以及其状态的有用信息。

在运行其他查询的同时，在单独的窗口中运行命令**mysqladmin -i 5 status**或**mysqladmin -i 5 -r status**，以生成统计信息。

尝试采用下述方法：

从gdb（或另一个调试器）启动**mysqld**。请参见E.1.3节，"在gdb环境下调试m**ysqld**"。

运行测试脚本。

在3个较低层面上输出backtrace（向后跟踪）和局部变量。在**gdb**中，当mysqld在gdb内崩溃时，可使用下述命令完成该任务：

```
backtrace

info local

up

info local

up

info local
```

使用**gdb**，你还能检查与info线程共存的线程，并切换至特定的线程N，其中，N是线程ID。

尝试用Perl脚本模拟你的应用程序，强制MySQL崩溃或行为异常。

发送正常的缺陷报告。请参见1.7.1.3节，“如何通报缺陷和问题”。应比通常的报告更详细。由于MySQL是为很多人提供服务的，它可能因仅存在于你的计算机上的某事崩溃（例如，与你的特定系统库有关的错误）。

如果你遇到与包含动态长度行的表有关的问题，而且你仅使用VARCHAR列（而不是BLOB或TEXT列），可尝试用ALTER TABLE将所有VARCHAR列更改为CHAR列。这样，就会强制MySQL使用固定大小的行。固定大小的行占用的空间略多，但对损坏的容忍度更高。

目前的动态行代码在MySQL AB已使用多年，很少遇到问题，但从本质上看，动态长度行更倾向于出现错误，因此，不妨尝试采用该策略以查看它是否有帮助，这不失为一个好主意。

诊断问题时不要将你的服务器硬件排除在外。有缺陷的硬件能够导致数据损坏。对硬件进行故障诊断与排除操作时，尤其应注意RAM和硬盘驱动器。

A.4.3. MySQL处理磁盘满的方式

在本节中，介绍了MySQL响应磁盘满错误的方式（如“设备上无剩余空间”），以及响应超配额错误的方式（如“写入失败”或“达到了用户屏蔽限制”）。

本节介绍的内容与写入MyISAM表有关。它也适用于写入二进制日志文件和二进制索引文件，但对“row”和“record”的应用应被视为“event”。

出现磁盘满状况时，MySQL将：

每分钟检查一次，查看是否有足够空间写入当前行。如果有足够空间，将继续，就像什么也未发生一样。

每10分钟将1个条目写入日志文件，提醒磁盘满状况。

为了减轻问题，可采取下述措施：

要想继续，仅需有足够的磁盘空间以插入所有记录。

要想放弃线程，必须使用**mysqladmin kill**。下次检查磁盘时将放弃线程（1分钟）。

其他线程可能会正在等待导致磁盘满状况的表。如果有数个“已锁定”的线程，杀死正在磁盘满状况下等待的某一线程，以便允许其他线程继续。

对前述行为的例外是，当你使用REPAIR TABLE或OPTIMIZE TABLE时，或当索引是在LOAD DATA

INFILE或ALTER TABLE语句后、在批操作中创建的。所有这些语句能创建大的临时文件，如果保留这些文件，会导致系统其他部分出现大问题。如果在MySQL执行这类操作的同时磁盘已满，它将删除大的临时文件，并将表标注为崩溃。但对于ALTER TABLE例外，旧表保持不变。

A.4.4. MySQL将临时文件储存在哪里

MySQL使用环境变量TMPDIR的值作为保存临时文件的目录的路径名。如果未设置TMPDIR，MySQL将使用系统的默认值，通常为/tmp、/var/tmp或/usr/tmp。如果包含临时文件目录的文件系统过小，可对mysqld使用"—tmpdir"选项，在具有足够空间的文件系统内指定1个目录。

在MySQL 5.1中，"—tmpdir"选项可被设置为数个路径的列表，以循环方式使用。在Unix平台上，路径用冒号字符":"隔开，在Windows、NetWare和OS/2平台上，路径用分号字符";"隔开。注意，为了有效分布负载，这些路径应位于不同的物理磁盘上，而不是位于相同磁盘的不同分区中。

如果MySQL服务器正作为复制从服务器使用，不应将"--tmpdir"设置为指向基于内存的文件系统的目录，或当服务器主机重启时将清空的目录。对于复制从服务器，需要在机器重启时仍保留一些临时文件，以便能够复制临时表或执行LOAD DATA INFILE操作。如果在服务器重启时丢失了临时文件目录下的文件，复制将失败。

MySQL会以隐含方式创建所有的临时文件。这样，就能确保中止**mysqld**时会删除所有临时文件。使用隐含文件的缺点在于，在临时文件目录所在的位置中，看不到占用了文件系统的大临时文件。

进行排序时（ORDER BY或GROUP BY），MySQL通常会使用1个或多个临时文件。所需的最大磁盘空间由下述表达式决定：

```
(length of what is sorted + sizeof(row pointer))

* number of matched rows

* 2
```

"row pointer"（行指针）的大小通常是4字节，但在以后，对于大的表，该值可能会增加。

对于某些SELECT查询，MySQL还会创建临时SQL表。它们不是隐含表，并具有SQL_*形式的名称。

ALTER TABLE会在与原始表目录相同的目录下创建临时表。

A.4.5. 如何保护或更改MySQL套接字文件/tmp/mysql.sock

对于服务器用来与本地客户端进行通信的Unix套接字文件，其默认位置是/tmp/mysql.sock。这有可能导致问题，原因在于，在某些版本的Unix上，任何人都能删除/tmp目录下的文件。

在大多数Unix版本中，可对/tmp目录进行保护，使得文件只能被其所有这或超级用户（根用户）删除。为此，以根用户身份登录，并使用下述命令在/tmp目录上设置粘着位：

```
shell> chmod +t /tmp
```

通过执行ls -ld /tmp，可检查是否设置了粘着位。如果最后一个许可字符是"t"，表明设置了粘着位。

另一种方法是改变服务器创建Unix套接字文件的位置。如果进行了这类操作，还应让客户端程序知道文件的位置。能够以多种不同方式指定文件位置：

在全局或局部选项文件中指定路径。例如，将下述行置于文件/etc/my.cnf中：

```
[mysqld]

socket=/path/to/socket
```

```
[client]
socket=/path/to/socket
```

请参见4.3.2节，“使用选项文件”。

在运行客户端程序时，在命令行上为**mysqld_safe**指定“--socket”选项。

将MYSQL_UNIX_PORT环境变量设置为Unix套接字文件的路径。

重新从源码编译MySQL，以使用不同的默认Unix套接字文件位置。运行**configure**时，用“--with-unix-socket-path”选项定义文件路径。请参见2.8.2节，“典型配置选项”。

用下述命令连接服务器，能够测试新的套接字位置是否工作：

```
shell> mysqladmin --socket=/path/to/socket version
```

A.4.6. 时区问题

如果遇到与SELECT NOW()有关的问题，它返回GMT值而不是当地时间，就应通知服务器你的当前失去。如果UNIX_TIMESTAMP()返回错误值，上述方式同样适用。应为服务器所运行的环境进行这类设置，例如，在**mysqld_safe**或**mysql.server**中。请参见附录F：环境变量。

也可以对**mysqld_safe**使用“--timezone=*timezone_name*”选项，为服务器设置失去。也可以在启动mysqld之前，通过设置TZ环境变量完成该设置。

“--timezone”或TZ的允许值与系统有关。关于可接受的值，请参见操作系统文档。

A.5. 与查询有关的事宜

A.5.1. 搜索中的大小写敏感性
A.5.2. 使用DATE列方面的问题
A.5.3. 与NULL值有关的问题
A.5.4. 与列别名有关的问题
A.5.5. 非事务表回滚失败
A.5.6. 从相关表删除行
A.5.7. 解决与不匹配行有关的问题
A.5.8. 与浮点比较有关的问题

A.5.1. 搜索中的大小写敏感性

在默认情况下，MySQL搜索不区分大小写（但某些字符集始终区分大小写，如czech）。这意味着，如果你使用*col_name* LIKE 'a%'进行搜索，你将获得以A或a开始的所有列。如果打算使搜索区分大小写，请确保操作数之一具有区分大小写的或二进制校对。例如，如果你正在比较均适用latin1字符集的列和字符串，可使用COLLATE操作符，使1个操作数具有latin1_general_cs或latin1_bin校对特性。例如：

```
col_name COLLATE latin1_general_cs LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_general_cs
col_name COLLATE latin1_bin LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_bin
```

如果希望总是以区分大小写的方式处理列，可使用区分大小写的或二进制校对声明它。请参见13.1.5节，“CREATE TABLE语法”。

简单的比较操作（>=, >, =, <, <=, 排序和分组）基于每个字符的“排序值”。具有相同排序值的字符（如‘E’, ‘e’,和‘Ã©’）将被当作相同的写字符。

A.5.2. 使用DATE列方面的问题

DATE值的格式是'YYYY-MM-DD'。按照标准的SQL，不允许其他格式。在UPDATE表达式以及SELECT语句的WHERE子句中应使用该格式。例如：

```
mysql> SELECT * FROM tbl_name WHERE date >= '2003-05-05';
```

为了方便，如果日期是在数值环境下使用的，MySQL会自动将日期转换为数值（反之亦然）。它还具有相当的智能，在更新时或在与TIMESTAMP、DATE或DATETIME列比较日期的WHERE子句中，允许“宽松的”字符串形式（“宽松形式”表示，任何标点字符均能用作各部分之间的分隔符。例如，'2004-08-15'和'2004#08#15'是等同的）。MySQL还能转换不含任何分隔符的字符串（如'20040815'），前体是它必须是有意义的日期。

使用<、<=、=、>=、>、或BETWEEN操作符将DATE、TIME、DATETIME或TIMESTAMP与常量字符串进行比较时，MySQL通常会将字符串转换为内部长整数，以便进行快速比较（以及略为“宽松”的字符串检查）。但是，该转换具有下述例外：

比较两列时

将DATE、TIME、DATETIME或TIMESTAMP列与表达式进行比较时

使用其他比较方法时，如IN或STRCMP()。

对于这些例外情形，会将对象转换为字符串并执行字符串比较，采用该方式进行比较。

为了保持安全，假定按字符串比较字符串，如果你打算比较临时值和字符串，将使用恰当的字符串函数。

对于特殊日期'0000-00-00'，能够以'0000-00-00'形式保存和检索。在MyODBC中使用'0000-00-00'日期时，对于MyODBC 2.50.12或更高版本，该日期将被自动转换为NULL，这是因为ODBC不能处理这类日期。

由于MySQL能够执行前面所介绍的转换，下述语句均能正常工作：

```
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';
```

但是，下述语句不能正常工作：

```
mysql> SELECT idate FROM tbl_name WHERE STRCMP(idate,'20030505')=0;
```

STRCMP()是一种字符串函数，它能将idate转换为'YYYY-MM-DD'格式的字符串，并执行字符串比较。它不能将'20030505'转换为日期'2003-05-05'并进行日期比较。

如果你正在使用ALLOW_INVALID_DATES SQL模式，MySQL允许以仅执行给定的有限检查方式保存日期：MySQL仅保证天位于1～31的范围内，月位于1～12的范围内。

这样就使得MySQL很适合于Web应用程序，其中，你能获得三个不同字段中的年、月、日值，也能准确保存用户插入的值（无日期验证）。

如果未使用NO_ZERO_IN_DATE SQL模式，“天”和“月”部分可能为0。如果你打算将生日保存在DATE列而且仅知道部分日期，它十分方便。

如果未使用NO_ZERO_DATE SQL模式，MySQL也允许你将'0000-00-00'保存为“伪日期”。在某些情况下，它比使用NULL值更方便。

如果无法将日期转换为任何合理值，“0”将保存在DATE列中，并被检索为'0000-00-00'。这是兼顾速度和便利性的事宜。我们认为，数据库服务器的职责是检索与你保存的日期相同的日期（即使在任何情况下，数据在逻辑上不正确也同样）。我们认为，对日期的检查应由应用程序而不是服务器负责。

如果你希望MySQL检查所有日期并仅接受合法日期（除非由IGNORE覆盖），应将sql_mode设置为"NO_ZERO_IN_DATE,NO_ZERO_DATE"。

A.5.3. 与NULL值有关的问题

对于SQL的新手，NULL值的概念常常会造成混淆，他们常认为NULL是与空字符串"相同的事。情况并非如此。例如，下述语句是完全不同的：

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);

mysql> INSERT INTO my_table (phone) VALUES ('');
```

这两条语句均会将值插入phone（电话）列，但第1条语句插入的是NULL值，第2条语句插入的是空字符串。第1种情况的含义可被解释为“电话号码未知”，而第2种情况的含义可被解释为“该人员没有电话，因此没有电话号码”。

为了进行NULL处理，可使用IS NULL和IS NOT NULL操作符以及IFNULL()函数。

在SQL中，NULL值与任何其它值的比较（即使是NULL）永远不会为“真”。包含NULL的表达式总是会导出NULL值，除非在关于操作符的文档中以及表达式的函数中作了其他规定。下述示例中的所有列均返回NULL：

```
mysql> SELECT NULL, 1+NULL, CONCAT('Invisible',NULL);
```

如果打算搜索列值为NULL的列，不能使用expr = NULL测试。下述语句不返回任何行，这是因为，对于任何表达式，expr = NULL永远不为“真”：

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

要想查找NULL值，必须使用IS NULL测试。在下面的语句中，介绍了查找NULL电话号码和空电话号码的方式：

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;

mysql> SELECT * FROM my_table WHERE phone = '';
```

更多信息和示例，请参见3.3.4.6节，“使用NULL值”。

如果你正在使用MyISAM、InnoDB、BDB、或MEMORY存储引擎，能够在可能具有NULL值的列上增加1条索引。如不然，必须声明索引列为NOT NULL，而且不能将NULL插入到列中。

用LOAD DATA INFILE读取数据时，对于空的或丢失的列，将用"更新它们。如果希望在列中具有NULL值，应在数据文件中使用\N。在某些情况下，也可以使用文字性单词“NULL”。请参见13.2.5 “LOAD DATA INFILE语法”。

使用DISTINCT、GROUP BY或ORDER BY时，所有NULL值将被视为等同的。

使用ORDER BY时，首先将显示NULL值，如果指定了DESC按降序排列，NULL值将最后显示。

对于聚合（累计）函数，如COUNT()、MIN()和SUM()，将忽略NULL值。对此的例外是COUNT(*)，它将计数行而不是单独的列值。例如，下述语句产生两个计数。首先计数表中的行数，其次计数age列中的非NULL值数目：

```
mysql> SELECT COUNT(*), COUNT(age) FROM person;
```

对于某些列类型，MySQL将对NULL值进行特殊处理。如果将NULL插入TIMESTAMP列，将插入当前日期和时间。如果将NULL插入具有AUTO_INCREMENT属性的整数列，将插入序列中的下一个编号。

A.5.4. 与列别名有关的问题

可以使用别名来引用GROUP BY、ORDER BY或HAVING子句中的列。别名也能用于为列提供更好的名称：

```
SELECT SQRT(a*b) AS root FROM tbl_name GROUP BY root HAVING root > 0;
SELECT id, COUNT(*) AS cnt FROM tbl_name GROUP BY id HAVING cnt > 0;
SELECT id AS 'Customer identity' FROM tbl_name;
```

标准SQL不允许在WHERE子句中已用列别名。这是因为，执行WHERE代码时，可能尚未确定列值。例如，下述查询是非法的：

```
SELECT id, COUNT(*) AS cnt FROM tbl_name WHERE cnt > 0 GROUP BY id;
```

执行WHERE语句以确定哪些行应被包含在GROUP BY部分中，而HAVING用于确定应使用结果集中的哪些行。

A.5.5. 非事务表回滚失败

执行ROLLBACK（回滚）时，如果收到下述消息，表示事务中使用的1个或多个表不支持事务：

警告：某些更改的非事务性表不能被回滚。

这些非事务性表不受ROLLBACK语句的影响。

如果在事务中意外地混合了事务性表和非事务性表，导致该消息的最可能原因是，你认为本应是事务性的表实际上不是。如你试图使用mysqld服务器不支持的事务性存储引擎（或用启动选项禁止了它）创建表，就可能出现该情况。如果**mysqld**不支持存储引擎，它将以MyISAM表创建表，这是非事务性表。

可使用下述语句之一检查表的标类型：

```
SHOW TABLE STATUS LIKE 'tbl_name';
SHOW CREATE TABLE tbl_name;
```

请参见13.5.4.18节，“SHOW TABLE STATUS语法以及13.5.4.5节，“SHOW CREATE TABLE语法”。

使用下述语句，可检查mysqld服务器支持的存储引擎：

```
SHOW ENGINES;
```

也可以使用下述语句，检查与你感兴趣的存储引擎有关的变量值：

```
SHOW VARIABLES LIKE 'have_%';
```

例如，要想确定InnoDB存储引擎是否可用，可检查have_innodb变量的值。

请参见13.5.4.8节，“SHOW ENGINES语法”和13.5.4.21节，“SHOW VARIABLES语法”。

A.5.6. 从相关表删除行

如果针对related_table的DELETE语句的总长度超过1MB（系统变量max_allowed_packet的默认值），应将其分为较小的部分，并执行多个DELETE语句。如果related_column是索引列，为每条语句指定100～1000个related_column值，或许能获得更快的DELETE速度。如果related_column不是索引列，速度与IN子句中的参量数目无关。

A.5.7. 解决与不匹配行有关的问题

如果有使用了很多表的复杂查询，但未返回任何行，应采用下述步骤找出什么出错：

用EXPLAIN测试查询，以检查是否发现某事显然出错。请参见7.2.1节，“EXPLAIN语法（获取关于SELECT的信息）”。

仅选择在WHERE子句中使用的列。

从查询中1次删除1个表，直至返回了某些行为止。如果表很大，较好的主意是在查询中使用LIMIT 10。

对于具有与上次从查询中删除的表匹配的行的列，发出SELECT查询。

如果将FLOAT或DOUBLE列与具有数值类型的数值进行比较，不能使用等式(=)比较。在大多数计算机语言中，该问题很常见，这是因为，并非所有的浮点值均能以准确的精度保存。在某些情况下，将FLOAT更改为DOUBLE可更正该问题。请参见A.5.8节，“与浮点比较有关的问题”。

如果仍不能找出问题之所在，请创建能与显示问题的“mysql test < query.sql”一起运行的最小测试。通过使用**mysqldump --quick db_name** *tbl_name_1* **...** *tbl_name_n* **> query.sql**转储表，可创建测试文件。在编辑器中打开文件，删除某些插入的行（如果有超出演示问题所需的行），并在文件末尾添加SELECT语句。

通过执行下述命令，验证测试文件能演示问题：

```
shell> mysqladmin create test2

shell> mysql test2 < query.sql
```

使用**mysqlbug**将测试文件张贴到哟娜通用MySQL邮件列表。请参见1.7.1.1节，“The MySQL邮件列表”。

A.5.8. 与浮点比较有关的问题

注意，下述部分主要与DOUBLE和FLOAT列相关，原因在于浮点数的不准确本质。MySQL使用64位十进制数值的精度执行DECIMAL操作，当处理DECIMAL列时，应能解决大多数常见的不准确问题。

浮点数有时会导致混淆，这是因为它们无法以准确值保存在计算机体系结构中。你在屏幕上所看到的值通常不是数值的准确值。对于FLOAT和DOUBLE列类型，情况就是如此。DECIMAL列能保存具有准确精度的值，这是因为它们是由字符串表示的。

在下面的示例中，介绍了使用DOUBLE时的问题：

```
mysql> CREATE TABLE t1 (i INT, d1 DOUBLE, d2 DOUBLE);
mysql> 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 |  21.4 | 21.4 |
|    2 |  76.8 | 76.8 |
|    3 |   7.4 |  7.4 |
|    4 |  15.4 | 15.4 |
|    5 |   7.2 |  7.2 |
|    6 | -51.4 |    0 |
+------+-------+------+
```

结果是正确的。尽管前5个记录看上去不应能进行比较测试（a和b的值看上去没有什么不同），但它们能进行比较，这是因为显示的数值间的差异在十分位左右，具体情况取决于计算机的体系结构。

如果列d1和d2定义为DECIMAL而不是DOUBLE，SELECT查询的结果仅包含1行，即上面显示的最后1行。

A.6. 与优化器有关的事宜

MySQL采用了基于开销的优化器，以确定处理查询的最解方式。在很多情况下，MySQL能够计算最佳的可能查询计划，但在某些情况下，MySQL没有关于数据的足够信息，不得不就数据进行"有教养"的估测。

当MySQL未能做"正确的"事时，可使用下述工具来帮助MySQL：

使用EXPLAIN语句获取关于MySQL如何处理查询的信息。要想使用它，可在SELECT语句前添加关键字EXPLAIN：

```
mysql> EXPLAIN SELECT * FROM t1, t2 WHERE t1.i = t2.i;
```

关于EXPLAIN的详细讨论，请参见7.2.1节，"EXPLAIN语法（获取关于SELECT的信息）"。

使用ANALYZE TABLE *tbl_name*，为已扫描的表更新键分配。请参见13.5.2.1节，“ANALYZE TABLE语法”。

为已扫描的表使用FORCE INDEX，通知MySQL：与使用给定的索引相比，表扫描开销昂贵。请参见13.2.7节，“SELECT语法”。

```
SELECT * FROM t1, t2 FORCE INDEX (index_for_column)

WHERE t1.col_name=t2.col_name;
```

USE INDEX和IGNORE INDEX也有一定的帮助。

关于全局和表级别的STRAIGHT_JOIN。请参见13.2.7节，“SELECT语法”。

你可以调节全局或线程类系统变量。例如，用“--max-seeks-for-key=1000”选项启动**mysqld**，或使用“SET max_seeks_for_key=1000”来通知优化器：假定任何表扫描均不会导致1000个以上的键搜索。请参见5.3.3节，“服务器系统变量”。

A.7. 与表定义有关的事宜

A.7.1. 与ALTER TABLE有关的问题
A.7.2. 如何更改表中的列顺序
A.7.3. TEMPORARY TABLE问题

A.7.1. 与ALTER TABLE有关的问题

ALTER TABLE将表更改为当前字符集。如果在执行ALTER TABLE操作期间遇到重复键错误，原因在于新的字符集将2个键映射到了相同值，或是表已损坏。在后一种情况下，应在表上运行REPAIR TABLE。

如果ALTER TABLE失败并给出下述错误，问题可能是因为在ALTER TABLE操作的早期阶段出现MySQL崩溃，没有名为A-*xxx*或B-*xxx*的旧表：

```
Error on rename of './database/name.frm'

to './database/B-xxx.frm' (Errcode: 17)
```

在该情况下，进入MySQL数据目录，并删除其名称为以A-或B-开始的所有文件（或许你希望将它们移动到其他地方而不是删除它们）。

ALTER TABLE的工作方式如下：

- 用请求的结构变化创建名为A-*xxx*的新表。
- 将所有行从原始表拷贝到A-*xxx*。
- 将原始表重命名为B-*xxx*。
- 将A-*xxx*重命名为原始表的名称。
- 删除B-*xxx*。

如果在重命名操作中出错，MySQL将尝试撤销更改。如果错误很严重（尽管这不应出现），MySQL会将旧表保留为B-*xxx*。简单地在系统级别上重命名表文件，应能使数据复原。

如果在事务性表上使用ALTER TABLE，或正在使用Windows或OS/2操作系统，如果已在表上执行了LOCK TABLE操作，ALTER TABLE将对表执行解锁操作。这是因为InnoDB和这类操作系统不能撤销正在使用的表。

A.7.2. 如何更改表中的列顺序

首先，请考虑是否的确需要更改表中的列顺序。SQL的核心要点是从数据存储格式获取应用。总应指定检索数据的顺序。在下面的第1条语句中，以*col_name1*、*col_name2*、*col_name3*顺序返回列；在第2条语句中，以*col_name1*、*col_name3*、*col_name2*顺序返回列：

```
mysql> SELECT col_name1, col_name2, col_name3 FROM tbl_name;

mysql> SELECT col_name1, col_name3, col_name2 FROM tbl_name;
```

如果决定更改表列的顺序，可执行下述操作：

1. 用具有新顺序的列创建新表。
2. 执行该语句：

```
mysql> INSERT INTO new_table

    -> SELECT columns-in-new-order FROM old_table;
```

3. 撤销或重命名old_table。
4. 将新表重命名为原始名称：

```
mysql> ALTER TABLE new_table RENAME old_table;
```

SELECT *十分适合于测试查询。但是，在应用程序中，永远不要依赖SELECT *的使用，不要依赖根据其位置检索列。如果添加、移动或删除了列，所返回的列的顺序和位置不会保持相同。对表结构的简单更改也会导致应用程序失败。

A.7.3. TEMPORARY TABLE问题

下面介绍了对使用TEMPORARY表的限制：

- TEMPORARY表只能是HEAP、ISAM、MyISAM、MERGE、或InnoDB类型。
- 在相同的查询中，不能引用TEMPORARY表1次以上。例如，下例不能正常工作：

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

错误1137：不能再次打开表：'temp_table'
```

- SHOW TABLES语句不会列出TEMPORARY表。
- 不能使用RENAME重命名TEMPORARY表。但能使用ALTER TABLE取而代之：

```
mysql> ALTER TABLE orig_name RENAME new_name;
```

A.8. MySQL中的已知事宜

A.8.1. MySQL中的打开事宜

在本节中，列出了当前MySQL版本中的已知事宜。

关于平台相关事宜的更多信息，请参见2.12节，“具体操作系统相关的注意事项”和附录E：移植到其他系统中的安装和移植说明。

A.8.1. MySQL中的打开事宜

下面列出了已知问题，更正它们具有较高的优先级：

- 如果将NULL值与使用ALL/ANY/SOME的子查询进行比较，而且子查询返回空的结果，比较操作会评估NULL的非标准结果而不是TRUE或FALSE。在MySQL 5.1中将更正该问题。
- 对于IN的线子查询优化不像"="那样有效。
- 即使使用了lower_case_table_names=2（允许MySQL记住数据库名和表名使用的大小写），对于函数DATABASE()或在各种日志内（在不区分大小写的系统上），MySQL也不会记住数据库名使用的大小写情况。
- 在复制操作中，撤销FOREIGN KEY约束不工作，这是因为约束可能在从服务器上有另一个名称。
- REPLACE（以及具有REPLACE选项的LOAD DATA）不会触发ON DELETE CASCADE。
- 如果未使用所有列而且仅使用DISTINCT列表中的列，在GROUP_CONCAT()中，DISTINCT不能与ORDER BY一起工作。
- 如果1位用户拥有长时间运行的事务，而且另1位用户撤销了在事务中更新的某1表，那么在表用于事务本身之前，存在较小的机会，会在二进制日志中包含DROP TABLE命令。我们计划更正该问题，方法是让DROP TABLE命令等待，直至表未在任何事务中使用为止。
- 将大的整数值（介于2^{63}和$2^{64}-1$之间）插入数值或字符串列时，它将作为负值插入，这是因为该数值是在有符号整数环境下评估的。
- 如果服务器运行在不具备二进制日志功能的条件下，FLUSH TABLES WITH READ LOCK不能屏蔽COMMIT，执行完整备份时这可能会导致问题（表间的一致性问题）。
- 在某些情况下，作用在BDB表上的ANALYZE TABLE会导致表不可用，直至重启**mysqld**为止。如果出现该情况，请在MySQL错误文件中查找下述形式的错误：

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

- 在所有事务完成之前，不要在BDB表（正在其上运行多语句事务）上执行ALTER TABLE（可能会忽略事务）。
- 对于正在使用INSERT DELAYED的表，在其上执行ANALYZE TABLE、OPTIMIZE TABLE和REPAIR TABLE时，可能会导致问题。
- 在表上执行LOCK TABLE ...和FLUSH TABLES ...时，不保证没有完成一半的事务。
- BDB表打开的速度相对较慢。如果你在数据库上有很多BDB表，如果未使用"-A"选项或正使用再混编功能，要想在数据库上使用mysql客户端，需要花费较长的时间。当你有大的表高速缓冲时，这点尤其明显。
- 复制功能采用了查询级日志功能：主服务器将已执行的查询写入二进制日志。这是一种速度很快、简洁和有效的记录方法，在大多数情况下工作良好。

如果以特定的方式设计查询，使得数据更改是非决定性（通常不推荐，即使在复制之外也同样），主服务器和从服务器上的数据将变得不同。

例如：

- 将0或NULL值插入AUTO_INCREMENT列中的CREATE ... SELECT或INSERT ... SELECT语句。
- DELETE，如果从具有ON DELETE CASCADE属性的外键的表中删除行。
- REPLACE ... SELECT、INSERT IGNORE ... SELECT，如果在插入的数据中具有重复键。

当且仅当前述查询没有保证决定行顺序的**ORDER BY**子句时。

例如，对于不具有ORDER BY的INSERT ... SELECT，SELECT可能会以不同的顺序返回行（它会导致具有不同等级的行，从而导致AUTO_INCREMENT列中的不同数值），具体情况取决于优化器在主服务器和从服务器上所作的选择。

在主服务器和从服务器上，查询将进行不同的优化，仅当：

- 使用不同的存储引擎在主服务器上而不是从服务器上保存表。（能够在主服务器和从服务器上使用不同的存储引擎。例如，如果从服务器具有较少的可用磁盘空间，可以在主服务器上使用InnoDB，但在 从服务器桑使用MyISAM）。
- 在主服务器和从服务器上，MySQL缓冲区大小是不同的（key_buffer_size等）。
- 在主服务器和从服务器上运行不同的MySQL版本，版本间的优化器代码也不同。

该问题也会影响使用**mysqlbinlog | mysql**的数据库恢复。

避免该问题的最简单方法是，为前述的非决定性查询增加ORDER BY子句，以确保总是以相同的顺序保存或更改行。

在将来的MySQL版本中，需要时，我们将自动增加ORDER BY子句。

下面列出了已知的事宜，这些事宜将在恰当的时候更正：

- 日志文件名基于服务器主机名（如果未使用启动选项指定文件名的话）。如果更改了主机名，你将不得不使用诸如"--log-bin=*old_host_name*-bin"等选下美国。另一种选择是重命名旧文件，以反映主机名变更情况（如果是二进制日志，需要编辑二进制日志索引文件，并更正binlog名称）。请参见5.3.1节，"**mysqld**命令行选项"。
- **Mysqlbinlog**不删除执行LOAD DATA INFILE命令后遗留的临时文件。请参见8.6节，"mysqlbinlog：用于处理二进制日志文件的实用工具"。
- RENAME不能与TEMPORARY表一起工作，也不能与MERGE表中使用的表一起工作。
- 由于表定义文件的保存方式，不能在表名、列名或枚举中使用字符255（CHAR(255)）。按照安排，当我们实施了新的表定义格式文件时，将在5.1版中更正该问题。
- 使用SET CHARACTER SET时，不能在数据库、表和列名中使用转换的字符。
- 不能在LIKE ... ESCAPE中与ESCAPE一起使用'_'或'%'。
- 如果你有1个DECIMAL列，其中，相同的数值以不同的格式保存（例如，+01.00、1.00、01.00），GROUP BY可能会将每个值当作不同的值。
- 使用MIT-pthreads时，不能在另一个目录下创建服务器。这是因为它需要更改MIT-pthreads，我们不太会更正该问题。请参见2.8.5 "MIT-pthreads注意事项"。
- 在GROUP BY、ORDER BY或DISTINCT中，不能可靠地使用BLOB和TEXT值。在这类情况下，与BLOB值进行比较时，仅使用最前的max_sort_length字节。max_sort_length的默认值是1024，可在服务器启动时或运行时更改它。
- 数值计算是使用BIGINT或DOUBLE（正常情况下均为64位长）进行的。你所能获得的精度取决于函数。通用规则是位函数是按BIGINT精度执行的，IF和ELT()是按BIGINT或DOUBLE精度执行的，其余的函数是按DOUBLE精度执行的。对于除位字段外的其他数，如果大于63位（9223372036854775807），应避免使用无符号长long值。
- 在1个表中，最多能有255个ENUM和SET列。
- 在MIN()、MAX()以及其他聚合函数中，MySQL目前会根据其字符串值比较ENUM和SET列，而不是根据字符串在集合中的相对位置。
- **mysqld_safe**会将来自**mysqld**的所有消息再定向到**mysqld**日志。与之相关的1个问题是，如果你执行**mysqladmin refresh**关闭并再次打开日志，stdout和stderr仍会被重定向到旧的日志。如果你以广义方式使用"--log"，应编辑**mysqld_safe**以记录到*host_name*.err而不是*host_name*.log，以便通过删除它并执行**mysqladmin refresh**，方便地收回为旧日志分配的空间。
- 在UPDATE语句中，列从左向右更新。如果引用了已更新的列，你将得到更新值而不是原始值。例如，下述语句会将KEY增加2，而不是1：

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

- 你可以在相同查询中引用多个临时表，但不能引用任何给定的临时表1次以上。例如，下述语句不能正常工作：

```
mysql> SELECT * FROM temp_table, temp_table AS t2;
错误1137：不能再次打开表：'temp_table'
```

- 当你在联合操作中使用"隐含"列时，与未使用隐含列相比，优化器将以不同的方式处理DISTINCT。在联合操作中，隐含列将作为结果的组成部份计数（即使未显示），但在正常查询中，隐含列不参与DISTINCT比较。在以后，我们可能会更改该情况，在执行DISTINCT时不比较隐含列。

例如：

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

以及

```
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;
```

在第2种情况下，使用MySQL服务器3.23.x，可在结果集中获得2个等同行（这是因为，隐藏ID列中的值可能不同）。

注意，在结果集中，仅对不含ORDER BY列的查询才会出现该情况。

- 如果在返回空集的查询上执行PROCEDURE，在某些情况下，PROCEDURE不转换列。
- 创建具有MERGE类型的表时，不检查基本表是否具有兼容的类型。
- 如果使用ALTER TABLE为MERGE表中使用的表增加了UNIQUE索引，然后在MERGE表上增加了正常索引，如果在表中存在旧的、非UNIQUE键，对于这些表，键顺序是不同的。这是因为，ALTER TABLE会将UNIQUE索引放在正常索引之前，以便能尽早检测到重复的键。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

附录B：错误代码和消息

目录

B.1. 服务器错误代码和消息
B.2. 客户端错误代码和消息

本章列出了当你用任何主机语言调用MySQL时可能出现的错误。首先列出了服务器错误消息。其次列出了客户端程序消息。

B.1. 服务器错误代码和消息

服务器错误信息来自下述源文件：

· 错误消息信息列在share/errmsg.txt文件中。"%d"和"%s"分别代表编号和字符串，显示时，它们将被消息值取代。

· 错误值列在share/errmsg.txt文件中，用于生成include/mysqld_error.h和include/mysqld_ername.h MySQL源文件中的定义。

· SQLSTATE值列在share/errmsg.txt文件中，用于生成include/sql_state.h MySQL源文件中的定义。

由于更新很频繁，这些文件中可能包含这里未列出的额外错误消息。

· 错误：1000 SQLSTATE: HY000 (ER_HASHCHK)

消息：hashchk

· 错误：1001 SQLSTATE: HY000 (ER_NISAMCHK)

消息：isamchk

· 错误：1002 SQLSTATE: HY000 (ER_NO)

消息：NO

· 错误：1003 SQLSTATE: HY000 (ER_YES)

消息：YES

· 错误：1004 SQLSTATE: HY000 (ER_CANT_CREATE_FILE)

消息：无法创建文件'%s' (errno: %d)

· 错误：1005 SQLSTATE: HY000 (ER_CANT_CREATE_TABLE)

消息：无法创建表'%s' (errno: %d)

· 错误：1006 SQLSTATE: HY000 (ER_CANT_CREATE_DB)

消息：无法创建数据库'%s' (errno: %d)

· 错误：1007 SQLSTATE: HY000 (ER_DB_CREATE_EXISTS)

消息：无法创建数据库'%s'，数据库已存在。

- 错误：1008 SQLSTATE: HY000 (ER_DB_DROP_EXISTS)

消息：无法撤销数据库'%s'，数据库不存在。

- 错误：1009 SQLSTATE: HY000 (ER_DB_DROP_DELETE)

消息：撤销数据库时出错（无法删除'%s'，errno: %d）

- 错误：1010 SQLSTATE: HY000 (ER_DB_DROP_RMDIR)

消息：撤销数据库时出错（can't rmdir '%s', errno: %d）

- 错误：1011 SQLSTATE: HY000 (ER_CANT_DELETE_FILE)

消息：删除'%s'时出错 (errno: %d)

- 错误：1012 SQLSTATE: HY000 (ER_CANT_FIND_SYSTEM_REC)

消息：无法读取系统表中的记录。

- 错误：1013 SQLSTATE: HY000 (ER_CANT_GET_STAT)

消息：无法获取'%s'的状态(errno: %d)

- 错误：1014 SQLSTATE: HY000 (ER_CANT_GET_WD)

消息：无法获得工作目录(errno: %d)

- 错误：1015 SQLSTATE: HY000 (ER_CANT_LOCK)

消息：无法锁定文件(errno: %d)

- 错误：1016 SQLSTATE: HY000 (ER_CANT_OPEN_FILE)

消息：无法打开文件：'%s' (errno: %d)

- 错误：1017 SQLSTATE: HY000 (ER_FILE_NOT_FOUND)

消息：无法找到文件： '%s' (errno: %d)

- 错误：1018 SQLSTATE: HY000 (ER_CANT_READ_DIR)

消息：无法读取'%s'的目录 (errno: %d)

- 错误：1019 SQLSTATE: HY000 (ER_CANT_SET_WD)

消息：无法为'%s'更改目录 (errno: %d)

- 错误：1020 SQLSTATE: HY000 (ER_CHECKREAD)

消息：自上次读取以来表'%s'中的记录已改变。

- 错误：1021 SQLSTATE: HY000 (ER_DISK_FULL)

消息：磁盘满(%s)；等待某人释放一些空间...

- 错误：1022 SQLSTATE: 23000 (ER_DUP_KEY)

消息：无法写入；复制表'%s'的 键。

- 错误：1023 SQLSTATE: HY000 (ER_ERROR_ON_CLOSE)

消息：关闭'%s'时出错 (errno: %d)

- 错误：1024 SQLSTATE: HY000 (ER_ERROR_ON_READ)

消息：读取文件'%s'时出错 (errno: %d)

- 错误：1025 SQLSTATE: HY000 (ER_ERROR_ON_RENAME)

消息：将'%s'重命名为'%s'时出错 (errno: %d)

- 错误：1026 SQLSTATE: HY000 (ER_ERROR_ON_WRITE)

消息：写入文件'%s'时出错 (errno: %d)

- 错误：1027 SQLSTATE: HY000 (ER_FILE_USED)

消息：'%s'已锁定，拒绝更改。

- 错误：1028 SQLSTATE: HY000 (ER_FILSORT_ABORT)

消息：分类失败

- 错误：1029 SQLSTATE: HY000 (ER_FORM_NOT_FOUND)

消息：对于'%s'，视图'%s'不存在。

- 错误：1030 SQLSTATE: HY000 (ER_GET_ERRNO)

消息：从存储引擎中获得错误%d。

- 错误：1031 SQLSTATE: HY000 (ER_ILLEGAL_HA)

消息：关于'%s'的表存储引擎不含该选项。

- 错误：1032 SQLSTATE: HY000 (ER_KEY_NOT_FOUND)

消息：无法在'%s'中找到记录。

- 错误：1033 SQLSTATE: HY000 (ER_NOT_FORM_FILE)

消息：文件中的不正确信息：'%s'

- 错误：1034 SQLSTATE: HY000 (ER_NOT_KEYFILE)

消息：对于表'%s'， 键文件不正确，请尝试修复。

- 错误：1035 SQLSTATE: HY000 (ER_OLD_KEYFILE)

消息：旧的键文件，对于表'%s'，请修复之！

- 错误：1036 SQLSTATE: HY000 (ER_OPEN_AS_READONLY)

消息：表'%s'是只读的。

- 错误：1037 SQLSTATE: HY001 (ER_OUTOFMEMORY)

消息：内存溢出，重启服务器并再次尝试（需要%d字节）。

- 错误：1038 SQLSTATE: HY001 (ER_OUT_OF_SORTMEMORY)

消息：分类内存溢出，增加服务器的分类缓冲区大小。

- 错误：1039 SQLSTATE: HY000 (ER_UNEXPECTED_EOF)

消息：读取文件'%s'时出现意外EOF (errno: %d)

- 错误：1040 SQLSTATE: 08004 (ER_CON_COUNT_ERROR)

消息：连接过多。

- 错误：1041 SQLSTATE: HY000 (ER_OUT_OF_RESOURCES)

消息：内存溢出，请检查是否mysqld或其他进程使用了所有可用内存，如不然，或许应使用'ulimit'允许mysqld使用更多内存，或增加交换空间的大小。

- 错误：1042 SQLSTATE: 08S01 (ER_BAD_HOST_ERROR)

消息：无法获得该地址给出的主机名。

- 错误：1043 SQLSTATE: 08S01 (ER_HANDSHAKE_ERROR)

消息：不良握手

- 错误：1044 SQLSTATE: 42000 (ER_DBACCESS_DENIED_ERROR)

消息：拒绝用户'%s'@'%s'访问数据库'%s'。

- 错误：1045 SQLSTATE: 28000 (ER_ACCESS_DENIED_ERROR)

消息：拒绝用户'%s'@'%s'的访问（使用密码：%s）

- 错误：1046 SQLSTATE: 3D000 (ER_NO_DB_ERROR)

消息：未选择数据库。

- 错误：1047 SQLSTATE: 08S01 (ER_UNKNOWN_COM_ERROR)

消息：未知命令。

- 错误：1048 SQLSTATE: 23000 (ER_BAD_NULL_ERROR)

消息：列'%s'不能为空。

- 错误：1049 SQLSTATE: 42000 (ER_BAD_DB_ERROR)

消息：未知数据库'%s'。

- 错误：1050 SQLSTATE: 42S01 (ER_TABLE_EXISTS_ERROR)

消息：表'%s'已存在。

- 错误：1051 SQLSTATE: 42S02 (ER_BAD_TABLE_ERROR)

消息：未知表'%s'。

- 错误：1052 SQLSTATE: 23000 (ER_NON_UNIQ_ERROR)

消息：%s中的列'%s'不明确。

- 错误：1053 SQLSTATE: 08S01 (ER_SERVER_SHUTDOWN)

消息：在操作过程中服务器关闭。

- 错误：1054 SQLSTATE: 42S22 (ER_BAD_FIELD_ERROR)

消息：'%s'中的未知列'%s'。

- 错误：1055 SQLSTATE: 42000 (ER_WRONG_FIELD_WITH_GROUP)

消息：'%s'不在GROUP BY中。

- 错误：1056 SQLSTATE: 42000 (ER_WRONG_GROUP_FIELD)

消息：无法在'%s'上创建组。

- 错误：1057 SQLSTATE: 42000 (ER_WRONG_SUM_SELECT)

消息：语句中有sum函数和相同语句中的列。

- 错误：1058 SQLSTATE: 21S01 (ER_WRONG_VALUE_COUNT)

消息：列计数不匹配值计数。

- 错误：1059 SQLSTATE: 42000 (ER_TOO_LONG_IDENT)

消息：ID名称'%s'过长。

- 错误：1060 SQLSTATE: 42S21 (ER_DUP_FIELDNAME)

消息：重复列名'%s'。

- 错误：1061 SQLSTATE: 42000 (ER_DUP_KEYNAME)

消息：重复键名称'%s'。

- 错误：1062 SQLSTATE: 23000 (ER_DUP_ENTRY)

消息：键%d的重复条目'%s'。

- 错误：1063 SQLSTATE: 42000 (ER_WRONG_FIELD_SPEC)

消息：对于列'%s'，列分类符不正确。

- 错误：1064 SQLSTATE: 42000 (ER_PARSE_ERROR)

消息：在行%d上，%s靠近'%s'。

- 错误：1065 SQLSTATE: 42000 (ER_EMPTY_QUERY)

消息：查询为空。

- 错误：1066 SQLSTATE: 42000 (ER_NONUNIQ_TABLE)

消息：非唯一的表/别名：'%s'

- 错误：1067 SQLSTATE: 42000 (ER_INVALID_DEFAULT)

消息：关于'%s'的无效默认值。

- 错误：1068 SQLSTATE: 42000 (ER_MULTIPLE_PRI_KEY)

消息：定义了多个主键。

- 错误：1069 SQLSTATE: 42000 (ER_TOO_MANY_KEYS)

消息：指定了过多键：允许的最大键数是%d。

- 错误：1070 SQLSTATE: 42000 (ER_TOO_MANY_KEY_PARTS)

消息：指定了过多键部分：允许的最大键部分是%d。

- 错误：1071 SQLSTATE: 42000 (ER_TOO_LONG_KEY)

消息：指定的键过长，最大键长度是%d字节。

- 错误：1072 SQLSTATE: 42000 (ER_KEY_COLUMN_DOES_NOT_EXITS)

消息：键列'%s'在表中不存在。

- 错误：1073 SQLSTATE: 42000 (ER_BLOB_USED_AS_KEY)

消息：BLOB列'%s'不能与已使用的表类型用在 键说明中。

- 错误：1074 SQLSTATE: 42000 (ER_TOO_BIG_FIELDLENGTH)

消息：对于列'%s'，列长度过大 (max = %d)，请使用BLOB或TEXT取而代之。

- 错误：1075 SQLSTATE: 42000 (ER_WRONG_AUTO_KEY)

消息：不正确的表定义，只能有1个auto列，而且必须将其定义为 键。

- 错误：1076 SQLSTATE: HY000 (ER_READY)

消息：%s，连接就绪。版本：'%s'，套接字：'%s'，端口：%d

- 错误：1077 SQLSTATE: HY000 (ER_NORMAL_SHUTDOWN)

消息：%s，正常关闭。

- 错误：1078 SQLSTATE: HY000 (ER_GOT_SIGNAL)

消息：%s，获得信号%d。放弃！

- 错误：1079 SQLSTATE: HY000 (ER_SHUTDOWN_COMPLETE)

消息：%s，关闭完成

- 错误：1080 SQLSTATE: 08S01 (ER_FORCING_CLOSE)

消息：%s，强制关闭线程%ld，用户：'%s'

- 错误：1081 SQLSTATE: 08S01 (ER_IPSOCK_ERROR)

消息：无法创建IP套接字

- 错误：1082 SQLSTATE: 42S12 (ER_NO_SUCH_INDEX)

消息：表'%s'中没有与CREATE INDEX中索引类似的索引，重新创建表。

- 错误：1083 SQLSTATE: 42000 (ER_WRONG_FIELD_TERMINATORS)

消息：字段分隔符参量不是预期的，请参考手册。

- 错误：1084 SQLSTATE: 42000 (ER_BLOBS_AND_NO_TERMINATED)

消息：不能与BLOB一起使用固定行长度，请使用'fields terminated by'。

- 错误：1085 SQLSTATE: HY000 (ER_TEXTFILE_NOT_READABLE)

消息：文件'%s'必须在数据库目录下，或能被所有人读取。

- 错误：1086 SQLSTATE: HY000 (ER_FILE_EXISTS_ERROR)

消息：文件'%s'已存在。

- 错误：1087 SQLSTATE: HY000 (ER_LOAD_INFO)

消息：记录，%ld；已删除，%ld；已跳过，%ld；警告，%ld

- 错误：1088 SQLSTATE: HY000 (ER_ALTER_INFO)

消息：记录，%ld；重复，%ld

- 错误：1089 SQLSTATE: HY000 (ER_WRONG_SUB_KEY)

消息：不正确的子部分键，使用的键部分不是字符串，所用的长度长于键部分，或存储引擎不支持唯一子键。

- 错误：1090 SQLSTATE: 42000 (ER_CANT_REMOVE_ALL_FIELDS)

消息：不能用ALTER TABLE删除所有列，请使用DROP TABLE取而代之。

- 错误：1091 SQLSTATE: 42000 (ER_CANT_DROP_FIELD_OR_KEY)

消息：不能撤销'%s'，请检查列/键是否存在。

- 错误：1092 SQLSTATE: HY000 (ER_INSERT_INFO)

消息：记录，%ld；复制，%ld；告警，%ld

- 错误：1093 SQLSTATE: HY000 (ER_UPDATE_TABLE_USED)

消息：不能在FROM子句中制定要更新的目标表'%s'。

- 错误：1094 SQLSTATE: HY000 (ER_NO_SUCH_THREAD)

消息：未知线程ID：%lu

- 错误：1095 SQLSTATE: HY000 (ER_KILL_DENIED_ERROR)

消息：你不是线程%lu的所有者。

- 错误：1096 SQLSTATE: HY000 (ER_NO_TABLES_USED)

消息：未使用任何表。

- 错误：1097 SQLSTATE: HY000 (ER_TOO_BIG_SET)

消息：列%s和SET的字符串过多。

- 错误：1098 SQLSTATE: HY000 (ER_NO_UNIQUE_LOGFILE)

消息：不能生成唯一的日志文件名%s.(1-999)

- 错误：1099 SQLSTATE: HY000 (ER_TABLE_NOT_LOCKED_FOR_WRITE)

消息：表'%s'已用READ锁定，不能更新。

- 错误：1100 SQLSTATE: HY000 (ER_TABLE_NOT_LOCKED)

消息：未使用LOCK TABLES锁定表'%s'。

- 错误：1101 SQLSTATE: 42000 (ER_BLOB_CANT_HAVE_DEFAULT)

消息：BLOB/TEXT列'%s'不能有默认值。

- 错误：1102 SQLSTATE: 42000 (ER_WRONG_DB_NAME)

消息：不正确的数据库名'%s'。

- 错误：1103 SQLSTATE: 42000 (ER_WRONG_TABLE_NAME)

消息：不正确的表名'%s'。

- 错误：1104 SQLSTATE: 42000 (ER_TOO_BIG_SELECT)

消息：SELECT将检查超过MAX_JOIN_SIZE的行，如果SELECT正常，请检查WHERE，并使用SET SQL_BIG_SELECTS=1或SET SQL_MAX_JOIN_SIZE=#。

- 错误：1105 SQLSTATE: HY000 (ER_UNKNOWN_ERROR)

消息：未知错误。

- 错误：1106 SQLSTATE: 42000 (ER_UNKNOWN_PROCEDURE)

消息：未知过程'%s'

- 错误：1107 SQLSTATE: 42000 (ER_WRONG_PARAMCOUNT_TO_PROCEDURE)

消息：对于过程'%s'，参数计数不正确

- 错误：1108 SQLSTATE: HY000 (ER_WRONG_PARAMETERS_TO_PROCEDURE)

消息：对于过程'%s'，参数不正确

- 错误：1109 SQLSTATE: 42S02 (ER_UNKNOWN_TABLE)

消息：%s中的未知表%s

- 错误：1110 SQLSTATE: 42000 (ER_FIELD_SPECIFIED_TWICE)

消息：列'%s'被指定了两次。

- 错误：1111 SQLSTATE: HY000 (ER_INVALID_GROUP_FUNC_USE)

消息：无效的分组函数使用

- 错误：1112 SQLSTATE: 42000 (ER_UNSUPPORTED_EXTENSION)

消息：表'%s'使用了该MySQL版本中不存在的扩展。

- 错误：1113 SQLSTATE: 42000 (ER_TABLE_MUST_HAVE_COLUMNS)

消息：1个表至少要有1列。

- 错误：1114 SQLSTATE: HY000 (ER_RECORD_FILE_FULL)

消息：表'%s'已满。

- 错误：1115 SQLSTATE: 42000 (ER_UNKNOWN_CHARACTER_SET)

消息：未知字符集'%s'。

- 错误：1116 SQLSTATE: HY000 (ER_TOO_MANY_TABLES)

消息：表过多，MySQL在1个联合操作中只能使用%d个表。

- 错误：1117 SQLSTATE: HY000 (ER_TOO_MANY_FIELDS)

消息：列过多。

- 错误：1118 SQLSTATE: 42000 (ER_TOO_BIG_ROWSIZE)

消息：行的大小过大。对于所使用的表类型，不包括BLOB，最大行大小为%ld。必须将某些列更改为TEXT或BLOB。

- 错误：1119 SQLSTATE: HY000 (ER_STACK_OVERRUN)

消息：线程堆栈溢出，已使用，%ld堆栈的%ld。如果需要，请使用'mysqld -O thread_stack=#'指定较大的堆栈。

- 错误：1120 SQLSTATE: 42000 (ER_WRONG_OUTER_JOIN)

消息：在OUTER JOIN中发现交叉关联，请检查ON条件。

- 错误：1121 SQLSTATE: 42000 (ER_NULL_COLUMN_IN_INDEX)

消息：列'%s'与UNIQUE或INDEX一起使用，但未定义为NOT NULL。

- 错误：1122 SQLSTATE: HY000 (ER_CANT_FIND_UDF)

消息：无法加载函数'%s'。

- 错误：1123 SQLSTATE: HY000 (ER_CANT_INITIALIZE_UDF)

消息：无法初始化函数'%s'; %s

- 错误：1124 SQLSTATE: HY000 (ER_UDF_NO_PATHS)

消息：对于共享库，不允许任何路径。

- 错误：1125 SQLSTATE: HY000 (ER_UDF_EXISTS)

消息：函数'%s'已存在。

- 错误：1126 SQLSTATE: HY000 (ER_CANT_OPEN_LIBRARY)

消息：不能打开共享库'%s' (errno: %d %s)

- 错误：1127 SQLSTATE: HY000 (ER_CANT_FIND_DL_ENTRY)

消息：不能发现库中的符号'%s'。

- 错误：1128 SQLSTATE: HY000 (ER_FUNCTION_NOT_DEFINED)

消息：函数'%s'未定义。

- 错误：1129 SQLSTATE: HY000 (ER_HOST_IS_BLOCKED)

消息：由于存在很多连接错误，主机'%s'被屏蔽，请用'mysqladmin flush-hosts'解除屏蔽。

- 错误：1130 SQLSTATE: HY000 (ER_HOST_NOT_PRIVILEGED)

消息：不允许将主机'%s'连接到该MySQL服务器。

- 错误：1131 SQLSTATE: 42000 (ER_PASSWORD_ANONYMOUS_USER)

消息：你正在已匿名用户身份使用MySQL，不允许匿名用户更改密码。

- 错误：1132 SQLSTATE: 42000 (ER_PASSWORD_NOT_ALLOWED)

消息：必须有更新mysql数据库中表的权限才能更改密码。

- 错误：1133 SQLSTATE: 42000 (ER_PASSWORD_NO_MATCH)

消息：无法在用户表中找到匹配行。

- 错误：1134 SQLSTATE: HY000 (ER_UPDATE_INFO)

消息：行匹配，%ld；已更改，%ld；警告，%ld

- 错误：1135 SQLSTATE: HY000 (ER_CANT_CREATE_THREAD)

消息：无法创建新线程(errno %d)，如果未出现内存溢出，请参阅手册以了解可能的与操作系统有关的缺陷。

- 错误：1136 SQLSTATE: 21S01 (ER_WRONG_VALUE_COUNT_ON_ROW)

消息：列计数不匹配行%ld上的值计数。

- 错误：1137 SQLSTATE: HY000 (ER_CANT_REOPEN_TABLE)

消息：无法再次打开表'%s'。

- 错误：1138 SQLSTATE: 22004 (ER_INVALID_USE_OF_NULL)

消息：NULL值使用无效。

- 错误：1139 SQLSTATE: 42000 (ER_REGEXP_ERROR)

消息：获得来自regexp的错误'%s'。

- 错误：1140 SQLSTATE: 42000 (ER_MIX_OF_GROUP_FUNC_AND_FIELDS)

消息：如果没有GROUP BY子句，GROUP列 (MIN(),MAX(),COUNT(),...)与非GROUP列的混合不合法。

- 错误：1141 SQLSTATE: 42000 (ER_NONEXISTING_GRANT)

消息：没有为主机'%s'上的用户'%s'定义这类授权。

- 错误：1142 SQLSTATE: 42000 (ER_TABLEACCESS_DENIED_ERROR)

消息：拒绝用户'%s'@'%s'在表'%s'上使用%s命令。

- 错误：1143 SQLSTATE: 42000 (ER_COLUMNACCESS_DENIED_ERROR)

消息：拒绝用户'%s'@'%s'在表'%s'的'%s'上使用%s命令。

- 错误：1144 SQLSTATE: 42000 (ER_ILLEGAL_GRANT_FOR_TABLE)

消息：非法GRANT/REVOKE命令，请参阅手册以了解可使用那种权限。

- 错误：1145 SQLSTATE: 42000 (ER_GRANT_WRONG_HOST_OR_USER)

消息：GRANT的主机或用户参量过长。

- 错误：1146 SQLSTATE: 42S02 (ER_NO_SUCH_TABLE)

消息：表'%s.%s'不存在。

- 错误：1147 SQLSTATE: 42000 (ER_NONEXISTING_TABLE_GRANT)

消息：在表'%s'上没有为主机'%s'上的用户'%s'定义的这类授权。

- 错误：1148 SQLSTATE: 42000 (ER_NOT_ALLOWED_COMMAND)

消息：所使用的命令在该MySQL版本中不允许。

- 错误：1149 SQLSTATE: 42000 (ER_SYNTAX_ERROR)

消息：存在SQL语法错误，请参阅与你的MySQL版本对应的手册，以了解正确的语法。

- 错误：1150 SQLSTATE: HY000 (ER_DELAYED_CANT_CHANGE_LOCK)

消息：对于表%s，延迟的插入线程不能获得请求的锁定。

- 错误：1151 SQLSTATE: HY000 (ER_TOO_MANY_DELAYED_THREADS)

消息：使用的延迟线程过多。

- 错误：1152 SQLSTATE: 08S01 (ER_ABORTING_CONNECTION)

消息：与数据库'%s'和用户'%s'的连接%ld失败 (%s)

- 错误：1153 SQLSTATE: 08S01 (ER_NET_PACKET_TOO_LARGE)

消息：获得信息包大于'max_allowed_packet'字节。

- 错误：1154 SQLSTATE: 08S01 (ER_NET_READ_ERROR_FROM_PIPE)

消息：获得来自连接管道的读错误。

- 错误：1155 SQLSTATE: 08S01 (ER_NET_FCNTL_ERROR)

消息：获得来自fcntl()的错误。

- 错误：1156 SQLSTATE: 08S01 (ER_NET_PACKETS_OUT_OF_ORDER)

消息：获得信息包无序。

- 错误：1157 SQLSTATE: 08S01 (ER_NET_UNCOMPRESS_ERROR)

消息：无法解压缩通信信息包。

- 错误：1158 SQLSTATE: 08S01 (ER_NET_READ_ERROR)

消息：读取通信信息包时出错。

- 错误：1159 SQLSTATE: 08S01 (ER_NET_READ_INTERRUPTED)

消息：读取通信信息包时出现超时。

- 错误：1160 SQLSTATE: 08S01 (ER_NET_ERROR_ON_WRITE)

消息：写入通信信息包时出错。

- 错误：1161 SQLSTATE: 08S01 (ER_NET_WRITE_INTERRUPTED)

消息：写入通信信息包时出现超时。

- 错误：1162 SQLSTATE: 42000 (ER_TOO_LONG_STRING)

消息：结果字符串长于'max_allowed_packet'字节。

- 错误：1163 SQLSTATE: 42000 (ER_TABLE_CANT_HANDLE_BLOB)

消息：所使用的表类型不支持BLOB/TEXT列。

- 错误：1164 SQLSTATE: 42000 (ER_TABLE_CANT_HANDLE_AUTO_INCREMENT)

消息：所使用的表类型不支持AUTO_INCREMENT列。

- 错误：1165 SQLSTATE: HY000 (ER_DELAYED_INSERT_TABLE_LOCKED)

消息：由于用LOCK TABLES锁定了表，INSERT DELAYED不能与表'%s'一起使用。

- 错误：1166 SQLSTATE: 42000 (ER_WRONG_COLUMN_NAME)

消息：不正确的列名'%s'。

- 错误：1167 SQLSTATE: 42000 (ER_WRONG_KEY_COLUMN)

消息：所使用的存储引擎不能为列'%s'编制索引。

- 错误：1168 SQLSTATE: HY000 (ER_WRONG_MRG_TABLE)

消息：MERGE表中的所有表未同等定义。

- 错误：1169 SQLSTATE: 23000 (ER_DUP_UNIQUE)

消息：由于唯一性限制，不能写入到表'%s'。

- 错误：1170 SQLSTATE: 42000 (ER_BLOB_KEY_WITHOUT_LENGTH)

消息：在未指定键长度的键说明中使用了BLOB/TEXT列'%s'。

- 错误：1171 SQLSTATE: 42000 (ER_PRIMARY_CANT_HAVE_NULL)

消息：PRIMARY KEY的所有部分必须是NOT NULL，如果需要为NULL的关键字，请使用UNIQUE取而代之。

- 错误：1172 SQLSTATE: 42000 (ER_TOO_MANY_ROWS)

消息：结果有1个以上的行组成。

- 错误：1173 SQLSTATE: 42000 (ER_REQUIRES_PRIMARY_KEY)

消息：该表类型要求主键。

- 错误：1174 SQLSTATE: HY000 (ER_NO_RAID_COMPILED)

消息：该MySQL版本是未使用RAID支持而编译的。

- 错误：1175 SQLSTATE: HY000 (ER_UPDATE_WITHOUT_KEY_IN_SAFE_MODE)

消息：你正在使用安全更新模式，而且试图在不使用WHERE的情况下更新使用了KEY列的表。

- 错误：1176 SQLSTATE: HY000 (ER_KEY_DOES_NOT_EXITS)

消息：在表'%s'中，键'%s'不存在。

- 错误：1177 SQLSTATE: 42000 (ER_CHECK_NO_SUCH_TABLE)

消息：无法打开表。

- 错误：1178 SQLSTATE: 42000 (ER_CHECK_NOT_IMPLEMENTED)

消息：用于表的引擎不支持%s。

- 错误：1179 SQLSTATE: 25000 (ER_CANT_DO_THIS_DURING_AN_TRANSACTION)

消息：不允许在事务中执行该命令。

- 错误：1180 SQLSTATE: HY000 (ER_ERROR_DURING_COMMIT)

消息：在COMMIT期间出现错误%d。

- 错误：1181 SQLSTATE: HY000 (ER_ERROR_DURING_ROLLBACK)

消息：在ROLLBACK期间出现错误%d。

- 错误：1182 SQLSTATE: HY000 (ER_ERROR_DURING_FLUSH_LOGS)

消息：在FLUSH_LOGS期间出现错误%d。

- 错误：1183 SQLSTATE: HY000 (ER_ERROR_DURING_CHECKPOINT)

消息：在CHECKPOINT期间出现错误%d。

- 错误：1184 SQLSTATE: 08S01 (ER_NEW_ABORTING_CONNECTION)

消息：与数据库'%s'、用户'%s'和主机'%s'的连接%ld失败 (%s)。

- 错误：1185 SQLSTATE: HY000 (ER_DUMP_NOT_IMPLEMENTED)

消息：针对表的存储引擎不支持二进制表转储。

- 错误：1186 SQLSTATE: HY000 (ER_FLUSH_MASTER_BINLOG_CLOSED)

消息：Binlog已关闭，不能RESET MASTER。

- 错误：1187 SQLSTATE: HY000 (ER_INDEX_REBUILD)

消息：重新创建转储表'%s'的索引失败。

- 错误：1188 SQLSTATE: HY000 (ER_MASTER)

消息：来自主连接'%s'的错误。

- 错误：1189 SQLSTATE: 08S01 (ER_MASTER_NET_READ)

消息：读取主连接时出现网络错误。

- 错误：1190 SQLSTATE: 08S01 (ER_MASTER_NET_WRITE)

消息：写入主连接时出现网络错误。

- 错误：1191 SQLSTATE: HY000 (ER_FT_MATCHING_KEY_NOT_FOUND)

消息：无法找到与列列表匹配的FULLTEXT索引。

- 错误：1192 SQLSTATE: HY000 (ER_LOCK_OR_ACTIVE_TRANSACTION)

消息：由于存在活动的锁定表或活动的事务，不能执行给定的命令。

- 错误：1193 SQLSTATE: HY000 (ER_UNKNOWN_SYSTEM_VARIABLE)

消息：未知的系统变量'%s'。

- 错误：1194 SQLSTATE: HY000 (ER_CRASHED_ON_USAGE)

消息：表'%s'被标记为崩溃，应予以修复。

- 错误：1195 SQLSTATE: HY000 (ER_CRASHED_ON_REPAIR)

消息：表'%s'被标记为崩溃，而且上次修复失败（自动？）

- 错误：1196 SQLSTATE: HY000 (ER_WARNING_NOT_COMPLETE_ROLLBACK)

消息：不能回滚某些非事务性已变动表。

- 错误：1197 SQLSTATE: HY000 (ER_TRANS_CACHE_FULL)

消息：多语句事务要求更多的'max_binlog_cache_size'存储字节，增大mysqld变量，并再次尝试。

- 错误：1198 SQLSTATE: HY000 (ER_SLAVE_MUST_STOP)

消息：运行从实例时不能执行该操作，请首先运行STOP SLAVE。

- 错误：1199 SQLSTATE: HY000 (ER_SLAVE_NOT_RUNNING)

消息：该操作需要运行的从实例，请配置SLAVE并执行START SLAVE。

- 错误：1200 SQLSTATE: HY000 (ER_BAD_SLAVE)

消息：服务器未配置为从服务器，请更正config文件，或使用CHANGE MASTER TO。

- 错误：1201 SQLSTATE: HY000 (ER_MASTER_INFO)

消息：无法初始化主服务器信息结构，在MySQL错误日志中可找到更多错误消息。

- 错误：1202 SQLSTATE: HY000 (ER_SLAVE_THREAD)

消息：无法创建从线程，请检查系统资源。

- 错误：1203 SQLSTATE: 42000 (ER_TOO_MANY_USER_CONNECTIONS)

消息：用户%s已有了超过'max_user_connections'的活动连接。

- 错误：1204 SQLSTATE: HY000 (ER_SET_CONSTANTS_ONLY)

消息：或许仅应与SET一起使用常量表达式。

- 错误：1205 SQLSTATE: HY000 (ER_LOCK_WAIT_TIMEOUT)

消息：超过了锁定等待超时，请尝试重新启动事务。

- 错误：1206 SQLSTATE: HY000 (ER_LOCK_TABLE_FULL)

消息：总的锁定数超出了锁定表的大小。

- 错误：1207 SQLSTATE: 25000 (ER_READ_ONLY_TRANSACTION)

消息：在READ UNCOMMITTED事务期间，无法获得更新锁定。

- 错误：1208 SQLSTATE: HY000 (ER_DROP_DB_WITH_READ_LOCK)

消息：当线程保持为全局读锁定时，不允许DROP DATABASE。

- 错误：1209 SQLSTATE: HY000 (ER_CREATE_DB_WITH_READ_LOCK)

消息：当线程保持为全局读锁定时，不允许CREATE DATABASE。

- 错误：1210 SQLSTATE: HY000 (ER_WRONG_ARGUMENTS)

消息：为%s提供的参量不正确。

- 错误：1211 SQLSTATE: 42000 (ER_NO_PERMISSION_TO_CREATE_USER)

消息：不允许'%s'@'%s'创建新用户。

- 错误：1212 SQLSTATE: HY000 (ER_UNION_TABLES_IN_DIFFERENT_DIR)

消息：不正确的表定义，所有的MERGE表必须位于相同的数据库中。

- 错误：1213 SQLSTATE: 40001 (ER_LOCK_DEADLOCK)

消息：试图获取锁定时发现死锁，请尝试重新启动事务。

- 错误：1214 SQLSTATE: HY000 (ER_TABLE_CANT_HANDLE_FT)

消息：所使用的表类型不支持FULLTEXT索引。

- 错误：1215 SQLSTATE: HY000 (ER_CANNOT_ADD_FOREIGN)

消息：无法添加外键约束。

- 错误：1216 SQLSTATE: 23000 (ER_NO_REFERENCED_ROW)

消息：无法添加或更新子行，外键约束失败。

- 错误：1217 SQLSTATE: 23000 (ER_ROW_IS_REFERENCED)

消息：无法删除或更新父行，外键约束失败。

- 错误：1218 SQLSTATE: 08S01 (ER_CONNECT_TO_MASTER)

消息：连接至主服务器%s时出错。

- 错误：1219 SQLSTATE: HY000 (ER_QUERY_ON_MASTER)

消息：在主服务器%s上执行查询时出错。

- 错误：1220 SQLSTATE: HY000 (ER_ERROR_WHEN_EXECUTING_COMMAND)

消息：执行命令%s: %s时出错。

- 错误：1221 SQLSTATE: HY000 (ER_WRONG_USAGE)

消息：%s和%s的用法不正确。

- 错误：1222 SQLSTATE: 21000 (ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT)

消息：所使用的SELECT语句有不同的列数。

- 错误：1223 SQLSTATE: HY000 (ER_CANT_UPDATE_WITH_READLOCK)

消息：由于存在冲突的读锁定，无法执行查询。

- 错误：1224 SQLSTATE: HY000 (ER_MIXING_NOT_ALLOWED)

消息：禁止混合事务性表和非事务性表。

- 错误：1225 SQLSTATE: HY000 (ER_DUP_ARGUMENT)

消息：在语句中使用了两次选项'%s'。

- 错误：1226 SQLSTATE: 42000 (ER_USER_LIMIT_REACHED)

消息：用户'%s'超出了'%s'资源（当前值：%ld）。

- 错误：1227 SQLSTATE: 42000 (ER_SPECIFIC_ACCESS_DENIED_ERROR)

消息：拒绝访问，需要%s权限才能执行该操作。

- 错误：1228 SQLSTATE: HY000 (ER_LOCAL_VARIABLE)

消息：变量'%s'是1种SESSION变量，不能与SET GLOBAL一起使用。

- 错误：1229 SQLSTATE: HY000 (ER_GLOBAL_VARIABLE)

消息：变量'%s'是1种GLOBAL变量，应使用SET GLOBAL来设置它。

- 错误：1230 SQLSTATE: 42000 (ER_NO_DEFAULT)

消息：变量'%s'没有默认值。

- 错误：1231 SQLSTATE: 42000 (ER_WRONG_VALUE_FOR_VAR)

消息：变量'%s'不能设置为值'%s'。

- 错误：1232 SQLSTATE: 42000 (ER_WRONG_TYPE_FOR_VAR)

消息：变量'%s'的参量类型不正确。

- 错误：1233 SQLSTATE: HY000 (ER_VAR_CANT_BE_READ)

消息：变量'%s'只能被设置，不能被读取。

- 错误：1234 SQLSTATE: 42000 (ER_CANT_USE_OPTION_HERE)

消息：不正确的'%s'用法/位置。

- 错误：1235 SQLSTATE: 42000 (ER_NOT_SUPPORTED_YET)

消息：该MySQL版本尚不支持'%s'。

- 错误：1236 SQLSTATE: HY000 (ER_MASTER_FATAL_ERROR_READING_BINLOG)

消息：从二进制日志读取数据时，获得来自主服务器的致命错误%d: '%s'。

- 错误：1237 SQLSTATE: HY000 (ER_SLAVE_IGNORED_TABLE)

消息：由于"replicate-*-table"规则，从SQL线程忽略了查询。。

- 错误：1238 SQLSTATE: HY000 (ER_INCORRECT_GLOBAL_LOCAL_VAR)

消息：变量'%s'是一种%s变量。

- 错误：1239 SQLSTATE: 42000 (ER_WRONG_FK_DEF)

消息：对于 '%s': %s， 外键定义不正确。

- 错误：1240 SQLSTATE: HY000 (ER_KEY_REF_DO_NOT_MATCH_TABLE_REF)

消息：键引用和表引用不匹配。

- 错误：1241 SQLSTATE: 21000 (ER_OPERAND_COLUMNS)

消息：操作数应包含%d列。

- 错误：1242 SQLSTATE: 21000 (ER_SUBQUERY_NO_1_ROW)

消息：子查询返回1行以上。

- 错误：1243 SQLSTATE: HY000 (ER_UNKNOWN_STMT_HANDLER)

消息：指定给%s的未知预处理语句句柄。

- 错误：1244 SQLSTATE: HY000 (ER_CORRUPT_HELP_DB)

消息：帮助数据库崩溃或不存在。

- 错误：1245 SQLSTATE: HY000 (ER_CYCLIC_REFERENCE)

消息：对子查询的循环引用。

- 错误：1246 SQLSTATE: HY000 (ER_AUTO_CONVERT)

消息：将列'%s'从%s转换为%s。

- 错误：1247 SQLSTATE: 42S22 (ER_ILLEGAL_REFERENCE)

消息：引用'%s'不被支持 (%s)。

- 错误：1248 SQLSTATE: 42000 (ER_DERIVED_MUST_HAVE_ALIAS)

消息：所有的导出表必须有自己的别名。

- 错误：1249 SQLSTATE: 01000 (ER_SELECT_REDUCED)

消息：在优化期间简化了选择%u。

- 错误：1250 SQLSTATE: 42000 (ER_TABLENAME_NOT_ALLOWED_HERE)

消息：来自某一SELECT的表'%s'不能在%s中使用。

- 错误：1251 SQLSTATE: 08004 (ER_NOT_SUPPORTED_AUTH_MODE)

消息：客户端不支持服务器请求的鉴定协议，请考虑升级MySQL客户端。

- 错误：1252 SQLSTATE: 42000 (ER_SPATIAL_CANT_HAVE_NULL)

消息：SPATIAL索引的所有部分必须是NOT NULL。

- 错误：1253 SQLSTATE: 42000 (ER_COLLATION_CHARSET_MISMATCH)

消息：对于CHARACTER SET '%s'，COLLATION '%s'无效。

- 错误：1254 SQLSTATE: HY000 (ER_SLAVE_WAS_RUNNING)

消息：从服务器正在运行。

- 错误：1255 SQLSTATE: HY000 (ER_SLAVE_WAS_NOT_RUNNING)

消息：从服务器已停止。

- 错误：1256 SQLSTATE: HY000 (ER_TOO_BIG_FOR_UNCOMPRESS)

消息：解压的数据过大，最大大小为%d（也可能是，解压数据的长度已损坏）。

- 错误：1257 SQLSTATE: HY000 (ER_ZLIB_Z_MEM_ERROR)

消息：ZLIB，无足够内存。

- 错误：1258 SQLSTATE: HY000 (ER_ZLIB_Z_BUF_ERROR)

消息：ZLIB，输出缓冲区内无足够空间（也可能是，解压数据的长度已损坏）。

- 错误：1259 SQLSTATE: HY000 (ER_ZLIB_Z_DATA_ERROR)

消息：ZLIB，输入数据已损坏。

- 错误：1260 SQLSTATE: HY000 (ER_CUT_VALUE_GROUP_CONCAT)

消息：%d行被GROUP_CONCAT()截去。

- 错误：1261 SQLSTATE: 01000 (ER_WARN_TOO_FEW_RECORDS)

消息：行%ld不包含所有列的数据。

- 错误：1262 SQLSTATE: 01000 (ER_WARN_TOO_MANY_RECORDS)

消息：行%ld被解短，它包含的数据大于输入列中的数据。

- 错误：1263 SQLSTATE: 22004 (ER_WARN_NULL_TO_NOTNULL)

消息：列被设为默认值，在行%ld上将NULL提供给了NOT NULL列。

- 错误：1264 SQLSTATE: 22003 (ER_WARN_DATA_OUT_OF_RANGE)

消息：为行%ld上的列'%s'调整超出范围的值。

- 错误：1265 SQLSTATE: 01000 (WARN_DATA_TRUNCATED)

消息：为行%ld上的列'%s'截短数据。

- 错误：1266 SQLSTATE: HY000 (ER_WARN_USING_OTHER_HANDLER)

消息：为表%s使用存储引擎%s。

- 错误：1267 SQLSTATE: HY000 (ER_CANT_AGGREGATE_2COLLATIONS)

消息：对于操作'%s'，非法混合了校对(%s,%s)和(%s,%s)。

- 错误：1268 SQLSTATE: HY000 (ER_DROP_USER)

消息：无法撤销1个或多个请求的用户。

- 错误：1269 SQLSTATE: HY000 (ER_REVOKE_GRANTS)

消息：无法撤销所有权限，为1个或多个请求的用户授权。

- 错误：1270 SQLSTATE: HY000 (ER_CANT_AGGREGATE_3COLLATIONS)

消息：对于操作'%s'，非法混合了校对(%s,%s)、(%s,%s)和(%s,%s)。

- 错误：1271 SQLSTATE: HY000 (ER_CANT_AGGREGATE_NCOLLATIONS)

消息：对于操作'%s'，非法混合了校对。

- 错误：1272 SQLSTATE: HY000 (ER_VARIABLE_IS_NOT_STRUCT)

消息：变量'%s'不是变量组分（不能用作XXXX.variable_name）。

- 错误：1273 SQLSTATE: HY000 (ER_UNKNOWN_COLLATION)

消息：未知校对'%s'。

- 错误：1274 SQLSTATE: HY000 (ER_SLAVE_IGNORED_SSL_PARAMS)

消息：由于该MySQL从服务器是在不支持SSL的情况下编译的，CHANGE MASTER中的SSL参数被忽略，随后，如果启动了具备SSL功能的MySQL，可使用这些参数。

- 错误：1275 SQLSTATE: HY000 (ER_SERVER_IS_IN_SECURE_AUTH_MODE)

消息：服务器正运行在"--secure-auth"模式下，但'%s'@'%s'有1个采用旧格式的密码，请将密码更改为新格式。

- 错误：1276 SQLSTATE: HY000 (ER_WARN_FIELD_RESOLVED)

消息：SELECT #%d的字段或引用'%s%s%s%s%s'是在SELECT #%d中确定的。

- 错误：1277 SQLSTATE: HY000 (ER_BAD_SLAVE_UNTIL_COND)

消息：对于START SLAVE UNTIL，不正确的参数或参数组合。

- 错误：1278 SQLSTATE: HY000 (ER_MISSING_SKIP_SLAVE)

消息：与START SLAVE UNTIL一起执行按步复制时，建议使用"--skip-slave-start"，否则，如果发生未预料的从服务器mysqld重启，间出现问题。

- 错误：1279 SQLSTATE: HY000 (ER_UNTIL_COND_IGNORED)

消息：SQL线程未启动，因而UNTIL选项被忽略。

- 错误：1280 SQLSTATE: 42000 (ER_WRONG_NAME_FOR_INDEX)

消息：不正确的索引名'%s'。

- 错误：1281 SQLSTATE: 42000 (ER_WRONG_NAME_FOR_CATALOG)

消息：不正确的目录名'%s'。

- 错误：1282 SQLSTATE: HY000 (ER_WARN_QC_RESIZE)

消息：查询高速缓冲设置大小%lu时失败，新的查询高速缓冲的大小是%lu。

- 错误：1283 SQLSTATE: HY000 (ER_BAD_FT_COLUMN)

消息：列'%s'不能是FULLTEXT索引的一部分。

- 错误：1284 SQLSTATE: HY000 (ER_UNKNOWN_KEY_CACHE)

消息：未知的键高速缓冲'%s'。

- 错误：1285 SQLSTATE: HY000 (ER_WARN_HOSTNAME_WONT_WORK)

消息：MySQL是在"--skip-name-resolve"模式下启动的，必须在不使用该开关的情况下重启它，以便该授权能起作用。

- 错误：1286 SQLSTATE: 42000 (ER_UNKNOWN_STORAGE_ENGINE)

消息：未知的表引擎'%s'。

- 错误：1287 SQLSTATE: HY000 (ER_WARN_DEPRECATED_SYNTAX)

消息：'%s'已过时，请使用'%s'取而代之。

- 错误：1288 SQLSTATE: HY000 (ER_NON_UPDATABLE_TABLE)

消息：%s的目标表%s不可更新。

- 错误：1289 SQLSTATE: HY000 (ER_FEATURE_DISABLED)

消息：'%s'特性已被禁止，要想使其工作，需要用'%s'创建MySQL。

- 错误：1290 SQLSTATE: HY000 (ER_OPTION_PREVENTS_STATEMENT)

消息：MySQL正使用%s选项运行，因此不能执行该语句。

- 错误：1291 SQLSTATE: HY000 (ER_DUPLICATED_VALUE_IN_TYPE)

消息：列'%s'在%s中有重复值'%s'。

- 错误：1292 SQLSTATE: 22007 (ER_TRUNCATED_WRONG_VALUE)

消息：截短了不正确的%s值: '%s'

- 错误：1293 SQLSTATE: HY000 (ER_TOO_MUCH_AUTO_TIMESTAMP_COLS)

消息：不正确的表定义，在DEFAULT或ON UPDATE子句中，对于CURRENT_TIMESTAMP，只能有一个TIMESTAMP列。

- 错误：1294 SQLSTATE: HY000 (ER_INVALID_ON_UPDATE)

消息：对于'%s'列，ON UPDATE子句无效。

- 错误：1295 SQLSTATE: HY000 (ER_UNSUPPORTED_PS)

消息：在预处理语句协议中，尚不支持该命令。

- 错误：1296 SQLSTATE: HY000 (ER_GET_ERRMSG)

消息：从%s获得错误%d '%s'。

- 错误：1297 SQLSTATE: HY000 (ER_GET_TEMPORARY_ERRMSG)

消息：从%s获得临时错误%d '%s'。

- 错误：1298 SQLSTATE: HY000 (ER_UNKNOWN_TIME_ZONE)

消息：未知或不正确的时区: '%s'

- 错误：1299 SQLSTATE: HY000 (ER_WARN_INVALID_TIMESTAMP)

消息：在行%ld的列'%s'中存在无效的TIMESTAMP值。

- 错误：1300 SQLSTATE: HY000 (ER_INVALID_CHARACTER_STRING)

消息：无效的%s字符串: '%s'

- 错误：1301 SQLSTATE: HY000 (ER_WARN_ALLOWED_PACKET_OVERFLOWED)

消息：%s()的结果大于max_allowed_packet (%ld)，已截短

- 错误：1302 SQLSTATE: HY000 (ER_CONFLICTING_DECLARATIONS)

消息：冲突声明：'%s%s'和'%s%s'

- 错误：1303 SQLSTATE: 2F003 (ER_SP_NO_RECURSIVE_CREATE)

消息：不能从另一个存储子程序中创建%s。

- 错误：1304 SQLSTATE: 42000 (ER_SP_ALREADY_EXISTS)

消息：%s %s已存在。

- 错误：1305 SQLSTATE: 42000 (ER_SP_DOES_NOT_EXIST)

消息：%s %s不存在。

- 错误：1306 SQLSTATE: HY000 (ER_SP_DROP_FAILED)

消息：DROP %s %s失败

- 错误：1307 SQLSTATE: HY000 (ER_SP_STORE_FAILED)

消息：CREATE %s %s失败。

- 错误：1308 SQLSTATE: 42000 (ER_SP_LILABEL_MISMATCH)

消息：%s无匹配标签: %s

- 错误：1309 SQLSTATE: 42000 (ER_SP_LABEL_REDEFINE)

消息：重新定义标签%s

- 错误：1310 SQLSTATE: 42000 (ER_SP_LABEL_MISMATCH)

消息：末端标签%s无匹配项

- 错误：1311 SQLSTATE: 01000 (ER_SP_UNINIT_VAR)

消息：正在引用未初始化的变量%s。

- 错误：1312 SQLSTATE: 0A000 (ER_SP_BADSELECT)

消息：PROCEDURE %s不能在给定场景下返回结果集。

- 错误：1313 SQLSTATE: 42000 (ER_SP_BADRETURN)

消息：仅在FUNCTION中允许RETURN。

- 错误：1314 SQLSTATE: 0A000 (ER_SP_BADSTATEMENT)

消息：在存储程序中不允许%s。

- 错误：1315 SQLSTATE: 42000 (ER_UPDATE_LOG_DEPRECATED_IGNORED)

消息：更新日志已被放弃，并用二进制日志取代，SET SQL_LOG_UPDATE被忽略。

- 错误：1316 SQLSTATE: 42000 (ER_UPDATE_LOG_DEPRECATED_TRANSLATED)

消息：更新日志已被放弃，并用二进制日志取代，SET SQL_LOG_UPDATE已被截短为SET SQL_LOG_BIN。

- 错误：1317 SQLSTATE: 70100 (ER_QUERY_INTERRUPTED)

消息：查询执行被中断。

- 错误：1318 SQLSTATE: 42000 (ER_SP_WRONG_NO_OF_ARGS)

消息：对于%s %s，参量数目不正确，预期为%u，但却是%u。

- 错误：1319 SQLSTATE: 42000 (ER_SP_COND_MISMATCH)

消息：未定义的CONDITION: %s

- 错误：1320 SQLSTATE: 42000 (ER_SP_NORETURN)

消息：在FUNCTION %s中未发现RETURN。

- 错误：1321 SQLSTATE: 2F005 (ER_SP_NORETURNEND)

消息：FUNCTION %s结束时缺少RETURN。

- 错误：1322 SQLSTATE: 42000 (ER_SP_BAD_CURSOR_QUERY)

消息：光标语句必须是SELECT。

- 错误：1323 SQLSTATE: 42000 (ER_SP_BAD_CURSOR_SELECT)

消息：光标SELECT不得有INTO。

- 错误：1324 SQLSTATE: 42000 (ER_SP_CURSOR_MISMATCH)

消息：未定义的CURSOR: %s

- 错误：1325 SQLSTATE: 24000 (ER_SP_CURSOR_ALREADY_OPEN)

消息：光标已打开

- 错误：1326 SQLSTATE: 24000 (ER_SP_CURSOR_NOT_OPEN)

消息：光标未打开

- 错误：1327 SQLSTATE: 42000 (ER_SP_UNDECLARED_VAR)

消息：未声明的变量：%s

- 错误：1328 SQLSTATE: HY000 (ER_SP_WRONG_NO_OF_FETCH_ARGS)

消息：不正确的FETCH变量数目。

- 错误：1329 SQLSTATE: 02000 (ER_SP_FETCH_NO_DATA)

消息：FETCH无数据。

- 错误：1330 SQLSTATE: 42000 (ER_SP_DUP_PARAM)

消息：重复参数: %s

- 错误：1331 SQLSTATE: 42000 (ER_SP_DUP_VAR)

消息：重复变量: %s

- 错误：1332 SQLSTATE: 42000 (ER_SP_DUP_COND)

消息：重复条件: %s

- 错误：1333 SQLSTATE: 42000 (ER_SP_DUP_CURS)

消息：重复光标: %s

- 错误：1334 SQLSTATE: HY000 (ER_SP_CANT_ALTER)

消息：ALTER %s %s失败。

- 错误：1335 SQLSTATE: 0A000 (ER_SP_SUBSELECT_NYI)

消息：不支持Subselect值。

- 错误：1336 SQLSTATE: 0A000 (ER_STMT_NOT_ALLOWED_IN_SF_OR_TRG)

消息：在存储函数或触发程序中，不允许%s。

- 错误：1337 SQLSTATE: 42000 (ER_SP_VARCOND_AFTER_CURSHNDLR)

消息：光标或句柄声明后面的变量或条件声明。

- 错误：1338 SQLSTATE: 42000 (ER_SP_CURSOR_AFTER_HANDLER)

消息：句柄声明后面的光标声明。

- 错误：1339 SQLSTATE: 20000 (ER_SP_CASE_NOT_FOUND)

消息：对于CASE语句，未发现Case。

- 错误：1340 SQLSTATE: HY000 (ER_FPARSER_TOO_BIG_FILE)

消息：配置文件'%s'过大。

- 错误：1341 SQLSTATE: HY000 (ER_FPARSER_BAD_HEADER)

消息：文件'%s'中存在残缺的文件类型标题。

- 错误：1342 SQLSTATE: HY000 (ER_FPARSER_EOF_IN_COMMENT)

消息：解析'%s'时，文件意外结束。

- 错误：1343 SQLSTATE: HY000 (ER_FPARSER_ERROR_IN_PARAMETER)

消息：解析参数'%s'时出错（行：'%s'）。

- 错误：1344 SQLSTATE: HY000 (ER_FPARSER_EOF_IN_UNKNOWN_PARAMETER)

消息：跳过未知参数'%s'时，文件意外结束。

- 错误：1345 SQLSTATE: HY000 (ER_VIEW_NO_EXPLAIN)

消息：EXPLAIN/SHOW无法发出，缺少对基本表的权限。

- 错误：1346 SQLSTATE: HY000 (ER_FRM_UNKNOWN_TYPE)

消息：文件'%s'在其题头中有未知的类型'%s'。

- 错误：1347 SQLSTATE: HY000 (ER_WRONG_OBJECT)

消息：'%s.%s'不是%s。

- 错误：1348 SQLSTATE: HY000 (ER_NONUPDATEABLE_COLUMN)

消息：列'%s'不可更新。

- 错误：1349 SQLSTATE: HY000 (ER_VIEW_SELECT_DERIVED)

消息：视图的SELECT在FROM子句中包含子查询。

- 错误：1350 SQLSTATE: HY000 (ER_VIEW_SELECT_CLAUSE)

消息：视图的SELECT包含'%s'子句。

- 错误：1351 SQLSTATE: HY000 (ER_VIEW_SELECT_VARIABLE)

消息：视图的SELECT包含1个变量或参数。

- 错误：1352 SQLSTATE: HY000 (ER_VIEW_SELECT_TMPTABLE)

消息：视图的SELECT引用了临时表'%s'。

- 错误：1353 SQLSTATE: HY000 (ER_VIEW_WRONG_LIST)

消息：视图的SELECT和视图的字段列表有不同的列计数。

- 错误：1354 SQLSTATE: HY000 (ER_WARN_VIEW_MERGE)

消息：此时，不能在这里使用视图合并算法（假定未定义算法）。

- 错误：1355 SQLSTATE: HY000 (ER_WARN_VIEW_WITHOUT_KEY)

消息：正在更新的视图没有其基本表的完整键。

- 错误：1356 SQLSTATE: HY000 (ER_VIEW_INVALID)

消息：视图'%s.%s'引用了无效的表、列、或函数，或视图的定义程序／调用程序缺少使用它们的权限。

- 错误：1357 SQLSTATE: HY000 (ER_SP_NO_DROP_SP)

消息：无法从另一个存储子程序中撤销或更改%s。

- 错误：1358 SQLSTATE: HY000 (ER_SP_GOTO_IN_HNDLR)

消息：在存储子程序句柄中不允许GOTO。

- 错误：1359 SQLSTATE: HY000 (ER_TRG_ALREADY_EXISTS)

消息：触发程序已存在。

- 错误：1360 SQLSTATE: HY000 (ER_TRG_DOES_NOT_EXIST)

消息：触发程序不存在。

- 错误：1361 SQLSTATE: HY000 (ER_TRG_ON_VIEW_OR_TEMP_TABLE)

消息：触发程序的'%s'是视图或临时表。

- 错误：1362 SQLSTATE: HY000 (ER_TRG_CANT_CHANGE_ROW)

消息：在%strigger中，不允许更新%s行。

- 错误：1363 SQLSTATE: HY000 (ER_TRG_NO_SUCH_ROW_IN_TRG)

消息：在%s触发程序中没有%s行。

- 错误：1364 SQLSTATE: HY000 (ER_NO_DEFAULT_FOR_FIELD)

消息：字段'%s'没有默认值。

- 错误：1365 SQLSTATE: 22012 (ER_DIVISION_BY_ZERO)

消息：被0除。

- 错误：1366 SQLSTATE: HY000 (ER_TRUNCATED_WRONG_VALUE_FOR_FIELD)

消息：不正确的%s值，'%s'，对于行%ld 上的列'%s'。

- 错误：1367 SQLSTATE: 22007 (ER_ILLEGAL_VALUE_FOR_TYPE)

消息：解析过程中发现非法%s '%s'值。

- 错误：1368 SQLSTATE: HY000 (ER_VIEW_NONUPD_CHECK)

消息：不可更新视图'%s.%s'上的CHECK OPTION。

- 错误：1369 SQLSTATE: HY000 (ER_VIEW_CHECK_FAILED)

消息：CHECK OPTION失败，'%s.%s'

- 错误：1370 SQLSTATE: 42000 (ER_PROCACCESS_DENIED_ERROR)

消息：对于子程序'%s'，拒绝用户'%s'@'%s'使用%s命令。

- 错误：1371 SQLSTATE: HY000 (ER_RELAY_LOG_FAIL)

消息：清除旧中继日志失败，%s

- 错误：1372 SQLSTATE: HY000 (ER_PASSWD_LENGTH)

消息：密码混编应是%d位的十六进制数。

- 错误：1373 SQLSTATE: HY000 (ER_UNKNOWN_TARGET_BINLOG)

消息：在binlog索引中未发现目标日志。

- 错误：1374 SQLSTATE: HY000 (ER_IO_ERR_LOG_INDEX_READ)

消息：读取日志索引文件时出现I/O错误。

- 错误：1375 SQLSTATE: HY000 (ER_BINLOG_PURGE_PROHIBITED)

消息：服务器配置不允许binlog清除。

- 错误：1376 SQLSTATE: HY000 (ER_FSEEK_FAIL)

消息：fseek()失败。

- 错误：1377 SQLSTATE: HY000 (ER_BINLOG_PURGE_FATAL_ERR)

消息：在日志清除过程中出现致命错误。

- 错误：1378 SQLSTATE: HY000 (ER_LOG_IN_USE)

消息：可清除的日志正在使用，不能清除。

- 错误：1379 SQLSTATE: HY000 (ER_LOG_PURGE_UNKNOWN_ERR)

消息：在日志清除过程中出现未知错误。

- 错误：1380 SQLSTATE: HY000 (ER_RELAY_LOG_INIT)

消息：初始化中继日志位置失败，%s

- 错误：1381 SQLSTATE: HY000 (ER_NO_BINARY_LOGGING)

消息：未使用二进制日志功能。

- 错误：1382 SQLSTATE: HY000 (ER_RESERVED_SYNTAX)

消息：'%s'语法保留给MySQL服务器内部使用。

- 错误：1383 SQLSTATE: HY000 (ER_WSAS_FAILED)

消息：WSAStartup失败。

- 错误：1384 SQLSTATE: HY000 (ER_DIFF_GROUPS_PROC)

消息：尚不能用不同的组处理过程。

- 错误：1385 SQLSTATE: HY000 (ER_NO_GROUP_FOR_PROC)

消息：对于该过程，SELECT必须有1个组。

- 错误：1386 SQLSTATE: HY000 (ER_ORDER_WITH_PROC)

消息：不能与该过程一起使用ORDER子句。

- 错误：1387 SQLSTATE: HY000 (ER_LOGGING_PROHIBIT_CHANGING_OF)

消息：二进制日志功能和复制功能禁止更改全局服务器%s。

- 错误：1388 SQLSTATE: HY000 (ER_NO_FILE_MAPPING)

消息：无法映射文件: %s, errno: %d

- 错误：1389 SQLSTATE: HY000 (ER_WRONG_MAGIC)

消息：%s中有错

- 错误：1390 SQLSTATE: HY000 (ER_PS_MANY_PARAM)

消息：预处理语句包含过多的占位符。

- 错误：1391 SQLSTATE: HY000 (ER_KEY_PART_0)

消息：键部分'%s'的长度不能为0。

- 错误：1392 SQLSTATE: HY000 (ER_VIEW_CHECKSUM)

消息：视图文本校验和失败。

- 错误：1393 SQLSTATE: HY000 (ER_VIEW_MULTIUPDATE)

消息：无法通过联合视图'%s.%s'更改1个以上的基本表。

- 错误：1394 SQLSTATE: HY000 (ER_VIEW_NO_INSERT_FIELD_LIST)

消息：不能在没有字段列表的情况下插入联合视图'%s.%s'。

- 错误：1395 SQLSTATE: HY000 (ER_VIEW_DELETE_MERGE_VIEW)

消息：不能从联合视图'%s.%s'中删除。

- 错误：1396 SQLSTATE: HY000 (ER_CANNOT_USER)

消息：对于%s的操作%s失败。

- 错误：1397 SQLSTATE: XAE04 (ER_XAER_NOTA)

消息：XAER_NOTA: 未知XID

- 错误：1398 SQLSTATE: XAE05 (ER_XAER_INVAL)

消息：XAER_INVAL: 无效参量（或不支持的命令）

- 错误：1399 SQLSTATE: XAE07 (ER_XAER_RMFAIL)

消息：XAER_RMFAIL: 当全局事务处于%s状态时，不能执行命令。

- 错误：1400 SQLSTATE: XAE09 (ER_XAER_OUTSIDE)

消息：XAER_OUTSIDE: 某些工作是在全局事务外完成的。

- 错误：1401 SQLSTATE: XAE03 (ER_XAER_RMERR)

消息：XAER_RMERR: 在事务分支中出现致命错误，请检查数据一致性。

- 错误：1402 SQLSTATE: XA100 (ER_XA_RBROLLBACK)

消息：XA_RBROLLBACK: 回滚了事务分支。

- 错误：1403 SQLSTATE: 42000 (ER_NONEXISTING_PROC_GRANT)

消息：在子程序'%s'上没有为主机'%s'上的用户'%s'定义的这类授权。

- 错误：1404 SQLSTATE: HY000 (ER_PROC_AUTO_GRANT_FAIL)

消息：无法授予EXECUTE和ALTER ROUTINE权限。

- 错误：1405 SQLSTATE: HY000 (ER_PROC_AUTO_REVOKE_FAIL)

消息：无法撤销已放弃子程序上的所有权限。

- 错误：1406 SQLSTATE: 22001 (ER_DATA_TOO_LONG)

消息：对于行%ld上的列'%s'来说，数据过长。

- 错误：1407 SQLSTATE: 42000 (ER_SP_BAD_SQLSTATE)

消息：不良SQLSTATE: '%s'

- 错误：1408 SQLSTATE: HY000 (ER_STARTUP)

消息：%s，连接就绪；版本，'%s'；套接字，'%s'；端口，%d %s

- 错误：1409 SQLSTATE: HY000 (ER_LOAD_FROM_FIXED_SIZE_ROWS_TO_VAR)

消息：不能从具有固定大小行的文件中将值加载到变量。

- 错误：1410 SQLSTATE: 42000 (ER_CANT_CREATE_USER_WITH_GRANT)

消息：不允许用GRANT创建用户。

- 错误：1411 SQLSTATE: HY000 (ER_WRONG_VALUE_FOR_TYPE)

消息：不正确的%s值，'%s'，对于函数%s

- 错误：1412 SQLSTATE: HY000 (ER_TABLE_DEF_CHANGED)

消息：表定义已更改，请再次尝试事务。

- 错误：1413 SQLSTATE: 42000 (ER_SP_DUP_HANDLER)

消息：在相同块中声明了重复句柄。

- 错误：1414 SQLSTATE: 42000 (ER_SP_NOT_VAR_ARG)

消息：子程序%s的OUT或INOUT参量不是变量。

- 错误：1415 SQLSTATE: 0A000 (ER_SP_NO_RETSET)

消息：不允许从%s返回结果集。

- 错误：1416 SQLSTATE: 22003 (ER_CANT_CREATE_GEOMETRY_OBJECT)

消息：不能从发送给GEOMETRY字段的数据中获取几何对象。

- 错误：1417 SQLSTATE: HY000 (ER_FAILED_ROUTINE_BREAK_BINLOG)

消息：1个子程序失败，在其声明没有NO SQL或READS SQL DATA，而且二进制日志功能已启用，如果更新了非事务性表，二进制日志将丢失其变化信息。

- 错误：1418 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_ROUTINE)

消息：在该子程序的在其声明没有DETERMINISTIC、NO SQL或READS SQL DATA，而且二进制日志功能已启用（你或许打算使用不太安全的log_bin_trust_routine_creators变量）。

- 错误：1419 SQLSTATE: HY000 (ER_BINLOG_CREATE_ROUTINE_NEED_SUPER)

消息：你没有SUPER权限，而且二进制日志功能已启用（你或许打算使用不太安全的log_bin_trust_routine_creators变量）。

- 错误：1420 SQLSTATE: HY000 (ER_EXEC_STMT_WITH_OPEN_CURSOR)

消息：不能执行该预处理语句，该预处理语句有与之相关的打开光标。请复位语句并再次执行。

- 错误：1421 SQLSTATE: HY000 (ER_STMT_HAS_NO_OPEN_CURSOR)

消息：语句(%lu)没有打开的光标。

- 错误：1422 SQLSTATE: HY000 (ER_COMMIT_NOT_ALLOWED_IN_SF_OR_TRG)

消息：在存储函数或触发程序中，不允许显式或隐式提交。

- 错误：1423 SQLSTATE: HY000 (ER_NO_DEFAULT_FOR_VIEW_FIELD)

消息：视图'%s.%s'基本表的字段没有默认值。

- 错误：1424 SQLSTATE: HY000 (ER_SP_NO_RECURSION)

消息：不允许递归存储子程序。

- 错误：1425 SQLSTATE: 42000 (ER_TOO_BIG_SCALE)

消息：为列'%s'指定了过大的标度%d。最大为%d。

- 错误：1426 SQLSTATE: 42000 (ER_TOO_BIG_PRECISION)

消息：为列'%s'指定了过高的精度%d。最大为%d。

- 错误：1427 SQLSTATE: 42000 (ER_M_BIGGER_THAN_D)

消息：对于float(M,D)、double(M,D)或decimal(M,D)，M必须>= D (列'%s')。

- 错误：1428 SQLSTATE: HY000 (ER_WRONG_LOCK_OF_SYSTEM_TABLE)

消息：不能将系统'%s.%s'表的写锁定与其他表结合起来。

- 错误：1429 SQLSTATE: HY000 (ER_CONNECT_TO_FOREIGN_DATA_SOURCE)

消息：无法连接到外部数据源，数据库'%s'！

- 错误：1430 SQLSTATE: HY000 (ER_QUERY_ON_FOREIGN_DATA_SOURCE)

消息：处理作用在外部数据源上的查询时出现问题。数据源错误：'%s'

- 错误：1431 SQLSTATE: HY000 (ER_FOREIGN_DATA_SOURCE_DOESNT_EXIST)

消息：你试图引用的外部数据源不存在。数据源错误：'%s'

- 错误：1432 SQLSTATE: HY000 (ER_FOREIGN_DATA_STRING_INVALID_CANT_CREATE)

消息：无法创建联合表。数据源连接字符串'%s'格式不正确。

- 错误：1433 SQLSTATE: HY000 (ER_FOREIGN_DATA_STRING_INVALID)

消息：数据源连接字符串'%s'格式不正确。

- 错误：1434 SQLSTATE: HY000 (ER_CANT_CREATE_FEDERATED_TABLE)

消息：无法创建联合表。外部数据源错误：'%s'

- 错误：1435 SQLSTATE: HY000 (ER_TRG_IN_WRONG_SCHEMA)

消息：触发程序位于错误的方案中。

- 错误：1436 SQLSTATE: HY000 (ER_STACK_OVERRUN_NEED_MORE)

消息：线程堆栈溢出，%ld字节堆栈用了%ld字节，并需要%ld字节。请使用'mysqld -O thread_stack=#'指定更大的堆栈。

- 错误：1437 SQLSTATE: 42000 (ER_TOO_LONG_BODY)

消息：'%s'的子程序主体过长。

- 错误：1438 SQLSTATE: HY000 (ER_WARN_CANT_DROP_DEFAULT_KEYCACHE)

消息：无法撤销默认的keycache。

- 错误：1439 SQLSTATE: 42000 (ER_TOO_BIG_DISPLAYWIDTH)

消息：对于列'%s'，显示宽度超出范围(max = %d)

- 错误：1440 SQLSTATE: XAE08 (ER_XAER_DUPID)

消息：XAER_DUPID: XID已存在

- 错误：1441 SQLSTATE: 22008 (ER_DATETIME_FUNCTION_OVERFLOW)

消息：日期时间函数，%s字段溢出。

- 错误：1442 SQLSTATE: HY000 (ER_CANT_UPDATE_USED_TABLE_IN_SF_OR_TRG)

消息：由于它已被调用了该存储函数／触发程序的语句使用，不能在存储函数／触发程序中更新表'%s'。

- 错误：1443 SQLSTATE: HY000 (ER_VIEW_PREVENT_UPDATE)

消息：表'%s'的定义不允许在表'%s上执行操作%s。

- 错误：1444 SQLSTATE: HY000 (ER_PS_NO_RECURSION)

消息：预处理语句包含引用了相同语句的存储子程序调用。不允许以这类递归方式执行预处理语句。

- 错误：1445 SQLSTATE: HY000 (ER_SP_CANT_SET_AUTOCOMMIT)

消息：不允许从存储函数或触发程序设置autocommit。

- 错误：1446 SQLSTATE: HY000 (ER_NO_VIEW_USER)

消息：视图定义人不完全合格。

- 错误：1447 SQLSTATE: HY000 (ER_VIEW_FRM_NO_USER)

消息：视图%s.%s没有定义人信息（旧的表格式）。当前用户将被当作定义人。请重新创建视图!

- 错误：1448 SQLSTATE: HY000 (ER_VIEW_OTHER_USER)

消息：需要SUPER权限才能创建具有%s@%s定义器的视图。

- 错误：1449 SQLSTATE: HY000 (ER_NO_SUCH_USER)

消息：没有注册的%s@%s。

- 错误：1450 SQLSTATE: HY000 (ER_FORBID_SCHEMA_CHANGE)

消息：不允许将方案从'%s'变为'%s'。

- 错误：1451 SQLSTATE: 23000 (ER_ROW_IS_REFERENCED_2)

消息：不能删除或更新父行，外键约束失败(%s)。

- 错误：1452 SQLSTATE: 23000 (ER_NO_REFERENCED_ROW_2)

消息：不能添加或更新子行，外键约束失败(%s)。

- 错误：1453 SQLSTATE: 42000 (ER_SP_BAD_VAR_SHADOW)

消息：必须用`...`引用变量，或重新命名变量。

- 错误：1454 SQLSTATE: HY000 (ER_PARTITION_REQUIRES_VALUES_ERROR)

消息：对于每个分区，%s PARTITIONING需要VALUES %s的定义。

- 错误：1455 SQLSTATE: HY000 (ER_PARTITION_WRONG_VALUES_ERROR)

消息：在分区定义中，只有%s PARTITIONING能使用VALUES %s。

- 错误：1456 SQLSTATE: HY000 (ER_PARTITION_MAXVALUE_ERROR)

消息：MAXVALUE只能在最后1个分区定义中使用。

- 错误：1457 SQLSTATE: HY000 (ER_PARTITION_SUBPARTITION_ERROR)

消息：子分区只能是哈希分区，并按键分区。

- 错误：1458 SQLSTATE: HY000 (ER_PARTITION_WRONG_NO_PART_ERROR)

消息：定义了错误的分区数，与前面的设置不匹配。

- 错误：1459 SQLSTATE: HY000 (ER_PARTITION_WRONG_NO_SUBPART_ERROR)

消息：定义了错误的子分区数，与前面的设置不匹配。

- 错误：1460 SQLSTATE: HY000 (ER_CONST_EXPR_IN_PARTITION_FUNC_ERROR)

消息：在分区（子分区）函数中不允许使用常量／随机表达式。

- 错误：1461 SQLSTATE: HY000 (ER_NO_CONST_EXPR_IN_RANGE_OR_LIST_ERROR)

消息：RANGE/LIST VALUES中的表达式必须是常量。

- 错误：1462 SQLSTATE: HY000 (ER_FIELD_NOT_FOUND_PART_ERROR)

消息：在表中未发现分区函数字段列表中的字段。

- 错误：1463 SQLSTATE: HY000 (ER_LIST_OF_FIELDS_ONLY_IN_HASH_ERROR)

消息：仅在KEY分区中允许使用字段列表。

- 错误：1464 SQLSTATE: HY000 (ER_INCONSISTENT_PARTITION_INFO_ERROR)

消息：frm文件中的分区信息与能够写入到frm文件中的不一致。

- 错误：1465 SQLSTATE: HY000 (ER_PARTITION_FUNC_NOT_ALLOWED_ERROR)

消息：%s函数返回了错误类型。

- 错误：1466 SQLSTATE: HY000 (ER_PARTITIONS_MUST_BE_DEFINED_ERROR)

消息：对于%s分区，必须定义每个分区。

- 错误：1467 SQLSTATE: HY000 (ER_RANGE_NOT_INCREASING_ERROR)

消息：对于各分区，VALUES LESS THAN值必须严格增大。

- 错误：1468 SQLSTATE: HY000 (ER_INCONSISTENT_TYPE_OF_FUNCTIONS_ERROR)

消息：VALUES值必须与分区函数具有相同的类型。

- 错误：1469 SQLSTATE: HY000 (ER_MULTIPLE_DEF_CONST_IN_LIST_PART_ERROR)

消息：Multiple definition of same constant in list partitioning

- 错误：1470 SQLSTATE: HY000 (ER_PARTITION_ENTRY_ERROR)

消息：在查询中，不能独立使用分区功能。

- 错误：1471 SQLSTATE: HY000 (ER_MIX_HANDLER_ERROR)

消息：在该MySQL版本中，不允许分区中的句柄组合。

- 错误：1472 SQLSTATE: HY000 (ER_PARTITION_NOT_DEFINED_ERROR)

消息：对于分区引擎，有必要定义所有的%s。

- 错误：1473 SQLSTATE: HY000 (ER_TOO_MANY_PARTITIONS_ERROR)

消息：定义了过多分区。

- 错误：1474 SQLSTATE: HY000 (ER_SUBPARTITION_ERROR)

消息：对于子分区，仅能将RANGE/LIST分区与HASH/KEY分区混合起来。

- 错误：1475 SQLSTATE: HY000 (ER_CANT_CREATE_HANDLER_FILE)

消息：无法创建特定的句柄文件。

- 错误：1476 SQLSTATE: HY000 (ER_BLOB_FIELD_IN_PART_FUNC_ERROR)

消息：在分区函数中，不允许使用BLOB字段。

- 错误：1477 SQLSTATE: HY000 (ER_CHAR_SET_IN_PART_FIELD_ERROR)

消息：如果为分区函数选择了二进制校对，才允许使用VARCHAR。

- 错误：1478 SQLSTATE: HY000 (ER_UNIQUE_KEY_NEED_ALL_FIELDS_IN_PF)

消息：在分区函数中，%s需要包含所有文件。

- 错误：1479 SQLSTATE: HY000 (ER_NO_PARTS_ERROR)

消息：%s的数目= 0不是允许的值。

- 错误：1480 SQLSTATE: HY000 (ER_PARTITION_MGMT_ON_NONPARTITIONED)

消息：无法在非分区表上进行分区管理。

- 错误：1481 SQLSTATE: HY000 (ER_DROP_PARTITION_NON_EXISTENT)

消息：分区列表中的错误出现变化。

- 错误：1482 SQLSTATE: HY000 (ER_DROP_LAST_PARTITION)

消息：不能删除所有分区，请使用DROP TABLE取而代之。

- 错误：1483 SQLSTATE: HY000 (ER_COALESCE_ONLY_ON_HASH_PARTITION)

消息：COALESCE PARTITION仅能在HASH/KEY分区上使用。

- 错误：1484 SQLSTATE: HY000 (ER_ONLY_ON_RANGE_LIST_PARTITION)

消息：%s PARTITION仅能在RANGE/LIST分区上使用。

- 错误：1485 SQLSTATE: HY000 (ER_ADD_PARTITION_SUBPART_ERROR)

消息：试图用错误的子分区数增加分区。

- 错误：1486 SQLSTATE: HY000 (ER_ADD_PARTITION_NO_NEW_PARTITION)

消息：必须至少添加1个分区。

- 错误：1487 SQLSTATE: HY000 (ER_COALESCE_PARTITION_NO_PARTITION)

消息：必须至少合并1个分区。

- 错误：1488 SQLSTATE: HY000 (ER_REORG_PARTITION_NOT_EXIST)

消息：重组的分区数超过了已有的分区数。

- 错误：1489 SQLSTATE: HY000 (ER_SAME_NAME_PARTITION)

消息：在表中，所有分区必须有唯一的名称。

- 错误：1490 SQLSTATE: HY000 (ER_CONSECUTIVE_REORG_PARTITIONS)

消息：重组分区集合时，它们必须连续。

- 错误：1491 SQLSTATE: HY000 (ER_REORG_OUTSIDE_RANGE)

消息：新分区的范围超过了已重组分区的范围。

- 错误：1492 SQLSTATE: HY000 (ER_DROP_PARTITION_FAILURE)

消息：在该版本的句柄中，不支持撤销分区。

- 错误：1493 SQLSTATE: HY000 (ER_DROP_PARTITION_WHEN_FK_DEFINED)

消息：在表上定义了外键约束时，不能舍弃分区。

- 错误：1494 SQLSTATE: HY000 (ER_PLUGIN_IS_NOT_LOADED)

消息：未加载插件'%s'

B.2. 客户端错误代码和消息

客户端错误信息来自下述源文件：

- 圆括号中的错误值和符号与include/errmsg.h MySQL源文件中的定义对应。
- 消息值与libmysql/errmsg.c文件中列出的错误消息对应。%d和%s分别代表数值和字符串，显示时，它们将被消息值取代。

由于更新很频繁，这些文件中可能包含这里未列出的额外错误消息。

- 错误：2000 (CR_UNKNOWN_ERROR)

消息：未知MySQL错误。

- 错误：2001 (CR_SOCKET_CREATE_ERROR)

消息：不能创建UNIX套接字(%d)

- 错误：2002 (CR_CONNECTION_ERROR)

消息：不能通过套接字'%s' (%d)连接到本地MySQL服务器。

- 错误：2003 (CR_CONN_HOST_ERROR)

消息：不能连接到'%s' (%d)上的MySQL服务器。

- 错误：2004 (CR_IPSOCK_ERROR)

消息：不能创建TCP/IP套接字(%d)

- 错误：2005 (CR_UNKNOWN_HOST)

消息：未知的MySQL服务器主机'%s' (%d)

- 错误：2006 (CR_SERVER_GONE_ERROR)

消息：MySQL服务器不可用。

- 错误：2007 (CR_VERSION_ERROR)

消息：协议不匹配，服务器版本= %d，客户端版本= %d

- 错误：2008 (CR_OUT_OF_MEMORY)

消息：MySQL客户端内存溢出。

- 错误：2009 (CR_WRONG_HOST_INFO)

消息：错误的主机信息

- 错误：2010 (CR_LOCALHOST_CONNECTION)

消息：通过UNIX套接字连接的本地主机。

- 错误：2011 (CR_TCP_CONNECTION)

消息：%s，通过TCP/IP

- 错误：2012 (CR_SERVER_HANDSHAKE_ERR)

消息：服务器握手过程中出错。

- 错误：2013 (CR_SERVER_LOST)

消息：查询过程中丢失了与MySQL服务器的连接。

- 错误：2014 (CR_COMMANDS_OUT_OF_SYNC)

消息：命令不同步，你现在不能运行该命令。

- 错误：2015 (CR_NAMEDPIPE_CONNECTION)

消息：命名管道，%s

- 错误：2016 (CR_NAMEDPIPEWAIT_ERROR)

消息：无法等待命名管道，主机，%s；管道，%s (%lu)

- 错误：2017 (CR_NAMEDPIPEOPEN_ERROR)

消息：无法打开命名管道，主机，%s；管道，%s (%lu)

- 错误：2018 (CR_NAMEDPIPESETSTATE_ERROR)

消息：无法设置命名管道的状态，主机，%s；管道，%s (%lu)

- 错误：2019 (CR_CANT_READ_CHARSET)

消息：无法初始化字符集%s (路径：%s)

- 错误：2020 (CR_NET_PACKET_TOO_LARGE)

消息：获得的信息包大于'max_allowed_packet'字节。

- 错误：2021 (CR_EMBEDDED_CONNECTION)

消息：嵌入式服务器。

- 错误：2022 (CR_PROBE_SLAVE_STATUS)

消息：SHOW SLAVE STATUS出错：

- 错误：2023 (CR_PROBE_SLAVE_HOSTS)

消息：SHOW SLAVE HOSTS出错：

- 错误：2024 (CR_PROBE_SLAVE_CONNECT)

消息：连接到从服务器时出错：

- 错误：2025 (CR_PROBE_MASTER_CONNECT)

消息：连接到主服务器时出错：

- 错误：2026 (CR_SSL_CONNECTION_ERROR)

消息：SSL连接错误

- 错误：2027 (CR_MALFORMED_PACKET)

消息：残缺信息包。

- 错误：2028 (CR_WRONG_LICENSE)

消息：该客户端库仅授权给具有'%s'许可的MySQL服务器使用。

- 错误：2029 (CR_NULL_POINTER)

消息：空指针的无效使用。

- 错误：2030 (CR_NO_PREPARE_STMT)

消息：语句未准备好。

- 错误：2031 (CR_PARAMS_NOT_BOUND)

消息：没有为预处理语句中的参数提供数据。

- 错误：2032 (CR_DATA_TRUNCATED)

消息：数据截短。

- 错误：2033 (CR_NO_PARAMETERS_EXISTS)

消息：语句中不存在任何参数。

- 错误：2034 (CR_INVALID_PARAMETER_NO)

消息：无效的参数编号。

- 错误：2035 (CR_INVALID_BUFFER_USE)

消息：不能为非字符串／非二进制数据类型发送长数据（参数：%d）。

- 错误：2036 (CR_UNSUPPORTED_PARAM_TYPE)

消息：正使用不支持的缓冲区类型， %d （参数：%d）

- 错误：2037 (CR_SHARED_MEMORY_CONNECTION)

消息：共享内存，%s

- 错误：2038 (CR_SHARED_MEMORY_CONNECT_REQUEST_ERROR)

消息：不能打开共享内存，客户端不能创建请求事件(%lu)

- 错误：2039 (CR_SHARED_MEMORY_CONNECT_ANSWER_ERROR)

消息：不能打开共享内存，未收到服务器的应答事件(%lu)

- 错误：2040 (CR_SHARED_MEMORY_CONNECT_FILE_MAP_ERROR)

消息：不能打开共享内存，服务器不能分配文件映射(%lu)

- 错误：2041 (CR_SHARED_MEMORY_CONNECT_MAP_ERROR)

消息：不能打开共享内存，服务器不能获得文件映射的指针(%lu)

- 错误：2042 (CR_SHARED_MEMORY_FILE_MAP_ERROR)

消息：不能打开共享内存，客户端不能分配文件映射(%lu)

- 错误：2043 (CR_SHARED_MEMORY_MAP_ERROR)

消息：不能打开共享内存，客户端不能获得文件映射的指针(%lu)

- 错误：2044 (CR_SHARED_MEMORY_EVENT_ERROR)

消息：不能打开共享内存，客户端不能创建%s事件(%lu)

- 错误：2045 (CR_SHARED_MEMORY_CONNECT_ABANDONED_ERROR)

消息：不能打开共享内存，无来自服务器的应答 (%lu)

- 错误：2046 (CR_SHARED_MEMORY_CONNECT_SET_ERROR)

消息：不能打开共享内存，不能将请求事件发送到服务器(%lu)

- 错误：2047 (CR_CONN_UNKNOW_PROTOCOL)

消息：错误或未知协议

- 错误：2048 (CR_INVALID_CONN_HANDLE)

消息：无效的连接句柄

- 错误：2049 (CR_SECURE_AUTH)

消息：拒绝使用旧鉴定协议（早于4.1.1）的连接（开启了客户端'secure_auth'选项）。

- 错误：2050 (CR_FETCH_CANCELED)

消息：行检索被mysql_stmt_close()调用取消。

- 错误：2051 (CR_NO_DATA)

消息：在未事先获取行的情况下试图读取列。

- 错误：2052 (CR_NO_STMT_METADATA)

消息：预处理语句不含元数据。

- 错误：2053 (CR_NO_RESULT_SET)

消息：在没有与语句相关的结果集时试图读取行。

- 错误：2054 (CR_NOT_IMPLEMENTED)

消息：该特性尚未实施。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

附录C：感谢

目录

C.1. MySQL AB处的开发人
C.2. MySQL贡献人
C.3. 资料员和译员
C.4. MySQL使用和包含的库
C.5. 支持MySQL的软件包
C.6. 用于创建MySQL的工具
C.7. MySQL支持人员

在本附录中，列出了帮助MySQL达到其现状的开发人员、贡献人和支持人员。

C.1. MySQL AB处的开发人

他们是MySQL AB公司雇佣或曾经雇佣的、负责MySQL数据库软件的开发人员，大概按照他们与我们一起工作的时间顺序排列。在每位开发人员后面，列出了其负责的一些任务，或取得的部分成就。所有的开发人员均参与支持。

- Michael (Monty) Widenius
  - 领导MySQL服务器的开发人员和主要作者（**mysqld**）。
  - 用于字符串库的新函数。
  - 大多数mysys库。
  - ISAM和MyISAM库（B-tree索引文件处理程序，以及索引压缩和不同的记录格式）。
  - HEAP库。内存表系统，以及优异的全动态混编处理功能。1981年其使用，1984年左右发布。
  - **replace**程序（不妨一看，它很酷！）
  - Connector/ODBC (MyODBC)，用于Windows的ODBC驱动程序。
  - 更正了MIT-pthreads中存在的问题，使之能与MySQL服务器一起工作。以及Unireg，具备众多功能的基于光标的应用工具。
  - mSQL工具的移植，如msqlperl、DBD/DBI和DB2mysql等。
  - 大多数crash-me，以及MySQL基准的建立。
- David Axmark
  - “参考手册”的最初主要作者，包括对**texi2html**的增强。
  - 从手动更新到自动网站更新。
  - 最初的Autoconf、Automake和Libtool支持。
  - 许可事宜.

- o 参与了所有文本文件的编制（目前仅剩下README。其余仅出现在手册中）。
- o 众多对新特性的测试。
- o 我方内部的免费软件法律专家。
- o 邮件列表维护人（从没时间恰当维护它）。
- o 我方最初的移植代码（到目前为止超过10年）。目前仅剩下mysys的一部分。
- o 当他刚使新特性起作用时，是Monty半夜呼叫的重要人物。
- o 首席"开放源码人员"（MySQL社区关系）
- · Jani Tolonen
- o **mysqlimport**
- o 对命令行的众多扩展。
- o PROCEDURE ANALYSE()
- · Sinisa Milivojevic（目前提供支持服务）
- o 客户端／服务器协议中的压缩（用zlib）。
- o 完美的混编，针对词汇分析器阶段。
- o 多行INSERT。
- o **mysqldump** –e选项。
- o LOAD DATA LOCAL INFILE
- o SQL_CALC_FOUND_ROWS SELECT选项。
- o --max-user-connections=...选项。
- o net_read and net_write_timeout
- o GRANT/REVOKE和SHOW GRANTS FOR
- o 新的客户端／服务器协议，用于4.0
- o 4.0中的UNION。
- o 多表DELETE/UPDATE。
- o 4.1中的导出表。
- o 用户资源管理。
- o MySQL++ C++ API和MySQLGUI客户端的最初开发人员
- · Tonu Samuel（过去的开发人员）
- o VIO接口（加密客户端／服务器协议的基础）。

- MySQL Filesystem（将MySQL数据库作为文件和目录使用的一种方式）
- CASE表达式
- MD5()和COALESCE()函数。
- 对MyISAM表的RAID支持。
- Sasha Pachev（过去的开发人员）
- 最初的复制史事（直至4.0版）。
- SHOW CREATE TABLE.
- mysql-bench
- Matt Wagner
- MySQL测试套件。
- Webmaster（直到2002）。
- Miguel Solorzano（目前提供支持服务）
- Win32开发和版本创建。
- Windows NT服务器代码。
- WinMySQLAdmin
- Timothy Smith（目前提供支持服务）
- 动态字符集支持。
- 配置、RPM、以及创建系统的其他部分。
- 嵌入式服务器libmysqld的最初开发人员。
- Sergei Golubchik
- 全文本搜索。
- 为MERGE库增加了多个 键。
- 精度数学
- Jeremy Cole（过去的开发人员）
- 本精细手册的较对和编辑。
- ALTER TABLE ... ORDER BY ....
- UPDATE ... ORDER BY ....
- DELETE ... ORDER BY ....
- Indrek Siitan

- 我方Web接口的设计／编程。
- 我方时事通报管理系统的作者。
- Jorge del Conde（目前提供支持服务）
- **MySQLCC**（**MySQL**控制中心)
- Win32开发
- 网站门户的最初实施。
- Venu Anuganti（过去的开发人员）
- MyODBC 3.51
- 用于4.1的新客户端／服务器协议（对于预处理语句）。
- Arjen Lentz（目前负责社区事宜）
- MySQL参考手册的维护人员。
- 准备了该手册的O'Reilly印刷版。
- Alexander (Bar) Barkov、Alexey (Holyfoot) Botchkov、以及Ramil Kalimullin
- 针对4.1的空间数据（GIS）和R-Trees实施。
- 用于4.1的Unicode和字符集，以及文档。
- Oleksandr (Sanja) Byelkin
- 4.0中的查询高速缓冲。
- 子查询的实施（4.1）。
- 视图和导出表的实施（5.0）。
- Aleksey (Walrus) Kishkin和Alexey (Ranger) Stroganov
- 基准设计和分析。
- MySQL测试套件的维护。
- Zak Greant（以前的雇员）
- 开放源码宣传人，MySQL社区关系。
- Carsten Pedersen
- MySQL认证计划。
- Lenz Grimmer
- 生产（创建和发布）工程。

- Peter Zaitsev
  - SHA1()、AES_ENCRYPT()和AES_DECRYPT()函数。
  - 调试并整理了各种特性。
- Alexander (Salle) Keremidarski
  - 支持。
  - 调试。
- Per-Erik Martin
  - 存储程序的主要开发人员（5.0）。
- Jim Winstead
  - 以前的首席Web开发人员。
  - 改进了服务器，更正了缺陷。
- Mark Matthews
  - Connector/J driver (Java).
- Peter Gulutzan
  - SQL标准兼容。
  - 现有MySQL代码/算法的文档记录。
  - 字符集文档。
- Guilhem Bichot
  - 复制，从MySQL 4.0板开始。
  - 更正了DECIMAL指数的处理功能。
  - Author of mysql_tableinfo.
  - 备份（5.1中）。
- Antony T. Curtis
  - 将MySQL数据库软件移植到OS/2。
- Mikael Ronstrom
  - 2000年前，大量关于NDB簇的初始工作。在那时完成了约一半的代码基数。事物协议、节点恢复、系统重启、以及API功能的重启代码和部分内容。
  - 1994～2004，NDB簇的首席设计师、开发人员和调试人员。
  - 众多优化。

- Jonas Oreland
  - 在线备份
  - MySQL簇的自动测试环境
  - 用于NDB簇的移植库
  - 众多其他事项
- Pekka Nouisiainen
  - MySQL簇的有序索引实施
  - MySQL簇中的BLOB支持
  - MySQL簇中的字符集支持。
- Martin Skold
  - MySQL簇的为一索引实施
  - 将NDB簇集成到MySQL
- Magnus Svensson
  - MySQL簇的测试架构
  - 将NDB簇集成到MySQL
- Tomas Ulin
  - 大量关于配置变更方面的工作，简化了MySQL簇的安装和使用。
- Konstantin Osipov
  - 预处理语句。
  - 光标
- Dmitri Lenev
  - 时区支持。
  - 触发程序（5.0中）。

C.2. MySQL贡献人

尽管MySQL AB公司拥有MySQL服务器和MySQL手册的所有版权，对那些为MySQL的发布作出这样或那样贡献的人员，我们深表感谢和赞赏。下面按一定的随机顺序列出了这些贡献者：

- Gianmassimo Vigazzola <qwerg@mbox.vol.it> or <qwerg@tin.it>

对Win32/NT的最初移植

- Per Eric Olsson

或多或少的有建设性意义的批评，以及对动态记录格式实际测试。

- Irena Pancirov <irena@mail.yacc.it>

与Borland 编译器一起使用的Win32端口。mysqlshutdown.exe和mysqlwatch.exe

- David J. Hughes

为了创建共享软件SQL数据库，在MySQL AB公司的前身TcX，我们从mSQL着手，但发现它不能满足的目的要求，因此，我们为自己的应用程序构造程序Unireg编写了SQL接口。**Mysqladmin**和**mysql**客户端是在很大程度上受其mSQL对等物影响的程序。我们进行了大量努力，努力使MySQL语法成为mSQL的超集。我们从mSQL借鉴了很多API的概念，以便能更容易地将免费的mSQL程序移植到MySQL API。MySQL软件不含任何来自mSQL的代码。分发版中的两个文件（client/insert_test.c和client/select_test.c）分别基于mSQL发布版中对应的（非版权保护）文件，但作为示例对其进行了更改，给出了从mSQL代码转换为MySQL服务器所需的更改（mSQL的版权归David J. Hughes所有）。

- Patrick Lynch

帮助我们获得http://www.mysql.com/。

- Fred Lindberg

设置了qmail以处理MySQL邮件列表，并在管理MySQL邮件列表方面为我们提供了无与伦比的帮助。

- Igor Romanenko <igor@frog.kiev.ua>

Mysqldump（以前为msqldump，但被Monty移植并予以增强）。

- Yuri Dario

保持并扩展了MySQL OS/2端口。

- Tim Bunce

mysqlhotcopy的作者。

- Zarko Mocnik <zarko.mocnik@dem.si>

为斯洛文尼亚语进行了分类。

- "TAMITO" <tommy@valley.ne.jp>

_MB字符集宏以及ujis和sjis字符集。

- Joshua Chamas <joshua@chamas.com>

为并发插入奠定了基础，扩展了日期语法，在NT上进行了调试，并在MySQL邮件列表上回答问题。

- Yves Carlier <Yves.Carlier@rug.ac.be>

Mysqlaccess，显示用户访问权限的程序。

- Rhys Jones <rhys@wales.com> (And GWE Technologies Limited)

提供了早期的JDBC驱动程序之一。

- Dr Xiaokun Kelvin ZHU <X.Zhu@brad.ac.uk>

进一步发展了早期的JDBC驱动程序之一，并开发了其他与MySQL有关的Java工具。

- James Cooper <pixel@organic.com>

在他的站点建立了可搜索的邮件列表档案。

- Rick Mehalick <Rick_Mehalick@i-o.com>

贡献了xmysql，一种用于MySQL服务器的图形化X客户端。

- Doug Sisk <sisk@wix.com>

为Red Hat Linux提供了MySQL的RPM软件包。

- Diemand Alexander V. <axeld@vial.ethz.ch>

为Red Hat Linux-Alpha提供了MySQL的RPM软件包。

- Antoni Pamies Olive <toni@readysoft.es>

为Intel和SPARC提供了众多MySQL客户端的RPM版本。

- Jay Bloodworth <jay@pathways.sde.state.sc.us>

为MySQL 3.21提供了RPM版本。

- David Sacerdote <davids@secnet.com>

贡献了关于DNS主机名安全检查的多项概念。

- Wei-Jou Chen <jou@nematic.ieo.nctu.edu.tw>

对中文字符（BIG5）的一些支持。

- Wei He <hewei@mail.ied.ac.cn>

为中文字符集（GBK）提供了众多功能。

- Jan Pazdziora <adelton@fi.muni.cz>

捷克语分类顺序。

- Zeev Suraski <bourbon@netvision.net.il>

FROM_UNIXTIME()时间格式化，ENCRYPT()函数，以及**bison**顾问。主动邮件列表成员。

- Luuk de Boer <luuk@wxs.nl>

将基准套件移植到（并扩展了它）DBI/DBD。对于crash-me和运行基准很有帮助。一些新的日期函数。**mysql_setpermission**脚本。

- Alexis Mikhailov <root@medinf.chuvashia.su>

用户定义函数（UDFs），CREATE FUNCTION和DROP FUNCTION。

- Andreas F. Bobak <bobak@relog.ch>

对用户定义函数的AGGREGATE扩展。

- Ross Wakelin <R.Wakelin@march.co.uk>

帮助创建了用于MySQL-Win32的InstallShield。

- Jethro Wright III <jetman@li.net>

libmysql.dll库。

- James Pereria <jpereira@iafrica.com>

Mysqlmanager，1种用于管理MySQL服务器的Win32 GUI工具。

- Curt Sampson <cjs@portal.ca>

将MIT-pthreads移植到NetBSD/Alpha和NetBSD 1.3/i386。

- Martin Ramsch <m.ramsch@computer.org>

MySQL教程中的示例。

- Steve Harvey

使得创建**mysqlaccess**更安全。

- Konark IA-64 Centre of Persistent Systems Private Limited

http://www.pspl.co.in/konark/。为MySQL服务器的Win64端口提供了帮助。

- Albert Chin-A-Young.

为Tru64配置了更新项，大文件支持和更好的TCP包装支持。

- John Birrell

模拟了用于OS/2的pthread_mutex()。

- Benjamin Pflugmann

扩展了MERGE表以处理INSERTS。MySQL邮件列表上的积极成员。

- Jocelyn Fournier

准确判定并通报了无数缺陷（尤其是在MySQL 4.1子查询代码中）。

- Marc Liyanage

维护Mac OS X软件包，并就如何创建Mac OS X PKG提供了无价的反馈意见。

- Robert Rutherford

就QNX端口提供了无价的信息和反馈。

- NDB簇以前的开发人员

以各种途径包含众多人员，访问学生、做课题的学生、雇员。总数超过100，人数太多，不可能在此意义提及。著名人员包括Ataullah Dabaghi，在1999年之前，贡献了约1/3的代码基数。特别感谢AXE系统的开发人员，该系统为具备块、信号和崩溃跟踪功能的NDB簇提供了结构基础。此外，还要感谢从1992年到现在、对这些理念坚信不疑并将其预算用于该项开发的人员。

其他的贡献人、缺陷发现人和测试人：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。

此外，还有很多由邮件列表上的人员贡献的缺陷通报／补丁。

十分感谢那些帮助我们在MySQL邮件列表上回答问题的人员：

- Daniel Koch <dkoch@amcity.com>

Irix设置。

- Luuk de Boer <luuk@wxs.nl>

基准问题。

- Tim Sailer <tps@users.buoy.com>

DBD::mysql问题。

- Boyd Lynn Gerber <gerberb@zenez.com>

与SCO有关的问题。

- Richard Mehalick <RM186061@shellus.com>

与xmysql有关的问题和基本的安装问题。

- Zeev Suraski <bourbon@netvision.net.il>

Apache模块配置问题（登录和鉴定），与PHP有关的问题，与SQL语法有关的问题，以及其他一般问题。

- Francesc Guasch <frankie@citel.upc.es>

一般问题。

- Jonathan J Smith <jsmith@wtp.net>

与Linux操作系统有关的问题，SQL语法，以及需要某些工作的其他事宜。

- David Sklar <sklar@student.net>

从PHP和Perl使用MySQL。

- Alistair MacDonald <A.MacDonald@uel.ac.uk>

尚未指定，但很灵活，并能处理与Linux以及或许HP-UX有关的问题，帮助用户使用**mysqlbug**。

- John Lyon <jlyon@imag.net>

关于在Linux系统上安装MySQL的问题，使用.rpm文件，或从源码进行编译。

- Lorvid Ltd. <lorvid@WOLFENET.com>

简单的记账/许可/支持/版权事宜。

- Patrick Sherrill <patrick@coconet.com>

与ODBC和VisualC++接口有关的问题。

- Randy Harmon <rjharmon@uptimecomputers.com>

与DBD、Linux、以及一些SQL语法有关的问题。

C.3. 资料员和译员

下述人员帮助我们编写了MySQL文档，并翻译了文档以及MySQL中的错误信息。

- Paul DuBois

正在为本手册的正确性和可理解性提供帮助。包括改写了Monty和David的英文尝试，使其成为其他人理解的英语。

- Kim Aldale

帮助改写了Monty和David的英文早期尝试，将其改写为英语。

- Michael J. Miller Jr. <mke@terrapin.turbolift.com>

第1版MySQL手册。更正了关于常见问题的很多拼写/语言问题（很久以前被纳入MySQL手册）。

- Yan Cailin

2000年初，将MySQL参考手册翻译为简体中文的译员，Big5和HK编码（http://mysql.hitstar.com/）版本以其翻译本为基础。在linuxdb.yeah.net上的个人主页。

- Jay Flaherty <fty@mediapulse.com>

本手册中Perl DBI/DBD一节的很大部分。

- Paul Southworth <pauls@etext.org>, Ray Loyzaga <yar@cs.su.oz.au>

校对参考手册。

- Therrien Gilbert <gilbert@ican.net>, Jean-Marc Pouyot <jmp@scalaire.fr>

法语错误消息。

- Petr Snajdr, <snajdr@pvt.net>

捷克语错误消息。

- Jaroslaw Lewandowski <jotel@itnet.com.pl>

波兰语错误消息。

- Miguel Angel Fernandez Roiz

西班牙语错误消息。

- Roy-Magne Mo <rmo@www.hivolda.no>

挪威语错误消息，以及MySQL 3.21.xx的测试。

- Timur I. Bakeyev <root@timur.tatarstan.ru>

俄语错误消息。

- <brenno@dewinter.com> & Filippo Grassilli <phil@hyppo.com>

意大利语错误消息。

- Dirk Munzinger <dirk@trinity.saar.de>

德语错误消息。

- Billik Stefan <billik@sun.uniag.sk>

斯洛伐克语错误消息。

- Stefan Saroiu <tzoompy@cs.washington.edu>

罗马尼亚语错误消息。

- Peter Feher

匈牙利语错误消息。

- Roberto M. Serqueira

葡萄牙语错误消息。

- Carsten H. Pedersen

丹麦语错误消息。

- Arjen G. Lentz

荷兰语错误消息，完成了早期的部分翻译（也负责一致性和拼写问题）。

C.4. MySQL使用和包含的库

下面列出了在MySQL服务器源码中包含的库的创建人，源码简化了MySQL的编译和安装。对于创建这些库的所有个人我们深表谢意，它使得我们的生活变得简单。

- Fred Fish

感谢他提供的优异C调试和跟踪库。Monty对库进行了大量的小改进（速度和额外选项）。

- Richard A. O'Keefe

感谢他提供的公共域字符串库。

- Henry Spencer

感谢他的regex库，用在WHERE列REGEXP regexp中。

- Chris Provenzano

便携用户级别的pthreads。由于版权：本产品包含由加利福尼亚大学伯克利分校的Chris Provenzano以及多名贡献人开发的软件。我们目前使用的是由Monty修补的1_60_beta6版（请参见mit-pthreads/Changes-mysql）。

- Jean-loup Gailly and Mark Adler

zlib库（用于Windows平台上的MySQL）。

- Bjorn Benson

感谢他的safe_malloc（内存检验器）软件包，当你用"--debug"配置MySQL时将用到它。

- Free Software Foundation

readline库（由**mysql**命令行客户端使用）。

- The NetBSD foundation

libedit软件包（可被**mysql**命令行客户端使用）。

C.5. 支持MySQL的软件包

下面列出了一些最重要API／软件包／应用程序的创建人和维护人，很多人员与MySQL一起使用了它们。

我们无法在此列出所有的软件包，这是因为，如果这样，该列表将很难维护。关于其他软件包，请访问软件门户站点http://solutions.mysql.com/software/。

- Tim Bunce, Alligator Descartes

DBD (Perl)接口。

- Andreas Koenig <a.koenig@mind.de>

用于MySQL服务器的Perl接口。

- Jochen Wiedmann <wiedmann@neckar-alb.de>

维护Perl DBD::mysql模块。

- Eugene Chan <eugene@acenet.com.sg>

移植针对MySQL服务器的PHP。

- Georg Richter

MySQL 4.1测试和缺陷追踪。新的PHP 5.0 mysqli扩展（API），用于MySQL 4.1和更高版本。

- Giovanni Maruzzelli <maruzz@matrice.it>

移植iODBC (Unix ODBC)。

· Xavier Leroy <Xavier.Leroy@inria.fr>

LinuxThreads的作者（供Linux平台上的MySQL服务器使用）。

C.6. 用于创建MySQL的工具

下面列出了一些我们用于创建MySQL的工具。我们借此表达对那些创建了它们的人员的谢意，如果没有它们，我们将无法使MySQL达到目前的状况。

· 自由软件基金会

从该基金会，我们获得了优异的编译器（**gcc**），良好的调试器（**gdb**）和libc库（我们从中借用了strto.c，从而使得部分代码能工作在Linux环境下）。

· 自由软件基金会和Xemacs开发团队

MySQL AB公司几乎人员使用的强大的编辑器/环境。

· Julian Seward

valgrind的作者，valgrind是一种优异的内存检验器工具，它帮助我们找到了MySQL中很多用其他方法难以发现的缺陷。

· Dorothea Lütkehaus和Andreas Zeller

DDD（数据显示调试器），1种优秀的gdb图形前端。

C.7. MySQL支持人员

尽管MySQL AB公司拥有MySQL服务器和MySQL手册的所有版权，对那些从财政上帮助我们开发MySQL服务器（如向我们支付开发新特性的费用，或为我们提供用于MySQL服务器开发的硬件）的下述公司，我们深表感谢和赞赏。

· VA Linux / Andover.net

资助复制。

· NuSphere

MySQL手册的编辑。

· Stork Design studio

1998-2000期间使用的MySQL网站。

· Intel

为Windows和Linux平台上的开发提供了捐助。

· Compaq

为Linux/Alpha平台上的开发提供了捐助。

- SWSoft

嵌入式**mysqld**版本的开发

- FutureQuest

--skip-show-database

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

附录D：MySQL变更史

目录

D.1. 5.1.x版中的变更情况（开发）

D.1.1. 5.1.2版中的变更情况（尚未发布）
D.1.2. 5.1.1版中的变更情况（尚未发布）

D.2. MyODBC的变更情况

D.2.1. MyODBC 3.51.12的变更情况
D.2.2. MyODBC 3.51.11的变更情况

在本附录中，列出了MySQL源码在不同版本中的变化，直至最新的MySQL 5.1版本，目前是MySQL 5.1.2-alpha。从MySQL 5.0版开始，我们为每个新的MySQL版本系列提供了新的手册版本（5.0、5.1等）。关于MySQL数据库软件以前版本中的变更信息，请参见该手册的对应版本。关于4.1系列之前旧版MySQL软件的更多信息，请参见*MySQL 4.1*参考手册。

由于在5.1系列中增加了新的特性，我们对本节进行了更新，以便所有人都能跟上发展步伐。

注意，我们倾向于在更改MySQL的同时更新手册。如果你发现这里列出的最新版MySQL无法在我们的下载页面上（http://dev.mysql.com/downloads/）找到，那表示该版本尚未发布。

与发布版本一起给出的日期指得是发布版所基于的最后BitKeeper ChangeSet的日期，不是提供软件包的日期。二进制版的提供日期比标记ChangeSet的日期晚几天，这是因为创建和测试所有的软件包需要一段时间。

当其出现在版本变更日志的条目中时，源码分发版和二进制分发版中包含的手册可能不是完全准确的，这是因为，在创建时还会对手册进行综合处理。关于最新版的变更日志，请参阅在线版本。

D.1. 5.1.x版中的变更情况（开发）

D.1.1. 5.1.2版中的变更情况（尚未发布）
D.1.2. 5.1.1版中的变更情况（尚未发布）

下述变更日志给出了在5.1树中所作的改动：

关于完整的变更清单，请参阅各5.1.x版的变更日志部分。

D.1.1. 5.1.2版中的变更情况（尚未发布）

增加或改变的功能：

更正的缺陷：

- 在TEMPORARY表中，未恰当地强制外键。现在，在TEMPORARY表中，禁止了外键（Bug #12084）

D.1.2. 5.1.1版中的变更情况（尚未发布）

增加或改变的功能：

更正的缺陷：

- 在预处理语句中与PARTITION BY子句一起执行CREATE TABLE语句时，会导致运行在调试模式下的服务器崩溃（Bug #12097）。

- NDB：如果使用分区功能在CREATE TABLE中指定了错误的nodegroup，当CREATE TABLE失败后，会导致表名被锁定（也就是说，无法重新使用表名）（Bug #12114）。

- 在64位操作系统上，在具有分区表的查询中使用ORDER BY时，会导致服务器崩溃（Bug #12116）。

- 当两个线程争用相同的表时，如果某一线程通过LOCK TABLES在另一个表上还存在锁定，而且该线程正试图以某种方式删除表，而另一线程希望在两个表上都存在锁定，此时，将出现死锁（Bug #10600）。

D.2. MyODBC的变更情况

D.2.1. MyODBC 3.51.12的变更情况
D.2.2. MyODBC 3.51.11的变更情况

D.2.1. MyODBC 3.51.12的变更情况

增加或改变的功能：

- N/A

更正的缺陷：

- SQLColumns()未返回关于表（该表有使用保留字命名的列）的信息（Bug #9539）。

D.2.2. MyODBC 3.51.11的变更情况

增加或改变的功能：无变化。

更正的缺陷：

- mysql_list_dbcolumns()和insert_fields()正在检索某一表的所有行。更正了这些函数生成的查询，不返回任何行（Bug #8198）。

- 对于SQL_VARBINARY，SQLGetTypoInfo()返回tinyblob，对于SQL_BINARY，SQLGetTypoInfo()不返回任何内容。予以了更正，对于SQL_VARBINARY，返回varbinary，对于SQL_BINARY，返回binary，对于SQL_LONGVARBINARY，返回longblob（Bug #8138）。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

附录E：移植到其它系统

目录

E.1. 调试MySQL服务器

E.1.1. 针对调试编译MySQL
E.1.2. 创建跟踪文件
E.1.3. 在**gdb**环境下调试**mysqld**
E.1.4. 使用堆栈跟踪
E.1.5. 使用日志文件找出**mysqld**出错原因
E.1.6. 如果出现表格崩溃，请生成测试案例

E.2. 测试MySQL 客户端
E.3. DBUG 软件包
E.4. 关于RTS线程的注释
E.5. 线程软件包之间的差异

这个附录帮助你把MySQL移植到其它操作系统。请先查看一下当前支持操作系统列表。请参阅http://www.mysql.com/），推荐给其它的用户。

注意：如果你创建一个新的MySQL移植，你可以在GPL许可证下任意复制和发布它，但这不能使你成为MySQL的版权持有者。

这个服务器需要一个正在工作的POSIX 线程库在。在Solaris 2.5 上我们使用Sun PThreads （在2.4版和更早的版本上，原生线程支持得不是很好），在Linux上，我们使用Xavier Leroy<Xavier.Leroy@inria.fr>的LinuxThreads。

对于那些对原生线程支持不好的新Unix变体，移植到其上的艰难部分大概就是移植MIT-pthreads包。请参阅mit-pthreads/README 和Programming POSIX Threads (http://www.humanfactor.com/pthreads/)。

直到MySQL 4.0.2版，MySQL发布包包括来自MIT经过补丁的Chris Provenzano的Pthreads（请参阅MIT Pthreads 网页http://www.mit.edu/afs/sipb/project/pthreads/ 以及http://www.mit.edu:8001/people/proven/IAP_2000/上的编程指导）。对于某些没有POSIX线程的操作系统可能有用。请参阅2.8.5节，“MIT-pthreads 注意事项”。

也可能会用到另一个名为 FSU Pthreads的用户级线程软件包（请参阅http://moss.csc.ncsu.edu/~mueller/pthreads/）。 这个工具被用来到SCO的移植。

参阅 mysys目录下的thr_lock.c 和thr_alarm.c 程序获取一些关于这些问题的测试/例子。

服务器和客户端需要一个能用的C++编译器。我们在很多平台上使用**gcc**。其它编译器，据了解，可用的编译器是SPARCworks, Sun Forte, Irix **cc**, HP-UX **aCC**, IBM AIX **xlC_r**), Intel **ecc/icc** 和 Compaq **cxx**)。

要仅编译客户端，请使用**./configure --without-server**.

现在不支持仅编译服务器，也不能加这个功能，除非有人找出一个好的理由。

如果你想/需要改变任何Makefile 或配置脚本，你也会需要到GNU Automake 和 Autoconf。请参阅2.8.3节，“从开发源树安装”。

所有步骤需要从最基本的文件重新生成（remake）所有东西。

```
/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
```

如果在新移植MySQL上遇到问题，最好做一些调试！请参阅E.1节，“调试MySQL服务器”。

注意：在你开始调试**mysqld**之前，首先要让测试程序mysys/thr_alarm和mysys/thr_lock工作。这会确保你的线程安装只有非常小的机会能运行！

E.1. 调试MySQL服务器

E.1.1. 为调试编译MySQL
E.1.2. 创建追踪文件
E.1.3. 在**gdb**环境下调试
E.1.4. 使用堆栈跟踪
E.1.5. 使用日志文件找出**mysqld**错误原因
E.1.6. 如果发生表崩溃则做一个测试案例

如果你使用MySQL某些非常新的功能，你可以带--skip-new参数（这个选项禁止掉所有新的潜在不安全的功能）或带 --safe-mode参数（它禁止掉很多可能导致问题的优化设置）来运行**mysqld**。 请参阅A.4.2节，“如果MySQL依旧崩溃，应该做什么”。

如果 **mysqld** 不启动，你应该查证有没有干扰你的设置的my.cnf文件。你可以用**mysqld --print-defaults...**检查my.cnf参量，并用**mysqld --no-defaults**来启动去避免它们。

如果**mysqld** 启动耗尽CPU或内存资源，或者它“挂”了起来，你可以使用 **mysqladmin processlist status**去找出是否有人执行了一个占用很长时间的查询。如果你正面临着性能问题或新客户端不能连 之时的问题，在某些窗口中运行**mysqladmin -i10 processlist status**可能是一个好主意。

mysqladmin debug 命令把一些有关使用中的锁，使用的内存以及查询使用的信息转储到MySQL日志文件里。这将有助于解决一些问题。即使你没有为调试编译MySQL，这个命令也提供一些有用的信息！

如果问题是一些表变得越来越慢，你应该试着用PTIMIZE TABLE或**myisamchk**优化表。I请参阅第5章：数据库管理。你也可以用EXPLAIN检查慢 的查询。

对那些于你的环境是独特的问题，你也应该查阅这个手册里OS规格的部分请参阅 2.12节，“操作系统系统的注意事项”。

E.1.1. 针对调试编译MySQL

如果你遇到一些非常明确的问题，你可以总是试着调试MySQL。要调试MySQL，你必须用--with-debug或--with-debug=full选项来配置MySQL。你可以检查MySQL是否是通过**mysqld --help**来和调试一起编译的。如果--debug标记和选项一起被列出了，你就可以调试了。在这种情况**mysqladmin ver**下把**mysqld**版本列成**mysql ... --debug**。

如果你使用**gcc** 或 **egcs**，推荐的**configure** 行如下：

```
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
```

这避免了libstdc++库和C++异常（很多编译器在线程代码里有C++异常的问题）的问题，并编译了一个支持所有字符集的MySQL版本。

如果你怀疑内存溢出错误，你可以用--with-debug=full来配置MySQL，这会安装一个内存分配（SAFEMALLOC）检查器。可是，运行SAFEMALLOC是非常慢的，所以如果你遇到性能上的问题，你应该 用--skip-safemalloc选项启动**mysqld**。这样禁止掉对调用malloc()和free()的内存检查。

当你用--with-debug编译**mysqld**时，如果它不再崩溃，你大致已经在MySQL内找到一个编译器缺陷或计时缺陷。这种情况下，你可以试着把-g加到上面的CFLAGS和CXXFLAGS变量，并且不使用--with-debug。如果**mysqld**失败，你至少可以**gdb**用附着上它或使用核心文件上的**gdb**去找出发生什么问题。

当你为调试配置MySQL时，你就自动允许许多额外的监视**mysqld**健康的安全检查函数。如果它们发现一些"不期望"的事，会写一个条目到stderr，**safe_mysqld**，指引这个stderr到错误日志！这也意味着如果MySQL发生什么意外的问题，并且你正使用一个源文件发布版本，那么你要做的第一件事就是去为调试配置MySQL！（第二件事是发邮件到MySQL邮件列表请求帮助）。请参阅1.7.1.1节，"MySQL邮件列表"。请根据你使用的MySQL版本对所有缺陷报告或问题使用**mysqlbug**脚本！

在Windows MySQL发布包里，mysqld.exe默认编译为支持追踪文件。

E.1.2. 创建跟踪文件

如果**mysqld** 服务器没有启动或者你可以快速地使其崩溃，你可以创建一个跟踪文件来找出问题。

要这么做的话，你必须有一个编译了支持调试的**mysqld**， 你可以通过执行mysqld -V来检查一下。如果版本号后面跟着-debug，它就是被编译成支持跟踪文件。（在 Windows中，调试服务器被命名为**mysqld-debug**而不是象MySQL 4.1 那样的**mysqld** ）。

如下命令，启动带跟踪文件的 **mysqld** 服务器，跟踪文件位于Unix上的/tmp/mysqld.trace目录里，Windows上的C:\mysqld.trace目录里：

```
shell> mysqld --debug
```

在Windows上，你也可以使用--standalone参数，启动**mysqld**让它不作为服务。在控制台窗口，使用这个命令：

```
C:\> mysqld-debug --debug --standalone
```

完毕之后，你可以使用第二个窗口中的 mysql.exe 命令行工具重新制造问题。你可以用**mysqladmin shutdown**命令停止**mysqld**服务器。

注意，跟踪文件会变得很大！如果你想生成一个小一点的跟踪文件，你可以使用类似这样的调制选项：

mysqld --debug=d,info,error,query,general,where:O,/tmp/mysqld.trace

这样就仅把带最感兴趣标记的信息写进跟踪文件里.

如果你生成一个有关于此的缺陷报告，请只用把跟踪文件中的相关行发送到恰当的邮件列表去，那里关注你报告出问题的部分。如果你不能找出哪里出问题，你可以ftp上载整个跟踪文件到ftp://ftp.mysql.com/pub/mysql/upload/，并附有完全的缺陷报告，MySQL开发人员会看到它的。

追踪文件是由Fred Fish用**DBUG**软件包生成的，请参阅E.3节，"DBUG软件包"。

E.1.3. 在**gdb**环境下调试**mysqld**

如果**mysqld**崩溃了，在大多数系统上，你也可是从**gdb**启动**mysqld**来获取更多信息。

Linux上，有一些老版本的**gdb**，如果你想要能调试**mysqld**线程，你必须使用run --one-threadsome。在这种情况下，你可以一次只激活一个线程。我们推荐你升级到gdb 5.1 ASAP ，这个版本上线程调试工作得更好！

NTPL 线程（Linux上的新线程库）可能会在**gdb**下运行**mysqld**时遇到问题。一些症状如下：

- **mysqld** 在启动过程中挂起（在它写ready for connections之前）。
- **mysqld** 在调用pthread_mutex_lock()或pthread_mutex_unlock()过程中崩溃。

在这种情况下你应该在启动**gdb**之前在外壳上设置如下环境变量：

```
LD_ASSUME_KERNEL=2.4.1
export LD_ASSUME_KERNEL
```

在**gdb**下运行**mysqld**时，你应该用--skip-stack-trace来禁止堆栈跟踪，以便能捕获**gdb**内的段错误。

在MySQL 4.0.14和以上版本，你应该对mysqld使用--gdb选项。 这会为SIGINT安装一个中断处理器（需要用^C停止**mysqld**来设置断点），并且禁止堆栈跟踪和核心文件处理。

当**gdb**没有给旧线程释放内存的整个时间里，如果你做了大量的新连接，在**gdb**下调试MySQL是非常困难的。你可以通过带 -O thread_cache_size= 'max_connections +1' 启动**mysqld** 来避免这个问题。在多数情况下，只使用-O thread_cache_size=5'就受益无穷了！

如果**mysqld**带着SIGSEGV信号死掉了，而你想在Linux上转储核心，你可以带--core-file选项启动**mysqld**。这个核心文件可以被用来生成 向后跟踪，它可以帮你找出**mysqld** 为何死掉：

```
shell> gdb mysqld core
gdb>   backtrace full
gdb>   exit
```

请参阅A.4.2节，“如果MySQL依旧崩溃，该如何去做”。

如果你在Linux上使用**gdb** 4.17.x 或以上版本，你应该安装一个带有如下信息的 .gdb 文件到你当前目录：

```
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
```

如果你用**gdb**调试线程遇到问题，你应该下载gdb 5.x版本并用它试一下调试。新版本的 **gdb** 大大改善了线程处理！

下面是如何调试mysqld的例子：

```
shell> gdb /usr/local/libexec/mysqld
gdb> run
...
backtrace full # Do this when mysqld crashes
```

把上面的输入写进一个用**mysqlbug**生成的邮件里，发送到综合MySQL邮件列表。请参阅1.7.1.1节，“MySQL邮件列表”。

如果**mysqld** 挂起，你可以试着用一些诸如strace 或 /usr/proc/bin/pstack 这样的系统工具连检查**mysqld** 在哪里挂起。

```
strace /tmp/log libexec/mysqld
```

如果你使用 Perl DBI 接口，你可以使用trace方法或设置DBI_TRACE环境变量来打开调试信息。

E.1.4. 使用堆栈跟踪

在一些操作系统上，如果**mysqld**意外死掉，错误日志包含一个堆栈跟踪。你可以用它来找出**mysqld** 在哪里（也许可能找出为什么）死掉。请参阅5.11.1节，“错误日志”。要获得堆栈跟踪，你不能用-fomit-frame-pointer 选项编译**mysqld** 为gcc。 请参阅E.1.1节，“针对调试编译MySQL”。

如果错误文件包含类似下面的一些内容:

```
mysqld got signal 11;
The manual section 'Debugging a MySQL server' tells you how to use a
stack trace and/or the core file to produce a readable backtrace that may
help in finding out why mysqld died
Attempting backtrace. You can use the following information to find out
where mysqld died.  If you see no messages after this, something went
terribly wrong...
stack range sanity check, ok, backtrace follows
0x40077552
0x81281a0
0x8128f47
0x8127be0
0x8127995
0x8104947
0x80ff28f
0x810131b
0x80ee4bc
0x80c3c91
0x80c6b43
0x80c1fd9
0x80c1686
```

你可以使用如下步骤找出**mysqld**在什么地方出现问题:

1. 复制前面的数字到一个文件，如mysqld.stack。
2. 为 **mysqld** 服务器生成符号文件:

   ```
   nm -n libexec/mysqld > /tmp/mysqld.sym
   ```

   注意，多数MySQL二进制发布包（"debug" 软件包，包含这些信息的地方就在二进制发布包本身之内）带上述文件，在其中这些文件名为mysqld.sym.gz。在这种情况下，你可以简单地解压缩它:

   ```
   gunzip < bin/mysqld.sym.gz > /tmp/mysqld.sym
   ```

3. 执行 resolve_stack_dump -s /tmp/mysqld.sym -n mysqld.stack.

   这个命令会打印出**mysqld**死在哪里。如果这个不能帮你找出**mysqld**为什么死掉，你应该生成一个缺陷报告，并在缺陷报告里包含上述命令的输出结果。

   注意，尽管在多数情况下，仅有一个堆栈跟踪不能帮助我们找出问题的原因。为了定位缺陷或找到一个大致范围，我们在大多数情况下需要知道杀掉**mysqld**的查询，并最好知道一个测试案例 ，以便我们能重复问题！请参阅1.7.1.3节，“如何报告缺陷和问题”。

E.1.5. 使用日志文件找出**mysqld**中的错误原因

注意，在带--log选项启动 **mysqld**之前，你应该用**myisamchk**检查所有的表。请参阅第5章：数据库管理.

如果 **mysqld** 死了或挂起，你应该用--log启动 **mysqld** 。当**mysqld** 再次死掉时，你可以检查日志文件的最

后，找出杀掉**mysqld**的查询。

如果你不带文件名使用 --log ，日志被保存在名为*host_name*.log的数据库目录里。在多数情况下，日志文件中的最后一个查询杀掉**mysqld**，但如果有可能，你应该重启**mysqld**并从**mysq**命令行工具执行找到的查询来验证一下。如果这个查询杀掉了**mysqld**，你也应该测试所有没有完成的复杂查询。

你也可以在所有占用长时间的SELECT声明上用命令EXPLAIN来确认 **mysqld**正适当地使用索引。请参阅7.2.1节，"EXPLAIN 语法（获得关于SELECT的信息）"。

你可以带 --log-slow-queries启动**mysqld**来找出占用长时间来执行的查询。请参阅5.11.4节 ，"缓慢查询日志"。

如果你在错误日志文件（通常名为hostname.err）中发现 mysqld restarted 字样，你大致已经找到导致**mysqld**的查询。如果发生这种情况，你应该用**myisamchk**检查所有表（参阅 第5章：数据库管理），并在MySQL日志文件中测试这些查询看是否有不执行的。如果找到这样一个查询，试着升级到最新的MySQL版本。如果这样不能帮助你，你不能在mysql邮件存档中发现任何相关内容，你应该把缺陷报告给MySQL邮件列表。邮件列表在http://lists.mysql.com/订阅，这个地址上也有连到在线列表存档的链接。

如果你已经用myisam-recover启动了**mysqld**，MySQL自动检查并试着修复MyISAM 表，看是否它们被标志为"未正常关闭"或"崩溃"。如果发生这种情况，MySQL在文件hostname.err 写一个条目'Warning: Checking table ...'，后面跟着警告：如果表需要修复，请修复它。如果你遇上大量的这些错误而**mysqld**没有意外死掉，那就是有问题了，需要进一步调查。请参阅5.3.1节，"**mysqld**命令行选项"。

如果**mysqld**意外死掉，这可不是一个好兆头，但在这种情况下不用研究Checking table...信息，而是要找出**mysqld**为什么死掉。

E.1.6. 如果出现表崩溃，请生成测试案例

如果在一些更新命令之后，**mysqld**总是当掉，或者如果你遇到被破坏的表，你可以用下面的操作测试看这个缺陷是否是可重复产生的：

- 卸掉MySQL守护进程（用**mysqladmin shutdown**）。
- 给该表做备份（防止修复操作反而搞坏这种很不可能出现的情况）。
- 用 **myisamchk -s database/*.MYI** 检查所有的表，用**myisamchk -r database/***table***.MYI**修理有错误的表。
- 对该表做第二次备份。
- 如果需要更多的空间就从MySQL数据库目录删除（或移走）旧日志文件。
- 带--log-bin启动Start **mysqld** 。请参阅5.11.3节，"二进制日志"。如果你想找出搞垮**mysqld**的查询，你应该使用use --log --log-bin。
- 当你已经遭遇一个被破坏的表时，请停止mysqld server 。
- 还原备份。
- 不带--log-bin重启动**mysqld** 服务器。
- 重新执行**mysqlbinlog update-log-file | mysql**命令。更新的日志用名字hostname-bin.#保存在MySQL数据库目录下。
- 如果该表再次被破坏，或者你可用上诉命令让**mysqld** 死掉，你就已经找到可重复产生的缺陷，它应该

很容易被修复！可以ftp上传表和二进制日志到 ftp://ftp.mysql.com/pub/mysql/upload/ 然后把它输入我们在 http://bugs.mysql.com/ 上的缺陷系统。（请注意，/pub/mysql/upload/ 在FTP时是不可以列出（内容）的，所以不能在FTP客户端看见你已经上载的东西。）如果你是一个支持客户，你可以使用MySQL客户支持中心https://support.mysql.com/ 来 提醒MySQL 技术人员这个问题，让这个问题尽快得到解决。

如果你想缩小问题的范围，你也可以使用 mysql_find_rows脚本来只执行一些 更新语句。

E.2. 调试MySQL客户端

为能够用集成的调试软件包调试MySQL客户端 ，你应该用--with-debug或--with-debug=full配置MySQL。请参阅2.8.2节，“典型的配置选项”。

在运行客户端之前，你应该设置 MYSQL_DEBUG环境变量：

```
shell> MYSQL_DEBUG=d:t:O,/tmp/client.trace
shell> export MYSQL_DEBUG
```

这会导致客户端在 /tmp/client.trace目录产生一个跟踪文件。

如果你自己的客户端代码有问题，你应该试着连接到服务器，用已知可用的客户端运行你的查询。在调试模式下，按下面命令运行（假设你已经带调试编译了MySQL）：

```
shell> mysql --debug=d:t:O,/tmp/client.trace
```

万一你要发送一个缺陷报告邮件，这会提供给你有用的信息。请查阅“如何报告缺陷或问题”。

如果你的客户端在一些看起来合法的代码处崩溃了，你应该检查你的mysql.h文件是否包括匹配你的MySQL库文件。一个常见的错误就是用新的版本的MySQL库使用一个来自老版本安装的mysql.h文件。

E.3. DBUG软件包

MySQL服务器和多数MySQL客户端都带着由Fred Fish初创的DBUG 软件包编译成的。当你为调试配置MytSQL之时，这个软件使你可以得到一个程序正在调试什么的跟踪文件。请参阅E.1.2节，“创建跟踪文件”。

这一节总结了你对已建立支持调试的MySQL程序在命令行的调试选项处可以指定的参量值。要获取更多使用DBUG软件包来编程的信息，请参阅MySQL源发布包里dbug目录下的DBUG手册。最好使用最近的MySQL 5.1发布包以获得最近更新的DBUG手册。

你通过用--debug="..."或the -#... 选项调用一个程序来使用调试软件包。

多数MySQL程序有默认的调试字符串，如果你不给--debug指定一个选项，就使用这个默认的。默认的跟踪文件通常是/tmp/program_name.trace（在Unix上）和\program_name.trace （在Windows上）。

调试字符串是一系列冒号隔开的区段，如下：

```
<field_1>:<field_2>:...:<field_N>
```

每个区段包含一个强制标志字符，后面跟着已和可选的 ‘,’ 以及一列用逗号隔开的修改量：

```
flag[,modifier,modifier,...,modifier]
```

当前被识别的标记符号是:

标记	描述
d	允许对当前状态从DBUG_<N>宏输出。可能跟着一列关键词，这些关键词仅对那些带有关键词的DBUG宏选择输出。一个空的关键词列意味着对所有宏输出。
D	在每个调试起输出行后延迟。参量一个十分之一秒为单位来延迟的数，它受限于机器的能力。比如 -#D,20 指定一个2秒的延迟。
f	限制调试和/或跟踪，以及简单设定于列出名字的函数。注意，空列将禁止所用函数。应该给出适当的d 或 t 标记，如果它们被允许了，这个标记仅限制它们的动作。
F	对调试或跟踪输出的每一行识别源文件名。
i	对调试或跟踪输出的每一行用PID或线程ID识别进程。
g	允许解析，创建名为的dbugmon.out文件，它包含可用来简单设定程序的信息。可能跟着一列关键词，它们是选择只对列中的函数做简单设定。一个空列意味着所有函数都要考虑到。
L	为调试或跟踪输出的每一行识别源文件行号。
n	为调试或跟踪输出的每一行打印当前函数嵌套深度。
N	给调试输出的每一行编号。
o	重定向调试器输出流到指定文件。默认输出是stderr 文件。
O	类似于 o，但是文件在每次写操作之间被冲刷。当需要之时，文件在每次写操作之间被关闭然后重新打开。
p	限制调试器作用于指定进程。为使调试器动作，一个进程必须用DBUG_PROCESS宏来识别，且匹配列表中的一个。
P	为调试或跟踪输出的每一行打印当前进程名字。
r	当推出一个新状态时，不继承前状态的操作嵌套深度级别。当输出在左边空白开始时有用。
S	在每个调试过的函数做_sanity(_file_,_line_)函数直到 _sanity() 返回不同于0的结果。（大多数的时候与safemalloc 一起用来找出内存漏洞）。
t	允许函数调用/退出跟踪行。可能跟着一个给出最大跟踪级别的数字列（只含一个修改量），超过这个数字，调试中或跟踪中的宏不能产生任何输出。 默认为一个编译时间选项。

可能出现在外壳命令行（-# 典型地被用来引入一个控制字符串到一个应用程序中） 的调试控制字符串的一些例子如下:

```
-#d:t
-#d:f,main,subr1:F:L:t,20
-#d,input,output,files:n
-#d:t:i:O,\\mysqld.trace
```

在MySQL中, 打印的一般标记是（用 d 选项）是 enter, exit, error, warning, info, 和 loop 。

E.4. 关于RTS线程的注释

我曾尝试让MySQL使用RTS线程软件包，但是在下面的问题上遇到阻碍:

RTS线程软件包很多老版本的POSIX调用，对所有函数的写封装就很枯燥。我倾向于认为把线程库换成最新的POSIX规格，会更容易些。。

一些封装正在编写中。更多信息请参阅mysys/my_pthread.c 文件。

至少下面说道的应该改变一下:

pthread_get_specific该使用一个参量。 sigwait应该使用两个参量。很多函数（至少pthread_cond_wait, pthread_cond_timedwait()）应该返回错误的错误代码。现在它们返回 -1 且设置 errno。

另一个问题是，用户级线程使用ALRM信号，这会终止很多函数（read, write, open...）。MySQL应该重试一下所有这上面的中断，但是这并非很容易去验证。

最大的未解决问题如下:

要获得线程级警报，我使用pthread_cond_timedwait()改变 mysys/thr_alarm.c，让它在警报之间等待。但是它发生EINTR错误，终止了。我试着调试线程库找出为什么会出这个错误，但是找不到一个简便 的解决办法。

如果人人想要用RTS线程跑一下MySQL，我建议以下几点:

- 把MySQL使用的函数从线程库变到POSIX。这不会占据那么长时间。
- 用-DHAVE_rts_threads编译所有库。
- 编译thr_alarm。
- 若在执行中有一些小的差异，可以改变my_pthread.h和my_pthread.c来修复它们。
- 运行thr_alarm。如果它没有任何警告，错误或终止信息地运行，你就做对了。这里是一个在Solaris成功运行的例子:

```
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.5. 线程软件包之间的差异

MySQL非常依赖使用中的线程软件包。 所以当为MySQL选择一个好平台的时候，线程软件包就非常重要。

至少有三种线程软件包:

- 用户线程在单个进程中。线程切换用警报管理，线程库用锁管理所有非线程安全函数。读，写和选择操作通常被线程专有的切换器管理， 如果运行中的线程要等待数据，这个切换器就会切换操作到另一个线程。如果用户线程软件包集成在标准库（FreeBSD 和 BSDI 线程软件包）里，这样的 线程软件包比那些不得不映射所有不安全调用（MIT-pthreads, FSU Pthreads 和 RTS 线程软件包）的线程软件包需要更少的系统开销。在某些环境下（如SCO），所有系统调用都是线程安全的，所以映射非常容易（SCO上的FSU Pthreads包）。不足之处是：所有映射的调用占用很少的时间，于是想要能处理所有的情况就相当繁杂。有一些系统调用通常不被线程软件包（类似MIT-pthreads and sockets包）处理。线程计划不总是最优化的。

- 在分离进程中的用户线程。线程切换是由内核来做，且所有的数据在线程之间共享。线程软件包管理标准线程调用，允许在线程之间共享数据。LinuxThreads包就使用这种方法。不足之处：太多进程。线程创建得慢，如果一个线程死掉了，其余得线程通常就挂起来，你必须在重启之前杀掉这些挂起的线程。线程切换开销有些大。

- 内核线程。线程切换由线程库或内核来做，并且非常快。一个进程就可以了。但在一些系统中**ps**可能显示不同线程。如果一个线程终止，整个进程就终止了。多数系统 调用是线程安全的，并且只要非常小的系统开销。Solaris, HP-UX, AIX 和OSF/1 都有内核线程。

在一些系统中内核线程被系统库中整合用户级线程管理。在这种情况下，线程切换只能由线程库来做，而内核并不是真正的“线程感知”的。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

附录F：环境变量

在本附录中，列出了MySQL直接或间接使用的环境变量。大多数环境变量均能在本手册的其他部分找到。

注意，命令行上的任何选项均优先于选项文件和环境变量中指定的值，而且选项文件中的值优先于环境变量中的值。。

在很多情况下，要想更改MySQL的行为，应优先使用选项文件而不是环境变量。请参见4.3.2节，“使用选项文件”。

变量	描述
CXX	C++编译器的名称（用于运行**configure**）
CC	C编译器的名称（用于运行**configure**）
CFLAGS	C编译器的标志（用于运行**configure**）
CXXFLAGS	C++编译器的标志（用于运行**configure**）
DBI_USER	Perl DBI的默认用户名
DBI_TRACE	用于Perl DBI的跟踪选项
HOME	**mysql**历史文件的默认路径是$HOME/.mysql_history。
LD_RUN_PATH	用于指定libmysqlclient.so的位置。
MYSQL_DEBUG	调试时的调试跟踪选项。
MYSQL_HISTFILE	**mysql**历史文件的路径。如果设置了该变量，它的值将覆盖$HOME/.mysql_history的默认值。
MYSQL_HOST	**mysql**命令行客户端使用的默认主机名。
MYSQL_PS1	在**mysql**命令行客户端中使用的命令提示。
MYSQL_PWD	连接到**mysqld**时的默认密码。注意，使用它并不安全。请参见5.8.6节，“使你的密码安全”。
MYSQL_TCP_PORT	默认的TCP/IP端口号
MYSQL_UNIX_PORT	默认的Unix套接字文件名，用于连接到本地主机。
PATH	由shell使用以找到MySQL程序。
TMPDIR	创建临时文件的目录。
TZ	用于设置当地时区。请参见A.4.6节，“时区问题”。
UMASK_DIR	创建目录时的用户目录创建掩码。注意，这是带有UMASK的ANDed。
UMASK	创建文件时的用户文件创建掩码。
USER	在Windows和NetWare上连接到**mysqld**时使用的默认用户名。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

附录G：MySQL正则表达式

正则表达式是为复杂搜索指定模式的强大方式。

MySQL采用Henry Spencer的正则表达式实施，其目标是符合POSIX 1003.2。请参见附录C：感谢。MySQL采用了扩展的版本，以支持在SQL语句中与REGEXP操作符一起使用的模式匹配操作。请参见3.3.4.7节，“模式匹配”。

在本附录中，归纳了在MySQL中可用于REGEXP操作的特殊字符和结构，并给出了一些示例。本附录未包含可在Henry Spencer的regex(7)手册页面中发现的所有细节。该手册页面包含在MySQL源码分发版中，位于regex目录下的regex.7文件中。

正则表达式描述了一组字符串。最简单的正则表达式是不含任何特殊字符的正则表达式。例如，正则表达式hello匹配hello。

非平凡的正则表达式采用了特殊的特定结构，从而使得它们能够与1个以上的字符串匹配。例如，正则表达式hello|word匹配字符串hello或字符串word。

作为一个更为复杂的示例，正则表达式B[an]*s匹配下述字符串中的任何一个：Bananas，Baaaaas，Bs，以及以B开始、以s结束、并在其中包含任意数目a或n字符的任何其他字符串。

对于REGEXP操作符，正则表达式可以使用任何下述特殊字符和结构：

- ^

匹配字符串的开始部分。

```
mysql> SELECT 'fo\nfo' REGEXP '^fo$';                    -> 0
mysql> SELECT 'fofo' REGEXP '^fo';                       -> 1
```

- $

匹配字符串的结束部分。

```
mysql> SELECT 'fo\no' REGEXP '^fo\no$';                  -> 1
mysql> SELECT 'fo\no' REGEXP '^fo$';                     -> 0
```

- .

匹配任何字符（包括回车和新行）。

```
mysql> SELECT 'fofo' REGEXP '^f.*$';                     -> 1
mysql> SELECT 'fo\r\nfo' REGEXP '^f.*$';                 -> 1
```

- a*

匹配0或多个a字符的任何序列。

```
mysql> SELECT 'Ban' REGEXP '^Ba*n';                      -> 1
mysql> SELECT 'Baaan' REGEXP '^Ba*n';                    -> 1
mysql> SELECT 'Bn' REGEXP '^Ba*n';                       -> 1
```

- a+

匹配1个或多个a字符的任何序列。

```
mysql> SELECT 'Ban' REGEXP '^Ba+n';                    -> 1
mysql> SELECT 'Bn' REGEXP '^Ba+n';                     -> 0
```

- a?

匹配0个或1个a字符。

```
mysql> SELECT 'Bn' REGEXP '^Ba?n';                     -> 1
mysql> SELECT 'Ban' REGEXP '^Ba?n';                    -> 1
mysql> SELECT 'Baan' REGEXP '^Ba?n';                   -> 0
```

- de|abc

匹配序列de或abc。

```
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)*

匹配序列abc的0个或多个实例。

```
mysql> SELECT 'pi' REGEXP '^(pi)*$';                   -> 1
mysql> SELECT 'pip' REGEXP '^(pi)*$';                  -> 0
mysql> SELECT 'pipi' REGEXP '^(pi)*$';                 -> 1
```

- {1}, {2,3}

{n}或{m,n}符号提供了编写正则表达式的更通用方式，能够匹配模式的很多前述原子（或"部分"）。m和n均为整数。

o a*

可被写入为a{0,}。

o a+

可被写入为a{1,}。

o a?

可被写入为a{0,1}。

更准确地讲，a{n}与a的n个实例准确匹配。a{n,}匹配a的n个或更多实例。a{m,n}匹配a的m～n个实例，包含m和n。

m和n必须位于0～RE_DUP_MAX（默认为255）的范围内，包含0和RE_DUP_MAX。如果同时给定了m和n，m必须小于或等于n。

```
mysql> SELECT 'abcde' REGEXP 'a[bcd]{2}e';              -> 0

mysql> SELECT 'abcde' REGEXP 'a[bcd]{3}e';              -> 1

mysql> SELECT 'abcde' REGEXP 'a[bcd]{1,10}e';           -> 1
```

· [a-dX], [^a-dX]

匹配任何是（或不是，如果使用^的话）a、b、c、d或X的字符。两个其他字符之间的"-"字符构成一个范围，与从第1个字符开始到第2个字符之间的所有字符匹配。例如，[0-9]匹配任何十进制数字 。要想包含文字字符"]"，它必须紧跟在开括号"["之后。要想包含文字字符"-"，它必须首先或最后写入。对于[]对内未定义任何特殊含义的任何字符，仅与其本身匹配。

```
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.]

在括号表达式中（使用[和]），匹配用于校对元素的字符序列。字符为单个字符或诸如新行等字符名。在文件regexp/cname.h中，可找到字符名称的完整列表。

```
mysql> SELECT '~' REGEXP '[[.~.]]';                     -> 1

mysql> SELECT '~' REGEXP '[[.tilde.]]';                 -> 1
```

· [=character_class=]

在括号表达式中（使用[和]），[=character_class=]表示等同类。它与具有相同校对值的所有字符匹配，包括它本身，例如，如果o和(+)均是等同类的成员，那么[[=o=]]、[[=(+)=]]和[o(+)]是同义词。等同类不得用作范围的端点。

· [:character_class:]

在括号表达式中（使用[和]），[:character_class:]表示与术语类的所有字符匹配的字符类。标准的类名称是：

alnum	文字数字字符
alpha	文字字符
blank	空白字符
cntrl	控制字符
digit	数字字符
graph	图形字符
lower	小写文字字符
print	图形或空格字符
punct	标点字符
space	空格、制表符、新行、和回车
upper	大写文字字符

xdigit	十六进制数字字符

它们代表在ctype(3)手册页面中定义的字符类。特定地区可能会提供其他类名。字符类不得用作范围的端点。

```
mysql> SELECT 'justalnums' REGEXP '[[:alnum:]]+';       -> 1

mysql> SELECT '!!' REGEXP '[[:alnum:]]+';               -> 0
```

- [[:<:]], [[:>:]]

这些标记表示word边界。它们分别与word的开始和结束匹配。word是一系列字字符，其前面和后面均没有字字符。字字符是alnum类中的字母数字字符或下划线(_)。

```
mysql> SELECT 'a word a' REGEXP '[[:<:]]word[[:>:]]';   -> 1

mysql> SELECT 'a xword a' REGEXP '[[:<:]]word[[:>:]]';  -> 0
```

要想在正则表达式中使用特殊字符的文字实例，应在其前面加上2个反斜杠"\"字符。MySQL解析程序负责解释其中一个，正则表达式库负责解释另一个。例如，要想与包含特殊字符"+"的字符串"1+2"匹配，在下面的正则表达式中，只有最后一个是正确的：

```
mysql> SELECT '1+2' REGEXP '1+2';                       -> 0

mysql> SELECT '1+2' REGEXP '1\+2';                      -> 0

mysql> SELECT '1+2' REGEXP '1\\+2';                     -> 1
```

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

附录H：MySQL中的限制

目录

H.1. 联合的限制

本附录列出了MySQL 5.1中的当前限制。

H.1. 联合的限制

在单个联合中可引用的最大表数为61。它也适用于在视图定义中可引用的最大表数。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

附录I：特性限制

目录

I.1. 对存储子程序和触发程序的限制
I.2. 对服务器端光标的限制
I.3. 对子查询的限制
I.4. 对视图的限制
I.5. 对XA事务的限制

在本附录中，介绍了使用诸如子查询或视图等MySQL特性时存在的限制。

I.1. 对存储子程序和触发程序的限制

这里介绍的某些限制适用于所有的存储子程序，即存储程序和存储函数。某些限制仅适用于存储函数而不是存储程序。

对存储函数的的所有限制也适用于触发程序。

注释：如果SQL语句，如SELECT ... INTO语句包含具有相同名称的对列的引用以及声明的局部变量，MySQL会将引用解释为变量的名称。这是一种非标准的行为，优先顺序通常是列名、然后是SQL变量和参数。请参见20.2.9.3节，“SELECT ... INTO语句”。

存储子程序不能包含任意SQL语句。在存储子程序中，禁止使用下述语句：

- CHECK TABLES
- LOCK TABLES, UNLOCK TABLES
- LOAD DATA, LOAD TABLE
- SQL预处理语句（PREPARE、EXECUTE、DEALLOCATE PREPARE）。隐含意义：不能在存储子程序中使用动态SQL语句（其中，能够以字符串形式构造动态语句，然后执行它们）。从MySQL 5.0.13开始，对于存储程序放宽了该限制，但该限制仍适用于存储函数和触发程序。
- OPTIMIZE TABLE

对于存储函数（而不是存储程序），禁止下述额外语句：

- 执行显式或隐式提交或回滚操作的语句。
- 返回结果集的语句。包括没有INFO子句的SELECT语句，以及SHOW语句。能够用SELECT ... INTO，或使用光标和FETCH语句处理结果集的函数。
- FLUSH语句。注意，尽管能够在存储程序中使用FLUSH，但不能从存储函数或触发程序调用这类存储程序。

注意，尽管某些限制在正常情况下适用于存储函数和触发程序，不适用于存储程序，如果它们是从存储函数或触发程序中调用的，这些限制也适用于存储程序。

使用存储子程序会导致复制问题。关于这方面的进一步讨论，请参见20.4节，“存储子程序和触发程序的二进制日志功能”。

INFORMATION_SCHEMA尚不包含PARAMETERS表，因此，对于需要在运行时获取子程序参数信息的应用程序来说，必须采用相应的规避错误，如解析SHOW CREATE语句的输出。

没有存储子程序调试工具。

存储子程序使用了具体化的光标，而不是固有光标（在服务器端生成结果集并对结果集进行高速缓冲处理，然后在客户端获取结果集时按行返回）。

不能提前处理CALL语句。无论是对服务器端预处理语句还是SQL预处理语句，均成立。

为了防止服务器线程间的交互问题，当客户端发出语句时，服务器将使用可用的、用于语句执行的子程序和触发程序快照。也就是说，服务器将计算出可在语句执行期间使用的存储程序、函数和触发程序的列表，加载它们，然后进入语句执行。这意味着，在语句执行的同时，它不会看到其他线程对子程序所作的变更。

I.2. 对服务器端光标的限制

从MySQL 5.0.2开始，通过mysql_stmt_attr_set() C API函数实现了服务器端光标。服务器端光标允许在服务器端生成结果集，但不会将其传输到客户端，除非客户端请求这些行。例如，如果客户端执行了查询，但仅对第1行感兴趣，那么不会传输剩余的行。

光标是只读的，不能使用光标来更新行。

未实施UPDATE WHERE CURRENT OF和DELETE WHERE CURRENT OF，这是因为不支持可更新的光标。

光标是不可保持的（提交后不再保持打开）。

光标是不敏感的。

光标是不可滚动的。

光标是未命名的。语句处理程序起着光标ID的作用。

对于每条预处理语句，仅能打开1个光标。如果需要多个光标，必须处理多条语句。

如果在预处理模式下不支持语句，不能在生成结果集的语句上使用光标。包括CHECK TABLES、HANDLER READ和SHOW BINLOG EVENTS语句。

I.3. 对子查询的限制

随后将更正的一致缺陷：如果将NULL值与使用ALL、ANY或SOME的子查询进行比较，而且子查询返回空结果，比较操作将对NULL的非标准结果进行评估，而不是TRUE或FALSE。

子查询的外部语句可以是SELECT、INSERT、UPDATE、DELETE、SET或DO中的任何一个。

仅部分支持行比较操作：

- 对于*expr* IN (*subquery*)，*expr*可以是*n*-tuple（通过行构造程序语法指定），而且子查询能返回*n*-tuples个行。

- 对于*expr op* {ALL|ANY|SOME} (*subquery*)，*expr*必须是标度值，子查询必须是列子查询，不能返回多列行。

换句话讲，对于返回*n*-tuples行的子查询，支持：

```
(val_1, ..., val_n) IN (subquery)
```

但不支持：

```
(val_1, ..., val_n) op {ALL|ANY|SOME} (subquery)
```

支持针对IN的行比较，但不支持针对其他的行比较，原因在于，IN实施是通过将其重新编写为"="比较和AND操作的序列完成的。该方法不能用于ALL、ANY或SOME。

未良好优化行构造程序。下面的两个表达式是等效的，但只有第2个表达式能被优化：

```
(col1, col2, ...) = (val1, val2, ...)
```

```
col1 = val1 AND col2 = val2 AND ...
```

对于IN的子查询优化不如对"="的优化那样有效。

对于不良IN性能的一种典型情况是，当子查询返回少量行，但外部查询返回将与子查询结果相比较的大量行。

FROM子句中的子查询不能与子查询有关系。在评估外部查询之前，将对它们进行具体化处理（执行以生成结果集），因此，不能按照外部查询的行对它们进行评估。

一般而言，不能更改表，并从子查询内的相同表进行选择。例如，该限制适用于具有下述形式的语句：

```
DELETE FROM t WHERE ... (SELECT ... FROM t ...);
```

```
UPDATE t ... WHERE col = (SELECT ... FROM t ...);
```

```
{INSERT|REPLACE} INTO t (SELECT ... FROM t ...);
```

例外：如果为FROM子句中更改的表使用子查询，前述禁令将不再适用。例如：

```
UPDATE t ... WHERE col = (SELECT (SELECT ... FROM t...) AS _t ...);
```

禁令在此不适用，这是因为FROM中的子查询已被具体化为临时表，因此"t"中的相关行已在满足"t"条件的情况下、在更新时被选中。

与子查询相比，针对联合的优化程序更成熟，因此，在很多情况下，如果将其改写为join（联合），使用子查询的语句能够更有效地执行。

但下述情形例外：IN子查询可被改写为SELECT DISTINCT联合。例如：

```
SELECT col FROM t1 WHERE id_col IN (SELECT id_col2 FROM t2 WHERE condition);
```

可将该语句改写为：

```
SELECT DISTINCT col FROM t1, t2 WHERE t1.id_col = t2.id_col AND condition;
```

但在该情况下，联合需要额外的DISTINCT操作，而且与子查询相比，效率并不高。

可能的未来优化：MySQL不改写针对子查询评估的联合顺序。在某些情况下，如果MySQL将其改写为联合，能够更有效地执行子查询。这样，优化程序就能在更多的执行方案间进行选择。例如，它能决定是否首先读取某一表或其他。

例如：

```
SELECT a FROM outer_table AS ot
```

```
WHERE a IN (SELECT a FROM inner_table AS it WHERE ot.b = it.b);
```

对于该查询，MySQL总会首先扫描outer_table，如然后针对每一行在inner_table上执行子查询。如果outer_table有很多行而inner_table只有少量行，查询的执行速度或许要慢于本应有的速度。

前述查询可改写为：

```
SELECT a FROM outer_table AS ot, inner_table AS it

WHERE ot.a = it.a AND ot.b = it.b;
```

在该情况下，我们能扫描小的表（inner_table）并查询outer_table中的行，如果在"ot.a,ot.b"上有索引，速度会更快。

可能的未来优化：对外部查询的每一行评估关联的子查询。更好的方法是，如果外部行的值与之前的行相比没有变化，不对子查询进行再次评估，而是使用以前的结果。

可能的未来优化：通过将结果具体化到临时表，而且该表不使用索引，对FROM子句中的子查询进行评估。在查询中与其他表进行比较时，尽管可能是有用的，但不允许使用索引。

可能的未来优化：如果FROM子句中的子查询类似于可施加MERGE算法的视图，改写查询并采用MERGE算法，以便能够使用索引。下述语句包含这类子查询：

```
SELECT * FROM (SELECT * FROM t1 WHERE t1.t1_col) AS _t1, t2 WHERE t2.t2_col;
```

该语句可被改写为联合，如下所示：

```
SELECT * FROM t1, t2 WHERE t1.t1_col AND t2.t2_col;
```

这类改写具有两个优点：

1. 避免使用那些不能使用索引的临时表。在改写的查询中，优化程序可在t1上使用索引。

2. 优化程序在选择不同的执行计划方面具有更大的自由。例如，将查询改写为联合，那么就允许优化程序首先使用t1或t2。

可能的未来优化：对于没有关联子查询的IN、= ANY、<> ANY、= ALL、以及<> ALL，为结果使用"内存中"哈希处理，或对较大的结果使用具有索引的临时表。例如：

```
SELECT a FROM big_table AS bt

WHERE non_key_field IN (SELECT non_key_figld FROM table WHERE condition)
```

在该情况下，可创建临时表：

```
CREATE TABLE t (key (non_key_field))

(SELECT non_key_field FROM table WHERE condition)
```

然后，对big_table中的每一行，根据bt.non_key_field，在"t"中进行键查找。

I.4. 对视图的限制

视图处理功能概念未优化：

- 不能在视图上创建索引。

- 对于使用MERGE算法处理的视图，可以使用索引。但是，对于使用临时表算法处理的视图，不能在其基表上利用索引提供的优点（尽管能够在临时表的生成过程中使用索引）。

在视图的FROM子句中不能使用子查询。未来该限制将被放宽。

存在一个一般原则，不能更改某一表并在子查询的相同表内进行选择。请参见I.3节，“对子查询的限制”。

如果从表选择了视图并接着从视图进行了选择，同样的原理也适用，如果在子查询中从表选择了视图并使用MERGE算法评估了视图，也同样。例如：

```
CREATE VIEW v1 AS

SELECT * FROM t2 WHERE EXISTS (SELECT 1 FROM t1 WHERE t1.a = t2.a);


UPDATE t1, v2 SET t1.a = 1 WHERE t1.b = v2.b;
```

如果视图是使用临时表评估的，可从视图子查询中的表进行选择，并仍能更改外部查询中的表。在该情况下，视图将被具体化，因此，你实际上不能从子查询的表中进行选择并“同时”更改它（这是你打算强制MySQL使用临时表算法的另一原因，其方法是在视图定义中指定ALGORITHM = TEMPTABLE关键字）。

可以使用DROP TABLE或ALTER TABLE来舍弃或更改视图定义中使用的表（它会是视图失效），而且舍弃或更改操作不会导致告警。但在以后使用视图时会出错。

视图定义是通过特定语句“冻结”的：

- 如果PREPARE预处理的语句引用了视图，以后每次执行语句时看到的视图内容与预处理视图时的内容相同。即使在语句预处理完成之后、在执行语句之前更改了视图定义，情况也同样。例如：

```
CREATE VIEW v AS SELECT 1;
PREPARE s FROM 'SELECT * FROM v';
ALTER VIEW v AS SELECT 2;
EXECUTE s;
```

EXECUTE语句返回的结果是1，而不是2。

- 如果存储子程序中的语句引用了视图，语句所见到的视图内容与首次执行语句时的相同。这意味着，如果语句是以循环方式执行的，进一步的语句迭代见到的视图内容是相同的，即使在循环过程中更改了视图定义也同样。例如：

```
CREATE VIEW v AS SELECT 1;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 0;
  WHILE i < 5 DO
    SELECT * FROM v;
    SET i = i + 1;
    ALTER VIEW v AS SELECT 2;
  END WHILE;
END;
//
delimiter ;
```

- `CALL p();`

如果调用了程序p()，每次通过循环时SELECT返回1，即使在循环内更改了视图定义也同样。

关于视图的可更新性，对于视图，其总体目标是，如果任何视图从理论上讲是可更新的，在实际上也应是可更新的。这包括在其定义中有UNION的视图。目前，并非所有理论上可更新的视图均能被更新。最初的视图实施有意采用该方式，为的是尽快地在MySQL中获得有用的可更新视图。很多理论上可更新的视图现已能更新，但限制依然存在：

- 其子查询位于WHERE子句之外任何位置的可更新视图。对于某些其子查询位于SELECT列表中的视图，也是可更新的。
- 不能使用UPDATE来更新定义为Join的视图的1个以上的基表。
- 不能使用DELETE来更新定义为Join的视图。

I.5. 对XA事务的限制

XA事务支持限于InnoDB存储引擎。

MySQL XA实施是针对外部XA的，其中，MySQL服务器作为资源管理器，而客户端程序作为事务管理器。未实施"内部XA"。这样，就允许MySQL服务器内的单独存储引擎作为RM（资源管理器），而服务器本身作为TM（事务管理器）。处理包含1个以上存储引擎的XA事务时，需要内部XA。内部XA的实施是不完整的，这是因为，它要求存储引擎在表处理程序层面上支持两阶段提交，目前仅对InnoDB实现了该特性。

对于XA START，不支持JOIN和RESUME子句。

对于XA END，不支持SUSPEND [FOR MIGRATE]子句。

在全局事务内，对于每个XA事务，xid值的bqual部分应是不同的，该要求是对当前MySQL XA实施的限制。它不是XA规范的组成部分。

如果XA事务达到PREPARED状态而且MySQL服务器宕机，当服务器重启后，能够继续处理事务。就像原本应当的那样。但是，如果客户端连接中止而服务器继续运行，服务器将回滚任何未完成的XA事务，即使该事务已达到PREPARED状态也同样。它应能提交或回滚PREPARED XA事务，但在不更改二进制日志机制的情况下不能这样。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

附录J：GNU通用公共许可

1991.6 第二版

版权所有©，1989，1991，Free Software Foundation有限公司

59 Temple Place，Suite 330，Boston，MA 02111-1307，USA
允许每个人复制和发布这一许可证原始文档的副本，但绝对不允许对它进行任何修改。

前言

大多数软件许可证决意剥夺你的共享和修改软件的自由。对比之下，GNU通用公共许可证力图保证你的共享和修改自由软件的自由。保证自由软件对所有用户是自由的。该GPL适用于大多数自由软件基金会的软件，以及由使用这些软件而承担义务的作者所开发的软件。（自由软件基金会的其他一些软件受GNU库通用许可证的保护）。你也可以将它用到你的程序中。

当我们谈到自由软件（free software）时，我们指的是自由而不是价格。我们的GNU通用公共许可证决意保证你有发布自由软件的自由（如果你愿意，你可以对此项服务收取一定的费用）；保证你能收到源码或者在你需要时能得到它；保证你能修改软件或将它的一部分用于新的自由软件；而且还保证你知道你能做这些事情。

为了保护你的权利，我们需要做出规定：禁止任何人不承认你的权利，或者要求你放弃这些权利。如果你修改了自由软件或者发布了软件的副本，这些规定就转化为你的责任。

例如，如果你发布这样一个程序的副本，不管是收费的还是免费的，你必须将你具有的一切权利给予你的接受者；你必须保证他们能收到或得到源码；并且将这些条款给他们看，使他们知道他们有这样的权利。

我们采取两项措施来保护你的权利：（1）给软件以版权保护。（2）给你提供许可证。它给你复制，发布和修改这些软件的法律许可。

同样，为了保护每个作者和我们自己，我们需要清楚地让每个人明白，自由软件没有担保（no warranty）。如果由于其他某个人修改了软件，并继续加以传播。我们需要它的接受者明白：他们所得到的并不是原来的自由软件。由其他人引入的任何问题，不应损害原作者的声誉。

最后，任何自由软件不断受到软件专利的威胁。我们希望避免这样的风险，自由软件的再发布者以个人名义获得专利许可证。事实上，将软件变为私有。为防止这一点，我们必须明确：任何专利必须以允许每个人自由使用为前提，否则就不准许有专利。

下面是有关复制，发布和修改的确切的条款和条件。

GNU通用公共许可证有关复制，发布和修改的条款和条件

1.　此许可证适用于任何包含版权所有者声明的程序和其他作品，版权所有者在声明中明确说明程序和作品可以在GPL条款的约束下发布。下面提到的"程序"指的是任何这样的程序或作品。而"基于程序的作品"指的是程序或者任何受版权法约束的衍生作品。也就是说包含程序或程序的一部分的作品。可以是原封不动的，或经过修改的和／或翻译成其他语言的（在下文中，翻译包含在条款"修改"中，无任何限制）。每个许可证接受人（licensee）用你来称呼。

许可证条款不适用于复制，发布和修改以外的活动。这些活动超出这些条款的范围。运行程序的活动不受条款的限止。仅当程序的输出构成基于程序作品的内容时，这一条款才适用（如果只运行程序就无关）。是否普遍适用取决于程序具体用来做什么。

2.　只要你在每一副本上明显和恰当地出版版权声明和拒绝担保声明，保持此许可证的声明和没有担保的声明完整无损，并和程序一起给每个其他的程序接受者一份许可证的副本，你就可以用任何媒体复制和发布你收到

的原始的程序的源代码。

你可以为转让副本的实际行动收取一定费用。你也有权选择提供担保以换取一定的费用。

3. 你可以修改程序的一个或几个副本或程序的任何部分，以此形成基于程序的作品。只要你同时满足下面的所有条件，你就可以按前面第一款的要求复制和发布这一经过修改的程序或作品：

a. 你必须在修改的文件中附有明确的说明：你修改了这一文件及具体的修改日期。

b. 你必须使你发布或出版的作品（它包含程序的全部或一部分，或包含由程序的全部或部分衍生的作品）允许第三方作为整体按许可证条款免费使用

c. 如果修改的程序在运行时以交互方式读取命令，你必须使它在开始进入常规的交互使用方式时打印或显示声明：包括适当的版权声明和没有担保的声明（或者你提供担保的声明）；用户可以按此许可证条款重新发布程序的说明；并告诉用户如何看到这一许可证的副本。（例外的情况：如果原始程序以交互方式工作，它并不打印这样的声明，你的基于程序的作品也就不用打印声明）。

这些要求适用于修改了的作品的整体。如果能够确定作品的一部分并非程序的衍生产品，可以合理地认为这部分是独立的，是不同的作品。当你将它作为独立作品发布时，它不受此许可证和它的条款的约束。但是当你将这部分作为基于程序的作品的一部分发布时，作为整体它将受到许可证条款约束。准予其他许可证持有人的使用范围扩大到整个产品。也就是每个部分，不管它是谁写的。

因此，本条款的意图不在于索取权利；或剥夺全部由你写成的作品的权利。而是履行权利来控制基于程序的集体作品或衍生作品的发布。

此外，将与程序无关的作品和该程序或基于程序的作品一起放在存贮体或发布媒体的同一卷上，并不导致将其他作品置于此许可证的约束范围之内。

4. 你可以以目标码或可执行形式复制或发布程序（或符合第2款的基于程序的作品)，只要你遵守前面的第1，2款，并同时满足下列3条中的1条：

a. 在通常用作软件交换的媒体上，和目标码一起附有机器可读的完整的源码。这些源码的发布应符合上面第1，2款的要求。或者

b. 在通常用作软件交换的媒体上，和目标码一起，附有给第三方提供相应的机器可读的源码的书面报价。有效期不少于3年，费用不超过实际完成源程序发布的实际成本。源码的发布应符合上面的第1，2款的要求。或者

c. 和目标码一起，附有你收到的发布源码的报价信息。（这一条款只适用于非商业性发布，而且你只收到程序的目标码或可执行代码和按b）款要求提供的报价）。

作品的源码指的是对作品进行修改最优先择取的形式。对可执行的作品讲，完整的源码包括：所有模块的所有源代码，加上有关的接口的定义，加上控制可执行作品的安装和编译的脚本。作为特殊例外，发布版（以源码形式或二进制形式）不必包含任何常规发布的供可执行代码在上面运行的操作系统的主要组成部分（如编译程序，内核等）。除非这些组成部分和可执行作品结合在一起。

如果采用提供对指定地点的访问和复制的方式发布可执行码或目标码，那么，提供对同一地点的访问和复制源码可以算作源码的发布，即使第三方不强求与目标码一起复制源码。

5. 除非你明确按许可证提出的要求去做，否则你不能复制，修改，转发许可证和发布程序。任何试图用其他方式复制，修改，转发许可证和发布程序是无效的。而且将自动结束许可证赋予你的权利。然而，对那些从你那里按许可证条款得到副本和权利的人们，只要他们继续全面履行条款，许可证赋予他们的权利仍然有效。

6. 你没有在许可证上签字，因而你没有必要一定接受这一许可证。然而，没有任何其他东西赋予你修改和发布程序及其衍生作品的权利。如果你不接受许可证，这些行为是法律禁止的。因此，如果你修改或发布程序

（或任何基于程序的作品），你就表明你接受这一许可证以及它的所有有关复制，发布和修改程序或基于程序的作品的条款和条件。

7. 每当你重新发布程序（或任何基于程序的作品）时，接受者自动从原始许可证颁发者那里接到受这些条款和条件支配的复制，发布或修改程序的许可证。你不可以对接受者履行这里赋予他们的权利强加其他限制。你也没有强求第三方履行许可证条款的义务。

8. 如果由于法院判决或违反专利的指控或任何其他原因（不限于专利问题）的结果，强加于你的条件（不管是法院判决，协议或其他）和许可证的条件有冲突，他们也不能用许可证条款为你开脱。在你不能同时满足本许可证规定的义务及其他相关的义务时，作为结果，你可以根本不发布程序。例如，如果某一专利许可证不允许所有那些直接或间接从你那里接受副本的人们在不付专利费的情况下重新发布程序，唯一能同时满足两方面要求的办法是停止发布程序。

如果本节的任何部分在特定的环境下无效或无法实施，就使用条款的其余部分,并将条款作为整体用于其他环境。

本节的目的不在于引诱你侵犯专利或其他财产权的要求，或争论这种要求的有效性。本条款的唯一目的在于保护自由软件发布系统的完整性，它是通过通用公共许可证的应用来实现的。许多人坚持应用这一系统，已经为通过这一系统发布大量自由软件做出慷慨的供献。作者／捐献者有权决定他或她是否通过任何其他系统发布软件。许可证持有人不能强制这种选择。

本节的目的在于明确说明许可证其余部分可能产生的结果。

9. 如果由于专利或者由于有版权的接口问题使程序在某些国家的发布和使用受到限止，将此程序置于许可证约束下的原始版权拥有者可以增加限止发布地区的条款，将这些国家明确排除在外。并在这些国家以外的地区发布程序。在这种情况下，许可证包含的限止条款和许可证正文一样有效。

10.自由软件基金会可能随时出版通用公共许可证的修改版和/或新版。新版和当前的版本在原则上保持一致，但在提到新问题时或有关事项时，在细节上可能出现差别。

每一版本都有不同的版本号。如果程序指定适用于它的许可证版本号以及"任何更新的版本"。你有权选择遵循指定的版本或自由软件基金会以后出版的新版本，如果程序未指定许可证版本，你可选择自由软件基金会已经出版的任何版本。

11.如果你愿意将程序的一部分结合到其他自由程序中，而它们的发布条件不同。写信给作者，要求准予使用。如果是自由软件基金会加以版权保护的软件，写信给自由软件基金会。我们有时会作为例外的情况处理。我们的决定受两个主要目标的指导。这两个主要目标是：我们的自由软件的衍生作品继续保持自由状态。以及从整体上促进软件的共享和重复利用。

无担保

12.由于程序准予免费使用，在适用法准许的范围内，对程序没有担保。除非另有书面说明，版权所有者和／或其他提供程序的人们"一样"不提供任何类型的担保。不论是明确的，还是隐含的。包括但不限于隐含的适销和适合特定用途的保证。全部的风险，如程序的质量和性能问题都由你来承担。如果程序出现缺陷，你承担所有必要的服务，修复和改正的费用。

13.除非适用法或书面协议的要求，在任何情况下，任何版权所有者或任何按许可证条款修改和发布程序的人们都不对你的损失负有任何责任。包括由于使用或不能使用程序引起的任何一般的，特殊的，偶然发生的或重大的损失（包括但不限于数据的损失，或者数据变得不精确，或者你或第三方的持续的损失，或者程序不能和其他程序协调运行等）。即使版权所有者和其他人提到这种损失的可能性也不例外。

条款和条件结束

如何将这些条款用到你的新程序

如果你开发了新程序，而且你需要它得到公众最大限度的利用。要做到这一点的最好办法是将它变为自由软件。使得每个人都能在遵守条款的基础上对它进行修改和重新发布。

为了做到这一点，给程序附上下列声明。最安全的方式是将它放在每个源码文件的开头，以便最有效地传递拒绝担保的信息。每个文件至少应有"版权所有"行以及在什么地方能看到声明全文的指示。

用一行空间给出程序的名称和它用来做什么的简单说明。

版权所有（C）,yyyy，作者姓名

这一程序是自由软件，你可以遵照自由软件基金会出版的GNU通用公共许可证条款来修改和重新发布这一程序。或者用许可证的第二版，或者（根据你的选择）用任何更新的版本。

发布这一程序的目的是希望它有用，但没有任何担保。甚至没有适合特定目的的隐含的担保。更详细的情况请参阅GNU通用公共许可证。

你应该已经和程序一起收到一份GNU通用公共许可证的副本。如果还没有，

写信给：Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA。

此外，还应加上以电子邮件和纸张邮件方式与你保持联系的信息。

如果程序以交互方式进行工作，当它开始进入交互方式工作时，使它输出类似下面的简短声明：
Gnomovision 第69版， 版权所有（C） 19XX， 作者姓名，

Gnomovision绝对没有担保。要想了解详细情况，请输入'show w'。

这是自由软件，欢迎你遵守一定的条件重新发布它，要想了解详细情况，请输入'show c'。

假设的命令'show w'和'show c'应显示通用公共许可证的恰当部分。当然，你使用的命令名称可以不同于'show w'和'show c'。根据你的程序的具体情况，也可以用菜单或鼠标选项来显示这些条款。

如果需要，你应该取得你的雇主（如果你是程序员）或你的学校签署放弃程序版权的声明。下面只是一个例子，你应该改变相应的名称：

Yoyodyne有限公司以此方式放弃James Harker所写的Gnomovision程序（在编译器上通过）的全部版权利益。

*Ty coon*签名，1989.4.1

Ty coon付总裁

该通用公共许可证不允许你将程序并入专用程序。如果你的程序是一个子程序库，你可能会认为用库的方式和专用应用程序链接更有用。如果这是你想做的事，使用GNU库通用公共许可证代替本许可证。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

附录K：MySQL FLOSS许可例外

MySQL AB关于使用MySQL客户端库的免费／自由软件以及开发源码软件应用程序的例外（“FLOSS例外”）。

版本*0.4*，*2005*年*9*月*8*日

例外意图

我们希望指定的免费／自由软件以及开发源码软件（FLOSS）应用程序能够使用指定的、获得GPL许可的MySQL客户端库（程序），而不考虑这样的事实，即并非所有的FLOSS许可均与GNU通用公共许可（GPL）2.0版相容。

法律术语和条款

作为对GPL 2.0版所规定术语和条款的特别例外：

1. 你能自由分发完全由“程序”构成的衍生作品，以及获得第1节所列的1个或多个许可计划许可的1个或多个作品（每个称为1个FLOSS作品），但应遵循下述条件：

a. 遵从GPL关于“程序”及其衍生作品的规定，除非衍生作品的可确认部分不是由“程序”衍生而来的，因而可将其本身合理地视为独立和单独的作品。

b. 衍生作品的所有可确认部分不是由“程序”衍生而来的，因而可将其本身合理地视为独立和单独的作品。

i. 对于这类作品，可按照下面所列的FLOSS许可之一予以分发，以及

ii. 在相同的媒介上并依据相同的FLOSS许可，对于这些部分的目标码或可执行形式，应与这些部分对应的机器可读的完整的源码一起提供，作为这些部分对应的目标码或可执行形式，以及

c. 依据GPL，对于在同一存储卷或分发媒介上与“程序”或衍生作品一起提供的任何作品，如果它们不是“程序”、“衍生作品”或“FLOSS作品”的衍生物，其本身可被合理地视为独立和单独的作品。

如果不满足上述条件，仅能根据GPL或MySQL AB公司其他有效许可选项的条款和条件，在此条件下拷贝、更改、分发或使用“程序”。

2. **FLOSS**许可列表

许可名称	版本/版权日期
学院免费许可	2.0
Apache软件许可	1.0/1.1/2.0
Apple公共源码许可	2.0
艺术许可	从Perl 5.8.0起
BSD许可	"July 22 1999"
普通公众许可	1.0
GNU库或"Lesser"通用公共许可(LGPL)	2.0/2.1
Jabber公共源码许可	1.0
MIT许可	-
Mozilla公共许可(MPL)	1.0/1.1
开放软件许可	2.0

OpenSSL许可（与原始的SSLeay许可）	"2003" ("1998")
PHP许可	3.0
Python许可(CNRI Python许可)	—
Python软件基金会许可	2.1.1
Sleepycat许可	"1999"
W3C许可	"2001"
X11许可	"2001"
Zlib/libpng许可	—
Zope公共许可	2.0

3. 由于上述许可中的一些存在很多变体，我们要求任何版本均应符合自由软件基金会的自由软件定义第2003版（http://www.gnu.org/philosophy/free-sw.html）或Open Source Initiative的开放源码定义第1.9版（http://www.opensource.org/docs/definition.php）。

4. 定义

a. 这里所使用而不是定义的术语的含义与GPL中给出的相同。

b. 衍生作品指得是根据版权法衍生的作品。

5. 适用性：该FLOSS例外适用于包含MySQL AB声明的所有"程序"，在这类声明中指明："程序"可依据本FLOSS例外的条款分发。如果你创建或分发了某一作品，而该作品是遵从GPL许可的"程序"和任何其他作品的衍生作品，那么本FLOSS例外不适用于该作品，因此，你必须将FLOSS例外声明从作品中删除，并在任何方面遵从GPL的要求，包括保留所有的GPL声明。你可以从程序的副本中删除FLOSS例外声明，仅依照GPL的条款再分发程序的副本，但前提是程序未被你或任何第三方更改。

附录**A**。有资格的库和软件包

下面列出了部分遵从FLOSS许可例外的库和软件包。请注意，本附录仅是作为特定FLOSS项目的额外服务而提供的，目的是为了简化针对用户的许可信息，但前提是需遵从"FLOSS许可列表"部分中列出的许可之一。

软件包名称	许可授权和版本
Apache Portable Runtime (APR)	Apache软件许可2.0

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。

索引

符号

! (logical NOT)，逻辑操作符
!= (not equal)，比较函数和操作符
"，数据库、表、索引、列和别名
% (modulo)，数学函数
% (wildcard character)，字符串
& (bitwise AND)，位函数
&& (logical AND)，逻辑操作符
() (parentheses)，圆括号
(Control-Z) \Z，字符串
* (multiplication)，算术操作符
+ (addition)，算术操作符
- (subtraction)，算术操作符
- (unary minus)，算术操作符
--password option，使你的密码安全
-p option，使你的密码安全
.my.cnf file，使用选项文件，与MySQL服务器连接，拒绝访问错误的原因，使你的密码安全，在多服务器环境中使用客户端程序
.mysql_history file，选项
.pid (process ID) file，建立表维护计划
/ (division)，算术操作符
/etc/passwd，使MySQL在攻击者面前保持安全，SELECT语法
< (less than)，比较函数和操作符
<<，根据天计算访问量
<< (left shift)，位函数
<= (less than or equal)，比较函数和操作符
<=> (equal to)，比较函数和操作符
<> (not equal)，比较函数和操作符
= (equal)，比较函数和操作符
> (greater than)，比较函数和操作符
>= (greater than or equal)，比较函数和操作符
>> (right shift)，位函数
\" (double quote)，字符串
\' (single quote)，字符串
\0 (ASCII 0)，字符串
\b (backspace)，字符串
\n (linefeed)，字符串
\n (newline)，字符串
\r (carriage return)，字符串
\t (tab)，字符串
\Z (Control-Z) ASCII 26，字符串
\\ (escape)，字符串
^ (bitwise XOR)，位函数
_ (wildcard character)，字符串
`，数据库、表、索引、列和别名
| (bitwise OR)，位函数
|| (logical OR)，逻辑操作符
~，位函数

A

aborted clients，通信错误和失效连接
aborted connection，通信错误和失效连接
ABS()，数学函数
access control，访问控制 ，阶段1：连接核实
access denied errors，拒绝访问
access privileges，MySQL访问权限系统
Access program，已知的能与MyODBC一起工作的程序
account privileges

    adding，向MySQL增加新用户账户

accounts

    anonymous user，使初始MySQL账户安全
    root，使初始MySQL账户安全

ACID，事务和原子操作，InnoDB概述
ACLs，MySQL访问权限系统
ACOS()，数学函数
ActiveState Perl，在Windows下安装ActiveState Perl
ADDDATE()，日期和时间函数
adding

    character sets，添加新的字符集
    native functions，添加新的固有函数
    new account privileges，向MySQL增加新用户账户
    new functions，为MySQL添加新函数
    new user privileges，向MySQL增加新用户账户
    new users，在其它类Unix系统中安装MySQL，源码安装概述
    procedures，为MySQL添加新步骤
    user-defined functions，添加新的自定义函数

addition (+)，算术操作符
ADDTIME()，日期和时间函数
ADO program，已知的能与MyODBC一起工作的程序
AES_DECRYPT()，加密函数
AES_ENCRYPT()，加密函数
age

    calculating，日期计算

alias，与列别名有关的问题
alias names

    case sensitivity，识别符大小写敏感性

aliases

    for expressions，具有隐含字段的GROUP BY
    for tables，SELECT语法
    in GROUP BY clauses，具有隐含字段的GROUP BY
    in ORDER BY clauses，具有隐含字段的GROUP BY
    names，数据库、表、索引、列和别名

on expressions，SELECT语法

ALL，SELECT语法
ALLOW_INVALID_DATES SQL mode，SQL服务器模式
ALTER COLUMN，ALTER TABLE语法
ALTER DATABASE，ALTER DATABASE语法
ALTER FUNCTION，ALTER PROCEDURE和ALTER FUNCTION
ALTER PROCEDURE，ALTER PROCEDURE和ALTER FUNCTION
ALTER SCHEMA，ALTER DATABASE语法
ALTER TABLE，ALTER TABLE语法，与ALTER TABLE有关的问题
ALTER VIEW，ALTER VIEW语法
altering

database，ALTER DATABASE语法
schema，ALTER DATABASE语法

ANALYZE TABLE，ANALYZE TABLE语法
AND

bitwise，位函数
logical，逻辑操作符

anonymous user，使初始MySQL账户安全，访问控制 ，阶段1：连接核实，访问控制 ，阶段2：请求核实
ANSI mode

running，在ANSI模式下运行MySQL

ANSI SQL mode，SQL服务器模式
ANSI_QUOTES SQL mode，SQL服务器模式
answering questions

etiquette，在邮件列表上回答问题的指南

Apache，与Apache一起使用MySQL
API's

list of，支持MySQL的软件包

APIs，API和库

Perl，MySQL Perl API

approximate-value literals，精度数学
ARCHIVE storage engine，存储引擎和表类型，ARCHIVE存储引擎
ARCHIVE table type，存储引擎和表类型，ARCHIVE存储引擎
Area()，Polygon函数，MultiPolygon函数
argument processing，UDF参量处理
arithmetic expressions，算术操作符
arithmetic functions，位函数
AS，SELECT语法，JOIN语法
AsBinary()，Geometry格式转换函数
ASCII()，字符串函数
ASIN()，数学函数
AsText()，Geometry格式转换函数
ATAN()，数学函数

ATAN2()，数学函数
attackers

security against，使MySQL在攻击者面前保持安全

AUTO-INCREMENT

ODBC，如何在ODBC中获取AUTO_INCREMENT列的值

AUTO_INCREMENT，使用AUTO_INCREMENT

and NULL values，与NULL值有关的问题

AVG()，GROUP BY（聚合）函数
AVG(DISTINCT)，GROUP BY（聚合）函数

B

backslash

escape character，文字值

backspace (\b)，字符串
BACKUP TABLE，BACKUP TABLE语法
backups，数据库备份

database，BACKUP TABLE语法

batch

mysql option，选项

batch mode，在批处理模式下使用mysql
BDB storage engine，存储引擎和表类型，BDB (BerkeleyDB)存储引擎
BDB table type，存储引擎和表类型，BDB (BerkeleyDB)存储引擎
BDB tables，事务和原子操作
BdMPolyFromText()，使用WKT函数创建Geometry（几何）值
BdMPolyFromWKB()，使用WKB函数创建Geometry（几何）值
BdPolyFromText()，使用WKT函数创建Geometry（几何）值
BdPolyFromWKB()，使用WKB函数创建Geometry（几何）值
BEGIN，START TRANSACTION，COMMIT和ROLLBACK语法，BEGIN ... END复合语句

XA transactions，XA事务SQL语法

benchmark suite，MySQL基准套件
BENCHMARK()，信息函数
benchmarks，使用自己的基准
BerkeleyDB storage engine，存储引擎和表类型，BDB (BerkeleyDB)存储引擎
BerkeleyDB table type，存储引擎和表类型，BDB (BerkeleyDB)存储引擎
BETWEEN ... AND，比较函数和操作符
Big5 Chinese character encoding，搜索中的大小写敏感性
BIGINT data type，数值类型概述
BIN()，字符串函数
BINARY，Cast函数和操作符
BINARY data type，字符串类型概述，BINARY和VARBINARY类型

binary distributions，MySQL AB编译的MySQL二进制版本

installing，在其它类Unix系统中安装MySQL
on Linux，Linux二进制分发版说明

binary log，二进制日志
BIT data type，数值类型概述
BitKeeper tree，从开发源码树安装
BIT_AND()，GROUP BY（聚合）函数
BIT_COUNT，根据天计算访问量
BIT_COUNT()，位函数
bit_functions

example，根据天计算访问量

BIT_LENGTH()，字符串函数
BIT_OR，根据天计算访问量
BIT_OR()，GROUP BY（聚合）函数
BIT_XOR()，GROUP BY（聚合）函数
BLACKHOLE storage engine，存储引擎和表类型，BLACKHOLE存储引擎
BLACKHOLE table type，存储引擎和表类型，BLACKHOLE存储引擎
BLOB

inserting binary data，字符串
size，列类型存储需求

BLOB columns

default values，BLOB和TEXT类型
indexing，列索引，CREATE TABLE语法

BLOB data type，字符串类型概述，BLOB和TEXT类型
BOOL data type，数值类型概述
BOOLEAN data type，数值类型概述
Borland Builder 4 program，已知的能与MyODBC一起工作的程序
Borland C++ compiler，Borland C++
Boundary()，通用几何函数
brackets

square，列类型

buffer sizes

client，API和库
mysqld server，调节服务器参数

Buffer()，Spatial 操作符
bug reports

criteria for，如何通报缺陷和问题

bugs

known，MySQL中的已知事宜
reporting，如何通报缺陷和问题

bugs database，如何通报缺陷和问题
bugs.mysql.com，如何通报缺陷和问题
building

client programs，创建客户端程序

C

C API

data types，MySQL C API
functions，C API函数概述
linking problems，与C API有关的问题

C Prepared statement API

functions，C API预处理语句函数概述

C++ APIs，MySQL C++ API
C++ Builder，已知的能与MyODBC一起工作的程序
C++ compiler

gcc，典型配置选项

C++ compiler cannot create executables，处理MySQL编译问题
C:\my.cnf file，在多服务器环境中使用客户端程序
CACHE INDEX，CACHE INDEX语法
caches

clearing，FLUSH Syntax

calculating

dates，日期计算

calendar，MySQL使用什么日历？
CALL，CALL语句
calling sequences for aggregate functions

UDF，对于聚合函数的UDF调用顺序

calling sequences for simple functions

UDF，对于简单函数的UDF调用顺序

can't create/write to file，无法创建文件／写入文件
carriage return (\r)，字符串
CASE，控制流程函数，CASE语句
case sensitivity

in identifiers，识别符大小写敏感性
in names，识别符大小写敏感性
in searches，搜索中的大小写敏感性
in string comparisons，字符串比较函数

case-sensitivity

in access checking，权限系统工作原理
of database names，MySQL对标准SQL的扩展
of table names，MySQL对标准SQL的扩展

CAST，Cast函数和操作符
cast functions，Cast函数和操作符
cast operators，Cast函数和操作符
casts，比较函数和操作符，Cast函数和操作符
CC environment variable，典型配置选项，处理MySQL编译问题，环境变量
cc1plus problems，处理MySQL编译问题
CEILING()，数学函数
Centroid()，MultiPolygon函数
CFLAGS environment variable，典型配置选项，处理MySQL编译问题，环境变量
CHANGE MASTER TO，CHANGE MASTER TO语法
ChangeLog，MySQL变更史
changes

log，MySQL变更史
MySQL 5.1，5.1.x版中的变更情况（开发）

changes to privileges，访问控制 ，阶段2：请求核实
changing

column，ALTER TABLE语法
column order，如何更改表中的列顺序
field，ALTER TABLE语法
table，ALTER TABLE语法，与ALTER TABLE有关的问题

changing socket location，典型配置选项，自动启动和停止MySQL，如何保护或更改MySQL套接字文件/tmp/mysql.sock
CHAR data type，字符串类型概述，String类型
CHAR VARYING data type，字符串类型概述
CHAR()，字符串函数
CHARACTER data type，字符串类型概述
character sets，典型配置选项，数据和排序用字符集

adding，添加新的字符集

Character sets，字符集支持
CHARACTER VARYING data type，字符串类型概述
character-sets-dir

mysql option，选项

characters

multi-byte，多字节字符支持

CHARACTER_LENGTH()，字符串函数
CHARACTER_SETS

INFORMATION_SCHEMA table，INFORMATION_SCHEMA CHARACTER_SETS表

CHARSET()，信息函数
CHAR_LENGTH()，字符串函数

check options

myisamchk，用于myisamchk的检查选项

CHECK TABLE，CHECK TABLE语法
checking

tables for errors，如何检查MyISAM表的错误

checksum errors，Solaris注意事项
CHECKSUM TABLE，CHECKSUM TABLE语法
Chinese，搜索中的大小写敏感性
choosing

a MySQL version，选择要安装的MySQL分发版

choosing types，选择正确的列类型
clearing

caches，FLUSH Syntax

client programs

building，创建客户端程序

client tools，API和库
clients

debugging，调试MySQL客户端
threaded，如何生成线程式客户端

CLOSE，Cursor CLOSE Statement
closing

tables，MySQL如何打开和关闭表

COALESCE()，比较函数和操作符
COERCIBILITY()，信息函数
ColdFusion program，已知的能与MyODBC一起工作的程序
collating

strings，字符串比较支持

COLLATION()，信息函数
COLLATIONS

INFORMATION_SCHEMA table，INFORMATION_SCHEMA COLLATIONS表

COLLATION_CHARACTER_SET_APPLICABILITY

INFORMATION_SCHEMA table，INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY表

column

changing，ALTER TABLE语法

column comments，CREATE TABLE语法
column names

case sensitivity，识别符大小写敏感性

columns

changing，如何更改表中的列顺序
indexes，列索引
names，数据库、表、索引、列和别名
other types，使用来自其他数据库引擎的列类型
selecting，选择特殊列
storage requirements，列类型存储需求
types，列类型

COLUMNS

INFORMATION_SCHEMA table，INFORMATION_SCHEMA COLUMNS表

COLUMN_PRIVILEGES

INFORMATION_SCHEMA table，INFORMATION_SCHEMA COLUMN_PRIVILEGES表

command syntax，本手册采用的惯例
command-line history

mysql，选项

command-line options，mysqld命令行选项

mysql，选项
mysqladmin，mysqladmin：用于管理MySQL服务器的客户端

commands

for binary distribution，在其它类Unix系统中安装MySQL
replication masters，用于控制主服务器的SQL语句
replication slaves，用于控制从服务器的SQL语句

commands out of sync，命令不同步
Comment syntax，注释语法
comments

adding，注释语法
starting，'--'作为注释起始标记

COMMIT，事务和原子操作，START TRANSACTION，COMMIT和ROLLBACK语法

XA transactions，XA事务SQL语法

comparison operators，比较函数和操作符
compatibility

with mSQL，字符串比较函数
with ODBC，识别符限制条件，数值类型概述，比较函数和操作符，CREATE TABLE语法，JOIN语法
with Oracle，MySQL对标准SQL的扩展，GROUP BY（聚合）函数，DESCRIBE语法（获取有关列的信

息）
with PostgreSQL，MySQL对标准SQL的扩展
with standard SQL，MySQL标准的兼容性
with Sybase，USE语法

compiler

C++ gcc，典型配置选项

compiling

on Windows，在Windows下编译MySQL客户端
optimizing，系统因素和启动参数的调节
problems，处理MySQL编译问题
speed，编译和链接怎样影响MySQL的速度
statically，典型配置选项
user-defined functions，编译和安装用户定义的函数

compliance

Y2K，2000年兼容性

composite subpartitioning，子分区
compress

mysql option，选项

COMPRESS()，字符串函数
compressed tables，压缩表特性
CONCAT()，字符串函数
CONCAT_WS()，字符串函数
concurrent inserts，锁定方法
Conditions，DECLARE条件
config-file

mysqld_multi option，mysqld_multi：管理多个MySQL服务器的程序

config.cache，处理MySQL编译问题
config.cache file，处理MySQL编译问题
configuration files，拒绝访问错误的原因
configuration options，典型配置选项
configure

running after prior invocation，处理MySQL编译问题

configure option

--with-charset，典型配置选项
--with-collation，典型配置选项
--with-extra-charsets，典型配置选项
--with-low-memory，处理MySQL编译问题

configure script，典型配置选项
connecting

remotely with SSH，用SSH以远程方式从Windows连接到MySQL

to the server，连接与断开服务器，与MySQL服务器连接
verification，访问控制，阶段1：连接核实

connection

aborted，通信错误和失效连接

CONNECTION_ID()，信息函数
Connector/JDBC，连接器
Connector/ODBC，连接器，MySQL Connector/ODBC

reporting problems，如何通报MyODBC问题或缺陷

连接器

MySQL，连接器

connect_timeout variable，选项，mysqladmin：用于管理MySQL服务器的客户端
constant table，EXPLAIN语法（获取关于SELECT的信息），MySQL怎样优化WHERE子句
constraints，MySQL处理约束的方式
CONSTRAINTS

INFORMATION_SCHEMA table，INFORMATION_SCHEMA TABLE_CONSTRAINTS表

Contains()，测试几何类之间空间关系的函数
contributing companies

list of，MySQL支持人员

contributors

list of，MySQL贡献人

control access，访问控制，阶段1：连接核实
control flow functions，控制流程函数
CONV()，字符串函数
conventions

typographical，本手册采用的惯例

CONVERT，Cast函数和操作符
CONVERT TO，ALTER TABLE语法
CONVERT_TZ()，日期和时间函数
ConvexHull()，Spatial 操作符
copying databases，将MySQL数据库拷贝到另一台机器
copying tables，CREATE TABLE语法
COS()，数学函数
COT()，数学函数
COUNT()，GROUP BY（聚合）函数
COUNT(DISTINCT)，GROUP BY（聚合）函数
counting

table rows，计数行

crash，调试MySQL服务器

recovery，将myisamchk用于崩溃恢复
repeated，如果MySQL依然崩溃，应作些什么

crash-me，MySQL基准套件
crash-me program，为可移植性设计应用程序，MySQL基准套件
CRC32()，数学函数
CREATE DATABASE，CREATE DATABASE语法
CREATE FUNCTION，CREATE PROCEDURE和CREATE FUNCTION，CREATE FUNCTION/DROP FUNCTION语法
CREATE INDEX，CREATE INDEX语法
CREATE PROCEDURE，CREATE PROCEDURE和CREATE FUNCTION
CREATE SCHEMA，CREATE DATABASE语法
CREATE TABLE，CREATE TABLE语法
CREATE USER，CREATE USER语法
CREATE VIEW，CREATE VIEW语法
creating

bug reports，如何通报缺陷和问题
database，CREATE DATABASE语法
databases，创建并使用数据库
default startup options，使用选项文件
schema，CREATE DATABASE语法
tables，创建表

creating user accounts，CREATE USER语法
CROSS JOIN，JOIN语法
Crosses()，测试几何类之间空间关系的函数
CR_SERVER_GONE_ERROR，MySQL服务器不可用
CR_SERVER_LOST_ERROR，MySQL服务器不可用
CSV storage engine，存储引擎和表类型，CSV存储引擎
CSV table type，存储引擎和表类型，CSV存储引擎
CURDATE()，日期和时间函数
CURRENT_DATE，日期和时间函数
CURRENT_TIME，日期和时间函数
CURRENT_TIMESTAMP，日期和时间函数
CURRENT_USER()，信息函数
光标，光标
CURTIME()，日期和时间函数
customer support

mailing address，如何通报缺陷和问题

customers

of MySQL，我们已将MySQL用在何处？

CXX environment variable，典型配置选项，处理MySQL编译问题，环境变量
CXXFLAGS environment variable，典型配置选项，处理MySQL编译问题，环境变量

D

data

character sets，数据和排序用字符集

loading into tables，将数据装入表中
retrieving，从表检索信息
size，使你的数据尽可能小

data type

BIGINT，数值类型概述
BINARY，字符串类型概述，BINARY和VARBINARY类型
BIT，数值类型概述
BLOB，字符串类型概述，BLOB和TEXT类型
BOOL，数值类型概述，使用来自其他数据库引擎的列类型
BOOLEAN，数值类型概述，使用来自其他数据库引擎的列类型
CHAR，字符串类型概述，String类型
CHAR VARYING，字符串类型概述
CHARACTER，字符串类型概述
CHARACTER VARYING，字符串类型概述
DATE，日期和时间类型概述，DATETIME、DATE和TIMESTAMP类型
DATETIME，日期和时间类型概述，DATETIME、DATE和TIMESTAMP类型
DEC，数值类型概述
DECIMAL，数值类型概述，精度数学
DOUBLE，数值类型概述
DOUBLE PRECISION，数值类型概述
ENUM，字符串类型概述，ENUM类型
FIXED，数值类型概述
FLOAT，数值类型概述
GEOMETRY，MySQL空间数据类型
GEOMETRYCOLLECTION，MySQL空间数据类型
INT，数值类型概述
INTEGER，数值类型概述
LINESTRING，MySQL空间数据类型
LONG，BLOB和TEXT类型
LONGBLOB，字符串类型概述
LONGTEXT，字符串类型概述
MEDIUMBLOB，字符串类型概述
MEDIUMINT，数值类型概述
MEDIUMTEXT，字符串类型概述
MULTILINESTRING，MySQL空间数据类型
MULTIPOINT，MySQL空间数据类型
MULTIPOLYGON，MySQL空间数据类型
NATIONAL CHAR，字符串类型概述
NCHAR，字符串类型概述
NUMERIC，数值类型概述
POINT，MySQL空间数据类型
POLYGON，MySQL空间数据类型
REAL，数值类型概述
SET，字符串类型概述，SET类型
SMALLINT，数值类型概述
TEXT，字符串类型概述，BLOB和TEXT类型
TIME，日期和时间类型概述，TIME类型
TIMESTAMP，日期和时间类型概述，DATETIME、DATE和TIMESTAMP类型
TINYBLOB，字符串类型概述
TINYINT，数值类型概述
TINYTEXT，字符串类型概述
VARBINARY，字符串类型概述，BINARY和VARBINARY类型

VARCHAR，字符串类型概述，String类型
VARCHARACTER，字符串类型概述
YEAR，日期和时间类型概述，YEAR类型

data types，列类型

C API，MySQL C API
overview，列类型概述

database

altering，ALTER DATABASE语法
creating，CREATE DATABASE语法
deleting，DROP DATABASE语法
mysql option，选项

database design，设计选择
Database information

obtaining，SHOW语法

database metadata，INFORMATION_SCHEMA信息数据库
database names

case sensitivity，识别符大小写敏感性
case-sensitivity，MySQL对标准SQL的扩展

DATABASE()，信息函数
databases

backups，数据库备份
copying，将MySQL数据库拷贝到另一台机器
creating，创建并使用数据库
defined，MySQL数据库管理系统概述
information about，获得数据库和表的信息
names，数据库、表、索引、列和别名
replicating，MySQL中的复制
selecting，创建并选择数据库
symbolic links，在Unix上使用针对数据库的符号链接
using，创建并使用数据库

DataJunction，已知的能与MyODBC一起工作的程序
DATE，使用DATE列方面的问题
date and time functions，日期和时间函数
Date and Time types，日期和时间类型
date calculations，日期计算
DATE columns

problems，使用DATE列方面的问题

DATE data type，日期和时间类型概述，DATETIME、DATE和TIMESTAMP类型
date functions

Y2K compliance，2000年兼容性

date types，列类型存储需求

Y2K issues，Y2K事宜和日期类型

date values

problems，DATETIME、DATE和TIMESTAMP类型

DATE()，日期和时间函数
DATEDIFF()，日期和时间函数
DATETIME data type，日期和时间类型概述，DATETIME、DATE和TIMESTAMP类型
DATE_ADD()，日期和时间函数
DATE_FORMAT()，日期和时间函数
DATE_SUB()，日期和时间函数
DAY()，日期和时间函数
DAYNAME()，日期和时间函数
DAYOFMONTH()，日期和时间函数
DAYOFWEEK()，日期和时间函数
DAYOFYEAR()，日期和时间函数
db table

sorting，访问控制 ，阶段2：请求核实

DB2 SQL mode，SQL服务器模式
DBI interface，MySQL Perl API
DBI->quote，字符串
DBI->trace，在gdb环境下调试mysqld
DBI/DBD interface，MySQL Perl API
DBI_TRACE environment variable，在gdb环境下调试mysqld，环境变量
DBI_USER environment variable，环境变量
DBUG package，DBUG软件包
DEALLOCATE PREPARE，用于预处理语句的SQL语法
debug

mysql option，选项

debug-info

mysql option，选项

debugging

client，调试MySQL客户端
server，调试MySQL服务器

DEC data type，数值类型概述
decimal arithmetic，精度数学
DECIMAL data type，数值类型概述，精度数学
decimal point，列类型
DECLARE，DECLARE语句
DECODE()，加密函数
decode_bits myisamchk variable，用于myisamchk的一般选项
DEFAULT

constraint，对无效数据的约束

default

privileges，使初始MySQL账户安全

default hostname，与MySQL服务器连接
default installation location，安装布局
default options，使用选项文件
DEFAULT value clause，CREATE TABLE语法
default values，MySQL设计局限与折衷，CREATE TABLE语法，INSERT语法

BLOB and TEXT columns，BLOB和TEXT类型
implicit，CREATE TABLE语法
suppression，对无效数据的约束

DEFAULT()，其他函数
default-character-set

mysql option，选项

defaults

embedded，与嵌入式服务器一起使用的选项

DEGREES()，数学函数
DELAYED，INSERT DELAYED语法
delayed_insert_limit，INSERT DELAYED语法
DELETE，DELETE语法
deleting

database，DROP DATABASE语法
foreign key，ALTER TABLE语法，FOREIGN KEY约束
function，CREATE FUNCTION/DROP FUNCTION语法
index，ALTER TABLE语法，DROP INDEX语法
primary key，ALTER TABLE语法
rows，从相关表删除行
schema，DROP DATABASE语法
table，DROP TABLE语法
user，从MySQL删除用户账户，DROP USER语法
users，从MySQL删除用户账户，DROP USER语法

deletion

mysql.sock，如何保护或更改MySQL套接字文件/tmp/mysql.sock

Delphi program，已知的能与MyODBC一起工作的程序
derived tables，FROM子句中的子查询
DESC，DESCRIBE语法（获取有关列的信息）
DESCRIBE，获得数据库和表的信息，DESCRIBE语法（获取有关列的信息）
design

choices，设计选择
issues，MySQL中的已知事宜
limitations，MySQL设计局限与折衷

DES_DECRYPT()，加密函数
DES_ENCRYPT()，加密函数

developers

list of，感谢

development source tree，从开发源码树安装
Difference()，Spatial 操作符
digits，列类型
Dimension()，通用几何函数
directory structure

default，安装布局

DISCARD TABLESPACE，ALTER TABLE语法，使用按表的表空间
disconnecting

from the server，连接与断开服务器

Disjoint()，测试几何类之间空间关系的函数
disk full，MySQL处理磁盘满的方式
disk issues，磁盘事宜
disks

splitting data across，在Windows平台上使用关于数据库的符号链接

display size，列类型
display triggers，SHOW TRIGGERS语法
display width，列类型
displaying

information

Cardinality，SHOW INDEX语法
Collation，SHOW INDEX语法
SHOW，SHOW语法，SHOW COLUMNS语法，SHOW INDEX语法，SHOW OPEN TABLES语法，SHOW TABLES语法

table status，SHOW TABLE STATUS语法

Distance()，测试几何类之间空间关系的函数
DISTINCT，选择特殊列，MySQL如何优化DISTINCT，GROUP BY（聚合）函数，SELECT语法
DISTINCTROW，SELECT语法
DIV，算术操作符
division (/)，算术操作符
DNS，MySQL如何使用DNS
DO，DO语法
DocBook XML

documentation source format，关于本手册

Documenters

list of，资料员和译员

DOUBLE data type，数值类型概述
DOUBLE PRECISION data type，数值类型概述
double quote (\")，字符串

downgrading，降级MySQL
downloading，怎样获得MySQL
DROP DATABASE，DROP DATABASE语法
DROP FOREIGN KEY，ALTER TABLE语法，FOREIGN KEY约束
DROP FUNCTION，DROP PROCEDURE和DROP FUNCTION，CREATE FUNCTION/DROP FUNCTION语法
DROP INDEX，ALTER TABLE语法，DROP INDEX语法
DROP PRIMARY KEY，ALTER TABLE语法
DROP PROCEDURE，DROP PROCEDURE和DROP FUNCTION
DROP SCHEMA，DROP DATABASE语法
DROP TABLE，DROP TABLE语法
DROP USER，DROP USER语法
DROP VIEW，DROP VIEW语法
dropping

    user，从MySQL删除用户账户，DROP USER语法

DUAL，SELECT语法
DUMPFILE，SELECT语法
dynamic table characteristics，动态表特性

E

Eiffel Wrapper，MySQL Eiffel Wrapper
ELT()，字符串函数
email lists，The MySQL邮件列表
embedded MySQL server library，libmysqld，嵌入式MySQL服务器库
ENCODE()，加密函数
ENCRYPT()，加密函数
encryption functions，加密函数
END，BEGIN ... END复合语句
EndPoint()，LineString函数
entering

    queries，输入查询

ENUM

    size，列类型存储需求

ENUM data type，字符串类型概述，ENUM类型
Envelope()，通用几何函数
environment variable

    CC，典型配置选项，处理MySQL编译问题
    CFLAGS，典型配置选项，处理MySQL编译问题
    CXX，典型配置选项，处理MySQL编译问题
    CXXFLAGS，典型配置选项，处理MySQL编译问题
    HOME，选项
    LD_RUN_PATH，Linux源码分发版说明，Solaris注意事项
    MYSQL_DEBUG，客户端脚本和实用工具概述
    MYSQL_HISTFILE，选项
    MYSQL_HOST，与MySQL服务器连接
    MYSQL_PWD，与MySQL服务器连接，客户端脚本和实用工具概述
    MYSQL_TCP_PORT，在Unix中运行多个服务器，在多服务器环境中使用客户端程序，客户端脚本和实用

工具概述
MYSQL_UNIX_PORT，在Unix中运行多个服务器，在多服务器环境中使用客户端程序，客户端脚本和实用工具概述
PATH，在其它类Unix系统中安装MySQL，调用MySQL程序
USER，与MySQL服务器连接

Environment variable

CC，环境变量
CFLAGS，环境变量
CXX，处理MySQL编译问题，环境变量
CXXFLAGS，环境变量
DBI_TRACE，在gdb环境下调试mysqld，环境变量
DBI_USER，环境变量
HOME，环境变量
LD_LIBRARY_PATH，使用Perl DBI/DBD接口的问题
LD_RUN_PATH，使用Perl DBI/DBD接口的问题，环境变量
MYSQL_DEBUG，调试MySQL客户端，环境变量
MYSQL_HISTFILE，环境变量
MYSQL_HOST，环境变量
MYSQL_PS1，环境变量
MYSQL_PWD，环境变量
MYSQL_TCP_PORT，环境变量
MYSQL_UNIX_PORT，与运行mysql_install_db有关的问题，环境变量
PATH，环境变量
TMPDIR，与运行mysql_install_db有关的问题，环境变量
TZ，时区问题，环境变量
UMASK，与文件许可有关的问题，环境变量
UMASK_DIR，与文件许可有关的问题，环境变量
USER，环境变量

Environment variables

CXX，处理MySQL编译问题

environment variables，用环境变量指定选项，拒绝访问错误的原因，客户端脚本和实用工具概述

list of，环境变量

equal (=)，比较函数和操作符
Equals()，测试几何类之间空间关系的函数
error messages

can't find file，与文件许可有关的问题
languages，设置错误消息语言

errors

access denied，拒绝访问
checking tables for，如何检查MyISAM表的错误
common，问题和常见错误
directory checksum，Solaris注意事项
handling for UDFs，UDF返回值和错误处理
known，MySQL中的已知事宜
linking，与MySQL客户端库的链接问题

list of，使用MySQL程序时的常见错误
reporting，一般信息，MySQL邮件列表，如何通报缺陷和问题

ERROR_FOR_DIVISION_BY_ZERO SQL mode，SQL服务器模式
escape (\\)，字符串
escape characters，文字值
estimating

query performance，估计查询性能

exact-value literals，精度数学
example

mysqld_multi option，mysqld_multi：管理多个MySQL服务器的程序

EXAMPLE storage engine，存储引擎和表类型，EXAMPLE存储引擎
EXAMPLE table type，存储引擎和表类型，EXAMPLE存储引擎
examples

compressed tables，myisampack：生成压缩、只读MyISAM表
myisamchk output，获取关于表的信息
queries，常用查询的例子

Excel，已知的能与MyODBC一起工作的程序
execute

mysql option，选项

EXECUTE，用于预处理语句的SQL语法
EXP()，数学函数
EXPLAIN，EXPLAIN语法（获取SELECT相关信息）
EXPORT_SET()，字符串函数
expression aliases，具有隐含字段的GROUP BY，SELECT语法
expressions

extended，模式匹配

extensions

to standard SQL，MySQL标准的兼容性

ExteriorRing()，Polygon函数
EXTRACT()，日期和时间函数
extracting

dates，日期计算

F

FALSE，数值

testing for，比较函数和操作符

fatal signal 11，处理MySQL编译问题
features of MySQL，MySQL的的主要特性

FEDERATED storage engine，存储引擎和表类型，FEDERATED存储引擎
FEDERATED table type，存储引擎和表类型，FEDERATED存储引擎
FETCH，光标FETCH语句
field

changing，ALTER TABLE语法

FIELD()，字符串函数
FILE，字符串函数
files

binary log，二进制日志
config.cache，处理MySQL编译问题
error messages，设置错误消息语言
log，典型配置选项，日志文件维护
my.cnf，复制特性和已知问题
not found message，与文件许可有关的问题
permissions，与文件许可有关的问题
query log，通用查询日志
repairing，myisamchk的修复选项
script，在批处理模式下使用mysql
size limits，MySQL表最大能达到多少
slow query log，慢速查询日志
tmp，与运行mysql_install_db有关的问题
update log (obsolete)，二进制日志

filesort optimization，MySQL如何优化ORDER BY
FIND_IN_SET()，字符串函数
FIXED data type，数值类型概述
fixed-point arithmetic，精度数学
FLOAT data type，数值类型概述
floating-point number，数值类型概述
floats，数值
FLOOR()，数学函数
FLUSH，FLUSH Syntax
flush tables，mysqladmin：用于管理MySQL服务器的客户端
FOR UPDATE，SELECT语法
force

mysql option，选项

FORCE INDEX，SELECT语法，JOIN语法，与优化器有关的事宜
FORCE KEY，SELECT语法，JOIN语法
foreign key

constraint，PRIMARY KEY和UNIQUE索引约束
deleting，ALTER TABLE语法，FOREIGN KEY约束

foreign keys，Foreign Keys，使用外键，ALTER TABLE语法
FORMAT()，其他函数
Forums，MySQL论坛上的MySQL社区支持
FOUND_ROWS()，信息函数
FreeBSD troubleshooting，处理MySQL编译问题
FROM，SELECT语法
FROM_DAYS()，日期和时间函数

FROM_UNIXTIME()，日期和时间函数
ft_max_word_len myisamchk variable，用于myisamchk的一般选项
ft_min_word_len myisamchk variable，用于myisamchk的一般选项
ft_stopword_file myisamchk variable，用于myisamchk的一般选项
full disk，MySQL处理磁盘满的方式
full-text search，全文搜索功能
FULLTEXT，全文搜索功能
fulltext

stopword list，微调MySQL全文搜索

function

deleting，CREATE FUNCTION/DROP FUNCTION语法

functions，函数和操作符

arithmetic，位函数
bit，位函数
C API，C API函数概述
C Prepared statement API，C API预处理语句函数概述
cast，Cast函数和操作符
control flow，控制流程函数
date and time，日期和时间函数
encryption，加密函数
GROUP BY，GROUP BY（聚合）函数
grouping，圆括号
information，信息函数
mathematical，数学函数
miscellaneous，其他函数
native

adding，添加新的固有函数

new，为MySQL添加新函数
string，字符串函数
string comparison，字符串比较函数
user-defined，为MySQL添加新函数

adding，添加新的自定义函数

Functions

user-defined，CREATE FUNCTION/DROP FUNCTION语法

functions for SELECT and WHERE clauses，函数和操作符
Future development of MySQL簇，MySQL簇发展的重要历程

G

gcc，典型配置选项
gdb

using，在gdb环境下调试mysqld

general information，一般信息
General Public License，MySQL数据库管理系统概述
geographic feature，前言
GeomCollFromText()，使用WKT函数创建Geometry（几何）值
GeomCollFromWKB()，使用WKB函数创建Geometry（几何）值
geometry，前言
GEOMETRY data type，MySQL空间数据类型
GEOMETRYCOLLECTION data type，MySQL空间数据类型
GeometryCollection()，使用MySQL函数创建几何值
GeometryCollectionFromText()，使用WKT函数创建Geometry（几何）值
GeometryCollectionFromWKB()，使用WKB函数创建Geometry（几何）值
GeometryFromText()，使用WKT函数创建Geometry（几何）值
GeometryFromWKB()，使用WKB函数创建Geometry（几何）值
GeometryN()，GeometryCollection函数
GeometryType()，通用几何函数
GeomFromText()，使用WKT函数创建Geometry（几何）值，Geometry格式转换函数
GeomFromWKB()，使用WKB函数创建Geometry（几何）值，Geometry格式转换函数
geospatial feature，前言
getting MySQL，怎样获得MySQL
GET_FORMAT()，日期和时间函数
GET_LOCK()，其他函数
GIS，MySQL中的空间扩展，前言
GLength()，LineString函数，MultiLineString函数
global privileges，GRANT和REVOKE语法
goals of MySQL，MySQL数据库管理系统概述
GPL

    General Public License，GNU通用公共许可
    GNU通用公共许可，GNU通用公共许可
    MySQL FLOSS许可例外，MySQL FLOSS许可例外

GRANT，GRANT和REVOKE语法
GRANT statement，向MySQL增加新用户账户，SSL GRANT 选项
grant tables，访问控制 ，阶段2：请求核实

    re-creating，Unix下安装后的过程
    sorting，访问控制 ，阶段1：连接核实，访问控制 ，阶段2：请求核实
    upgrading，升级授权表

granting

    privileges，GRANT和REVOKE语法

GRANTS，SHOW GRANTS语法
greater than (>)，比较函数和操作符
greater than or equal (>=)，比较函数和操作符
GREATEST()，比较函数和操作符
GROUP BY，MySQL如何优化GROUP BY

    aliases in，具有隐含字段的GROUP BY
    extensions to standard SQL，具有隐含字段的GROUP BY，SELECT语法

GROUP BY functions，GROUP BY（聚合）函数
grouping

expressions，圆括号

GROUP_CONCAT()，GROUP BY（聚合）函数

H

HANDLER，HANDLER语法
Handlers，DECLARE Handlers
handling

errors，UDF返回值和错误处理

hash partitioning，HASH分区
hash partitions，managing，HASH和KEY分区的管理
hash partitions，splitting and merging，HASH和KEY分区的管理
HAVING，SELECT语法
HEAP storage engine，存储引擎和表类型，MEMORY (HEAP)存储引擎
HEAP table type，存储引擎和表类型，MEMORY (HEAP)存储引擎
help

mysql option，选项
mysqld_multi option，mysqld_multi：管理多个MySQL服务器的程序

HEX()，字符串函数
hexadecimal values，十六进制值
HIGH_NOT_PRECEDENCE SQL mode，SQL服务器模式
HIGH_PRIORITY，SELECT语法
hints，MySQL对标准SQL的扩展，SELECT语法，JOIN语法

index，SELECT语法，JOIN语法

history of MySQL，MySQL的历史
HOME environment variable，选项，环境变量
host

mysql option，选项

host table，访问控制，阶段2：请求核实

sorting，访问控制，阶段2：请求核实

host.frm

problems finding，Unix下安装后的过程

hostname

default，与MySQL服务器连接

hostname caching，MySQL如何使用DNS
HOUR()，日期和时间函数
html

mysql option，选项

I

ID

unique，如何获得上次插入行的唯一ID

identifiers，数据库、表、索引、列和别名

case sensitivity，识别符大小写敏感性
quoting，数据库、表、索引、列和别名

IF，IF语句
IF()，控制流程函数
IFNULL()，控制流程函数
IGNORE INDEX，SELECT语法，JOIN语法
IGNORE KEY，SELECT语法，JOIN语法
ignore-space

mysql option，选项

IGNORE_SPACE SQL mode，SQL服务器模式
implicit default values，CREATE TABLE语法
IMPORT TABLESPACE，ALTER TABLE语法，使用按表的表空间
IN，比较函数和操作符
increasing

performance，复制FAQ

increasing with replication

speed，MySQL中的复制

index

deleting，ALTER TABLE语法，DROP INDEX语法

index hints，SELECT语法，JOIN语法
indexes，CREATE INDEX语法

and BLOB columns，列索引，CREATE TABLE语法
and IS NULL，MySQL如何使用索引
and LIKE，MySQL如何使用索引
and NULL values，CREATE TABLE语法
and TEXT columns，列索引，CREATE TABLE语法
assigning to key cache，CACHE INDEX语法
block size，服务器系统变量
columns，列索引
leftmost prefix of，MySQL如何使用索引
multi-column，多列索引
multiple-part，CREATE INDEX语法
names，数据库、表、索引、列和别名
use of，MySQL如何使用索引

INET_ATON()，其他函数
INET_NTOA()，其他函数

information about partitions，obtaining，获取关于分区的信息
information functions，信息函数
INFORMATION_SCHEMA，INFORMATION_SCHEMA信息数据库，INFORMATION_SCHEMA表
INNER JOIN，JOIN语法
InnoDB，InnoDB概述

Solaris 10 x86_64 issues，Solaris注意事项

InnoDB storage engine，存储引擎和表类型，InnoDB存储引擎
InnoDB table type，存储引擎和表类型，InnoDB存储引擎
InnoDB tables，事务和原子操作
INSERT，INSERT语句的速度，INSERT语法
INSERT ... SELECT，INSERT语法
INSERT DELAYED，INSERT DELAYED语法
INSERT statement

grant privileges，向MySQL增加新用户账户

INSERT()，字符串函数
inserting

speed of，INSERT语句的速度

installation layouts，安装布局
installation overview，使用源码分发版安装MySQL
installing

binary distribution，在其它类Unix系统中安装MySQL
Linux RPM packages，在Linux下安装MySQL
Mac OS X PKG packages,在Mac OS X中安装MySQL
overview，安装MySQL
Perl，Perl安装注意事项
Perl on Windows，在Windows下安装ActiveState Perl
source distribution，使用源码分发版安装MySQL
user-defined functions，编译和安装用户定义的函数

INSTR()，字符串函数
INT data type，数值类型概述
integer arithmetic，精度数学
INTEGER data type，数值类型概述
integers，数值
InteriorRingN()，Polygon函数
internal compiler errors，处理MySQL编译问题
internal locking，锁定方法
internals，MySQL内部控件
Internet Relay Chat，IRC（在线聊天系统）上的MySQL社区支持
Intersection()，Spatial 操作符
Intersects()，测试几何类之间空间关系的函数
INTERVAL()，比较函数和操作符
introducer

string literal，字符串，字符串文字字符集和校对

invalid data

constraint，对无效数据的约束

IRC，IRC（在线聊天系统）上的MySQL社区支持
IS boolean_value，比较函数和操作符
IS NOT boolean_value，比较函数和操作符
IS NOT NULL，比较函数和操作符
IS NULL，MySQL如何优化IS NULL，比较函数和操作符

and indexes，MySQL如何使用索引

ISAM storage engine，存储引擎和表类型
ISAM table type，存储引擎和表类型
IsClosed()，MultiLineString函数
IsEmpty()，通用几何函数
ISNULL()，比较函数和操作符
ISOLATION LEVEL，SET TRANSACTION语法
IsRing()，LineString函数
IsSimple()，通用几何函数
IS_FREE_LOCK()，其他函数
IS_USED_LOCK()，其他函数
ITERATE，ITERATE语句

J

JOIN，JOIN语法

K

Key cache

MyISAM，MyISAM键高速缓冲

key cache

assigning indexes to，CACHE INDEX语法

key partitioning，KEY分区
key partitions，managing，HASH和KEY分区的管理
key partitions，splitting and merging，HASH和KEY分区的管理
key space

MyISAM，键所需的空间

keys，列索引

foreign，Foreign Keys，使用外键
multi-column，多列索引
searching on two，根据两个键搜索

keywords，MySQL中保留字的处理
key_buffer_size myisamchk variable，用于myisamchk的一般选项
KEY_COLUMN_USAGE

INFORMATION_SCHEMA table，INFORMATION_SCHEMA KEY_COLUMN_USAGE表

KILL，KILL语法
known errors，MySQL中的已知事宜

L

language support，设置错误消息语言
last row

unique ID，如何获得上次插入行的唯一ID

LAST_DAY()，日期和时间函数
LAST_INSERT_ID()，事务和原子操作，INSERT语法
LAST_INSERT_ID([expr])，信息函数
layout of installation，安装布局
LCASE()，字符串函数
LD_LIBRARY_PATH environment variable，使用Perl DBI/DBD接口的问题
LD_RUN_PATH environment variable，Linux源码分发版说明，Solaris注意事项，使用Perl DBI/DBD接口的问题，环境变量
LEAST()，比较函数和操作符
LEAVE，LEAVE语句
LEFT JOIN，MySQL如何优化LEFT JOIN和RIGHT JOIN，JOIN语法
LEFT OUTER JOIN，JOIN语法
LEFT()，字符串函数
leftmost prefix of indexes，MySQL如何使用索引
legal names，数据库、表、索引、列和别名
LENGTH()，字符串函数
less than (<)，比较函数和操作符
less than or equal (<=)，比较函数和操作符
libmysqld，libmysqld，嵌入式MySQL服务器库

options，与嵌入式服务器一起使用的选项

libraries

list of，MySQL使用和包含的库

library

mysqlclient，API和库
mysqld，API和库

License，MySQL FLOSS许可例外
LIKE，字符串比较函数

and indexes，MySQL如何使用索引
and wildcards，MySQL如何使用索引

LIMIT，MySQL如何优化LIMIT，信息函数，SELECT语法
limitations

design，MySQL设计局限与折衷
MySQL Limitations，MySQL中的限制
replication，复制特性和已知问题

limits

file-size，MySQL表最大能达到多少
MySQL Limits，limits in MySQL，MySQL中的限制

linear hash partitioning，LINEAR HASH分区
linear key partitioning，KEY分区
linefeed (\n)，字符串
LineFromText()，使用WKT函数创建Geometry（几何）值
LineFromWKB()，使用WKB函数创建Geometry（几何）值
LINESTRING data type，MySQL空间数据类型
LineString()，使用MySQL函数创建几何值
LineStringFromText()，使用WKT函数创建Geometry（几何）值
LineStringFromWKB()，使用WKB函数创建Geometry（几何）值
linking，创建客户端程序

errors，与MySQL客户端库的链接问题
problems，与C API有关的问题
speed，编译和链接怎样影响MySQL的速度

links

symbolic，使用符号链接

Linux

binary distribution，Linux二进制分发版说明
source distribution，Linux源码分发版说明

list partitioning，LIST分区
list partitions，adding and dropping，RANGE和LIST分区的管理
list partitions，managing，RANGE和LIST分区的管理
literals，文字值
LN()，数学函数
LOAD DATA FROM MASTER，LOAD DATA FROM MASTER语法
LOAD DATA INFILE，LOAD DATA INFILE语法，与NULL值有关的问题
LOAD TABLE FROM MASTER，LOAD TABLE tbl_name FROM MASTER语法
loading

tables，将数据装入表中

LOAD_FILE()，字符串函数
local-infile

mysql option，选项

LOCALTIME，日期和时间函数
LOCALTIMESTAMP，日期和时间函数
LOCATE()，字符串函数
LOCK IN SHARE MODE，SELECT语法
LOCK TABLES，LOCK TABLES和UNLOCK TABLES语法
locking，系统因素和启动参数的调节

page-level，锁定方法
row-level，事务和原子操作，锁定方法
table-level，锁定方法

locking methods，锁定方法
log

changes，MySQL变更史
mysqld_multi option，mysqld_multi：管理多个MySQL服务器的程序

log files，典型配置选项

maintaining，日志文件维护
names，数据库备份

Log files，MySQL日志文件
LOG()，数学函数
LOG10()，数学函数
LOG2()，数学函数
logical operators，逻辑操作符
LONG data type，BLOB和TEXT类型
LONGBLOB data type，字符串类型概述
LONGTEXT data type，字符串类型概述
LOOP，LOOP语句
LOWER()，字符串函数
LPAD()，字符串函数
LTRIM()，字符串函数

M

Mac OS X

installation,在Mac OS X中安装MySQL

mailing address

for customer support，如何通报缺陷和问题

mailing list address，一般信息
mailing lists，The MySQL邮件列表

archive location，请教问题或通报缺陷
guidelines，在邮件列表上回答问题的指南

main features of MySQL，MySQL的的主要特性
maintaining

log files，日志文件维护
tables，建立表维护计划

MAKEDATE()，日期和时间函数
MAKETIME()，日期和时间函数
make_binary_distribution，服务器端脚本和实用工具概述
MAKE_SET()，字符串函数
manual

available formats，关于本手册
online location，关于本手册
typographical conventions，本手册采用的惯例

master/slave setup，复制实施概述
MASTER_POS_WAIT()，其他函数，MASTER_POS_WAIT()语法
MATCH ... AGAINST()，全文搜索功能
matching

patterns，模式匹配

math，精度数学
mathematical functions，数学函数
MAX()，GROUP BY（聚合）函数
MAX(DISTINCT)，GROUP BY（聚合）函数
MAXDB SQL mode，SQL服务器模式
maximum memory used，mysqladmin：用于管理MySQL服务器的客户端
maximums

maximum tables per join，联合的限制

max_allowed_packet variable，选项
MAX_CONNECTIONS_PER_HOUR，限制账户资源
max_join_size variable，选项
MAX_QUERIES_PER_HOUR，限制账户资源
MAX_UPDATES_PER_HOUR，限制账户资源
MAX_USER_CONNECTIONS，限制账户资源
MBR，关于几何最小边界矩形（MBR）的关系
MBRContains()，关于几何最小边界矩形（MBR）的关系
MBRDisjoint()，关于几何最小边界矩形（MBR）的关系
MBREqual()，关于几何最小边界矩形（MBR）的关系
MBRIntersects()，关于几何最小边界矩形（MBR）的关系
MBROverlaps()，关于几何最小边界矩形（MBR）的关系
MBRTouches()，关于几何最小边界矩形（MBR）的关系
MBRWithin()，关于几何最小边界矩形（MBR）的关系
MD5()，加密函数
MEDIUMBLOB data type，字符串类型概述
MEDIUMINT data type，数值类型概述
MEDIUMTEXT data type，字符串类型概述
MEMORY storage engine，存储引擎和表类型，MEMORY (HEAP)存储引擎
MEMORY table type，存储引擎和表类型，MEMORY (HEAP)存储引擎
memory usage

myisamchk，myisamchk内存使用

memory use，MySQL如何使用内存，mysqladmin：用于管理MySQL服务器的客户端
MERGE storage engine，存储引擎和表类型，MERGE存储引擎
MERGE table type，存储引擎和表类型，MERGE存储引擎
MERGE tables

defined，MERGE存储引擎

messages

languages，设置错误消息语言

metadata

database，INFORMATION_SCHEMA信息数据库

methods

locking，锁定方法

MICROSECOND()，日期和时间函数
MID()，字符串函数
MIN()，GROUP BY（聚合）函数
MIN(DISTINCT)，GROUP BY（聚合）函数
Minimum Bounding Rectangle，关于几何最小边界矩形（MBR）的关系
minus

unary (-)，算术操作符

MINUTE()，日期和时间函数
mirror sites，怎样获得MySQL
miscellaneous functions，其他函数
MIT-pthreads，MIT-pthreads注意事项
MLineFromText()，使用WKT函数创建Geometry（几何）值
MLineFromWKB()，使用WKB函数创建Geometry（几何）值
MOD (modulo)，数学函数
MOD()，数学函数
modes

batch，在批处理模式下使用mysql

modules

list of，MySQL稳定性

modulo (%)，数学函数
modulo (MOD)，数学函数
monitor

terminal，教程

MONTH()，日期和时间函数
MONTHNAME()，日期和时间函数
MPointFromText()，使用WKT函数创建Geometry（几何）值
MPointFromWKB()，使用WKB函数创建Geometry（几何）值
MPolyFromText()，使用WKT函数创建Geometry（几何）值
MPolyFromWKB()，使用WKB函数创建Geometry（几何）值
mSQL compatibility，字符串比较函数
MSSQL SQL mode，SQL服务器模式
multi-byte character sets，无法初始化字符集
multi-byte characters，多字节字符支持
multi-column indexes，多列索引
MULTILINESTRING data type，MySQL空间数据类型
MultiLineString()，使用MySQL函数创建几何值
MultiLineStringFromText()，使用WKT函数创建Geometry（几何）值
MultiLineStringFromWKB()，使用WKB函数创建Geometry（几何）值
multiple servers，在同一台机器上运行多个MySQL服务器
multiple-part index，CREATE INDEX语法
multiplication (*)，算术操作符
MULTIPOINT data type，MySQL空间数据类型

MultiPoint()，使用MySQL函数创建几何值
MultiPointFromText()，使用WKT函数创建Geometry（几何）值
MultiPointFromWKB()，使用WKB函数创建Geometry（几何）值
MULTIPOLYGON data type，MySQL空间数据类型
MultiPolygon()，使用MySQL函数创建几何值
MultiPolygonFromText()，使用WKT函数创建Geometry（几何）值
MultiPolygonFromWKB()，使用WKB函数创建Geometry（几何）值
My

derivation，MySQL的历史

my.cnf file，复制特性和已知问题
MyISAM

compressed tables，压缩表特性
size，列类型存储需求

MyISAM key cache，MyISAM键高速缓冲
MyISAM storage engine，存储引擎和表类型，MyISAM存储引擎
MyISAM table type，存储引擎和表类型，MyISAM存储引擎
myisamchk，典型配置选项，服务器端脚本和实用工具概述

example output，获取关于表的信息
options，用于myisamchk的一般选项

myisampack，客户端脚本和实用工具概述，压缩表特性
myisam_block_size myisamchk variable，用于myisamchk的一般选项
MyODBC，MySQL Connector/ODBC

reporting problems，如何通报MyODBC问题或缺陷

MySQL

defined，MySQL数据库管理系统概述
introduction，MySQL数据库管理系统概述
pronunciation，MySQL数据库管理系统概述

mysql，客户端脚本和实用工具概述
MySQL AB

defined，MySQL AB概述

MySQL binary distribution，选择要安装的MySQL分发版
MYSQL C type，C API数据类型
MySQL C type，C API预处理语句的数据类型
MySQL簇 in MySQL 5.0 and 5.1，MySQL簇发展的重要历程
mysql command-line options，选项
mysql commands

list of，mysql命令

MySQL Dolphin name，MySQL的历史
MySQL history，MySQL的历史
mysql history file，选项
MySQL mailing lists，MySQL邮件列表

MySQL name，MySQL的历史
mysql prompt command，mysql命令
MySQL source distribution，选择要安装的MySQL分发版
mysql status command，mysql命令
MySQL storage engines，存储引擎和表类型
MySQL table types，存储引擎和表类型
MySQL version，怎样获得MySQL
MySQL++，MySQL C++ API
mysql.server，服务器端脚本和实用工具概述
mysql.sock

    changing location of，典型配置选项
    protection，如何保护或更改MySQL套接字文件/tmp/mysql.sock

MYSQL323 SQL mode，SQL服务器模式
MYSQL40 SQL mode，SQL服务器模式
mysqlaccess，客户端脚本和实用工具概述
mysqladmin，客户端脚本和实用工具概述，CREATE DATABASE语法，DROP DATABASE语法，SHOW STATUS语法，SHOW VARIABLES语法，FLUSH Syntax，KILL语法

    mysqld_multi option，mysqld_multi：管理多个MySQL服务器的程序

mysqladmin command-line options，mysqladmin：用于管理MySQL服务器的客户端
mysqlbinlog，客户端脚本和实用工具概述
mysqlbug，服务器端脚本和实用工具概述
mysqlbug script，如何通报缺陷和问题

    location，一般信息

mysqlcheck，客户端脚本和实用工具概述
mysqlclient library，API和库
mysqld，服务器端脚本和实用工具概述

    mysqld_multi option，mysqld_multi：管理多个MySQL服务器的程序
    starting，如何以普通用户身份运行MySQL

mysqld library，API和库
mysqld options，mysqld命令行选项，调节服务器参数
mysqld server

    buffer sizes，调节服务器参数

mysqld-max，服务器端脚本和实用工具概述，mysqld-max扩展MySQL服务器
mysqldump，将MySQL数据库拷贝到另一台机器，客户端脚本和实用工具概述
mysqld_multi，服务器端脚本和实用工具概述
mysqld_safe，服务器端脚本和实用工具概述
mysqlhotcopy，客户端脚本和实用工具概述
mysqlimport，将MySQL数据库拷贝到另一台机器，客户端脚本和实用工具概述，LOAD DATA INFILE语法
mysqlshow，客户端脚本和实用工具概述
mysqltest

    MySQL测试套件，MySQL测试套件

mysql_affected_rows()，mysql_affected_rows()
mysql_autocommit()．，mysql_autocommit()

MYSQL_BIND C type，C API预处理语句的数据类型
mysql_change_user()，mysql_change_user()
mysql_character_set_name()，mysql_character_set_name()
mysql_close()，mysql_close()
mysql_commit()．，mysql_commit()
mysql_connect()，mysql_connect()
mysql_create_db()，mysql_create_db()
mysql_data_seek()，mysql_data_seek()
MYSQL_DEBUG environment variable，客户端脚本和实用工具概述，调试MySQL客户端，环境变量
mysql_debug()，mysql_debug()
mysql_drop_db()，mysql_drop_db()
mysql_dump_debug_info()，mysql_dump_debug_info()
mysql_eof()，mysql_eof()
mysql_errno()，mysql_errno()
mysql_error()，mysql_error()
mysql_escape_string()，mysql_escape_string()
mysql_fetch_field()，mysql_fetch_field()
mysql_fetch_fields()，mysql_fetch_fields()
mysql_fetch_field_direct()，mysql_fetch_field_direct()
mysql_fetch_lengths()，mysql_fetch_lengths()
mysql_fetch_row()，mysql_fetch_row()
MYSQL_FIELD C type，C API数据类型
mysql_field_count()，mysql_field_count()，mysql_num_fields()
MYSQL_FIELD_OFFSET C type，C API数据类型
mysql_field_seek()，mysql_field_seek()
mysql_field_tell()，mysql_field_tell()
mysql_fix_privilege_tables，服务器端脚本和实用工具概述，拒绝访问错误的原因
mysql_free_result()，mysql_free_result()
mysql_get_character_set_info()，mysql_get_character_set_info()
mysql_get_client_info()，mysql_get_client_info()
mysql_get_client_version()，mysql_get_client_version()
mysql_get_host_info()，mysql_get_host_info()
mysql_get_proto_info()，mysql_get_proto_info()
mysql_get_server_info()，mysql_get_server_info()
mysql_get_server_version()，mysql_get_server_version()
mysql_hex_string()，mysql_hex_string()
MYSQL_HISTFILE environment variable，选项，环境变量
MYSQL_HOST environment variable，与MySQL服务器连接，环境变量
mysql_info()，ALTER TABLE语法，INSERT语法，LOAD DATA INFILE语法，UPDATE语法，mysql_info()
mysql_init()，mysql_init()
mysql_insert_id()，事务和原子操作，INSERT语法，mysql_insert_id()
mysql_install_db，服务器端脚本和实用工具概述
mysql_install_db script，与运行mysql_install_db有关的问题
mysql_kill()，mysql_kill()
mysql_library_end()，mysql_library_end()
mysql_library_init()，mysql_library_init()
mysql_list_dbs()，mysql_list_dbs()
mysql_list_fields()，mysql_list_fields()
mysql_list_processes()，mysql_list_processes()
mysql_list_tables()，mysql_list_tables()
mysql_more_results()．，mysql_more_results()
mysql_next_result()．，mysql_next_result()
mysql_num_fields()，mysql_num_fields()
mysql_num_rows()，mysql_num_rows()

mysql_options(), mysql_options()
mysql_ping(), mysql_ping()
MYSQL_PS1 environment variable, 环境变量
MYSQL_PWD environment variable, 与MySQL服务器连接, 客户端脚本和实用工具概述, 环境变量
mysql_query(), mysql_query(), 使用C API时的常见问题
mysql_real_connect(), mysql_real_connect()
mysql_real_escape_string(), 字符串, mysql_real_escape_string()
mysql_real_query(), mysql_real_query()
mysql_refresh(), mysql_refresh()
mysql_reload(), mysql_reload()
MYSQL_RES C type, C API数据类型
mysql_rollback(), mysql_rollback()
MYSQL_ROW C type, C API数据类型
mysql_row_seek(), mysql_row_seek()
mysql_row_tell(), mysql_row_tell()
mysql_select_db(), mysql_select_db()
mysql_server_end(), mysql_server_end()
mysql_server_init(), mysql_server_init()
mysql_set_character_set(), mysql_set_character_set()
mysql_set_server_option(), mysql_set_server_option()
mysql_shutdown(), mysql_shutdown()
mysql_sqlstate(), mysql_sqlstate()
mysql_ssl_set(), mysql_ssl_set()
mysql_stat(), mysql_stat()
MYSQL_STMT C type, C API预处理语句的数据类型
mysql_stmt_affected_rows(), mysql_stmt_affected_rows()
mysql_stmt_attr_get(), mysql_stmt_attr_get()
mysql_stmt_attr_set(), mysql_stmt_attr_set()
mysql_stmt_bind_param(), mysql_stmt_bind_param()
mysql_stmt_bind_result(), mysql_stmt_bind_result()
mysql_stmt_close(), mysql_stmt_close()
mysql_stmt_data_seek(), mysql_stmt_data_seek()
mysql_stmt_errno(), mysql_stmt_errno()
mysql_stmt_error(), mysql_stmt_error()
mysql_stmt_execute(), mysql_stmt_execute()
mysql_stmt_fetch(), mysql_stmt_fetch()
mysql_stmt_fetch_column(), mysql_stmt_fetch_column()
mysql_stmt_field_count(), mysql_stmt_field_count()
mysql_stmt_free_result(), mysql_stmt_free_result()
mysql_stmt_init(), mysql_stmt_init()
mysql_stmt_insert_id(), mysql_stmt_insert_id()
mysql_stmt_num_rows(), mysql_stmt_num_rows()
mysql_stmt_param_count(), mysql_stmt_param_count()
mysql_stmt_param_metadata(), mysql_stmt_param_metadata()
mysql_stmt_prepare(), mysql_stmt_prepare()
mysql_stmt_reset(), mysql_stmt_reset()
mysql_stmt_result_metadata(), mysql_stmt_result_metadata()
mysql_stmt_row_seek(), mysql_stmt_row_seek()
mysql_stmt_row_tell(), mysql_stmt_row_tell()
mysql_stmt_send_long_data(), mysql_stmt_send_long_data()
mysql_stmt_sqlstate(), mysql_stmt_sqlstate()
mysql_stmt_store_result(), mysql_stmt_store_result()
mysql_store_result(), mysql_store_result(), 使用C API时的常见问题
MYSQL_TCP_PORT environment variable, 在Unix中运行多个服务器, 在多服务器环境中使用客户端程序, 客

户端脚本和实用工具概述，环境变量
mysql_thread_end()，mysql_thread_end()
mysql_thread_id()，mysql_thread_id()
mysql_thread_init()，mysql_thread_init()
mysql_thread_safe()，mysql_thread_safe()
MYSQL_UNIX_PORT environment variable，与运行mysql_install_db有关的问题，在Unix中运行多个服务器，在多服务器环境中使用客户端程序，客户端脚本和实用工具概述，环境变量
mysql_use_result()，mysql_use_result()
mysql_warning_count().，mysql_warning_count()
my_init()，my_init()
my_ulonglong C type，C API数据类型
my_ulonglong values

    printing，C API数据类型

N

named pipes，选择MySQL服务器类型，测试MySQL安装
named-commands

    mysql option，选项

names，数据库、表、索引、列和别名

    case sensitivity，识别符大小写敏感性
    variables，用户变量

NAME_CONST()，其他函数
naming

    releases of MySQL，选择要安装的MySQL版本

NATIONAL CHAR data type，字符串类型概述
native functions

    adding，添加新的固有函数

native thread support，MySQL支持的操作系统
NATURAL LEFT JOIN，JOIN语法
NATURAL LEFT OUTER JOIN，JOIN语法
NATURAL RIGHT JOIN，JOIN语法
NATURAL RIGHT OUTER JOIN，JOIN语法
NCHAR data type，字符串类型概述
negative values，数值
nested queries，Subquery语法
net etiquette，请教问题或通报缺陷，在邮件列表上回答问题的指南
netmask notation

    in mysql.user table，访问控制，阶段1：连接核实

NetWare，在NetWare中安装MySQL
net_buffer_length variable，选项
New features in MySQL簇，MySQL簇发展的重要历程
new procedures

adding，为MySQL添加新步骤

new users

adding，在其它类Unix系统中安装MySQL，源码安装概述

newline (\n)，字符串
no matching rows，解决与不匹配行有关的问题
no-auto-rehash

mysql option，选项

no-beep

mysql option，选项

no-log

mysqld_multi option，mysqld_multi：管理多个MySQL服务器的程序

no-named-commands

mysql option，选项

no-pager

mysql option，选项

no-tee

mysql option，选项

non-delimited strings，DATETIME、DATE和TIMESTAMP类型
Non-transactional tables，非事务表回滚失败
NOT

logical，逻辑操作符

NOT BETWEEN，比较函数和操作符
not equal (!=)，比较函数和操作符
not equal (<>)，比较函数和操作符
NOT IN，比较函数和操作符
NOT LIKE，字符串比较函数
NOT NULL

constraint，对无效数据的约束

NOT REGEXP，字符串比较函数
Novell NetWare，在NetWare中安装MySQL
NOW()，日期和时间函数
NO_AUTO_CREATE_USER SQL mode，SQL服务器模式
NO_AUTO_VALUE_ON_ZERO SQL mode，SQL服务器模式
NO_BACKSLASH_ESCAPES SQL mode，SQL服务器模式
NO_DIR_IN_CREATE SQL mode，SQL服务器模式
NO_FIELD_OPTIONS SQL mode，SQL服务器模式
NO_KEY_OPTIONS SQL mode，SQL服务器模式

NO_TABLE_OPTIONS SQL mode，SQL服务器模式
NO_UNSIGNED_SUBTRACTION SQL mode，SQL服务器模式
NO_ZERO_DATE SQL mode，SQL服务器模式
NO_ZERO_IN_DATE SQL mode，SQL服务器模式
NUL，字符串
NULL，Working with NULL值，与NULL值有关的问题

testing for null，比较函数和操作符，控制流程函数

NULL value，Working with NULL值，NULL值
NULL values

and AUTO_INCREMENT columns，与NULL值有关的问题
and indexes，CREATE TABLE语法
and TIMESTAMP columns，与NULL值有关的问题
vs. empty values，与NULL值有关的问题

NULLIF()，控制流程函数
numbers，数值
NUMERIC data type，数值类型概述
numeric types，列类型存储需求
NumGeometries()，GeometryCollection函数
NumInteriorRings()，Polygon函数
NumPoints()，LineString函数

O

OCT()，字符串函数
OCTET_LENGTH()，字符串函数
ODBC，MySQL Connector/ODBC
ODBC compatibility，识别符限制条件，数值类型概述，比较函数和操作符，CREATE TABLE语法，JOIN语法
odbcadmin program，已知的能与MyODBC一起工作的程序
OLAP，GROUP BY修改程序
OLD_PASSWORD()，加密函数
ON DUPLICATE KEY，INSERT语法
one-database

mysql option，选项

online location of manual，关于本手册
ONLY_FULL_GROUP_BY SQL mode，SQL服务器模式
OPEN，光标OPEN语句
Open Source

defined，MySQL数据库管理系统概述

open tables，MySQL如何计算打开的表，mysqladmin：用于管理MySQL服务器的客户端
OpenGIS，前言
opening

tables，MySQL如何打开和关闭表

opens，mysqladmin：用于管理MySQL服务器的客户端
OpenSSL，使用安全连接，Using yaSSL
open_files_limit variable，mysqlbinlog：用于处理二进制日志文件的实用工具

operating systems

file-size limits，MySQL表最大能达到多少
supported，MySQL支持的操作系统
Windows versus Unix，Windows版MySQL同Unix版MySQL对比

operations

arithmetic，算术操作符

operators，函数和操作符

cast，算术操作符，Cast函数和操作符
logical，逻辑操作符

optimization

tips，其它优化技巧

optimizations，MySQL怎样优化WHERE子句，索引合并优化
OPTIMIZE TABLE，OPTIMIZE TABLE语法
optimizer

controlling，控制查询优化器的性能

optimizing

DISTINCT，MySQL如何优化DISTINCT
filesort，MySQL如何优化ORDER BY
GROUP BY，MySQL如何优化GROUP BY
LEFT JOIN，MySQL如何优化LEFT JOIN和RIGHT JOIN
LIMIT，MySQL如何优化LIMIT
tables，Table 优化

option files，使用选项文件，拒绝访问错误的原因
options

command-line，mysqld命令行选项

mysql，选项
mysqladmin，mysqladmin：用于管理MySQL服务器的客户端

configure，典型配置选项
embedded server，与嵌入式服务器一起使用的选项
libmysqld，与嵌入式服务器一起使用的选项
myisamchk，用于myisamchk的一般选项
provided by MySQL，教程
replication，复制特性和已知问题

OR，根据两个键搜索，索引合并优化

bitwise，位函数
logical，逻辑操作符

OR 索引 Merge optimization，索引合并优化
Oracle compatibility，MySQL对标准SQL的扩展，GROUP BY（聚合）函数，DESCRIBE语法（获取有关列的信

息）
ORACLE SQL mode，SQL服务器模式
ORD()，字符串函数
ORDER BY，分类行，ALTER TABLE语法，SELECT语法

aliases in，具有隐含字段的GROUP BY

OUTFILE，SELECT语法
Overlaps()，测试几何类之间空间关系的函数
overview，一般信息

P

packages

list of，支持MySQL的软件包

page-level locking，锁定方法
pager

mysql option，选项

parameters

server，调节服务器参数

parentheses (and)，圆括号
PARTITION，分区
partition management，分区管理
partitioning，分区
partitioning by hash，HASH分区
partitioning by key，KEY分区
partitioning by linear hash，LINEAR HASH分区
partitioning by linear key，KEY分区
partitioning by list，LIST分区
partitioning by range，RANGE分区
partitioning concepts，MySQL中的分区概述
partitioning information commands，获取关于分区的信息
partitioning support，MySQL中的分区概述
partitioning types，分区类型
partitioning，advantages，MySQL中的分区概述
partitioning，enabling，MySQL中的分区概述
partitions，adding and dropping，分区管理
partitions，analyzing，分区维护
partitions，checking，分区维护
partitions，managing，分区管理
partitions，modifying，分区管理
partitions，optimizing，分区维护
partitions，repairing，分区维护
partitions，splitting and merging，分区管理
password

mysql option，选项
mysqld_multi option，mysqld_multi：管理多个MySQL服务器的程序
root user，使初始MySQL账户安全

password encryption

reversibility of，加密函数

PASSWORD()，访问控制，阶段1：连接核实，设置账户密码，加密函数，忽略用户
passwords

for users，MySQL用户名和密码
forgotten，如何复位根用户密码
lost，如何复位根用户密码
resetting，如何复位根用户密码
security，权限系统的作用
setting，设置账户密码，GRANT和REVOKE语法，SET PASSWORD语法

PATH environment variable，在其它类Unix系统中安装MySQL，调用MySQL程序，环境变量
pattern matching，模式匹配
performance

benchmarks，使用自己的基准
disk issues，磁盘事宜
estimating，估计查询性能
improving，复制FAQ，使你的数据尽可能小

PERIOD_ADD()，日期和时间函数
PERIOD_DIFF()，日期和时间函数
Perl

installing，Perl安装注意事项
installing on Windows，在Windows下安装ActiveState Perl

Perl API，MySQL Perl API
Perl DBI/DBD

installation problems，使用Perl DBI/DBD接口的问题

permission checks

effect on speed，优化SELECT语句和其它查询

perror，客户端脚本和实用工具概述
PHP API，MySQL PHP API
PI()，数学函数
PIPES_AS_CONCAT SQL mode，SQL服务器模式
POINT data type，MySQL空间数据类型
point in time recovery，自动恢复
Point()，使用MySQL函数创建几何值
PointFromText()，使用WKT函数创建Geometry（几何）值
PointFromWKB()，使用WKB函数创建Geometry（几何）值
PointN()，LineString函数
PointOnSurface()，MultiPolygon函数
PolyFromText()，使用WKT函数创建Geometry（几何）值
PolyFromWKB()，使用WKB函数创建Geometry（几何）值
POLYGON data type，MySQL空间数据类型
Polygon()，使用MySQL函数创建几何值

PolygonFromText()，使用WKT函数创建Geometry（几何）值
PolygonFromWKB()，使用WKB函数创建Geometry（几何）值
port

mysql option，选项

portability，为可移植性设计应用程序

types，使用来自其他数据库引擎的列类型

porting

to other systems，移植到其他系统

POSITION()，字符串函数
post-install

multiple servers，在同一台机器上运行多个MySQL服务器

post-installation

setup and testing，安装后的设置和测试

PostgreSQL compatibility，MySQL对标准SQL的扩展
POSTGRESQL SQL mode，SQL服务器模式
POW()，数学函数
POWER()，数学函数
precision

arithmetic，精度数学

precision math，精度数学
PREPARE，用于预处理语句的SQL语法

XA transactions，XA事务SQL语法

PRIMARY KEY，ALTER TABLE语法，CREATE TABLE语法

constraint，PRIMARY KEY和UNIQUE索引约束

primary key

deleting，ALTER TABLE语法

privilege

changes，访问控制，阶段2：请求核实

privilege information

location，MySQL提供的权限

privilege system，权限系统的作用

described，权限系统工作原理

privileges

access，MySQL访问权限系统
adding，向MySQL增加新用户账户
default，使初始MySQL账户安全
deleting，从MySQL删除用户账户，DROP USER语法
display，SHOW GRANTS语法
dropping，从MySQL删除用户账户，DROP USER语法
granting，GRANT和REVOKE语法
revoking，GRANT和REVOKE语法

problems

access denied errors，拒绝访问
common errors，问题和常见错误
compiling，处理MySQL编译问题
DATE columns，使用DATE列方面的问题
date values，DATETIME、DATE和TIMESTAMP类型
installing on IBM-AIX，IBM-AIX注意事项
installing on Solaris，Solaris注意事项
installing Perl，使用Perl DBI/DBD接口的问题
linking，与MySQL客户端库的链接问题
ODBC，如何通报MyODBC问题或缺陷
reporting，如何通报缺陷和问题
starting the server，启动MySQL服务器以及其故障诊断和排除
table locking，表锁定事宜
time zone，时区问题

PROCEDURE，SELECT语法
procedures

adding，为MySQL添加新步骤
stored，存储程序和触发程序，存储程序和函数

process support，MySQL支持的操作系统
processes

display，SHOW PROCESSLIST语法

processing

arguments，UDF参量处理

PROCESSLIST，SHOW PROCESSLIST语法
program variables

setting，使用选项设置程序变量

programs

client，创建客户端程序
crash-me，为可移植性设计应用程序
list of，服务器端脚本和实用工具概述

prompt

mysql option，选项

prompts

meanings，输入查询

pronunciation

MySQL，MySQL数据库管理系统概述

protocol

mysql option，选项

PURGE MASTER LOGS，PURGE MASTER LOGS语法
Python API，MySQL Python API

Q

QUARTER()，日期和时间函数
queries

entering，输入查询
estimating performance，估计查询性能
examples，常用查询的例子
speed of，优化SELECT语句和其它查询
Twin Studies project，孪生项目的查询

Query Cache，MySQL查询高速缓冲
query log，通用查询日志
questions，mysqladmin：用于管理MySQL服务器的客户端

answering，在邮件列表上回答问题的指南

quick

mysql option，选项

QUOTE()，字符串函数
quotes

in strings，字符串

quoting，字符串
quoting binary data，字符串
quoting of identifiers，数据库、表、索引、列和别名

R

RADIANS()，数学函数
RAND()，数学函数
range partitioning，RANGE分区
range partitions，adding and dropping，RANGE和LIST分区的管理
range partitions，managing，RANGE和LIST分区的管理
raw

mysql option

，选项

re-creating

grant tables，Unix下安装后的过程

read_buffer_size myisamchk variable，用于myisamchk的一般选项
REAL data type，数值类型概述
REAL_AS_FLOAT SQL mode，SQL服务器模式
reconfiguring，处理MySQL编译问题
reconnect

mysql option，选项

RECOVER

XA transactions，XA事务SQL语法

recovery

from crash，将myisamchk用于崩溃恢复
point in time，自动恢复

reducing

data size，使你的数据尽可能小

references，ALTER TABLE语法
ref_or_null，MySQL如何优化IS NULL
regex，MySQL正则表达式
REGEXP，字符串比较函数
regular expression syntax

described，MySQL正则表达式

Related()，测试几何类之间空间关系的函数
relational databases

defined，MySQL数据库管理系统概述

release numbers，选择要安装的MySQL分发版
RELEASE SAVEPOINT，SAVEPOINT和ROLLBACK TO SAVEPOINT语法
releases

naming scheme，选择要安装的MySQL版本
testing，选择要安装的MySQL版本
updating，如何发布更新以及何时发布更新

RELEASE_LOCK()，其他函数
RENAME TABLE，RENAME TABLE语法
RENAME USER，RENAME USER语法
renaming user accounts，RENAME USER语法
reordering

columns，如何更改表中的列顺序

repair options

myisamchk，myisamchk的修复选项

REPAIR TABLE，REPAIR TABLE语法
repairing

tables，如何修复表

REPEAT，REPEAT语句
REPEAT()，字符串函数
replace，客户端脚本和实用工具概述
REPLACE，REPLACE语法
REPLACE ... SELECT，INSERT语法
REPLACE()，字符串函数
replication，MySQL中的复制
replication limitations，复制特性和已知问题
replication masters

commands，用于控制主服务器的SQL语句

replication options，复制特性和已知问题
replication slaves

commands，用于控制从服务器的SQL语句

reporting

bugs，如何通报缺陷和问题
Connector/ODBC problems，如何通报MyODBC问题或缺陷
errors，一般信息，MySQL邮件列表
MyODBC problems，如何通报MyODBC问题或缺陷

REQUIRE GRANT option，SSL GRANT 选项，GRANT和REVOKE语法
reserved words

exceptions，MySQL中保留字的处理

RESET MASTER，RESET MASTER语法
RESET SLAVE，RESET SLAVE语法
restarting

the server，Unix下安装后的过程

RESTORE TABLE，RESTORE TABLE语法
restrictions

server-side cursors，对服务器端光标的限制
stored routines，对存储子程序和触发程序的限制
subqueries，对子查询的限制
triggers，对存储子程序和触发程序的限制
views，对视图的限制

retrieving

data from tables，从表检索信息

return (\r)，字符串
return values

UDFs，UDF返回值和错误处理

REVERSE()，字符串函数
REVOKE，GRANT和REVOKE语法
revoking

privileges，GRANT和REVOKE语法

RIGHT JOIN，JOIN语法
RIGHT OUTER JOIN，JOIN语法
RIGHT()，字符串函数
RLIKE，字符串比较函数
ROLLBACK，事务和原子操作，START TRANSACTION，COMMIT和ROLLBACK语法

XA transactions，XA事务SQL语法

ROLLBACK TO SAVEPOINT，SAVEPOINT和ROLLBACK TO SAVEPOINT语法
ROLLUP，GROUP BY修改程序
root password，使初始MySQL账户安全
root user

password resetting，如何复位根用户密码

ROUND()，数学函数
rounding，精度数学
rounding errors，数值类型概述
ROUTINES

INFORMATION_SCHEMA table，INFORMATION_SCHEMA ROUTINES表

row-level locking，锁定方法
rows

counting，计数行
deleting，从相关表删除行
locking，事务和原子操作
matching problems，解决与不匹配行有关的问题
selecting，选择特殊行
sorting，分类行

ROW_COUNT()，信息函数
RPAD()，字符串函数
RPM file，在Linux下安装MySQL
RPM Package Manager，在Linux下安装MySQL
RTRIM()，字符串函数
RTS-threads，关于RTS线程的注释
running

ANSI mode，在ANSI模式下运行MySQL
batch mode，在批处理模式下使用mysql
multiple servers，在同一台机器上运行多个MySQL服务器
queries，输入查询

running configure after prior invocation，处理MySQL编译问题

S

safe-updates

mysql option，选项

safe-updates option，使用--safe-updates选项
Sakila，MySQL的历史
SAVEPOINT，SAVEPOINT和ROLLBACK TO SAVEPOINT语法
scale

arithmetic，精度数学

schema

altering，ALTER DATABASE语法
creating，CREATE DATABASE语法
deleting，DROP DATABASE语法

SCHEMA()，信息函数
SCHEMATA

INFORMATION_SCHEMA table，INFORMATION_SCHEMA SCHEMATA表

SCHEMA_PRIVILEGES

INFORMATION_SCHEMA table，INFORMATION_SCHEMA SCHEMA_PRIVILEGES表

script files，在批处理模式下使用mysql
scripts

mysqlbug，如何通报缺陷和问题
mysql_install_db，与运行mysql_install_db有关的问题

searching

and case sensitivity，搜索中的大小写敏感性
full-text，全文搜索功能
MySQL Web pages，请教问题或通报缺陷
two keys，根据两个键搜索

SECOND()，日期和时间函数
secure-auth

mysql option，选项

security

against attackers，使MySQL在攻击者面前保持安全

security system，MySQL访问权限系统
SEC_TO_TIME()，日期和时间函数
SELECT

LIMIT，SELECT语法
optimizing，EXPLAIN语法（获取SELECT相关信息）
Query Cache，MySQL查询高速缓冲

SELECT INTO，SELECT ... INTO语句
SELECT INTO TABLE，SELECT INTO TABLE
SELECT speed，SELECT查询的速度
selecting

databases，创建并选择数据库

select_limit variable，选项
SEQUENCE，使用AUTO_INCREMENT
sequence emulation，信息函数
sequences，使用AUTO_INCREMENT
server

connecting，连接与断开服务器，与MySQL服务器连接
debugging，调试MySQL服务器
disconnecting，连接与断开服务器
restart，Unix下安装后的过程
shutdown，Unix下安装后的过程
starting，Unix下安装后的过程
starting and stopping，自动启动和停止MySQL
starting problems，启动MySQL服务器以及其故障诊断和排除

server variables，服务器系统变量，系统变量，SHOW VARIABLES语法
server-side cursor restrictions，对服务器端光标的限制
servers

multiple，在同一台机器上运行多个MySQL服务器

SESSION_USER()，信息函数
SET，SET语法，Variable SET Statement

size，列类型存储需求

SET data type，字符串类型概述，SET类型
SET GLOBAL SQL_SLAVE_SKIP_COUNTER，SET GLOBAL SQL_SLAVE_SKIP_COUNTER语法
SET OPTION，SET语法
SET PASSWORD，SET PASSWORD语法
SET PASSWORD statement，设置账户密码
SET SQL_LOG_BIN，SET SQL_LOG_BIN语法
SET TRANSACTION，SET TRANSACTION语法
setting

passwords，设置账户密码

setting passwords，SET PASSWORD语法
setting program variables，使用选项设置程序变量
setup

post-installation，安装后的设置和测试

SHA()，加密函数

SHA1()，加密函数
shell syntax，本手册采用的惯例
SHOW BINARY LOGS，SHOW MASTER LOGS语法
SHOW BINLOG EVENTS，SHOW语法，SHOW BINLOG EVENTS语法
SHOW CHARACTER SET，SHOW CHARACTER SET语法
SHOW COLLATION，SHOW COLLATION Syntax
SHOW COLUMNS，SHOW语法，SHOW COLUMNS语法
SHOW CREATE DATABASE，SHOW语法，SHOW CREATE DATABASE语法
SHOW CREATE FUNCTION，SHOW CREATE PROCEDURE和SHOW CREATE FUNCTION
SHOW CREATE PROCEDURE，SHOW CREATE PROCEDURE和SHOW CREATE FUNCTION
SHOW CREATE SCHEMA，SHOW语法，SHOW CREATE DATABASE语法
SHOW CREATE TABLE，SHOW语法，SHOW CREATE TABLE语法
SHOW CREATE VIEW，SHOW CREATE VIEW语法
SHOW DATABASES，SHOW语法，SHOW DATABASES语法
SHOW ENGINE，SHOW语法，SHOW ENGINE语法
SHOW ENGINES，SHOW语法，SHOW ENGINES语法
SHOW ERRORS，SHOW语法，SHOW ERRORS语法
SHOW extensions，SHOW语句的扩展
SHOW FIELDS，SHOW语法
SHOW FUNCTION STATUS，SHOW PROCEDURE STATUS和SHOW FUNCTION STATUS
SHOW GRANTS，SHOW语法，SHOW GRANTS语法
SHOW INDEX，SHOW语法，SHOW INDEX语法
SHOW INNODB STATUS，SHOW语法
SHOW KEYS，SHOW语法，SHOW INDEX语法
SHOW MASTER LOGS，SHOW语法，SHOW MASTER LOGS语法
SHOW MASTER STATUS，SHOW语法，SHOW MASTER STATUS语法
SHOW OPEN TABLES，SHOW语法，SHOW OPEN TABLES语法
SHOW PRIVILEGES，SHOW语法，SHOW PRIVILEGES语法
SHOW PROCEDURE STATUS，SHOW PROCEDURE STATUS和SHOW FUNCTION STATUS
SHOW PROCESSLIST，SHOW语法，SHOW PROCESSLIST语法
SHOW SCHEMAS，SHOW语法，SHOW DATABASES语法
SHOW SLAVE HOSTS，SHOW语法，SHOW SLAVE HOSTS语法
SHOW SLAVE STATUS，SHOW语法，SHOW SLAVE STATUS语法
SHOW STATUS，SHOW语法
SHOW STORAGE ENGINES，SHOW ENGINES语法
SHOW TABLE STATUS，SHOW语法
SHOW TABLE TYPES，SHOW语法，SHOW ENGINES语法
SHOW TABLES，SHOW语法，SHOW TABLES语法
SHOW TRIGGERS，SHOW语法，SHOW TRIGGERS语法
SHOW VARIABLES，SHOW语法
SHOW WARNINGS，SHOW语法，SHOW WARNINGS语法
SHOW with WHERE，INFORMATION_SCHEMA信息数据库，SHOW语句的扩展
show-warnings

mysql option，选项

shutdown_timeout variable，mysqladmin：用于管理MySQL服务器的客户端
shutting down

the server，Unix下安装后的过程

sigint-ignore

mysql option，选项

SIGN()，数学函数
silent

mysql option，选项
mysqld_multi option，mysqld_multi：管理多个MySQL服务器的程序

silent column changes，沉寂的列规格变更
SIN()，数学函数
single quote (\')，字符串
size of tables，MySQL表最大能达到多少
sizes

display，列类型

skip-column-names

mysql option，选项

skip-line-numbers

mysql option，选项

SLEEP()，其他函数
slow queries，mysqladmin：用于管理MySQL服务器的客户端
slow query log，慢速查询日志
SMALLINT data type，数值类型概述
socket

mysql option，选项

socket location

changing，典型配置选项

Solaris installation problems，Solaris注意事项
Solaris troubleshooting，处理MySQL编译问题
Solaris x86_64 issues，InnoDB性能调节提示
sorting

character sets，数据和排序用字符集
data，分类行
grant tables，访问控制，阶段1：连接核实，访问控制，阶段2：请求核实
table rows，分类行

sort_buffer_size myisamchk variable，用于myisamchk的一般选项
sort_key_blocks myisamchk variable，用于myisamchk的一般选项
SOUNDEX()，字符串函数
SOUNDS LIKE，字符串函数
source distribution

installing，使用源码分发版安装MySQL

source distributions

on Linux，Linux源码分发版说明

SPACE()，字符串函数
MySQL中的空间扩展，前言
speed

compiling，编译和链接怎样影响MySQL的速度
increasing with replication，MySQL中的复制
inserting，INSERT语句的速度
linking，编译和链接怎样影响MySQL的速度
of queries，优化SELECT语句和其它查询，SELECT查询的速度

SQL

defined，MySQL数据库管理系统概述

SQL commands

replication masters，用于控制主服务器的SQL语句
replication slaves，用于控制从服务器的SQL语句

SQL-92

extensions to，MySQL标准的兼容性

SQL_BIG_RESULT，SELECT语法
SQL_BUFFER_RESULT，SELECT语法
SQL_CACHE，查询高速缓冲SELECT选项，SELECT语法
SQL_CALC_FOUND_ROWS，SELECT语法
SQL_NO_CACHE，查询高速缓冲SELECT选项，SELECT语法
SQL_SMALL_RESULT，SELECT语法
sql_yacc.cc problems，处理MySQL编译问题
SQRT()，数学函数
square brackets，列类型
SRID()，通用几何函数
SSH，用SSH以远程方式从Windows连接到MySQL
SSL and X509 Basics，使用安全连接
SSL command-line options，SSL命令行选项
SSL related options，SSL GRANT 选项，GRANT和REVOKE语法
stability，MySQL稳定性
standard SQL

extensions to，MySQL标准的兼容性

Standard SQL

differences from，GRANT和REVOKE语法

standards compatibility，MySQL标准的兼容性
START

XA transactions，XA事务SQL语法

START SLAVE，START SLAVE语法
START TRANSACTION，START TRANSACTION，COMMIT和ROLLBACK语法
starting

comments，'--'作为注释起始标记

mysqld，如何以普通用户身份运行MySQL
the server，Unix下安装后的过程
the server automatically，自动启动和停止MySQL

Starting many servers，在同一台机器上运行多个MySQL服务器
StartPoint()，LineString函数
startup options

default，使用选项文件

startup parameters，调节服务器参数

mysql，选项
mysqladmin，mysqladmin：用于管理MySQL服务器的客户端
tuning，系统因素和启动参数的调节

statements

GRANT，向MySQL增加新用户账户
INSERT，向MySQL增加新用户账户

statically

compiling，典型配置选项

STATISTICS

INFORMATION_SCHEMA table，INFORMATION_SCHEMA STATISTICS表

stats_method myisamchk variable，用于myisamchk的一般选项
status

tables，SHOW TABLE STATUS语法

status command

results，mysqladmin：用于管理MySQL服务器的客户端

status variables，服务器状态变量，SHOW STATUS语法
STD()，GROUP BY（聚合）函数
STDDEV()，GROUP BY（聚合）函数
STDDEV_POP()，GROUP BY（聚合）函数
STDDEV_SAMP()，GROUP BY（聚合）函数
STOP SLAVE，STOP SLAVE语法

stopping

the server，自动启动和停止MySQL

stopword list

user-defined，微调MySQL全文搜索

storage engine

ARCHIVE，ARCHIVE存储引擎

storage engines

choosing，存储引擎和表类型

storage of data，设计选择
storage requirements

column type，列类型存储需求

storage space

minimising，使你的数据尽可能小

stored procedures，存储程序和函数
stored procedures and triggers

defined，存储程序和触发程序

stored routine restrictions，对存储子程序和触发程序的限制
STRAIGHT_JOIN，SELECT语法，JOIN语法
STRCMP()，字符串比较函数
STRICT SQL mode，SQL服务器模式
STRICT_ALL_TABLES SQL mode，SQL服务器模式
STRICT_TRANS_TABLES SQL mode，SQL服务器模式
string collating，字符串比较支持
string comparison functions，字符串比较函数
string comparisons

case sensitivity，字符串比较函数

string functions，字符串函数
string literal introducer，字符串，字符串文字字符集和校对
string types，String类型
strings

defined，文字值
escaping characters，文字值
non-delimited，DATETIME、DATE和TIMESTAMP类型

striping

defined，磁盘事宜

STR_TO_DATE()，日期和时间函数
SUBDATE()，日期和时间函数
subpartitioning，子分区
subpartitions，子分区
subqueries，Subquery语法
subquery，Subquery语法
subquery restrictions，对子查询的限制
subselects，Subquery语法
SUBSTR()，字符串函数
SUBSTRING()，字符串函数
SUBSTRING_INDEX()，字符串函数
SUBTIME()，日期和时间函数

subtraction (-)，算术操作符
SUM()，GROUP BY（聚合）函数
SUM(DISTINCT)，GROUP BY（聚合）函数
superuser，使初始MySQL账户安全
support

for operating systems，MySQL支持的操作系统
mailing address，如何通报缺陷和问题

suppression

default values，对无效数据的约束

Sybase compatibility，USE语法
symbolic links，使用符号链接，在Windows平台上使用关于数据库的符号链接
SymDifference()，Spatial 操作符
syntax

regular expression，MySQL正则表达式

SYSDATE()，日期和时间函数
system

privilege，权限系统的作用
security，一般安全问题

system optimization，系统因素和启动参数的调节
system table，EXPLAIN语法（获取SELECT相关信息）
system variables，服务器系统变量，系统变量，SHOW VARIABLES语法
SYSTEM_USER()，信息函数

T

tab (\t)，字符串
table

changing，ALTER TABLE语法，与ALTER TABLE有关的问题
deleting，DROP TABLE语法
mysql option，选项

table aliases，SELECT语法
table cache，MySQL如何打开和关闭表
table is full，SET语法，表已满
table names

case sensitivity，识别符大小写敏感性
case-sensitivity，MySQL对标准SQL的扩展

Table scans

avoiding，如何避免表扫描

table types

choosing，存储引擎和表类型

table-level locking，锁定方法
tables

ARCHIVE，ARCHIVE存储引擎
BDB，BDB (BerkeleyDB)存储引擎
Berkeley DB，BDB (BerkeleyDB)存储引擎
BLACKHOLE，BLACKHOLE存储引擎
changing column order，如何更改表中的列顺序
checking，用于myisamchk的检查选项
closing，MySQL如何打开和关闭表
compressed format，压缩表特性
constant，EXPLAIN语法（获取关于SELECT的信息），MySQL怎样优化WHERE子句
copying，CREATE TABLE语法
counting rows，计数行
creating，创建表
CSV，CSV存储引擎
defragment，建立表维护计划，动态表特性
defragmenting，OPTIMIZE TABLE语法
deleting rows，从相关表删除行
displaying status，SHOW TABLE STATUS语法
dynamic，动态表特性
error checking，如何检查MyISAM表的错误
EXAMPLE，EXAMPLE存储引擎
FEDERATED，FEDERATED存储引擎
flush，mysqladmin：用于管理MySQL服务器的客户端
fragmentation，OPTIMIZE TABLE语法
grant，访问控制，阶段2：请求核实
HEAP，MEMORY (HEAP)存储引擎
host，访问控制，阶段2：请求核实
improving performance，使你的数据尽可能小
information，获取关于表的信息
information about，获得数据库和表的信息
InnoDB，InnoDB存储引擎
loading data，将数据装入表中
maintenance regimen，建立表维护计划
maximum size，MySQL表最大能达到多少
MEMORY，MEMORY (HEAP)存储引擎
MERGE，MERGE存储引擎
merging，MERGE存储引擎
multiple，使用1个以上的表

MyISAM，MyISAM存储引擎
names，数据库、表、索引、列和别名
open，MySQL如何计算打开的表
opening，MySQL如何打开和关闭表
optimizing，Table 优化
partitioning，MERGE存储引擎
repairing，如何修复表
retrieving data，从表检索信息
selecting columns，选择特殊列
selecting rows，选择特殊行
sorting rows，分类行
symbolic links，在Unix平台上使用表的符号链接

system，EXPLAIN语法（获取SELECT相关信息）
too many，在同一个数据库中创建多个表的缺陷
unique ID for last row，如何获得上次插入行的唯一ID
updating，事务和原子操作

TABLES

INFORMATION_SCHEMA table，INFORMATION_SCHEMA TABLES表

table_cache，MySQL如何打开和关闭表
TABLE_PRIVILEGES

INFORMATION_SCHEMA table，INFORMATION_SCHEMA TABLE_PRIVILEGES表

TAN()，数学函数
tar

problems on Solaris，Solaris注意事项

Tcl API，MySQL Tcl API
tcp-ip

mysqld_multi option，mysqld_multi：管理多个MySQL服务器的程序

TCP/IP，选择MySQL服务器类型，测试MySQL安装
technical support

mailing address，如何通报缺陷和问题

tee

mysql option，选项

temporary file

write access，与运行mysql_install_db有关的问题

temporary tables

problems，TEMPORARY TABLE问题

terminal monitor

defined，教程

testing

connection to the server，访问控制，阶段1：连接核实
installation，Unix下安装后的过程
of MySQL releases，选择要安装的MySQL版本
post-installation，安装后的设置和测试

testing mysqld

mysqltest，MySQL测试套件

TEXT

size，列类型存储需求

TEXT columns

default values，BLOB和TEXT类型
indexing，列索引，CREATE TABLE语法

TEXT data type，字符串类型概述，BLOB和TEXT类型
thread packages

differences between，线程软件包之间的差异

thread support，MySQL支持的操作系统

non-native，MIT-pthreads注意事项

threaded clients，如何生成线程式客户端
threads，mysqladmin：用于管理MySQL服务器的客户端，SHOW PROCESSLIST语法，MySQL内部控件

display，SHOW PROCESSLIST语法
RTS，关于RTS线程的注释

TIME data type，日期和时间类型概述，TIME类型
time types，列类型存储需求
time zone problems，时区问题
TIME()，日期和时间函数
TIMEDIFF()，日期和时间函数
timeout，服务器系统变量，其他函数，INSERT DELAYED语法

connect_timeout variable，选项，mysqladmin：用于管理MySQL服务器的客户端
shutdown_timeout variable，mysqladmin：用于管理MySQL服务器的客户端

TIMESTAMP

and NULL values，与NULL值有关的问题

TIMESTAMP data type，日期和时间类型概述，DATETIME、DATE和TIMESTAMP类型
TIMESTAMP()，日期和时间函数
TIMESTAMPADD()，日期和时间函数
TIMESTAMPDIFF()，日期和时间函数
TIME_FORMAT()，日期和时间函数
TIME_TO_SEC()，日期和时间函数
TINYBLOB data type，字符串类型概述
TINYINT data type，数值类型概述
TINYTEXT data type，字符串类型概述
tips

optimization，其它优化技巧

TMPDIR environment variable，与运行mysql_install_db有关的问题，环境变量
TODO

embedded server，嵌入式服务器中尚需完成的事项(TODO)
symlinks，在Unix平台上使用表的符号链接

tools

list of，用于创建MySQL的工具

Touches()，测试几何类之间空间关系的函数
TO_DAYS()，日期和时间函数
trace DBI method，在gdb环境下调试mysqld
TRADITIONAL SQL mode，SQL服务器模式
transaction-safe tables，事务和原子操作，InnoDB概述
transactions

support，事务和原子操作，InnoDB概述

Translators

list of，资料员和译员

trigger restrictions，对存储子程序和触发程序的限制
triggers，存储程序和触发程序，SHOW TRIGGERS语法，触发程序
TRIGGERS

INFORMATION_SCHEMA table，INFORMATION_SCHEMA TRIGGERS表

TRIM()，字符串函数
troubleshooting

FreeBSD，处理MySQL编译问题
Solaris，处理MySQL编译问题

TRUE，数值

testing for，比较函数和操作符

TRUNCATE，TRUNCATE语法
TRUNCATE()，数学函数
tutorial，教程
Twin Studies

queries，孪生项目的查询

type conversions，比较函数和操作符
types

columns，列类型，选择正确的列类型
data，列类型
date，列类型存储需求
Date and Time，日期和时间类型
numeric，列类型存储需求
of tables，存储引擎和表类型
portability，使用来自其他数据库引擎的列类型
strings，String类型
time，列类型存储需求

typographical conventions，本手册采用的惯例
TZ environment variable，时区问题，环境变量

U

UCASE()，字符串函数
UCS-2，字符集支持
UDFs，CREATE FUNCTION/DROP FUNCTION语法

compiling，编译和安装用户定义的函数
defined，为MySQL添加新函数
return values，UDF返回值和错误处理

ulimit，文件未找到
UMASK environment variable，与文件许可有关的问题，环境变量
UMASK_DIR environment variable，与文件许可有关的问题，环境变量
unary minus (-)，算术操作符
unbuffered

mysql option，选项

UNCOMPRESS()，字符串函数
UNCOMPRESSED_LENGTH()，字符串函数
UNHEX()，字符串函数
Unicode，字符集支持
Unicode Collation Algorithm，Unicode字符集
UNION，根据两个键搜索，UNION语法

Union()，Spatial 操作符
UNIQUE，ALTER TABLE语法

constraint，PRIMARY KEY和UNIQUE索引约束

unique ID，如何获得上次插入行的唯一ID
UNIX_TIMESTAMP()，日期和时间函数
UNKNOWN

testing for，比较函数和操作符

unloading

tables，从表检索信息

UNLOCK TABLES，LOCK TABLES和UNLOCK TABLES语法
unnamed views，FROM子句中的子查询
UNTIL，REPEAT语句
UPDATE，UPDATE语法
updating

releases of MySQL，如何发布更新以及何时发布更新
tables，事务和原子操作

upgrading，升级MySQL

different architecture，将MySQL数据库拷贝到另一台机器
grant tables，升级授权表

UPPER()，字符串函数

uptime，mysqladmin：用于管理MySQL服务器的客户端
URLs for downloading MySQL，怎样获得MySQL
USE，USE语法
USE INDEX，SELECT语法，JOIN语法
USE KEY，SELECT语法，JOIN语法
user

mysql option，选项
mysqld_multi option，mysqld_multi：管理多个MySQL服务器的程序

user accounts

creating，CREATE USER语法
renaming，RENAME USER语法

USER environment variable，与MySQL服务器连接，环境变量
user privileges

adding，向MySQL增加新用户账户
deleting，从MySQL删除用户账户，DROP USER语法
dropping，从MySQL删除用户账户，DROP USER语法

user table

sorting，访问控制，阶段1：连接核实

user variables，用户变量
USER()，信息函数
user-defined functions

adding，为MySQL添加新函数，添加新的自定义函数

User-defined functions，CREATE FUNCTION/DROP FUNCTION语法
usernames

and passwords，MySQL用户名和密码

users

adding，在其它类Unix系统中安装MySQL，源码安装概述
deleting，从MySQL删除用户账户，DROP USER语法
root，使初始MySQL账户安全

USER_PRIVILEGES

INFORMATION_SCHEMA table，INFORMATION_SCHEMA USER_PRIVILEGES表

uses

of MySQL，我们已将MySQL用在何处？

using multiple disks to start data，在Windows平台上使用关于数据库的符号链接
UTC_DATE()，日期和时间函数
UTC_TIME()，日期和时间函数
UTC_TIMESTAMP()，日期和时间函数
UTF-8，字符集支持

UTF8，字符集支持
UUID()，其他函数

V

valid numbers

examples，数值

VALUES()，其他函数
VARBINARY data type，字符串类型概述，BINARY和VARBINARY类型
VARCHAR

size，列类型存储需求

VARCHAR data type，字符串类型概述，String类型
VARCHARACTER data type，字符串类型概述
variables

mysqld，调节服务器参数
server，服务器系统变量，SHOW VARIABLES语法
status，服务器状态变量，SHOW STATUS语法
system，服务器系统变量，SHOW VARIABLES语法
System，系统变量
user，用户变量
values，服务器系统变量

VARIANCE()，GROUP BY（聚合）函数
VAR_POP()，GROUP BY（聚合）函数
VAR_SAMP()，GROUP BY（聚合）函数
verbose

mysql option，选项
mysqld_multi option，mysqld_multi：管理多个MySQL服务器的程序

version

choosing，选择要安装的MySQL分发版
latest，怎样获得MySQL
mysql option，选项
mysqld_multi option，mysqld_multi：管理多个MySQL服务器的程序

VERSION()，信息函数
vertical

mysql option，选项

view restrictions，对视图的限制
views，视图，视图，CREATE VIEW语法

updatable，视图，CREATE VIEW语法

VIEWS

INFORMATION_SCHEMA table，INFORMATION_SCHEMA VIEWS表

virtual memory

problems while compiling，处理MySQL编译问题

Visual Basic，已知的能与MyODBC一起工作的程序

W

wait

mysql option，选项

WEEK()，日期和时间函数
WEEKDAY()，日期和时间函数
WEEKOFYEAR()，日期和时间函数
Well-Known Binary format，著名的二进制(WKB)格式
Well-Known Text format，著名的文本(WKT)格式
What is an X509/Certificate?，Basic SSL Concepts
What is encryption，Basic SSL Concepts
WHERE，MySQL怎样优化WHERE子句

with SHOW，INFORMATION_SCHEMA信息数据库，SHOW语句的扩展

WHILE，WHILE Statement
widths

display，列类型

Wildcard character (%)，字符串
Wildcard character (_)，字符串
wildcards

and LIKE，MySQL如何使用索引
in mysql.columns_priv table，访问控制，阶段2：请求核实
in mysql.db table，访问控制，阶段2：请求核实
in mysql.host table，访问控制，阶段2：请求核实
in mysql.tables_priv table，访问控制，阶段2：请求核实
in mysql.user table，访问控制，阶段1：连接核实

Windows，MySQL Connector/ODBC

compiling on，在Windows下编译MySQL客户端
open issues，Windows版MySQL同Unix版MySQL对比
upgrading，在Windows下升级MySQL
versus Unix，Windows版MySQL同Unix版MySQL对比

with-big-tables option，典型配置选项
Within()，测试几何类之间空间关系的函数
without-server option，典型配置选项
WKB format，著名的二进制(WKB)格式
WKT format，著名的文本(WKT)格式
Word program，已知的能与MyODBC一起工作的程序
wrappers

Eiffel，MySQL Eiffel Wrapper

write access

tmp，与运行mysql_install_db有关的问题

write_buffer_size myisamchk variable，用于myisamchk的一般选项

X

X()，Point函数

XA BEGIN，XA事务SQL语法
XA COMMIT，XA事务SQL语法
XA PREPARE，XA事务SQL语法
XA RECOVER，XA事务SQL语法
XA ROLLBACK，XA事务SQL语法
XA START，XA事务SQL语法
XA transactions，XA事务

transaction identifiers，XA事务SQL语法

xid

XA transaction identifier，XA事务SQL语法

xml

mysql option，选项

XOR

bitwise，位函数
logical，逻辑操作符

Y

Y()，Point函数

Year 2000 compliance，2000年兼容性
Year 2000 issues，Y2K事宜和日期类型
YEAR data type，日期和时间类型概述，YEAR类型
YEAR()，日期和时间函数
YEARWEEK()，日期和时间函数

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问dev.mysql.com。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。