http://hi.baidu.com/whhitprince
更改了目录中的错误

前　言

MySQL 是一个广受 Linux 社区人们喜爱的半商业的数据库。MySQL 是可运行在大多数的 Linux 平台(i386，Sparc，etc)，以及少许非 Linux 甚至非 Unix 平台。

MySQL 的普及很大程度上源于它的宽松，除了略显不寻常的许可费用。MySQL 的价格随平台和安装方式变化。MySQL 的 Windows 版本（NT 和 9X）在任何情况下都不免费，而任何 Unix 变种（包括 Linux）的 MySQL 如果由用户自己或系统管理员而不是第三方安装则是免费的，第三方安装则必须付许可费。而且现在已经有了符合 GPL 的 MySQL 版本。

MySQL 具有这样明显的优势，并且由于它优异的性能，MySQL 的应用也越来越广泛，使用者也越来越多。但是在市场上，相应的适合培训的教材却屈指可数，广大爱好者苦于资料的缺乏而无法对 MySQL 做进一步的了解。为了适应培训市场的需求，本书定位于具有一定计算机知识读者的培训教材。读者在本书中，可以循序渐进的掌握 MySQL 的安装、使用、管理、备份和安全等多方面的知识。由于内容的翔实和学习内容的由浅入深，本书也适合于个别读者的自学过程。

本书的一大特色是——对于每一项具体的服务器操作，都尽量给出了多种的解决方案。读者既可以学会如何使用 MySQL，同时也可以通过这样的安排一窥 MySQL 的灵活性，并且可以通过这种举一反三的方法，对 MySQL 系统有一个非常详尽的了解，也能够加深对培训内容的理解和记忆。

此外，每一章的结束，本书都提出若干思考题，基本上都覆盖了该章的内容，可以用来测试培训的效果，也可以用来读者可以用之检测自己的掌握程度。在正文中的例子，读者也不应该忽视，阅读后建议重复正文中所有的例子，再考虑思考题中的内容。

本书结构的这种安排就是最大限度的满足培训教材的要求，同时为普通读者的阅读创造方便，使本书包含较大的信息量。在通读本书之后，相信读者可以对 MySQL 有了很深的了解，可以独立的对 MySQL 数据库系统进行管理。

编者时间仓促，难免在文中有疏漏之处，如果给您的阅读造成困难，请谅解。

编著者

内 容 提 要

本书详细介绍了如何安装、管理、备份、维护和优化一个 MySQL 系统。对于每一件服务器操作都提出了多种的解决方案。对于每一种的方法，虽然不一定都是非常实用的方法，读者能通过这些方法，熟悉 MySQL 的特点和强大的功能。

第一章简单介绍了 MySQL 的历史、特点，同时对 SQL 的语法进行了简单的介绍。如果读者对第一章的内容不能很好的掌握，可以略过不了解的内容，在阅读二、三章之后重新理解其中内容。第二章介绍了如何安装一个 MySQL 系统。第三、四章详细叙述了如何利用 SQL 语言以及其它的客户工具对 MySQL 数据库中的数据进行操作。第五、六章介绍了数据库目录以及如何备份、恢复和维护数据库安装。第七章介绍了 MySQL 权限系统和如何为数据库创建、撤销授权。第八章则对优化数据库性能的各个方面进行了阐述。

附录一中详细列出了 MYSQL 列类型、函数和 SQL 语句的参考。附录二全面介绍了本书出现的程序的全部选项及其含义。

目　　录

第1章

MySQL 入门与初步

本章要点：

- MySQL 标准简介
- 关系数据库系统的知识
- MySQL 的优点
- MySQL 的 SQL 语言

MySQL 是完全网络化的跨平台关系型数据库系统，同时是具有客户机/服务器体系结构的分布式数据库管理系统。MySQL 在 UNIX 等操作系统上是免费的，在 Windows 操作系统上，可免费使用其客户机程序和客户机程序库。

MySQL 是一个精巧的 SQL 数据库管理系统，虽然它不是开放源代码的产品，但在某些情况下你可以自由使用。由于它的功能强大、使用简便、管理方便、运行速度快、安全可靠性强、灵活性、丰富的应用编程接口（API）以及精巧的系统结构，受到了广大自由软件爱好者甚至是商业软件用户的青睐，特别是与 Apache 和 PHP/PERL 结合，为建立基于数据库的动态网站提供了强大动力。

1.1 MySQL 简介

MySQL 的进展是非常快了，越来越多的领域里都可以见到 MySQL 的身影，在学习如何使用这个强大的系统之前，我们首先要了解 MySQL 的历史、功能、特点，你会对学习 MySQL 更具信心。

在正式开始之前，我们现了解一下它的读音。MySQL 的官方发音是“My Ess Que Ell”(不是 MY-SEQUEL)。

1.1.1 MySQL 是什么？

MySQL 是一个真正的多用户、多线程 SQL 数据库服务器。SQL（结构化查询语言）是世界上最流行的和标准化的数据库语言。MySQL 是以一个客户机/服务器结构的实现，它由一个服务器守护程序 mysqld 和很多不同的客户程序和库组成。

SQL 是一种标准化的语言，它使得存储、更新和存取信息更容易。例如，你能用 SQL 语言为一个网站检索产品信息及存储顾客信息，同时 MySQL 也足够快和灵活以允许你存储记录文件和图像。

MySQL 主要目标是快速、健壮和易用。最初是因为我们需要这样一个 SQL 服务器，它能处理与任何可不昂贵硬件平台上提供数据库的厂家在一个数量级上的大型数据库，但速度更快，MySQL 就开发出来。自 1996 年以来，我们一直都在使用 MySQL，其环境有超过 40 个数据库，包含 10,000 个表，其中 500 多个表超过 7 百万行，这大约有 100 个吉字节(GB)的关键应用数据。

MySQL 建立的基础是业已用在高要求的生产环境多年的一套实用例程。尽管 MySQL 仍在开发中，但它已经提供一个丰富和极其有用的功能集。

MySQL 最早起始于 1979 年，开始是 Michael“Monty”Widenius 为瑞典的 TcX 公司创建的 UNIREG 数据库工具。1994 年，TcX 开始寻找一个用来开发 Web 应用程序的 SQL 服务器。他们测试了一些商业服务器，但是发现所有服务器对于 TcX 的大型表来说都太慢。他们也试了 mSQL，但它缺乏 TcX 需要的某些功能。因此，Monty 开始开发一种新的服务器。其编程接口明确地设计为类似 mSQL 的编程接口，因为 mSQL 可得到几个免费的工具，所以利用与 mSQL 类似的接口，可以将这些相同的工具用于 MySQL 从而大大减少了开发接口的工作。

1995 年，Detron HB 公司的 David Axmark 努力争取 TcX 公司在因特网上发布 MySQL。David 还做了文档资料方面的工作和使 MySQL 与 GNU 的配置实用程序一起建造的工作。MySQL 3.11.1 在 1996 年以用于 Linux 和 Solaris 系统的二进制分发形式发布。今天，MySQL 正工作在许多平台上，并且二进制和源代码的形式都可以得到。

MySQL 并不是一个开放源代码的产品，因为在某些条件下使用它需要许可证。但是，MySQL 很愿意在开放源代码的团体内得以普及，因为“认证”这个术语并不是非常有约束力的（除非通过出售 MySQL 或出售需要它的服务来挣钱，否则，大体上说 MySQL 一般是免费的）。

MySQL 的普及并不限于开放源代码团体内。虽然它在个人计算机上运行（确实，

MySQL 的开发一般在不昂贵的 Linux 系统上进行），但它是可移植的，并且运行在商用操作系统（如 Solaris、Irix 和 Windows）和一直到企业服务器的各种硬件上。此外，它的性能也足以和任何其他系统相匹敌，而且它还可以处理具有数百万个记录的大型数据库。

MySQL 的广泛应用前景在我们面前尚未完全展开，如运行在功能强但不昂贵的硬件上的免费可用操作系统，将丰富的处理功能和能力提供给比以往更多的人，在比过去范围更广的系统上运行等等。信息处理的经济障碍的降低使强有力的数据库解决方案到达了比过去任何时候更多的人和机构的手中。例如，本人在运行 LinuxPPC 的 G3 PowerBook 笔记本电脑上使用 MySQL 与 Perl、Apache 和 PHP，这允许本人在任何地方都可以进行工作，总的成本只是 PowerBook 的成本。

过去只能梦想将高性能的 RDBMS 用于自己工作的机构，现在可以这样做了，并且开销很低。数据库的利用在单一的层次上也在不断地增加。过去从未想过要使用数据库的人现在也开始考虑一旦得到一个数据库，怎样将其用于自己的各种目的，例如用来存储和访问系统的研究结果，跟踪和维护最喜爱的收藏物（蝴蝶、邮票、捧球明星卡等等），帮助管理新开张的公司，或者提供个人 Web 站点的搜索能力。

1.1.2 我需要 MySQL 吗？

如果您正在寻找一种免费的或不昂贵的数据库管理系统，可以有几个选择，如，MySQL、mSQL、Postgres（一种免费的但不支持来自商业供应商引擎的系统）等。在将 MySQL 与其他数据库系统进行比较时，所要考虑的最重要的因素是性能、支持、特性（与 SQL 的一致性、扩展等等）、认证条件和约束条件、价格等。相比之下，MySQL 具有许多吸引人之处：

1、速度。MySQL 运行速度很快。开发者声称 MySQL 可能是目前能得到的最快的数据库。可访问 http://www.mysql.com/benchmark.html （MySQL Web 站点上的性能比较页），调查一下这个性能。

2、容易使用。MySQL 是一个高性能且相对简单的数据库系统，与一些更大系统的设置和管理相比，其复杂程度较低。

3、价格。MySQL 对多数个人用户来说是免费的。详细的信息请参阅本前言后面的“MySQL 是否免费”一节。

4、支持查询语言。MySQL 可以利用 SQL（结构化查询语言），SQL 是一种所有现代数据库系统都选用的语言。也可以利用支持 ODBC（开放式数据库连接）的应用程序，ODBC 是 Microsoft 开发的一种数据库通信协议。

5、性能。许多客户机可同时连接到服务器。多个客户机可同时使用多个数据库。可利用几个输入查询并查看结果的界面来交互式地访问 MySQL。这些界面为：命令行客户机程序、Web 浏览器或 X Window System 客户机程序。此外，还有由各种语言（如 C、Perl、Java、PHP 和 Python）编写的界面。因此，可以选择使用已编好的客户机程序或编写自己的客户机应用程序。

6、连接性和安全性。MySQL 是完全网络化的，其数据库可在因特网上的任何地方访问，因此，可以和任何地方的任何人共享数据库。而且 MySQL 还能进行访问控制，可以

控制哪些人不能看到您的数据。

7、可移植性。MySQL 可运行在各种版本的 UNIX 以及其他非 UNIX 的系统（如 Windows 和 OS/2）上。MySQL 可运行在从家用 PC 到高级的服务器上。

如果，你对上面的特性非常在意，尤其是价格和速度、性能方面，那么我认为 MySQL 十分适合你。

1.1.3 我需要付钱吗?

基本上，我们许可证政策如下：

对于一般的内部使用，MySQL 通常是免费的。如果你不想，就不必付钱给我们。

一个许可证是必需的，如果：

你直接销售 MySQL 服务器或作为其他产品或服务的一部分；

你在某些客户那里为了安装和维护一个 MySQL 服务器而收费；

你在不可再分发的分发中包括 MySQL 并且你对该分发的某些部分收费；

在必须有 MySQL 许可证的情况下，对每台运行 mysqld 服务器的机器，你都需要一个许可证，但多 CPU 机器按单 CPU 计算，并且在一台机器上运行 MySQL 服务器的数量或并发连接到这台运行一个服务器的机器的客户数量也无限制！

你在商业程序中包含客户代码不需要一个许可证，MySQL 的客户端存取部分不属公共领域，mysql 命令行客户程序包含在 GNU 通用许可证下的 readline 库的代码。

对于已经购买了 10 个许可证或一种足够级别的技术支持的消费者，我们提供附加的功能。目前，这意味着我们提供 myisampack 实用工具，它能生成快速的压缩的只读数据库（服务器支持读取这样的数据库，但不包含用于生成它们的压缩工具)。当支持协议产生了足够的收入时，我们将在与 MySQL 服务器同样的许可证下发行这个工具。

如果你使用不需要一个许可证的 MySQL，但是你的确喜欢 MySQL 并且有志于更进一步的开发，无论如何肯定欢迎你购买一个许可证。

如果你在一个商业环境中使用 MySQL 以便通过它获利，我们要求你购买一定级别的技术支持以推进开发。我们感到，如果 MySQL 有助于你的业务，要求你帮助 MySQL 也是合理的（否则，如果向我们你询问支持问题，你不仅是正在免费使用我们倾注大量精力的产品，而且你正在要求我们提供免费的支持。)。

对于在微软操作系统下面(Win95/Win98/WinNT)的使用，你在一个 30 天的试用期后需要一个 MySQL 许可证，除了教育用途或大学或政府资助的研究机构的许可证可免费申请获得，见 K 针对微软操作系统的 MySQL 许可证。一个共享软件版本的 MySQL -Win32 可在购买前从 http://www.mysql.com/mysql_w32.htmy 下载试用。在你付钱后，你将得到一个口令让你能够访问最新 MySQL - Win32 版本。

如果你需要一个 MySQL 许可证，最容易的付款方法是使用在 https://www.mysql.com/license.htmy 网站的 Tcx 的安全服务器上的许可证表格。

1.1.4 如何得到 MySQL?

在本小节读者不仅讲知道如何得到 MySQL 还将知道如何得到与 MySQL 有关的很多软件，虽然由于篇幅和本书写作目的的原因，我们不可能一一介绍它们：

从 TcX 公司的 MySQL 网站上 http://www.mysql.com/doc.html 你可以得到大部分需要的软件包：

MySQL 分发包（包括各种平台——Windows、Linux 等——，各种形式——二进制、源代码、RPM 包——的分发），现在已经有适用 GPL 许可证的 MySQL 分发，请仔细阅读最新的许可信息。

MySQL 用于 ODBC 驱动程序 MyODBC。

MySQL 的 JDBC 驱动程序 mm.mysql。

用 PHP 写成的客户程序 phpMyAdmin。

以及其它各种第三方客户机。

获得 Active Perl

http://www.activestate.com/

获得 Perl DBI

http://www.symbolstone.org/technology/perl/DBI

获得 PHP

www.php.net

获得 Apache

http://www.apache.org

1.1.5 总结

本节简单介绍的 MySQL 的方方面面，相信读者对 MySQL 已经有了一定的了解了。看看你是不是了解了它：

- MySQL 是一个关系数据库系统，支持 SQL 查询语言。
- MySQL 可以是免费的，你不需要为它付费。
- MySQL 系统的速度非常快，同样它的性能也是十分优良的。
- MySQL 是一个管理简捷的数据库，它没有庞大而臃肿的可视化管理工具。

1.2 关系数据库管理系统

在过去的许多年里，有许多关于“数据库”这个名词的定义。数据库是一个服务于一个核心目标的数据的有组织的集合。数据库中的数据是有组织的，从某种意义上说，数据库中存储的数据采用一种不变的方式被存储、格式化、存取以及显示。因为数据库不含有无关的或冗余的数据，它可以适用于一个核心目标。一本电话簿就是一个很好的数据库例子，它包含有关的数据（名字），让人们能够查找电话号码；它不包含无关的数据，如某人的电话机的颜色；它只贮存那些与它的目标相关的信息。最常见的，一个数据库的目标是商务应用，但是也可能贮存科学、军事或其他数据，这些数据通常不能当作商务数据看待。因此，有商业数据库、科学数据库、军事数据库以及其他的数据库等等。另外，数据不仅能根据它的应用分类，还能根据它的格式分类，现代数据库包括多种类型的数据。例如，现在数据库贮存图像、图表、声音、视频或包括两种或多种类型的复合文档，已经是很普

通的事了。

1.2.1 关系数据库系统

所谓RDBMS，即关系数据库管理系统，

为了进一步了解一个RDBMS 是由什么构成的，你必须先了解关系模型。下列情况出现在一个关系模型中：

- 数据的基础项是关系。
- 在这些表上的操作只产生关系（关系型闭合）。

什么是关系？这是一个描述两个集合的元素如何相互联系或如何一一对应的数学概念。因此，关系模型是建立在数学基础上的。然而，对你来说，关系只是一个带有一些特殊属性的表，一个关系模型把数据组织到表中，而且仅在表中。客户、数据库设计者、数据库系统管理员和用户都以同样的方式—即从表中—查看数据。那么，表就是关系模型的近义词。

一个关系型表有一组命名的属性（attribute ）或列，以及一组元组（tuple ）或行。有时列被称为域，行被称为记录，列和行的交集通常被叫做单元。列标示位置，有作用域或数据类型，例如字符或整数。行自己就是数据。

一个关系表必须符合某些特定条件，才能成为关系模型的一部分：

1、贮存在单元中的数据必须是原子的。

每个单元只能存贮一条数据，这也叫信息原则（Information Principle ）。尽管在过去的数年中按某些违反这一条的方式已经建立了许多系统，但违反这一条将不能运用良好的设计原则。当一个单元包含多于一条的信息时，这叫做信息编码（information coding ）。在这样的情况下，是否采用违背理论的方案是一个设计的选择问题，尽管在多数情况下，结果证明这对数据的完整性是一不利的。

2、贮存在列下的数据必须具有相同数据类型。

3、每行是唯一的（没有完全相同的行）。

4、列没有顺序。

5、行没有顺序。

6、列有一个唯一性的名称。

除了表和它们的属性，关系模型有它自己特殊的操作。不需要深入研究关系型数学，只需说明这些操作可能包括列的子集、行的子集、表的连接以及其他数学集合操作（如联合）等就足够了。真正要知道的事情是这些操作把表当作输入，而将产生的表作为输出。

SQL 是当前RDBMS 的ANSI 标准语言，它包含这些关系型操作。允许数据操作或数据处理的主要语句是SELECT 、INSERT 、UPDATE 和DELETE 。因此，这些数据处理操作中任何一个都是一个事务。

允许数据定义或结构化处理的基本语句是CREATE 、ALTER 和DROP 。关系模型要求的最后一件事是两个基础的完整性原则。它们是实体完整性原则（entity integrity rule ）和引用完整性原则（referential integrity rule ）。首先，让我们看看两个定义：

1、主键（primary key ）是能唯一标识行的一列或一组列的集合。有时，多个列或多

组列可以被当作主键。

2、由多个列构成的主键被称为连接键（concatenated key ）、组合键（compound key ），或者更常称为复合键（composite key ）。

数据库设计者决定哪些列的组合能够最准确和有效地反映业务情形，这并不意味着其他数据未被存贮，只是那一组列被选作主键而已。

剩余有可能被选为主键的列被叫做候选键（candidate key ）或替代键（alternate key ）。一个外键（foreign key ）是一个表中的一列或一组列，它们在其他表中作为主键而存在。一个表中的外键被认为是对另外一个表中主键的引用。实体完整性原则简洁地表明主键不能全部或部分地空缺或为空，引用完整性原则简洁地表明一个外键必须为空或者与它所引用的主键当前存在的值相一致。

一个ＲＤＢＭＳ 就是一个建立在前面这些关系模型基础上的，一般能满足所提到的全部要求

的ＤＢＭＳ 。但是，在７０ 年代末到８０ 年代初，ＲＤＢＭＳ 开始销售的时候，ＳＱＬ超越了本质为非关系型的系统，受到普遍欢迎，并被称作关系型。

1.2.2 数据库系统的发展

1969 年美国的 IBM 公司开发了第一个数据库系统 IMS。这是一个层次数据库系统，在数据库系统发展史上有着重要的地位。同年，美国的数据系统语言委员会(CODASYL)下属的数据库任务组提出了著名的 DBTG 报告，并在 1970 年提出了该报告的修订版。这份报告定义了数据库操纵语言、模式定义语言和子模式定义语言的概念。数据库操纵语言用于编写操纵概念视图的应用程序，模式定义语言用来编写概念视图和内部视图相结合的模式程序。在七十年代，开发了许多遵循 DBTG 报告的网状数据库系统，如：IDMS, IDS, DMSIIOO 等。七十年代初，E.F.Codd 提出了关系数据模型的概念，提出了关系代数和关系演算。在整个七十年代，关系数据库从理论到实践都取得了辉煌成果。在理论上，确立了完整的关系理论、数据依赖理论以及关系数据库的设计理论等等；在实践上，开发了许多著名的关系数据库系统，如：system R, INGRES, ORACLE 等。1986 年美国国家标准协会(ANSI)通过了关系数据库查询语言 SQL 的文本标准。进入八十年代以后，随着计算机硬件技术的提，使得计算机应用不断深入，产生了许多新的应用领域，如：计算机辅助设计、计算机辅助教学、计算机辅助制造、计算机辅助工程、计算机集成制造、办公自动化、地理信息处理、智能信息处理等等。这些新的应用领域对数据库系统提出了新要求。由于没能设计出一个统一的数据模型来表示这些新型数据及其相互联系，所以出现了百家争鸣的局面，产生了演绎数据库（逻辑数据库，知识库）、面向对象数据库、工程数据库、时态数据库、地理数据库、模糊数据库、积极数据库、…等新型数据库的研究。到八十年代后期和九十年代初期，出现了面向对象数据库系统，如：GemStone, VBASE, ORION, Iris 等。到目前为止，真正的新一代数据库系统还没有出现。

1.2.3 与数据库系统通讯

结构化查询语言（Structured Query Language ，SQL）是当今主要的查询语言，它主要用于管理主流类型的ＤＢＭＳ —关系型ＤＢＭＳ （ＲＤＢＭＳ ）。所有与数据库相关的

通信往来都将通过ＤＢＭＳ 完成，为了做这件事，你可以使用ＳＱＬ 或其他类似的东西。数据库系统管理员（ＤＢＡ ）使用查询语言来建立并维护数据库，用户使用查询语言来访问数据库并查看或更改数据。

1.2.4 MySQL 的体系结构

因为 MySQL 采用的是客户机/服务器体系结构，所以你在使用 MySQL 时存取数据时，必须至少使用两个或者说两类程序：

1、一个位于存放您的数据的主机上的程序——数据库服务器。数据库服务器监听从网络上传过来的客户机的请求并根据这些请求访问数据库的内容，以便向客户机提供它们所要求的信息。

2、连接到数据库服务器的程序——客户机，这些程序是用户和服务器交互的工具，告诉服务器需要什么信息的查询。

MySQL 分发包包括服务器和几个客户机程序。可根据要你要达到的目的来选择使用客户机。最常用的客户机程序为 mysql，这是一个交互式的客户机程序，它能发布查询并看到结果。其他的客户机程序有： mysqldump 和 mysqlimport，分别导出表的内容到某个文件或将文件的内容导入某个表；mysqladmin 用来查看服务器的状态并完成管理任务，如告诉服务器关闭、重起服务器、刷新缓存等。如果具有现有的客户程序不满足你的需要，那么 MySQL 还提供了一个客户机编程库，可以编写自己的程序。客户机编程库可直接从 C 程序中调用，如果希望使用 C 语言以外的其他语言，还有几种其他的接口可用。

MySQL 的客户机/服务器体系结构具有如下优点：

1、服务器提供并发控制，使两个用户不能同时修改相同的记录。所有客户机的请求都通过服务器处理，服务器分类辨别谁准备做什么，何时做。如果多个客户机希望同时访问相同的表，它们不必互相裁决和协商，只要发送自己的请求给服务器并让它仔细确定完成这些请求的顺序即可。

2、不必在数据库所在的机器上注册。MySQL 可以非常出色的在因特网上工作，因此您可以在任何位置运行一个客户机程序，只要此客户机程序可以连接到网络上的服务器。

当然不是任何人都可以通过网络访问你的 MySQL 服务器。MySQL 含有一个灵活而又有成效的安全系统，只允许那些有权限访问数据的人访问。而且可以保证用户只能够做允许他们做的事。

1.2.5 总结

本节简单介绍了一下关系数据库系统的基础知识，真正要详尽描述数据库系统的理论，完全可以写成几本书。读完本节之后，笔者希望读者能够了解什么是关系模型、SQL 语言以及 MySQL 的系统结构，这不仅对继续阅读本书有利，也可以在读者获取相关知识时，起到一个启发的作用。

1.3 MySQL 使用的 SQL 语言

MySQL 使用结构化查询语言（Structured Query Language，SQL）与服务器通讯。MySQL 系统使用的 SQL 语言基本上符合 SQL92 的标准，但是其对 SQL92 标准既有扩展，又有未

实现的地方。

1.3.1 表、列和数据类型

表是数据在一个 MySQL 数据库中的存储机制，如表 1-1 所示，它包含一组固定的列。表中的列描述该表所跟踪的实体的属性，每个列都有一个名字及各自的特性。

列由两部分组成：数据类型(datatype)和长度(length)。对于使用 NUMERIC 数据类型的列，可以指定列的小数位及精度特性，精度决定数值的有效位数，小数位表示数值的小数点位置。说明为 NUMERIC(9,2)的列表示该列总共有 9 位数，其中 2 位数在小数点右边。缺省的数值精度为 10 位数。

表 1-1　一个数据表 techers

Id	Name	Tel	Sex
1	Tom	62763218	M
2	Marry	21532777	F
3	Mike	45769021	M
4	Jerry	3245012	M
…	…	…	…

注意，在上表中你可能会有每行记录是按顺序记录的印象，假设你想取出表中的前十个记录。使用传统的编程语言，你可以做一个循环，取出前十个记录后结束循环。但使用标准的 SQL 查询，这是不可能实现的。因为在关系数据模型中，记录就是行是没有顺序的，也就是说，在一个表中不存在前十个记录这种概念。

1.3.2 函数

函数(function)是存储在数据库中的代码块。其差别在于函数可以把值返回调用程序。你可以建立自己的函数，并在 SQL 语句中调用它们，就像执行 Oracle 提供的函数一样。例如，MySQL 提供一个 SUBSTRING 函数来执行字符串上的“取子串”操作，如果创建一个叫做 MYSUB 的函数来执行一个自定义的取子串操作，就可以在一个 SQL 命令中调用它：

```
SELECT MYSUB("This is a test",6,2)
```

1.3.3 SQL 的语句

SQL 是一种典型的非过程化程序设计语言，这种语言的特点是：只指定哪些数据被操纵，至于对这些数据要执行哪些操作，以及这些操作是如何执行的，则未被指定。非过程化程序设计语言的优点在于它的简单易学，因此已经成为关系数据库访问和操纵数据的标准语言。

与之相对应的是过程化程序设计语言，我们平常熟悉的各种高级程序设计语言都属于这一范畴。这种语言的特点是：一条语句的执行是与其前后的语句和控制结构（如条件语句、循环语句等）相关的。与 SQL 相比，这些语言显得比较复杂，但优点是使用灵活，数

据操纵能力非常强大。

这种语言被设计为不允许你按照某种特定的顺序来取出记录，因为这样做会降低 SQL Sever 取记录的效率。使用 SQL，你只能按查询条件来读取记录。

当考虑如何从表中取出记录时，自然会想到按记录的位置读取它们。例如，也许你会尝试通过一个循环，逐个记录地扫描，来选出特定的记录。在使用 SQL 时，你必须训练自己，不要有这种思路。

假如你想选出所有的名字是“Tom”的记录，如果使用传统的编程语言，你也许会构造一个循环，逐个查看表中的记录，看名字域是否是“Tom”。

这种选择记录的方法是可行的，但是效率不高。使用 SQL，你只要说，“选择所有名字域等于 Tom 的记录”，SQL 就会为你选出所有符合条件的记录。SQL 会确定实现查询的最佳方法。

例如，我们从表 1-1 中取出 id 为 1 的数据：

```
SELECT * FROM teachers WHERE id=1
```

相同的功能用普通的高级语言，也许需要一个复杂的循环。

1.3.4 总结

在这一节中，简单介绍了 SQL 查询语言的基本知识，对于列数据类型、SQL 的各种语句，将在以后的章节中介绍，这是一个很长的过程。

1.4 MySQL 数据处理

MySQL 支持大量的列类型，它可以被分为 3 类：数字类型、日期和时间类型以及字符串(字符)类型。本节首先给出可用类型的一个概述，并且总结每个列类型的存储需求，然后提供每个类中的类型性质的更详细的描述。概述有意简化，更详细的说明应该考虑到有关特定列类型的附加信息，例如你能为其指定值的允许格式。

1.4.1 MySQL 的数据

MySQL 有几种数据类型，你应该了解它们是怎么使用的：

1.4.1.1、字符串值

字符串是类似”I like mysql.”和’MySQL is powerful.’等这样的值，它们既可以用双引号括起来，也可以是用单引号。

在字符串中不仅可以使用普通的字符，也可使用几个转义序列，它们用来表示特殊的字符，见表 1-2。每个转义序列以一个反斜杠（“\”）开始，指出后面的字符使用转义字符来解释，而不是普通字符。注意 NUL 字节与 NULL 值不同；NUL 为一个零值字节，而 NULL 代表没有值。

表 1-2　字符串转移序列表

序列	涵义
\0	一个 ASCII 0 (NUL)字符

\n	一个新行符
\r	一个回车符（Windows 中使用\r\n 作为新行标志）
\t	一个定位符
\b	一个退格符
\’	一个单引号(“’”)符
\”	一个双引号(“"”)符
\\	一个反斜线(“\”)符
\%	一个“%”符。它用于在正文中搜索“%”的文字实例，否则这里“%”将解释为一个通配符
_	一个“_”符。它用于在正文中搜索“_”的文字实例，否则这里“_”将解释为一个通配符

现在需要注意的是，如何在串中使用引号，你可以有多种办法：

- 如果串是用相同的引号括起来的，那么在串中需要引号的地方重复写该引号即可。
- 如果串是用另外的引号括起来的，则不需要双写相应引号，直接在串中使用，该引号不被特殊对待。
- 使用反斜杠，用转移序列的方式表示；这种方法不去管用来将串括起的是单引号还是双引号。

例如，下面语句的结果是：

```
mysql> SELECT 'hello', '"hello"', '""hello""', 'hel''lo', '\'hello';
+-------+---------+-----------+-------+---------+
| hello | "hello" | ""hello"" | hel'lo | 'hello |
+-------+---------+-----------+-------+---------+
```

1.4.1.2 数字值

MySQL 中的数字是类似于 100 或 3.1215936 这样的值。MySQL 支持说明为整数（无小数部分）或浮点数（有小数部分）的值：

- 整数由数字序列组成。浮点数由一个阿拉伯数字序列、一个小数点和另一个阿拉伯数字序列组成。两个阿拉伯数字序列可以分别为空，但不能同时为空。
- MySQL 支持科学表示法。科学表示法由整数或浮点数后跟“e”或“E”、一个符号（“+”或“-”，必须具有）和一个整数指数来表示。
- 数值前可放一个负号“-”以表示负值。

例如，下面都是合法的数值：

整数值：1221　0　-32

浮点数：294.42　-32032.6809e+10　148.

下面的值是错误的：

1.34E12

1.4.1.3 十六进制值

MySQL 支持十六进制值。以十六进制形式表示的整数由“0x”后跟一个或多个十六进制数字（”0”到“9”及“a”到“f”）组成。例如，0x0a 为十进制的 10，而 0xffff 为十进制的 65535。十六进制数字不区分大小写，但其前缀“0x”不能为“0X”。即 0x0a 和 0x0A 都是合法的，但 0X0a 和 0X0A 不是合法的。

在数字上下文，它们表现类似于一个整数(64 位精度)。在字符串上下文，它们表现类似于一个二进制字符串，这里每一对十六进制数字被变换为一个字符。

例如：

```
mysql> select 0x5061756c+0, 0x5061756c;
```

其结果为：

```
+-------------+-----------+
| 0x5061756c+0 | 0x5061756c |
+-------------+-----------+
|   1348564332 | Paul       |
+-------------+-----------+
```

1.4.1.4 日期和时间值

日期和时间值是一些类似于“1999-06-17”或“12:30:43”这样的值。MySQL 还支持日期/时间的组合，如“1999-06-17 12:30:43”。

需要要特别注意的是，MySQL 是按年-月-日的顺序表示日期的。这可能与你学过的高级语言不同。

1.4.1.4 NULL 值

NULL 值可适用于各种列类型，它通常用来表示“没有值”、 “无数据”等意义，并且不同于例如数字类型的 0 为或字符串类型的空字符串。

1.4.2 列类型概述

数据库中的每个表都是由一个或多个列构成的。可以用 CREATE TABLE 语句创建一个表，创建表时要为每列指定一个类型。列的类型与数据类型相对应，但是比数据类型更为具体，用列类型描述表列可能包含的值的种类以及范围，列的值必须符合规定，不能包含对应的数据类型所允许的所有值。例如，CHAR（16）就规定了存储的字符串值必须是 16 位。当然不是你必须存储 16 个字符，而是指列在表中要占 16 个字符的宽度的。

MySQL 的列类型是一种手段，通过这种手段可以描述一个表列包含什么类型的值，这又决定了 MySQL 怎样处理这些值。例如，数值值既可用数值也可用串的列类型来存放，但是根据存放这些值的类型，MySQL 对它们的处理将会有些不同。每种列类型都有几个特性如下：

- 其中可以存放什么类型的值。
- 值要占据多少空间，以及该值是否是定长的（所有值占相同数量的空间）或可变长的（所占空间量依赖于所存储的值）。

- 该类型的值怎样比较和存储。
- 此类型是否允许 NULL 值。
- 此类型是否可以索引。

下面是创建一个表的例子：

```
CREATE TABLE teacher
(
id TINYINT UNSIGNED NOT NULL,
name CHAR(16) NOT NULL,
tele NUMERIC(8),
sex ENUM("F","M") DEFAULT "M"
)
```

由上面这个例子可以知道，创建列类型的语法是：

```
col_name col_type [col_attributes][general_attributes]
```

- col_name 列的名字
- col_type 列类型，控制存储在列中的数据类型
- col_attributes 专用属性，只能应用于制定列，例如，我们还不知道的 BINARY。如果你使用专用属性，必须在列的类型之后，列的通用属性之前。
- general_attributes 通用属性，可以应用在出少数列的任意列，例如上面提到了 NULL、NOT NULL、和 DEFAULT。

我们将简要地考察一下 MySQL 列类型以获得一个总的概念，然后更详细地讨论描述每种列类型的属性。

MySQL 的数字类型如表 1-3 所示，可以包括浮点类型和整数类型：

表 1-3 MySQL 的数字列类型

类型名	涵义
TINYINT	一个很小的整数
SMALLINT	一个小整数
MEDIUMINT	一个中等大小整数
INT、INTEGER	一个正常大小整数
BIGINT	一个大整数
FLOAT	一个小(单精密)浮点数字
DOUBLE	一个正常大小(双精密)浮点数字
DOUBLE PRECISION	
REAL	
DECIMAL	一个未压缩(unpack)的浮点数字，“未压缩”意味着数字作为一个字符串被存储
NUMERIC	

MySQL 的字符串类型如表 1-4 所示，串类型中不仅可以存储字符串，实际上任何二进制数据，例如，图象、音频、视频等，都可以存储在串类型中。

表 1-4　MySQL 的字符串列类型

类型名	涵义
CHAR	一个定长字符串
VARCHAR	一个变长字符串
TINYBLOB	最大长度为 255(2^8-1)个字符的 BLOB 或 TEXT 列
TINYTEXT	
BLOB	最大长度为 65535(2^16-1)个字符的 BLOB 或 TEXT 列
TEXT	
MEDIUMBLOB	最大长度为 16777215(2^24-1)个字符的 BLOB 或 TEXT 列
MEDIUMTEXT	
LONGBLOB	最大长度为 4294967295(2^32-1)个字符的 BLOB 或 TEXT 列
LONGTEXT	
ENUM('value1','value2',...)	枚举：列只能赋值为某个枚举成员或 NULL
SET('value1','value2',...)	集合：列可以赋值为多个集合成员或 NULL

MySQL 的日期与时间类型如表 1-5 所示。MySQL 允许你存储某个“不严格地”合法的日期值，例如 1999-11-31，原因我们认为它是应用程序的责任来处理日期检查，而不是 SQL 服务器。为了使日期检查更“快”，MySQL 仅检查月份在 0-12 的范围，天在 0-31 的范围。

表 1-5　MySQL 的时间和日期列类型

类型名	涵义
DATE	一个日期，以'YYYY-MM-DD'格式来显示
TIME	一个日期和时间组合，以'YYYY-MM-DD HH:MM:SS'格式来显示
DATETIME	最大长度为 255(2^8-1)个字符的 BLOB 或 TEXT 列
TIMESTAMP	一个时间戳，以 YYYMMDDHHMMSS 格式来显示

1.4.3 数字列类型

MySQL 支持所有的 ANSI/ISO SQL92 的数字类型。这些类型包括准确数字的数据类型(NUMERIC, DECIMAL, INTEGER,和 SMALLINT)，也包括近似数字的数据类型(FLOAT, REAL,和 DOUBLE PRECISION)。关键词 INT 是 INTEGER 的一个同义词，而关键词 DEC 是 DECIMAL 一个同义词。

MySQL 的数字列类型有两种：

- 整型 MySQL 提供了五种整型 TINYINT、SAMLLINT、MEDIUMINT、INT 和 BIGINT。整数列可以用 UNSIGNED 禁用负数值。
- 浮点型 MySQL 提供了三种浮点型，FLOAT、DOUBLE 和 DECIMAL。

下面，我们将详细描述数字类型的定义，取值范围和存储要求。

1.4.3.1 整数类型

MySQL 支持所有的 ANSI/ISO SQL92 的数字类型，其中的整数类型为 INTEGER,和 SMALLINT，关键词 INT 是 INTEGER 的一个同义词：

SMALLINT[(M)] [UNSIGNED] [ZEROFILL]

取值范围：有符号：-32768 到 32767（-215 到 215-1）无符号：0 到 65535（0 到 216-1）

存储要求：2 个字节

INT[(M)] [UNSIGNED] [ZEROFILL]、INTEGER[(M)] [UNSIGNED] [ZEROFILL]

取值范围：有符号：-2147483648 到 2147483647（-231 到 231-1）无符号：0 到 4294967295（0 到 232－1）

存储要求：4 个字节

作为对 ANSI/ISO SQL92 标准的扩展，MySQL 也支持上表所列的整型类型 TINYINT、MEDIUMINT 和 BIGINT：

TINYINT[(M)] [UNSIGNED] [ZEROFILL]

取值范围：有符号：-128 到 127（-27 和 27－1）无符号：0 到 255（0 到 28-1）

存储要求：1 个字节

MEDIUMINT[(M)][UNSIGNED] [ZEROFILL]

取值范围：有符号：-8388608 到 8388607（-223 到 223-1）无符号：0 到 16777215（0 到 224－1）

存储要求：3 个字节

BIGINT[(M)] [UNSIGNED] [ZEROFILL]

取值范围：有符号：-9223372036854775808 到 9223372036854775807（-263 到 263-1）无符号：0 到 18446744073709551615（0 到 264－1）

存储要求： 8 个字节

在为列选择了使用某种数值类型时，除了要考虑数据的类型外，还应该注意所要表示的值的范围和存储需求，只需选择能覆盖要取值的范围的最小类型即可。选择较大类型会对空间造成浪费，使表不必要地增大，处理起来没有选择较小类型那样有效。对于整型值，如果数据取值范围较小，则 TINYINT 最合适。MEDIUMINT 和 INT 虽然能表示能够表示更大的数值并且可用于更多类型的值，但存储代价更大。BIGINT 在全部整型中取值范围最大，而且需要的存储空间是表示范围次大的整型 INT 类型的两倍，因此只在确实需要时才用。

作为对 ANSI/ISO SQL92 标准的的另一个扩展， MySQL 支持可选地指定一个整型值显示的宽度，用括号跟在基本关键词之后(例如，INT(4))。这个可选的宽度指定被用于其宽度小于列指定宽度的值得左填补显示，但是不限制能在列中被存储的值的范围；如果某个特定值的可打印表示需要不止 M 个字符，则显示完全的值；不会将值截断以适合 M 个字符。当与可选的扩展属性 ZEROFILL 一起使用时，缺省的空格填补用零代替。例如，对于声明为 INT(5) ZEROFILL 的列，一个为 4 的值作为 00004 被检索。如果定义了一个没有明确宽度的整数列，将会自动分配给它一个缺省的宽度。缺省值为每种类型的“最长”值的长度。

我们可以这样知道 M 和 D 的缺省值。先创建一个表

```
CREATE TABLE number
(
    tiny TINYINT,u_tiny TINYINT UNSIGNED,
    small SMALLINT,u_small SMALLINT UNSIGNED,
    medium MEDIUMINT,u_medium MEDIUMINT UNSIGNED,
    num INT,u_num INT UNSIGNED,
    big BIGINT,u_big BIGINT UNSIGNED
)
```

然后显示这个表的结构

```
DESCRIBE number
```

在 MySQL3.23 上的结果如下：

```
+---------+--------------------+
| Field   | Type               |
+---------+--------------------+
| tiny    | tinyint(4)         |
| u_tiny  | tinyint(3) unsigned |
| small   | smallint(6)        |
| u_small | smallint(5) unsigned |
| medium  | mediumint(9)       |
| u_medium | mediumint(8) unsigned |
```

```
| num      | int(11)              |
| u_num    | int(10) unsigned     |
| big      | bigint(20)           |
| u_big    | bigint(20) unsigned  |
+---------+----------------------+
```

1.4.3.2 浮点数类型

MySQL 支持所有的 ANSI/ISO SQL92 的浮点数字类型。这些类型包括准确数字的数据类型(NUMERIC, DECIMAL)，也包括近似数字的数据类型(FLOAT, REAL,和 DOUBLE PRECISION)。关键词 DEC 是 DECIMAL 一个同义词：

FLOAT[(M,D)] [ZEROFILL]

取值范围：-3.402823466E+38 到 -1.175494351E-38，0 和 1.175494351E-38 到 3.402823466E+38

存储要求：4 个字节

DOUBLE[(M,D)] [ZEROFILL]、DOUBLE PRECISION[(M,D)] [ZEROFILL]

取值范围：-1.7976931348623157E+308 到 -2.2250738585072014E-308、0 和 2.2250738585072014E-308 到 1.7976931348623157E+308

存储要求：8 个字节

DECIMAL(M[,D]) [ZEROFILL]、NUMERIC(M[, D]) [ZEROFILL]

取值范围：实际的范围可以通过 M 和 D 的选择被限制

存储要求：M 字节（低于 3.23 版本）M+2 字节（3.23 或更高版本）

NUMERIC 和 DECIMAL 类型

表 1-6 M 和 D 对 DECIMAL 类型取值范围的影响

类 型 说 明	取值范围（MySQL < 3.23）	取值范围（MySQL * 3.23）
DECIMAL(4, 1)	-9.9 到 99.9	-999.9 到 9999.9
DECIMAL(5, 1)	-99.9 到 999.9	-9999.9 到 99999.9
DECIMAL(6, 1)	-999.9 到 9999.9	-99999.9 到 999999.9
DECIMAL(6, 2)	-99.99 到 999.99	-9999.99 到 99999.99
DECIMAL(6, 3)	-9.999 到 99.999	-999.999 到 9999.999

给定的 DECIMAL 类型的取值范围取决于 MySQL 的版本。

对于 MySQL 3.23 以前的版本，DECIMAL(M, D) 列的每个值占用 M 字节，而符号（如果需要）和小数点包括在 M 字节中。因此，类型为 DECIMAL(5,2) 的列，其取值范围为 -9.99 到 99.99，因为它们覆盖了所有可能的 5 个字符的值。

对于 MySQL 3.23，DECIMAL 值是根据 ANSI 规范进行处理的，ANSI 规范规定 DECIMAL(M, D) 必须能够表示 M 位数字及 D 位小数的任何值。例如，DECIMAL(5, 2) 必须能够表示从 -999.99 到 999.99 的所有值。而且必须存储符号和小数点，因此自 MySQL 3.23 以来 DECIMAL 值占 M + 2 个字节。对于 DECIMAL(5, 2)，“最长”的值（-999.99）需要 7 个字节。在正取值范围的一端，不需要正号，因此 MySQL 利用它扩充了取值范围，使其超过了 ANSI 所规范所要求的取值范围。如 DECIMAL(5, 2) 的最大值为 9999.99，因为有 7 个字节可用。

简而言之，在 MySQL 3.23 及以后的版本中，DECIMAL(M, D) 的取值范围等于更早版本中的 DECIMAL(M + 2, D) 的取值范围。

1.4.4 日期和时间类型

MySQL 提供几种时间和日期类型，包括日期和时间类型是 DATETIME、DATE、TIMESTAMP、TIME 和 YEAR。对这几种时间和日期类型概述如下：

DATA

取值范围：“1000-01-01”到“9999-12-31”

存储需求：3 字节

TIME

取值范围：“-838:59:59”到“838:59:59”

存储需求：3 字节

DATATIME

取值范围：“1000-01-01 00:00:00”到“9999-12-31 23:59:59”

存储需求：8 字节

TIMESTAMP[(M)]

取值范围：“19700101000000”到 2037 年的某个时刻

存储需求：4 字节

YEAR[(M)]

取值范围：1901 到 2155

存储需求：1 字节

1.4.4.1 Y2K 问题和日期类型

MySQL 本身 Y2K 安全的，但是呈交给 MySQL 的输入值可能不是。一个包含 2 位年份值的任何输入是由二义性的，因为世纪是未知的。这样的值必须被解释成 4 位形式，因为 MySQL 内部使用 4 位存储年份。

对于 DATETIME, DATE, TIMESTAMP 和 YEAR 类型，MySQL 使用下列规则的解释二义性的年份值：

- 在范围 00-69 的年值被变换到 2000-2069。
- 在范围 70-99 的年值被变换到 1970-1999。

记得这些规则仅仅提供对于你数据的含义的合理猜测。如果 MySQL 使用的启发规则不产生正确的值，你应该提供无二义的包含 4 位年值的输入。

1.4.4.2 DATETIME,DATE 和 TIMESTAMP 类型

DATETIME, DATE 和 TIMESTAMP 类型是相关的。本节描述他们的特征，他们是如何类似的而又不同的。

DATETIME 类型用在你需要同时包含日期和时间信息的值时。MySQL 检索并且以'YYYY-MM-DD HH:MM:SS'格式显示 DATETIME 值，支持的范围是'1000-01-01 00:00:00'到'9999-12-31 23:59:59'。（“支持”意味着尽管更早的值可能工作，但不能保证他们可以。）

DATE 类型用在你仅需要日期值时，没有时间部分。MySQL 检索并且以'YYYY-MM-DD'格式显示 DATE 值，支持的范围是'1000-01-01'到'9999-12-31'。

TIMESTAMP 列类型提供一种类型，你可以使用它自动地用当前的日期和时间标记 INSERT 或 UPDATE 的操作。如果你有多个 TIMESTAMP 列，只有第一个自动更新。TIMESTAMP 值可以从 1970 的某时的开始一直到 2037 年，精度为一秒，其值作为数字显示。

自动更新第一个 TIMESTAMP 列在下列任何条件下发生：

- 列没有明确地在一个 INSERT 或 LOAD DATA INFILE 语句中指定。
- 列没有明确地在一个 UPDATE 语句中指定且一些另外的列改变值。（注意一个 UPDATE 设置一个列为它已经有的值，这将不引起 TIMESTAMP 列被更新，因为如果你设置一个列为它当前的值，MySQL 为了效率而忽略更改。）
- 你明确地设定 TIMESTAMP 列为 NULL.

例如，我们创建一个表：

```
CREATE TABLE my_test
(
    id INT,
    ts TIMESTAMP
)
```

然后，用如下语句录入数据：

```
INSERT my_test VALUES(1,20010101000000)
INSERT my_test(id) VALUES(2)
```

然后，查询表中的内容：

```
SELECT * from my_test
```

你可以看到内容：

```
+------+---------------+
| id   | ts            |
```

```
+------+---------------+
|    1 | 20010101000000 |
|    2 | 20010113165713 |
+------+---------------+
```

1.4.4.3 TIME 类型

MySQL 检索并以'HH:MM:SS'格式显示 TIME 值(或对大小时值，'HHH:MM:SS'格式)。TIME 值的范围可以从'-838:59:59'到'838:59:59'。小时部分可能很大的的原因是 TIME 类型不仅可以被使用在表示一天的时间(它必须是不到 24 个小时)，而且用在表示在 2 个事件之间经过的时间或时间间隔(它可以是比 24 个小时大些，或甚至是负值)。

你能用多中格式指定 TIME 值：

- 作为'HH:MM:SS'格式的一个字符串。“宽松”的语法被允许--任何标点符号可用作时间部分的分隔符，例如，'10:11:12'和'10.11.12'是等价的。
- 作为没有分隔符的'HHMMSS'格式的一个字符串，如果它作为一个时间解释。例如，'101112'被理解为'10:11:12'，但是'109712'是不合法的(它有无意义的分钟部分)并变成'00:00:00'。
- 作为 HHMMSS 格式的一个数字，如果它能解释为一个时间。例如，101112 被理解为'10:11:12'。

1.4.4.4 YEAR 类型

YEAR 类型是一个有效的利用 1 字节类型表示年份。MySQL 检索并且以 YYYY 格式显示 YEAR 值，其范围是 1901 到 2155。 如果，只想保存日期，那么 YEAR 比其它类型比如 SAMLLINT 更为有效。

你能用多种格式指定 YEAR 值，既可以用 4 为字符，也可以使用 4 为字符串，当然要在 1901 到 2155 范围之内。

作为 YEAR 的一个优点是，你可以指定一个在'00'到'99'范围的 2 位字符串或者一个在'00'到'69'和'70'到'99'范围的值被变换到在 2000 到 2069 范围和 1970 到 1999 的 YEAR 值。

1.4.5 字符串类型

MySQL 提供的字符串类型包括 CHAR、VARCHAR、BLOB、TEXT、ENUM 和 SET。对这些类型作一个简要的叙述如下：

CHAR(M) [BINARY]

一个定长字符串，当存储时，总是是用空格填满右边到指定的长度。在 MySQL3.23 以前的版本，M 的范围是 1 ~ 255 个字符，在 MySQL3.23 版中，M 值的范围是 0 ~ 255 个字符.当值被检索时，空格尾部被删除。CHAR 类型在排序和比较时不区分大小写，除非给出 BINARY 关键词。NATIONAL CHAR（短形式 NCHAR）是 ANSI SQL 的方式来定义 CHAR 列应该使用缺省字符集。这是 MySQL 的缺省。CHAR 是 CHARACTER 的一个缩写。CHAR(0) 可以用于在希望定义一个列，但由于尚不知道其长度，所以不想给其分配空间的情况下，CHAR(0) 列作为占位符很有用处。以后可以用 ALTER TABLE 来加宽这个列。

存储需求：M 字节

[NATIONAL] VARCHAR(M) [BINARY]

一个变长字符串。注意：当值被存储时，尾部的空格被删除(这不同于 ANSI SQL 规范)。M 的范围是 1 ~ 255 个字符。 VARCHAR 类型在排序和比较时不区分大小写，除非给出 BINARY 关键词。VARCHAR 是 CHARACTER VARYING 一个缩写。

存储需求：L+1 字节（L 是存储实际值需要的长度，1 为存储该值实际长度）

TINYBLOB、TINYTEXT

一个 BLOB 或 TEXT 列，最大长度为 255(2^8-1)个字符。

存储需求：L+1 字节

BLOB、TEXT

一个 BLOB 或 TEXT 列，最大长度为 65535(2^16-1)个字符。

存储需求：L+2 字节

MEDIUMBLOB、MEDIUMTEXT

一个 BLOB 或 TEXT 列，最大长度为 16777215(2^24-1)个字符。

存储需求：L+3 字节

LONGBLOB、LONGTEXT

一个 BLOB 或 TEXT 列，最大长度为 4294967295(2^32-1)个字符。

存储需求：L+4 字节

ENUM('value1','value2',...)

枚举。一个仅有一个值的字符串对象，这个值式选自与值列表'value1'、'value2', ...，或 NULL。一个 ENUM 最多能有 65535 不同的值。

存储需求：1 或 2 字节

SET('value1','value2',...)

一个集合。能有零个或多个值的一个字符串对象，其中每一个必须从值列表'value1', 'value2', ...选出。一个 SET 最多能有 64 个成员。

存储需求：1、2、3、4 或 8 字节

在某种意义上，串实际是一种非常“通用”类型，因为可用它们来表示任意值，不仅仅是字符串。例如，可用串类型来存储二进制数据，如图像、视频或音频。

对于所有串类型，都要剪裁过长的值使其适合于相应的串类型。但是串类型的取值范围很不同，有的取值范围很小，有的则很大。取值大的串类型能够存储近 4GB 的数据。因此，应该使串足够长以免您的信息被切断（由于受客户机/服务器通信协议的最大块尺寸限制，列值的最大限额为 24MB）。

另外，对于串类型，在比较时是忽略大小写的，使用 BINARY 关键字，则比较时采用 ASCII 码的方式，即不再忽略大小写。可以使用 BINARY 的串类型为 CHAR 和 VARCHAR。

1.4.5.1 CHAR 和 VARCHAR 类型

CHAR 和 VARCHAR 类型是类似的，但是在他们被存储和检索的方式不同。 其具体的异同为：

- 当给定一个 CHAR 列的值时，其长度将被被修正为在你创建表时所声明的长度。长度可以是 1 和 255 之间的任何值。（在 MySQL 3.23 中，CHAR 长度可以是 0～255。) 当 CHAR 值被存储时，他们被用空格在右边填补到指定的长度。当 CHAR 值被检索时，拖后的空格被删去。
- 在 VARCHAR 列中的值是变长字符串。你可以声明一个 VARCHAR 列是在 1 和 255 之间的任何长度，就像对 CHAR 列。然而，与 CHAR 类型相反，VARCHAR 值只存储所需的字符，外加一个字节记录长度，值不被填补；相反，当值被存储时，拖后的空格被删去。（这个空格删除不同于 ANSI SQL 规范。）
- 如果你把一个超过列最大长度的值赋给一个 CHAR 或 VARCHAR 列，值被截断以适合串类型。

例如，我们用下表来说明存储一系列不同的串值到 CHAR(4)和 VARCHAR(4)列的结果：

表 1-7　CHAR 类型和 VARCHAR 类型的对比

值	CHAR(4)	存储需求	VARCHAR(4)	存储需求
''	'　　'	4 字节	''	1 字节
'ab'	'ab　'	4 字节	'ab'	3 字节
'abcd'	'abcd'	4 字节	'abcd'	5 字节
'abcdefgh'	'abcd'	4 字节	'abcd'	5 字节

虽然实际存储的值并不一样，但是查询时，这两种类型是一致的，因为 CHAR(4)类型多于的空格将被忽略。

需要注意的是，除了少数情况外，在同一个表中不能混用 CHAR 和 VARCHAR 这两种类型，你只能使用其中之一。如果你创建表时，包括这两种类型，在一般情况下，MySQL 会将列从一种类型转换为另一种类型。这样做的原因如下：

- 行定长的表比行可变长的表容易处理，效率更高。
- 只有所有的类型是定长时，行才是定长的，才能提高性能。
- 有时为了节省存储空间，使用了变长类型，在这种情况下最好也将定长列转换为

可变长列。

这表示，如果表中有 VARCHAR 列，那么表中不可能同时有 CHAR 列；MySQL 会自动地将它们转换为 VARCHAR 列。

转换的规则：

- 长度小于 4 的 VARCHAR 被改变为 CHAR。
- 如果在一个表中的任何列有可变长度，结果是整个行是变长的。因此，如果一张表包含任何变长的列(VARCHAR、TEXT 或 BLOB)，所有大于 3 个字符的 CHAR 列被改变为 VARCHAR 列。

例如，我们创建下面一个表：

```
CREATE TABLE ch_type
(
    ch1 char(3),
    ch2 varchar(3),
    ch3 char(4),
    ch4 varchar(4)
)
```

然后查看表的结构：

```
DESCRIBE ch_type
```

在 MySQL3.23 上结果为：

```
+------+----------+
| Field | Type     |
+------+----------+
| ch1   | char(3)    |
| ch2   | char(3)    |
| ch3   | varchar(4) |
| ch4   | varchar(4) |
+------+----------+
```

1.4.5.2 BLOB 和 TEXT 类型

一个 BLOB 是一个能保存可变数量的数据的二进制的大对象。4 个 BLOB 类型 TINYBLOB、BLOB、MEDIUMBLOB 和 LONGBLOB 仅仅在他们能保存值的最大长度方面有所不同。

4 个 TEXT 类型 TINYTEXT、TEXT、MEDIUMTEXT 和 LONGTEXT 对应于 4 个 BLOB 类型，并且有同样的最大长度和存储需求。在 BLOB 和 TEXT 类型之间的唯一差别是对 BLOB 值的排序和比较以大小写敏感方式执行，而对 TEXT 值是大小写不敏感的。换句话说，一个 TEXT 是一个大小写不敏感的 BLOB。

如果你把一个超过列类型最大长度的值赋给一个 BLOB 或 TEXT 列，值被截断以适合它。

在大多数方面，你可以认为一个TEXT 行列是你所希望大的一个 VARCHAR 列。同样，

你可以认为一个 BLOB 列是一个 VARCHAR BINARY 列。差别是：

- 用 MySQL 版本 3.23.2 和更新，你能在 BLOB 和 TEXT 列上索引。更旧的 MySQL 版本不支持这个。
- 当值被存储时，对 BLOB 和 TEXT 列没有拖后空格的删除，因为对 VARCHAR 列有删除。
- BLOB 和 TEXT 列不能有 DEFAULT 值。

由于，BLOB 和 TEXT 类型可以存储非常多的数据，因此使用 BLOB 和 TEXT 类型需要注意的是：

- BLOB 或 TEXT 列在 MySQL 3.23 以上版本中可以进行索引，虽然在索引时必须指定一个用于索引的约束尺寸，以免建立出很大的索引项从而抵消索引所带来的好处。
- 由于 BLOB 和 TEXT 值的大小变化很大，如果进行的删除和更新很多，则存储它们的表出现高碎片率会很高。应该定期地运行 OPTIMIZE TABLE 减少碎片率以保持良好的性能。

1.4.5.3 ENUM 和 SET 类型

1、ENUM 和 SET 类型的定义

ENUM 和 SET 类型是两种特殊的字符串类型，它们有很多相似之处，使用方法也是分类系，通常都从一个在表创建时明确列举的允许值的一张表中选择，其主要的区别是 ENUM 列必须是值集合中的一个成员，而 SET 列可以包括其中的任意成员。

例如，创建如下两个串列：

color ENUM(“red”,”black”,”green”,”yellow”)

property SET(“car”,”house”,”stock”) NOT NULL

那么 color 和 property 可能的值分别为：

color：NULL、”red”、”black”、”green”和”yellow”

而 property 可能的值就复杂的多：

“”

“car”

“house”

“car,house”

“stock”

“car,stock”

“house,stock”

“car,house,stock”

由于空串可以表示不具备值的集合的任何一个值，所以这也是一个合法的 SET 值。

ENUM 类型可以有 65536 个成员，而 SET 类型最多可以有 64 个成员。

2、ENUM 和 SET 类型是如何存储的

ENUM 和 SET 类型在数据库内部并不是用字符的方式存储的，而是使用一系列的数

字，因此更为高效。

ENUM 和 SET 类型的合法值列表的原则为：

- 此列表决定了列的可能合法值。
- 可按任意的大小写字符插入 ENUM 或 SET 值，但是列定义中指定的串的大小写字符决定了以后检索它们时的大小写。
- 在 ENUM 定义中的值顺序就是排序顺序。SET 定义中的值顺序也决定了排序顺序，但是这个关系更为复杂，因为列值可能包括多个集合成员。
- SET 定义中的值顺序决定了在显示由多个集合成员组成的 SET 列值时，子串出现的顺序。

对于 ENUM 列类型，成员是从 1 开始顺序编号的。（0 被 MySQL 用作错误成员，如果以串的形式表示就是空串。）枚举值的数目决定了 ENUM 列的存储大小。一个字节可表示 256 个值，两个字节可表示 65 536 个值。因此，枚举成员的最大数目为 65 536（包括错误成员），并且存储大小依赖于成员数目是否多于 256 个。在 ENUM 定义中，可以最多指定 65 535（而不是 65 536）个成员，因为 MySQL 保留了一个错误成员，它是每个枚举的隐含成员。在将一个非法值赋给 ENUM 列时，MySQL 自动将其换成错误成员。

对于 SET 类型，SET 列的集合成员不是顺序编号的，而是每个成员对应 SET 值中的一个二进制位。第一个集合成员对应于 0 位，第二个成员对应于 1 位，如此等等。数值 SET 值 0 对应于空串。SET 成员以位值保存。每个字节的 8 个集合值可按此方式存放，因此 SET 列的存储大小是由集合成员的数目决定的，最多 64 个成员。对于大小为 1 到 8、9 到 16、17 到 24、25 到 32、33 到 64 个成员的集合，其 SET 值分别占用 1、2、3、4 或 8 个字节。

例如，还是上面的例子，我们从一个表中检索出 ENUM 和 SET 列的值，及其对应的数值：

对于 ENUM 类型的 color 列

```
SELECT color,color+0 from my_table
```

其结果为：

```
+--------+--------+
| color  | color+0 |
+--------+--------+
| NULL   |    NULL |
| black  |       2 |
| green  |       3 |
| yellow |       4 |
| red    |       1 |
| red    |       1 |
| green  |       3 |
| green  |       3 |
| yellow |       4 |
```

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

对于 SET 类型的 property 列，同样的

SELECT property,property+0 FROM my_table;

其结果为

```
+----------------+-----------+
| property       | property+0 |
+----------------+-----------+
|                |          0 |
| house,stock    |          6 |
| car,stock      |          5 |
| stock          |          4 |
| car,house,stock |          7 |
| car,house      |          3 |
| house          |          2 |
| car,stock      |          5 |
| house,stock    |          6 |
+----------------+-----------+
```

你可以仔细了解它们之间的对应关系。

因此，在给列赋值、检索时，你不仅可以使用值表中的字符串，也可以使用数值来表示一个值，例如下列语句是等价的：

INSERT my_table SET property=’car,house,stock’

INSERT my_table SET property=7

对于 ENUM 列也同样如此：

INSERT my_table SET color=’red’;

INSERT my_table SET color=1

1.4.6 总结

本节对 MySQL 的数据类型和列类型进行了简单的描述。这一部分是我们继续学习 SQL 语言的基础，但是如果你作为一个初学者可能无法立刻理解本节的内容，没有关系，你完全可以略过，继续下去，相信在读完本书的后几章之后，在返回来，就会容易理解了。

本节中出现大量的 SQL 语句，尤其是建表的语句。你可能现在还不知道如何完成这些查询，没有关系，你只需要理解其中的含义就可以了。阅读了以后的几章，你就会理解这一切。

第2章

MySQL 的安装

本章要点：

- MySQL 的安装布局
- 如何安装 MySQL
- 如何为 MySQL 服务器设定参数
- 如何为一个 MySQL 安装升级

本章介绍 MySQL 安装和升级的知识。由于 MySQL 可以在多种平台上使用，并且即使在同一平台上，也存在多种不同的安装分发，因此，MySQL 的安装存在多种方法供你选择。

相对于其它的多数关系数据库系统，MySQL 的安装并不复杂。本章将各种安装的说明详细记载一起，会为你安装系统提供方便。当然，还是建议你仔细阅读安装包中的说明，因为对于不同版本安装可能有的特殊性，本章无法兼顾和预期。另外，安装包中的文档相对也详尽。

MySQL 在 Linux 上最为常用。Linux 上提供各种 RPM 文件，可以非常方便的安装 MySQL 数据库系统。如果你喜欢编译源代码或者 RPM 分发无法在你的平台使用，你可以使用源代码安装，或者还有二进制分发可以采用。

2.1 MySQL 系统的安装布局

在说明如何安装 MySQL 数据库系统之前，读者有必要首先了解一下 MySQL 的安装布局，这样才能有的放矢，更容易理解后面的叙述。这节描述安装二进制代码和源代码分发时创建的缺省目录布局。

2.1.1 二进制安装

二进制分发通过在你选择的安装地点(典型的“/usr/local/mysql”)解压缩来安装，并且在该处创建下列目录:

表 2-1 二进制分发的安装布局

目录	目录的内容
Bin	客户程序和 mysqld 服务器
Data	日志文件, 数据库
include	包含(头)文件
Lib	库文件
scripts	mysql_install_db
share/mysql	错误消息文件
sql-bench	基准程序

2.1.2 RPM 安装

rpm 安装本质上也属于二进制分发，因此安装布局同上。不同之处在于它将自动安装启动脚本到/etc/rc.local 目录中，并且缺省时，MySQL 随系统的启动而自动启动，安装十分方便。

对于 RPM 分发程序、库文件、头文件和配置文件，分别安装 Red Hat Linu 标准的目录/usr/bin、/usr/lib/mysql、/usr/include/mysql 和/etc/mysql 等处。

需要特别注意的是数据库目录的位置，缺省时，/var/lib/mysql

2.1.3 源代码安装

源代码分发在你配置并编译后进行安装。缺省地，安装步骤将文件安装在“/usr/local”下的下列子目录：

表 2-2 二源代码分发的安装布局

目录	目录的内容
bin	客户程序和脚本
include/mysql	包含(头)文件
Info	Info 格式的文档
lib/mysql	库文件

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，源代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“/usr/local/bin”目录而非“/usr/local/mysql/scripts”内。
- 头文件和库目录是“include/mysql”和“lib/mysql”而非“include”和“lib”。

2.1.4 总结

本节介绍了 MySQL 系统的安装布局，对于源代码分发、二进制分发和 RPM 分发都做了阐述。阅读本章，有利于你对 MySQL 有一个清楚的概念，在下几节中，对于诸多繁琐的安装过程，就不再详述安装布局的内容，而是直接使用，读者应该不会感到突兀。

2.2 安装 MySQL 系统的分发

MySQL 是一个复杂的系统，因此相对于其它普通软件的安装，其过程要复杂、困难一些，本节将指导您完成这一过程。

2.2.1 在 Windows 下的安装一个二进制分装

在 windows 下安装一个 MySQL 的二进制分发十分的容易。首先你必须通过前文所述的方法获得相应的软件包，其文件名应该类似于 mysql-x.xx.xx-beta-win.zip 或者 mysql-x.xx.xx-win.zip 。如果你下载的是共享版本，应该类似于 mysq-shareware-x.xx.xx-win.zip。由于 MySQL 的 windows 版本必修付费使用，共享版本有少许功能限制，因此如果只是用来学习数据库的使用，建议安装测试版本。

安装 MySQL 首先在某个空目录解压缩安装文件，并且运行 Setup.exe 程序。

缺省地，MySQL-Win32 被配置安装在“C:\mysql”。如果你想要在其它地方安装 MySQL，在“C:\mysql”安装它，然后将安装移到你想要的地方。或者你也可以在安装时选择其它目录。如果你真的移走 MySQL，你可能需要通过提供选项给 mysqld 告诉 mysqld 需要的参数信息，尤其是数据库目录的位置。

使用 C:\mysql\bin\mysqld --help 显示所有的参数选项的信息。例如，如果你移动 MySQL 分发到“D:\programs\mysql”，你可以用 D:\programs\mysql\bin\mysqld --basedir D:\programs\mysql 来启动 mysqld。当然有时你不必这么做，服务器也可以正常运作，不过这是一个保险的方法。

用登记的 MySQL 版本,你也可以创建一个“C:\my.cnf”文件来保存用于 MySQL 服务器的任何缺省选项。拷贝文件“\mysql\my-example.cnf”到“C:\my.cnf”并且编辑它来适用于你的安装。注意你应该用"/"而不是"\"指定所有的路径。如果你使用“\”，你需要指定两

次(“\\”)，因为在 MySQL 中“\”是转义字符。

2.2.2 在 Windows NT/2000 下安装成系统服务

1、安装成系统服务的方法

对于 NT，比较方便的方法是把 MySQL 安装成 NT 系统服务。如上小节的方法安装完毕后，为了以系统服务的方式服务器的名字是 mysqld-nt：

C:\mysql\bin\mysqld-nt --install

 你可以在 NT 上使用 mysqld 或 mysqld-opt 服务器，但是那些不能作为一种系统服务启动或使用命名管道。

你可以用下列命令启动和停止 MySQL 服务：

c:\>net start mysql

c:\>net stop mysql

2、安装选项文件

注意，如果你按照前述方法把 MySQL 安装成系统服务，在这种情况下，你不能对 mysqld-nt 使用任何其他选项! 所以如果 MySQL 没有安装在缺省的位置，那么 MySQL 不能正确定位数据库目录，将无法启动。

作为一个变通的办法，创建一个“C:\my.cnf”文件来保存用于 MySQL 服务器的任何缺省选项。拷贝文件“\mysql\my-example.cnf”到“C:\my.cnf”并且编辑它来适用于你的安装。无论什么情况都建议你安装这个选项文件。你需要注意的内容是 basedir，作为一个非标准安装，这是必须提供的参数：

[mysqld]

basedir = x:/path/to/mysql/

注意你应该用"/"而不是"\"指定所有的路径。如果你使用“\”，你需要指定两次(“\\”)，因为在 MySQL 中“\”是转义字符。

3、为服务器制定参数

当你独立使用 mysqld 守护程序时，你可以提供参数，例如，指定一个独立的数据库目录：

c:\mysql\bin>mysqld –datadir=”d:\data\”

如果你把 MySQL 安装成系统服务，那么你只能在全局选项文件 c:\my.cnf 中为服务器提供参数。

例如这样

[mysqld]

option1

option2=value

这样相当于用如下参数启动服务器：

c:\mysql\bin>mysqld –option1 –option2=value

4、可能出现的问题

服务用 MySql 名字被安装，一旦安装，它必须使用服务控制管理器(SCM)实用程序启动服务(在控制面板中的管理工具中找到)或使用 NET START MySQL 命令。如果需要任何选项，在你启动 MySQL 服务前必须作为 SCM 实用程序的“启动参数”指定它们。一旦运行，可使用 mysqladmin 或从 SCM 实用程序或使用命令 NET STOP MySQL 停止 mysqld-nt。如果你使用 SCM 停止 mysqld-nt，SCM 有一条关于 mysqld shutdown normally 奇怪的消息，当作为一种服务运行时，mysqld-nt 没有控制台的存取权限，所以没有消息可以看见。

在 NT 上你可以得到下列服务错误消息：

Permission Denied （权限拒绝） 意味着它不能找到 mysqld-nt.exe

Cannot Register （不能登记） 意味着路径是不正确的

如果你作为一种服务安装 mysqld-nt 有问题，尝试用完整的路径启动它：

C:\mysql\bin\mysqld --install

如果你不想作为一种服务启动 mysqld-nt，你可以如下启动它：

C:\mysql\bin\mysqld-nt --standalone

或 C:\mysql\bin\mysqld-nt --standalone --debug

最新的版本在“C:\mysqld.trace”给你一个调试踪迹。

2.2.3 在 Linux 下安装一个 RPM 分发

在写这个部分的时候，我假设你已经对以下提及的有基本认识并拥有相应的环境：

- 懂基本的 Unix 命令、基本的 HTML 语言和 SQL
- 一个工作正常 TCP/IP 网络
- 一个工作正常的 Linux 系统（将作为你安装软件的环境）
- 在 Linux 环境下编译程序所必须的一些软件包,名字应该类似于：

  MySQL-3.22.21-1.i386.rpm 中包含了用于 i386 机器的服务器程序。

  MySQL-client-3.22.21-1.i386.rpm 包含了用于 i386 机器的客户端程序。

  MySQL-devel-3.22.21-1.i386.rpm 包含了用于在 i386 机器上进行开发用的包含文件和库文件（一般也可以不安装）。
- 系统没有安装 MySQL

在 Linux 下安装一个 MySQL 分发也不象想象的那么困难，因为现在大多数的发行版都将 MySQL 打包成 rpm 并且集成到系统中。如果在安装系统时没有安装 MySQL，你必须成为 root 用户才能使用 rpm 安装程序，以下是安装过程：

$mount /dev/cdrom /mnt/cdrom

$cd /mnt/cdrom/Redhat/RPMS

$rpm –ihv MySQL*.rpm

rpm 包的安装比较简单，因为所有的事情，Red Hat Linux 都为你做好了，甚至包括如何启动，以及运行服务器的用户（该方法只在 Red Hat Linux 系统及其兼容的系统上测试成功，应该试用大多数支持 rpm 系统的 Linux 发行版）。

2.2.4 在 Linux 下安装二进制分发

相对于用 RPM 安装来讲，用二进制安装是稍微繁琐了点。但是我们可以在安装脚本中可以自定义安装的相关参数，而不用象 rpm 方式只能安装默认的来安装，具有更大的自由性。

1、安装二进制分发包

如果你下载的是二进制代码，它的名字类似于： mysql-3.22.21-pc-linux-gnu-i686.tar.gz。你必须成为 root 用户，然后解压到 /usr/local 目录，操作步骤如下：

```
$ cd /usr/local
$ su
# tar -zxvf /tmp/mysql-3.22.21-pc-linux-gnu-i686.tar.gz
```

2、建立符号链接

在所有文件解压完后，一个名字叫 mysql-3.22.21-pc-linux-gnu-i686 的目录将被创建出来。我们为这个目录做个符号链接，并给它一个更友好的名字 mysql （免得叫 mysql-3.22.21-pc-linux-gnu-i686 这么长）：

```
# ln -s mysql-3.22.21-pc-linux-gnu-i686 mysql
```

如果以后有新版本的 MySQL 的话，你可以仅仅将源码解压到新的路径，然后只需要做个符号链接就可以了。这样非常方便，数据也更加安全。

2.2.5 在 Linux 下安装源代码分发

挑选一个你要在其下面解包分发的目录，并且进入该目录，这里假设为/tmp。获得 MySQL 一个分发文件。MySQL 源代码分发以压缩的 tar 档案提供，并且有类似于“mysql-VERSION.tar.gz”的名字，这里的 VERSION 是一个类似 3.23.7-alpha 的数字。

1、在当前目录下解包分发：

```
# gunzip < mysql-VERSION.tar.gz | tar   xvf –
or # tar zxvf < mysql-VERSION.tar.gz
```

这个命令创建名为“mysql-VERSION”的一个目录。

2、进入解包分发的顶级目录：

```
#cd   mysql-VERSION
```

3、设置发行版本并且编译：

```
#./configure--prefix=/usr/local/mysql
#make
```

当你运行 configure 时，你可能想要指定一些选项，运行./configure --help 得到一张选项表。如果 configure 失败，你将发送包含你认为能帮你解决该问题的“config.log”的邮件，如果 configure 异常退出，也要包括 configure 的最后几行输出。用 mysqlbug 脚本邮寄错误报告。

4、安装所有东西：

```
#make install
```

你可能需要 root 用户来运行这个命令。

2.2.6 总结

本节讲述 MySQL 安装过程的最初部分，只有你看完下面的各节，MySQL 你才能了解 MySQL

的完整安装过程。对于 Windows 平台上的安装以及 Linux 上的 RPM 安装则比较特殊，本节之后就可以使用了。

源代码安装虽然麻烦，带可以提供更大的自由性和定制性，从而使你的数据库系统更具安全性。对于不被二进制分发或者 RPM 分发支持的平台，使用源代码安装是唯一的途径。

2.3 安装后期的的设置与测试

本节内容主要适用与在 Unix 系统上，所有内容都在 Red Hat Linux 系统上验证通过，应该也是用于其他 Linux 发行版。但是对于 MySQL 用户权限和密码等的安全问题，Windows 平台上的分发，本节的有关说明也是适用的。对于 Red Hat Linux 平台上的 RPM 分发，这些过程已经由系统完成，你也只需注意 MySQL 的用户安全问题。

2.3.1 建立启动 MySQL 的帐户

为了安全性，你应该避免使用 root 帐户启动 MySQL 守护程序，创建一个专门用于启动守护程序的帐户 mysql，并且让数据库属于这个帐户。

1、建立帐户 mysql

```
#adduser mysql
```

2、改变数据库目录的所有者

让我们将 MySQL 目录和文件的拥有权改成 mysql 用户和 root 组：

- 对于二进制分发的安装,根据前文所述的安装方法（以缺省位置为例）：

```
# cd /usr/local
# chown -R mysql:root mysql mysql-3.22.21-pc-linux-gnu-i686
```

- 对于源码安装（以缺省位置为例）：

```
# cd /usr/local
# chown -R mysql:root var
```

- 对于 rpm 包的安装，由于实现已经全部做好，就不必处理了。

3、修改 mysql.server 脚本或者全局选项文件，使守护程序以规定的用户运行

改变 mysql.server 脚本：（mysql.server 可在 **MySQL** 安装目录下的"share/mysql"目录里找到，或在 **MySQL** 源代码树的"support-files"目录下找到。）

找到行 mysql_daemon_user=root,把 root 改为你设定的用户 mysql

或者修改选项文件/etc/my.cnf：

增加选项下述选项

```
[mysql.server]
user=mysql #add this line
basedir=/usr/local/mysql
```

2.3.2 初始化授权表

一旦你已经安装了 MySQL（从一个二进制代码或源代码分发），你需要初始化授权表，因为此时 MySQL 权限系统需要的数据库和表并不存在。然后再，启动服务器并且保证服务器正确运行。你也可以希望安排服务器在你的系统开机和关机时自动地被启动和停止。

安装授权表之前，首先请切换到 mysql 帐户，这也是我们很少使用这一前面建立的帐户的机会。

- 通常，对从源代码分发的安装，你象这样安装授权表并启动服务器：

```
#su mysql
    $cd   BASEDIR(缺省为/usr/local/mysql)
$./scripts/mysql_install_db
```

- 对二进制分发，这样做：

```
#su mysql
$cd BASEDIR(缺省为/usr/local/mysql)
$./bin/mysql_install_db
```

- 对于 rpm 分发，这样做：

```
#su mysql
$mysql_install_db
```

 对于 Win32 平台上的分发，由于安装后，授权表已经存在，故没有必要使用 mysql_install_db，而且也不存在这个脚本。如果你需要初始化授权表，你可以删除数据库目录中的 mysql 目录，即删除 mysql 数据库，然后再运行 setup.exe 文件，即可初始化授权表。

典型地，mysql_install_db 仅在你第一次安装 MySQL 时需要运行，因此，如果你正在升级现有的安装，你可以跳过这一步。（然而，mysql_install_db 的使用相当安全，并且将不更新已经存在的任何表，因此如果你是不能肯定做什么，你总是可以运行 mysql_install_db。）mysql_install_db 在 mysql 数据库里创建 6 个表(user、db、host、tables_priv、columns_priv 和 func)，初始权限的描述在第七章数据库安全中给出。简单地说，这些权限允许 MySQL root 用户做任何事情，并且允许任何人创建立或使用一个名字以'test'或'test_'开始的数据库。

如果你不设置权限表，当你启动服务器时，下列错误将在日志文件出现：

```
mysqld: Can't find file: 'host.frm'
```

2.3.3 测试服务器是否工作

1、象这样启动 MySQL 服务器

对于 Unix 下的分发 ，首先切换到 mysql 用户，否则如果第一次启动服务器是使用的是其他用户，那么数据库文件就为这个用户所有，之后再由用户 mysql 启动守护程序，就会因为数据库文件权限的问题得到一个错误。

当数据库安装完毕时，我们利用 mysql.server 脚本，将不必每次启动前切换到 mysql 用户。如果你再选项文件中指定的合适的选项，也可以不比切换到 mysql 用户。这一点，将在不久之后提到。

- 在 Unix 平台

```
$su mysql
```

$cd MYSQLD_DIR

$bin/safe_mysqld &

- 对于 Win32 平台的分发

c:\>cd MYSQLD_DIR

c:\>bin\mysqld –standalone

如果你安装成系统服务,也可以这样：

c:\>net start mysql

MYSQLD_DIR 的值，在二进制分发中缺省为/usr/local/mysql，在源代码分发中缺省为/usr/local/libexec

2、使用 mysqladmin 证实服务器正在运行。

下列命令提供简单的测试来检查服务器启动和连接的应答：

$mysqladmin version

$mysqladmin variables

从 mysqladmin version 的输出根据你的平台和 MySQL 版本略有不同,但是应该类似如下显示：

$mysqladmin version

mysqladmin Ver 6.3 Distrib 3.22.9-beta, for pc-linux-gnu on i686

TCX Datakonsult AB, by Monty

Server version 3.22.9-beta

Protocol version 10

Connection Localhost via UNIX socket

TCP port 3306

UNIX socket /tmp/mysql.sock

Uptime: 16 sec

Running threads: 1 Questions: 20 Reloads: 2 Open tables: 3

为了能感受到，你能用 mysqladmin 做其他事情，用--help 选项调用它，查看输出的帮助。

3、测试客户端连接

运行客户端程序：

mysql

然后你可以看到屏幕显示出以下信息：

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 2 to server version: 3.22.21

Type 'help' for help.

mysql>

接着，用 show databases 命令可以将安装的数据库列出来：

mysql> show databases;

你就可以看到：

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

2 rows in set (0.00 sec)

如果一切正常的话，那说明 MySQL 可以完全工作了!恭喜你！如果要退出程序，输入 exit，显示结果应该类似于：

mysql> exit;

Bye

2.3.4 自动运行和停止 MySQL

如果安装一个实用的数据库系统,那么让每次系统重新启动时数据库服务器自动运行是一个好主意。

一、我们现在可以由两种方法启动数据库：

1、 可以用 safe_mysqld 脚本来启动数据库：

$ safe_mysqld &

safe_mysqld 脚本安装在 MySQL 安装目录的 bin 目录下，或可在 MySQL 源代码分发的 scripts 目录下找到。

2、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：

$ mysql.server start

$mysql.server stop

mysql.server stop 通过向服务器发出一个信号停止它。你可手工执行 mysqladmin shutdown 关闭服务器。

mysql.server 可在 **MySQL** 二进制分发安装目录下的"share/mysql"目录里找到，或在 **MySQL** 源代码树的"support-files"目录下找到。

二、让服务器以指定的 Unix 用户执行

方法一：修改 mysql.server 脚本

当然，这样只对使用 mysql.server 脚本启动系统的方法其作用。

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。如果你有在一个非标准的地点安装的二进制分发，你可能需要编辑 mysql.server。修改它，运行 safe_mysqld 前，cd 到正确的目录。如果你想要作为一些特定的用户运行服务器，你可以改变 mysql_daemon_user=root 行使用其他用户，你也能修改

mysql.server 把其他选项传给 safe_mysqld。

方法二：使用选项文件

通过使用选项文件，你也可以也可以避免修改 mysql.server 脚本。使用选项文件也可以使 safe_mysqld 脚本通过特定用户启动 mysqld 守护程序。

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”文件的一部分可能看起来像这样：

[mysql.server]

user=mysql

basedir=/usr/local/mysql

mysql.server 脚本使用下列变量：user、datadir、basedir、bindir 和 pid-file。

当然这样只对 mysql.server 脚本起作用，可以把 user 选项加到[mysqld]段中，由于这两个脚本都调用 mysqld 守护程序，因此，这样可以从特定用户启动。

[mysqld]

user=mysql

basedir=/usr/local/mysql

mysqld 守护程序接受的选项可以由命令 mysqld –help 查看。

 说明，修改了 mysql.server 脚本或者选项文件后，你可能注意到我们没有提供用户的密码，所以如果你做了如上的修改，那么就不可以从普通帐户启动服务器，因为只有 root 用户才有切换到任意用户的权力。

三、让 MySQL 随系统启动而自动启动

当你开始使用 **MySQL** 作为生产应用时，你可能想要增加这些启动并且停止命令到在你的“/etc/rc＊ 文件中适当的地方。注意如果你修改 mysql.server，那么如果某个时候你升级 **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。

下面简述一下在 Red Hat Linux 系统中的设置方法：

1、修改 mysql.server，把它复制到/etc/rc.d/init.d 目录里面：

cd /etc/rc.d/init.d

cp /usr/local/mysql/support-files/mysql.server mysql

2、接着把它的属性改为“x”（executable，可执行）

chmod +x mysql

3、最后，运行 chkconfig 把 MySQL 添加到你系统的启动服务组里面去。

/sbin/chkconfig --del mysql

/sbin/chkconfig --add mysql

 说明

如果你的系统使用"/etc/rc.local"启动外部脚本，你也可以添加下列到其中：

二进制分发：

/bin/sh -c 'cd /usr/local/mysql ; ./bin/safe_mysqld &'

源代码分发：

/bin/sh -c 'cd /usr/local/ ; ./bin/safe_mysqld &'

当然，作者更推荐使用 mysqlv.server 的方法，因为这样可以提供系统更好的整合性。

2.3.5 更改 root 用户的密码

初始化授权表之后，最重要的工作是更改数据库服务器root 用户的密码，这对于数据库系统的安全性非常重要，因为 root 用户是数据库的主宰，拥有不受限制的权限，root 用户的密码失窃，就意味着 MySQL 不再属于你了。

在一切正常后，要做的第一件事情是更改管理员的密码，初始化授权表之后，root 用户是没有密码的，这对安全有毁灭性的影响。你可以运行 **mysqladmin** （请注意，此命令不一定在你的path 中，所以最好是转到此命令的目录中直接执行）：

```
# mysqladmin -u root password 'newpassword'
```

2.3.6 修改选项文件

自版本 3.22 以来，MySQL 允许在一个选项文件中存储连接参数。然后在运行 mysql 时就不用重复键入这些参数了；仅当您曾经在命令行上键入过它们时可以使用。这些参数也可以为其他 MySQL 客户机所用，如为 mysqlimport 所用。这也表示在使用这些程序时，选项文件减少了键入工作。

1、选项文件的位置

- 在 Unix 上，MySQL 从下列文件读取缺省选择：

表 2-3　Unix 平台 MySQL 选项文件的位置

文件名	用途
/etc/my.cnf	全局选项
DATADIR/my.cnf	服务器特定的选项目
~/.my.cnf	用户特定的选项

 DATADIR 是 MySQL 的数据目录(典型地对二进制安装是“/usr/local/mysql/data”或对源代码安装是“/usr/local/var”，RPM 安装为/var/mysql)。注意：这是在配置时间指定的目录，不是 mysqld 启动时用--datadir 指定的目录！(--datadir 在服务器寻找选项文件的地方无效，因为它以前寻找他们，它处理任何命令行参数。)

- MySQL 在 Win32 上从下列文件读取缺省选项：

表 2-3　Win32 平台 MySQL 选项文件的位置

文件名	用途

windows-system-directory\my.ini	
C:\my.cnf	全局选项
DATADIR\my.cnf	服务器特定的选项

注意，在 Win32 上的选项文件中，你应用/而不是\指定所有的路径，如果你使用\，你需要指定两个，因为在 MySQL 里面\是转义字符。

2、选项文件的应用顺序

MySQL 试图以上述表格所列的顺序读取选项文件。如果存在多个选项文件，在一个后面文件读取的选项优先于在先前读取的一个文件中指定的同一个选项，在命令行上指定的选项优先于在任何选项文件指定了的选项。有些选择能使用环境变量指定，在命令行或在选项文件指定的选项优先于环境变量。

3、支持选项文件的程序

下列程序支持选项文件：mysql、mysqladmin、mysqld、mysqldump、mysqlimport、mysql.server、myisamchk 和 myisampack。 这也意味着，在选项文件中，每个程序对应着[程序名]的部分。

你能使用选项文件指定一个程序支持的任意长的选项！用--help 选项运行程序可得到的可用选项的表。

4、选项文件的格式合作用

一个选项文件可以包含下列形式的行：

#comment

注释行以“#”或“;”开始，空行被忽略。

[group]

group 是你想为其设置选项的程序或组的名字。在一个组行后，任何 option 或 set-variable 行应用于命名的组，直到选择文件结束或其他组的给出。

option

这等价于在命令行上的--option。

option=value

这等价于在命令行上的--option=value。

set-variable = variable=value

这等价于在命令行上的--set-variable variable=value。该语法必须被用来设置一个 mysqld 变量。

client 组允许你指定适用于所有 MySQL 客户的选项(不是 mysqld)。这是理想的组来指定你用来与服务器连接的口令。(但是保证只是选项文件本身是可读的和可写的。)

注意，对域选项和值，所有头部和尾部空白自动被删除。你可以在的值串中使用转义顺序“\b”、“\t”、“\n”、“\r”、“\\”和“\s”(“\s”==空白)。

这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
set-variable = key_buffer=16M
set-variable = max_allowed_packet=1M

[mysqldump]
quick
```

这是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
```

5、分发文件中，样品选项文件的位置

如果你有一个源代码分发，你将在“support-files”目录下找到一个名为“my-example.cnf”样品配置文件。如果你有二进制分发，在“DIR/share/mysql”目录下查找，在此 DIR 是 MySQL 安装目录的路径(一般是“/usr/local/mysql”)。你可以拷贝“my-example.cnf”到你的主目录(重新命名拷贝为“.my.cnf”)来试验。

为了告诉一个 MySQL 程序不读任何选项文件，在命令行上指定--no-defaults 作为第一个选项。这必须是第一个选项，否则它将无效！如果你想检查使用哪个选项，你可以给出 --print-defaults 选项作为第一个选项。

如果你想要强制使用一个特定配置文件，你可以使用选项 --defaults-file=full-path-to-default-file。如果你这样做，只有指定的文件将被读取。

开发者注意：选项文件的处理简单地通过处理所有在任何命令行前参数的匹配选项来实现(即，在适当的组里的选项)，这对使用多次指定的一个选项的最后实例的程序工作的很好。如果你有这样一个处理多重指定选项但不读选项文件的旧程序， 你只需要增加 2 行给与它那种能力。检查任何标准的 MySQL 客户的源代码看怎样做。

2.3.7 总结

MySQL 的安装不是把程序拷贝到硬盘上就结束了，还需要考虑多方面的内容，尤其使安全性问题。本节对 MySQL 的后期安装过程做了详细的叙述：

- 初始化授权表
- 用特定用户启动守护程序
- 修改 root 用户的密码
- 让 MySQL 自动启动

2.4 系统的升级

因为各种各样的原因，你可能希望升级你的数据库系统。由于数据的原因，升级比新安装一个分发要来的困难的多。

2.4.1 备份数据库与其他配置文件

如果你偏执于或担心新版本，你总能重新命名的旧 mysqld 为类似于 mysqld-'old-version-number'。那么如果你的新 mysqld 做一些意外的事情，你可以简单地关掉它并重启你的旧 mysqld！

当你做一个升级时，当然你也应该备份你的旧数据库。有时有点偏执狂是好的！

你需要备份的内容有：

- 数据库目录
- 守护程序
- 配置文件，例如选项文件

2.4.2 重新安装二进制分发

如果你是严格按照前文提示的方法安装的方法安装 MySQL 的二进制分发，那么升级是非常容易的。甚至没有必要备份数据库，因为升级不会覆盖旧的数据

首先所作的无非是把安装过程重复一遍。

1、把分发包解压到目录/usr/local 目录

2、恢复数据。旧分发的数据库目录，完全复制到新的服务器目录中。

3、重新把/usr/local/mysql 链接到新的服务器目录。

4、启动测试服务器的升级。

2.4.3 重新安装源代码分发

升级一个源代码分发要复杂一些。这里假定已经有一个按照

1、 移动并备份原来的安装，因为升级过程中会覆盖数据库文件和配置文件。

2、 将源代码包解压，编译，安装

```
#tar zxvf mysql-version.tar.gz
#cd mysql-version
#./config
#make
#make install
```

3、 把原来的数据库目录/usr/local/var/的内容恢复。

4、 启动测试新的安装

2.4.4 升级一个 RPM 分发

升级一个 RPM 包分发是最为简单的，原则上你不必备份什么。因为，RPM 升级可以旧版本的配置文件，不兼容的文件也是换名保存，而不是简单覆盖。但是为了安全，你可能还是会小心的备份一些数据。

找到么升级的 RPM 包，例如 MySQL-3.23-xx.i386.rpm。

和安装类似，只是使用-U 选项代替-i 选项：

%rpm -Uhv MySQL-3.23-xx.i386.rpm

2.4.5 检查数据库是否工作及完整

1、 如果备份了数据，可以把数据库目录完全替换掉新分发的数据库目录

2、 启动服务器

3、 使用客户机连接到数据库，检查数据是否安全迁移

4、 检查授权表是否正常

作为升级，和安装一个新的分发有一定的不同。由于已经在上次安装中完成了各种工作，因此安装后期的工作，即第三节的安装内容，可以省略。

2.4.6 总结

本节对如何升级一个系统做了简单的介绍。这里只是介绍了升级的一般过程，对于一些特殊的情况，例如从 3.22 升级到 3.23，表的格式有所变化，可能你还想转换表的格式。在这里，就不再详细叙述了。

对于不同的分发形式，升级的过程都不复杂，原则上要注意升级失败时，你可以恢复旧的系统，那么升级就不是一件危险的事情。

2.5 在同一台机器上运行多个 MySQL 服务器

有些情况下你可能想要在同一台机器上运行多个服务器。例如，你可能想要测试一个新的 MySQL 版本而让你现有生产系统的设置不受到干扰，或你可能是想要为不同的客户提供独立的 MySQL 安装一个因特网服务供应商。同时运行多个服务器，对于升级系统有着很大的意义，因为你可以充分测试新的分发，再决定是否升级。

同时运行多个服务器最重要的工作是，让不同的服务器运行在不同的端口和套接字上。

2.5.1 使用重新编译的方法

如果你想要运行多个服务器，最容易的方法是用不同的 TCP/IP 端口和套接字文件重新编译服务器，因此他们不是侦听同一个 TCP/IP 端口或套接字。

1、如何获知当前服务器使用的端口和套接字

你可以用这个命令检查由任何当前执行的 MySQL 所使用的套接字和端口：

shell> mysqladmin -h hostname --port=port_number variables

应该注意下面的输出内容：

+——————————————+——————————————+

```
| Variable_name          | Value                          |
+------------------------+--------------------------------+
| ansi_mode              | OFF                            |
| back_log               | 50                             |
| ...                    | ...                            |
| port                   | 3306                           |
| ...                    | ...                            |
| socket                 | MySQL                          |
| ...                    | ...                            |
+------------------------+--------------------------------+
```

你可以看到，缺省的情况下，端口是 3306，套接字的名字是 MySQL。

2、如何指定服务器参数

这里只叙述于标准安装不同之处，其余相同的部分不再重复，但是你同样需要完成。

假设一个现存服务器配置为缺省端口号和套接字文件，那么用一个这样的 configure 命令行设置新的服务器：

```
% ./configure    --with-tcp-port=port_number \
                 --with-unix-socket=file_name \
                 --prefix=/usr/local/mysql-3.23.15
```

这里 port_number 和 file_name 应该不同于缺省端口号和套接字文件路径名，而且 --prefix 值应该指定一个不同于现存的 MySQL 安装所在的安装目录。

3、启动服务器

这时你可以直接使用 safe_mysqld 脚本了

```
%su mysql
$/path/to/safe_mysqld &
```

你也应该编辑你机器的初始化脚本(可能是“mysql.server”)来启动并杀死多个 mysqld 服务器。

2.5.2 使用指定服务器参数的方法

你不必重新编译一个新 MySQL 服务器，只要以一个不同的端口和套接字启动即可。

例如，对于一个二进制分发，安装到一个目录后，你可以通过指定在运行时 safe_mysqld 使用的选项来改变端口和套接字：

```
# su mysql
$ /path/to/safe_mysqld --socket=file_name --port=port_number
```

如果你在与开启日志的另一个服务器相同的一个数据库目录下运行新服务器，你也应该用 safe_mysqld 的--log 和--log-update 选项来指定日志文件的名字，否则，两个服务器可能正在试图写入同一个日志文件。

警告：通常你决不应该有在同一个数据库中更新数据的 2 个服务器！如果你的 OS 不

支持无故障(fault-free)的系统锁定，这可能导致令人惊讶的事情发生！

如果你想要为第二个服务器使用另一个数据库目录，你可以使用 safe_mysqld 的 --datadir=path 选项。

由于一般你需要再不同的数据库目录上运行服务器，因此通常最少的参数为：

```
# su mysql
$ /path/to/safe_mysqld --socket=file_name --port=port_number --datadir=path_name
```

2.5.3 有关选项文件的问题

由于全局的选项文件/etc/my.cnf 和用户的选项文件~/.my.cnf 被多个服务器共享，因此在这两个文件中一般只保存内容被多个服务器所共享的选项。

对于个别服务器所使用的选项，请存储在服务器的选项文件中，其位置是 DATADIR/my.cnf。当然你要最先在命令行中指定--datadir=DATADIR。

对于客户机连接，当你想要连接一个正在运行的使用一个不同于编译进你的客户程序中端口的 MySQL 服务器时，你可以使用下列方法之一：

- 以--host 'hostname' --port=port_numer 或[--host localhost] --socket=file_name 启动客户。
- 在你的 C 或 Perl 程序中，当连接 MySQL 服务器时，你可以给出端口和套接字参数。
- 在你启动客户程序之前，设置 MYSQL_UNIX_PORT 和 MYSQL_TCP_PORT 环境变量，指向 Unix 套接字和 TCP/IP 的端口。如果你通常使用一个特定的套接字或端口，你应该将设置这些环境变量的命令放进你的“.login”文件中。见 12.1 不同的 MySQL 程序概述。
- 在你的主目录下的“.my.cnf”文件中指定缺省套接字和 TCP/IP 端口。

2.5.4 总结

本章总结了同时运行多个服务器拷贝的方法，在实际问题中，这是很有用的，不尽对于测试新的分发，同时对备份数据库，保证数据安全也有十分重要的作用。最为一个管理员，掌握这种方法是十分必要的。

思考题

1、 请亲自安装一个 MySQL 分发，平台和分发包可以任意选择，要严格按照本章所介绍的方法。有可能的话，尽量在 Linux 平台上安装一个 MySQL 分发，因为这是 MySQL 应用最广泛的场所。

2、 安装一个 MySQL 分发，如何初始化授权表？之后如何更改 root 用户密码？

3、 将已知 root@localhost 用户的密码（例如，上题所更改）改变，假定原来的密码为 oldpass，新的密码为 newpass。使用实用程序 mysqladmin。

4、 假定 root 用户密码为 newpass，如何使用 root 用户连接服务器，并且发布下面

的查询：

SELECT User,Host FROM mysql.user

5、 如何在选项文件中设定选项，提供 mysql 客户程序上题命令 行的参数 。

第3章

数据库的基本操作

本章要点：

- 服务器的启动
- 客户机与服务器的连接
- 一些 SQL 语句的功能
- 一些专用客户机的使用

在我们阅读了前两章，了解了 MySQL 并且安装成功 MySQL 服务器之后，可以开始学习这个强大而复杂的系统。

客户机/服务器的体系结构是 MySQL 的特点之一，我们首先需要熟悉 MySQL 客户机和服务器的使用，了解客户机如何与服务器连接。

客户机使用 SQL 语言与 MySQL 服务器通信。为了与 MySQL 服务器进行通信，必须熟练掌握 SQL。例如，在使用诸如 mysql 客户机这样的程序时，其功能首先是作为一种发送 SQL 语句给服务器执行的工具。而且，如果编写使用编程语言所提供的 MySQL 接口的程序时，也必须熟悉 SQL 语言，因为需要发送 SQL 语句与服务器沟通。

本章也将对大部分 SQL 语句作一个详细的描述。本章涉及的 SQL 语句多数都是和数据的存储、检索有关，其它少量的 SQL 语句将在其它各章陆续介绍。

3.1 MySQL 的启动与终止

由于 MySQL 服务器具有多种安装分发，而且能够运行在多种操作平台之上，因此它的启动与停止的方法也多种多样。你可以根据实际情况使用其中的一种。在你安装、升级或者维护系统时，你可能需要多次启动和终止服务器，你需要了解启动和终止服务器的方方面面。

3.1.1 直接运行守护程序

1、你可以采用的方法

一般的，你可以有以下几种选择启动 MySQL 服务器，它们的功能和用法几乎是相同的，所以一起在这里介绍：

- 直接使用 MySQL 守护程序 mysqld 启动数据库系统，尤其是 Win32 平台上的分发，这是因为，在 Win32 平台上没有 mysql.server 等服务器脚本可用。
- 通过调用 safe_mysqld 脚本，它接受与 mysqld 相同的参数，并试图为 mysqld 决定正确的选项，然后选择用那些运行它。

2、脚本或者守护程序的存放位置

对于二进制分发的安装，mysqld 守护程序安装在 MySQL 安装目录的 bin 目录下，或者可在 MySQL 源代码分发的 libexec 目录下找到，缺省为/usr/local/libexec/。对于 rpm 分发，mysqld 应该位于 PATH 变量决定的程序搜索路径中，因此可以直接引用。

safe_mysqld 仍旧是一个脚本，并且只存在于 Unix 平台的分发中。safe_mysqld 脚本安装在 MySQL 安装目录的 bin 目录下，或可在 MySQL 源代码分发的 scripts 目录下找到。对于 rpm 分发，该脚本应该位于 PATH 变量决定的程序搜索路径中，因此可以直接引用。

3、为什么要使用 safe_mysqld 脚本

safe_mysqld 接受和 mysqld 同样的参数，并试图确定服务器程序和数据库目录的位置，然后利用这些位置调用服务器。safe_mysqld 将服务器的标准错误输出重定向到数据库目录中的错误文件中，并以记录的形式存在。启动服务器后，safe_mysqld 还监控服务器，并在其死机时重新启动。safe_mysqld 通常用于 Unix 的 BSD 风格的版本。

如果，你曾经为 root 或在系统启动程序中启动 safe_mysqld，其错误日志将有 root 拥有。如果再用非特权的用户身份调用 safe_mysqld，则可能引起“Access Denied”(即，“所有权被拒绝”)的错误。此时可以删除错误文件在试一次。

由于 safe_mysqld 脚本的功能，使用 safe_mysqld 脚本明显比直接启动 mysqld 守护程序来得有效。

4、启动服务器的完整过程

对于 Unix 平台上的分发，如果你以 root 或在系统引导期间启动 sqfe_mysqld，出错日志由 root 拥有，这可能在你以后试图用一个非特权用户调用 safe_mysqld 时将导致“permission denied”（权限拒绝）错误，删除出错日志再试一下。因此建议你在启动服务器前，首先切换到一个专门的用户 mysql。

具体方法如下：

- Unix 平台

  $ su mysql

  $ safe_mysql & （或者 mysqld & ，不推荐）

- Win32 平台

  C: \ mysql\bin>mysqld --standalone

  或者 C: \ mysql\bin>mysqld-nt –standalone

如果你使用 mysqld 并且没有把 mysql 安装在标准的位置，通常需要提供—basedir 选项你的数据库的安装位置。

$safe_mysqld --basedir=”/path/to/mysql” &（Unix 平台）

c:\mysql\bin>mysqld --basedir=”x:/path/to/mysql” (Win32 平台)

5、使用 safe_mysqld 脚本实现服务器的自动启动

同样你可以利用 safe_mysqld 和 mysqld 实现服务器随操作系统自动启动。对于 Linux 系统以及 BSD 风格的系统（FreeBSD，OpenBSD 等)），通常在/etc 目录下有几个文件在引导时初始化服务，这些文件通常有以“rc”开头的名字，且它有可能由一个名为“rc.local”的文件（或类似的东西），特意用于启动本地安装的服务。

在这样的系统上，你可能将类似于下列的行加入 rc.local 文件中以启动服务器（如果 safe_mysqld 的目录在你的系统上不同，修改它即可）：

if [-x /usr/local/bin/safe_mysqld]; then /usr/local/bin/safe_mysqld & fi

由于这样在引导时启动，将使用 root 身份启动数据库，在某些时候可能会产生问题和麻烦。这是你可以指定--user 选项，因此可以将上面的代码修改为：

```
if [ -x /usr/local/bin/safe_mysqld ]; then
    /usr/local/bin/safe_mysqld --user=mysql --datadir=/path/to/data    &
fi
```

3.1.2 使用脚本 mysql.server 启动关闭数据库

对于 Unix 平台上的分发，比较好的办法是使用数据库脚本 mysql.server,启动和关闭数据库。

1、mysql.server 脚本的存放位置

mysql.server 脚本安装在 MySQL 安装目录下的 share/mysqld 目录下或可以在 MySQL 源代码分发的 support_files 目录下找到。对于 rpm 分发，该脚本已经改名 mysql 位于 /etc/rc.d/init.d 目录中，另外在….中存在一个副本 mysql.server。下文的讨论对 RPM 分发来说，都在安装中完成了。如果你想使用它们，你需要将它们拷贝到适当的目录下。

2、如何使用 mysql.server 脚本启动停止服务器

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：

$ mysql.server start

$ mysql.server stop

3、mysql.server 的功能简述

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。如果你有在一个非标准的地点安装的二进制分发，你可能需要编辑mysql.server。修改它，运行 safe_mysqld 前，cd 到正确的目录。

4、让 mysql.server 以特定的用户启动服务器

如果你想要作为一些特定的用户运行服务器，你可以改变 mysql_daemon_user=root 行使用其他用户，你也能修改 mysql.server 把其他选项传给 safe_mysqld。

当然你也可以利用选项文件给 mysql.server 脚本提供参数。

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=/tmp/mysqld.sock
port=3306

[mysql.server]
user=mysql
basedir=/usr/local/mysql
```

mysql.server 脚本使用下列变量：user、datadir、basedir、bindir 和 pid-file。

5、利用 mysql.server 脚本让服务器自动启动

mysql.server 脚本的重要性在于你可以使用它配置一个随操作系统自动启动的数据库安装，这是一个实际的系统中常用的方法。

- 把 mysql.server 复制到/etc/rc.d/init.d 目录里面：

```
# cd /etc/rc.d/init.d
# cp /usr/local/mysql/support-files/mysql.server mysql
```

- 接着把它的属性改为“x”（executable，可执行）

```
# chmod +x mysql
```

- 最后，运行 chkconfig 把 MySQL 添加到你系统的启动服务组里面去。

```
# /sbin/chkconfig --del mysql
# /sbin/chkconfig --add mysql
```

你也可以这样做，手工建立链接：

```
#cd /etc/rc.d/rc3.d          (根据你的运行级不同而不同，可以是 rc5.d)
#ln –s ../init.d/mysql S99mysql
```

在系统启动期间，S99mysql 脚本利用 start 参数自动启动。

3.1.3 使用 mysqladmin 实用程序关闭、重启数据库

日常维护数据库，经常需要关闭或者重新启动数据库服务器。mysql.server stop 只能用于关闭数据库系统，并不方便，而且 mysql.server 脚本用特定用户身份启动服务器，因此使用该教本需要超级用户的权限。相比之下 mysqladmin 就方便的多，并且适用于 MySQL

所有类型、平台的安装。

- 关闭数据库服务器
  mysqladmin shutdown
- 重启数据库服务器
  mysqladmin reload
- 获得帮助
  mysqladmin –help
  mysqladmin 实用程序非常有使用价值，仔细阅读帮助输出，你会得到更多的用法。

 如果出现下面的错误:

mysqladmin: connect to server at 'localhost' failed

error: 'Access denied for user: 'root@localhost' (Using password: YES)'

表示你需要一个可以正常连接的用户，请指定-u -p 选项，具体方法与 3.2 节介绍相同，在第七章中你将会学到用户授权的知识。例如，你现在可以:

shell>mysqladmin -u root -p shutdown

Enter Password:**********

输入你在阅读第二章时修改过的密码即可。

3.1.4 启动或停止 NT 平台上的系统服务

上面几节介绍了 Unix 平台上使数据库服务器自动启动的方法，而在 windows（NT）平台上为了让 MySQL 数据库自动启动，你需要将 MySQL 服务器安装成 NT 系统的一种服务。

1、将 MySQL 安装成系统服务

对于 NT，服务器名字是 mysqld-nt。

C:\mysql\bin> mysqld-nt --install

（你可以在 NT 上使用 mysqld 或 mysqld-opt 服务器，但是那些不能作为一种服务启动或使用命名管道。）

2、修改选项文件

如果你的 mysql 没有安装缺省的位置 c:\mysql，那么因为数据库将无法确定数据库目录的位置，而无法启动。这种情况下，你需要提供一个全局的选项文件 c:\my.cnf。将安装目录中的 my-example.cnf 文件拷贝到 c 盘根目录下。修改或加入：

[mysqld]

basedir = x:/path/to/mysql/

3、启动、停止服务器的方法

你可以用下列命令启动和停止 MySQL 服务：

NET START mysql

NET STOP mysql

注意在这种情况下，你不能对 mysqld-nt 使用任何其他选项!你需要使用选项文件提供参数。可以在选项文件中提供合适的参数。

3.1.5 使用选项文件提供服务器的参数

本节介绍守护程序需要的比较重要的选项。

对于全局选项文件（windows:c:/my.cnf 或者 unix:/etc/my.cnf），比较重要的选项有：

user 运行守护程序的用户，如果你使用 mysql.server，并且在[mysql.server]中指定了 user，那么这个选项没有必要。

log-update=file 更新日志名，对于恢复数据库有重要意义，数据库服务器将生成 file.n 为名的更新日志文件。

-l, --log[=file] 指定常规日志名

--log-bin[=file] 指定使用的二进制日志文件，应用于服务器复制。

-Sg, --skip-grant-tables 启动时不加载授权表，维护时使用。

-u, --user=user_name 启动服务器使用的 Unix 帐户

--socket=... 指定是用的 Unix 套接字文件名，在同时运行多个服务器时使用。

-P, --port=... 指定服务器使用的端口，在同时运行多个服务器时使用。

3.1.6 总结

作为 MySQL 的管理员，你平时的指责就是确保服务器尽可能的正常运行，使得客户机能够正常访问。本章讨论了如何启动和重启、关闭服务器，如果你具备了这个能力，就可以更好的维护服务器安装。本章介绍的方法，多数适合于 Unix 系统，读者请仔细分辨。

3.2 MySQL 与客户机的连接

本章通过演示如何使用 mysql 客户程序与数据库服务器连接。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。使用 mysql 的两个方法都在这里涉及。

为了看清由 mysql 提供的一个选择项目表了，用--help 选项调用它：

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 服务器你可以连接。如果这不是真的，联络你的 MySQL 管理员。(如果你是管理员，你将需要请教这本手册的其他章节。)

3.2.1 建立和中止与服务器的连接

1、如何使用客户机建立连接

为了连接到服务器，从外壳程序（即从 UNIX 提示符，或从 Windows 下的 DOS 控制台）激活 mysql 程序。命令如下：

shell>mysql

又如，直接连接一个数据库：

shell> mysql db_name

其中的“$”在本书中代表外壳程序提示符。这是 UNIX 标准提示符之一；另一个为“#”。在 Windows 下，提示符类似 “c:\>”。

2、客户机最常使用的选项：主机、用户和密码

为了连接服务器，当你调用 mysql 时，你通常将需要提供一个 MySQL 用户名和很可能，一个口令。如果服务器运行在不是你登录的一台机器上，你也将需要指定主机名。联系你的管理员以找出你应该使用什么连接参数进行连接(即，那个主机，用户名字和使用的口令)。一旦你知道正确的参数，你应该能象这样连接：

shell> mysql -h host -u user -p

Enter password: ********

********代表你的口令；当 mysql 显示 Enter password:提示时输入它。

在刚开始学习 MySQL 时，大概会为其安全系统而烦恼，因为它使您难于做自己想做的事。(您必须取得创建和访问数据库的权限，任何时候连接到数据库都必须给出自己的名字和口令。)但是，在您通过数据库录入和使用自己的记录后，看法就会马上改变了。这时您会很欣赏 MySQL 阻止了其他人窥视（或者更恶劣一些，破坏!）您的资料。

下面介绍选项的含义：

-h host_name（可选择形式：--host=host_name）

希望连接的服务器主机。如果此服务器运行在与 mysql 相同的机器上，这个选项一般可省略。

- -u user_name（可选择的形式：--user=user_name）

您的 MySQL 用户名。如果使用 UNIX 且您的 MySQL 用户名与注册名相同，则可以省去这个选项；mysql 将使用您的注册名作为您的 MySQL 名。

在 Windows 下，缺省的用户名为 ODBC。这可能不一定非常有用。可在命令行上指定一个名字，也可以通过设置 USER 变量在环境变量中设置一个缺省名。如用下列 set 命令指定 paul 的一个用户名：

- -p（可选择的形式：--password）

这个选项告诉 mysql 提示键入您的 MySQL 口令。注意：可用 -pyour_password 的形式（可选择的形式：--password=your_password）在命令行上键入您的口令。但是，出于安全的考虑，最好不要这样做。选择 -p 不跟口令告诉 mysql 在启动时提示您键入口令。例如：

在看到 Enter password: 时，键入口令即可。(口令不会显到屏幕，以免给别人看到。)请注意，MySQL 口令不一定必须与 UNIX 或 Windows 口令相同。

如果完全省略了 -p 选项，mysql 就认为您不需要口令，不作提示。

请注意：-h 和 -u 选项与跟在它们后面的词有关，无论选项和后跟的词之间是否有空格。而 -p 却不是这样，如果在命令行上给出口令，-p 和口令之间一定不加空格。

例如，假定我的 MySQL 用户名和口令分别为 tom 和 secret，希望连接到在我注册

的同一机器上运行的服务器上。下面的 mysql 命令能完成这项工作：

shell>mysql –u tom -p

在我键入命令后，mysql 显示 Enter password: 提示键入口令。然后我键入口令（******表明我键入了 secret）。

如果一切顺利的话，mysql 显示一串消息和一个“mysql>”提示，表示它正等待我发布查询。完整的启动序列如下所示：

为了连接到在其他某个机器上运行的服务器，需要用 -h 指定主机名。如果该主机为 mysql.domain.net，则相应的命令如下所示：

shell>mysql –h mysql.domain.net –u tom -p

在后面的说明 mysql 命令行的多数例子中，为简单起见，我们打算省去 -h、-u 和 -p 选项。并且假定您将会提供任何所需的选项。

有很多设置账号的方法，从而不必在每次运行 mysql 时都在连接参数中进行键入。这个问题在前面已经介绍过，你只需在选项文件中提供参数，具体请看 3.2.2。您可能会希望现在就跳到该节，以便找到一些更易于连接到服务器的办法。

3、结束会话

在建立了服务器的一个连接后，可在任何时候键入下列命令来结束会话：

quit exit

还可以键入 Control-D 来退出，至少在 UNIX 上可以这样。

3.2.2 利用选项文件简化连接

在激活 mysql 时，有可能需要指定诸如主机名、用户名或口令这样的连接参数。运行一个程序需要做很多输入工作，这很快就会让人厌烦。利用选项文件可以存储连接参数，减少输入工作。

例如，对于使用 mysqladmin 客户机来管理数据库，那么你很快会厌烦每次使用这么长的命令行：

shell>mysql –u root –p varialbles

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

你也许会选择使用全局选项文件来存贮你的参数：

[mysqladmin]

#也可以用[client]，为所有的客户机存储参数

user=root

password=yourpassword

这样执行 mysqladmin variables 就不会显示任何存储拒绝的错误，并且你可以用 root 用户身份维护数据库了。

且慢，你立刻会发现，这样做是一个很大的安全漏洞，因为任何可以读取选项文件的用户都可以获得你的密码！解决方法是，只提供 password 选项不提供密码：

[mysqladmin]

user=root

password

这样你在命令行执行时，会提示你数据密码：

shell>mysql varialbles

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

但是，你不能不提供 password 选项，否则你还是要在命令行中提供-p 选项。

3.2.3 利用 mysql 的输入行编辑器

mysql 具有内建的 GNU Readline 库，允许对输入行进行编辑。可以对当前录入的行进行处理，或调出以前输入的行并重新执行它们（原样执行或做进一步的修改后执行）。在录入一行并发现错误时，这是非常方便的；您可以在按 Enter 键前，在行内退格并进行修正。如果录入了一个有错的查询，那么可以调用该查询并对其进行编辑以解决问题，然后再重新提交它。（如果您在一行上键入了整个查询，这是最容易的方法。）

表 2-1 中列出了一些非常有用的编辑序列，除了此表中给出的以外，还有许多输入编辑命令。利用因特网搜索引擎，应该能够找到 Readline 手册的联机版本。此手册也包含在 Readline 分发包中，可在 http://www.gnu. org/ 的 Gnu Web 站点得到。

表 2-1 mysql 输入编辑命令

键 序 列	说 明
Up 箭头，Ctrl-p	调前面的行
Down 箭头，Ctrl-N	调下一行
Left 箭头，Ctrl-B	光标左移（向后）
Right 箭头，Ctrl-F	光标右移（向前）
Escape Ctrl-B	向后移一个词
Escape Ctrl-F	向前移一个词
Ctrl-A	将光标移到行头
Ctrl-E	将光标移到行尾
Ctrl-D	删除光标下的字符
Delete	删除光标左边的字符
Escape D	删词
Escape Backspace	删除光标左边的词
Ctrl-K	删除光标到行尾的所有字符
Ctrl-_	撤消最后的更改；可以重复

下面的例子描述了输入编辑的一个简单的使用。假定用 mysql 输入了下列查询：

如果在按 Enter 前，已经注意到将“president”错拼成了“persident”，则可按左箭头

或 Ctrl-B 多次移动光标到“s”的左边。然后按 Delete 两次删除“er”，键入“re”改正错误，并按 Enter 发布此查询。如果没注意到错拼就按了 Enter，也不会有问题。在 mysql 显示了错误消息后，按上箭头或 Ctrl-P 调出该行，然后对其进行编辑。

输入行编辑在 mysql 的 Windows 版中不起作用，但是可从 MySQL Web 站点取得免费的 cygwin_32 客户机分发包。在该分发包中的 mysqlc 程序与 mysql 一样，但它支持输入行编辑命令。

3.2.4 批处理模式连接

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
```

如果你需要在命令行上指定连接参数，命令可能看起来像这样：

```
shell> mysql -h host -u user -p < batch-file
Enter password: ********
```

当你这样使用 mysql 时，你正在创建一个脚本文件，然后执行脚本。

为什么要使用一个脚本？有很多原因：

- 如果你重复地运行查询(比如说，每天或每周)，把它做成一个脚本使得你在每次执行它时避免重新键入。
- 你能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行语句序列。如果你犯了一个错误，你不必重新打入所有一切，只要编辑你的脚本来改正错误，然后告诉 mysql 再次执行它。
- 如果你有一个产生很多输出的查询，你可以通过一个分页器而不是盯着它翻屏到你屏幕的顶端来运行输出：

  ```
  $ mysql < batch-file | more
  ```

- 你能捕捉输出到一个文件中进行更一步的处理：

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

- 你可以散发脚本给另外的人，因此他们也能运行命令。
- 一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，其缺省输出格式是不同的(更简明些)。例如，当交互式运行 SELECT DISTINCT species FROM pet 时，输出看起来像这样：

```
+---------+
| species |
+---------+
```

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

但是当以批模式运行时，像这样：

```
species
bird
cat
dog
hamster
snake
```

如果你想要在批模式中得到交互的输出格式，使用 mysql -t。为了回显以输出被执行的命令，使用 mysql -vvv。

3.2.5 总结

本章中列举了客户机与服务器连接的几种情况，读者需要注意的是下面几点：

1、 如何提供参数，以及参数的意义

2、 如何让客户机提示输入密码

3、 交互模式和批处理模式

4、 mysql 客户机的行编辑功能

3.3 有关数据库的操作

从本节开始正式介绍各种 SQL 语句。本节介绍有关数据库级的 SQL 以及相关操作，查看、建立和删除等操作。

3.3.1 用 SHOW 显示已有的数据库

句法：SHOW DATABASES [LIKE wild]

如果使用 LIKE wild 部分，wild 字符串可以是一个使用 SQL 的“%”和“_”通配符的字符串。

功能：SHOW DATABASES 列出在 MySQL 服务器主机上的数据库。

你可以尝试下面举例，观察输出结果，例如：

```
mysql>show databases;
+----------+
| Database |
```

```
+---------+
| first   |
| mysql   |
| mytest  |
| test    |
| test1   |
+---------+
mysql>show databases like 'my%';
+--------------+
| Database (my%) |
+--------------+
| mysql        |
| mytest       |
+--------------+
```

用 mysqlshow 程序也可以得到已有数据库列表。

3.3.2 用 Create Dabase 创建数据库

句法：CREATE DATABASE db_name

功能：CREATE DATABASE 用给定的名字创建一个数据库。

如果数据库已经存在，发生一个错误。

在 MySQL 中的数据库实现成包含对应数据库中表的文件的目录。因为数据库在初始创建时没有任何表，CREATE DATABASE 语句只是在 MySQL 数据目录下面创建一个目录。

例如：

mysql>create database myfirst;

然后利用 show databases 观察效果。

3.3.3 用 DROP DATABASE 删除数据库

句法：DROP DATABASE [IF EXISTS] db_name

功能：DROP DATABASE 删除数据库中的所有表和数据库。要小心地使用这个命令！

DROP DATABASE 返回从数据库目录被删除的文件的数目。通常，这 3 倍于表的数量，因为每张表对应于一个“.MYD”文件、一个“.MYI”文件和一个“.frm”文件。

在 MySQL 3.22 或以后版本中，你可以使用关键词 IF EXISTS 阻止一个错误的发生，如果数据库不存在。

3.3.4 使用 mysqladmin 工具创建和删除

在命令行环境下可以使用 mysqladmin 创建和删除数据库。

创建数据库：

shell> mysqladmin create db_name

删除数据库：

shell> mysqladmin drop db_name

 如果出现下面的错误:

mysqladmin: connect to server at 'localhost' failed

error: 'Access denied for user: 'root@localhost' (Using password: YES)'

表示你需要一个可以正常连接的用户，请指定-u -p 选项，具体方法与 3.2 节介绍相同，在第七章中你将会学到用户授权的知识。

3.3.5 直接在数据库目录中创建或删除

用上述方法创建数据库，只是 MySQL 数据目录下面创建一个与数据库同名目录，同样删除数据库是把这个目录删除。

所以，你可以直接这么做，创建或删除数据库，或者给数据库更名。这对备份和恢复备份有一定意义。

3.3.6 用 USE 选用数据库

句法：USE db_name

USE db_name 语句告诉 MySQL 使用 db_name 数据库作为随后的查询的缺省数据库。数据库保持到会话结束，或发出另外一个 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
```

如果你不是用 USE 语句，那么上面的例子应该写成：

```
mysql> SELECT count(*) FROM db1.mytable;
mysql> SELECT count(*) FROM db2.mytable;
```

 由于 use 也是一个 mysql 客户程序的命令，所以你可以在命令行最后不加分号，客户程序可以得到结果。

3.3.7 总结

本节介绍了有关数据库操作的 SQL 语句、实用程序，其中包括：

- SQL 语句：CREATE/DROP DATABASE，SHOW DATABASES，USE
- 程序 mysqladmin
- 直接创建或删除数据库的目录

3.4 有关数据表的操作

用 MySQL，目前(版本 3.23)你可以在三种基本数据库表格式间选择。当你创建一张表时，你可以告诉 MySQL 它应该对于表使用哪个表类型。MySQL 将总是创建一个.frm 文件保存表和列定义。视表类型而定，索引和数据将在其他文件中存储。

你能用 ALTER TABLE 语句在不同类型的表之间变换。见 7.8 ALTER TABLE 语法。

- MyISAM

在 MySQL 3.23 中，MyISAM 是缺省表格类型，它是基于 ISAM 代码并且有很多有用的扩展。索引存储在一个有.MYI（MYindex）扩展名的文件并且数据存储在有.MYD（MYData）扩展名的文件中。你能用 myisamchk 实用程序检查/修复 MyISAM 表。

- ISAM

你也可以使用放弃的 ISAM。这将在不久消失，因为 MyISAM 是同一个东西的更好实现。ISAM 使用一个 B-tree 索引，这个索引存储在一个有.ISM 扩展名的文件中并且数据存储在有.ISD 扩展名的文件中，你可用 isamchk 实用程序检查/修复 ISAM 表。ISAM 表不是跨 OS/平台二进制可移植的。

- HEAP

HEAP 表格使用一个杂凑（hashed）索引并且存储在内存中。这使他们更快，但是如果 MySQL 崩溃，你将失去所有存储的数据。HEAP 作为临时表很可用！

3.4.1 用 SHOW/ DESCRIBE 语句显示数据表的信息

句法：

```
SHOW TABLES [FROM db_name] [LIKE wild]
or SHOW COLUMNS FROM tbl_name [FROM db_name] [LIKE wild]
or SHOW INDEX FROM tbl_name [FROM db_name]
or SHOW TABLE STATUS [FROM db_name] [LIKE wild]

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

你能使用 `db_name.tbl_name` 作为 `tbl_name FROM db_name` 句法的另一种选择。

- SHOW TABLES 列出在一个给定的数据库中的表。你也可以用 mysqlshow db_name 命令得到这张表。

注意：如果一个用户没有一个表的任何权限，表将不在 SHOW TABLES 或 mysqlshow db_name 中的输出中显示。

- SHOW COLUMNS 列出在一个给定表中的列。如果列类型不同于你期望的是基于 CREATE TABLE 语句的那样，注意，MySQL 有时改变列类型。
- DESCRIBE 语句提供了类似 SHOW COLUMNS 的信息。DESCRIBE 提供关于一张表的列的信息。col_name 可以是一个列名字或包含 SQL 的“%”和“_”通配符的一个字符串。这个语句为了与 Oracle 兼容而提供的。
- SHOW TABLE STATUS（在版本 3.23 引入)运行类似 SHOW STATUS，但是提供

每个表的更多信息。你也可以使用 mysqlshow --status db_name 命令得到这张表。

- SHOW FIELDS 是 SHOW COLUMNS 一个同义词，SHOW KEYS 是 SHOW INDEX 一个同义词。
- 你也可以用 mysqlshow db_name tbl_name 或 mysqlshow -k db_name tbl_name 列出一张表的列或索引。
- SHOW INDEX 以非常相似于 ODBC 的 SQLStatistics 调用的格式返回索引信息。

3.4.2 使用 mysqlshow 工具得到信息

下面简单介绍一下 mysqlshow 实用程序的用法，在得到数据库和表的信息上，使用起来非常方便。

得到已有数据库的列表：

```
shell> mysqlshow
```

列出某数据库 db_name 中已有的表：

```
shell> mysqlshow db_name
```

列出某数据库表 db_name.tbl_name 的结构信息：

```
shell>mysqlshow db_name tbl_name
```

列出一张表的索引：

```
shell> mysqlshow –k db_name tbl_name
```

3.4.3 用 CREATE TABLE 语句创建数据表

用 CREATE TABLE 语句创建表。此语句的完整语法是相当复杂的，因为存在那么多的可选子句，但在实际中此语句的应用相当简单。如我们在第 1 章中使用的所有 CREATE TABLE 语句都不那么复杂。

有意思的是，大多数复杂东西都是一些子句，这些子句 MySQL 在分析后扔掉。参阅附录 1 可以看到这些复杂的东西。

1、CREATE TABLE 语句的基本语法

```
CREATE TABLE tbl_name(create_definition,...) [TYPE =table_type]
```

create_definition: col_name type [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT][PRIMARY KEY]

在 MySQL3.22 或以后版本中，表名可以被指定为 db_name.tbl_name，不管有没有当前的数据库都可以。

例如，创建一个访问者留言表：

```
shell> mysql –u root –p
mysql> create database mytest;
mysql> CREATE  TABLE  guestbook
    -> (
    -> visitor VARCHAR(40),
    -> comments TEXT,
    -> entrydate DATETIME
```

->);

如果一切正常，祝贺你，你已经建立了你的第一个表!

你所创建的表名为 guestbook，你可以使用这个表来存储来字你站点访问者的信息。你是用 REEATE TABLE 语句创建的这个表，这个语句有两部分：第一部份指定表的名子；第二部份是括在括号中的各字段的名称和属性，相互之间用逗号隔开。

表 guestbook 有三个字段：visitor,comments 和 entrydate。visitor 字段存储访问者的名字，comments 字段存储访问者对你站点的意见，entrydate 字段存储访问者访问你站点的日期和时间。

注意每个字段名后面都跟有一个专门的表达式。例如，字段名 comments 后面跟有表达式 TEXT。这个表达式指定了字段的数据类型。数据类型决定了一个字段可以存储什么样的数据。因为字段 comments 包含文本信息，其数据类型定义为文本型。

2、如何指定表的类型

你也可以在创建表时指定表的类型，如果不指定表的类型，在 3.22 及以前版本中缺省为 ISAM 表，在 3.23 版本中缺省为 MyISAM 表。你应该尽量使用 MyISAM 表。指定表的类型经常用于创建一个 HEAP 表：

```
mysql> CREATE TABLE fast(id int,articles TEXT) TYPE=HEAP;
```

3、隐含的列说明的改变

在某些情况下，MySQL 隐含地改变在一个 CREATE TABLE 语句给出的一个列说明。（这也可能在 ALTER TABLE。）

- 长度小于 4 的 VARCHAR 被改变为 CHAR。
- 如果在一个表中的任何列有可变长度，结果是整个行是变长的。因此, 如果一张表包含任何变长的列(VARCHAR、TEXT 或 BLOB)，所有大于 3 个字符的 CHAR 列被改变为 VARCHAR 列。这在任何方面都不影响你如何使用列；在 MySQL 中，VARCHAR 只是存储字符的一个不同方法。MySQL 实施这种改变，是因为它节省空间并且使表操作更快捷。
- TIMESTAMP 的显示尺寸必须是偶数且在 2 ～ 14 的范围内。如果你指定 0 显示尺寸或比 14 大，尺寸被强制为 14。从 1～13 范围内的奇数值尺寸被强制为下一个更大的偶数。
- 你不能在一个 TIMESTAMP 列里面存储一个文字 NULL；将它设为 NULL 将设置为当前的日期和时间。因为 TIMESTAMP 列表现就是这样，NULL 和 NOT NULL 属性不以一般的方式运用并且如果你指定他们，将被忽略。DESCRIBE tbl_name 总是报告该 TIMESTAMP 列可能赋予了 NULL 值。

如果你想要知道 MySQL 是否使用了除你指定的以外的一种列类型，在创建或改变你的表之后，发出一个 DESCRIBE tbl_name 语句即可。

3.4.4 利用 SELECT 的结果创建表

关系数据库的一个重要概念是，任何数据都表示为行和列组成的表，而每条 SELECT

语句的结果也都是一个行和列组成的表。在许多情况下，来自 SELECT 的“表”仅是一个随着您的工作在显示屏上滚动的行和列的图像。在 MySQL 3.23 以前，如果想将 SELECT 的结果保存在一个表中以便以后的查询使用，必须进行特殊的安排：

1) 运行 DESCRIBE 或 SHOW COLUMNS 查询以确定想从中获取信息的表中的列类型。

2) 创建一个表，明确地指定刚才查看到的列的名称和类型。

3) 在创建了该表后，发布一条 INSERT ... SELECT 查询，检索出结果并将它们插入所创建的表中。

在 MySQL 3.23 中，全都作了改动。CREATE TABLE ... SELECT 语句消除了这些浪费时间的东西，使得能利用 SELECT 查询的结果直接得出一个新表。只需一步就可以完成任务，不必知道或指定所检索的列的数据类型。这使得很容易创建一个完全用所喜欢的数据填充的表，并且为进一步查询作了准备。

- 如果你在 CREATE 语句后指定一个 SELECT，MySQL 将为在 SELECT 中所有的单元创键新字段。例如：

```
mysql> CREATE TABLE test
    -> (a int not null auto_increment,primary key (a), key(b))
    -> SELECT b,c from test2;
```

这将创建一个有 3 个列(a，b，c)的表，其中 b，c 列的数据来自表 test2。注意如果在拷贝数据进表时发生任何错误，表将自动被删除。

- 可以通过选择一个表的全部内容（无 WHERE 子句）来拷贝一个表，或利用一个总是失败的 WHERE 子句来创建一个空表，如：

```
mysql> CREATE TABLE test SELECT * from test2;
mysql> CREATE TABLE test SELECT * from test2 where 0;
```

如果希望利用 LOAD DATA 将一个数据文件装入原来的文件中，而不敢肯定是否具有指定的正确数据格式时，创建空拷贝很有用。您并不希望在第一次未得到正确的选项时以原来表中畸形的记录而告终。利用原表的空拷贝允许对特定的列和行分隔符用 LOAD DATA 的选项进行试验，直到对输入数据的解释满意时为止。在满意之后，就可以将数据装入原表了。

可结合使用 CREATE TEMPORARY TABLE 与 SELECT 来创建一个临时表作为它自身的拷贝，如：

这允许修改 my_tbl 的内容而不影响原来的内容。在希望试验对某些修改表内容的查询，而又不想更改原表内容时，这样做很有用。为了使用利用原表名的预先编写的脚本，不需要为引用不同的表而编辑这些脚本；只需在脚本的起始处增加 CREATE TEMPORARY TABLE 语句即可。相应的脚本将创建一个临时拷贝，并对此拷贝进行操作，当脚本结束时服务器会自动删除这个拷贝。

要创建一个作为自身的空拷贝的表，可以与 CREATE TEMPORARY ... SELECT 一起使用 WHERE 0 子句，例如：

但创建空表时有几点要注意。在创建一个通过选择数据填充的表时，其列名来自所选择的列名。如果某个列作为表达式的结果计算，则该列的“名称”为表达式的文本。表达式不是合法的列名，可在 mysql 中运行下列查询了解这一点：

为了正常工作，可为该列提供一个合法的别称：

如果选择了来自不同表的具有相同名称的列，将会出现一定的困难。假定表 t1 和 t2 两者都具有列 c，而您希望创建一个来自两个表中行的所有组合的表。那么可以提供别名指定新表中惟一性的列名，如：

通过选择数据进行填充来创建一个表并会自动拷贝原表的索引。

3.4.5 用 ALTER TABLE 语句修改表的结构

有时你可能需要改变一下现有表的结构，那么 Alter Table 语句将是你的合适选择。

- 增加列

alter table tbl_name add col_name type

例如,给表增加一列 weight

mysql>alter table pet add weight int;

- 删除列

alter table tbl_name drop col_name

例如，删除列 weight：

mysql>alter table pet drop weight;

- 改变列

alter table tbl_name modify col_name type

例如，改变 weight 的类型：

mysql> alter table pet modify weight samllint;

另一种方法是：

alter table tbl_name change old_col_name col_name type

例如：

mysql> alter table pet change weight weight samllint;

- 给列更名

mysql>alter table pet change weight wei;

- 给表更名

alter table tbl_name rename new_tbl

例如，把 pet 表更名为 animal

mysql>alter table pet rename animal;

- 改变表的类型

另外，可以为列增加或删除索引等属性，不再详述，请参阅附录。

3.4.6 用 DROP TABLE 语句删除数据表

DROP TABLE [IF EXISTS] tbl_name [, tbl_name,...]

DROP TABLE 删除一个或多个数据库表。所有表中的数据和表定义均被删除，故小心使用这个命令!

在 MySQL 3.22 或以后版本，你可以使用关键词 IF EXISTS 类避免不存在表的一个错误发生。

例如：

mysql>USE mytest;

mysql>DROP TABLE guestbook;

或者，也可以同时指定数据库和表：

mysql>DROP TABLE mytest. guestbook;

3.4.7 总结

本节讲述了有关表的大部分操作，现在将所述内容总结如下：

- MySQL 的表的三种类型
- 如何创建表、删除表
- 如何改变表的结构、名字
- 如何使用 mysqlshow 实用程序

3.5 向数据表插入行记录

3.5.1 使用 INSERT 语句插入新数据

语法：INSERT [INTO] tbl_name [(col_name,...)] VALUES (pression,...),…

INSERT [INTO] tbl_name SET col_name=expression, ...

让我们开始利用 INSERT 语句来增加记录，这是一个 SQL 语句，需要为它指定希望插入数据行的表或将值按行放入的表。INSERT 语句具有几种形式：

- 可指定所有列的值：

例如：

shell> mysql –u root –p

mysql> use mytest;

mysql> insert into worker values(“tom”,”tom@yahoo.com”);

“INTO”一词自 MySQL 3.22.5 以来是可选的。(这一点对其他形式的 INSERT 语句也成立。) VALUES 表必须包含表中每列的值，并且按表中列的存放次序给出。(一般，这就是创建表时列的定义次序。如果不能肯定的话，可使用 DESCRIBE tbl_name 来查看这个次序。)

- 使用多个值表，可以一次提供多行数据。

Mysql>insert into worker values(‘tom’,’tom@yahoo.com’),(‘paul’,’paul@yahoo.com’);

有多个值表的 INSERT ... VALUES 的形式在 MySQL 3.22.5 或以后版本中支持。

- 可以给出要赋值的那个列，然后再列出值。这对于希望建立只有几个列需要初始设置的记录是很有用的。

  例如：

  mysql>insert into worker (name) values ('tom');

  自 MySQL 3.22.5 以来，这种形式的 INSERT 也允许多个值表：

  mysql>insert into worker (name) values ('tom'), ('paul');

  在列的列表中未给出名称的列都将赋予缺省值。

  自 MySQL 3.22 .10 以来，可以 col_name = value 的形式给出列和值。

  例如：

  mysql>insert into worker set name='tom';

  在 SET 子句中未命名的行都赋予一个缺省值。

  使用这种形式的 INSERT 语句不能插入多行。

- 一个 expression 可以引用在一个值表先前设置的任何列。例如，你能这样：

  mysql> INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);

  但不能这样：

  mysql> INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);

3.5.2 使用 INSERT...SELECT 语句插入从其他表选择的行

当我们在上一节学习创建表时，知道可以使用 select 从其它表来直接创建表，甚至可以同时复制数据记录。如果你已经拥有了一个表，你同样可以从 select 语句的配合中获益。

从其它表中录入数据，例如：

mysql>insert into tbl_name1(col1,col2) select col3,col4 from tbl_name2;

你也可以略去目的表的列列表，如果你每一列都有数据录入。

mysql>insert into tbl_name1 select col3,col4 from tbl_name2;

INSERT INTO ... SELECT 语句满足下列条件：

- 查询不能包含一个 ORDER BY 子句。
- INSERT 语句的目的表不能出现在 SELECT 查询部分的 FROM 子句，因为这在 ANSI SQL 中被禁止让从你正在插入的表中 SELECT。（问题是 SELECT 将可能发现在同一个运行期间内先前被插入的记录。当使用子选择子句时，情况能很容易混淆）

3.5.3 使用 replace、replace...select 语句插入

REPLACE 功能与 INSERT 完全一样，除了如果在表中的一个老记录具有在一个唯一索引上的新记录有相同的值，在新记录被插入之前，老记录被删除。对于这种情况，insert 语句的表现是产生一个错误。

REPLACE 语句也可以褐 SELECT 相配合，所以上两小节的内容完全适合 REPALCE.。

应该注意的是，由于 REPLACE 语句可能改变原有的记录，因此使用时要小心。

3.5.4 使用 LOAD 语句批量录入数据

本章的前面讨论如何使用 SQL 向一个表中插入数据。但是，如果你需要向一个表中添加许多条记录，使用 SQL 语句输入数据是很不方便的。幸运的是，MySQL 提供了一些方法用于批量录入数据，使得向表中添加数据变得容易了。本节以及下一节，将介绍这些方法。本节将介绍SQL 语言级的解决方法。

1、基本语法

语法：LOAD DATA [LOCAL] INFILE 'file_name.txt' [REPLACE | IGNORE] INTO TABLE tbl_name

LOAD DATA INFILE 语句从一个文本文件中以很高的速度读入一个表中。如果指定 LOCAL 关键词，从客户主机读文件。如果 LOCAL 没指定，文件必须位于服务器上。(LOCAL 在 MySQL3.22.6 或以后版本中可用。)

为了安全原因，当读取位于服务器上的文本文件时，文件必须处于数据库目录或可被所有人读取。另外，为了对服务器上文件使用 LOAD DATA INFILE，在服务器主机上你必须有 file 的权限。见第七章 数据库安全。

REPLACE 和 IGNORE 关键词控制对现有的唯一键记录的重复的处理。如果你指定 REPLACE，新行将代替有相同的唯一键值的现有行。如果你指定 IGNORE，跳过有唯一键的现有行的重复行的输入。如果你不指定任何一个选项，当找到重复键键时，出现一个错误，并且文本文件的余下部分被忽略时。

如果你使用 LOCAL 关键词从一个本地文件装载数据，服务器没有办法在操作的当中停止文件的传输，因此缺省的行为好像 IGNORE 被指定一样。

2、文件的搜寻原则

当在服务器主机上寻找文件时，服务器使用下列规则：

- 如果给出一个绝对路径名，服务器使用该路径名。
- 如果给出一个有一个或多个前置部件的相对路径名，服务器相对服务器的数据目录搜索文件。
- 如果给出一个没有前置部件的一个文件名，服务器在当前数据库的数据库目录寻找文件。

注意这些规则意味着一个像“./myfile.txt”给出的文件是从服务器的数据目录读取，而作为“myfile.txt”给出的一个文件是从当前数据库的数据库目录下读取。也要注意，对于下列哪些语句，对 db1 文件从数据库目录读取，而不是 db2：

```
mysql> USE db1;
mysql> LOAD DATA INFILE "./data.txt" INTO TABLE db2.my_table;
```

3、FIELDS 和 LINES 子句的语法

如果你指定一个 FIELDS 子句，它的每一个子句(TERMINATED BY, [OPTIONALLY] ENCLOSED BY 和 ESCAPED BY)也是可选的，除了你必须至少指定他们之一。

如果你不指定一个 FIELDS 子句，缺省值与如果你这样写的相同：

FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'

如果你不指定一个 LINES 子句，缺省值与如果你这样写的相同：

LINES TERMINATED BY '\n'

换句话说，缺省值导致读取输入时，LOAD DATA INFILE 表现如下：

- 在换行符处寻找行边界
- 在定位符处将行分进字段
- 不要期望字段由任何引号字符封装
- 将由“\”开头的定位符、换行符或“\”解释是字段值的部分字面字符

LOAD DATA INFILE 能被用来读取从外部来源获得的文件。例如，以 dBASE 格式的文件将有由逗号分隔并用双引号包围的字段。如果文件中的行由换行符终止，下面显示的命令说明你将用来装载文件的字段和行处理选项：

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE tbl_name
        FIELDS TERMINATED BY ',' ENCLOSED BY '"'
        LINES TERMINATED BY '\n';
```

任何字段或行处理选项可以指定一个空字符串('')。如果不是空，FIELDS [OPTIONALLY] ENCLOSED BY 和 FIELDS ESCAPED BY 值必须是一个单个字符。FIELDS TERMINATED BY 和 LINES TERMINATED BY 值可以是超过一个字符。例如，写入由回车换行符对（CR+LF）终止的行，或读取包含这样行的一个文件，指定一个 LINES TERMINATED BY '\r\n'子句。

FIELDS [OPTIONALLY] ENCLOSED BY 控制字段的包围字符。对于输出(SELECT ... INTO OUTFILE)，如果你省略 OPTIONALLY，所有的字段由 ENCLOSED BY 字符包围。对于这样的输出的一个例子(使用一个逗号作为字段分隔符)显示在下面：

```
"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"
```

如果你指定 OPTIONALLY，ENCLOSED BY 字符仅被用于包围 CHAR 和 VARCHAR 字段：

```
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，一个字段值中的 ENCLOSED BY 字符的出现通过用 ESCAPED BY 字符作为其前缀来转义。也要注意，如果你指定一个空 ESCAPED BY 值，可能产生不能被 LOAD DATA INFILE 正确读出的输出。例如，如果转义字符为空，上面显示的输出显示如下。注意到在第四行的第二个字段包含跟随引号的一个逗号，它(错误地)好象要终止字段：

```
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
```

FIELDS ESCAPED BY 控制如何写入或读出特殊字符。如果 FIELDS ESCAPED BY 字符不是空的，它被用于前缀在输出上的下列字符：

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（实际上将后续转义字符写成 ASCII'0'，而不是一个零值字节）

如果 FIELDS ESCAPED BY 字符是空的，没有字符被转义。指定一个空转义字符可能不是一个好主意，特别是如果在你数据中的字段值包含刚才给出的表中的任何字符。

对于输入，如果 FIELDS ESCAPED BY 字符不是空的，该字符的出现被剥去并且后续字符在字面上作为字段值的一个部分。例外是一个转义的“0”或“N”（即，\0 或\N，如果转义字符是“\”)。这些序列被解释为 ASCII 0（一个零值字节）和 NULL。见下面关于 NULL 处理的规则。

3.5.5 总结

为数据库装载数据是管理员的重要职责之一，正因为重要，所以 MySQL 提供的方法也是非常繁多。其中主要的在本节已经列举：

1、 使用 INSERT、REPLACE 语句

2、 使用 INSERT/REPLACE…SELECT 语句

3、 使用 LOAD DATA INFILE 语句

4、 使用实用程序 mysqlimport（将在第五章介绍）

3.6 查询数据表中的记录

除非最终检索它们并利用它们来做点事情，否则将记录放入数据库没什么好处。这就是 SELECT 语句的用途，即帮助取出数据。SELECT 大概是 SQL 语言中最常用的语句，而且怎样使用它也最为讲究；用它来选择记录可能相当复杂，可能会涉及许多表中列之间的比较。本节介绍 Select 语句关于查询的最基本功能。

SELECT 语句的语法如下：

SELECT selection_list　选择哪些列
FROM table_list　从何处选择行
WHERE primary_constraint　行必须满足什么条件
GROUP BY grouping_columns　怎样对结果分组
HAVING secondary_constraint　行必须满足的第二条件
ORDER BY sorting_columns 怎样对结果排序
LIMIT count　结果限定

 注意：所有使用的关键词必须精确地以上面的顺序给出。例如，一个 HAVING 子句必须跟在 GROUP BY 子句之后和 ORDER BY 子句之前。

除了词“SELECT”和说明希望检索什么的 column_list 部分外，语法中的每样东西都是可选的。有的数据库还需要 FROM 子句。MySQL 有所不同，它允许对表达式求值而不引用任何表。

3.6.1 普通查询

- SELECT 最简单的形式是从一张表中检索每样东西：

mysql> SELECT * FROM pet;
其结果为：

```
+---------+-------+---------+-----+------------+------------+
| 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       |
| Chirpy  | Gwen  | bird    | f   | 1998-09-11 | NULL       |
| Fang    | Benny | dog     | m   | 1990-08-27 | NULL       |
| Bowser  | Diane | dog     | m   | 1990-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
| Slim    | Benny | snake   | m   | 1996-04-29 | NULL       |
| Puffball | Diane | hamster | f   | 1999-03-30 | NULL       |
+---------+-------+---------+-----+------------+------------+
```

- 查询特定行：

你能从你的表中只选择特定的行。例如，如果你想要验证你对 Bowser 的出生日期所做的改变，像这样精选 Bowser 的记录：

mysql> SELECT * FROM pet WHERE name = "Bowser";
其结果为：

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

```
+--------+------+--------+-----+------------+------------+
| Bowser | Diane | dog     | m    | 1990-08-31 | 1995-07-29 |
+--------+------+--------+-----+------------+------------+
```

你可以对照前一个例子来验证。

- 查询特定列

如果你不想要看到你的表的整个行，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，精选 name 和 birth 列：

mysql> SELECT name, birth FROM pet where owner="Gwen";

其结果为：

```
+---------+-----------+
| name     | birth      |
+---------+-----------+
| Claws    | 1994-03-17 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
+---------+-----------+
```

- 进行表达式计算

前面的多数查询通过从表中检索值已经产生了输出结果。MySQL 还允许作为一个公式的结果来计算输出列的值。表达式可以简单也可以复杂。下面的查询求一个简单表达式的值（常量）以及一个涉及几个算术运算符和两个函数调用的较复杂的表达式的值。例如，计算 Browser 生活的天数：

mysql> SELECT death-birth FROM pet WHERE name="Bowser";

其结果是：

```
+------------+
| death-birth |
+------------+
|       49898 |
+------------+
```

由于 MySQL 允许对表达式求值而不引用任何表。所以也可以这样使用:

mysql>select (2+3*4.5)/2.5;

其结果为：

```
+--------------+
| (2+3*4.5)/2.5 |
+--------------+
|         6.200 |
+--------------+
```

3.6.2 条件查询

不必每次查询都返回所有的行记录，你能从你的表中只选择特定的行。为此你需要使用 WHERE 或者 HAVING 从句。HAVING 从句与 WHERE 从句的区别是，HAVING 表达的是第二条件,在与其他从句配合使用,显然不能在 WHERE 子句中的项目使用 HAVING。因此本小节紧介绍 WHERE 从句的使用，HAVING 从句的使用方法类似。另外 WHERE 从句也可以实现 HAVING 从句的绝大部分功能。

为了限制 SELECT 语句检索出来的记录集，可使用 WHERE 子句，它给出选择行的条件。可通过查找满足各种条件的列值来选择行。

WHERE 子句中的表达式可使用表 1-1 中的算术运算符、表 1-2 的比较运算符和表 1-3 的逻辑运算符。还可以使用圆括号将一个表达式分成几个部分。可使用常量、表列和函数来完成运算。在本教程的查询中，我们有时使用几个 MySQL 函数，但是 MySQL 的函数远不止这里给出的这些。请参阅附录 一，那里给出了所有 MySQL 函数的清单。

表 3-1 算术运算符

运算符	说明	运算符	说明
+	加	*	乘
-	减	/	除

表 3-2 比较运算符

运算符	说明	运算符	说明
<	小于	!= 或 <>	不等于
<=	小于或等于	>=	大于或等于
=	等于	>	大于

表 3-3 逻辑运算符

运算符	说明
NOT 或 ！	逻辑非
OR 或 ||	逻辑或
AND 或 &&	逻辑与

例如，如果你想要验证你对 Bowser 的出生日期所做的改变，像这样精选 Bowser 的记录：

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

```
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1990-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
```

输出证实出生年份现在正确记录为 1990，而不是 1909。

字符串比较通常是大小些无关的，因此你可以指定名字为"bowser"、"BOWSER"等等，查询结果将是相同的。

你能在任何列上指定条件，不只是 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  |
+----------+-------+---------+------+------------+-------+
```

你能组合条件，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。如果你这样做，使用括号指明条件应该如何被分组是一个好主意：

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

3.6.3 查询排序

使用 ORDER BY 子句对查询返回的结果按一列或多列排序。ORDER BY 子句的语法格式为：

```
ORDER BY column_name [ASC|DESC] [,…]
```

其中 ASC 表示升序，为默认值，DESC 为降序。ORDER BY 不能按 text、text 和 image 数据类型进行排 序。另外，可以根据表达式进行排序。

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

为了以逆序排序，增加 DESC（下降)关键字到你正在排序的列名上：

```
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 |
| Bowser   | 1990-08-31 |
| Fang     | 1990-08-27 |
| Buffy    | 1989-05-13 |
+---------+-----------+
```

你能在多个列上排序。例如，按动物的种类排序，然后按生日，首先是动物种类中最年轻的动物，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 |
| Bowser   | dog     | 1990-08-31 |
| Fang     | dog     | 1990-08-27 |
| Buffy    | dog     | 1989-05-13 |
| Puffball | hamster | 1999-03-30 |
| Slim     | snake   | 1996-04-29 |
+---------+--------+-----------+
```

注意 DESC 关键词仅适用于紧跟在它之前的列名字(birth)；species 值仍然以升序被排序。注意，输出首先按照 species 排序，然后具有相同 species 的宠物再按照 birth 降序排列。

3.6.4 查询分组与行计数

GROUP BY 从句根据所给的列名返回分组的查询结果，可用于查询具有相同值的列。其语法为：

GROUP BY col_name，….

你可以为多个列分组。

例如：

mysql>SELECT * FROM pet GROUP BY species;

```
+---------+-------+---------+-----+-----------+-------+
| name    | owner | species | sex | birth     | death |
+---------+-------+---------+-----+-----------+-------+
| Chirpy  | Gwen  | bird    | f   | 1998-09-11 | NULL  |
| Fluffy  | Harold | cat    | f   | 1993-02-04 | NULL  |
| Buffy   | Harold | dog    | f   | 1989-05-13 | NULL  |
| Puffball | Diane | hamster | f   | 1999-03-30 | NULL  |
| Slim    | Benny | snake   | m   | 1996-04-29 | NULL  |
+---------+-------+---------+-----+-----------+-------+
```

由以上结果可以看出：

查询显示结果时，被分组的列如果有重复的值，只返回靠前的记录，并且返回的记录集是排序的。这并不是一个很好的结果。仅仅使用 GROUP BY 从句并没有什么意义，该从句的真正作用在于与各种组合函数配合，用于行计数。

1、COUNT()函数计数非 NULL 结果的数目。

你可以这样计算表中记录行的数目：

mysql> select count(*) from pet;

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

计算 sex 为非空的记录数目：

mysql> select count(sex) from pet;

```
+------------+
| count(sex) |
+------------+
|          8 |
+------------+
```

现在配合 GROUP BY 从句使用。

例如：要知道每个主人有多少宠物

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Benny  |        2 |
| Diane  |        2 |
| Gwen   |        3 |
| Harold |        2 |
+--------+----------+
```

又如，每种宠物的个数：

```
mysql> SELECT species,count(*) FROM pet GROUP BY species;
+---------+----------+
| species | count(*) |
+---------+----------+
| bird    |        2 |
| cat     |        2 |
| dog     |        3 |
| hamster |        1 |
| snake   |        1 |
+---------+----------+
```

如果你除了计数还返回一个列的值，那么必须使用 GROU BY 语句，否则无法计算记录。例如上例，使用 GROUP BY 对每个 owner 分组所有记录，没有它，你得到的一切是一条错误消息：

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

也可以根据多个列分组，例如：

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

3.6.5 查询多个表

查询多个表，FROM 子句列出表名，并用逗号分隔，因为查询需要从他们两个拉出信息。

当组合(联结-join)来自多个表的信息时，你需要指定在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。这通过把表名附在列名前做到。

现在有一个 event 表：

mysql>select * from event;

```
+---------+----------+---------+------------------------+
| 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  | NULL                   |
| Fang    | 1991-10-12 | kennel  | NULL                   |
| 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         |
+---------+----------+---------+------------------------+
```

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。event 表指出何时发生，但是为了计算母亲的年龄，你需要她的出生日期。既然它被存储在 pet 表中，为了查询你需要两张表：

```
mysql> SELECT pet.name, (TO_DAYS(date) - TO_DAYS(birth))/365 AS age, remark
    -> FROM pet, event
    -> WHERE pet.name = event.name AND type = "litter";
```

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

同样方法也可用于同一张表中，你不必有 2 个不同的表来执行一个联结。如果你想要将一个表的记录与同一个表的其他记录进行比较，联结一个表到自身有时是有用的。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关联于哪个表实例更直观。

3.3.6 总结

本文总结了 SELECT 语句检索记录的简单使用方法。其中涉及到的内容主要包括以下一些内容：

1、 WHERE 从句的用法
2、 GROUP BY 从句的用法
3、 ORDER BY 从句的用法
4、 连接多个表的简单介绍

3.7 修改、删除数据记录

有时，希望除去某些记录或更改它们的内容。DELETE 和 UPDATE 语句令我们能做到这一点。

3.7.1 用 update 修改记录

UPDATE tbl_name SET 要更改的列

WHERE 要更新的记录

这里的 WHERE 子句是可选的，因此如果不指定的话，表中的每个记录都被更新。

例如，在 pet 表中，我们发现宠物 Whistler 的性别没有指定，因此我们可以这样修改这个记录：

mysql> update pet set sex='f' where name='' Whistler";

3.7.2 用 delete 删除记录

DELETE 语句有如下格式：

DELETE FROM tbl_name WHERE 要删除的记录

WHERE 子句指定哪些记录应该删除。它是可选的，但是如果不选的话，将会删除所有的记录。这意味着最简单的 DELETE 语句也是最危险的。

这个查询将清除表中的所有内容。一定要当心！

为了删除特定的记录，可用 WHERE 子句来选择所要删除的记录。这类似于 SELECT 语句中的 WHERE 子句。

mysql> delete from pet where name="Whistler";

可以用下面的语句清空整个表：

mysql>delete from pet;

3.7.3 总结

本节介绍了两个 SQL 语句的用法。使用 UPDATE 和 DELETE 语句要十分小心，因为可能对你的数据造成危险。尤其是 DELETE 语句，很容易会删除大量数据。使用时，一定小心。

思考题

1、 请亲自按照本章所述的步骤，让 MySQL 服务器在 Linux 系统启动时，自动启动。并尝试其它启动、重启、关闭服务器的方法。

2、 现在有一个位于主机 database.domain.net 的 MySQL 服务器，用 root 用户的身份，密码为 newpass，连接到数据库 test。如何给出合适的命令行？如果使用选项文件，如何添加选项？

3、 在 test 数据库中建立一个本章举例中所述的表 pet，其结构如下所述：

name：30 个宽度的定长字符串

owner：30 个宽度的定长字符串

species：10 个宽度的定长字符串

sex：由 m 和 f 组成的非空枚举类型

birth：date 类型

death：date 类型

4、 本章中 pet 表的数据录入表中：

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
Chirpy	Gwen	bird	f	1998-09-11	NULL
Fang	Benny	dog	m	1990-08-27	NULL
Bowser	Diane	dog	m	1990-08-31	1995-07-29
Whistler	Gwen	bird	NULL	1997-12-09	NULL
Slim	Benny	snake	m	1996-04-29	NULL
Puffball	Diane	hamster	f	1999-03-30	NULL

请把数据记录到一个数据文件中，然后使用 LOAD DATA INFILE 语句装载数据。提示：如果在 Windows 环境中，那么文件的换行符是”\r\n”。

如果是使用实用程序 mysqlimport 命令行如何书写。

5、重复本章中的 3.6 节对表 pet 进行检索的例子。

第4章

MYSQL 高级特性

本章要点：

- 集合函数与时间函数
- 字符串的模式匹配
- 如何创建索引
- 检索数据中的一些技巧

第三章向你初步介绍了 SQL。你学会了如何用 SELECT 语句进行查询，你还学会了如何建立自己的表以及如何录入数据等。在这一章里，你将加深你 SQL 语言知识。你将学习如何建立索引来加快查询速度。你还将学会如果用更多的 SQL 语句和函数来操作表中的数据。

尤其是对检索语句 SELECT 的使用，其中技巧繁多，可以说这是 SQL 语言中最重要的语句，也是使用者最常使用的语句。本章将详细介绍 SELECT 语句的用法。

无论如何，这里的介绍都不能包括 SQL 语言的所有技巧，读者应该在平时的不断使用中积累经验。

4.1 集合函数

到现在为止，你只学习了如何根据特定的条件从表中取出一条或多条记录。但是，假如你想对一个表中的记录进行数据统计。例如，如果你想统计存储在表中的一次民意测验的投票结果。或者你想知道一个访问者在你的站点上平均花费了多少时间。要对表中的任何类型的数据进行统计，都需要使用集合函数。你可以统计记录数目，平均值，最小值，最大值，或者求和。当你使用一个集合函数时，它只返回一个数，该数值代表这几个统计值之一。

这些函数的最大特点就是经常和 GROUP BY 语句配合使用，需要注意的是集合函数不能和非分组的列混合使用。

4.1.1 行列计数

- 计算查询语句返回的记录行数

直接计算函数 COUNT(*)的值，例如，计算 pet 表中猫的只数：

```
mysql>SELECT count(*) FROM pet WHERE species='cat';
+----------+
| count(*) |
+----------+
|        2 |
+----------+
```

4.1.2 统计字段值的数目

例如，计算 pet 表中 species 列的数目：

```
mysql> SELECT count(species) FROM pet;
+----------------+
| count(species) |
+----------------+
|              9 |
+----------------+
```

如果相同的种类出现了不止一次，该种类将会被计算多次。如果你想知道种类为某个特定值的宠物有多少个，你可以使用 WHERE 子句，如下例所示：

```
mysql> SELECT COUNT(species) FROM pet WHERE species='cat' ;
```

注意这条语句的结果：

```
+----------------+
| COUNT(species) |
+----------------+
|              2 |
```

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

这个例子返回种类为'cat'的作者的数目。如果这个名字在表 pet 中出现了两次，则次函数的返回值是 2。 而且它和上面提到过的语句的结果是一致的：

SELECT count(*) FROM pet WHERE species='cat'

实际上，这两条语句是等价的。

假如你想知道有多少不同种类的的宠物数目。你可以通过使用关键字 DISTINCT 来得到该数目。如下例所示：

```
mysql> SELECT COUNT(DISTINCT species) FROM pet;
+----------------------+
| COUNT(DISTINCT species) |
+----------------------+
|                      5 |
+----------------------+
```

如果种类'cat'出现了不止一次，它将只被计算一次。关键字 DISTINCT 决定了只有互不相同的值才被计算。

通常，当你使用 COUNT（）时，字段中的空值将被忽略。

另外，COUNT()函数通常和 GROUP BY 子句配合使用，例如可以这样返回每种宠物的数目：

```
mysql> SELECT species,count(*) FROM pet GROUP BY species;
+---------+---------+
| species | count(*) |
+---------+---------+
| bird    |        2 |
| cat     |        2 |
| dog     |        3 |
| hamster |        1 |
| snake   |        1 |
+---------+---------+
```

4.1.3 计算字段的平均值

需要计算这些值的平均值。使用函数 AVG（），你可以返回一个字段中所有值的平均值。

假如你对你的站点进行一次较为复杂的民意调查。访问者可以在 1 到 10 之间投票，表示他们喜欢你站点的程度。你把投票结果保存在名为 vote 的 INT 型字段中。要计算你的用户投票的平均值，你需要使用函数 AVG（）：

SELECT AVG(vote) FROM opinion

这个 SELECT 语句的返回值代表用户对你站点的平均喜欢程度。函数 AVG（）只能对数值型字段使用。这个函数在计算平均值时也忽略空值。

再给出一个实际例子，例如我们要计算 pet 表中每种动物年龄的平均值，那么使用AVG()函数和 GROUP BY 子句：

mysql> SELECT species,AVG(CURDATE()-birth) FROM pet GROUP BY species;

返回的结果为：

```
+---------+--------------------+
| species | AVG(CURDATE()-birth) |
+---------+--------------------+
| bird    |              34160 |
| cat     |            74959.5 |
| dog     |    112829.66666667 |
| hamster |              19890 |
| snake   |              49791 |
+---------+--------------------+
```

4.1.4 计算字段值的和

假设你的站点被用来出售某种商品，已经运行了两个月，是该计算赚了多少钱的时候了。假设有一个名为 orders 的表用来记录所有访问者的定购信息。要计算所有定购量的总和，你可以使用函数 SUM（）：

SELECT SUM(purchase_amount) FROM orders

函数 SUM（）的返回值代表字段 purchase_amount 中所有值的总和。字段 purchase_amount 的数据类型也许是 DECIMAL 类型，但你也可以对其它数值型字段使用函数 SUM（）。

用一个不太恰当的例子说明，我们计算 pet 表中同种宠物的年龄的总和：

mysql> SELECT species,SUM(CURDATE()-birth) FROM pet GROUP BY species;

你可以查看结果，与前一个例子对照：

```
+---------+--------------------+
| species | SUM(CURDATE()-birth) |
+---------+--------------------+
| bird    |              68320 |
| cat     |             149919 |
| dog     |             338489 |
| hamster |              19890 |
| snake   |              49791 |
+---------+--------------------+
```

4.1.5 计算字段值的极值

求字段的极值，涉及两个函数 MAX()和 MIN()。

例如，还是 pet 表，你想知道最早的动物出生日期，由于日期最早就是最小，所以可

以使用 MIN()函数：

```
mysql> SELECT MIN(birth) FROM pet;
+------------+
| MIN(birth) |
+------------+
| 1989-05-13 |
+------------+
```

但是，你只知道了日期，还是无法知道是哪只宠物，你可能想到这样做：

```
SELECT name,MIN(birth) FROM pet;
```

但是，这是一个错误的 SQL 语句，因为集合函数不能和非分组的列混合使用，这里 name 列是没有分组的。所以，你无法同时得到 name 列的值和 birth 的极值。

MIN()函数同样可以与 GROUP BY 子句配合使用，例如，找出每种宠物中最早的出生日期：

```
mysql> SELECT species,MIN(birth) FROM pet GROUP BY species;
```

下面是令人满意的结果：

```
+---------+------------+
| species | MIN(birth) |
+---------+------------+
| bird    | 1997-12-09 |
| cat     | 1993-02-04 |
| dog     | 1989-05-13 |
| hamster | 1999-03-30 |
| snake   | 1996-04-29 |
+---------+------------+
```

另一方面，如果你想知道最近的出生日期，就是日期的最大值，你可以使用 MAX()函数，如下例所示：

```
mysql> SELECT species,MAX(birth) FROM pet GROUP BY species;
+---------+------------+
| species | MAX(birth) |
+---------+------------+
| bird    | 1998-09-11 |
| cat     | 1994-03-17 |
| dog     | 1990-08-31 |
| hamster | 1999-03-30 |
| snake   | 1996-04-29 |
+---------+------------+
```

4.1.6 总结

在本节中，介绍了一些典型的集合函数的用法，包括计数、均值、极值和总和，这些都是 SQL 语言中非常常用的函数。

这些函数之所以称之为集合函数，是因为它们应用在多条记录中，所以集合函数最常见的用法就是与 GROUP BY 子句配合使用，最重要的是集合函数不能同未分组的列混合使用。

4.2 操作日期和时间

日期和时间函数对建立一个站点是非常有用的。站点的主人往往对一个表中的数据何时被更新感兴趣。通过日期和时间函数，你可以在秒级跟踪一个表的改变。

日期和时间类型是 DATETIME、DATE、TIMESTAMP、TIME 和 YEAR。这些的每一个都有合法值的一个范围，而“零”当你指定确实不合法的值时被使用。注意，MySQL 允许你存储某个“不严格地”合法的日期值，例如 1999-11-31，原因我们认为它是应用程序的责任来处理日期检查，而不是 SQL 服务器。为了使日期检查更“快”，MySQL 仅检查月份在 0-12 的范围，天在 0-31 的范围。上述范围这样被定义是因为 MySQL 允许你在一个 DATE 或 DATETIME 列中存储日期，这里的天或月是零。这对存储你不知道准确的日期的一个生日的应用程序来说是极其有用的，在这种情况下，你简单地存储日期象 1999-00-00 或 1999-01-00。(当然你不能期望从函数如 DATE_SUB()或 DATE_ADD()得到类似以这些日期的正确值)。

4.2.1 返回当前日期和时间

通过函数 GETDATE（），你可以获得当前的日期和时间。例如，

- CURDATE() 返回当前日期

CURRENT_DATE

以'YYYY-MM-DD'或 YYYYMMDD 格式返回今天日期值，取决于函数是在一个字符串还是数字上下文被使用。

```
mysql> select CURDATE();
+------------+
| CURDATE()  |
+------------+
| 2001-02-20 |
+------------+
mysql> select CURDATE() + 0;
+-------------+
| CURDATE()+0 |
+-------------+
|    20010220 |
```

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

- CURTIME() 返回当前时间

以'HH:MM:SS'或 HHMMSS 格式返回当前时间值，取决于函数是在一个字符串还是在数字的上下文被使用。

```
mysql> select CURTIME();
+-----------+
| CURTIME() |
+-----------+
| 10:42:38  |
+-----------+
mysql> select CURTIME() + 0;
+-------------+
| CURTIME()+0 |
+-------------+
|      104525 |
+-------------+
```

- NOW() 返回当前时期和时间

NOW()以 YYYY-MM-DD HH:MM:SS 的格式或者 YYYYMMDDHHMMSS 的格式返回日期和时间值，取决于上下文。

```
mysql>select now();
+--------------------+
| now()              |
+--------------------+
| 2001-02-20 10:45:57 |
+--------------------+
mysql>select now()+0;
+---------------+
| now()+0       |
+---------------+
| 20010220105635 |
+---------------+
```

这些得到当前日期和时间的函数，对于日期和时间的计算很方便，尤其是计算一个时间到现在的时间差。例如，在 pet 表中，我们以天为单位计算宠物的年龄：

```
mysql> SELECT name,CURDATE()-birth FROM pet;
+---------+---------------+
| name    | CURDATE()-birth |
+---------+---------------+
| Fluffy  |          80016 |
```

```
| Claws    |          69903 |
| Buffy    |         119707 |
| Chirpy   |          29309 |
| Fang     |         109393 |
| Bowser   |         109389 |
| Whistler |          39011 |
| Slim     |          49791 |
| Puffball |          19890 |
+----------+----------------+
```

4.2.2 自动记录数据的改变时间

TIMESTAMP 列类型提供一种类型,TIMESTAMP 值可以从 1970 的某时的开始一直到 2037 年，精度为一秒，其值作为数字显示。你可以使用它自动地用当前的日期和时间标记 INSERT 或 UPDATE 的操作。如果你有多个 TIMESTAMP 列，只有第一个自动更新。

自动更新第一个 TIMESTAMP 列在下列任何条件下发生：

- 列没有明确地在一个 INSERT 或 LOAD DATA INFILE 语句中指定。
- 列没有明确地在一个 UPDATE 语句中指定且一些另外的列改变值。（注意一个 UPDATE 设置一个列为它已经有的值，这将不引起 TIMESTAMP 列被更新，因为如果你设置一个列为它当前的值，MySQL 为了效率而忽略更改。）
- 你明确地设定 TIMESTAMP 列为 NULL.

除第一个以外的 TIMESTAMP 列也可以设置到当前的日期和时间，只要将列设为 NULL，或 NOW()。

例如，创建如下的表：

```
mysql> CREATE TABLE student
    -> (
    -> id int,
    -> name char(16),
    -> english tinyint,
    -> chinese tinyint,
    -> history tinyint,
    -> time timestamp
    -> );
```

向表中插入记录，可以查看效果：

```
mysql> INSERT student(id,name,englisht,Chinese,history) VALUES(11,”Tom”,66,93,67);
```

查看记录的存储情况：

```
mysql> SELECT * FROM student;

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

```
| id   | name    | english | chinese | history | time           |
+------+---------+---------+---------+---------+----------------+
|   11 | Tom     |      66 |      93 |      67 | 20010220123335 |
+------+---------+---------+---------+---------+----------------+
```

你可以看到 time 列纪录下了数据录入时的时间值。如果你更新改记录，在查看操作的结果：

```
mysql> UPDATE student SET english=76 WHERE id=11;
mysql> SELECT * FROM student;
+------+------+---------+---------+---------+----------------+
| id   | name | english | chinese | history | time           |
+------+------+---------+---------+---------+----------------+
|   11 | Tom  |      76 |      93 |      67 | 20010220125736 |
+------+------+---------+---------+---------+----------------+
```

可以清楚的看到，time 列的时间被自动更改为修改记录的时间。

有时候你希望不更改任何值，也能打到修改 TIMESTAMP 列的值，这时只要设置该列的值为 NULL，MySQL 就可以自动更新 TIMESTAMP 列的值：

```
mysql> UPDATE student SET time=NULL WHERE id=11;
mysql> select * from student where id=11;
+------+------+---------+---------+---------+----------------+
| id   | name | english | chinese | history | time           |
+------+------+---------+---------+---------+----------------+
|   11 | Tom  |      76 |      93 |      67 | 20010220130517 |
+------+------+---------+---------+---------+----------------+
```

通过明确地设置希望的值，你可以设置任何 TIMESTAMP 列为不同于当前日期和时间的值，即使对第一个 TIMESTAMP 列也是这样。例如，如果，当你创建一个行时，你想要一个 TIMESTAMP 被设置到当前的日期和时间，但在以后无论何时行被更新时都不改变，你可以使用这样使用：

- 让 MySQL 在行被创建时设置列，这将初始化它为当前的日期和时间。
- 当你执行随后的对该行中其他列的更改时，明确设定 TIMESTAMP 列为它的当前值。

例如，当你在修改列时，可以把原有的值付给 TIMESTAMP 列：

```
mysql> UPDATE student SET english=66,time=time WHERE id=11;
mysql> select * from student where id=11;
+------+------+---------+---------+---------+----------------+
| id   | name | english | chinese | history | time           |
+------+------+---------+---------+---------+----------------+
|   11 | Tom  |      66 |      93 |      67 | 20010220130517 |
```

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

另一方面，你可能发现，当你想要实现上面这个效果时，很容易用一个你用 NOW() 初始化的 DATETIME 列然后不再改变它，这样也许直接些。 但是，TIMESTAMP 列的以后好处是存储要求比较小，节省空间。TIMESTAMP 的存储需求是 4 字节，而 DATETIME 列的存储需求是 8 字节。

4.2.3 返回日期和时间范围

当你分析表中的数据时，你也许希望取出某个特定时间的数据。我们用下面一个表来模仿一个 web 站点的记录。

```
mysql> CREATE TABLE weblog
    -> (
    -> data float,
    -> entrydate datetime
    -> );
```

然后随机的增加几个数据：

```
mysql> INSERT weblog VALUES(rand(),now());
```

rand()函数返回一个随机的浮点值，now()函数返回当前时间。多执行上面语句几次，得到一个作为测试的表。

最为测试你还可以增加一个值：

```
mysql> INSERT weblog VALUES(rand(),"2001-02-08");
```

这条语句，插入一个 entry 为"2001-02-08 00:00:00"的值（假定现在为 2001 年 2 月 8 日），你可以查看这个表的值：

```
mysql> select * from weblog;
+----------+---------------------+
| data     | entrydate           |
+----------+---------------------+
|  0.973723 | 2001-02-08 00:00:00 |
|  0.437768 | 2001-02-08 13:57:06 |
|  0.327279 | 2001-02-08 13:57:09 |
| 0.0931809 | 2001-02-08 13:58:29 |
|  0.198805 | 2001-02-08 13:57:54 |
+----------+---------------------+
```

你也许对特定的某一天中——比如说 2001 年 2 月 18 日——访问者在你站点上的活动感兴趣。要取出这种类型的数据，你也许会试图使用这样的 SELECT 语句：

```
mysql> SELECT * FROM weblog WHERE entrydate="2001-02-08"
```

不要这样做。这个 SELECT 语句不会返回正确的记录——它将只返回值为 2000-02-08 00:00:00 的记录，换句话说，只返回当天零点零时的记录。上面语句的结果为：

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

```
| data      | entrydate           |
+---------+-------------------+
| 0.973723 | 2001-02-08 00:00:00 |
+---------+-------------------+
```

要返回正确的记录，你需要适用日期和时间范围。有不止一种途径可以做到这一点。

1、使用关系运算符和逻辑运算符来限制时间范围

例如，下面的这个 SELECT 语句将能返回正确的记录：

```
mysql> SELECT * FROM weblog
    -> WHERE  entrydate>="2001-02-08" AND entrydate<"2001-02-09" ;
```

这个语句可以完成任务，因为它选取的是表中的日期和时间大于等于 2001-02-08 00:00:00 并小于 2001-02-09 00:00:00 的记录。换句话说，它将正确地返回 2000 年 2 月 8 日这一天输入的每一条记录。 其结果为：

```
+----------+-------------------+
| data      | entrydate           |
+----------+-------------------+
|  0.973723 | 2001-02-08 00:00:00 |
|  0.437768 | 2001-02-08 13:57:06 |
|  0.327279 | 2001-02-08 13:57:09 |
| 0.0931809 | 2001-02-08 13:58:29 |
|  0.198805 | 2001-02-08 13:57:54 |
+----------+-------------------+
```

2、另一种方法是，你可以使用 LIKE 来返回正确的记录。通过在日期表达式中包含通配符“%”，你可以匹配一个特定日期的所有时间。

这里有一个例子：

```
mysql> SELECT * FROM weblog WHERE entrydate LIKE '2001-02-08%' ;
```

这个语句可以匹配正确的记录。因为通配符“%”代表了任何时间。

```
+----------+-------------------+
| data      | entrydate           |
+----------+-------------------+
|  0.973723 | 2001-02-08 00:00:00 |
|  0.437768 | 2001-02-08 13:57:06 |
|  0.327279 | 2001-02-08 13:57:09 |
| 0.0931809 | 2001-02-08 13:58:29 |
|  0.198805 | 2001-02-08 13:57:54 |
+----------+-------------------+
```

3、上面两种方法的异同

由于使用关系运算符进行的是比较过程，时转换成内部的存储格式后进行的，因此，

因此时间的书写可以不是那么严格要求。

例如，下面几种写法是等价的：

```
mysql> SELECT * FROM weblog WHERE entrydate>="2001-02-08";
mysql> SELECT * FROM weblog WHERE entrydate>="2001-2-8";
mysql> SELECT * FROM weblog WHERE entrydate>="2001*02*08";
mysql> SELECT * FROM weblog WHERE entrydate>="20010208";
SELECT * FROM weblog WHERE entrydate>="2001/2/8";
```

而使用 LIKE 运算符和模式匹配，是通过比较串值进行的，因此必须使用标准的时间书写格式，YYYY-MM-DD HH-MM-SS。

4.2.5 比较日期和时间

已知两个日期，比较它们的前后，可以直接求出它们的差和零值比较，也可以利用已知的时间函数：

TO_DAYS(date)

给出一个日期 date，返回一个天数(从 0 年的天数)，date 可以是一个数字，也可以是一个串值，当然更可以是包含日期的时间类型。

```
mysql> select TO_DAYS(960501);
+-----------------+
| TO_DAYS(960501) |
+-----------------+
|          729145 |
+-----------------+
mysql> select TO_DAYS('1997-07-01');
+----------------------+
| TO_DAYS('1997-07-01') |
+----------------------+
|                729571 |
+----------------------+
```

例如：返回 2 个时间相差的天数(21 世纪已经过去了多少天)

```
mysql> select to_days(now())-to_days('20010101');
+-----------------------------------------------+
| to_days(now()-00000012000000)-to_days('20010101') |
+-----------------------------------------------+
|                                            38 |
+-----------------------------------------------+
```

4.3 字符串模式匹配

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。

4.3.1 标准的 SQL 模式匹配

SQL 的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(包括零个字符)。在 MySQL 中，SQL 的模式缺省是忽略大小写的。下面显示一些例子。注意在你使用 SQL 模式时，你不能使用=或!=；而使用 LIKE 或 NOT LIKE 比较操作符。

例如，在表 pet 中，为了找出以“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 |
+--------+-------+---------+------+------------+------------+
```

为了找出以“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  |
+--------+-------+---------+------+------------+-------+
```

为了找出包含一个“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       |
+----------+-------+---------+------+------------+------------+
```

为了找出包含正好 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  |
+-------+--------+---------+------+------------+-------+
```

4.3.2 扩展正则表达式模式匹配

由 MySQL 提供的模式匹配的其他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。

一个字符类“[...]”匹配在方括号内的任何字符。例如，“[abc]”匹配“a”、“b”或“c”。为了命名字符的一个范围，使用一个“-”。“[a-z]”匹配任何小写字母，而“[0-9]”匹配任何数字。

“ * ”匹配零个或多个在它前面的东西。例如，“x*”匹配任何数量的“x”字符，“[0-9]*”匹配的任何数量的数字，而“.*”匹配任何数量的任何东西。

正则表达式是区分大小写的，但是如果你希望，你能使用一个字符类匹配两种写法。例如，“[aA]”匹配小写或大写的“a”而“[a-zA-Z]”匹配两种写法的任何字母。

如果它出现在被测试值的任何地方，模式就匹配(只要他们匹配整个值，SQL 模式匹配)。

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，上面所示的 LIKE 查询在下面使用 REGEXP 重写：

为了找出以“b”开头的名字，使用“^”匹配名字的开始并且“[bB]”匹配小写或大写的“b”：

```
mysql> SELECT * FROM pet WHERE name REGEXP "^[bB]";
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

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“[wW]”匹配小写或大写的“w”：

```
mysql> SELECT * FROM pet WHERE name REGEXP "[wW]";
+----------+-------+---------+-----+------------+------------+
| name     | owner | species | sex | birth      | death      |
+----------+-------+---------+-----+------------+------------+
| Claws    | Gwen  | cat     | m   | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m   | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+-----+------------+------------+
```

既然如果一个正规表达式出现在值的任何地方，其模式匹配了，就不必再先前的查询中在模式的两方面放置一个通配符以使得它匹配整个值，就像如果你使用了一个 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”操作符重写先前的查询：

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

4.3.3 总结

4.3 节介绍了有关字符串模式匹配的有关知识。标准的 SQL 模式匹配是 SQL 语言的标准，可以被其它关系数据库系统接受。扩展正规表达式模式匹配是根据 Unix 系统的标准开发了，一般只可使用在 MySQL 上，但是其功能要比标准的 SQL 模式匹配更强。

4.4 深入 SELECT 的查询功能

本节将讲述 SELECT 语句的一些高级功能。

4.4.1 列和表的别名

4.4.1.1 列的别名

精选输出的列可以用列名、列别名或列位置在 ORDER BY 和 GROUP BY 子句引用，列位置从 1 开始。

例如，我们从 pet 表中检索出宠物和种类，直接引用列名：

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

其输出为：

```
+----------+---------+
| name     | species |
+----------+---------+
| Bowser   | dog     |
| Buffy    | dog     |
| Chirpy   | bird    |
| Claws    | cat     |
| Fang     | dog     |
| Fluffy   | cat     |
| Puffball | hamster |
| Slim     | snake   |
| Whistler | bird    |
+----------+---------+
```

在子句中使用列的位置：

```
mysql> SELECT name,species FROM pet ORDER BY 1,2;
```

这条语句的输出与上面并无不同。

最后，你还可以为列命名：

```
mysql> SELECT name AS n,species AS s FROM pet ORDER BY n,s;
```

注意返回的结果：

```
+----------+---------+
| n        | s       |
```

```
+---------+--------+
| Bowser  | dog    |
| Buffy   | dog    |
| Chirpy  | bird   |
| Claws   | cat    |
| Fang    | dog    |
| Fluffy  | cat    |
| Puffball | hamster |
| Slim    | snake  |
| Whistler | bird   |
+---------+--------+
```

返回的记录顺序并无不同。但是列的名字有了改变，这一点在使用 CREATE TABLE...SELECT 语句创建表时是有意义的。

例如，我们想从 pet 表生成包括其中 name，owner 字段的表，但是想把 name 和 owner 字段的名字重新命名为 animal 和 child，一个很笨的方法就是创建表再录入数据，如果使用别名，则仅仅一条 SQL 语句就可以解决问题，非常简单，我们要使用的语句使 CREATE TABLE：

```
mysql> CREATE TABLE pet1
    -> SELECT name AS animal,owner AS child
    -> FROM pet;
```

然后，检索生成的表，看看是否打到目的：

```
mysql> SELECT * FROM pet1;
+---------+-------+
| animal  | child |
+---------+-------+
| Fluffy  | Harold |
| Claws   | Gwen  |
| Buffy   | Harold |
| Chirpy  | Gwen  |
| Fang    | Benny |
| Bowser  | Diane |
| Whistler | Gwen  |
| Slim    | Benny |
| Puffball | Diane |
+---------+-------+
```

4.4.1.2 在子句中使用列的别名

你可以在 GROUP BY、ORDER BY 或在 HAVING 部分中使用别名引用列。别名也可以用来为列取一个更好点的名字：

```
mysql> SELECT species,COUNT(*) AS total FROM pet
    -> GROUP BY species HAVING total>1;
+---------+-------+
| species | total |
+---------+-------+
| bird    |     2 |
| cat     |     2 |
| dog     |     3 |
+---------+-------+
```

注意，你的 ANSI SQL 不允许你在一个 WHERE 子句中引用一个别名。这是因为在 WHERE 代码被执行时，列值还可能没有终结。例如下列查询是不合法：

```
SELECT id,COUNT(*) AS total FROM pet WHERE total > 1 GROUP BY species
```

会有下面的错误：

```
ERROR 1054: Unknown column 'total' in 'where clause'
```

WHERE 语句被执行以确定哪些行应该包括 GROUP BY 部分中，而 HAVING 用来决定应该只用结果集合中的哪些行。

4.4.1.3 表的别名

别名不仅可以应用于列，也可以引用于表名，具体方法类似于列的别名，这里不再重复。

列的别名经常用于表自身的连接中。你不必有 2 个不同的表来执行一个联结。如果你想要将一个表的记录与同一个表的其他记录进行比较，联结一个表到自身有时是有用的。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关联于哪个表实例更直观。

4.4.2 取出互不相同的记录

有时候你可能希望取出的数据互不重复，因为重复的数据可能对你没有意义。

解决的办法是使用 DISTINCT 关键字，使用这个关键字保证结果集中不包括重复的记录，也就是说，你取出的记录中，没有重复的行。

例如，我们取出 pet 表中 Benny 所拥有的宠物的记录：

```
mysql> SELECT name,owner,species,sex FROM pet WHERE owner="Benny";
+------+-------+---------+------+
| name | owner | species | sex  |
+------+-------+---------+------+
| Fang | Benny | dog     | m    |
| Slim | Benny | snake   | m    |
+------+-------+---------+------+
```

注意上面的结果，因为我们要使用它。

假定我们指定 DISTINCT 关键字，并返回列 name，species，sex 列：

```
mysql> SELECT DISTINCT name,species,sex FROM pet WHERE owner="Benny";
+------+---------+------+
| name | species | sex  |
+------+---------+------+
| Fang | dog     | m    |
| Slim | snake   | m    |
+------+---------+------+
```

你可以看到有两条结果，这是因为返回的结果集中的行不同，如果我们做以下更改，只返回 owner，sex 列，你可以观察变化：

```
mysql> SELECT DISTINCT owner,sex FROM pet WHERE owner="Benny";
+-------+-----+
| owner | sex |
+-------+-----+
| Benny | m   |
+-------+-----+
```

DISTINCT 关键字的存在，使查询只返回不同的记录行。

如果一个表中，有完全相同的行，你可以使用 DISTINCT，以去除冗余的输出：

```
SELECT DISTINCT * FROM tbl_name
```

4.4.3 NULL 值的问题

NULL 值可能很奇怪直到你习惯于它。概念上，NULL 意味着“没有值”或“未知值”，且它被看作有点与众不同的值。为了测试 NULL，你不能使用算术比较运算符例如=、<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 |
+----------+-----------+----------+----------+
```

很清楚你从这些比较中得到毫无意义的结果。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 |
+-----------+---------------+
```

在 MySQL 中，0 意味着假而 1 意味着真。

NULL 这样特殊的处理是为什么，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 而不是 death != NULL 是必要的：

```
mysql> SELECT * FROM pet WHERE death IS NOT NULL;
+--------+------+--------+-----+------------+------------+
```

```
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1990-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
```

NULL 值的概念是造成 SQL 的新手的混淆的普遍原因，他们经常认为 NULL 是和一个空字符串''的一样的东西。不是这样的！例如，下列语句是完全不同的：

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ("");
```

两个语句把值插入到 phone 列，但是第一个插入一个 NULL 值而第二个插入一个空字符串。第一个的含义可以认为是“电话号码不知道”，而第二个则可意味着“她没有电话”。

在 SQL 中，NULL 值在于任何其他值甚至 NULL 值比较时总是假的（FALSE）。包含 NULL 的一个表达式总是产生一个 NULL 值，除非在包含在表达式中的运算符和函数的文档中指出。在下列例子，所有的列返回 NULL：

```
mysql> SELECT NULL,1+NULL,CONCAT('Invisible',NULL);
+------+--------+-----------------------+
| NULL | 1+NULL | CONCAT('Invisible',NULL) |
+------+--------+-----------------------+
| NULL |   NULL | NULL                  |
+------+--------+-----------------------+
```

如果你想要寻找值是 NULL 的列，你不能使用=NULL 测试。下列语句不返回任何行，因为对任何表达式，expr = NULL 是假的：

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

要想寻找 NULL 值，你必须使用 IS NULL 测试。下例显示如何找出 NULL 电话号码和空的电话号码：

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
mysql> SELECT * FROM my_table WHERE phone = "";
```

在 MySQL 中，就像很多其他的 SQL 服务器一样，你不能索引可以有 NULL 值的列。你必须声明这样的列为 NOT NULL，而且，你不能插入 NULL 到索引的列中。

当使用 ORDER BY 时，首先呈现 NULL 值。如果你用 DESC 以降序排序，NULL 值最后显示。当使用 GROUP BY 时，所有的 NULL 值被认为是相等的。

为了有助于 NULL 的处理，你能使用 IS NULL 和 IS NOT NULL 运算符和 IFNULL() 函数。

对某些列类型，NULL 值被特殊地处理。如果你将 NULL 插入表的第一个 TIMESTAMP 列，则插入当前的日期和时间。如果你将 NULL 插入一个 AUTO_INCREMENT 列，则插入顺序中的下一个数字。

4.4.4 大小写敏感性

1、数据库和表名

在 MySQL 中，数据库和表对应于在那些目录下的目录和文件，因而，内在的操作系统的敏感性决定数据库和表命名的大小写敏感性。这意味着数据库和表名在 Unix 上是区分大小写的，而在 Win32 上忽略大小写。

注意：在 Win32 上，尽管数据库和表名是忽略大小写的，你不应该在同一个查询中使用不同的大小写来引用一个给定的数据库和表。下列查询将不工作，因为它作为 my_table 和作为 MY_TABLE 引用一个表：

```
mysql> SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

2、列名

列名在所有情况下都是忽略大小写的。

3、表的别名

表的别名是区分大小写的。下列查询将不工作，：因为它用 a 和 A 引用别名：

```
mysql> SELECT col_name FROM tbl_name AS a
        WHERE a.col_name = 1 OR A.col_name = 2;
```

4、列的别名

列的别名是忽略大小写的。

5、字符串比较和模式匹配

缺省地，MySQL 搜索是大小写不敏感的(尽管有一些字符集从来不是忽略大小写的，例如捷克语)。这意味着，如果你用 col_name LIKE 'a%'搜寻，你将得到所有以 A 或 a 开始的列值。如果你想要使这个搜索大小写敏感，使用象 INDEX(col_name, "A")=0 检查一个前缀。或如果列值必须确切是"A"，使用 STRCMP(col_name, "A") = 0。

简单的比较操作(>=、>、= 、< 、<=、排序和聚合)是基于每个字符的“排序值”。有同样排序值的字符(象 E，e)被视为相同的字符！

LIKE 比较在每个字符的大写值上进行(“E”=”e”)。

如果你想要一个列总是被当作大小写敏感的方式，声明它为 BINARY。

例如：

```
mysql> SELECT "E"="e","E"=BINARY "e";
+---------+--------------+
| "E"="e" | "E"=BINARY "e" |
+---------+--------------+
|       1 |              0 |
+---------+--------------+
```

4.4.5 检索语句与多个表的连接

SELECT 语句不仅可以从单个表中检索数据，也可以通过连接多个表来检索数据。这里将介绍全连接和左连接的作用。

我们创建两个表作为例子。

```
mysql> CREATE TABLE first
    -> (
    -> id TINYINT,
    -> first_name CHAR(10)
    -> );
```

录入如下数据：

```
+------+----------+
| id   | first_name|
+------+----------+
|    1 | Tom      |
|    2 | Marry    |
|    3 | Jarry    |
+------+----------+
```

```
mysql> CREATE TABLE last
    -> (
    -> id TINYINT,
    -> last_name CHAR(10)
    -> );
```

录入数据

```
+------+----------+
| id   | last_name |
+------+----------+
|    2 | Stone    |
|    3 | White    |
|    4 | Donald   |
+------+----------+
```

4.4.5.1 全连接

全连接：在检索时指定多个表，将每个表用都好分隔，这样每个表的数据行都和其他表的每行交叉产生所有可能的组合，这样就是一个全连接。所有可能的组和数即每个表的行数之和。

那么观察下面的检索的结果：

```
mysql> SELECT * FROM first,last;
+------+-----------+------+----------+
| id   | first_name | id   | last_name |
+------+-----------+------+----------+
|    1 | Tom       |    2 | Stone    |
|    2 | Marry     |    2 | Stone    |
|    3 | Jarry     |    2 | Stone    |
```

```
|    1 | Tom        |    3 | White     |
|    2 | Marry      |    3 | White     |
|    3 | Jarry      |    3 | White     |
|    1 | Tom        |    4 | Donald    |
|    2 | Marry      |    4 | Donald    |
|    3 | Jarry      |    4 | Donald    |
+------+------------+------+-----------+
```

你可以看到输出的结果集中共有 3×3=9 行，这就是全连接的结果。

你也可以这样使用 SQL 语句：

```
mysql> SELCT first.*,last.* FROM first,last;
```

输出结果与上面的例子相同，并无二致。记录集的输出的排序是以 FROM 子句后的表的顺序进行，即先排列位置靠前的表，即使你改变记录集中列的显示顺序，例如下面的例子：

```
mysql> SELECT last.*,first.* FROM first,last;
+------+-----------+------+-----------+
| id   | last_name | id   | first_name |
+------+-----------+------+-----------+
|    2 | Stone     |    1 | Tom        |
|    2 | Stone     |    2 | Marry      |
|    2 | Stone     |    3 | Jarry      |
|    3 | White     |    1 | Tom        |
|    3 | White     |    2 | Marry      |
|    3 | White     |    3 | Jarry      |
|    4 | Donald    |    1 | Tom        |
|    4 | Donald    |    2 | Marry      |
|    4 | Donald    |    3 | Jarry      |
+------+-----------+------+-----------+
```

上面的例子是两个非常小的表的例子，如果是几个非常大的表的全连接，例如，两个行数分别为 1000 的表，这样的连接可以产生非常大的结果集合 1000×1000＝100 万行。而实际上你并不需要这么多行的结果，通常你需要使用一个 WHERE 从句来限制返回的记录集的行数：

```
mysql> SELECT * FROM first,last WHERE first.id= last.id;
+------+------------+------+-----------+
| id   | first_name | id   | last_name |
+------+------------+------+-----------+
|    2 | Marry      |    2 | Stone     |
|    3 | Jarry      |    3 | White     |
```

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

4.4.5.2 左连接

左连接：全连接给出 FROM 子句中所有表都有匹配的行。对于左连接，不仅匹配类似前面的行记录，而且还显示左边的表有而右边的表中无匹配的行。对于这样的行，从右边表选择的列均被显示为 NULL。这样，每一匹配的行都从左边的表被选出，而如果右边表有一个匹配的行，则被选中，如果不匹配，行仍然被选中，不过，其中右边相应的列在结果集中均设为 NULL。即，LEFT JOIN 强制包含左边表的每一行，而不管右边表是否匹配。

语法：SELECT FROM table_reference LEFT JOIN table_reference ON conditional_expr

其中 table_reference 为连接的表，ON 子句后接类似 WHERE 子句的条件。

下面我们详细讲述左连接的使用：

- 首先，返回一个全连接的结果集：

```
mysql> SELECT * FROM first,last;
+-----+----------+-----+----------+
| id  | first_name | id  | last_name |
+-----+----------+-----+----------+
|   1 | Tom        |   2 | Stone     |
|   2 | Marry      |   2 | Stone     |
|   3 | Jarry      |   2 | Stone     |
|   1 | Tom        |   3 | White     |
|   2 | Marry      |   3 | White     |
|   3 | Jarry      |   3 | White     |
|   1 | Tom        |   4 | Donald    |
|   2 | Marry      |   4 | Donald    |
|   3 | Jarry      |   4 | Donald    |
+-----+----------+-----+----------+
```

注意上面的结果，下面的例子要与这个例子对照。

- 我们在给出一个限制条件的查询：

```
mysql> SELECT * FROM first,last WHERE first.id=last.id;
+-----+----------+-----+----------+
| id  | first_name | id  | last_name |
+-----+----------+-----+----------+
|   2 | Marry      |   2 | Stone     |
|   3 | Jarry      |   3 | White     |
+-----+----------+-----+----------+
```

这个结果类似于是从上一个全连接中选择出 first.id>last.id 的行。

现在我们给出一个真正的左连接的例子，你可以仔细观察它的结果，要了解检索的记录顺序：

```
mysql> SELECT * FROM first LEFT JOIN last ON first.id=last.id;
```

```
+------+-----------+------+----------+
| id   | first_name | id   | last_name |
+------+-----------+------+----------+
|    1 | Tom        | NULL | NULL      |
|    2 | Marry      |    2 | Stone     |
|    3 | Jarry      |    3 | White     |
+------+-----------+------+----------+
```

上面的结果，即用左边表的每一行与右边表匹配，如果匹配，则选择到结果集中，如果没有匹配，则结果集中，右边表相应的列置为 NULL。

- 为了进一步理解这一点，我们给出一个有点奇怪的例子：

```
mysql> SELECT * FROM first LEFT JOIN last ON first.id=1;
+------+-----------+------+----------+
| id   | first_name | id   | last_name |
+------+-----------+------+----------+
|    1 | Tom        |    2 | Stone     |
|    1 | Tom        |    3 | White     |
|    1 | Tom        |    4 | Donald    |
|    2 | Marry      | NULL | NULL      |
|    3 | Jarry      | NULL | NULL      |
+------+-----------+------+----------+
```

因为，在结果的最后两行有似乎你不希望的结果。记住，如果只有 ON 子句的条件，那么左边表的每一行都会返回，只是如果没有匹配的右边表（虽然本例没有约束右边表的列），则记录集中显示为 NULL。

前面只是帮助你理解左连接，下面 LEFT JOIN 的一些有用的技巧。LEFT JOIN 最常见的是与 WHERE 子句共同使用。

使用 IS NULL 或者 IS NOT NULL 操作符可以筛选 NULL 或者非 NULL 值的列，这是最常见的技巧。

例如，选出 first.id=last.id 的组合,并且剔除其中没有右表的匹配记录：

```
mysql> SELECT * FROM first LEFT JOIN last ON first.id=last.id
    -> WHERE last.id IS NOT NULL;

+------+-----------+------+----------+
| id   | first_name | id   | last_name |
+------+-----------+------+----------+
|    2 | Marry      |    2 | Stone     |
|    3 | Jarry      |    3 | White     |
+------+-----------+------+----------+
```

你可以看到这样做的例子结果与语句

SELECT * FROM first,last WHERE first.id=last.id
的输出是相同的。
又如，检索 id 值只在左边表出现，而不再右边表出现的记录：

```
mysql> SELECT first.* FROM first LEFT JOIN last ON first.id=last.id
    -> WHERE last.id IS NULL;
+------+------------+
| id   | first_name |
+------+------------+
|    1 | Tom        |
+------+------------+
```

这个语句是不能用功能相同的带 WHERE 子句的全连接代替的。
注意：全连接和左连接的结果集排列顺序是不同的，例如：

```
mysql> SELECT * FROM first,last WHERE first.id!=last.id;
+------+------------+------+-----------+
| id   | first_name | id   | last_name |
+------+------------+------+-----------+
|    1 | Tom        |    2 | Stone     |
|    3 | Jarry      |    2 | Stone     |
|    1 | Tom        |    3 | White     |
|    2 | Marry      |    3 | White     |
|    1 | Tom        |    4 | Donald    |
|    2 | Marry      |    4 | Donald    |
|    3 | Jarry      |    4 | Donald    |
+------+------------+------+-----------+
```

```
mysql> SELECT * FROM first LEFT JOIN last ON  first.id!=last.id;
+------+------------+------+-----------+
| id   | first_name | id   | last_name |
+------+------------+------+-----------+
|    1 | Tom        |    2 | Stone     |
|    1 | Tom        |    3 | White     |
|    1 | Tom        |    4 | Donald    |
|    2 | Marry      |    3 | White     |
|    2 | Marry      |    4 | Donald    |
|    3 | Jarry      |    2 | Stone     |
|    3 | Jarry      |    4 | Donald    |
+------+------------+------+-----------+
```

4.4.6 总结

本节的内容非常繁杂，各小节之间可能没有什么联系，但是本节所述的都是检索数据时很常用的技巧，主要的一些内容如下：

1、 为表和列使用别名
2、 注意 NULL 值在查询中的使用
3、 注意表名、列名、别名和字符串的大小写问题
4、 如何避免取出重复的记录

4.5 索引属性

索引是加速表内容访问的主要手段，特别对涉及多个表的连接的查询更是如此。这是第 8 章数据库优化中的一个重要内容，第 4 章讨论了为什么需要索引，索引如何工作以及怎样利用它们来优化查询。本节中，我们将介绍索引的特点，以及创建和删除索引的语法。

4.5.1 索引的特点

所有的 MySQL 列类型能被索引。在相关的列上的使用索引是改进 SELECT 操作性能的最好方法。

- 一个表最多可有 16 个索引。最大索引长度是 256 个字节，尽管这可以在编译 MySQL 时被改变。
- 对于 CHAR 和 VARCHAR 列，你可以索引列的前缀。这更快并且比索引整个列需要较少的磁盘空间。对于 BLOB 和 TEXT 列，你必须索引列的前缀，你不能索引列的全部。
- MySQL 能在多个列上创建索引。一个索引可以由最多 15 个列组成。（在 CHAR 和 VARCHAR 列上，你也可以使用列的前缀作为一个索引的部分）。

虽然随着 MySQL 的进一步开发创建索引的约束将会越来越少，但现在还是存在一些约束的。下面的表根据索引的特性，给出了 ISAM 表和 MyISAM 表之间的差别：

表 2-1 通道信息特征字对照表

索引的特点	ISAM 表	MyISAM 表
NULL 值	不允许	允许
BLOB 和 TEXT 列	不能索引	只能索引列的前缀
每个表中的索引数	16	32
每个索引中的列数	16	16
最大索引行尺寸	256 字节	500 字节

从此表中可以看到，对于 ISAM 表来说，其索引列必须定义为 NOT NULL，并且不能对 BLOB 和 TEXT 列进行索引。MyISAM 表类型去掉了这些限制，而且减缓了其他

的一些限制。两种表类型的索引特性的差异表明，根据所使用的 MySQL 版本的不同，有可能对某些列不能进行索引。例如，如果使用 3.23 版以前的版本，则不能对包含 NULL 值的列进行索引。

索引有如下的几种情况：

- INDEX 索引：通常意义的索引，某些情况下 KEY 是它的一个同义词。索引的列可以包括重复的值。
- UNIQUE 索引：唯一索引，保证了列不包含重复的值，对于多列唯一索引，它保证值的组合不重复。
- PRIMARY KEY 索引：也 UNIQUE 索引非常类似。事实上，PRIMARY KEY 索引仅是一个具有 PRIMARY 名称的 UNIQUE 索引。这表示一个表只能包含一个 PRIMARY KEY。

4.5.2 用 Alter Table 语句创建与删除索引

为了给现有的表增加一个索引，可使用 ALTER TABLE 或 CREATE INDEX 语句。ALTER TABLE 最常用，因为可用它来创建普通索引、UNIQUE 索引或 PRIMARY KEY 索引，如：

```
ALTER TABLE tbl_name ADD INDEX index_name  (column_list)
ALTER TABLE tbl_name ADD UNIQUE index_name  (column_list)
ALTER TABLE tbl_name ADD PRIMARY KEY index_name  (column_list)
```

其中 tbl_name 是要增加索引的表名，而 column_list 指出对哪些列进行索引。一个(col1,col2,..)形式的列表创造一个多列索引。索引值有给定列的值串联而成。如果索引由不止一列组成，各列名之间用逗号分隔。索引名 index_name 是可选的，因此可以不写它，MySQL 将根据第一个索引列赋给它一个名称。ALTER TABLE 允许在单个语句中指定多个表的更改，因此可以在同时创建多个索引。

同样，也可以用 ALTER TABLE 语句产出列的索引：

```
ALTER TABLE tbl_name DROP INDEX index_name
ALTER TABLE tbl_name DROP PRIMARY KEY
```

注意上面第一条语句可以用来删除各种类型的索引，而第三条语句只在删除 PRIMARY KEY 索引时使用；在此情形中，不需要索引名，因为一个表只可能具有一个这样的索引。如果没有明确地创建作为 PRIMARY KEY 的索引，但该表具有一个或多个 UNIQUE 索引，则 MySQL 将删除这些 UNIQUE 索引中的第一个。

如果从表中删除了列，则索引可能会受到影响。如果所删除的列为索引的组成部分，则该列也会从索引中删除。如果组成索引的所有列都被删除，则整个索引将被删除。

例如，对于上面所使用的 student 为例，你可能想为之创建这样的索引，以加速表的检索速度：

```
mysql> ALTER TABLE student
    -> ADD PRIMARY KEY(id),
    -> ADD INDEX mark(english,Chinese,history);
```

这个例子，既包括 PRIMARY 索引，也包括多列索引。记住，使用 PRIMARY 索引的列，必须是一个具有 NOT NULL 属性的列，如果你愿意产看创建的索引的情况，可以使用 SHOW INDEX 语句：

```
mysql> SHOW INDEX FROM student;
```

其结果为：

```
+---------+-----------+---------+-------------+------------+-
| Table   | Non_unique | Key_name | Seq_in_index | Column_name |
+---------+-----------+---------+-------------+------------+-
| student |          0 | PRIMARY  |            1 | id         |
| student |          1 | mark     |            1 | english    |
| student |          1 | mark     |            2 | chinese    |
| student |          1 | mark     |            3 | history    |
+---------+-----------+---------+-------------+------------+-
```

由于列数太多，上表并没有包括所有的输出，读者可以试着自己查看。

再使用 ALTER TABLE 语句删除索引，删除索引需要知道索引的名字，你可以通过 SHOW INDEX 语句得到：

```
mysql> ALTER TABLE student DROP PRIMARY KEY,
    -> DROP INDEX mark;
```

再产看表中的索引，其语句和输出为：

```
mysql> SHOW INDEX FROM student;
Empty set (0.01 sec)
```

4.5.3 用 CREATE\DROP INDEX 创建索引

还可以用 CREATE INDEX 语句来创建索引.CREATE INDEX 是在 MySQL 3.23 版中引入的，但如果使用 3.23 版以前的版本，可利用 ALTER TABLE 语句创建索引（MySQL 通常在内部将 CREATE INDEX 映射到 ALTER TABLE）。该语句创建索引的语法如下：

```
CREATE UNIQUE INDEX index_name ON tbl_name (column_list)
CREATE INDEX index_name ON tbl_name (column_list)
```

tbl_name、index_name 和 column_list 具有与 ALTER TABLE 语句中相同的含义。这里索引名不可选。很明显，CREATE INDEX 可对表增加普通索引或 UNIQUE 索引，不能用 CREATE INDEX 语句创建 PRIMARY KEY 索引。

可利用 DROP INDEX 语句来删除索引。类似于 CREATE INDEX 语句，DROP INDEX 通常在内部作为一条 ALTER TABLE 语句处理，并且 DROP INDEX 是在 MySQL 3.22 中引入的。

删除索引语句的语法如下：

```
DROP INDEX index_name ON tbl_name
```

还是上一节的例子，由于 CREATE INDEX 不能创建 PRIMARY 索引，所以这里我们值创建一个多列索引：

```
mysql> CREATE INDEX mark ON student(english,chinese,history);
```

同样的检查 student 表，可知：

```
mysql> SHOW INDEX FROM student;
+---------+----------+---------+------------+------------+
| Table   | Non_unique | Key_name | Seq_in_index | Column_name |
+---------+----------+---------+------------+------------+
| student |          1 | mark    |            1 | english    |
| student |          1 | mark    |            2 | chinese    |
| student |          1 | mark    |            3 | history    |
+---------+----------+---------+------------+------------+
```

然后使用下面的语句删除索引：

```
mysql> DROP INDEX mark ON student;
```

4.5.4 在创建表时指定索引

要想在发布 CREATE TABLE 语句时为新表创建索引，所使用的语法类似于 ALTER TABLE 语句的语法，但是应该在您定义表列的语句部分指定索引创建子句，如下所示：

```
CREATE TABLE tbl_name
(
    …
    INDEX index_name (column_list),
    KEY index_name (column_list),
    UNIQUE index_name (column_list),
    PRIMARY KEY index_name (column_list),
    …
)
```

与 ALTER TABLE 一样，索引名对于 INDEX 和 UNIQUE 都是可选的，如果未给出，MySQL 将为其选一个。另外，这里 KEY 时 INDEX 的一个别名，具有相同的意义。

有一种特殊情形：可在列定义之后增加 PRIMARY KEY 创建一个单列的 PRIMARY KEY 索引，如下所示：

```
CREATE TABLE tbl_name
(
  i INT NOT NULL PRIMARY KEY
)
```

该语句等价于以下的语句:

```
CREATE TABLE tbl_name
(
  i INT NOT NULL,
  PRIMARY KEY (i)
```

```
)
```

前面所有表创建样例都对索引列指定了 NOT NULL。如果是 ISAM 表，这是必须的，因为不能对可能包含 NULL 值的列进行索引。如果是 MyISAM 表，索引列可以为 NULL，只要该索引不是 PRIMARY KEY 索引即可。

在 CREATE TBALE 语句中可以某个串列的前缀进行索引（列值的最左边 n 个字符）。

如果对某个串列的前缀进行索引，应用 column_list 说明符表示该列的语法为 col_name(n) 而不用 col_name。例如，下面第一条语句创建了一个具有两个 CHAR 列的表和一个由这两列组成的索引。第二条语句类似，但只对每个列的前缀进行索引：

```
CREATE TABLE tbl_name
(
    name CHAR(30),
    address CHAR(60),
    INDEX (name,address)
)
CREATE TABLE tbl_name
(
    name CHAR(30),
    address CHAR(60),
    INDEX (name(10),address(20))
)
```

你可以检查所创建表的索引：

```
mysql> SHOW INDEX FROM tbl_name;
+---------+----------+---------+------------+-----------+-
| Table   | Non_unique | Key_name | Seq_in_index | Column_name |
+---------+----------+---------+------------+-----------+-
| tbl_name |          1 | name    |            1 | name        |
| tbl_name |          1 | name    |            2 | address     |
+---------+----------+---------+------------+-----------+-
```

在某些情况下，可能会发现必须对列的前缀进行索引。例如，索引行的长度有一个最大上限，因此，如果索引列的长度超过了这个上限，那么就可能需要利用前缀进行索引。在 MyISAM 表索引中，对 BLOB 或 TEXT 列也需要前缀索引。

对一个列的前缀进行索引限制了以后对该列的更改；不能在不删除该索引并使用较短前缀的情况下，将该列缩短为一个长度小于索引所用前缀的长度的列。

4.5.5 总结

本节对索引的类型，已经如何创建索引做了介绍，其中涉及三个比较重要的 SQL 语句——ALTER TABLE、CREATE/DROP INDEX 和 CREATE TABLE，注意它们的用法。

索引最重要的功能是，通过使用索引加速表的检索，有关这方面的知识，将在第十章数据库优化中介绍。

思考题

1、 建立一个如下所述的表：

data：FLOAT 列，使用随机函数填充数据

birth：DATETIME 列，填充当前时间。

然后，请录入几条数据。最后计算 data 列的平均值、总和、极值，并且按照 data 列降序排序检索值。

2、 分别使用标准 SQL 模式和扩展正规表达式模式匹配，匹配上面创建的表，假设你创建表的当前日期为 2001-01-01，用模式匹配检索出 birth 列包含该日期的值。（实际上，上面的表中记录都是同一日期录入的，因此实际将返回全部记录。）

3、 为前几章使用的数据表创建索引：

student：为 id 段创建一个 PRIMARY 索引，为 english、chinese 和 history 创建一个多列索引。

pet：为 name 和 owner 段创建一个多类索引。

4、删除为 pet 表创建的索引。

第5章

数据库的备份与恢复

本章要点：

- 数据库目录的内容
- 如何备份和恢复数据
- 如何使用日志
- 如何使用内建复制

有多种可能会导致数据表的丢失或者服务器的崩溃，一个简单的 DROP TABLE 或者 DROP DATABASE 的语句，就会让你的数据表化为乌有。更危险的是 DELETE * FROM tbl_name，可以轻易的清空你的数据表，而这样的错误是很容易的发生的。

因此，拥有能够恢复的数据对于一个数据库系统来说是非常重要的。一般的说，MySQL 有三种保证数据安全的方法：

- 常规日志和更新日志

通过保存执行的查询供你必要时恢复

- 数据库备份

通过导出数据或者表文件的拷贝来保护数据

- 数据库复制

MySQL 内部复制功能是建立在两个或两个以上服务器之间，通过设定它们之间的主-从关系来实现的。其中一个作为主服务器，其它的作为从服务器。这个复制功能是在 3.23.15 版以后才有的

5.1 数据库目录

数据库目录是 MySQL 数据库服务器存放数据文件的地方，不仅包括有关表的文件，还包括数据文件和 MySQL 的服务器选项文件。不同的分发，数据库目录的缺省位置是不同的。

5.1.1 数据目录的位置

- 缺省的数据库位置

  缺省数据库的位置编译在服务器中。

  1、 如果您是在一个源程序分发包中安装 MySQL，典型的缺省位置可能是 /usr/local/var；

  2、 如果在二进制分发包中安装 MySQL，则为 /usr/local/mysql/ data；

  3、 在 RPM 文件中安装，为 /var/lib/mysql。

  4、 对于 windwos 平台上的分发，其位置时 BASEDIR\data

- 数据目录的位置可以在启动服务器时通过--datadir = / path / to / dir 明确地指定。如果您想将数据目录放置在其他地方而非缺省的位置，则这个选项是有用的。

- 了解数据库目录的位置

作为一名 MySQL 管理员，您应该知道数据目录在哪里。如果运行多个服务器，那么您应该掌握所有数据目录的位置。但是，如果不知道目录的位置（或许您正在代替前一位管理员，而他留下的记录很糟糕），有几种方法可以用来查找它：

1、 可使用 mysqladmin 变量直接从服务器中得到数据目录路径名。在 UNIX 中，输出结果类似于如下所示：

```
$mysqladmin variables
+---------------------+-----------------------
| Variable_name       | Value
+---------------------+-----------------------
| ansi_mode           | OFF
| back_log            | 50
| basedir             | /var/local
| connect_timeout     | 5
| concurrent_insert   | ON
| datadir             | /usr/local/var
```

该输出结果指明了服务器主机中数据目录的位置 /usr/local/var。

在 Windows 中，输出结果类似于如下所示：

```
c:\mysql\bin>mysqladmin variables
+---------------------+-----------------------
| Variable_name       | Value
```

```
+---------------------+----------------------
| ansi_mode           | OFF
| back_log            | 50
| basedir             | c:\mysql\
| connect_timeout     | 5
| concurrent_insert   | ON
| datadir             | c:\mysql\data\
```

如果正在运行多个服务器，它们将监听不同的 TCP/IP 端口号和套接字。可以通过提供合适的--port 或 --socket 选项连接到每个服务器监听的端口和套接字上：

$mysqladmin –port=port=port_num variables

$mysqladmin –socket=/path/to/socket variables

mysqladmin 命令可在您连接服务器的任何一台主机上运行。如果需要连接到远程主机上的服务器，则使用 --host = host_name 选项：

$mysqladmin –host=host_name varibles

2、在 Unix 平台上，可使用 ps 来查看任何当前执行 mysql 进程的命令行。试一试下列的命令（根据您的系统所支持的 ps 版本）并查找显示在输出结果中的这些命令的 --datadir：

$ps au | grep mysqld

如果系统运行多个服务器（因为一次发现了多个数据目录位置），则 ps 命令将会特别有用。它的缺点是：ps 必须运行在服务器的主机上，并且除非 --datadir 选项在 mysqld 命令行中明确指定，否则将产生无用的信息。

3、 如果 MySQL 从源程序分发包中安装，可以检查其配置信息以确定数据目录的位置。例如，在最高级的 Makefile 中该位置是可用的。但是，要小心：位置是 Makefile 中的变量 localstatedir 的值，而不是 datadir 的值。同样，如果分发包定位在 NFS 装配文件系统中，并且是用于为几个主机建立 MySQL 的，则配置信息反映最近建立分发包的主机。它可能不显示您感兴趣的主机的数据目录。

4、 如果前面的任何方法都不成功，可使用 find 搜索数据库文件。下列命令将搜索 .frm（描述）文件，它是 MySQL 安装程序的组成部分：

$find / -name “*.frm” –print

在 windows 平台上的搜索非常简单，本节就不给出例子了。

在本章的这些例子中，笔者将 MySQL 数据目录的位置表示为 DATADIR。您可以将其解释成为您自己的机器中的数据目录的位置。

5.1.2 数据库的表示法

由 MySQL 管理的每个数据库都有自己的数据库目录，它们是数据目录的子目录，与所表示的数据库有相同的名称。例如，数据库 my_db 对应于数据库目录 DATADIR/my_db。

这个表示法使得几个数据库级的语句的实现是非常容易的。CREATE DATABASE

db_name 使用只允许对 MySQL 服务器用户（服务器运行的 UNIX 用户）进行访问的所有权和方式，并在数据目录中创建一个空目录 db_name。这等价于以服务器主机中的服务器用户的身份通过执行下列命令手工创建数据库：

$ mkdir DATADIR/db_name 创建数据库目录

$ chmod 700 DATADIR/db_name 使它仅对 MySQL 服务器用户可访问

通过空目录表示新数据库的方法与其他数据库系统完全不同，那些数据库系统甚至要为“空”数据库创建许多控制文件或系统文件。

DROP DATABASE 语句也很容易实现。DROP DATABASE db_name 删除数据目录中的 db_name 目录以及其中的所有表文件。这个语句类似于下列命令：

$rm -rf DATADIR/db_name

其区别是，服务器只删除带有表的扩展名的文件。如果已经在该数据库目录中创建了其他的文件，服务器将使它们保持完整，并且不删除该目录本身。

SHOW DATABASE 只不过是对应位于数据目录中的子目录名称的一个列表。有些数据库系统需要保留一个列出所有需要维护的数据库的主表，但是，在 MySQL 中没有这样的结构。由于数据目录结构的简单性，数据库的列表是隐含在该数据目录的内容中的，像主表这样的表可能会引起不必要的开销。

5.1.3 数据库表的表示法

数据库中的每个表在数据库目录中都作为三个文件存在：一个格式（描述）文件、一个数据文件和一个索引文件。每个文件的基名是该表名，扩展名指明该文件的类型。扩展名如表 5-1 所示。数据和索引文件的扩展名指明该表是否使用较老的 ISAM 索引或较新的 MyISAM 索引。

表 5-1 MySQL 文件类型

文件类型	文件扩展名	文件内容
格式文件	.frm	描述表的结构（列、列类型、索引，等等）
数据文件	.ISD(ISAM) 或.MYD(MyISAM)	包含表的数据—即它的行
索引文件	ISD(ISAM) 或.MYI(MyISAM)	包含数据文件中任何索引的索引树。无论该表有无索引，索引文件都存在

当发布定义一个表结构的 CREATE TABLE tbl_name 语句时，服务器创建 tbl_name.frm 文件，它包含该结构的内部编码。该语句还创建空的数据文件和索引文件，这些文件的初始信息表明没有记录和索引（如果 CREATE TABLE 语句包含索引说明，则该索引文件将反映这些索引）。描述表的文件的所有权和方式被设置为只允许对 MySQL 服务器用户的访问。

当发布 ALTER TABLE 语句时，服务器对 tbl_name.frm 重新编码并修改数据文件和

索引文件的内容以反映由该语句表明的结构变化。对于 CREATE 和 DROP INDEX 也是如此，因为服务器认为它们等价于 ALTER TABLE 语句。DROP TABLE 删除代表该表的三个文件。

尽管可以通过删除数据库目录中的对应某个表的三个文件来删除该表，但不能手工创建或更改表。例如，如果 my_db 是当前的数据库，DROP TABLE my_tbl 大致等价于下列命令：

来自于 SHOW TABLES my_db 的输出结果正是 my_db 数据库目录中 .frm 文件基名的一个列表。某些数据库系统维护一个列出了数据库中的所有表的登记。但 MySQL 不这样做，因为没有必要，这个“登记”隐含在了数据目录的结构中。

5.1.4 MySQL 的状态文件

除数据库目录外，MySQL 数据目录还包含许多状态文件。表 10-3 概括介绍了这些文件。大多数状态文件的缺省名称从服务器主机名字中生成，在此表中表示为 HOSTNAME。

表 5-2 MySQL 状态文件

文件类型	缺省名	文件内容
进程 ID	HOSTNAME.pid	服务器进程 ID
错误日志	HOSTNAME.err	启动和关闭事件和错误状态
常规日志	HOSTNAME.log	连接/断开事件和查询信息
更新日志	HOSTNAME.nnn	修改表的内容或结构的所有查询的文本

服务器在启动时将它的进程 ID（PID ）写入 PID 文件，并在关闭时删除该文件。PID 文件是一种方法，用这种方法，其他的进程可以找到该服务器。例如，如果您在系统关闭时运行 mysql.server 脚本来关闭 MySQL 服务器，则该脚本将检查 PID 文件以确定它需要哪个进程来发送一个终止信号。

错误日志由 safe_mysqld 产生，作为服务器标准错误输出结果的重定向，它包含服务器写入 stderr 的所有消息。这意味着仅当通过调用 safe_mysqld 启动服务器时，错误日志才存在（总之，这是启动服务器的首选方法，因为，如果由于一个错误使错误日志存在，则 safe_mysqld 将重新启动服务器）。

常规日志和更新日志是可选的，可以用 --log 和 --log-update 服务器选项开启需要的日志类型。

常规进程提供有关服务器运作的常规信息：谁从哪里进行了连接，以及他们发布了什么查询。更新日志也提供查询信息，但仅仅是修改过的数据库内容的查询信息。更新日志的内容是一些 SQL 语句，这些语句可以通过将它们输入到 mysql 客户机程序来运行。如果出现崩溃且必须转到备份文件时，更新日志将是有用的，因为您能够通过将更新日志输入到服务器来重复这些自崩溃以来所完成的更新操作。这将使得数据库恢复到崩溃发生时所处的状态上。

下面是一个实例，它是作为一个短客户机会话的结果出现在常规日志中的信息中的，这个会话在 test 数据库中从 mytest.pet 复制一个表，并插入一行到该表中，然后删除该表：

```
010206 23:08:21          3 Connect     root@localhost on
010206 23:09:07          3 Init DB     test
010206 23:09:36          3 Query       create table mytest select * from pet
010206 23:09:43          3 Query       create table mytest select * from mytest.pet
010206 23:11:34          3 Query       insert into mytest set
name='tom',owner='jerry',species='cat',sex='f',birth='2000-01-01'
010206 23:11:49          3 Query       drop table mytest
010206 23:14:05          3 Quit
```

 注意第二行是一个错误的语句，但是也被记录下来。

常规日志包含日期和时间、服务器线程 ID、事件类型以及特定事件信息的列。

同一个会话出现在如下的更新日志中：

```
use test;
create table mytest select * from mytest.pet;
insert into mytest set name='tom',owner='jerry',species='cat',sex='f',birth='2000-01-01';
drop table mytest;
```

 更新纪录中没有记录错误的语句，因此对于恢复被破坏的数据库内容非常有意义。

对于更新日志，日志的扩展格式是可用的，即使是用 --log - long - format 选项。扩展的日志提供有关谁何时发布查询的信息。当然，这将使用更多的磁盘空间，但是，如果您不将更新日志的内容与常规日志中的连接事件相联系就想知道谁正在做什么的话，扩展日志或许是可用的。

确保日志文件的安全且不被用户任意读取是个好注意。常规日志和更新日志都包含有诸如口令这样的敏感信息，这是因为它们包含了查询的文本。下面是您不想让任何人都能读取的日志项，因为它显示了 root 用户的口令：

```
010206 23:30:02                4 Query       update mysql.user set
password=password("peking77.") where User="root"
```

有关检查可设置数据目录许可权的信息，请参阅第 7 章。数据目录安全的简短指令由下列命令组成：

```
$ chmod 700 DATADIR
```

以拥有该数据目录的 UNIX 用户身份来运行此命令。还要确保服务器以该用户身份运行，否则此命令不仅将其他用户排斥在该数据目录之外（您想要的），还将阻止服务器访问您的数据库（您不要的）。

状态文件出现在数据目录的最高级，就像数据库目录一样，因此您可能会想到那些文件的名字是否会相互混淆或者被误认为是数据库名（例如，当服务器正在执行 SHOW

DATABASE 语句时)。答案是：不会的。状态和日志信息存储在文件中，而数据库是目录，因此可执行程序可以将它们与一个简单的 stat() 调用相区别（是服务器告诉它们怎样区分的)。如果您正在监视数据目录，则可以通过使用 ls -l 将状态文件从数据库目录中区分开来，并且检查该模式信息的第一个字符以查看它是‘-’还是‘d’：

您还可以通过查看名字而简单地告之：所有状态文件名都包含一个句点，但是数据库目录名没有句点（句点不是数据库名的合法字符)。

5.1.5 总结

通过本节，读者可以对 MySQL 的数据保存方式有了一定的认识。本节中较为重要的内容有：

1、 数据库目录的位置

2、 MySQL 是如何表示数据表的

3、 MySQL 的状态文件的种类和作用

了解 MySQL 如何保存数据，以及状态文件的作用，对于备份数据库是有重要意义的，根据 MySQL 数据库目录的特点，直接拷贝就是备份数据的重要方法之一。

5.2 重定位数据库目录的内容

上一节讨论了在其缺省配置中的数据目录的结构。所有数据库和状态文件都包含在其中。但是，在确定数据目录内容的布局中管理员有某些职责。本节讨论为什么要移动数据目录的各个部分（甚至是字典本身)、可以移动什么，以及怎样进行这些移动。

MySQL 允许您重定位其中的数据目录或元素。这样做有几个原因：

- 可以用比缺省定位的文件系统更大的容量在文件系统中放置数据目录。
- 如果数据目录在繁忙的磁盘上，可以将其放置到较少使用的驱动器上，以平衡物理设备之间的磁盘活动。为了类似的原因，可以将数据库和日志文件放在不同的驱动器上，或在驱动器之间对数据库进行再分布。
- 您可以运行多个服务器，并且每个服务器都有属于自己的数据目录。这是一种解决总进程文件描述符限制问题的方法，尤其是当不能重新配置系统的核心以得到更高的限制值时。
- 某些系统将 PID 文件保存在诸如 /var/run 的目录中。为了系统运作的一致性，您可以将 MySQL PID 文件也放在那里。

5.2.3 重定位方法

有两种对数据目录重定位的方法：

- 可以在命令行或在一个选项文件的 [mysqld] 组上，在服务器启动时间指定一个选项。
- 可以移动要重定位的内容，然后在原始的位置中做一个指向新位置的 symlink（symbolic link，符号链接)。

两种方法的任何一种都不能为您进行全部的重定位工作。表 5-3 综合了可重定位的内容以及可用于重定位的方法。如果您使用一个选项文件，可以指定在全局选项文件

/etc/my.cnf（Windows 中的 c:\my.cnf）中的选项。当前的 Windows 版本还访问系统目录（c:\windows 或 c:\WINNT）。

您还可以使用缺省数据目录的选项文件 my.cnf（该目录编译在服务器中）。笔者不建议使用此文件。如果要重定位数据目录本身，必须保持缺省数据目录的完整性，以便在数据目录中放置一个选项文件，该文件将说明服务器应该在哪里找到“真正”的数据目录！真乱。如果想要用一个选项文件来指定服务器的选项，则最好使用 /etc/my.cnf。

表 5-3 重定位方法概括

重定位的实体	可使用的重定位方法
全数据目录	启动选项、选项文件或 symlink
PID 文件	启动选项、选项文件
单个的数据库目录	Symlink
常规日志文件	启动选项、选项文件
单个的数据库表	Symlink
更新日志文件	启动选项、选项文件

5.2.1 重定位数据目录

现在说明重定位数据库目录的方法与步骤。

1、 关闭服务器，例如：

$ mysqladmin –u root -p shutdown

2、 将数据库目录移动到新的位置

3、 然后删除原来的数据库

4、 如果用 symlink 方法（Unix 平台上），例如：

$ ln –s NEWDIR DATADIR

DATADIR 是标准的数据库目录的位置，也是原来的数据库目录路径。

如果用起动选项的方法，可以这样启动：

$safe_mysqld --basedir=/path/to/dir/ & (Unix)

$mysqld --basedir=x:\datadir\ (Windows)

如果用选项文件的方法：

在/etc/my.cnf（Unix）或者 c:\my.cnf(Windows)中加入：

[mysqld]

datadir=/path/to/dir/

5、重新启动数据库

5.2.2 重定位数据库

重定位数据库只能通过 symlink 方法，因此 windows 平台无法重定位数据库。为了重定位数据库，应关闭数据库，移动数据库的目录。删除原来的数据库目录，用指向新位置

的 symlink 来代替她，然后启动服务器。

下面用一个实际的例子——备份 bigdb——说明这个过程：

```
$mysqladmin –u root –p
$cd DATADIR
$tar cf bigdb|(cd /var/db;tar xf)
$mv bigdb bigdb.old
$ln –s var/db/bigdb .
$safe_mysqld &
```

应该以数据库目录所有者的身份执行这些命令。服务器工作正常之后，可以删除备份目录 bigdb.old:

```
$rm –rf bigdb.old
```

5.2.3 重定位数据库表

对单个数据库表重新定位并不是特别有用，有时反而会造成很大的麻烦。可以通过将表的文件移动到另一个位置不再该书据库目录中创建指向这些文件的 symlink 来进行。方法类似于重定位数据库的方法。

但是，如果曾经发布过ALTER TABLE 或 OPTIMIZE TABLE 语句，则所做的这些重定位工作将被取消。其原因就不介绍了。

5.2.4 重定位状态文件

状态文件中 PID 文件、常规日志褐更新日志，可以用起动选项或者选项文件的方法重新定位。错误日志由 safe_mysqld 创建且不能够重新定位，除非编辑 safe_mysqld 脚本。

为了在另一个位置写状态文件，因关闭服务器，然后用制定新状态文件位置的恰当选项启动他。

启动选项：

```
--pid-file=pidfile PID 文件
--log=logfile  常规日志
--log-update=updatefile  更新日志
选项文件
[mysqld]
pid-file=pidfile
log=logfile
log-update=updatefile
```

状态文件的命名规则：

1、 如果以绝对路径指定一个状态文件的名称，则用该路径创建改文件。
例如，你指定—pid-file=/var/run/mysqld.pid,则该 PID 文件的就是/var/run/mysqld.pid

2、 如果你只给出文件名，则文件在数据库目录下创建。
例如，你指定—pid-file=mysqld.pid 则该 PID 文件为 DATADIR/mysqld.pid

3、 如果没有给出文件，则状态文件使用缺省的名字。
这样做将告诉服务器启用状态文件，这对 PID 文件没有意义，因为服务器总是使用它。例如：
$safe_mysqld --log --log-update &
或者，使用选项文件：
[mysqld]
log
log-update

对于常规日志，将生成 DATADIR/hostname.log 文件，hostname 是运行服务器的主机名。对于更新日志，生成 DATADIR/hostname.nnn 的顺序文件。

4、 对于更新日志，如果指定一个没有扩展名的更新日志，则 MySQL 在打开该更新日志是将生成数据的名字。这些名字用.nnn 扩展名创建，这里的.nnn 是违背已有的更新日志文件使用过的第一个号码（如，hostname.001、hostname.002 等等）。可以通过指定明确的扩展名来忽略顺序名字的生成，然后服务器将仅使用您指定的名字。

5.2.5 总结

在本节中你将学到如何重新定位数据库目录，以及状态文件等。你可能需要重定位的内容有：

1、 数据库目录
2、 数据库
3、 状态文件，包括常规日志，更新日志等

重定位的方法有：

1、 符号链接
2、 指定命令行参数
3、 使用选项文件

5.3 备份和恢复数据表的方法

备份是最简单的保护数据的方法，本节将介绍多种备份方法。为了得到一个一致的备份，在相关的表上做一个 LOCK TABLES，你只需一个读锁定，当你在数据库目录中做文件的一个拷贝时，这允许其他线程继续查询该表；当你恢复数据时，需要一个写锁定，以避免冲突。

5.3.1 使用 SQL 语句备份和恢复

你可以使用 SELECT INTO OUTFILE 语句备份数据，并用 LOAD DATA INFILE 语句恢复数据。这种方法只能导出数据的内容，不包括表的结构，如果表的结构文件损坏，你必须要先恢复原来的表的结构。

语法：

```
SELECT * INTO {OUTFILE | DUMPFILE} 'file_name' FROM tbl_name
```

LOAD DATA [LOW_PRIORITY] [LOCAL] INFILE 'file_name.txt' [REPLACE | IGNORE]

INTO TABLE tbl_name

SELECT ... INTO OUTFILE 'file_name'格式的 SELECT 语句将选择的行写入一个文件。文件在服务器主机上被创建，并且不能是已经存在的(不管别的，这可阻止数据库表和文件例如“/etc/passwd”被破坏)。SELECT ... INTO OUTFILE 是 LOAD DATA INFILE 逆操作。

LOAD DATA INFILE 语句从一个文本文件中以很高的速度读入一个表中。如果指定 LOCAL 关键词，从客户主机读文件。如果 LOCAL 没指定，文件必须位于服务器上。(LOCAL 在 MySQL3.22.6 或以后版本中可用。)

为了安全原因，当读取位于服务器上的文本文件时，文件必须处于数据库目录或可被所有人读取。另外，为了对服务器上文件使用 LOAD DATA INFILE，在服务器主机上你必须有 file 的权限。使用这种 SELECT INTO OUTFILE 语句，在服务器主机上你必须有 FILE 权限。

为了避免重复记录，在表中你需要一个 PRIMARY KEY 或 UNIQUE 索引。当在唯一索引值上一个新记录与一个老记录重复时，REPLACE 关键词使得老记录用一个新记录替代。如果你指定 IGNORE，跳过有唯一索引的现有行的重复行的输入。如果你不指定任何一个选项，当找到重复索引值时，出现一个错误，并且文本文件的余下部分被忽略时。

如果你指定关键词 LOW_PRIORITY，LOAD DATA 语句的执行被推迟到没有其他客户读取表后。

使用 LOCAL 将比让服务器直接存取文件慢些，因为文件的内容必须从客户主机传送到服务器主机。在另一方面，你不需要 file 权限装载本地文件。如果你使用 LOCAL 关键词从一个本地文件装载数据，服务器没有办法在操作的当中停止文件的传输，因此缺省的行为好像 IGNORE 被指定一样。

当在服务器主机上寻找文件时，服务器使用下列规则：

- 如果给出一个绝对路径名，服务器使用该路径名。
- 如果给出一个有一个或多个前置部件的相对路径名，服务器相对服务器的数据目录搜索文件。
- 如果给出一个没有前置部件的一个文件名，服务器在当前数据库的数据库目录寻找文件。

假定表 tbl_name 具有一个 PRIMARY KEY 或 UNIQUE 索引，备份一个数据表的过程如下：

1、锁定数据表，避免在备份过程中，表被更新

mysql>LOCK TABLES READ tbl_name;

关于表的锁定的详细信息，将在下一章介绍。

2、导出数据

mysql>SELECT * INTO OUTFILE ‘tbl_name.bak’ FROM tbl_name;

3、解锁表

mysql>UNLOCK TABLES;

相应的恢复备份的数据的过程如下：

1、为表增加一个写锁定：

mysql>LOCK TABLES tbl_name WRITE;

2、恢复数据

mysql>LOAD DATA INFILE ‘tbl_name.bak’

->REPLACE INTO TABLE tbl_name;

如果，你指定一个 LOW_PRIORITY 关键字，就不必如上要对表锁定，因为数据的导入将被推迟到没有客户读表为止：

mysql>LOAD DATA LOW_PRIORITY INFILE ‘tbl_name’

->REPLACE INTO TABLE tbl_name;

3、解锁表

mysql->UNLOCAK TABLES;

5.3.2 使用 mysqlimport 恢复数据

如果你仅仅恢复数据，那么完全没有必要在客户机中执行 SQL 语句，因为你可以简单的使用 mysqlimport 程序，它完全是与 LOAD DATA 语句对应的，由发送一个 LOAD DATA INFILE 命令到服务器来运作。执行命令 mysqlimport --help，仔细查看输出，你可以从这里得到帮助。

shell> mysqlimport [options] db_name filename ...

对于在命令行上命名的每个文本文件，mysqlimport 剥去文件名的扩展名并且使用它决定哪个表导入文件的内容。例如，名为“patient.txt”、“patient.text”和“patient”将全部被导入名为 patient 的一个表中。

常用的选项为：

-C, --compress 如果客户和服务器均支持压缩，压缩两者之间的所有信息。

-d, --delete 在导入文本文件前倒空表格。

l, --lock-tables 在处理任何文本文件前为写入所定所有的表。这保证所有的表在服务器上被同步。

--low-priority，--local，--replace，--ignore 分别对应 LOAD DATA 语句的 LOW_PRIORITY，LOCAL，REPLACE，IGNORE 关键字。

例如恢复数据库 db1 中表 tbl1 的数据，保存数据的文件为 tbl1.bak，假定你在服务器主机上：

shell>mysqlimport --lock-tables --replace db1 tbl1.bak

这样在恢复数据之前现对表锁定，也可以利用--low-priority 选项：

shell>mysqlimport --low-priority --replace db1 tbl1.bak

如果你为远程的服务器恢复数据，还可以这样：

shell>mysqlimport -C --lock-tables --replace db1 tbl1.bak

当然，解压缩要消耗 CPU 时间。

象其它客户机一样，你可能需要提供-u，-p 选项以通过身分验证，也可以在选项文件 my.cnf 中存储这些参数，具体方法和其它客户机一样，这里就不详述了。

5.3.3 使用 mysqldump 备份数据

同 mysqlimport 一样，也存在一个工具 mysqldump 备份数据，但是它比 SQL 语句多做的工作是可以在导出的文件中包括 SQL 语句，因此可以备份数据库表的结构，而且可以备份一个数据库，甚至整个数据库系统。

mysqldump [OPTIONS] database [tables]

mysqldump [OPTIONS] --databases [OPTIONS] DB1 [DB2 DB3...]

mysqldump [OPTIONS] --all-databases [OPTIONS]

如果你不给定任何表，整个数据库将被倾倒。

通过执行 mysqldump --help，你能得到你 mysqldump 的版本支持的选项表。

1、备份数据库的方法

例如，假定你在服务器主机上备份数据库 db_name

shell> mydqldump db_name

当然，由于 mysqldump 缺省时把输出定位到标准输出，你需要重定向标准输出。例如，把数据库备份到 bd_name.bak 中：

shell> mydqldump db_name>db_name.bak

你可以备份多个数据库，注意这种方法将不能指定数据表：

shell> mydqldump --databases db1 db1>db.bak

你也可以备份整个数据库系统的拷贝，不过对于一个庞大的系统，这样做没有什么实际的价值：

shell> mydqldump --all-databases>db.bak

虽然用 mysqldump 导出表的结构很有用，但是恢复大量数据时，众多 SQL 语句使恢复的效率降低。你可以通过使用--tab 选项，分开数据和创建表的 SQL 语句。

-T，--tab= 在选项指定的目录里，创建用制表符（tab）分隔列值的数据文件和包含创建表结构的 SQL 语句的文件，分别用扩展名.txt 和.sql 表示。该选项不能与--databases 或 --all-databases 同时使用，并且 mysqldump 必须运行在服务器主机上。

例如，假设数据库 db 包括表 tbl1，tbl2，你准备备份它们到/var/mysqldb

shell>mysqldump --tab=/var/mysqldb/ db

其效果是在目录/var/mysqldb 中生成 4 个文件，分别是 tbl1.txt、tbl1.sql、tbl2.txt 和 tbl2.sql。

2、mysqldump 实用程序时的身份验证的问题

同其他客户机一样，你也必须提供一个 MySQL 数据库帐号用来导出数据库，如果你不是使用匿名用户的话，可能需要手工提供参数或者使用选项文件：

如果这样：

shell>mysql -u root –pmypass db_name>db_name.sql

或者这样在选项文件中提供参数：

[mysqldump]

user=root

password=mypass

然后执行

shell>mysqldump db_name>db_name.sql

那么一切顺利，不会有任何问题，但要注意命令历史会泄漏密码，或者不能让任何除你之外的用户能够访问选项文件，由于数据库服务器也需要这个选项文件时，选项文件只能被启动服务器的用户（如，mysql）拥有和访问，以免泄密。在 Unix 下你还有一个解决办法，可以在自己的用户目录中提供个人选项文件（~/.my.cnf），例如，/home/some_user/.my.cnf，然后把上面的内容加入文件中，注意防止泄密。在 NT 系统中，你可以简单的让 c:\my.cnf 能被指定的用户访问。

你可能要问，为什么这么麻烦呢，例如，这样使用命令行：

shell>mysql -u root –p db_name>db_name.sql

或者在选项文件中加入

[mysqldump]

user=root

password

然后执行命令行：

shell>mysql db_name>db_name.sql

你发现了什么？往常熟悉的 Enter password:提示并没有出现，因为标准输出被重定向到文件 db_name.sql 中了，所以看不到往常的提示符，程序在等待你输入密码。在重定向的情况下，再使用交互模式，就会有问题。在上面的情况下，你还可以直接输入密码。然后在文件 db_name.sql 文件的第一行看到:

Enter password:#……..

你可能说问题不大，但是 mysqldump 之所以把结果输出到标准输出，是为了重定向到其它程序的标准输入，这样有利于编写脚本。例如：

用来自于一个数据库的信息充实另外一个 MySQL 数据库也是有用的：

shell>mysqldump --opt database | mysql --host=remote-host -C database

如果 mysqldump 仍运行在提示输入密码的交互模式下，该命令不会成功，但是如果 mysql 是否运行在提示输入密码的交互模式下，都是可以的。

如果在选项文件中的[client]或者[mysqldump]任何一段中指定了 password 选项，且不提供密码，即使，在另一段中有提供密码的选项 password=mypass,例如

[client]

user=root

password

[mysqldump]

user=admin

password=mypass

那么 mysqldump 一定要你输入 admin 用户的密码:

mysql>mysqldump db_name

即使是这样使用命令行:

mysql>mysqldump –u root –ppass1 db

也是这样，不过要如果-u 指定的用户的密码。

其它使用选项文件的客户程序也是这样

3、有关生成 SQL 语句的优化控制

--add-locks 生成的 SQL 语句中，在每个表数据恢复之前增加 LOCK TABLES 并且之后 UNLOCK TABLE。(为了使得更快地插入到 MySQL)。

--add-drop-table 生成的 SQL 语句中，在每个 create 语句之前增加一个 drop table。

-e, --extended-insert 使用全新多行 INSERT 语法。(给出更紧缩并且更快的插入语句)

下面两个选项能够加快备份表的速度：

-l, --lock-tables. 为开始导出数据前，读锁定所有涉及的表。

-q, --quick 不缓冲查询，直接倾倒至 stdout。

理论上，备份时你应该指定上诉所有选项。这样会使命令行过于复杂，作为代替，你可以简单的指定一个--opt 选项，它会使上述所有选项有效。

例如，你将导出一个很大的数据库：

shell> mysqldump --opt db_name > db_name.txt

当然，使用--tab 选项时，由于不生成恢复数据的 SQL 语句，使用--opt 时，只会加快数据导出。

4、恢复 mysqldump 备份的数据

由于备份文件是 SQL 语句的集合，所以需要在批处理模式下使用客户机

- 如果你使用 mysqldump 备份单个数据库或表，即：

shell>mysqldump --opt db_name > db_name.sql

由于 db_name.sql 中不包括创建数据库或者选取数据库的语句，你需要指定数据库

shell>mysql db2 < db_name.sql

- 如果，你使用--databases 或者--all-databases 选项，由于导出文件中已经包含创建和选用数据库的语句，可以直接使用，不比指定数据库，例如：

shell>mysqldump --databases db_name > db_name.sql

shell>mysql <db_name.sql

- 如果你使用--tab 选项备份数据，数据恢复可能效率会高些

例如，备份数据库 db_name 后在恢复：

shell>mysqldump --tab=/path/to/dir --opt test

如果要恢复表的结构，可以这样：

```
shell>mysql < /path/to/dir/tbl1.sql
…
```

如果要恢复数据，可以这样

```
shell>mysqlimport -l db /path/to/dir/tbl1.txt
…
```

如果是在 Unix 平台下使用（推荐），就更方便了：

```
shell>ls -l *.sql | mysql db
shell>mysqlimport --lock-tables db /path/to/dir/*.txt
```

5.3.4 用直接拷贝的方法备份恢复

根据本章前两节的介绍，由于 MySQL 的数据库和表是直接通过目录和表文件实现的，因此直接复制文件来备份数据库数据，对 MySQL 来说特别方便。而且自 MySQL 3.23 起 MyISAM 表成为缺省的表的类型，这种表可以为在不同的硬件体系中共享数据提供了保证。

使用直接拷贝的方法备份时，尤其要注意表没有被使用，你应该首先对表进行读锁定。

备份一个表，需要三个文件：

对于 MyISAM 表：

tbl_name.frm　表的描述文件

tbl_name.MYD　表的数据文件

tbl_name.MYI　表的索引文件

对于 ISAM 表：

tbl_name.frm　表的描述文件

tbl_name.ISD　表的数据文件

tbl_name.ISM　表的索引文件

你直接拷贝文件从一个数据库服务器到另一个服务器，对于 MyISAM 表，你可以从运行在不同硬件系统的服务器之间复制文件，例如，SUN 服务器和 INTEL PC 机之间。

5.3.5 总结

本节介绍了备份恢复数据库的多种方法，读者可以根据需要选用，对于文中涉及到的 SQL 语句、工具主要有：

1、 SELECT…INTO OUTFILE 和 LOAD DATA INFILE

2、 mysqldump

3、 mysqlimport

对于这些内容，读者需要注重掌握的是 mysqldump 实用程序的使用，以及 mysql 批处理模式运行包含 SQL 语句的文件的方法。这在备份和恢复数据库表中非常常用。另外要注意的是直接拷贝的方法。

5.4 使用更新日志文件

你不可能随时备份数据，但你的数据丢失时，或者数据库目录中的文件损坏时，

你只能恢复已经备份的文件，而在这之后的插入或更新的数据，就无能为力了。解决这个问题，就必须使用更新日志。更新日志可以实时记录更新、插入和删除记录的 SQL 语句。

5.4.1 启用日志

当以--log-update=file_name 选项启动时，mysqld 将所有更新数据的 SQL 命令写入记录文件中。文件被写入数据目录并且有一个名字 file_name.#，这里#是一个数字，它在每次执行 mysqladmin refresh 或 mysqladmin flush-logs、FLUSH LOGS 语句、或重启服务器时加 1。

如果你不指定 file_name，缺省使用服务器的主机名。

如果你在文件命中指定扩展名，那么更新日志不再使用顺序文件，使用指定的文件。但是当你它在每次执行 mysqladmin refresh 或 mysqladmin flush-logs、FLUSH LOGS 语句、或重启服务器时日志文件被清空。

更新记录很聪明，因为它仅仅记载真正更新数据的语句。因此一个用 WHERE 的 UPDATE 或 DELETE 找不到行，它就不被写入记录文件。它甚至跳过将设置一个列为它已经有的值的 UPDATE 语句。

5.4.2 重写日志

必须着重指出的是，在下列情况之一，将使用新的日志文件——日志文件的顺序自动增加（未指定 file_name 或者指定的 file_name 不包括扩展名）或者清空文件（指定的 file_name 包括扩展名）：

- 命令 mysqladmin refresh
- 命令 mysqladmin flush-logs
- SQL 语句 FLUSH LOGS
- 服务器重新启动

5.4.3 恢复日志内容

对于所有的更新日志文件，你都可以把它指定为 mysql 客户机的输入，来执行其中的 SQL 语句，恢复数据。例如：

```
shell>mysql <hostname.nnn
```

但是，你可能因为执行 DROP DATABASE 误删除了，希望只恢复该数据库的内容，为了这个目的，你可以使用--one-database 选项：

```
shell>mysql --one-database db_name < hostname.nnn
```

如果你要批量恢复更新日志的数据，在 Unix 中可以这样：

```
$ ls –t –r –l hostname.[0-9]* | xargs cat | mysql --one-database db_name
```

注意由于文件是按时间时间排序的，如果你修改的其中的任何文件，都会因为顺序的错误导致可能导入错误的数据。

如果按文件顺序恢复数据，就去掉-t 和-r 选项：

```
$ ls -l hostname.[0-9]* | xargs cat | mysql --one-database db_name
```

5.4.4 总结

本节介绍了有关日志文件尤其是更新日志的操作。需要注意的是如何启用更新日志、服务器重写日志的时机、以及如何恢复更新日志的内容。尤其是其中的各种技巧，例如如何按照日志文件生成的时间顺序恢复日志，如何只恢复指定数据库的内容。

5.5 使用 MySQL 内建复制功能

MySQL 内部复制功能是建立在两个或两个以上服务器之间，通过设定它们之间的主-从关系来实现的。其中一个作为主服务器，其它的作为从服务器。本节将详细讨论如何配置两台服务器，将一个设为主服务器，另一个设为从服务器。并且描述一下在它们之间进行切换的处理过程。本节是在 MySQL 的 3.23.25 版本上进行的配置设置过程，并且也是在这个版本上进行的测试。MySQL 开发人员建议最好使用最新版本，并且主-从服务器均使用相同的版本。同时 MySQL 3.23 版本仍然是 beta 测试版，而且这个版本可能不能向下兼容。

5.5.1 配置主服务器

我们将指定两台服务器。A（IP 为 192.168.0.1）作为主服务器（简称为主机）。B（IP 为 192.168.0.2）作为从服务器（简称为从机）。

1、建立用于备份的帐号

MySQL 的复制功能的实现过程为：从机(B)与主机(A)连接，然后读出主机的二进制更新日志，再将发生的变化合并到自己的数据库中。从机需要一个用户帐号来与主机连接，所以在主机上创建一个帐号，并只给它 FILE 权限，如下操作：

```
mysql>GRANT FILE ON *.* TO replicate@192.168.0.2 IDENTIFIED BY 'password';
```


有关授权的知识将在第七章介绍

为了从机能够与主机连接，要在主机上运行'FLUSH PRIVILEGES'：

```
mysql> 'FLUSH PRIVILEGES;
```

不过不要担心，因为我们将在下面的步骤中停掉服务器。

2、修改选项文件

现在我们需要主机数据库的一个快照，并且对主机进行配置，允许生成二进制的更新日志。首先编辑'my.cnf'文件，以便允许二进制更新日志，所以在[mysqld]部分的下面某个地方增加一行：'log-bin'。在下一次服务器启动时，主机将生成二进制更新日志(名为：<主机名>-bin.<增量序号#>)。

```
[mysqld]
… …
log-bin
```

为了让二进制更新日志有效，关闭 MySQL 服务程序，然后将主机上的所有数据库目录到另一个目录中，接着重新启动 mysqld。

3 得到服务器数据库的一个快照

请确定得到了所有数据库，否则在进行复制时，如果一个表在主机上存在但在从机上不存在，将因为出错而退出。现在你已经得到了数据的快照，和一个从建立快照以来的二进制日志，上面记录着任何对数据库的修改。

请注意 MySQL 数据文件(*.MYD，*.MYI 和*.frm)是不依赖于文件系统的，但是 ISAM 表的数据文件是依赖系统的，但是 MySQL3.23 以上均采用 MyISAM 表。所以你可以仅仅进行文件传输，如从 Solaris 到 Linux。只有你处于一个异种的服务器环境，并且仅仅复制不能共享数据表文件时，你将不得不使用 mysqldump 实用程序或其它的定制脚本来得到数据快照，但是这种可能性还是很小的。

5.5.2 配置从服务器

1、迁移主机的数据库目录

停掉从机上的 MySQL 服务程序，并且把从主机上拷贝来的数据库目录移到从机上的 data 目录下。请确认将目录的拥有者和属组改变为 MySQL 用户相应值，并且修改文件模式为 660(只对拥有者和属组可读、可写)，目录本身为 770(只对拥有者和属组可读、可写和可执行)。

在从机上启动 MySQL 服务程序，确认 MySQL 工作正常。运行几个 select 查询(不要 update 或 insert 查询)，看一看在第一步中得到的数据快照是否成功。接着，在测试成功后关掉 MySQL 服务程序。

2、修改选项文件

在从机上配置需要访问的主机，以便接收主机的更改。所以需要编辑务机上的'my.cnf'文件，在[mysqld]部分中增加下面几行：

```
master-host=192.168.0.1
master-user=replicate
master-password=password
```

在启动从机服务程序后，从机服务程序将查看在'my.cnf'文件中所指定的主机，查看是否有改变，并且将这些改变合并到自己的数据库中。从机保持了主机的更新记录，这些记录是从主机的'master.info'文件中接收下来的。从机线程的状态可以通过 sql 命令'SHOW SLAVE-STATUS'看到。在从机上处理二进制日志中如果发生错误，都将导致从机线程的退出，并且在*.err 的日志文件中生成一条信息。然后错误可以被改正，接着可以使用 sql 语句'SLAVE START'来重新启动从机线程。线程将从主机二进制日志处理中断的地方继续处理。

至此，在主机上所发生的数据改变应该已经复制到从机上了，要测试它，你可以在主机上插入或更新一条记录，而在从机上选择这条记录。

现在我们拥有了从 A 机到 B 机的这种主-从关系，这样当 A 机可能当机的时候，允许我们将所有的查询重定向到 B 机上去，但是当 A 机恢复时，我们没有办法将发生的改变恢复到 A 机中去。为了解决这个问题，我们创建从 B 机到 A 机的主-从关系。

5.5.3 创建相互的主从关系

1、从机的配置

首先在B机上的my.cnf文件中，在[mysqld]部分中加入'log-bin'，接着重新启动mysqld，然后创建可在它的上面执行复制功能的用户帐号，使用：

GRANT FILE ON *.* TO replicate@192.168.0.1 IDENTIFIED BY 'password';

在B机上运行'FLUSH PRIVILEGES'命令，以便装入在加入复制用户后的新的授权表。

2、主机的配置

接着回到A机上，在它的'my.cnf'中加入下面几行：

master-host=192.168.0.2

master-user=replicate

master-password=password

在重启A机的服务程序之后，现在我们一拥有了在A机与B机之间的相互主-从关系。不管在哪个服务器上更新一条记录或插入一条记录，都将被复制到另一台服务器上。要注意的是：我不敢确定一个从机合并二进制日志变化的速度有多快，所以用这种方法来进行插入或更新语句的负载平衡可能不是一个好办法。

5.5.4 总结

首先，要确定得到了一个完整的数据快照。如果忘记拷贝一个表或数据库将导致从机线程序停止。生成快照的时刻是很关健的。你应该确保在拷贝数据文件之前二进制日志功能是无效的。如果在得到快照之前就允许了二进制日志功能，从机的线程可能会停止，原因就是当线程试图导入重要的记录时，可能会由于主键重复而停止。最好就是接照第二部分所讨论的处理办法来做：关闭-拷贝-允许二进制日志功能重启。

你可能想要按照最初的一种方式来配制复制处理，并且在合适的时间关注从机，确保从机与主机保持同步。

5.5 总结:备份恢复数据的一般步骤

在上面的各节中详细叙述了备份和恢复数据的方方面面，读者可能还是对整个过程没有一个明确的认识，在这一节中，笔者将作一个简单的总结，是读者能够把握备份和恢复数据的一般过程。

下面是备份一个数据库的例子：

1、备份前读锁定涉及的表

mysql>LOCK TABLES tbl1 READ,tbl1 READ,…

如果，你在mysqldump实用程序中使用--lock-tables选项则不必使用如上SQL语句。

2、导出数据库中表的结构和数据

shell>mysqldump --opt db_name>db_name.sql

3、启用新的更新日志

shell>mysqladmin flush-logs

这样可以记录你备份后的数据改变为恢复数据准备。

4、解除表的读锁

mysql>UNLOCK TABLES;

为了加速上述过程，你可以这样做:

shell> mysqldump --lock-tables --opt db_name>db_name.sql; mysqladmin flush-logs

但是这样可能会有点小问题。上命令在启用新的更新日志前就恢复表的读锁，

在更新繁忙的站点，可能有备份后的更新数据没有记录在新的日志中。

现在恢复上面备份的数据库

1、对涉及的表使用写锁

mysql>LOCK TABLES tbl1 WRITE,tbl1 WRITE,…

2、恢复备份的数据

shell>mysql db_name < db_name.sql

3、恢复更新日志的内容

shell>mysql --one-database db_name < hostname.nnn

假设需要使用的日志名字为 hostname.nnn

4、启用新的更新日志

shell>mysqladmin flush-logs

5、解除表的写锁

mysql>UNLOCK TABLES;

希望上面的例子能给你启发，因为备份数据的手法多种多样，你所使用的和上面所述可能大不一样，但是对于备份和恢复中，表的锁定、启用新的更新日志的时机应该是类似的，仔细考虑这个问题。

思考题

请在完成这些思考题之前，启用常规日志和更新日志，因为下面的思考题将涉及到这个问题。

1、 使用带有带有--all-databases 选项的 mysqldump 实用程序，备份你现有的所有数据到一个文件 all.sql 中（如果你的数据不太多的话），查看输出的 SQL 语句。

2、 使用带有--ab 选项的 mysqldump，把你的 test 数据库中的表结构和数据分别备份到一个目录中。检查一下生成的 SQL 语句与上题中备份文件中的有什么不同。

3、 建立一个新的数据库 test1，把上题的备份文件恢复到该数据库中。请简述一下过程。

4、 查看你的更新日志和常规日志，看看上述操作是否都留下了记录以及留下了什

么记录。

5、 先在 test 数据库中的任意一个表中 INSERT 一个记录，然后在删除这个记录。（目的是在更新日志中留下记录。）删除数据库 test1，试试如何从更新日志恢复 test1 数据库。

第6章

数据库的维护与修复

本章要点：

- 维护表的工具及使用
- 如何对表进行由简到繁的检查
- 如何对表进行由容易到困难的修复
- 如何建立一个数据库的维护规范

你可能希望 MySQL 从安装以后就始终正常运行。但是，因为各种原因，MySQL 随时可能会出现问题。各种意外情况都可能是服务器崩溃或者主机意外停机，都会导致数据库表的损坏。这些错误通常是对表进行修改且未完全写入时引起的。而且这些数据往往是最新的数据，通常不可能在备份数据中找到，通常你需要修复数据表，以尽可能的抢救重要数据。

本章的主要内容是检查和修复表的问题，对于表的检查和修复，最重要的工具是 myisamchk 和 isamchk 实用程序。

6.1 数据库表的检查、修复与优化

表的故障检测和修正的一般过程如下：

- 检查出错的表。如果该表检查通过，则完成任务，否则必须修复出错的数据库表。
- 在开始修复之前对表文件进行拷贝，以保证数据的安全。
- 开始修复数据库表。
- 如果修复失败，从数据库的备份或更新日志中恢复数据。

在使用 myisamchk 或 isamchk 检查或修复表之前，应该首先注意：

- 建立数据库备份和使用更新日志，以防修复失败，丢失数据。
- 仔细阅读本章内容以后再进行操作，尤其是不应该在阅读“避免与 MySQL 服务器交互作用”之前进行操作。因为，在你没有足够的知识之前贸然操作，可能会引起严重的后果。
- 如果你在 Unix 平台上对表进行维护时，应该首先注册到专用的帐户 mysql，以避免对表读写访问产生所有权的问题，以及破坏数据库目录的所有权限。

6.1.1 数据库表的维护工具

MySQL 的 myisanchk 和 isamchk 实用程序很类似，基本上它们具有同样的使用方法。它们之间的主要区别时所使用的表的类型。为了检查/修复 MyISAM 表(.MYI 和.MYD)，你应该使用 myisamchk 实用程序。为了检查/修复 ISAM 表(.ISM 和.ISD)，你应该使用 isamchk 实用程序。

- 为了使用任一个使用程序，应指明你要检查或修复的表，myisamchk 和 isamchk 的使用方法为：

```
shell>myisamchk options tbl_name
shell>isamchk options tbl_name
```

如果你愿意，你可以在命令行命名几个表。

- 你也能指定一个名字作为一个索引文件(用“ .MYI”或“.ISM”后缀)，它允许你通过使用模式“*.MYI”或“.ISM”指定在一个目录所有的表。例如，如果你在一个数据库目录，你可以这样在目录下检查所有的表：

```
shell> myisamchk *.MYI
shell>isamchk *.ISM
```

- 如果你不在数据库目录下，你可指定目录的路径：

```
shell> myisamchk options /path/to/database_dir/*.MYI
shell> isamchk options   /path/to/database_dir/*.ISM
```

- 你甚至可以通过为 MySQL 数据目录的路径指定一个通配符来作用于所有的数据库中的所有表：

```
shell> myisamchk options /path/to/datadir/*/*.MYI
shell> isamchk options /path/to/database_dir/*/*.ISM
```

这个方法无法在 windows 平台下使用。

 注意　不论是 myisamchk 还是 isamchk 都不对表所在的位置做任何判断，因此，应该或者在包含表文件的目录运行程序，或者指定表的路径名。这允许你将表文件拷贝到另一个目录中并用该拷贝进行操作。

6.1.2 检查数据库表

myisamchk 和 isamchk 提供了表的检查方法，这些方法在彻底检查表的程度方面有差异。

标准的方法检查表

通常用标准的方法就足够了。对表使用标准的方法进行检查，不使用任何选项直接调用即可，或用-s 或--silent 选项的任何一个。:

myisamchk tbl_name

isamchk tbl_name

这能找出所有错误的 99.99%。它不能找出的是仅仅涉及数据文件的损坏(这很不常见)。

完全彻底的数据检查

为了执行扩充检查，使用--extend-check 或-e 选项，这个选项检查数据:

myisamchk -e tbl_name

isamchk -e tbl_name

它做一个完全彻底的数据检查(-e 意思是“扩展检查”)。它对每一行做每个键的读检查以证实他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，你能增加--verbose(-v)选项。这使得 myisamchk 或 isamchk 继续一直到最多 20 个错误。在一般使用中，一个简单的标准检查（没有除表名以外的参数）就足够了。

中等程度的检查

指定选项--medium-check 或-m

myisamchk -m tbl_name

中的程度的检查不如扩展检查彻底，但速度快一些。其意义不大，较少使用。

 如果对于--extend-check 检查不报告错误，则可以肯定表是完好的。如果你仍然感觉表有问题，那原因肯定在其它地方。应重新检查人和好像有问题的查询以验证查询是正确书写的。如果你认为问题可能是 MySQL 服务器的原因，应该考虑整理一份故障报告或者升级到新的版本上。

可能有用的检查选项：

1、-i 或—information 打印统计信息,例如：

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。

2、-C, --check-only-changed

只检查上一次检查后被修改的表

6.1.3 修复数据库表

一张损坏的表的症状通常是查询意外中断并且你能看到例如这些错误：

- “tbl_name.frm”被锁定不能改变。
- 不能找到文件“tbl_name.MYI”（Errcode ：### ）。
- 从表处理器的得到错误###(此时，错误 135 是一个例外)。
- 意外的文件结束。
- 记录文件被毁坏。

在这些情况下，你必须修复表。表的修复是一项非常困难的工作，很多情况下令人束手无策。然而，有一些常规的知道思想和过程，可以遵循它们来增加修正表的机会。通常，开始是可以用最快的修复方法，看看能否袖珍故障。如果发现不成功，可以逐步升级到更彻底的但更慢的修复方法。如果仍旧难以修复，就应该从备份中恢复了。在上一章已经详细介绍了这一部分内容。

1、简单安全的修复

为了修复一个表执行下列步骤：

- 首先，用--recover,-r 选项修正表，并且用--quick，-q 选项，来只根据索引文件的内容进行恢复。这样不接触数据文件来修复索引文件。(-r 意味着“恢复模式”)

```
myisamchk -r -q tbl_name
isamchk -r -q tbl_name
```

- 如果问题仍旧存在，则忽略--quick 选项，允许修复程序修改数据文件，因为这可能存在问题。下面的命令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：

```
myisamchk -r tbl_name
isamchk -r tbl_name
```

- 如果前面的步骤失败，使用。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)。

```
myisamchk --safe-recover tbl_name
isamchk --safe-recover tbl_name
```

2、困难的修理

如果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，你只应该到这个阶段 。在这种情况下，创建一个新的索引文件是必要的。按如下这样的步骤做：

- 定位到包含崩溃表的数据库目录中
- 把数据文件移更安全的地方。
- 使用表描述文件创建新的(空)数据和索引文件：

```
shell> mysql db_name
mysql> DELETE FROM tbl_name;
```

mysql> quit

上述语句将重新创建新的空表，并使用表的的描述文件 tbl_name.frm 重新生成新的数据和索引文件。

- 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。）
- 在使用标准的修复方法。现在 myisamchk -r -q 应该工作了。（这不应该是一个无限循环）。

如果你拥有表的备份文件，那么一切过程就容易的多。从备份文件中可以恢复表的描述文件，然后在检查表，有可能还要继续使用标准的修复方法，应该纠可以解决问题了。

3、非常困难的修复

只有描述文件也破坏了，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。

从一个备份恢复描述文件并且回到阶段 2。你也可以恢复索引文件并且回到阶段 1。对于后者，你应该用 myisamchk -r 启动。

如果因为某种原因，数据的备份文件丢失或者没有备份文件，但是你还记得建立表的 CREATE TABLE 语句，那么太好了，这样还是可以恢复索引文件：

- 定位到包含崩溃表的数据库目录中
- 把数据文件移更安全的地方。再把数据库目录中的对应的目录删去.。
- 调用 mysql 并发复 CREATE TABLE 语句建立该表。
- 退出 mysql，将原始的数据文件和索引文件移回到数据库的目录中，替换刚才新建的文件。
- 然后回到阶段 2，修复表。也可以只移回数据文件，这样保留新的描述和索引文件，然后回到阶段 1，继续用标准的方法修复表。

6.1.4 优化数据库表

修复表的方法，同样可以用来对数据表进行优化。

为了组合成碎片的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 和 isamchk：

shell> myisamchk -r tbl_name

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一张表，OPTIMIZE TABLE 比较容易，但是 myisamchk 更快。也没有在一个实用程序和服务器之间不必要的交互可能性，因为当你使用 OPTIMIZE TABLE 时，服务器做所有的工作：

OPTIMIZE TABLE tbl_name

6.1.5 指定维护过程中使用的内存

myisamchk 和 isamchk 的运行可能会花很长的时间，尤其是正在处理一个达标或者使用一个更广泛的检查和修复方法时。当你运行 myisamchk 和 isamchk 时，内存分配很重要。myisamchk 和 isamchk 使用不超过你用-O 选项指定的内存量。如果你想在很大的文件上使

用 myisamchk 和 isamchk，你首先应该确定你想要它使用多少内存。缺省仅使用大约 3M 来修复。通过使用更大的值，你能使 myisamchk 和 isamchk 更快地操作。下面列出最重的控制程序使用的缓冲区大小的变量：

表 6-1 控制缓冲区的变量

变量	含义
key_buffer_size	用于存放索引块的缓冲区大小
read_buff_size	读操作用的缓冲区大小
sort_buffer_size	排序用的缓冲区大小
write_buffer_size	写操作用的缓冲区大小

要想查看任一个程序使用的这些变量的缺省值，可用--help 选项运行该程序。要想使用其它的值，可在该命令上使用--set-variable variable=value 或 –O variable=value。有些材料上说，您可以将变量的名字简化成 key,read,sort 和 write，但是不是所有的分发的维护程序都可以这样，例如，如果有两个变量 sort_buffer_size 和 sort_key_size，sort 不能唯一决定使用那个变量，会产生一个错误。因此建议你用完整的值。

例如，如果你有多于 32M 内存，你能使用例如这些选项(除了任何你可能指定的选项)：

shell> myisamchk -O sort_buffer_size=16M -O key_buffer_size=16M -O read_buff_size =1M -O write_buffer_size =1M ...

shell> myisamchk -O sort_buffer_size=16M -O key_buffer_size=16M -O read_buff_size =1M -O write_buffer_size =1M ...

使用-O sort=16M 应该可能对大多数情形就足够了。

--sort_buffer_size 只能利用--recover 选项来使用（而不是利用--safe_recover）,并且在这种情况下，key_buffer 不能使用。

必须明白，myisamchk 和 isamchk 使用在 TMPDIR 里面的临时文件。如果 TMPDIR 指向一个内存文件系统，你可能很容易得到内存溢出的错误。如果它发生，设定 TMPDIR 指向有更多空间的某个目录并且重启 myisamchk 和 isamchk。

6.1.6 总结

本节介绍了 myisamchk 和 isamchk 实用程序的基本用法。本节主要介绍了如何一步步的检查表是否有问题，如何一步步修复有问题的表，以及如何使用这两个实用程序优化表。如果你拥有大量表，检查和修复时将需要消耗大量内存，注意如何在使用过程中为维护程序指定使用的内存数量。

6.2 避免与 MySQL 服务器交互作用

如果你同时运行表的检查/修复程序时，你或许不想让 MySQL 服务器和实用程序同时访问一个表。如果两个程序都向表中写数据显然会造成很大的麻烦，甚至会有意外情况发生。如果表正由一个程序写入，同时进行读取的另一个程序也会产生混乱的结果。

6.2.1 锁定表的的方法

防止客户机的请求互相干扰或者服务器与维护程序相互干扰的方法主要有多种。如果你关闭数据库，就可以保证服务器和 myisamchk 和 isamchk 之间没有交互作用。但是停止服务器的运行并不是一个好注意，因为这样做会使得没有故障的数据库和表也不可用。本节主要讨论的过程，是避免服务器和 myisamchk 或 isamchk 之间的交互作用。实现这种功能的方法是对表进行锁定。

服务器由两种表的锁定方法。

1、内部锁定。

内部锁定可以避免客户机的请求相互干扰——例如，避免客户机的 SELECT 查询被另一个客户机的 UPDATE 查询所干扰。也可以利用内部锁定机制防止服务器在利用 myisamchk 或 isamchk 检查或修复表时对表的访问。

语法：

锁定表：LOCK TABLES tbl_name {READ | WRITE},[tbl_name {READ | WRITE},…]

解锁表：UNLOCK TABLES

LOCK TABLES 为当前线程锁定表。UNLOCK TABLES 释放被当前线程持有的任何锁。当线程发出另外一个 LOCK TABLES 时，或当服务器的连接被关闭时，当前线程锁定的所有表自动被解锁。

如果一个线程获得在一个表上的一个 READ 锁，该线程(和所有其他线程)只能从表中读。如果一个线程获得一个表上的一个 WRITE 锁，那么只有持锁的线程 READ 或 WRITE 表，其他线程被阻止。

每个线程等待(没有超时)直到它获得它请求的所有锁。

WRITE 锁通常比 READ 锁有更高的优先级，以确保更改尽快被处理。这意味着，如果一个线程获得 READ 锁，并且然后另外一个线程请求一个 WRITE 锁，随后的 READ 锁请求将等待直到 WRITE 线程得到了锁并且释放了它。

显然对于检查，你只需要获得读锁。再者钟情跨下，只能读取表，但不能修改它，因此他也允许其它客户机读取表。对于修复，你必须获得些所以防止任何客户机在你对表进行操作时修改它。

2、外部锁定

服务器还可以使用外部锁定（文件级锁）来防止其它程序在服务器使用表时修改文件。通常，在表的检查操作中服务器将外部锁定与 myisamchk 或 isamchk 作合使用。但是，外部锁定在某些系统中是禁用的，因为他不能可靠的进行工作。对运行 myisamchk 或 isamchk 所选择的过程取决于服务器是否能使用外部锁定。如果不使用，则必修使用内部锁定协议。

如果服务器用--skip-locking 选项运行，则外部锁定禁用。该选项在某些系统中是缺省的，如 Linux。可以通过运行 mysqladmin variables 命令确定服务器是否能够使用外部锁定。检查 skip_locking 变量的值并按以下方法进行：

- 如果 skip_locking 为 off，则外部锁定有效您可以继续并运行人和一个实用程序来检查表。服务器和实用程序将合作对表进行访问。但是，运行任何一个实用程序

之前，应该使用 mysqladmin flush-tables。为了修复表，应该使用表的修复锁定协议。

- 如果 skip_locaking 为 on，则禁用外部锁定，所以在 myisamchk 或 isamchk 检查修复表示服务器并不知道，最好关闭服务器。如果坚持是服务器保持开启状态，月确保在您使用此表示没有客户机来访问它。必须使用卡党的锁定协议告诉服务器是该表不被其他客户机访问。

6.2.2 检查表的锁定协议

本节只介绍如果使用表的内部锁定。对于检查表的锁定协议，此过程只针对表的检查，不针对表的修复。

调用 mysql 发布下列语句：

$mysql –u root –p db_name

mysql>LOCK TABLE tbl_name READ;

mysql>FLUSH TABLES;

该锁防止其它客户机在检查时写入该表和修改该表。FLUSH 语句导致服务器关闭表的文件，它将刷新仍在告诉缓存中的任何为写入的改变。

2、执行检查过程

$myisamchk tbl_name

$ isamchk tbl_name

3、释放表锁

mysql>UNLOCK TABLES;

如果 myisamchk 或 isamchk 指出发现该表的问题，将需要执行表的修复。

6.2.3 修复表的锁定协议

本节只介绍如果使用表的内部锁定。修复表的锁定过程类似于检查表的锁定过程，但有两个区别。第一，你必须得到写锁而非读锁。由于你需要修改表，因此根本不允许客户机对其进行访问。第二，必须在执行修复之后发布 FLUSH TABLE 语句，因为 myisamchk 和 isamchk 建立的新的索引文件，除非再次刷新改表的高速缓存，否则服务器不会注意到这个改变。本例同样适合优化表的过程。

1、调用 mysql 发布下列语句：

$mysql –u root –p db_name

mysql>LOCK TABLE tbl_name WRITE;

mysql>FLUSH TABLES;

2、做数据表的拷贝，然后运行 myisamchk 和 isamchk：

$cp tbl_name.* /some/other/dir

$myisamchk --recover tbl_name

$ isamchk --recover tbl_name

--recover 选项只是针对安装而设置的。这些特殊选项的选择将取决与你执行修复的类型。

3、再次刷新高速缓存，并释放表锁

mysql>FLUSH TABLES;

mysql>UNLOCK TABLES;

6.2.4 总结

维护不是简单的运行 myisamchk 维护程序就可以的。因为大多数情况下，管理员进行表的检查和修复时，服务器都要持续运行，因此如果方法不当，很可能维护程序会与服务器产生冲突。

本节对锁定表的方法做了详细的介绍，表的锁定包括内部锁定和外部锁定，本节主要介绍了内部锁定，对于检查和修复，使用的锁定协议也不同，分别为读锁定和写锁定。通过表的锁定，可以避免维护过程中与服务器发生的交互作用。

6.3 日志文件维护

由于日志文件是恢复数据库数据的重要参考，因此日志文件的维护也有十分重要的意义。当 MySQL 与日志文件一起使用时，你有时想要删除/备份旧的日志文件并且告诉 MySQL 在新文件中开始记录。

6.3.1 如何使用新的更新日志

如果你只使用一个更新日志，你只须清空日志文件，然后移走旧的更新日志文件到一个备份中，然后启用新的更新日志。

用下列方法可以强制服务器启用新的更新日志：

- mysqladmin flush-logs

你一般需要在命令行提供使用的数据库用户：

mysqladmin –u root –p flush-logs

- mysqladmin refresh

你一般需要在命令行提供使用的数据库用户：

mysqladmin –u root –p refresh

如果你正在使用 MySQL 3.21 或更早的版本，你必须使用 mysqladmin refresh。

- SQL 命令

FLUSH LOGS

- 重启服务器

上述方法都具有这样的功能：

关闭并且再打开标准和更新记录文件。如果你指定了一个没有扩展名的更新记录文件，新的更新记录文件的扩展数字将相对先前的文件加 1。

mysql>FLUSH LOGS;

6.3.2 如何使用新的常规日志

用上一小节的方法同样可以强制更新常规日志。

要准备备份常规日志，其步骤可能复杂一些：

```
$ cd mysql-data-directory
$ mv mysql.log mysql.old
$ mysqladmin flush-tables
```

然后做一个备份并删除“mysql.old”。

6.3.3 总结

在维护表的过程中，经常涉及到日志文件的操作。本节涉及如何启用新的日志文件，包括更新日志和常规日志。这里所述的方法，同样也适用二进制日志，有关二进制日志的知识，请看前一章内建复制的部分。

6.4 建立日常维护规范

在一个定期基础而非等到问题出现才实施数据库表的检查是一个好主意。为维护目的，你能使用 myisamchk -s 检查桌子。-s 选项使 myisamchk 以沉默模式运行，当错误出现时，仅仅打印消息。

6.4.1 建立一个数据库表维护规范

在一个定期基础而非等到问题出现才实施数据库表的检查是一个好主意。应该考虑到建立一个预防性维护的时间表，以协助自动问题，是你可以采取措施进行修正：

执行常规的数据库备份并允许更新日志。

安排定期的常规表检查。通过检查表，将减少使用备份的机会。这个工作，在 Windows 下使用计划任务，Unix 使用 cron 作业（一般从运行服务器所示用的该帐号的 crontab 文件中调用），并且很容易实现。

例如，你作为 mysql 用户运行服务器，则可以从 mysql 的 crontab 文件中建立定期检查。如果你不知道如何使用 cron，应使用下列命令查看相关的 Unix 手册页：

```
$man cron
$man crontab
```

作为服务器启动前的系统引导期间检查数据库表。及其可能会因早期的崩溃而重新启动。如果这样的花，数据库表可能已被毁坏，应该对它进行彻底检查。

6.4.2 创建一个适用于定期维护的脚本

为了运行自动的表检查，可以编写一个脚本，将目录改变为服务器数据目录并对所有的数据库表进行 myisamchk 和 isamchk。如果你只有 MyISAM 表或者只有 ISAM 表，则只需要其中一个程序，可以将无关的那个程序从脚本中注释掉。

该教本可以被 cron 调用，或者在系统启动期间被调用。

为维护目的，你能使用 myisamchk -s 检查桌子。-s,--silent 选项使 myisamchk 和 isamchk 以沉默模式运行，只有当错误出现时，才仅仅打印消息。另外 myisamchk 支持--fast 选项,该选项允许程序跳过自上次检查以来没有被修改过的人和表。

1、一个简单的脚本

例如，一个较为容易理解的简单脚本，它在服务器目录中检查所有表（DATADIR 应该修改成对应你系统的合适的值）：

```
#!/bin/sh
cd DATADIR
myisamchk  --silent  --fast  */*.MYI
isamchk  --silent  */*.ISM
```

2、一个较为复杂的脚本

实用此脚本的一个潜在的问题时：如果有许多表，通配符模式‘*/*.MYI’和‘*/*.ISM’可能会由于“too many arguments(参数过多)”或者命令行超过 shell 允许的长度而无法使用。脚本可以进一步修改为（同样，DATADIR 修改为适合你系统上的值）：

```
#!/bin/sh
datadir=DATADIR
find $dtatdir –name “*. MYI” -print | xargs myisamchk  --silent  --fast
    find $dtatdir –name “*. ISM” -print | xargs isamchk  --silent
```

当然你也可以在脚本中指定多个数据库目录。

3、如何执行脚本

假定你将脚本存为 check_tables，应该确保它是可执行的,当然建议你首先切换到专用户 mysql：

```
$su mysql
$vi check_tables(编辑脚本，你也可以使用你喜欢的编辑器)
$chmod +x check_tables
```

手工执行，检测你的脚本是否有错误：

```
$check_tables
```

在理想情况下应该没有输出结果。如果系统不支持外部锁定，游客蒽那个服务器将在你检查表时改变它。此时，脚本可能会把实际没有问题的表报告呈有问题的。如果系统能够支持外部锁定，则该问题就不会出现。

6.4.3 在 unix 中用 cron 定期检查表

本小节将说明如何建立脚本，使它通过 cron 并在系统启动期间执行。在这小节的例子中，笔者假定把脚本安装在/usr/local/mysql/bin 中，你需要修改该过程来检查每个服务器数据目录中的表。你可以使用不同的 check_tables 拷贝来进行，或者通过修改它来接受一个命令行参数进行，该参数指定了想要检查的数据目录。

假定对 mysql 用户从 crontab 文件中调用脚本 check_tables。

1、首先用该用户的身份注册：

```
$su mysql
```

2、生成一个临时文件，捕获已经调度的任务：

```
$crontab –l>/tmp/entries
```

3、在生成的文件最后一行添加内容：

把这一行 0 0 * * 0　/usr/local/mysql/bin/check_tables 加到临时文件的最后一行：

$echo　“0　0　*　*　0　/usr/local/mysql/bin/check_tables” >>/tmp/entries

它告诉 cron 在每个星期日的凌晨 0 时运行此选项。可以按要求改变时间或安排。有关这些选项的格式，参阅 crontab 的手册页。

4、重新安排调度

$crontab /tmp/entries

如果检查后有任何信息，cron 作业通常生成一个邮件消息给用户。由于使用--silent 选项，只有表存在错误时，才会有输出，也才会有邮件信息，因此不会产生大量无用邮件信息。（你现在应该明白，脚本采用--silent 选项的原因）

 对于这样定期的维护，你的服务器最好支持外部锁定，这样在检查表时，就不会发生访问冲突的情况。如果无法做到这一点，你最好在没有用户使用服务器的时候维护，例如凌晨。

6.3.4 在系统启动期间检查表

- 如果你使用的是 BSD 风格的系统，例如 OpenBSD，FreeBSD 等，并且已经将服务器的启动命令增加到/etc/rc.local，要在启动期间检查表，可以在启动服务器前从相应的文件中调用 check_tables。

- 如果对于使用 Sytem V 风格的启动方法的系统，例如，大多数的商业 Unix 系统，其启动方法是从/etc/rc.d 目录之一调用 mysql.server 脚本，则在数据库启动前检查表的过程比较复杂，因为这些目录中的脚本必须理解 start 和 stop 参数。

例如，象这样编写脚本，取名 mysql.check 当参数时 start 时调用 check_tables，当参数是 stop 时什么也不做：

```
#!/bin/sh
#See hou we sere called
case　“$1”　in
  start)
    echo –n “Checking MySQL tables:”
    if　[ -x /usr/local/mysql/bin/check_tables ] ; then
      /usr/local/mysql/bin/ check_tables
    fi
    ;;
  stop)
    #don’t do anything
    ;;
  *)
```

```
    echo "Usage:$o{start|stop}"
    exit 1
esac
exit 0
```

现在你可以安装 mysql.check 了，该过程类似乎在第二章介绍的让服务器自动启动的安装 mysql.server 的过程。必须给 mysql.check 一个运行级目录中较低的前缀号，才能使它在 mysql.server 前运行。例如，如果在运行级目录中以 S99mysql.server 连接到 mysql.server，则应该以 S98mysql.check 链接到 mysql.check。

 由于 Linux 集中了 BSD 和 Sytem V 系统的优点，所以，上面两种方法完全适用于 Linux。为了简便起见，一般使用第一个方法。

6.3.5 总结

本节演示了如何建立一个 MySQL 数据维护维护规范，并且给出了一个简单的脚本例子，可能无法适应不同的主机的情况，读者可以自己设计适合自己的脚本。

使用 cron 定时维护数据库表以及在系统主机启动时检查数据库表，是管理员非常常用的技巧。这些方法很简单，也非常容易使用。维护数据库表的一个重要原则就是，避免与数据库服务器的交互作用，因此尽量在没有用户使用数据库时维护它。

思考题

1、 给 test 数据库中的 pet 表锁定，然后使用维护工具 myisamchk 或者 isamchk 检查数据表。

2、 创建一个数据库 test2，然后使用 CREATE TABLE…SELECT 语句从 test.pet 复制一个表到 test2.pet。然后到相应数据库的目录，删除 test2.pet 表的表描述文件 pet.frm 文件，试着用重建表的方法修复表。描述一下步骤。

3、 将更新日志备份到一个目录，然后启用新的日志。

第7章

数据库安全

本章要点：

- MySQL 权限系统原理
- 如何授予撤销用户和授权
- 如何直接修改授权表
- 授予用户权限的规范和注意事项

作为一个 MySQL 的系统管理员，你有责任维护你的 MySQL 数据库系统的数据安全性和完整性。本文主要主要介绍如何建立一个安全的 MySQL 系统，从系统内部和外部网络两个角度，为你提供一个指南。

本章主要考虑下列安全性有关的问题：

- 为什么安全性很重要，你应该防范那些攻击?
- 服务器面临的风险（内部安全性），如何处理？
- 连接服务器的客户端风险（外部安全性），如何处理？

MySQL 管理员有责任保证数据库内容的安全性，使得这些数据记录只能被那些正确授权的用户访问，这涉及到数据库系统的内部安全性和外部安全性。

内部安全性关心的是文件系统级的问题，即，防止 MySQL 数据目录（DATADIR）被在服务器主机有账号的人（合法或窃取的）进行攻击。如果数据目录内 容的权限过分授予，使得每个人均能简单地替代对应于那些数据库表的文件，那么确保控 制客户通过网络访问的授权表设置正确，对此毫无意义。

外部安全性关心的是从外部通过网络连接服务器的客户的问题，即，保护 MySQL 服务器免受来自通过网络对服务器的连接的攻击。你必须设置 MySQL 授权表（grant table），使得他们不允许访问服务器管理的数据库内容，除非提供有效的用户名和口令。

下面就详细介绍如何设置文件系统和授权表 mysql，实现 MySQL 的两级安全性。

7.1 MySQL 的权限系统

MySQL 有一套先进的但非标准的安全/授权系统，掌握其授权机制是开始操作 MySQL 数据库必须要走的第一步，对于一个熟悉 SQL 基本操作的人来说，也是 MySQL 所有的知识中比较难以理解的一个部分。本节通过揭开其授权系统的运作机制，希望大家能够可以更好地操作和使用这个优秀的数据库系统。

MySQL 的安全系统是很灵活的，它允许你以多种不同方式设置用户权限。一般地，你可使用标准的 SQL 语句 GRANT 和 REVOKE 语句做，他们为你修改控制客户访问的授权表，然而，你可能由一个不支持这些语句的老版本的 MySQL（在 3.22.11 之前这些语句不起作用），或者你发觉用户权限看起来不是以你想要的方式工作。对于这种情况，了解 MySQL 授权表的结构和服务器如何利用它们决定访问权限是有帮助的，这样的了解允许你通过直接修改授权表增加、删除或修改用户权限，它也允许你在检查这些表时诊断权限问题。

7.1.1 授权表的结构

通过网络连接服务器的客户对 MySQL 数据库的访问由授权表内容来控制。这些表位于 mysql 数据库中，并在第一次安装 MySQL 的过程中初始化（运行 mysql_install_db 脚本）。授权表共有 5 个表：user、db、host、tables_priv 和 columns_priv。

7.1.1.1 授权表 user、db 和 host 的结构和作用

表 7-1　授权表 user、db 和 host 的结构

User 表	Db 表	Host 表
	作用域列	
Host	Host	Host
	Db	Db
User	User	
Password		
	数据库/表的权限列	
Alter_priv	Alter_priv	Alter_priv
Create_priv	Create_priv	Delete_priv
Delete_priv	Drop_priv	Drop_priv
Index_priv	Index_priv	Index_priv
Insert_priv	Insert_priv	References_priv
References_priv	Select_priv	Select_priv
Update_priv	Update_priv	Update_priv
Alter_priv	Alter_priv	Alter_priv

File_priv Grant_priv Process_priv Reload_priv Shutdown_priv	Grant_priv	Grant_priv

授权表的内容有如下用途：

- user 表

user 表列出可以连接服务器的用户及其口令，并且它指定他们有哪种全局（超级用户）权限。在 user 表启用的任何权限均是全局权限，并适用于所有数据库。例如，如果你启用了 DELETE 权限，在这里列出的用户可以从任何表中删除记录，所以在你这样做之前要认真考虑。

- db 表

db 表列出数据库，而用户有权限访问它们。在这里指定的权限适用于一个数据库中的所有表。

- host 表

host 表与 db 表结合使用在一个较好层次上控制特定主机对数据库的访问权限，这可能比单独使用 db 好些。这个表不受 GRANT 和 REVOKE 语句的影响，所以，你可能发觉你根本不是用它。

7.1.1.2 授权表 tables_priv 和 columns_priv 的结构和作用

表 7-2　授权表 tables_priv 和 columns_priv 的结构

授权表 tables_priv	授权表 columns_priv
作用域列	
Host	Host
Db	Db
User	User
Table_name	Table_name
	Column_name
权限列	
Table_priv	Column_priv
其他列	
Timestamp	Timestamp
Grantor	

 MySQL 没有 rows_priv 表，因为它不提供记录级权限，例如，你不能限制用户于表中

包含特定列值的行。如果你确实需要这种能力，你必须用应用编程来提供。如果你想执行建议的记录级锁定，你可用 GET_LOCK()函数做到。

授权表的内容有如下用途：

- tables_priv 表

  tables_priv 表指定表级权限，在这里指定的一个权限适用于一个表的所有列。

- columns_priv 表

  columns_priv 表指定列级权限。这里指定的权限适用于一个表的特定列。

tables_priv 和 columns_priv 表在 MySQL 3.22.11 版引进（与 GRANT 语句同时）。如果你有较早版本的 MySQL，你的 mysql 数据库将只有 user、db 和 host 表。如果你从老版本升级到 3.22.11 或更新，而没有 tables_priv 和 columns_priv 表，运行 mysql_fix_privileges_tables 脚本创建它们。

7.1.2 用户的权限

权限信息用 user、db、host、tables_priv 和 columns_priv 表被存储在 mysql 数据库中(即在名为 mysql 的数据库中)。在 MySQL 启动时和在 7.5 权限修改何时生效所说的情况时，服务器读入这些数据库表内容。

7.1.2.1 数据库和表的权限

下列权限运用于数据库和表上的操作。

- SELECT

允许你使用 SELECT 语句从表中检索数据。SELECT 语句只有在他们真正从一个表中检索行是才需要 select 权限，你可以执行某个 SELECT 语句，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 客户作为一个简单的计算器：

```
mysql> SELECT 1+1;
mysql> SELECT PI()*2;
```

- UPDATE

允许你修改表中的已有的记录。

- INSERT

允许在表中插入记录

- DELETE

允许你从表中删除现有记录。

- ALTER

允许你使用 ALTER TABLE 语句，这其实是一个简单的第一级权限，你必须由其他权限，这看你想对数据库实施什么操作。

- CREATE

允许你创建数据库和表，但不允许创建索引。

- DROP

允许你删除（抛弃）数据库和表，但不允许删除索引。


注意：如果你将 mysql 数据库的 drop 权限授予一个用户，该用户能抛弃存储了 MySQL 存取权限的数据库！

- INDEX

允许你创建并删除索引。

- REFERENCES

目前不用。

7.1.2.2 管理权限

下列权限运用于控制服务器或用户授权能力的操作的管理性操作。

- FILE

允许你告诉服务器读或写服务器主机上的文件。该权限不应该随便授予，它很危险，见“回避授权表风险”。服务器确实较谨慎地保持在一定范围内使用该权限。你只能读任何人都能读的文件。你正在写的文件必须不是现存的文件，这防止你迫使服务器重写重要文件，如/etc/passwd 或属于别人的数据库的数据目录。

如果你授权 FILE 权限，确保你不以 UNIX 的 root 用户运行服务器，因为 root 可在文件系统的任何地方创建新文件。如果你以一个非特权用户运行服务器，服务器只能在给用户能访问的目录中创建文件。

- GRANT

允许你将你自己的权限授予别人，包括 GRANT。

- PROCESS

允许你通过使用 SHOW PROCESS 语句或 mysqladmin process 命令查看服务器内正在运行的线程（进程）的信息。这个权限也允许你用 KILL 语句或 mysqladmin kill 命令杀死线程。

你总是能看到或杀死你自己的线程。PROCESS 权限赋予你对任何线程做这些事情的能力。

- RELOAD

允许你执行大量的服务器管理操作。你可以发出 FLUSH 语句，你也能指性 mysqladmin 的 reload、refresh、flush-hosts、flush-logs、flush-privileges 和 flush-tables 等命令。

- SHUTDOWN

允许你用 mysqladmin shutdown 关闭服务器。

在 user、db 和 host 表中，每一个权限以一个单独的列指定。这些列全部声明为一个 ENUM("N","Y")类型，所以每个权的缺省值是“N”。在 tables_priv 和 columns_priv 中的权限以一个 SET 表示，它允许权限用一个单个列以任何组合指定。这两个表比其他三个表更新，这就是为什么它们使用更有效的表示方式的原因。（有可能在未来，user、db 和 host 表也用一个 SET 类型表示。）

7.1.3 授权表列的内容

7.1.3.1 作用域列内容

一些范围列要求文字值，但它们大多数允许通配符或其他特殊值。

表 7-3　作用域列的类型

字段名	类型
Host	CHAR(60)
User	CHAR(16)
Password	CHAR(16)
Db	CHAR(64) (tables_priv 和 columns_priv 表为 CHAR(60))

- Host

一个 Host 列值可以是一个主机名或一个 IP 地址。值 localhost 意味着本地主机，但它只在你用一个 localhost 主机名时才匹配，而不是你在使用主机名时。假如你的本地主机名是 pit.snake.net 并且在 user 表中有对你的两条记录，一个有一个 Host 值或 localhost，而另一个有 pit.snake.net，有 localhost 的记录将只当你连接 localhost 时匹配，其他在只在连接 pit.snake.net 时才匹配。如果你想让客户能以两种方式连接，你需要在 user 表中有两条记录。

你也可以用通配符指定 Host 值。可以使用 SQL 的模式字符“%”和“_”并具有当你在一个查询中使用 LIKE 算符同样的含义（不允许 regex 算符）。 SQL 模式字符都能用于主机名和 IP 地址。如%wisc.edu 匹配任何 wisc.edu 域内的主机，而%.edu 匹配任何教育学院的主机。类似地，192.168.%匹配任何在 192.168 B 类子网的主机，而 192.168.3.%匹配任何在 192.168.3 C 类子网的主机。

%值匹配所有主机，并可用于允许一个用户从任何地方连接。一个空白的 Host 值等同于%。（例外：在 db 表中，一个空白 Host 值含义是“进一步检查 host 表”，该过程在“查询访问验证”中介绍。）

从 MySQL 3.23 起，你也可以指定带一个表明那些为用于网络地址的网络掩码的 IP 地址，如 192.168.128.0/17 指定一个 17 位网络地址并匹配其 IP 地址是 192.168.128 前 17 位的任何主机。

- User

用户名必须是文字的或空白。一个空白值匹配任何用户。%作为一个 User 值不意味着空白，相反它匹配一个字面上的%名字，这可能不是你想要的。

当一个到来的连接通过 user 表被验证而匹配的记录包含一个空白的 User 值，客户被认为是一个匿名用户。

- Password

口令值可以是空或非空，不允许用通配符。一个空口令不意味着匹配任何口令，它意味着用户必须不指定口令。

口令以一个加密过的值存储，不是一个字面上的文本。如果你在 Password 列中存储一个照字面上的口令，用户将不能连接！GRANT 语句和 mysqladmin password 命令为你自动加密口令，但是如果你用诸如 INSERT、REPLACE、UPDATE 或 SET PASSWORD 等命令，一定要用 PASSWORD("new_password")而不是简单的"new_password"来指定口令。

- Db

在 columns_priv 和 tables_priv 表中，Db 值必须是真正的数据库名（照字面上），不允许模式和空白。在 db 和 host 中，Db 值可以以字面意义指定或使用 SQL 模式字符'%'或'_'指定一个通配符。一个'%'或空白匹配任何数据库。

- Table_name，Column_name

这些列中的值必须是照字面意思的表或列名，不允许模式和空白。

某些范围列被服务器视为大小写敏感的，其余不是。这些原则总结在下表中。特别注意 Table_name 值总是被看作大小写敏感的，即使在查询中的表名的大小写敏感性对待视服务器运行的主机的文件系统而定（UNIX 下是大小写敏感，而 Windows 不是）。

某些作用域列被服务器视为大小写敏感的，其余不是。这些原则总结在下表中。特别注意 Table_name 值总是被看作大小写敏感的，即使在查询中的表名的大小写敏感性对待视服务器运行的主机的文件系统而定（UNIX 下是大小写敏感，而 Windows 不是）。

表 7-4　作用域列的大小写敏感性

列	大小写敏感性
Host	No
User	Yes
Password	Yes
Db	Yes
Table_name	Yes
Column_name	No

7.1.3.2 授权表 User、Db 和 Host 的权限列的内容

在 user、db 和 host 表中，所有权限字段被声明为 ENUM('N','Y')--每一个都可有值'N'或'Y'，并且缺省值是'N'.

7.1.3.3 授权表 tables_priv 和 columns_priv 的权限列的内容

在 tables_priv 和 columns_priv 表中，权限字段被声明为 SET 字段：

表 7-5　授权表 tables_priv 和 columns_priv 的权限列的类型

表名	字段名	可能的集合成员
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'

7.1.4 权限系统工作原理

7.1.4.1 权限系统工作的一般过程

MySQL 权限系统保证所有的用户可以严格地做他们假定被允许做的事情。当你连接一个 MySQL 服务器时， 你的身份由你从那连接的主机和你指定的用户名来决定，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，是因为有很小的原因假定一个给定的用户在因特网上属于同一个人。例如，用户从 whitehouse.gov 连接的 bill 不必和从 mosoft.com 连接 bill 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用户来处理它：你可以对从 whitehouse.gov 连接授与 bill 一个权限集，而为从 microsoft.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：服务器检查你是否允许连接。
- 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，如果你从数据库中一个表精选(select)行或从数据库抛弃一个表，服务器确定你对表有 select 权限或对数据库有 drop 权限。

服务器在存取控制的两个阶段使用在 mysql 的数据库中的 user、db 和 host 表

对存取控制的第二阶段(请求证实)，如果请求涉及表，服务器可以另外参考 tables_priv 和 columns_priv 表。

简单地说，服务器使用这样的授权表：

- user 表范围字段决定是否允许或拒绝到来的连接。对于允许的连接，权限字段指出用户的全局(超级用户)权限。
- db 和 host 表一起使用：

db 表范围字段决定用户能从哪个主机存取哪个数据库。权限字段决定允许哪个操作。

当你想要一个给定的 db 条目应用于若干主机时，host 表作为 db 表的扩展被使用。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 表的 Host 条目设为空值，然后将那些主机的每一个移入 host 表。这个机制详细描述在 7.1.4.3 存取控制, 阶段 2：请求证实。

tables_priv 和 columns_priv 表类似于 db 表，但是更精致：他们在表和列级应用而非在数据库级。

注意管理权限(reload, 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，只需要请教 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，但你读或谢在服务器主机上的文

件的的能力独立于你正在存取的数据库。

当 mysqld 服务器启动时，读取一次授权表内容。对授权表的更改生效在 7.5 权限修改何时生效。

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 表仅检查存取。它不检查表或列级权限。

7.1.4.2 存取控制，阶段 1：连接证实

当你试图联接一个 MySQL 服务器时，服务器基于你的身份和你是否能通过供应正确的口令验证身份来接受或拒绝连接。如果不是，服务器完全具结你的存取，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)范围字段执行。服务器只有在一个 user 表条目匹配你的主机名和用户名并且你提供了正确的口令时才接受连接。

在 user 表范围字段可以如下被指定：

一个 Host 值可以是主机名或一个 IP 数字，或'localhost'指出本地主机。

你可以在 Host 字段里使用通配符字符“%”和“_”。

一个 Host 值'%'匹配任何主机名，一个空白 Host 值等价于'%'。注意这些值匹配能创建一个连接到你的服务器的任何主机！

通配符字符在 User 字段中不允许，但是你能指定空白的值，它匹配任何名字。如果 user 表匹配到来的连接的条目有一个空白的用户名，用户被认为是匿名用户(没有名字的用户)，而非客户实际指定的名字。这意味着一个空白的用户名被用于在连接期间的进一步的存取检查(即，在阶段 2 期间)。

Password 字段可以是空白的。这不意味着匹配任何口令，它意味着用户必须不指定一个口令进行连接。

非空白 Password 值代表加密的口令。MySQL 不以任何人可以看的纯文本格式存储口令，相反，正在试图联接的一个用户提供的口令被加密(使用 PASSWORD()函数)，并且与存储了 user 表中的已经加密的版本比较。如果他们匹配，口令是正确的。

下面的例子显示出各种 user 表中 Host 和 User 条目的值的组合如何应用于到来的连接：

表 7-6 Host 和 User 条目的值的组合

Host 值	User 值	被条目匹配的连接
'host.domain.cn'	'Gwen'	Gwen, 从 host.domain.cn 连接
'host.domain.cn'	''	任何用户, 从 host.domain.cn 连接,
'%'	'Gwen'	Gwen, 从任何主机连接

'%'	''	任何用户，从任何主机连接
'%.loc.gov'	'Gwen'	Gwen，从在 loc.gov 域的任何主机连接
'x.y.%'	'Gwen'	Gwen，从 x.y.net、x.y.com,x.y.edu 等联接。（这或许无用）
'144.155.166.177'	'Gwen'	Gwen，从有 144.155.166.177 IP 地址的主机连接
'144.155.166.%'	'Gwen'	Gwen，从 144.155.166 C 类子网的任何主机连接

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 Host 列。只有一个 IP 数字能匹配 IP 通配符值。

一个到来的连接可以被在 user 表中的超过一个条目匹配。例如，一个由 Gwen 从 host.domain.cn 的连接匹配多个条目如上所述。如果超过一个匹配，服务器怎么选择使用哪个条目呢？服务器在启动时读入 user 表后通过排序来解决这个问题，然后当一个用户试图连接时，以排序的顺序浏览条目，第一个匹配的条目被使用。

MySQL 服务器按一种特定方式排序符授权表中的记录，然后通过按序浏览记录匹配到来的连接。找到的第一个匹配决定了被使用的记录。理解 MySQL 使用的排序顺序很重要，特别是对 user 表。

当服务器读取 user 表内容时，它根据在 Host 和 User 列中的值排序记录，Host 值起决定作用（相同的 Host 值排在一起，然后再根据 User 值排序）。然而，排序不是典序（按词排序），它只是部分是。要牢记的是字面上的词优先于模式。这意味着如果你正从 client.your.net 连接服务器而 Host 有 client.your.net 和%.your.net 两个值，则第一个先选。类似地，%.your.net 优先于%.net，然后是%。IP 地址的匹配也是这样的。

总之一句话，越具体越优先。

user 表排序工作如下，假定 user 表看起来像这样：

```
+----------+---------+-
| Host     | User    | ...
+----------+---------+-
| %        | root    | ...
| %        | jerry   | ...
| localhost | root   | ...
| localhost |        | ...
+----------+---------+-
```

当服务器在表中读取时，它以最特定的 Host 值为先的次序排列('%'在 Host 列里意味着“任何主机”并且是最不特定的)。有相同 Host 值的条目以最特定的 User 值为先的次序排列(一个空白 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 |         | ...
| %         | jerry   | ...
| %         | root    | ...
+-----------+---------+-
```

当一个连接被尝试时，服务器浏览排序的条目并使用找到的第一个匹配。对于由 jeffrey 从 localhost 的一个连接，在 Host 列的'localhost'条目首先匹配。那些有空白用户名的条目匹配连接的主机名和用户名。('%'/'jeffrey'条目也将匹配，但是它不是在表中的第一匹配。)

这是另外一个例子。假定 user 表看起来像这样：

```
+---------------+---------+-
| Host          | User    | ...
+---------------+---------+-
| %             | jerry   | ...
| host.domain.cn |         | ...
+---------------+---------+-
```

排序后的表看起来像这样：

```
+---------------+---------+-
| Host          | User    | ...
+---------------+---------+-
| host.domain.cn |         | ...
| %             | jerry   | ...
+---------------+---------+-
```

一个由 jerry 从 host.domain.cn 的连接被第一个条目匹配，而一个由 jerry 从 whitehouse.gov 的连接被第二个匹配。

普遍的误解是认为，对一个给定的用户名，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。这明显不是事实。先前的例子说明了这点，在那里一个由 jerry 从 host.domain.cn 的连接没被包含'jerry'作为 User 字段值的条目匹配，但是由没有用户名的题目匹配!

如果你有服务器连接的问题，打印出 user 表并且手工排序它看看第一个匹配在哪儿进行。

7.1.4.3 存取控制，阶段 2：请求证实

一旦你建立了一个连接，服务器进入阶段 2。对在此连接上进来的每个请求，服务器检查你是否有足够的权限来执行它，它基于你希望执行的操作类型。这正是在授权表中的权限字段发挥作用的地方。这些权限可以来子 user、db、host、tables_priv 或 columns_priv 表的任何一个。授权表用 GRANT 和 REVOKE 命令操作。（你可以发觉参考第七章 权限系统怎样工作很有帮助，它列出了在每个权限表中呈现的字段。）

user 表在一个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'并且仅在一个特定数据库的基础上授权， 使用 db 和 host 表。

db 和 host 表授予数据库特定的权限。在范围字段的值可以如下被指定：

通配符字符“%”和“_”可被用于两个表的 Host 和 Db 字段。

在 db 表的'%'Host 值意味着“任何主机”，在 db 表中一个空白 Host 值意味着“对进一步的信息咨询 host 表”。

在 host 表的一个'%'或空白 Host 值意味着“任何主机”。

在两个表中的一个'%'或空白 Db 值意味着“任何数据库”。

在两个表中的一个空白 User 值匹配匿名用户。

db 和 host 表在服务器启动时被读取和排序(同时它读 user 表)。db 表在 Host、Db 和 User 范围字段上排序，并且 host 表在 Host 和 Db 范围字段上排序。对于 user 表，排序首先放置最特定的值然后最后最不特定的值，并且当服务器寻找匹配入条目时，它使用它找到的第一个匹配。

tables_priv 和 columns_priv 表授予表和列特定的权限。在范围字段的值可以如下被指定：

通配符“%”和“_”可用在使用在两个表的 Host 字段。

在两个表中的一个'%'或空白 Host 意味着“任何主机”。

在两个表中的 Db、Table_name 和 Column_name 字段不能包含通配符或空白。

tables_priv 和 columns_priv 表在 Host、Db 和 User 字段上被排序。这类似于 db 表的排序，尽管因为只有 Host 字段可以包含通配符，但排序更简单。

请求证实进程在下面描述。（如果你熟悉存取检查的源代码，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；它只是不同于使解释更简单。）

对管理请求(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。如果条目许可请求的操作，存取被授权了，否则拒绝。例如，如果你想要执行 mysqladmin shutdown，但是你的 user 表条目没有为你授予 shutdown 权限，存取甚至不用检查 db 或 host 表就被拒绝。（因为他们不包含 Shutdown_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 表条目来检查用户的全局(超级用户)权限。如果条目允许请求的操作，存取被授权。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：

服务器在 db 表的 Host、Db 和 User 字段上查找一个匹配。 Host 和 User 对应连接用户的主机名和 MySQL 用户名。Db 字段对应用户想要存取的数据库。如果没有 Host 和 User 的条目，存取被拒绝。

如果 db 表中的条目有一个匹配而且它的 Host 字段不是空白的，该条目定义用户的数据库特定的权限。

如果匹配的 db 表的条目的 Host 字段是空白的，它表示 host 表列举主机应该被允许存取数据库的主机。在这种情况下，在 host 表中作进一步查找以发现 Host 和 Db 字段上的匹配。如果没有 host 表条目匹配，存取被拒绝。如果有匹配，用户数据库特定的权限以在 db 和 host 表的条目的权限，即在两个条目都是'Y'的权限的交集(而不是并集！)计算。（这样你可以授予在 db 表条目中的一般权限，然后用 host 表条目按一个主机一个主机为基础地有选择地限制它们。）

在确定了由 db 和 host 表条目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，存取被授权。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存取。

用布尔术语表示，前面关于一个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 条目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、表和列的特定权限。原因是一个请求可能要求超过一种类型的权限。例如，如果你执行一个 INSERT ... SELECT 语句，你就都要 insert 和 select 权限。你的权限必须如此以便 user 表条目授予一个权限而 db 表条目授予另一个。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；两个条目授予的权限必须组合起来。

host 表能被用来维护一个“安全”服务器列表。在 TcX，host 表包含一个在本地的网络上所有的机器的表，这些被授予所有的权限。

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，它位于你不认为是安全的一个公共区域，你可以用下列的 host 表条目子允许除了那台机器外的网络上所有主机的存取：

```
+------------------+---+-
| Host             | Db | ...
```

```
+------------------+---+-
| public.your.domain | %  | ...（所有权限设为 'N'）
| %.your.domain      | %  | ...（所有权限设为 'Y'）
+------------------+---+-
```

当然，你应该总是测试你在授权表中的条目(例如，使用 mysqlaccess)让你确保你的存取权限实际上以你认为的方式被设置。

7.1.5 总结

本节详细叙述了 MySQL 权限系统的原理，包括 MySQL 存取证实的完整过程。MySQL 通过授权表实现了一个不同于 SQL92 标准的权限系统，本节所述的授权表的结构和内容对于本章以后的理解十分重要，因为涉及到直接修改授权表的知识。尽管你可以使用 GRANT 和 REVOKE 语句完成大部分的授权，但是只有了解授权表的结构和内容，你才能更好的控制系统，使你的系统更为安全。在很多时候，直接修改授权表是唯一的手段。

另外一个重要的知识是授权表中的 user 和 host 字段。了解 MySQL 服务器用户匹配的顺序以及身份验证的过程，有利于你配置一个安全的系统。

7.2 设置用户与并授权

你可以有 2 个不同的方法增加用户：通过使用 GRANT 语句或通过直接操作 MySQL 授权表。比较好的方法是使用 GRANT 语句，因为他们是更简明并且好像错误少些。

7.2.1 使用 SHOW GRANTS 语句显示用户的授权

你可以直接查看授权表，也可以使用 SHOW GRANTS 语句查看某个用户的授权，这种情况下使用 SHOW GRANTS 语句显然要方便一些。

语法：SHOW GRANTS FOR user_name

为了容纳对任意主机的用户授予的权利，MySQL 支持以 user@host 格式指定 user_name 值。

例如，下面的语句显示一个用户 admin 的权限：

mysql>SHOW GRANTS FOR admin@localhost;

其结果为创建该用户的 GRNAT 授权语句：

GRANT RELOAD, SHUTDOWN, PROCESS ON *.* TO 'admin'@'localhost' IDENTIFIED BY PASSWORD '28e89ebc62d6e19a'

密码是加密后的形式。

下小节会介绍如何使用 GRANT 语句授权。

7.2.2 使用 GRANT 语句创建用户并授权

7.2.2.1 GRANT 语句的语法

GRANT priv_type (columns)

ON what
TO user IDENTIFIED BY “password”
WITH GRANT OPTION

要使用该语句，需要填写以下部分：

- priv_type 分配给用户的权限。

priv_type 可以指定下列的任何一个：

ALL PRIVILEGES	FILE	RELOAD
ALTER	INDEX	SELECT
CREATE	INSERT	SHUTDOWN
DELETE	PROCESS	UPDATE
DROP	REFERENCES	USAGE

ALL 是 ALL PRIVILEGES 的一个同义词，REFERENCES 还没被实现，USAGE 当前是“没有权限”的一个同义词。它能用在你想要创建一个没有权限用户的时候。

对于表，你能指定的唯一 priv_type 值是 SELECT、INSERT、UPDATE、DELETE、CREATE、DROP、GRANT、INDEX 和 ALTER。

对于列，你能指定的唯一 priv_type 值是(即，当你使用一个 column_list 子句时)是 SELECT、INSERT 和 UPDATE。

- columns 权限适用的列。

这是可选的，只来设置列专有的权限。如果命名多于一个列，则用逗号分开。

- what 权限应用的级别

GRANT 允许系统主管在 4 个权限级别上授权 MySQL 用户的权利：

全局级别

全局权限作用于一个给定服务器上的所有数据库。这些权限存储在 mysql.user 表中。你能通过使用 ON *.*语法设置全局权限

数据库级别

数据库权限作用于一个给定数据库的所有表。这些权限存储在 mysql.db 和 mysql.host 表中。 你能通过使用 ON db_name.*语法设置数据库权限。如果你指定 ON *并且你有一个当前数据库，你将为该数据库设置权限。(警告：如果你指定 ON *而你没有一个当前数据库，你将影响全局权限！)

表级别

表权限作用于一个给定表的所有列。这些权限存储在 mysql.tables_priv 表中。你能透过 ON tbl_name，为具体的表名设置权限。

列级别

列权限作用于在一个给定表的单个列。这些权限存储在 mysql.columns_priv 表中。你可以通过指定一个 columns 子句将权限授予特定的列,同时要在 ON 子句中指定具体的表。

对与一个表或列的权限是由 4 个权限级别的逻辑或形成的。例如，如果 mysql.user 表指定一个用户有一个全局 select 权限，它不能被数据库、表或列的一个条目否认。

对于一个列的权限能如下计算：

global privileges

OR (database privileges AND host privileges)

OR table privileges

OR column privileges

在大多数情况下，你只授予用户一个权限级别上的权限，因此现实通常不象上面所说的那样复杂。

- user 使用权限的用户。

为了容纳对任意主机的用户授予的权利，MySQL 支持以 user@host 格式指定 user_name 值。如果你想要指定一个特殊字符的一个 user 字符串(例如“-”)，或一个包含特殊字符或通配符的 host 字符串(例如“%”)，你可以用括号括起能用户或主机名字 (例如，'test-user'@'test-hostname')。

你能在主机名中指定通配符。例如，user@"%.loc.gov"适用于在 loc.gov 域中任何主机的 user，并且 user@"144.155.166.%"适用于在 144.155.166 类 C 子网中任何主机的 user。

简单形式的 user 是 user@"%"的一个同义词。注意：如果你允许匿名用户连接 MySQL 服务器(它是缺省的)，你也应该增加所有本地用户如 user@localhost，因为否则，当用户试图从本地机器上登录到 MySQL 服务器时，对于 mysql.user 表中的本地主机的匿名用户条目将被使用！匿名用户通过插入有 User="的条目到 mysql.user 表中来定义。通过执行这个查询，你可以检验它是否作用于你：

mysql> SELECT Host,User FROM mysql.user WHERE User='';

- password 分配给该用户的口令。这也是可选的。

在 MySQL 3.22.12 或以后，如果创建一个新用户或如果你有全局授予权限，用户的口令将被设置为由 IDENTIFIED BY 子句指定的口令，如果给出一个。如果用户已经有了一个口令，它被一个新的代替。

警告：如果你创造一个新用户但是不指定一个 IDENTIFIED BY 子句，用户没有口令。这是不安全的。

- WITH GRANT OPTION 子句是可选的。

WITH GRANT OPTION 子句给与用户有授予其他用户在指定的权限水平上的任何权限的能力。你应该谨慎对待你授予他 grant 权限的用户，因为具有不同权限的两个用户也许能合并权限！

7.2.2.2 创建用户并授权的实例

创建一个具有超级用户权利的用户：

```
mysql>GRANT ALL ON *.* TO anyname@localhost IDENTIFIED BY "passwd"
    ->WITH GRANT OPTION
```

该语句将在 user 表中为 anyname@localhost 创建一个记录，打开所有权限。

数据库级权限用一个 ON db_name.*子句而不是 ON *.*进行授权：

```
mysql>GRANT ALL ON sample.* TO boris@localhost IDENTIFIED BY "ruby"
```

这些权限不是全局的，所以它们不存储在 user 表中，我们仍然需要在 user 表中创建一条记录（使得用户能连接），但我们也需要创建一个 db 表记录记录数据库集的权限。

7.2.3 直接修改授权表创建用户并授权

如果你还记得前面的介绍，你应该能即使不用GRANT语句也能做GRANT做的事情。记住在你直接修改授权表时，你将通知服务器重载授权表，否则他不知道你的改变。你可以执行一个mysqladmin flush-privileges或mysqladmin reload命令和FLUSH PRIVILEGES语句强迫一个重载。如果你忘记做这个，你会疑惑为什么服务器不做你想做的事情。

下列GRANT语句创建一个拥有所有权的超级用户。包括授权给别人的能力：

```
GRANT ALL ON *.* TO anyname@localhost IDENTIFIED BY "passwd"
    WITH GRANT OPTION
```

该语句将在user表中为anyname@localhost创建一个记录，打开所有权限，因为这里是超级用户（全局）权限存储的地方，要用INSERT语句做同样的事情，语句是：

```
INSERT INTO user  VALUES("localhost","anyname",PASSWORD("passwd"),
    "Y","Y","Y","Y","Y","Y","Y","Y","Y","Y","Y","Y","Y","Y")
```

你可能发现它不工作，这要看你的MySQL版本。授权表的结构已经改变而且你在你的user表可能没有14个权限列。用SHOW COLUMNS找出你的授权表包含的每个权限列，相应地调整你的INSERT语句。 下列GRANT语句也创建一个拥有超级用户身份的用户，但是只有一个单个的权限：

```
GRANT RELOAD ON *.* TO flush@localhost IDENTIFIED BY "flushpass"
```

本例的INSERT语句比前一个简单，它很容易列出列名并只指定一个权限列。所有其它列将设置为缺省的"N"：

```
INSERT INTO user (Host,Password,Reload)
VALUES("localhost","flush",PASSWORD("flushpass"),"Y")
```

数据库级权限用一个ON db_name.*子句而不是ON *.*进行授权：

```
GRANT ALL ON sample.* TO boris@localhost IDENTIFIED BY "ruby"
```

这些权限不是全局的，所以它们不存储在user表中，我们仍然需要在user表中创建一条记录（使得用户能连接），但我们也需要创建一个db表记录记录数据库集权限：

```
mysql> INSERT INTO user (Host,User,Password)
    -> VALUES("localhost","boris",PASSWORD("ruby")) ;
mysql> INSERT INTO db VALUES
    -> ("localhost","sample_db","boris","Y","Y","Y","Y","Y","Y","N","Y","Y","Y");
```

"N"列是为GRANT权限；对末尾的一个数据库级具有WITH GRANT OPTION的GRANT语句，你要设置该列为"Y"。

要设置表级或列级权限，你对tables_priv或columns_priv使用INSERT语句。当然，如果你没有GRANT语句，你将没有这些表，因为它们在MySQL中同时出现。如果你确实有这些表并且为了某些原因想要手工操作它们，要知道你不能用单独的列启用权限。

你设置tables_priv.Table_priv或columns_priv.Column_priv列来设置包含你想启用的权限值。例如，要对一个表启用SELECT和INSERT权限，你要在相关的tables_priv的记录中设置Table_priv为"Select,Insert"。

7.2.4 总结

本届介绍了如何建立用户，如何为用户分配权限。GRANT 语句虽然能够为用户分配权限，使用比较方便，但是从安全角度，从增加对系统了解角度，你应该尽量使用直接修改授权表的方法。这样可以更精确的控制授权，能够对授权表完全了解，避免因为 GRANT 语句的错误，是 MySQL 服务器的安全性降低。

7.3 撤销用户与授权

撤销用户和授权的有两种方法，一是使用 REVOKE 语句，二是直接修改授权表。

7.3.1 使用 REVOKE 语句撤销授权

为了收回某个用户的权限，可使用 REVOKE 语句。除了要用 FROM 替换 TO 并且没有 IDENTIFIED BY 或 WITH GRANT OPTION 子句外

语法：REVOKE privileges (columns) ON what FROM user;

user 部分必须页你想要取消其权限的用户的原始 GRANT 语句的 user 部分相匹配。

Privileges 部分不需要匹配，你可用 GRANT 语句授权，然后用 REVOKE 语句取消启动的一部分。

REVOKE 语句只删除权限，不删除用户。用户的项仍然保留在 user 表中，即使你去笑了改用户的所有权限也是如此。这意味着该用户仍然可以连接到服务其上。要删除整个用户，必须用 DELETE 语句将该用户的记录从 user 表中直接删除。要想删除整个用户，必须直接将该用户的记录从 user 表中直接删除。

例如，如果你为一个数据库授权，如果需要在 mysql.db 表中创建一个条目。

mysql>GRANT ALL ON sample.* TO kite@localhost IDENTIFIED BY "ruby";

当所有为数据库的授权用 REVOKE 删除时，这个条目被删除。

mysql>REVOKE ALL ON sample.* FROM kite@localhost;

但是，boris@localhost 用户的条目仍旧留在 user 表中，你可以查看：

mysql>SELECT * FROM mysql.user;

```
+-----------+------+
| Host      | User |
+-----------+------+
| localhost | root |
| localhost | kite |
| localhost | admin |
| %         | root |
+-----------+------+
```

7.3.2 直接修改授权表撤销用户或授权

显而易见，可以直接在授权表中给用户撤销授权，有时，这是唯一的方法，如撤销一个用户。记住在你直接修改授权表时，你将通知服务器重载授权表，否则他不知道你的改

变。你可以执行一个 mysqladmin flush-privileges 或 mysqladmin reload 命令和 FLUSH PRIVILEGES 语句强迫一个重载。如果你忘记做这个，你会疑惑为什么服务器不做你想做的事情。

在上节最后一个例子中，可以这样撤销用户：

```
mysql>DELETE FROM mysql.user
    ->WHERE User=" kite" AND Host=" localhost";
mysql>FLUSH PRIVILEGES;
mysql>SELECT * FROM mysql.user;
+-----------+------+
| Host      | User |
+-----------+------+
| localhost | root |
| localhost | admin |
| %         | root |
+-----------+------+
```

DELETE 语句删除该用户的项，FLUSH 语句告诉服务器重新假造授权表（但是用 GRANT 或 REVOKE 语句，而不是直接修改授权表时，这些表将自动重新加载）。

而 REVOKE ALL ON sample.* FROM boris@localhost 对应的，可以这样直接修改授权表：

```
mysql>DELETE FROM mysql.db
    ->WHERE User=" boris" AND Host=" localhost" AND db="sample";
mysql>FLUSH PRIVILEGES;
```

7.3.3 总结

本节介绍了如何撤销用户和授权，使用 REVOKE 语句只能撤销授权，不能撤销用户，而直接修改授权表既可以撤销授权，也可以撤销用户。使用时，注意直接修改授权表一般需要多个步骤，操作多个表的内容。

7.4 设置密码

由 MySQL 使用用户名和口令的方法与 Unix 或 Windows 使用的方式有很多不同之处：

- MySQL 使用于认证目的的用户名，与 Unix 用户名(登录名字)或 Windows 用户名无关。缺省地，大多数 MySQL 客户尝试使用当前 Unix 用户名作为 MySQL 用户名登录，但是这仅仅为了方便。客户程序允许用-u 或--user 选项指定一个不同的名字，这意味着无论如何你不能使得一个数据库更安全，除非所有的 MySQL 用户名都有口令。任何人可以试图用任何名字连接服务器，而且如果他们指定了没有口令的任何名字，他们将成功。
- MySQL 用户名最长可以是 16 各字符；典型地，Unix 用户名限制为 8 个字符。

- MySQL 口令与 Unix 口令没关系。在你使用登录到一台 Unix 机器口令和你使用在那台机器上存取一个数据库的口令之间没有必要有关联。
- MySQL 加密口令使用了一个 Unix 登录期间所用的不同算法。

本节将介绍如何为 MySQL 数据库系统的用户修改密码。

7.4.1 使用 myadmin 实用程序

使用 mysqladmin 实用程序修改密码的命令行是：

shell>mysqladmin -u user -p password "newpassword"

运行这个命令，在提示输入密码时，数据就密码，则用户 user 的密码就被改为 newpassword。

如果，原来的用户没有密码，则不比指定-p 选项。例如，初始化授权表之后，root 用户的密码就是空的，你可以这样为 root 用户设立密码：

shell>mysqladmin -u root password "newpassword"

7.4.2 使用语句 SET PASSWORD

使用 mysqladmin 为用户修改密码有一个明显的缺点，就是必须知道用户原来的密码，如果是为了给遗忘了密码的用户重设密码就无能为力了。一个专门用于修改密码的 SQL 语句为 SET PASSWORD：

- SET PASSWORD = PASSWORD('some password')

设置当前用户的口令。任何非匿名的用户能改变他自己的口令！

连接到服务器后，你可以这样改变自己的密码：

mysql> SET PASSWORD = PASSWORD('another pass');

- SET PASSWORD FOR user = PASSWORD('some password')

设置当前服务器主机上的一个特定用户的口令。只有具备存取 mysql 数据库的用户可以这样做。用户应该以 user@hostname 格式给出，这里 user 和 hostname 完全与他们列在 mysql.user 表条目的 User 和 Host 列一样。例如，如果你有一个条目其 User 和 Host 字段是'bob'和'%.loc.gov'，你将写成：

mysql> SET PASSWORD FOR bob@"%.loc.gov" = PASSWORD("newpass");

7.4.3 直接修改授权表

另一种修改，密码的方法是直接修改授权表 user。只有具备存取 mysql 数据库的用户可以这样做。

例如，如果你有一个条目其 User 和 Host 字段是'bob'和'%.loc.gov'，你将写成：

```
mysql> UPDATE mysql.user SET password=PASSWORD("newpass") where user="bob' AND host="%.loc.gov";
```

mysql>FLUSH PRIVILEGES;

7.4.4 重新设置一个遗忘的 root 口令

如果你遗忘了 root 用户的口令，那么将会是一件非常麻烦的事。除非你有其它有特权

的用户，否则很多操作都无法完成，例如，关闭数据库等等。

你应当选用--without-grant-tables 选项启动 mysqld 服务，你可以在这时更改授权表的相关内容，也可以用 mysqlaccess 检查你的授权是否到位。

例如，如果你忘记了你的 MYSQL 的 root 口令的话，你可以通过下面的过程恢复。

1、关闭 MySQL 服务器

向 mysqld server 发送 kill 命令关掉 mysqld server(不是 kill -9),存放进程ID的文件通常在 MYSQL 的数据库所在的目录中。

```
kill `cat /mysql-data-directory/hostname.pid`
```

你必须是 UNIX 的 root 用户或者是你所运行的 SERVER 上的同等用户，才能执行这个操作。

如果在 windows 平台上，也可以停止进程。如果是 NT 还可以用 net stop mysql 命令关闭数据库。

2、使用'--skip-grant-tables' 参数来启动 mysqld。

Unix 平台：

```
$su mysql
$safe_mysqld --skip-grant-tables &
```

Windows 平台：

```
C:\mysql\bin>mysqld --skip-grant-tables
```

以上语句，假定都位于正确的目录。

3、连接到服务器，修改口令

使用'mysql -h hostname mysql'命令登录到 mysqld server ，用 grant 命令改变口令：

```
mysql>GRANT ALL ON *.* TO root@localhost INDENTIFIED BY 'new password'
    -> WITH GRANT OPTION;
mysql>GRANT ALL ON *.* TO root@% INDENTIFIED BY 'new password'
    -> WITH GRANT OPTION;
```

(如果存在一个能从任意地址登录的 root 用户，初始化授权表后，生成该用户，为了安全，你可能已经删除该用户)。

其实也可以直接修改授权表：

```
mysql> use mysql;
mysql> update user set password =password('yourpass') where user='root';
```

你可能使用工具 mysqladmin 修改密码：

```
shell> mysqladmin -h hostname -u root password 'new password
```

但是它修改的密码语服务器匹配的用户有关。如果，你从服务器主机连接，那么服务器匹配的是 root@localhost，修改该用户密码，否则一般修改 root@%密码，除非你有其它 root 用户存在。

4. 载入权限表：

```
shell> mysqladmin -h hostname flush-privileges
```

或者使用 SQL 命令`FLUSH PRIVILEGES'。

当然，在这里，你也可以重启 mysqld。

7.4.5 总结

本节介绍了如何修改一个用户的密码，你可以使用三种方法，GRANT 语句、SET PASSWORD 语句、直接修改授权表以及使用管理工具 mysqladmin。

一个重要的应用就是如何在遗忘 root 用户密码的时候修改密码，使用的方法是启动 MySQL 服务器时忽略加载授权表。

7.5 权限修改何时生效

7.5.1 服务器重新启动的情况

当 mysqld 启动时，所有的授权表内容被读进存储器并且从那时开始生效。

7.5.2 被服务器立即应用的情况

用 GRANT、REVOKE 或 SET PASSWORD 对授权表施行的修改会立即被服务器注意到。

7.5.3 直接修改授权表的情况

如果你手工地修改授权表(使用 INSERT、UPDATE 等等)，你应该执行一个 FLUSH PRIVILEGES 语句或运行 mysqladmin flush-privileges 告诉服务器再装载授权表，否则你的改变将不生效，除非你重启服务器。

7.5.4 对现有客户连接的影响情况

当服务器注意到授权表被改变了时，现存的客户连接有如下影响：

- 表和列权限在客户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。
- 全局权限的改变和口令改变在下一次客户连接时生效。

7.5.5 总结

本小节总结了权限修改后以及服务器和客户机的权限生效的几种情况，读者要十分留意这些情况，这对你管理数据库系统有重要影响。当你为了权限疑惑时，不妨看一下本小节。

7.6 授权原则

无论怎么小心都难免在给用户授权时留有漏洞，希望下面的内容能给你一些帮助，你一般应该遵守这些规则。

7.6.1 只有 root 用户拥有授权表的改写权

不要把授权表的改写权授予除 root 用户之外的其它用户（当然，如果你可以用另一个用户代替 root 用户进行管理，以增加安全性）。因为这样，用户可以通过改写授权表而推翻现有的权限。产生安全漏洞。

一般情况下，你可能不会犯这个错误，但是在安装新的分发, 初始授权表之后。这个漏洞是存在的，如果你不了解这时授权表的内容你可能会犯错误。

在 Unix（Linux）上，在按照手册的指令安装好 MySQL 后，你必须运行 mysql_install_db 脚本建立包含授权表的 mysql 数据库和初始权限。在 Windows 上，运行分发中的 Setup 程序初始化数据目录和 mysql 数据库。假定服务器也在运行。

当你第一次在机器上安装 MySQL 时，mysql 数据库中的授权表是这样初始化的：

- 你可以从本地主机（localhost）上以 root 连接而不指定口令。root 用户拥有所有权限（包括管理权限）并可做任何事情。（顺便说明，MySQL 超级用户与 Unix 超级用户有相同的名字，他们彼此毫无关系。）
- 匿名访问被授予用户可从本地连接名为 test 和任何名字以 test_开始的数据库。匿名用户可对数据库做任何事情，但无管理权限。

一般地，建议你删除匿名用户记录：

```
mysql> DELETE FROM user WHERE User="";
```

更进一步，同时删除其他授权表中的任何匿名用户，有 User 列的表有 db、tables_priv 和 columns_priv。

另外要给 root 用户设置密码。

7.6.2 关于用户、口令及主机的设置

- 对所有 MySQL 用户使用口令。

记住，如果 other_user 没有口令，任何人能简单地用 mysql -u other_user db_name 作为任何其它的人登录。对客户机/服务器应用程序，客户可以指定任何用户名是常见的做法。在你运行它以前，你可以通过编辑 mysql_install_db 脚本改变所有用户的口令，或仅仅 MySQL root 的口令，象这样：

```
shell> mysql -u root mysql
mysql> UPDATE user SET Password=PASSWORD('new_password')
    -> WHERE user='root';
mysql> FLUSH PRIVILEGES;
```

- 删除匿名用户

匿名用户的存在不仅不仅容易引起存取拒绝错误，更会产生严重的安全漏洞，安装授权表后，自动安装匿名用户。缺省时你可以用任何用户名连接，不需要密码，并且具有修改授权表权限。

你可以这样删除匿名用户：

```
shell>mysql –u root –p mysql
mysql>delete from user where User="";
```

- 留意使用通配符的主机名，尽量缩小主机名的范围，适合用户的主机就足够了，不要让用户不使用的主机留在授权表里。
- 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。原则上讲，--secure 选项对 mysqld 应该使主机名更安全。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！

7.6.3 授予用户合适的权限

- 授权用户足够使用的权限，不要赋予额外的权限。

例如，对于用户只需要检索数据表的需求，赋予 SELECT 权限即可，不可赋予 UPDATE、INSERT 等写权限，不要怕被说成时吝啬鬼。

- 可能会产生安全漏洞的权限

grant 权限允许用户放弃他们的权限给其他用户。2 个有不同的权限并有 grant 权限的用户可以合并权限。

alter 权限可以用于通过重新命名表来推翻权限系统。 因为 ALTER 权限可能以你没有设想的任何方法被使用。例如，一个用户 user1 能访问 table1，但不能访问 table2。但是如果用户 user1 带有 ALTER 权限可能通过使用 ALTER TABLE 将 table2 重命名为 table1 来打乱你的设想。

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 。

- 可能会产生严重安全漏洞的权限

不要把 PROCESS 权限给所有用户。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 process 权限的用户保留一个额外的连接，以便一个 MySQL root 用户能登录并检查，即使所有的正常连接在使用。

不要把 FILE 权限给所有的用户。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！为了使这更安全一些，用 SELECT ... INTO OUTFILE 生成的所有文件对每个人是可读的，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 用户可存取的文件。这可能被滥用，因为不仅有该服务器主机帐号的用户可以读取它们，而且有 FILE 权限的任何客户机也可以通过网络读取它们。你的数据库目录和系统的各种文件可能成为全球范围共享的文件！例如，通过使用 LOAD DATA 装载“/etc/passwd”进一个数据库表，然后它能用 SELECT 被读入。

下面的过程说明如何进行此项操作：

1、 建具有 LONGBLOB 列的表：

```
mysql> USE test;
mysql> CREATE TABLE temp (b LONGBLOB);
```

2、用此表读取你要窃取的文件的内容：

```
mysql>LOAD DATA INFILE “/etc/passwd” INTO TABLE temp
    ->FIELDS ESCAPED BY  “” LINES TERMINATED BY  “”;
mysql>SELECT * FROM temp;
```

3、可以这样窃取你的数据表 data：

```
mysql>LOAD DATA INFILE “./other_db/data.frm” INTO TABLE temp
    ->FIELDS ESCAPED BY  “” LINES TERMINATED BY  “”;
```

```
mysql>SELECT * FROM temp INTO OUTFILE "./another_db/data.frm"
    ->FIELDS ESCAPED BY  ""  LINES TERMINATED BY  "";
mysql>DELETE FROM temp;
mysql>LOAD DATA INFILE "./other_db/data.MYD" INTO TABLE temp
    ->FIELDS ESCAPED BY  ""  LINES TERMINATED BY  "";
mysql>SELECT * FROM temp INTO OUTFILE "./another_db/data.MYD"
    ->FIELDS ESCAPED BY  ""  LINES TERMINATED BY  "";
mysql>DELETE FROM temp;
mysql>LOAD DATA INFILE "./other_db/data.MYI" INTO TABLE temp
    ->FIELDS ESCAPED BY  ""  LINES TERMINATED BY  "";
mysql>SELECT * FROM temp INTO OUTFILE "./another_db/data.MYI"
    ->FIELDS ESCAPED BY  ""  LINES TERMINATED BY  "";
mysql>DELETE FROM temp;
```

然后用户就拥有的一个新表 another.data，可以对它进行完全访问。

7.6.4 MySQL 权限系统无法完成的任务

有一些事情你不能用 MySQL 权限系统做到：

- 你不能明显地指定一个给定用户应该被拒绝存取。即，你不能明显地匹配一个用户并且然后拒绝连接。
- 你不能指定一个用户有权创建立或抛弃一个数据库中的表，也不能创建或抛弃数据库本身

7.6.5 总结

本节讲述了如何为用户分配合适的权限，几个重要的原则就是给用户分配仅够使用的最小权限，尽量不在影响整个数据库的 user 表中分配权限。

有些权限是有危险的，例如 FILE、GRANT、PROCESS，管理员要慎重使用。

7.7 MySQL 的其它安全问题

数据库系统的安全性包括很多方面。由于很多情况下，数据库服务器容许客户机从网络上连接，因此客户机连接的安全对 MySQL 数据库安全有很重要的影响。

7.7.1 不在客户机的命令行上提供密码

使用 mysql、mysqladmin 等客户机用一个用户身份与 MySQL 服务器连接时，需要为连接提供密码。

1 可以在命令行上提供密码

```
shell>mysql –u root –pmypass
```

注意，-p 选项与密码之间不可有空格，否则会提示你输入密码，并报错。

你也可以使用长格式

shell>mysql –user=root –password=mypass

现在你可以考察这样做的后果：

在 Unix 上，$ps –aux | grep mysql

在 win9x 上，你可以按住 Ctrl＋Alt＋Del 键，NT 上你可以打开任务管理器。

你发现了什么,你发现密码清清楚楚的显示在你的面前。所以，你无论何时也不要这么做。

所以你需要让客户机提示你的密码：

shell>mysql –u root –p

你也可以使用选项文件提供密码，但是注意为了安全，不能在选项文件中存储密码。你可以只提供 password 选项，让客户机提示你输入密码。

7.7.2 使用 SSH 加密客户机连接

这是一个关于怎样用 SSH 得到一个安全的连接远程 MySQL 服务器的注意事项(David Carlson)。

在你的 windows 机器上安装 SSH 客户 - 我使用了一个来自 http://www.doc.ic.ac.uk/~ci2/ssh/ 的免费 SSH 客户。其他有用的链接：http://www.npaci.edu/Security/npaci_security_software.html 和 http://www.npaci.edu/Security/samples/ssh32_windows/index.html.

启动 SSH。设置主机名字 = 你的 MySql 服务器名或 IP 地址，设置 userid=你的用户名登录到你的服务器。

点击“local forwords”。设定 local port: 3306, host: localhost, remote port: 3306

保存一切，否则下次你将必须再做一遍。

用 SSH 登录到你的服务器。

启动一些 ODBC 应用程序(例如 Access）。

创造一个新文件并且用 ODBC 驱动程序链接到 mySQL，就像你通常做的一样，除了对服务器用用户“localhost”。

搞定。它对一个直接的因特网连接工作得很好。

7.7.3 不要使用 Unix 的 root 用户运行 MySQL 守护进程

不要作为 Unix 的 root 用户运行 MySQL 守护进程。mysqld 能以任何用户运行，你也可以创造一个新的 Unix 用户 mysql 使一切更安全。如果你作为其它 Unix 用户运行 mysqld，你不需要改变在 user 表中的 root 用户名，因为 MySQL 用户名与 Unix 用户名没关系。

你可以作为其它 Unix 用户编辑 mysql.server 启动脚本 mysqld。或者使用选项文件。有关如何用非 root 用户启动 MySQL 服务器的细节，请见第二章。

7.7.4 数据库目录的安全

MySQL 服务器提供了一个通过 mysql 数据库中的授权表，实现了一个十分灵活的权限系统，保证了从网络访问数据的安全性。但是，如果服务器主机上的其它用户拥有对服务器目录的直接访问权，那么你的服务器的数据仍旧是不安全的。

一般你可能用一个非特权的 Unix 用户执行守护程序。检查那个运行 mysqld 的 Unix 用户是唯一的在数据库目录下有读/写权限的用户。

7.7.4.1 可能的安全漏洞

显然，你不会让服务器主机上的其它用户拥有对数据库目录文件的写访问权，但是，仅仅是读访问权也非常危险。

由于象 GRANT 和 SET PASSWORD 这样的查询都被记录到日志中去了，常规和更新日志文件包含了有关密码的敏感查询文本。如果一个攻具有对这些日志的读访问权，那么他只要在日志文件中查找 GRANT 或 PASSWORD 这样的敏感单词，就很容易找到密码的明文。

对于表文件的读访问也是很危险的，窃取文件并使 MySQL 以及以纯文本的形式显示表的内容是微不足道的事。可以按下列步骤进行：

1、 安装一个新的 MySQL 分发，可以是在另一台主机上，也可以就在当前的服务器主机上安装，使用与正式服务器不同的端口、套接字和数据文件。

2、 将窃取的表的相应文件拷贝到新服务数据库目录中的 test 目录下

3、 然后就可以启动作案服务器，可以随意访问所窃取表的内容。

7.7.4.2 在 Unix 设置合适的数据库目录权限

如果要消除这些安全漏洞，就要安排数据库目录及其中所有文件和目录的所有权，是的只有启动服务器的专用帐户才可以访问它们。操作步骤如下：

1、 切换到 root 用户

```
$su
```

2、 设置数据库目录及其中所有文件目录的所有权为运行该服务器的帐号所有，在本书中一直将这个帐号假定为 mysql，把所有的组设为 root 组

```
%chown –R mysql:root DATADIR
```

3、 修改数据库目录及其中所有文件目录的权限为只有所有者可以读写

```
%chmod –R go-rwx DATADIR
```

7.7.4.3 在 NT 系统中设置合适的数据库目录权限

在 NT 系统中的数据库目录的安全性可能比较简单：

读者可能想到，把所有目录文件改为只有某个帐户 administrator 例如可读写。但是，这样会有一个问题，就是这能在 administrator 帐户中用手动启动独立的服务器，如果让 mysql 系统服务自动启动的方法也不可行，解决方法是让数据库目录也可被 administrators 组用户读写，这样 MySQL 服务器就可以用系统服务的方法自动启动了，也可以在任何帐户中用 net start mysql 启动。

另外一个问题是，如果你在非 administrators 组用户中或者从网络都无法建立数据库连接，因为没有数据库目录的读的权利，如果要正常使用，还需要 写的权利。解决方法是让 SYSTEM 组用户能够队数据库目录读写。

由于许可证和费用的原因，通常建议你在 Linux 服务器上使用 MySQL，在 Windows 平台之用来测试或者数据录入工作。但是，如果你要在 Windows 上使用的话，可以注意本小节的一些内容。

7.7.5 影响安全的 mysqld 选项

下列 mysqld 选项影响安全：

--secure

由 gethostbyname()系统调用返回的 IP 数字被检查，确保他们解析回到原来的主机名。这对某些外人通过模仿其它主机获得存取权限变得更难。这个选项也增加一些聪明的主机名检查。在 MySQL3.21 里，选择缺省是关掉的，因为它有时它花很长时间执行反向解析。MySQL 3.22 缓存主机名并缺省地启用了这个选项。

--skip-grant-tables

这个选项导致服务器根本不使用权限系统。这给每个人以完全存取所有的数据库的权力！(通过执行 mysqladmin reload，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 数字或 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。这个选项对使用 MIT-pthreads 的系统是不合适的，因为 MIT-pthreads 包不支持 Unix 套接字。

7.7.6 总结

本节介绍了其它有关 MySQL 服务器安全的内容，例如，如何保证口令的安全，如何保证数据库目录的安全，如何使用非特权用户启动服务器。

习题

1、 在你刚刚新安装了一个 MySQL 服务器，在你增加了一个允许连接 MySQL 的用户，用下列语句：

GRANT ALL ON samp_db.* TO fox@*.zoo.net IDENTIFIED BY "cocoa"

而 fox 碰巧在服务器主机上有个账号，所以他试图连接服务器：

%mysql -u fox -pcocoa samp_db

ERROR 1045: Access denied for user: 'fox@localhost' (Using password: YES)

为什么？

2、 授权一个用户 admin，使之具有重启、关闭服务器权限。分别使用 GRANT 语句和直接修改授权表的方法。

3、 然后撤销上题创建的这个用户以及他的授权。

4、 创建一个数据库 mark，然后创建一个用户 teacher，允许他对数据库中的表进行一切操作（当然，不包括 FILE 权限）。

第8章

数据库优化

本章要点：

- 如何使用索引优化表
- 如何选用合适的列类型
- 如何优化 SQL 查询
- 如何设定服务器参数

关系数据库的世界是一个表与集合、表与集合上的运算占统治地位的世界。数据库是一个表的集合，而表又是行和列的集合。在发布一条 SELECT 查询从表中进行检索行时，得到另一个行和列的集合。这些都是一些抽象的概念，对于数据库系统用来操纵表中数据的基本表示没有多少参考价值。另一个抽象概念是，表上的运算都同时进行；查询是一种概念性的集合运算，并且集合论中没有时间概念。

当然，现实世界是相当不同的。数据库管理系统实现了抽象的概念，但是在实际的硬件范围内要受到实际的物理约束。结果是，查询要花时间，有时要花很长的时间。而人类很容易不耐烦，不喜欢等待，因此我们丢下了集合上的那些瞬间的数学运算的抽象世界去寻求加速查询的方法。幸运的是，有几种加速运算的技术，可对表进行索引使数据库服务器查找行更快。可考虑怎样充分利用这些索引来编写查询。可编写影响服务器调度机制的查询，使来自多个客户机的查询协作得更好。我们思考基本硬件怎样运行，以便想出怎样克服其物理约束对性能进行改善的方法。

这些正是本章所要讨论的问题，其目标是优化数据库系统的性能，使其尽可能快地处理各种查询。MySQL 已经相当快了，但即使是最快的数据库，在人的设计下还能运行得更快。

优化是一项复杂的任务，因为它最终需要对整个系统的理解。当用你的系统/应用的小知识做一些局部优化是可能的时候，你越想让你的系统更优化，你必须知道它也越多。

因此，本章将试图解释并给出优化 MySQL 的不同方法的一些例子。但是记住总是有某些(逐渐变难)是系统更快的方法留着去做。

8.1 索引的使用

我们首先讨论索引，因为它是加快查询的最重要的工具。还有其他加快查询的技术，但是最有效的莫过于恰当地使用索引了。在 MySQL 的邮件清单上，人们通常询问关于使查询更快的问题。在大量的案例中，都是因为表上没有索引，一般只要加上索引就可以立即解决问题。但这样也并非总是有效，因为优化并非总是那样简单。然而，如果不使用索引，在许多情形下，用其他手段改善性能只会是浪费时间。应该首先考虑使用索引取得最大的性能改善，然后再寻求其他可能有帮助的技术。

本节介绍索引是什么、它怎样改善查询性能、索引在什么情况下可能会降低性能，以及怎样为表选择索引。下一节，我们将讨论 MySQL 的查询优化程序。除了知道怎样创建索引外，了解一些优化程序的知识也是有好处的，因为这样可以更好地利用所创建的索引。某些编写查询的方法实际上会妨碍索引的效果，应该避免这种情况出现。(虽然并非总会这样。有时也会希望忽略优化程序的作用。我们也将介绍这些情况。)

8.1.1 索引对单个表查询的影响

索引被用来快速找出在一个列上用一特定值的行。没有索引，MySQL 不得不首先以第一条记录开始并然后读完整个表直到它找出相关的行。表越大，花费时间越多。如果表对于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，没有必要考虑所有数据。如果一个表有 1000 行，这比顺序读取至少快 100 倍。注意你需要存取几乎所有 1000 行，它较快的顺序读取，因为此时我们避免磁盘寻道。

例如对下面这样的一个 student 表：

```
mysql>SELECT * FROM student
+------+---------+---------+---------+---------+
| id   | name    | english | chinese | history |
+------+---------+---------+---------+---------+
|   12 | Tom     |      66 |      93 |      67 |
|   56 | Paul    |      78 |      52 |      75 |
|   10 | Marry   |      54 |      89 |      74 |
|    4 | Tina    |      99 |      83 |      48 |
|   39 | William |      43 |      96 |      52 |
```

```
|   74 | Stone   |      42 |      40 |      61 |
|   86 | Smith   |      49 |      85 |      78 |
|   37 | Black   |      49 |      63 |      47 |
|   89 | White   |      94 |      31 |      52 |
+------+---------+---------+---------+---------+
```

这样，我们试图对它进行一个特定查询时，就不得不做一个全表的扫描，速度很慢。例如，我们查找出所有 english 成绩不及格的学生：

mysql>SELECT name,english FROM student WHERE english<60;

```
+---------+---------+
| name    | english |
+---------+---------+
| Marry   |      54 |
| William |      43 |
| Stone   |      42 |
| Smith   |      49 |
| Black   |      49 |
+---------+---------+
```

其中，WHERE 从句不得不匹配每个记录，以检查是否符合条件。对于这个较小的表也许感觉不到太多的影响。但是对于一个较大的表，例如一个非常大的学校，我们可能需要存储成千上万的记录，这样一个检索的所花的时间是十分可观的。

如果，我们为 english 列创建一个索引：

mysql>ALTER TABLE student ADD INDEX (english) ;

```
+-----------------+
| index for english |
+-----------------+
|                42 |
|                43 |
|                49 |
|                49 |
|                54 |
|                66 |
|                78 |
|                94 |
|                99 |
+-----------------+
```

如上表，此索引存储在索引文件中，包含表中每行的 english 列值，但此索引是在 english 的基础上排序的。现在，不需要逐行搜索全表查找匹配的条款，而是可以利用索引进行查找。假如我们要查找分数小于 60 的所有行，那么可以扫描索引，结果得出 5 行。然

后到达分数为 66 的行，及 Tom 的记录，这是一个比我们正在查找的要大的值。索引值是排序的，因此在读到包含 Tom 的记录时，我们知道不会再有匹配的记录，可以退出了。如果查找一个值，它在索引表中某个中间点以前不会出现，那么也有找到其第一个匹配索引项的定位算法，而不用进行表的顺序扫描（如二分查找法）。这样，可以快速定位到第一个匹配的值，以节省大量搜索时间。数据库利用了各种各样的快速定位索引值的技术，这些技术是什么并不重要，重要的是它们工作正常，索引技术是个好东西。

因此在执行下述查询

mysql>SELECT name,english FROM user WHERE english<60;

其结果为：

```
+---------+---------+
| name    | english |
+---------+---------+
| Stone   |      42 |
| William |      43 |
| Smith   |      49 |
| Black   |      49 |
| Marry   |      54 |
+---------+---------+
```

你应该可以发现，这个结果与未索引 english 列之前的不同，它是排序的，原因正式如上所述。

8.1.2 索引对多个表查询的影响

前面的讨论描述了单表查询中索引的好处，其中使用索引消除了全表扫描，极大地加快了搜索的速度。在执行涉及多个表的连接查询时，索引甚至会更有价值。在单个表的查询中，每列需要查看的值的数目就是表中行的数目。而在多个表的查询中，可能的组合数目极大，因为这个数目为各表中行数之积。

假如有三个未索引的表 t1、t2、t3，分别只包含列 c1、c2、c3，每个表分别由含有数值 1 到 1000 的 1000 行组成。查找对应值相等的表行组合的查询如下所示：

此查询的结果应该为 1000 行，每个组合包含 3 个相等的值。如果我们在无索引的情况下处理此查询，则不可能知道哪些行包含那些值。因此，必须寻找出所有组合以便得出与 WHERE 子句相配的那些组合。可能的组合数目为 1000×1000×1000（十亿），比匹配数目多一百万倍。很多工作都浪费了，并且这个查询将会非常慢，即使在如像 MySQL 这样快的数据库中执行也会很慢。而这还是每个表中只有 1000 行的情形。如果每个表中有一百万行时，将会怎样？很显然，这样将会产生性能极为低下的结果。如果对每个表进行索引，就能极大地加速查询进程，因为利用索引的查询处理如下：

1) 如下从表 t1 中选择第一行，查看此行所包含的值。

2) 使用表 t2 上的索引，直接跳到 t2 中与来自 t1 的值匹配的行。类似，利用表 t3 上的索引，直接跳到 t3 中与来自 t1 的值匹配的行。

3) 进到表 t1 的下一行并重复前面的过程直到 t1 中所有的行已经查过。

在此情形下，我们仍然对表 t1 执行了一个完全扫描，但能够在表 t2 和 t3 上进行索引查找直接取出这些表中的行。从道理上说，这时的查询比未用索引时要快一百万倍。

如上所述，MySQL 利用索引加速了 WHERE 子句中与条件相配的行的搜索，或者说在执行连接时加快了与其他表中的行匹配的行的搜索。

8.1.3 多列索引对查询的影响

假定你发出下列SELECT 语句：

mysql> SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;

如果一个多列索引存在于col1 和 col2 上，适当的行可以直接被取出。如果分开的单行列索引存在于col1 和 col2 上，优化器试图通过决定哪个索引将找到更少的行并来找出更具限制性的索引并且使用该索引取行。

 你可以这样创建一个多列索引：

mysql>ALTER TABLE tbl_name ADD INDEX(col1,col2);

而你应该这样创建分开的单行列索引：

mysql>ALTER TABLE tble_name ADD INDEX(col1);

mysql>ALTER TABLE tble_name ADD INDEX(col1);

- 如果表有一个多列索引，任何最左面的索引前缀能被优化器使用以找出行。例如，如果你有一个 3 行列索引(col1,col2,col3)，你已经索引了在(col1)、(col1,col2)和(col1,col2,col3)上的搜索能力。

如果列不构成索引的最左面前缀，MySQL 不能使用一个部分的索引。假定你下面显示的 SELECT 语句：

mysql> SELECT * FROM tbl_name WHERE col1=val1;

mysql> SELECT * FROM tbl_name WHERE col2=val2;

mysql> SELECT * FROM tbl_name WHERE col2=val2 AND col3=val3;

如果一个索引存在于(col1、col2、col3)上，只有上面显示的第一个查询使用索引。第二个和第三个查询确实包含索引的列，但是(col2)和(col2、col3)不是(col1、col2、col3)的最左面前缀。

- 如果 LIKE 参数是一个不以一个通配符字符起始的一个常数字符串，MySQL 也为 LIKE 比较使用索引。例如，下列 SELECT 语句使用索引：

mysql> select * from tbl_name where key_col LIKE "Patrick%";

mysql> select * from tbl_name where key_col LIKE "Pat%_ck%";

在第一条语句中，只考虑有"Patrick" <= key_col < "Patricl"的行。在第二条语句中，只考虑有"Pat" <= key_col < "Pau"的行。

下列 SELECT 语句将不使用索引：

```
mysql> select * from tbl_name where key_col LIKE "%Patrick%";
mysql> 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 column_name 是一个索引，使用 column_name IS NULL 的搜索将使用索引。
- MySQL 通常使用找出最少数量的行的索引。一个索引被用于你与下列操作符作比较的列：=、>、>=、<、<=、BETWEEN 和一个有一个非通配符前缀象'something%'的 LIKE 的列。
- 对于一个多列索引，如果在 WHERE 子句的所有 AND 层次使用索引，将不使用来索引优化查询。为了能够使用索引优化查询，必须把一个多列索引的前缀使用在一个 AND 条件组中。

下列 WHERE 子句使用索引：

```
... WHERE index_part1=1 AND index_part2=2
... WHERE index=1 OR A=10 AND index=2          /* index = 1 OR index = 2 */
... WHERE index_part1='hello' AND index_part_3=5
              /* optimized like "index_part1='hello'" */
```

这些 WHERE 子句不使用索引：

```
... WHERE index_part2=1 AND index_part3=2    /* index_part_1 is not used */
... WHERE index=1 OR A=10                     /* No index */
... WHERE index_part1=1 OR index_part2=10    /* No index spans all rows */
```

8.1.4 索引的作用

所有的 MySQL 索引(PRIMARY、UNIQUE 和 INDEX)在 B 树中存储。字符串是自动地压缩前缀和结尾空间。CREATE INDEX 句法。

索引用于：

- 快速找出匹配一个 WHERE 子句的行。
- 在多个表的查询时，执行连接时加快了与其他表中的行匹配的行的搜索。
- 对特定的索引列找出 MAX()或 MIN()值。
- 如果排序或分组在一个可用索引的最左面前缀上进行(例如，ORDER BY key_part_1,key_part_2)，排序或分组一个表。如果所有键值部分跟随 DESC，键以倒序被读取。
- 在一些情况中，一个查询能被优化来检索值，不用咨询数据文件。如果对某些表的所有使用的列是数字型的并且构成某些键的最左面前缀，为了更快，值可以从索引树被检索出来。

8.1.5 索引的弊端

一般情况下，如果 MySQL 能够知道怎样用索引来更快地处理查询，它就会这样做。这表示，在大多数情况下，如果您不对表进行索引，则损害的是您自己的利益。可以看出，作者描绘了索引的诸多好处。但有不利之处吗？是的，有。实际上，这些缺点被优点所掩

盖了，但应该对它们有所了解。

首先，索引文件要占磁盘空间。如果有大量的索引，索引文件可能会比数据文件更快地达到最大的文件尺寸。其次，索引文件加快了检索，但增加了插入和删除，以及更新索引列中的值的时间（即，降低了大多数涉及写入的操作的时间），因为写操作不仅涉及数据行，而且还常常涉及索引。一个表拥有的索引越多，则写操作的平均性能下降就越大。在8.4.4节记录装载和修改的速度中，我们将更为详细地介绍这些性能问题，并讨论怎样解决。

8.1.6 选择索引的准则

创建索引的语法已经在4.5索引属性中进行了介绍。这里，我们假定您已经阅读过该节。但是知道语法并不能帮助确定表怎样进行索引。要决定表怎样进行索引需要考虑表的使用方式。本节介绍一些关于怎样确定和挑选索引列的准则：

1、搜索的索引列，不一定是所要选择的列

换句话说，最适合索引的列是出现在WHERE子句中的列，或连接子句中指定的列，而不是出现在SELECT关键字后的选择列表中的列，例如：

```
SELECT
col_a                          ←不适合作索引列
FROM
 Tbl1 LEFT JOIN tbl2
ON tbl1.col_b = tbl2.col_c     ←适合作索引列
WHERE
col_d = expr                   ←适合作索引列
```

当然，所选择的列和用于WHERE子句的列也可能是相同的。关键是，列出现在选择列表中不是该列应该索引的标志。

出现在连接子句中的列或出现在形如col1 = col2的表达式中的列是很适合索引的列。查询中的col_b和col_c就是这样的例子。如果MySQL能利用连接列来优化一个查询，表示它通过消除全表扫描相当可观地减少了表行的组合。

2、使用惟一索引

考虑某列中值的分布。对于惟一值的列，索引的效果最好，而具有多个重复值的列，其索引效果最差。例如，存放年龄的列具有不同值，很容易区分各行。而用来记录性别的列，只含有“M”和“F”，则对此列进行索引没有多大用处（不管搜索哪个值，都会得出大约一半的行）。

3、使用短索引

如果对串列进行索引，应该指定一个前缀长度，只要有可能就应该这样做。例如，如果有一个CHAR(200)列，如果在前10个或20个字符内，多数值是惟一的，那么就不要对整个列进行索引。对前10个或20个字符进行索引能够节省大量索引空间，也可能会使查询更快。较小的索引涉及的磁盘I/O较少，较短的值比较起来更快。更为重要的是，对于较短的键值，索引高速缓存中的块能容纳更多的键值，因此，MySQL也可以在内存中容纳更多的值。这增加了找到行而不用读取索引中较多块的可能性。（当然，应该利用一

些常识。如仅用列值的第一个字符进行索引是不可能有多大好处的，因为这个索引中不会有许多不同的值。）

4、利用最左前缀

在创建一个 n 列的索引时，实际是创建了 MySQL 可利用的 n 个索引。多列索引可起几个索引的作用，因为可利用索引中最左边的列集来匹配行。这样的列集称为最左前缀。（这与索引一个列的前缀不同，索引一个列的前缀是利用该的前 n 个字符作为索引值。）

假如一个表在分别名为 state、city 和 zip 的三个列上有一个索引。索引中的行是按 state/city/zip 的次序存放的，因此，索引中的行也会自动按 state/city 的顺序和 state 的顺序存放。这表示，即使在查询中只指定 state 值或只指定 state 和 city 的值，MySQL 也可以利用索引。因此，此索引可用来搜索下列的列组合：

MySQL 不能使用不涉及左前缀的搜索。例如，如果按 city 或 zip 进行搜索，则不能使用该索引。如果要搜索某个州以及某个 zip 代码（索引中的列 1 和列 3），则此索引不能用于相应值的组合。但是，可利用索引来寻找与该州相符的行，以减少搜索范围。

5、不要过度索引

不要以为索引“越多越好”，什么东西都用索引是错的。每个额外的索引都要占用额外的磁盘空间，并降低写操作的性能，这一点我们前面已经介绍过。在修改表的内容时，索引必须进行更新，有时可能需要重构，因此，索引越多，所花的时间越长。如果有一个索引很少利用或从不使用，那么会不必要地减缓表的修改速度。此外，MySQL 在生成一个执行计划时，要考虑各个索引，这也要费时间。创建多余的索引给查询优化带来了更多的工作。索引太多，也可能会使 MySQL 选择不到所要使用的最好索引。只保持所需的索引有利于查询优化。

如果想给已索引的表增加索引，应该考虑所要增加的索引是否是现有多列索引的最左索引。如果是，则就不要费力去增加这个索引了，因为已经有了。

6、考虑在列上进行的比较类型

索引可用于“<”、“<=”、“=”、“>=”、“>”和 BETWEEN 运算。在模式具有一个直接量前缀时，索引也用于 LIKE 运算。如果只将某个列用于其他类型的运算时（如 STRCMP( )），对其进行索引没有价值。

8.1.7 总结

本节介绍了索引在优化查询中的作用，包括了索引优化查询的原理，索引在各种情况的检索中的益处，也包括索引的的弊端：增加了存储的空间，使装载数据变慢。

索引是优化查询的最常用也是最有效的的方法，一个数据表，尤其是容量很大的表，建立合适的索引，会使查询的速度提高很大。

8.2 数据类型的问题

8.2.1 有助于效率的类型选择

1、使你的数据尽可能小

最基本的优化之一是使你的数据(和索引)在磁盘上(并且在内存中)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通常也用较少的主存储器。如果在更小的列上做索引，索引也占据较少的资源。

你能用下面的技术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)的类型。MySQL 有很多节省磁盘空间和内存的专业化类型。
- 如果可能使表更小，使用较小的整数类型。例如，MEDIUMINT 经常比 INT 好一些。
- 如果可能，声明列为 NOT NULL。它使任何事情更快而且你为每列节省一位。注意如果在你的应用程序中你确实需要 NULL，你应该毫无疑问使用它，只是避免缺省地在所有列上有它。

2、使用定长列，不使用可变长列

这条准则对被经常修改，从而容易产生碎片的表来说特别重要。例如，应该选择 CHAR 列而不选择 VARCHAR 列。所要权衡的是使用定长列时，表所占用的空间更多，但如果能够承担这种空间的耗费，使用定长行将比使用可变长的行处理快得多。

3、将列定义为 NOT NULL

这样处理更快，所需空间更少。而且有时还能简化查询，因为不需要检查是否存在特例 NULL。

4、考虑使用 ENUM 列

如果有一个只含有限数目的特定值的列，那么应该考虑将其转换为 ENUM 列。ENUM 列的值可以更快地处理，因为它们在内部是以数值表示的。

8.2.2 有关 BLOB 和 TEXT 类型

1、使用 BLOB 和 TEXT 类型的优点

用 BLOB 存储应用程序中包装或未包装的数据，有可能使原来需要几个检索操作才能完成的数据检索得以在单个检索操作中完成。而且还对存储标准表结构不易表示的数据或随时间变化的数据有帮助。

2、使用 BLOB 和 TEXT 类型的可能弊端

另一方面，BLOB 值也有自己的固有问题，特别是在进行大量的 DELETE 或 UPDATE 操作时更是如此。删除 BLOB 会在表中留下一个大空白，在以后将需用一个记录或可能是不同大小的多个记录来填充。

除非有必要，否则应避免检索较大的 BLOB 或 TEXT 值。例如，除非肯定 WHERE 子句能够将结果恰好限制在所想要的行上，否则 SELECT * 查询不是一个好办法。这样做可能会将非常大的 BLOB 值无目的地从网络上拖过来。这是存储在另一列中的 BLOB 标识信息很有用的另一种情形。可以搜索该列以确定想要的行，然后从限定的行中检索 BLOB 值。

3、必要的准则

- 对容易产生碎片的表使用 OPTIMIZE TABLE

大量进行修改的表，特别是那些含有可变长列的表，容易产生碎片。碎片不好，因为

它在存储表的磁盘块中产生不使用的空间。随着时间的增长，必须读取更多的块才能取到有效的行，从而降低了性能。任意具有可变长行的表都存在这个问题，但这个问题对 BLOB 列更为突出，因为它们尺寸的变化非常大。经常使用 OPTIMIZE TABLE 有助于保持性能不下降。

- 使用多列索引

多列索引列有时很有用。一种技术是根据其他列建立一个散列值，并将其存储在一个独立的列中，然后可通过搜索散列值找到行。这只对精确匹配的查询有效。（散列值对具有诸如“<”或“>=”这样的操作符的范围搜索没有用处）。在 MySQL 3.23 版及以上版本中，散列值可利用 MD5( ) 函数产生。

散列索引对 BLOB 列特别有用。有一事要注意，在 MySQL 3.23.2 以前的版本中，不能索引 BLOB 类型。甚至是在 3.23.2 或更新的版本中，利用散列值作为标识值来查找 BLOB 值也比搜索 BLOB 列本身更快。

- 将 BLOB 值隔离在一个独立的表中

在某些情况下，将 BLOB 列从表中移出放入另一个副表可能具有一定的意义，条件是移出 BLOB 列后可将表转换为定长行格式。这样会减少主表中的碎片，而且能利用定长行的性能优势。

8.2.3 使用 ANALYSE 过程检查表列

如果使用的是 MySQL 3.23 或更新的版本，应该执行 PROCEDURE ANALYSE( )，查看它所提供的关于表中列的信息

ANALYSE([max elements,[max memory]])

它检验来自你的查询的结果并返回结果的分析。

max elements（缺省 256)是 analyse 将注意的每列不同值的最大数量。这被 ANALYSE 用来检查最佳的列类型是否应该是 ENUM 类型。

max memory（缺省 8192)是在 analyse 尝试寻找所有不同值的时候应该分配给每列的最大内存量。

SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max elements,[max memory]])

例如：

mysql>SELECT * FROM student PROCEDURE ANALYSE();

mysql>SELECT * FROM student PROCEDURE ANALYSE(16,256);

相应输出中有一列是关于表中每列的最佳列类型的建议。第二个例子要求 PROCEDURE ANALYSE( ) 不要建议含有多于 16 个值或取多于 256 字节的 ENUM 类型（可根据需要更改这些值）。如果没有这样的限制，输出可能会很长；ENUM 的定义也会很难阅读。

根据 PROCEDURE ANALYSE( ) 的输出，会发现可以对表进行更改以利用更有效的类型。如果希望更改值类型，使用 ALTER TABLE 语句即可。

8.2.3 总结

最基本的优化之一是使你的数据(和索引)在磁盘上(并且在内存中)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通常也用较少的主存储器。如果在更小的列上做索引，索引也占据较少的资源。

当你不确定使用何种类型更合适时，你可以使用 ANALYSE 过程分析你的 SQL 语句的效率。

8.3 SQL 查询的优化

8.3.1 使用 EXPLAIN 语句检查 SQL 语句

当你在一条 SELECT 语句前放上关键词 EXPLAIN，MySQL 解释它将如何处理 SELECT，提供有关表如何联结和以什么次序联结的信息。

借助于 EXPLAIN，你可以知道你什么时候必须为表加入索引以得到一个使用索引找到记录的更快的 SELECT。

EXPLAIN tbl_name

or EXPLAIN SELECT select_options

EXPLAIN tbl_name 是 DESCRIBE tbl_name 或 SHOW COLUMNS FROM tbl_name 的一个同义词。

从 EXPLAIN 的输出包括下面列：

- table

输出的行所引用的表。

- type

联结类型。各种类型的信息在下面给出。

不同的联结类型列在下面，以最好到最差类型的次序：

system const eq_ref ref range index ALL possible_keys

- key

key 列显示 MySQL 实际决定使用的键。如果没有索引被选择，键是 NULL。

- key_len

key_len 列显示 MySQL 决定使用的键长度。如果键是 NULL，长度是 NULL。注意这告诉我们 MySQL 将实际使用一个多部键值的几个部分。

- ref

ref 列显示哪个列或常数与 key 一起用于从表中选择行。

- rows

rows 列显示 MySQL 相信它必须检验以执行查询的行数。

- Extra

如果 Extra 列包括文字 Only index，这意味着信息只用索引树中的信息检索出的。通常，这比扫描整个表要快。如果 Extra 列包括文字 where used，它意味着一个 WHERE 子句将被用来限制哪些行与下一个表匹配或发向客户。

通过相乘 EXPLAIN 输出的 rows 行的所有值，你能得到一个关于一个联结要多好的提

示。这应该粗略地告诉你 MySQL 必须检验多少行以执行查询。

例如，下面一个全连接：

```
mysql> EXPLAIN SELECT student.name From student,pet
    -> WHERE student.name=pet.owner;
```

其结论为：

```
+---------+------+---------------+------+---------+------+------+------------+
| table   | type | possible_keys | key  | key_len | ref  | rows | Extra      |
+---------+------+---------------+------+---------+------+------+------------+
| student | ALL  | NULL          | NULL |    NULL | NULL |   13 |            |
| pet     | ALL  | NULL          | NULL |    NULL | NULL |    9 | where used |
+---------+------+---------------+------+---------+------+------+------------+
```

8.3.2 SELECT 查询的速度

总的来说，当你想要使一个较慢的 SELECT ... WHERE 更快，检查的第一件事情是你是否能增加一个索引。见 10.4 MySQL 索引的使用。在不同表之间的所有引用通常应该用索引完成。你可以使用 EXPLAIN 来确定哪个索引用于一条 SELECT 语句。见 7.22 EXPLAIN 句法(得到关于一条 SELECT 的信息)。

一些一般的建议：

- 为了帮助 MySQL 更好地优化查询，在它已经装载了相关数据后，在一个表上运行 myisamchk --analyze。这为每一个更新一个值，指出有相同值地平均行数（当然，对唯一索引，这总是 1。）
- 为了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）。如果你有一个唯一索引，你想要根据该索引地次序读取所有的记录，这是使它更快的一个好方法。然而注意，这个排序没有被最佳地编写，并且对一个大表将花很长时间！

8.3.2.1 MySQL 怎样优化 WHERE 子句

where 优化被放在 SELECT 中，因为他们最主要在那里使用里，但是同样的优化被用于 DELETE 和 UPDATE 语句。

也要注意，本节是不完全的。MySQL 确实作了许多优化而我们没有时间全部记录他们。

由 MySQL 实施的一些优化列在下面：

1、删除不必要的括号：

```
   ((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)
```

2、常数调入：

```
   (a<b AND b=c) AND a=5
-> b>5 AND b=c AND a=5
```

3、删除常数条件(因常数调入所需)：

(B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)

-> B=5 OR B=6

4、索引使用的常数表达式仅计算一次。

5、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，对任何 NOT NULL 表达式也这样做。

6、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。

7、如果你不使用 GROUP BY 或分组函数(COUNT()、MIN()……), HAVING 与 WHERE 合并。

8、为每个子联结(sub join)，构造一个更简单的 WHERE 以得到一个更快的 WHERE 计算并且也尽快跳过记录。

9、所有常数的表在查询中的任何其他表前被首先读出。一个常数的表是：

- 一个空表或一个有 1 行的表。
- 与在一个 UNIQUE 索引、或一个 PRIMARY KEY 的 WHERE 子句一起使用的表，这里所有的索引部分使用一个常数表达式并且索引部分被定义为 NOT NULL。

所有下列的表用作常数表：

```
mysql> SELECT * FROM t WHERE primary_key=1;
mysql> SELECT * FROM t1,t2
          WHERE t1.primary_key=1 AND t2.primary_key=t1.id;
```

10、对联结表的最好联结组合是通过尝试所有可能性来找到:(。如果所有在 ORDER BY 和 GROUP BY 的列来自同一个表，那么当廉洁时，该表首先被选中。

11、如果有一个 ORDER BY 子句和一个不同的 GROUP BY 子句，或如果 ORDER BY 或 GROUP BY 包含不是来自联结队列中的第一个表的其他表的列，创建一个临时表。

12、如果你使用 SQL_SMALL_RESULT，MySQL 将使用一个在内存中的表。

13、因为 DISTINCT 被变换到在所有的列上的一个 GROUP BY，DISTINCT 与 ORDER BY 结合也将在许多情况下需要一张临时表。

14、每个表的索引被查询并且使用跨越少于 30% 的行的索引。如果这样的索引没能找到，使用一个快速的表扫描。

15、在一些情况下，MySQL 能从索引中读出行，甚至不咨询数据文件。如果索引使用的所有列是数字的，那么只有索引树被用来解答查询。

16、在每个记录被输出前，那些不匹配 HAVING 子句的行被跳过。

下面是一些很快的查询例子：

```
mysql> SELECT COUNT(*) FROM tbl_name;
mysql> SELECT MIN(key_part1),MAX(key_part1) FROM tbl_name;
mysql> SELECT MAX(key_part2) FROM tbl_name
          WHERE key_part_1=constant;
mysql> SELECT ... FROM tbl_name
          ORDER BY key_part1,key_part2,... LIMIT 10;
```

```
mysql> SELECT ... FROM tbl_name
            ORDER BY key_part1 DESC,key_part2 DESC,... LIMIT 10;
```

下列查询仅使用索引树就可解决(假设索引列是数字的)：

```
mysql> SELECT key_part1,key_part2 FROM tbl_name WHERE key_part1=val;
mysql> SELECT COUNT(*) FROM tbl_name
            WHERE key_part1=val1 AND key_part2=val2;
mysql> SELECT key_part2 FROM tbl_name GROUP BY key_part1;
```

下列查询使用索引以排序顺序检索，不用一次另外的排序：

```
mysql> SELECT ... FROM tbl_name ORDER BY key_part1,key_part2,...
mysql> SELECT ... FROM tbl_name ORDER BY key_part1 DESC,key_part2 DESC,...
```

8.3.2.2 MySQL 怎样优化 LEFT JOIN

在 MySQL 中，A LEFT JOIN B 实现如下：

1、表 B 被设置为依赖于表 A。

2、表 A 被设置为依赖于所有用在 LEFT JOIN 条件的表(除 B 外)。

3、所有 LEFT JOIN 条件被移到 WHERE 子句中。

4、进行所有标准的联结优化，除了一个表总是在所有它依赖的表之后被读取。如果有一个循环依赖，MySQL 将发出一个错误。

5、进行所有标准的 WHERE 优化。

6、如果在 A 中有一行匹配 WHERE 子句，但是在 B 中没有任何行匹配 LEFT JOIN 条件，那么在 B 中生成所有列设置为 NULL 的一行。

7、如果你使用 LEFT JOIN 来找出在某些表中不存在的行并且在 WHERE 部分你有下列测试：column_name IS NULL，这里 column_name 被声明为 NOT NULL 的列，那么 MySQL 在它已经找到了匹配 LEFT JOIN 条件的一行后，将停止在更多的行后寻找(对一特定的键组合)。

8.3.2.3 MySQL 怎样优化 LIMIT

在一些情况中，当你使用 LIMIT #而不使用 HAVING 时，MySQL 将以不同方式处理查询。

1、如果你用 LIMIT 只选择一些行，当 MySQL 一般比较喜欢做完整的表扫描时，它将在一些情况下使用索引。

2、如果你使用 LIMIT #与 ORDER BY，MySQL 一旦找到了第一个 # 行，将结束排序而不是排序整个表。

3、当结合 LIMIT #和 DISTINCT 时，MySQL 一旦找到#个唯一的行，它将停止。

4、在一些情况下，一个 GROUP BY 能通过顺序读取键(或在键上做排序)来解决，并然后计算摘要直到键值改变。在这种情况下，LIMIT #将不计算任何不必要的 GROUP。

5、只要 MySQL 已经发送了第一个#行到客户，它将放弃查询。

6、LIMIT 0 将总是快速返回一个空集合。这对检查查询并且得到结果列的列类型是有用的。

7、临时表的大小使用 LIMIT #计算需要多少空间来解决查询。

8.3.4 记录转载和修改的速度

很多时候关心的是优化 SELECT 查询，因为它们是最常用的查询，而且确定怎样优化它们并不总是直截了当。相对来说，将数据装入数据库是直截了当的。然而，也存在可用来改善数据装载操作效率的策略，其基本原理如下：

- 成批装载较单行装载更快，因为在装载每个记录后，不需要刷新索引高速缓存；可在成批记录装入后才刷新。
- 在表无索引时装载比索引后装载更快。如果有索引，不仅必须增加记录到数据文件，而且还要修改每个索引以反映增加了的新记录。
- 较短的 SQL 语句比较长的 SQL 语句要快，因为它们涉及服务器方的分析较少，而且还因为将它们通过网络从客户机发送到服务器更快。

这些因素中有一些似乎微不足道（特别是最后一个因素），但如果要装载大量的数据，即使是很小的因素也会产生很大的不同结果。

8.3.4.1 INSERT 查询的速度

插入一个记录的时间由下列组成：

- 连接：(3)
- 发送查询给服务器：(2)
- 分析查询：(2)
- 插入记录：（1 x 记录大小）
- 插入索引：（1 x 索引）
- 关闭：(1)

这里的数字有点与总体时间成正比。这不考虑打开表的初始开销(它为每个并发运行的查询做一次)。

表的大小以 N log N (B 树)的速度减慢索引的插入。

加快插入的一些方法：

- 如果你同时从同一客户插入很多行，使用多个值表的 INSERT 语句。这比使用分开 INSERT 语句快(在一些情况中几倍)。
- 如果你从不同客户插入很多行，你能通过使用 INSERT DELAYED 语句得到更高的速度。
- 注意，用 MyISAM，如果在表中没有删除的行，能在 SELECT:s 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。
- 当表有很多索引时，有可能多做些工作使得 LOAD DATA INFILE 更快些。使用下列过程：

1、有选择地用 CREATE TABLE 创建表。例如使用 mysql 或 Perl-DBI。

2、执行 FLUSH TABLES，或外壳命令 mysqladmin flush-tables。

3、使用 myisamchk --keys-used=0 -rq /path/to/db/tbl_name。这将从表中删除所有索引的使用。

4、用 LOAD DATA INFILE 把数据插入到表中，这将不更新任何索引，因此很快。

5、如果你有 myisampack 并且想要压缩表，在它上面运行 myisampack。

6、用 myisamchk -r -q /path/to/db/tbl_name 再创建索引。这将在将它写入磁盘前在内存中创建索引树，并且它更快，因为避免大量磁盘寻道。结果索引树也被完美地平衡。

7、执行 FLUSH TABLES，或外壳命令 mysqladmin flush-tables。

这个过程将被构造进在 MySQL 的某个未来版本的 LOAD DATA INFILE。

- 你可以锁定你的表以加速插入。

```
mysql> LOCK TABLES a WRITE;
mysql> INSERT INTO a VALUES (1,23),(2,34),(4,33);
mysql> INSERT INTO a VALUES (8,26),(6,29);
mysql> UNLOCK TABLES;
```

主要的速度差别是索引缓冲区仅被清洗到磁盘上一次，在所有 INSERT 语句完成后。一般有与有不同的 INSERT 语句那样夺的索引缓冲区清洗。如果你能用一个单个语句插入所有的行，锁定就不需要。锁定也将降低多连接测试的整体时间，但是对某些线程最大等待时间将上升(因为他们等待锁)。例如：

```
thread 1 does 1000 inserts
thread 2, 3, and 4 does 1 insert
thread 5 does 1000 inserts
```

如果你不使用锁定，2、3 和 4 将在 1 和 5 前完成。如果你使用锁定，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%。因为 INSERT, UPDATE 和 DELETE 操作在 MySQL 中是很快的，通过为多于大约 5 次连续不断地插入或更新一行的东西加锁，你将获得更好的整体性能。如果你做很多一行的插入，你可以做一个 LOCK TABLES，偶尔随后做一个 UNLOCK TABLES(大约每 1000 行)以允许另外的线程存取表。这仍然将导致获得好的性能。当然，LOAD DATA INFILE 对装载数据仍然是更快的。

为了对 LOAD DATA INFILE 和 INSERT 得到一些更快的速度，扩大关键字缓冲区。

8.3.4.2 UPDATE 查询的速度

更改查询被优化为有一个写开销的一个 SELECT 查询。写速度依赖于被更新数据大小和被更新索引的数量。

使更改更快的另一个方法是推迟更改并且然后一行一行地做很多更改。如果你锁定表，做一行一行地很多更改比一次做一个快。

注意，动态记录格式的更改一个较长总长的记录，可能切开记录。因此如果你经常这样做，时不时地 OPTIMIZE TABLE 是非常重要的。

8.3.4.3 DELETE 查询的速度

删除一个记录的时间精确地与索引数量成正比。为了更快速地删除记录，你可以增加索引缓存的大小。

从一个表删除所有行比删除行的一大部分也要得多。

8.3.4 索引对有效装载数据的影响

如果表是索引的，则可利用批量插入（LOAD DATA 或多行的 INSERT 语句）来减少索引的开销。这样会最小化索引更新的影响，因为索引只需要在所有行处理过时才进行刷新，而不是在每行处理后就刷新。

- 如果需要将大量数据装入一个新表，应该创建该表且在未索引时装载，装载数据后才创建索引，这样做较快。一次创建索引（而不是每行修改一次索引）较快。
- 如果在装载之前删除或禁用索引，装入数据后再重新创建或启用索引可能使装载更快。

如果想对数据装载使用删除或禁用策略，一定要做一些实验，看这样做是否值得（如果将少量数据装入一个大表中，重建和索引所花费的时间可能比装载数据的时间还要长）。

可用 DROP INDEX 和 CREATE INDEX 来删除和重建索引。

另一种可供选择的方法是利用 myisamchk 或 isamchk 禁用和启用索引。这需要在 MySQL 服务器主机上有一个帐户，并对表文件有写入权。为了禁用表索引，可进入相应的数据库目录，执行下列命令之一：

```
shell>myisamchk --keys-used=0 tbl_name
shell>isamchk --keys-used=0 tbl_name
```

对具有 .MYI 扩展名的索引文件的 MyISAM 表使用 myisamchk，对具有 .ISM 扩展名的索引文件的 ISAM 表使用 isamchk。在向表中装入数据后，按如下激活索引：

```
shell>myisamchk --recover --quick --keys-used=0 tbl_name
shell>isamchk --recover --quick --keys-used=0 tbl_name
```

n 为表具有的索引数目。可用 --description 选项调用相应的实用程序得出这个值：

```
shell>myisamchk --discription tbl_name
$isamchk --discription tbl_name
```

如果决定使用索引禁用和激活，应该使用第 13 章中介绍的表修复锁定协议以阻止服务器同时更改锁（虽然此时不对表进行修复，但要对它像表修复过程一样进行修改，因此

需要使用相同的锁定协议）。

上述数据装载原理也适用于与需要执行不同操作的客户机有关的固定查询。例如，一般希望避免在频繁更新的表上长时间运行 SELECT 查询。长时间运行 SELECT 查询会产生大量争用，并降低写入程序的性能。一种可能的解决方法为，如果执行写入的主要是 INSERT 操作，那么先将记录存入一个临时表，然后定期地将这些记录加入主表中。如果需要立即访问新记录，这不是一个可行的方法。但只要能在一个较短的时间内不访问它们，就可以使用这个方法。使用临时表有两个方面的好处。首先，它减少了与主表上 SELECT 查询语句的争用，因此，执行更快。其次，从临时表将记录装入主表的总时间较分别装载记录的总时间少；相应的索引高速缓存只需在每个批量装载结束时进行刷新，而不是在每行装载后刷新。

这个策略的一个应用是进入 Web 服务器的 Web 页访问 MySQL 数据库。在此情形下，可能没有保证记录立即进入主表的较高权限。

如果数据并不完全是那种在系统非正常关闭事件中插入的单个记录，那么减少索引刷新的另一策略是使用 MyISAM 表的 DELAYED_KEY_WRITE 表创建选项（如果将 MySQL 用于某些数据录入工作时可能会出现这种情况）。此选项使索引高速缓存只偶尔刷新，而不是在每次插入后都要刷新。

如果希望在服务器范围内利用延迟索引刷新，只要利用 --delayed-key-write 选项启动 mysqld 即可。在此情形下，索引块写操作延迟到必须刷新块以便为其他索引值腾出空间为止，或延迟到执行了一个 flush-tables 命令后，或延迟到该索引表关闭。

8.3.5 总结

本节介绍了如何优化 SQL 查询。你可以手工使用 EXPLAIN 语句检查 SQL 查询的效率。另外，还讲述了一些优化 SQL 语句的原则，主要是检索记录和装载数据时如何优化 SQL 语句的原则。

8.4 数据库表的处理

选择合适的表的类型，防止数据文件中产生碎块，同样可以大大优化检索的速度，是数据库的性能最大化。

8.4.1 选择一种表类型

用 MySQL，当前(版本 3.23.5)你能从一个速度观点在 4 可用表的格式之间选择。

- 静态 MyISAM

这种格式是最简单且最安全的格式，它也是在磁盘格式最快的。速度来自于数据能在磁盘上被找到的难易方式。当所定有一个索引和静态格式的东西时，它很简单，只是行长度乘以行数量。而且在扫描一张表时，用每次磁盘读取来读入常数个记录是很容易的。安全性来自于如果当写入一个静态 MyISAM 文件时，你的计算机崩溃，myisamchk 能很容易指出每行在哪儿开始和结束，因此它通常能回收所有记录，除了部分被写入的那个。注意，在 MySQL 中，所有索引总能被重建。

- 动态 MyISAM

这种格式有点复杂，因为每一行必须有一个头说明它有多长。当一个记录在更改时变长时，它也可以在多于一个位置上结束。你能使用 OPTIMIZE table 或 myisamchk 整理一张表。如果你在同一个表中有象某些 VARCHAR 或 BLOB 列那样存取/改变的静态数据，将动态列移入另外一个表以避免碎片可能是一个好主意。

- 压缩 MyISAM

这是一个只读类型，用可选的 myisampack 工具生成。

- 内存(HEAP 堆）

这种表格式对小型/中型查找表十分有用。对拷贝/创建一个常用的查找表(用联结)到一个(也许临时)HEAP 表有可能加快多个表联结。假定我们想要做下列联结，用同样数据可能要几倍时间。

```
SELECT tab1.a, tab3.a FROM tab1, tab2, tab3
        WHERE tab1.a = tab2.a and tab2.a = tab3.a and tab2.c != 0;
```

为了加速它，我们可用 tab2 和 tab3 的联结创建一张临时表，因为用相同列(tab1.a)查找。这里是创建该表和结果选择的命令。

```
CREATE TEMPORARY TABLE test TYPE=HEAP
        SELECT
                tab2.a as a2, tab3.a as a3
        FROM
                tab2, tab3
        WHERE
                tab2.a = tab3.a and c = 0;
SELECT tab1.a, test.a3 from tab1, test where tab1.a = test.a1;
SELECT tab1.b, test.a3 from tab1, test where tab1.a = test.a1 and something;
```

8.4.1.1 静态(定长)表的特点

这是缺省格式。它用在表不包含 VARCHAR、BLOB 或 TEXT 列时候。

所有的 CHAR、NUMERIC 和 DECIMAL 列充填到列宽度。

非常快。

容易缓冲。

容易在崩溃后重建，因为记录位于固定的位置。

不必被重新组织(用 myisamchk)，除非一个巨量的记录被删除并且你想要归还空闲磁盘空间给操作系统。

通常比动态表需要更多的磁盘空间。

8.4.1.2 动态表的特点

如果表包含任何 VARCHAR、BLOB 或 TEXT 列，使用该格式。

所有字符串列是动态的(除了那些长度不到 4 的列)。

每个记录前置一个位图，对字符串列指出哪个列是空的(")，或对数字列哪个是零(这不同于包含 NULL 值的列)。如果字符串列在删除尾部空白后有零长度，或数字列有零值，它在位图中标记并且不保存到磁盘上。非空字符串存储为一个长度字节加字符串内容。

通常比定长表占更多的磁盘空间。

每个记录仅使用所需的空间。如果一个记录变得更大，它按需要被切开多段，这导致记录碎片。

如果你与超过行长度的信息更新行，行将被分段。在这种情况中，你可能必须时时运行 myisamchk -r 以使性能更好。使用 myisamchk -ei tbl_name 做一些统计。

在崩溃后不容易重建，因为一个记录可以是分很多段并且一个连接(碎片)可以丢失。

对动态尺寸记录的期望行长度是：

3

+ (number of columns + 7) / 8

+ (number of char columns)

+ packed size of numeric columns

+ length of strings

+ (number of NULL columns + 7) / 8

对每个连接有 6 个字节的惩罚。无论何时更改引起记录的增大，一个动态记录被链接。每个新链接将至少是 20 个字节，因此下一增大将可能在同一链连中。如果不是，将有另外一个链接。你可以用 myisamchk -ed 检查有多少链接。所有的链接可以用 myisamchk -r 删除。

8.4.1.3 压缩表的特点

一张用 myisampack 实用程序制作的只读表。所有具有 MySQL 扩展电子邮件支持的客户可以为其内部使用保留一个 myisampack 拷贝。

解压缩代码存在于所有 MySQL 分发，以便甚至没有 myisampack 的客户能读取用 myisampack 压缩的表。

占据很小的磁盘空间，使磁盘使用量减到最小。

每个记录被单独压缩(很小的存取开销)。对一个记录的头是定长的(1-3 字节)，取决于表中最大的记录。每列以不同方式被压缩。一些压缩类型是：

通常对每列有一张不同的哈夫曼表。

后缀空白压缩。

前缀空白压缩。

用值 0 的数字使用 1 位存储。

如果整数列的值有一个小范围，列使用最小的可能类型来存储。例如，如果所有的值在 0 到 255 的范围，一个 BIGINT 列(8 个字节)可以作为一个 TINYINT 列(1 字节)存储。

如果列仅有可能值的一个小集合，列类型被变换到 ENUM。

列可以使用上面的压缩方法的组合。

能处理定长或动态长度的记录，然而不能处理 BLOB 或 TEXT 列。

能用 myisamchk 解压缩。

MySQL 能支持不同的索引类型，但是一般的类型是 ISAM。这是一个 B 树索引并且

你能粗略地为索引文件计算大小为(key_length+4)*0.67，在所有的键上的总和。（这是对最坏情况，当所有键以排序顺序被插入时。）

字符串索引是空白压缩的。如果第一个索引部分是一个字符串，它也将压缩前缀。如果字符串列有很多尾部空白或是一个总不能用到全长的 VARCHAR 列，空白压缩使索引文件更小。如果很多字符串有相同的前缀，前缀压缩是有帮助的。

8.4.1.4 内存表的特点

堆表仅存在于内存中，因此如果 mysqld 被关掉或崩溃，它们将丢失，但是因为它们是很快，不管怎样它们是有用的。

MySQL 内部的 HEAP 表使用没有溢出区的 100%动态哈希并且没有与删除有关的问题。

你只能通过使用在堆表中的一个索引的用等式存取东西(通常用=操作符)。

堆表的缺点是：

- 你要为你想要同时使用的所有堆表需要足够的额外内存。
- 你不能在索引的一个部分上搜索。
- 你不能顺序搜索下一个条目(即使用这个索引做一个 ORDER BY)。
- MySQL 也不能算出在 2 个值之间大概有多少行。这被优化器使用来决定使用哪个索引，但是在另一方面甚至不需要磁盘寻道。

8.4.2 数据库表的数量的问题

在同一个数据库中创建大量数据库表的缺点是,如果你在一个目录中有许多文件，打开、关闭和创建操作将会很慢。如果你执行在许多不同表上的 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。你可以通过使表缓冲更大些来减少这个开销。

为什么有这么多打开的表？

当你运行 mysqladmin status 时，你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，这可能有点令人困惑。

MySQL 是多线程的，因此它可以同时在同一个表上有许多询问。为了是 2 个线程在同一个文件上有不同状态的问题减到最小，表由每个并发进程独立地打开。这为数据文件消耗一些内存和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。

8.4.3 数据库表级锁定的问题

前面的内容主要将精力集中在使个别的查询更快上。MySQL 还允许影响语句的调度特性，这样会使来自几个客户机的查询更好地协作，从而单个客户机不会被锁定太长的时间。更改调度特性还能保证特定的查询处理得更快。我们先来看一下 MySQL 的缺省调度策略，然后来看看为改变这个策略可使用什么样的选项。出于讨论的目的，假设执行检索（SELECT）的客户机程序为读取程序。执行修改表操作（DELETE，INSERT，REPLACE 或 UPDATE）的另一个客户机程序为写入程序。

MySQL 的基本调度策略可总结如下：

- 写入请求应按其到达的次序进行处理。

- 写入具有比读取更高的优先权。

1、对此一个主要的问题如下：

- 一个客户发出一个花很长时间运行的SELECT。
- 然后其他客户在一个使用的表上发出一个 UPDATE；这个客户将等待直到SELECT 完成。
- 另一个客户在同一个表上发出另一个 SELECT 语句；因为 UPDATE 比 SELECT 有更高的优先级，该SELECT 将等待 UPDATE 的完成。它也将等待第一个 SELECT 完成！

对这个问题的一些可能的解决方案是：

- 试着使 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，在先前情形的最后的 SELECT 语句将在 INSERT 语句前执行。
- 你可以用 LOW_PRIORITY 属性给与一个特定的 INSERT、UPDATE 或 DELETE 语句较低优先级。
- 为 max_write_lock_count 指定一个低值来启动 mysqld 使得在一定数量的 WRITE 锁定后给出 READ 锁定。
- 通过使用 SQL 命令：SET SQL_LOW_PRIORITY_UPDATES=1，你可从一个特定线程指定所有的更改应该由用低优先级完成。见 SET OPTION 句法。
- 你可以用 HIGH_PRIORITY 属性指明一个特定 SELECT 是很重要的。见 SELECT 句法。
- 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。
- 如果你主要混合 INSERT 和 SELECT 语句，DELAYED 属性的 INSERT 将可能解决你的问题。INSERT 句法。
- 如果你有关于 SELECT 和 DELETE 的问题，LIMIT 选项的 DELETE 可以帮助你。见 DELETE 句法。

2、INSERT DELAYED 在客户机方的作用

如果其他客户机可能执行冗长的 SELECT 语句，而且您不希望等待插入完成，此时 INSERT DELAYED 很有用。发布 INSERT DELAYED 的客户机可以更快地继续执行，因为服务器只是简单地将要插入的行插入。

不过应该对正常的 INSERT 和 INSERT DELAYED 性能之间的差异有所认识。如果 INSERT DELAYED 存在语法错误，则向客户机发出一个错误，如果正常，便不发出信息。例如，在此语句返回时，不能相信所取得的 AUTO_INCREMENT 值。也得不到惟一索引上的重复数目的计数。之所以这样是因为此插入操作在实际的插入完成前返回了一个状态。其他还表示，如果 INSERT DELAYED 语句的行在等待插入中被排队，并且服务器崩溃或被终止（用 kill -9），那么这些行将丢失。正常的 TERM 终止不会这样，服务器会在退出前将这些行插入。

在表锁的帮助下实现调度策略。客户机程序无论何时要访问表，都必须首先获得该表的锁。可以直接用 LOCK TABLES 来完成这项工作，但一般服务器的锁管理器会在需要时自动获得锁。在客户机结束对表的处理时，可释放表上的锁。直接获得的锁可用 UNLOCK TABLES 释放，但服务器也会自动释放它所获得的锁。

执行写操作的客户机必须对表具有独占访问的锁。在写操作进行中，由于正在对表进行数据记录的删除、增加或更改，所以该表处于不一致状态，而且该表上的索引也可能需要作相应的更新。如果表处于不断变化中，此时允许其他客户机访问该表会出问题。让两个客户机同时写同一个表显然不好，因为这样会很快使该表不可用。允许客户机读不断变化的表也不是件好事，因为可能在读该表的那一刻正好正在对它进行更改，其结果是不正确的。

执行读取操作的客户机必须有一把防止其他客户机写该表的锁，以保证读表的过程中表不出现变化。不过，该锁无需对读取操作提供独占访问。此锁还允许其他客户机同时对表进行读取。读取不会更改表，所有没必要阻止其它客户机对该表进行读取。

MySQL 允许借助几个查询限修饰符对其调度策略施加影响。其中之一是 DELETE、INSERT、LOAD DATA、REPLACE 和 UPDATE 语句的 LOW_PRIORITY 关键字。另一个是 SELECT 语句的 HIGH_PRIORITY 关键字。第三个是 INSERT 和 REPLACE 语句的 DELAYED 关键字。

LOW_PRIORITY 关键字按如下影响调度。一般情况下，如果某个表的写入操作在表正被读取时到达，写入程序被阻塞，直到读取程序完成，因为一旦某个查询开始，就不能中断。如果另一读取请求在写入程序等待时到达，此读取程序也被阻塞，因为缺省的调度策略为写入程序具有比读取程序高的优先级。在第一个读取程序结束时，写入程序继续，在此写入程序结束时，第二个读取程序开始。

如果写入请求为 LOW_PRIORITY 的请求，则不将该写入操作视为具有比读取操作优先级高的操作。在此情形下，如果第二个读取请求在写入程序等待时到达，则让第二个读取操作排在等待的写入操作之前。仅当没有其他读取请求时，才允许写入程序执行。这种调度的更改从理论上说，其含义为 LOW_PRIORITY 写入可能会永远被阻塞。当正在处理前面的读取请求时，只要另一个读取请求到达，这个新的请求允许排在 LOW_PRIORITY 写入之前。

SELECT 查询的 HIGH_PRIORITY 关键字作用类似。它使 SELECT 插在正在等待的写入操作之前，即使该写入操作具有正常的优先级。

INSERT 的 DELAYED 修饰符作用如下，在表的一个 INSERT DELAYED 请求到达时，服务器将相应的行放入一个队列，并立即返回一个状态到客户机程序，以便该客户机程序可以继续执行，即使这些行尚未插入表中。如果读取程序正在对表进行读取，那么队列中的行挂起。在没有读取时，服务器开始开始插入延迟行队列中的行。服务器不时地停下来看看是否有新的读取请求到达，并进行等待。如果是这样，延迟行队列将挂起，并允许读取程序继续。在没有其他的读取操作时，服务器再次开始插入延迟行。这个过程一直进行到延迟行队列空为止。

此调度修饰符并非出现在所有 MySQL 版本中。下面的表列出了这些修饰符和支持这

些修饰符的 MySQL 版本。可利用此表来判断所使用的 MySQL 版本具有什么样的功能：

语句类型开始出现的版本

DELETE LOW_PRIOrITY 3.22.5

INSERT LOW+PRIOrITY 3.22.5

INSERT DELAYED 3.22.15

LOAD DATA LOW_PRIORITY 3.23.0

LOCK TABLES ... LOW_PRIORITY 3.22.8

REPLACE LOW_PRIORITY 3.22.5

REPLACE DELAYED 3.22.15

SELECT ... HIGH_PRIORITY 3.22.9

UPDATE LOW_PRIORITY 3.22.5

SET SQL_LOW_PRIORITY_UPDATES 3.22.5

8.4.4 对表进行优化

对表的长期使用，由于记录的删除与插入，会在表的数据文件产生碎片，下面的几种情况尤其会有产生碎片的效果：

- 使用 VARCHAR 类型

频繁使用可变长列的结果就是使表非常容易产生碎片，如果空间允许，尽量采用定长类型 CHAR。

- 使用类型 BLOB 和 TEXT

特别是进行大量的 DELETE 或 UPDATE 操作时更是如此。删除 BLOB 会在表中留下一个大空白，在以后将需用一个记录或可能是不同大小的多个记录来填充。

- 删除了一个表的大部分，这同样会在表中留下大量空白。

为了消除表的碎片对性能的影响，需要对表进行优化。在第七章我们介绍了优化表的方法：

1、使用 SQL 语句 OPTIMIZE

OPTIMIZE TABLE tbl_name

2、使用修复程序 myisamchk 或 isamchk

8.4.5 总结

本节介绍了有关数据表的优化技巧，主要内容有，选择表的类型，打开尽量少的表，锁定表与查询速度的关系以及如何优化表以达到提高查询速度的目的。

由于数据的录入和清除操作，很容易在表的数据文件重产生碎片，对于拥有大型表的数据库，应该经常使用 myisamchk 维护程序来清除碎片，优化表时要选择一个好的时机，尽量在没有用户访问时优化表。

8.5 服务器级优化

前面各段介绍了普通的 MySQL 用户利用表创建和索引操作，以及利用查询的编写能够进行的优化。不过，还有一些只能由 MySQL 管理员和系统管理员来完成的优化，这些

管理员在 MySQL 服务器或运行 MySQL 的机器上具有控制权。有的服务器参数直接适用于查询处理，可将它们打开。而有的硬件配置问题直接影响查询处理速度，应该对它们进行调整。

8.5.1 磁盘问题

正如前面所述，磁盘寻道是一个性能的大瓶颈。当数据开始增长以致缓存变得不可能时，这个问题变得越来越明显。对大数据库，在那你或多或少地要随机存取数据，你可以依靠你将至少需要一次磁盘寻道来读取并且几次磁盘寻道写入。为了使这个问题最小化，使用有低寻道时间的磁盘。

为了增加可用磁盘轴的数量(并且从而减少寻道开销)，符号联接文件到不同磁盘或分割磁盘是可能的。

1、使用符号连接

这意味着你将索引/数据文件符号从正常的数据目录链接到其他磁盘(那也可以被分割的)。这使得寻道和读取时间更好(如果磁盘不用于其他事情)。

2、分割

分割意味着你有许多磁盘并把第一块放在第一个磁盘上，在第二块放在第二个磁盘上，并且第 n 块在第(n mod number_of_disks)磁盘上，等等。这意味着，如果你的正常数据大小于分割大小(或完美地排列过)，你将得到较好一些的性能。注意，分割是否很依赖于 OS 和分割大小。因此用不同的分割大小测试你的应用程序。见 10.8 使用你自己的基准。注意对分割的速度差异很依赖于参数，取决于你如何分割参数和磁盘数量，你可以得出以数量级的不同。注意你必须选择为随机或顺序存取优化。

为了可靠，你可能想要使用袭击 RAID 0+1(分割+镜像)，但是在这种情况下，你将需要 2*N 个驱动器来保存 N 个驱动器的数据。如果你有钱，这可能是最好的选择！然而你也可能必须投资一些卷管理软件投资以高效地处理它。

一个好选择是让稍重要的数据(它能再生)上存在 RAID 0 磁盘上，而将确实重要的数据(像主机信息和日志文件)存在一个 RAID 0+1 或 RAID N 磁盘上。如果因为更新奇偶位你有许多写入，RAID N 可能是一个问题。

你也可以对数据库使用的文件系统设置参数。一个容易的改变是以 noatime 选项挂装文件系统。这是它跳过更新在 inode 中的最后访问时间，而且这将避免一些磁盘寻道。

8.5.2 硬件问题

可利用硬件更有效地改善服务器的性能：

1、在机器中安装更多的内存。这样能够增加服务器的高速缓存和缓冲区的尺寸，使服务器更经常地使用存放在内存中的信息，降低从磁盘取信息的要求。

2、如果有足够的 RAM 使所有交换在内存文件系统中完成，那么应该重新配置系统，去掉所有磁盘交换设置。否则，即使有足以满足交换的 RAM，某些系统仍然要与磁盘进行交换。

3、增加更快的磁盘以减少 I/O 等待时间。寻道时间是这里决定性能的主要因素。逐字地移动磁头是很慢的，一旦磁头定位，从磁道读块则较快。

在不同的物理设备上设法重新分配磁盘活动。如果可能，应将您的两个最繁忙的数据库存放在不同的物理设备上。请注意，使用同一物理设备上的不同分区是不够的。这样没有帮助，因为它们仍将争用相同的物理资源（磁盘头）。移动数据库的过程在第 10 章中介绍。

4、在将数据重新放到不同设备之前，应该保证了解该系统的装载特性。如果在特定的物理设备上已经有了某些特定的主要活动，将数据库放到该处实际上可能会使性能更坏。例如，不要把数据库移到处理大量 Web 通信的 Web 服务器设备上。

5、在设置 MySQL 时，应该配置其使用静态库而不是共享库。使用共享库的动态二进制系统可节省磁盘空间，但静态二进制系统更快（然而，如果希望装入用户自定义的函数，则不能使用静态二进制系统，因为 UDF 机制依赖于动态连接）。

8.5.3 服务器参数的选择

服务器有几个能够改变从而影响其操作的参数（或称变量）。系统变量的当前值可以通过执行 mysqladmin varibles 命令来检查，其中几个参数主要与查询有关，有必要在此提一下：

delayed_queue_size

此参数在执行其他 INSERT DELAYED 语句的客户机阻塞以前，确定来自 INSERT DELAYED 语句的放入队列的行的数目。增加这个参数的值使服务器能从这种请求中接收更多的行，因而客户机可以继续执行而不阻塞。

key_buffer_size

此参数为用来存放索引块的缓冲区尺寸。如果内存多，增加这个值能节省索引创建和修改的时间。较大的值使 MySQL 能在内存中存储更多的索引块，这样增加了在内存中找到键值而不用读磁盘块的可能性。

在 MySQL 3.23 版及以后的版本中，如果增加了键缓冲区的尺寸，可能还希望用 --init-file 选项启动服务器。这样能够指定一个服务器启动时执行的 SQL 语句文件。如果有想要存放在内存中的只读表，可将它们拷贝到索引查找非常快的 HEAP 表。

back_log

引入客户机连接请求的数量，这些请求在从当前客户机中处理时排队。如果你有一个很忙的站点，可以增加改变量的值。

8.5.4 编译和链接怎样影响 MySQL 的速度

大多数下列测试在 Linux 上并用 MySQL 基准进行的，但是它们应该对其他操作系统和工作负载给出一些指示。

当你用-static 链接时，你得到最快的可执行文件。使用 Unix 套接字而非 TCP/IP 连接一个数据库也可给出好一些的性能。

在 Linux 上，当用 pgcc 和-O6 编译时，你将得到最快的代码。为了用这些选项编译“sql_yacc.cc”，你需要大约 200M 内存，因为 gcc/pgcc 需要很多内存使所有函数嵌入（inline）。在配置 MySQL 时，你也应该设定 CXX=gcc 以避免包括 libstdc++库(它不需要)。

只通过使用一个较好的编译器或较好的编译器选项，在应用中你能得到一个 10-30%

的加速。如果你自己编译 SQL 服务器，这特别重要！

在 Intel 上，你应该例如使用 pgcc 或 Cygnus CodeFusion 编译器得到最大速度。我们已经测试了新的 Fujitsu 编译器，但是它是还没足够不出错来优化编译 MySQL。

这里是我们做过的一些测量表：

- 如果你以-O6 使用 pgcc 并且编译任何东西，mysqld 服务器是比用 gcc 快 11%（用字符串 99 的版本)。
- 如果你动态地链接(没有-static)，结果慢了 13%。注意你仍能使用一个动态连接的 MySQL 库。只有服务器对性能是关键的。
- 如果你使用 TCP/IP 而非 Unix 套接字，结果慢 7.5%。
- 在一个 Sun SPARCstation 10 上，gcc2.7.3 是比 Sun Pro C++ 4.2 快 13%。
- 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%。以更多的负载/cpus，差别应该变得更大。

由 TcX 提供的 MySQL-Linux 的分发用 pgcc 编译并静态链接。

8.5.5 总结

本节简单介绍了如何在服务器级优化数据库的性能，以及提高数据库性能涉及到的硬件问题。选择一个尽量快的系统，使用 RAID 磁盘阵列是非常容易想到的方法。

对于数据库守护程序，既可以在编译时就提供合适的参数，也可以在选项文件中提供需要优化的参数。

思考题

1、 使用 ANALYSE 过程分析语句 SELECT * FROM pet，判断是否有必要改变列的类型。

2、 现在让你需要优化表 pet，可以用那些手段做到？

思考题参考答案

第二章

2、在 Win32 平台上运行分发包中的 setup.exe 程序，安装 MySQL 系统后，就自动初始化授权表。

对于 Unix 平台，运行脚本 mysql_install_db，之前应该首先切换到启动 mysqld 的用户，例如 mysql：

#su mysql

$mysql_install_db

安装授权表之后修改 root 用户的密码，由于初始化后存在两个从 localhost 和任意主机连接的 root 用户，所以推荐使用下面的办法：

$mysql -u root mysql　(由于现在不存在密码，因此不比提供-p 选项)

mysql> UPDATE user SET password=PASSWORD("mypass") WHERE User=root;

3、使用如下命令更改密码：

shell> mysqladmin -u root -p password 'newpass'

Enter Password:*******

出现 Enter Password 的提示后输入原来的密码 oldpass 即可。

读者可以尝试其它所有本章介绍的方法。

4、首先以 root 用户的身份连接到服务器：

shell> mysql -u root -p

Enter password:*******

出现 Enter password 提后输入 root 用户的密码，然后即进入 mysql 客户机的交互模式，可以看到下面的提示：

Welcome to the MySQL monitor.　Commands end with ; or \g.

Your MySQL connection id is 4 to server version: 3.23.25-beta-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer

mysql>

然后发布查询，直接键入题目中的语句：

mysql> SELECT User,Host FROM mysql.user;

应该有类似于下面的结果：

```
+------+---------+
| User | Host    |
```

```
+------+----------+
| root  | %         |
| admin | localhost |
| root  | localhost |
+------+----------+
```

5、在全局选项文件（Unix 上位于/etc/my.cnf，Windows 上位于 c:\my.cnf）中加入下面的几行：

```
[mysql]
user=root
password
```

然后在运行 mysql 客户程序，就不必提供参数：

```
shell> mysql
Enter Password:********
```

你可以查看当前的连接，以确定是否是如此：

```
mysql> SHOW PROCESSLIST;
+---+-----+----------+-----+--------+-----+------+-----------------+
| Id | User | Host      | db   | Command | Time | State | Info            |
+---+-----+----------+-----+--------+-----+------+-----------------+
|  4 | root | localhost | NULL | Query   | 0    | NULL  | SHOW PROCESSLIST |
+---+-----+----------+-----+--------+-----+------+-----------------+
```

第三章

2、连接服务器的命令为：

```
shell>mysql -h database.horst.zoo.net -u root -p test
Enter Password:******
```

如果使用选项文件，将下面几行加入全局选项文件中：

```
[mysql]
host=database.horst.zoo.net
user=root
password
```

然后可以直接运行 mysql，不比提供连接参数：

```
shell> mysql
Enter Password:*******
```

3、创建表的语句为：

```
CREATE TABLE pet
(
    name CHAR(30),
    owner CHAR(30),
    species CHAR(10),
    sex ENUM("M","F") NOT NULL,
    birth DATE,
    death DATE
)
```

4、如下录入文件 pet.txt，段与段用制表符分隔

```
Fluffy    Harold    cat    F    1993-02-04    \N
Claws     Gwen      cat    M    1994-03-17    \N
Buffy     Harold    dog    F    1989-05-13    \N
Chirpy    Gwen      bird   F    1998-09-11    \N
FangBenny      dog    M    1990-08-27    \N
Bowser    Diane     dog    M    1990-08-31    1995-07-29
Whistler  Gwen      bird   \N   1997-12-09    \N
Slim Benny     snake     M    1996-04-29    \N
Puffball  Diane     hamster   F    1999-03-30    \N
```

然后连接服务器，发布查询：

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

由于缺省时，LOAD DATA 语句以特殊字符'\n'作为记录的结束，这和 Unix 系统的文本编辑器以'\n'为换行符是一致的，但是在 Windows 系统中，换行符是'\r\n'，因此，如果是在 Windows 系统编辑的文件，那么上面的语句不能成功的录入数据，要做如下的修改：

```
mysql> LOAD DATA INFILE "pet.txt" INTO TABLE pet
    -> LINES TERMINATED BY '\r\n';
```

如果使用 mysqlimport 程序，相应的命令行为：

```
shell> mysqlimport test 'pet.tt'
```

或者使用：

```
shell> mysqlimport test 'pet.tt' --lines-terminated-by='\r\n'
```

第四章

1、 创建表：

```
mysql> CREATE TABLE ex4
```

```
    -> (
    -> data FLOAT,
    -> birth DATETIME
    -> );
```

录入数据：

```
mysql> INSERT ex4 VALUES(RAND(),NOW());
```

多录入几个数据，现有的数据为：

```
mysql> select * from ex4;
+----------+---------------------+
| data     | birth               |
+----------+---------------------+
| 0.830329 | 2001-01-01 21:21:10 |
| 0.531143 | 2001-01-01 21:21:12 |
| 0.164729 | 2001-01-01 21:21:13 |
| 0.230213 | 2001-01-01 21:21:14 |
+----------+---------------------+
```

data 列的平均值：

```
mysql> SELECT AVG(data) AS average FROM ex4;
+-----------------+
| average         |
+-----------------+
| 0.43910377845168 |
+-----------------+
```

data 列的总和：

```
mysql> SELECT SUM(data) AS "sum of data" FROM ex4;
+---------------+
| sum of data   |
+---------------+
| 1.7564151138067 |
+---------------+
mysql> SELECT MAX(data) AS "max of data",MIN(data) AS "min of data"
    -> FROM ex4;
+-----------------+-----------------+
| max of data     | min of data     |
+-----------------+-----------------+
| 0.83032947778702 | 0.16472874581814 |
+-----------------+-----------------+
```

data 列降序排列：

mysql> SELECT * FROM ex4 ORDER BY data DESC;

```
+----------+---------------------+
| data     | birth               |
+----------+---------------------+
| 0.830329 | 2001-02-18 21:21:10 |
| 0.531143 | 2001-02-18 21:21:12 |
| 0.230213 | 2001-02-18 21:21:14 |
| 0.164729 | 2001-02-18 21:21:13 |
+----------+---------------------+
```

2、 使用标准 SQL 模式匹配：

mysql> select * from ex4 where birth like "2001-01-01%";

```
+----------+---------------------+
| data     | birth               |
+----------+---------------------+
| 0.830329 | 2001-01-01 21:21:10 |
| 0.531143 | 2001-01-01 21:21:12 |
| 0.164729 | 2001-01-01 21:21:13 |
| 0.230213 | 2001-01-01 21:21:14 |
+----------+---------------------+
```

使用扩张正则表达式模式匹配：

mysql> select * from ex4 where birth REGEXP "^2001-01-01";

```
+----------+---------------------+
| data     | birth               |
+----------+---------------------+
| 0.830329 | 2001-01-01 21:21:10 |
| 0.531143 | 2001-01-01 21:21:12 |
| 0.164729 | 2001-01-01 21:21:13 |
| 0.230213 | 2001-01-01 21:21:14 |
+----------+---------------------+
```

3、 为 student 表创建索引：

```
mysql> ALTER TABLE student
    -> ADD PRIMARY KEY(id),
    -> ADD INDEX mark(english,chinese,history);
```

这样查看创建的索引：

mysql> SHOW INDEX FROM student;

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

```
| Table   | Non_unique | Key_name | Seq_in_index | Column_name |...
+---------+------------+----------+--------------+-------------+---
| student |          0 | PRIMARY  |            1 | id          |...
| student |          1 | mark     |            1 | english     |...
| student |          1 | mark     |            2 | chinese     |...
| student |          1 | mark     |            3 | history     |...
+---------+------------+----------+--------------+-------------+---
```

为 pet 表创建索引，这次使用另一种方法：

mysql> CREATE INDEX name ON pet (name(10),owner(10));

查看创建的索引的情况：

mysql> SHOW INDEX FROM pet；

```
+-------+------------+----------+--------------+-------------+-
| Table | Non_unique | Key_name | Seq_in_index | Column_name |...
+-------+------------+----------+--------------+-------------+-
| pet   |          1 | name     |            1 | name        |...
| pet   |          1 | name     |            2 | owner       |...
+-------+------------+----------+--------------+-------------+-
```

4、删除 pet 表的索引：

mysql> DROP INDEX name ON pet;

或者可以：

mysql> ALTER TABLE pet DROP INDEX name;

然后再用 SHOW 语句产看可知

mysql> show index from pet;

Empty set (0.01 sec)

第五章

1、 shell> mysqldump --all-database --opt > data.sql

2、 shell> mysqldump --tab=”./bak” test

3、 创建数据库 test1

mysql> CREATE DATABASE test1;

恢复表的结构

shell> mysql test1< ./bak/pet.sql

恢复数据：

mysql> mysqlimport test1 ./bak/pet.txt

在 Win32 平台上，你必须一个个的恢复文件，在 Unix 平台上，可以通过 shell 的文件匹配的功能，简化命令行：

```
$mysql test1 <./bak/*.sql
$mysqlimport test1 ./bak/*bak
```

4、相应更新日志的内容是：

```
create database test1;
use test1;
CREATE TABLE pet (
  name char(32),
  owner char(32),
  species char(32),
  sex enum('F','M'),
  birth date,
  death date
);
use test1;
LOAD DATA  INFILE 'D:/Server/mysql/bin/./bak/pet.txt' INTO TABLE pet;
```

常规日志的内容由于过长，不再列出。常规日志列出了所有客户机发布的查询，不仅包括 mysql，还包括 mysqladmin，mysqldump，mysqlimport 等，你可以观察这些客户机完成任务所需要执行的查询。

5、 为了说明下面的操作，可以先进行几条 SQL 操作：

```
mysql> INSERT pet SET name="tes";
mysql> DELETE FROM pet WHERE name="tes";
```

相应的在更新日志中增加的记录为：

```
use test;
INSERT pet SET name="tes";
DELETE FROM pet WHERE name="tes";
```

然后我们删除 test1 数据库：

```
mysql> drop database test10;
```

相应的更新日志中的内容为：

```
drop database test10;
```

为了恢复数据库，假设更新日志的文件名是 wxy.058，假定该日志包括第四题中更新日志的内容。

首先删除更新日志中删除数据库的记录，然后用下面的语句恢复数据库表：

```
shell> mysql --one-databse test1< ../data/wxy.058
```

你可以看到数据库表得到了恢复，你也可以查看常规日志，以确定在这个过程中，test1 数据库之外的 SQL 语句被忽略了。

第六章

1、首先对表 pet 进行独锁定：

mysql>LOCK TABLE pet READ;

清空缓存：

mysql>FLUSH TABLES;

检查表：

shell> myisamchk /path/to/pet (在我的主机上，为../data/test/pet)

释放表锁

mysql>UNLOCK TABLES;

2、创建数据库的方法是：

mysql> CREATE DATABASE test2;

mysql> CREATE TABLE pet SELECT * FROM test.pet;

恢复数据库的方法是：

由于表的描述文件被删除，因此我们可以创建一个具有相同结构的表：

mysql> CREATE TABLE pet1(…);

为了节省时间，由于本题只是一个演示，所以你还可以从 test.pet 中恢复表的结构：

mysql> CREATE TABLE pet1 SELECT * FROM test.pet WHERE 0;

然后，进入数据库目录，把 pet1.frm 文件复制为 pet.frm 文件，完成操作即可。

3、备份过程在正文中有详细叙述。此处略。

第七章

1、原因是：

先考虑一下 mysql_install_db 如何建立初始权限表和服务器如何使用 user 表记录匹配客户连接。在你用 mysql_install_db 初始化你的数据库时，它创建类似这样的 user 表：

Host	User
localhost	
horst.zoo.net	
localhost	
horst.zoo.net	root
	root

头两个记录允许 root 指定 localhost 或主机名连接本地服务器，后两个允许匿名用户从本地连接。当增加 fox 用户后，

Host	User

localhost
horst.zoo.net
localhost
horst.zoo.net
%.zoo.netroot
root
fox

在服务器启动时，它读取记录并排序它们（首先按主机，然后按主机上的用户），越具体越排在前面：

Host User
localhost
localhost
horst.zoo.net
horst.zoo.net
%.zoo.net root
root
fox

有 localhost 的两个记录排在一起，而对 root 的记录排在第一，因为它比空值更具体。horst.zoo.net 的记录也类似。所有这些均是没有任何通配符的字面上的 Host 值，所以它们排在对 fox 记录的前面，特别是匿名用户排在 fox 之前。

结果是在 fox 试图从 localhost 连接时，Host 列中的一个空用户名的记录在包含%.snake.net 的记录前匹配。该记录的口令是空的，因为缺省的匿名用户没有口令。因为在 fox 连接时指定了一个口令，由一个错配且连接失败。

这里要记住的是，虽然用通配符指定用户可以从其连接的主机是很方便。但你从本地主机连接时会有问题，只要你在 table 表中保留匿名用户记录。

一般地，建议你删除匿名用户记录：

mysql> DELETE FROM user WHERE User="";

更进一步，同时删除其他授权表中的任何匿名用户，有 User 列的表有 db、tables_priv 和 columns_priv。

2、应该给用户分配全局的 RELOAD、SHUTDOWN 权限，使用 GRANT 语句：

```
mysql> GRANT RELOAD,SHUTDOWN ON *.* TO admin@localhost
    -> INDENTIFIED BY “mypass”;
```

使用直接修改授权表的方法：

```
mysql> INSERT mysql.user(User,Host,Password,Reload_priv,Shutdown_priv)
    -> VALUES(“amin”,”localhost”,PASSWORD(“mypass”),Y,Y);
```

3、由于该用户只在 user 表中有权限记录，所以只要删除该记录即可：

```
mysql> DELETE FROM mysql.user WHERE User=”admin”;
```

4、 创建数据库

```
mysql> CREATE DATABASE mark;
```

创建用户并授权：

```
mysql> GRANT SELECT,DROP,CREATE,INSERT,
    -> UPDATE,DELETE,INDEX,ALTER
    -> ON mark.*
    -> TO teacher
    -> IDENTIFIED BY "mypass";
```

也可以不使用 GRANT 语句，直接修改授权表：

```
mysql> INSERT mysql.user(User,Password,Host)
    -> VALUES(“teacher”,PASSWORD(“mypass”),”%”);
mysql> INSERT mysql.db VALUES
    -> ("%","mark","teacher","Y","Y","Y","Y","Y","Y","N","N","Y","Y");
```

第八章

1、由于没有非常好的例子，这里只是做一个简单的分析：

例如，对于表 pet：

```
mysql>SELECT species FROM pet PROCEDURE ANALYSE();
```

输出有 10 列，对于列 Optimal_fieldtype 是建议的优化列类型，都是 ENUM 类型，在这里，species 的数值有限，故可以改为输出所建议的类型：

```
ENUM('bird','cat','dog','hamster', 'snake') NOT NULL
```

2、可以用两种方法

使用 SQL 语句：

```
mysql> LOCK TABLE pet WRITE;
mysql>FLUSH TABLES;
mysql> OPTIMIZE TABLE tbl_name;
mysql> UNLOCK TABLES;
```

使用 myisamchk 维护程序：

```
mysql>LOCK TABLE pet WRITE;
mysql>FLUSH TABLES;
shell>myisamchk --recover ../data/test/pet
mysql>FLUSH TABLES;
mysql>UNLOCK TABLES;
```

附录一

MySQL 语言参考

数据类型参考

用户变量参考

列类型参考

SQL 语句参考

1.1 数据类型参考：怎么写字符串和数字

1.1.1 字符串

一个字符串是一个字符序列，由单引号(“'”)或双引号(“"”)字符(后者只有你不在ANSI 模式运行)包围。例如：

'a string'

"another string"

在字符串内，某个顺序有特殊的意义。这些顺序的每一个以一条反斜线(“\”)开始，称为转义字符。MySQL 识别下列转义字符：

\0

一个 ASCII 0 (NUL)字符。

\n

一个新行符。

\t

一个定位符。

\r

一个回车符。

\b

一个退格符。

\'

一个单引号(“'”)符。

\"

一个双引号(“"”)符。

\\

一个反斜线(“\”)符。

\%

一个“%”符。它用于在正文中搜索“%”的文字实例，否则这里“%”将解释为一个通配符。

_

一个“_”符。它用于在正文中搜索“_”的文字实例，否则这里“_”将解释为一个通配符。

注意，如果你在某些正文环境中使用“\%”或“\%_”，这些将返回字符串“\%”和“_”而不是“%”和“_”。

有几种方法在一个字符串内包括引号：

- 一个字符串内用“'”加引号的“'”可以被写作为“''”。
- 一个字符串内用“"”加引号的“"”可以被写作为“""”。

- 你可以把一个转义字符（“\”）放在引号前面。
- 一个字符串内用“"”加引号的“'”不需要特殊对待而且不必被重复或转义。同理，一个字符串内用“'”加引号的与“"”也不需要特殊对待。

下面显示的 SELECT 演示引号和转义如何工作：

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

如果你想要把二进制数据插入到一个 BLOB 列，下列字符必须由转义序列表示：

NUL

ASCII 0。你应该用'\0'（一个反斜线和一个 ASCII '0'）表示它。

\

ASCII 92，反斜线。用'\\'表示。

'

ASCII 39，单引号。用“\'”表示。

"

ASCII 34，双引号。用“\"”表示。

如果你写 C 代码，你可以使用 C API 函数 mysql_escape_string()来为 INSERT 语句转义字符。在 Perl 中，你可以使用 DBI 包中的 quote 方法变换特殊的字符到正确的转义序列。

你应该在任何可能包含上述任何特殊字符的字符串上使用转义函数！

1.1.2 数字

整数表示为一个数字顺序。浮点数使用“.”作为一个十进制分隔符。这两种类型的数字可以前置“-”表明一个负值。

有效整数的例子：

1221

0

-32

有效浮点数的例子：

294.42

-32032.6809e+10

148.00

一个整数可以在浮点上下文使用；它解释为等值的浮点数。

1.1.3 十六进制值

MySQL 支持十六进制值。在数字上下文，它们表现类似于一个整数(64 位精度)。在字符串上下文，它们表现类似于一个二进制字符串，这里每一对十六进制数字被变换为一个字符。

```
mysql> SELECT 0xa+0
    -> 10
mysql> select 0x5061756c;
    -> Paul
```

十六进制字符串经常被 ODBC 使用，给出 BLOB 列的值。

1.1.4 NULL 值

NULL 值意味着“无数据”并且不同于例如数字类型的 0 为或字符串类型的空字符串。

NULL 值的概念是造成 SQL 的新手的混淆的普遍原因，他们经常认为 NULL 是和一个空字符串''的一样的东西。不是这样的！例如，下列语句是完全不同的：

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ("");
```

两个语句把值插入到 phone 列，但是第一个插入一个 NULL 值而第二个插入一个空字符串。第一个的含义可以认为是“电话号码不知道”，而第二个则可意味着“她没有电话”。

在 SQL 中，NULL 值在于任何其他值甚至 NULL 值比较时总是假的（FALSE）。包含 NULL 的一个表达式总是产生一个 NULL 值，除非在包含在表达式中的运算符和函数的文档中指出。在下列例子，所有的列返回 NULL：

```
mysql> SELECT NULL,1+NULL,CONCAT('Invisible',NULL);
```

如果你想要寻找值是 NULL 的列，你不能使用=NULL 测试。下列语句不返回任何行，因为对任何表达式，expr = NULL 是假的：

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

要想寻找 NULL 值，你必须使用 IS NULL 测试。下例显示如何找出 NULL 电话号码和空的电话号码：

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
mysql> SELECT * FROM my_table WHERE phone = "";
```

在 MySQL 中，就像很多其他的 SQL 服务器一样，你不能索引可以有 NULL 值的列。你必须声明这样的列为 NOT NULL，而且，你不能插入 NULL 到索引的列中。

当用 LOAD DATA INFILE 读取数据时，空列用"更新。如果你想要在一个列中有 NULL 值，你应该在文本文件中使用\N。字面上的词'NULL'也可以在某些情形下使用。见 1.16 LOAD DATA INFILE 句法。

当使用 ORDER BY 时，首先呈现 NULL 值。如果你用 DESC 以降序排序，NULL 值最后显示。当使用 GROUP BY 时，所有的 NULL 值被认为是相等的。

为了有助于 NULL 的处理，你能使用 IS NULL 和 IS NOT NULL 运算符和 IFNULL() 函数。

对某些列类型，NULL 值被特殊地处理。如果你将 NULL 插入表的第一个 TIMESTAMP 列，则插入当前的日期和时间。如果你将 NULL 插入一个 AUTO_INCREMENT 列，则插入顺序中的下一个数字。

当使用文本文件导入或导出格式(LOAD DATA INFILE, SELECT ... INTO OUTFILE)时，NULL 可以用\N 表示。见 1.16 LOAD DATA INFILE 句法。

1.1.5 数据库、表、索引、列和别名的命名

数据库、表、索引、列和别名的名字都遵守 MySQL 同样的规则:

注意，从 MySQL3.23.6 开始规则改变了，此时我们引入了用'引用的标识符(数据库、表和列命名)（如果你以 ANSI 模式运行，"也将用于引用标识符）。

表附 1-1　命名规则

标识符	最大长度	允许的字符
数据库	64	在一个目录名允许的任何字符，除了/.
表	64	在文件名中允许的任何字符，除了/或.
列	64	所有字符
别名	255	所有字符

注意，除了以上，你在一个标识符中不能有 ASCII(0)或 ASCII(255)。

注意，如果标识符是一个限制词或包含特殊字符，当你使用它时，你必须总是用`引用它：

```
SELECT * from `select` where `select`.id > 100;
```

在 MySQL 的先前版本，命名规则如下：

- 一个名字可以包含来自当前字符集的数字字母的字符和“_”和“$”。缺省字符集是 ISO-8859-1 Latin1；这可以通过重新编译 MySQL 来改变。见 9.1.1 用于数据和排序的字符集。
- 一个名字可以以在一个名字中合法的任何字符开始。特别地，一个名字可以以一个数字开始(这不同于许多其他的数据库系统!)。然而，一个名字不能仅仅由数字

组成。

- 你不能在名字中使用“.”，因为它被用来扩充格式，你能用它引用列(见下面)。

建议你不使用象 1e 这样的名字，因为一个表达式如 1e+1 是二义性的。它可以解释为表达式 1e + 1 或数字 1e+1。

在 MySQL 中，你能使用下列表格的任何一种引用列：

表附 1-2 引用列的方法

列引用	含义
col_name	来自于任意表的列 col_name，用于包含该表的一个列的查询中
tbl_name.col_name	来自当前的数据库的表 tbl_name 的列 col_name
db_name.tbl_name.col_name	行列 col_name 从表格 tbl_name 数据库 db_name。这个形式在 MySQL3.22 或以后版本可用。
`column_name`	是一个关键词或包含特殊字符的列。

在一条语句的列引用中，你不必指定一个 tbl_name 或 db_name.tbl_name 前缀，除非引用会有二义性。例如，假定表 t1 和 t2，每个均包含列 c，并且你用一个使用 t1 和 t2 的 SELECT 语句检索 c。在这种情况下，c 有二义性，因为它在使用表的语句中不是唯一的，因此你必须通过写出 t1.c 或 t2.c 来指明你想要哪个表。同样，如果你从数据库 db1 中一个表 t 和在数据库 db2 的一个表 t 检索，你必须用 db1.t.col_name 和 db2.t.col_name 引用这些数据表的列。

句法.tbl_name 意味着在当前的数据库中的表 tbl_name，该句法为了 ODBC 的兼容性被接受，因为一些 ODBC 程序用一个“.”字符作为数据库表名的前缀。

1.1.5.1 名字的大小写敏感性

在 MySQL 中，数据库和表对应于在那些目录下的目录和文件，因而，内在的操作系统的敏感性决定数据库和表命名的大小写敏感性。这意味着数据库和表名在 Unix 上是区分大小写的，而在 Win32 上忽略大小写。

注意：在 Win32 上，尽管数据库和表名是忽略大小写的，你不应该在同一个查询中使用不同的大小写来引用一个给定的数据库和表。下列查询将不工作，因为它作为 my_table 和作为 MY_TABLE 引用一个表：

```
mysql> SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

列名在所有情况下都是忽略大小写的。

表的别名是区分大小写的。下列查询将不工作，：因为它用 a 和 A 引用别名：

```
mysql> SELECT col_name FROM tbl_name AS a
```

```
WHERE a.col_name = 1 OR A.col_name = 2;
```

列的别名是忽略大小写的。

1.2 用户变量

MySQL 支持线程特定的变量，用@variablename 句法。一个变量名可以由当前字符集的数字字母字符和“_”、“$”和“.”组成。缺省字符集是 ISO-8859-1 Latin1；这可以通过重新编译 MySQL 改变。

变量不必被初始化。缺省地，他们包含 NULL 并能存储整数、实数或一个字符串值。当线程退出时，对于一个线程的所有变量自动地被释放。

你可以用 SET 句法设置一个变量：

```
SET @variable= { integer expression | real expression | string expression }
[,@variable= ...].
```

你也可以用@variable:=expr 句法在一个表达式中设置一个变量：

```
select @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
+----------------------+------+------+------+
| @t1:=(@t2:=1)+@t3:=4 | @t1  | @t2  | @t3  |
+----------------------+------+------+------+
|                    5 |    5 |    1 |    4 |
+----------------------+------+------+------+
```

（这里，我们不得不使用:=句法，因为=是为比较保留的）

1.3 列类型

MySQL 支持大量的列类型，它可以被分为 3 类：数字类型、日期和时间类型以及字符串(字符)类型。本节首先给出可用类型的一个概述，并且总结每个列类型的存储需求，然后提供每个类中的类型性质的更详细的描述。概述有意简化，更详细的说明应该考虑到有关特定列类型的附加信息，例如你能为其指定值的允许格式。

由 MySQL 支持的列类型列在下面。下列代码字母用于描述中：

M

指出最大的显示尺寸。最大的合法的显示尺寸是 255 。

D

适用于浮点类型并且指出跟随在十进制小数点后的数码的数量。最大可能的值是 30，但是应该不大于 M-2。

方括号(“[”和“]”)指出可选的类型修饰符的部分。

注意，如果你指定一个了为 ZEROFILL，MySQL 将为该列自动地增加 UNSIGNED 属性。

TINYINT[(M)] [UNSIGNED] [ZEROFILL]

一个很小的整数。有符号的范围是-128 到 127，无符号的范围是 0 到 255。

SMALLINT[(M)] [UNSIGNED] [ZEROFILL]

一个小整数。有符号的范围是-32768 到 32767，无符号的范围是 0 到 65535。

MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]

一个中等大小整数。有符号的范围是-8388608 到 8388607，无符号的范围是 0 到 16777215。

INT[(M)] [UNSIGNED] [ZEROFILL]

一个正常大小整数。有符号的范围是-2147483648 到 2147483647，无符号的范围是 0 到 4294967295。

INTEGER[(M)] [UNSIGNED] [ZEROFILL]

这是 INT 的一个同义词。

BIGINT[(M)] [UNSIGNED] [ZEROFILL]

一个大整数。有符号的范围是-9223372036854775808 到 9223372036854775807，无符号的范围是 0 到 18446744073709551615。注意，所有算术运算用有符号的 BIGINT 或 DOUBLE 值完成，因此你不应该使用大于 9223372036854775807（63 位)的有符号大整数，除了位函数！注意，当两个参数是 INTEGER 值时，-、+和*将使用 BIGINT 运算！这意味着如果你乘 2 个大整数(或来自于返回整数的函数)，如果结果大于 9223372036854775807，你可以得到意外的结果。一个浮点数字，不能是无符号的，对一个单精度浮点数，其精度可以是<=24，对一个双精度浮点数，是在 25 和 53 之间，这些类型如 FLOAT 和 DOUBLE 类型马上在下面描述。FLOAT(X)有对应的 FLOAT 和 DOUBLE 相同的范围，但是显示尺寸和小数位数是未定义的。在 MySQL3.23 中，这是一个真正的浮点值。在更早的 MySQL 版本中，FLOAT(precision)总是有 2 位小数。该句法为了 ODBC 兼容性而提供。

FLOAT[(M,D)] [ZEROFILL]

一个小(单精密)浮点数字。不能无符号。允许的值是-3.402823466E+38 到 -1.175494351E-38，0 和 1.175494351E-38 到 3.402823466E+38。M 是显示宽度而 D 是小数的位数。没有参数的 FLOAT 或有<24 的一个参数表示一个单精密浮点数字。

DOUBLE[(M,D)] [ZEROFILL]

一个正常大小(双精密)浮点数字。不能无符号。允许的值是-1.7976931348623157E+308 到 -2.2250738585072014E-308 、 0 和 2.2250738585072014E-308 到 1.7976931348623157E+308。M 是显示宽度而 D 是小数位数。没有一个参数的 DOUBLE 或 FLOAT(X)（25 < = X < = 53）代表一个双精密浮点数字。

DOUBLE PRECISION[(M,D)] [ZEROFILL]

REAL[(M,D)] [ZEROFILL]

这些是 DOUBLE 同义词。

DECIMAL[(M[,D])] [ZEROFILL]

一个未压缩(unpack)的浮点数字。不能无符号。行为如同一个 CHAR 列：“未压缩”意味着数字作为一个字符串被存储，值的每一位使用一个字符。小数点，并且对于负数，“-”符号不在 M 中计算。如果 D 是 0，值将没有小数点或小数部分。DECIMAL 值的最大范围与 DOUBLE 相同，但是对一个给定的 DECIMAL 列，实际的范围可以通过 M 和 D 的选择被限制。如果 D 被省略，它被设置为 0。如果 M 被省掉，它被设置为 10。注意，在 MySQL3.22

里，M 参数包括符号和小数点。

NUMERIC(M,D) [ZEROFILL]

这是 DECIMAL 的一个同义词。

DATE

一个日期。支持的范围是'1000-01-01'到'9999-12-31'。MySQL 以'YYYY-MM-DD'格式来显示 DATE 值，但是允许你使用字符串或数字把值赋给 DATE 列。

DATETIME

一个日期和时间组合。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'格式来显示 DATETIME 值，但是允许你使用字符串或数字把值赋给 DATETIME 的列。

TIMESTAMP[(M)]

一个时间戳记。范围是'1970-01-01 00:00:00'到 2037 年的某时。MySQL 以 YYYYMMDDHHMMSS、YYMMDDHHMMSS、YYYYMMDD 或 YYMMDD 格式来显示 TIMESTAMP 值，取决于是否 M 是 14（或省略)、12、8 或 6，但是允许你使用字符串或数字把值赋给 TIMESTAMP 列。一个 TIMESTAMP 列对于记录一个 INSERT 或 UPDATE 操作的日期和时间是有用的，因为如果你不自己给它赋值，它自动地被设置为最近操作的日期和时间。你以可以通过赋给它一个 NULL 值设置它为当前的日期和时间。见 1.3.6 日期和时间类型。

TIME

一个时间。范围是'-838:59:59'到'838:59:59'。MySQL 以'HH:MM:SS'格式来显示 TIME 值，但是允许你使用字符串或数字把值赋给 TIME 列。

YEAR[(2|4)]

一个 2 或 4 位数字格式的年(缺省是 4 位)。允许的值是 1901 到 2155，和 0000（4 位年格式），如果你使用 2 位，1970-2069(70-69)。MySQL 以 YYYY 格式来显示 YEAR 值，但是允许你把使用字符串或数字值赋给 YEAR 列。(YEAR 类型在 MySQL3.22 中是新类型。)

CHAR(M) [BINARY]

一个定长字符串，当存储时，总是是用空格填满右边到指定的长度。M 的范围是 1 ～ 255 个字符。当值被检索时，空格尾部被删除。CHAR 值根据缺省字符集以大小写不区分的方式排序和比较，除非给出 BINARY 关键词。NATIONAL CHAR（短形式 NCHAR)是 ANSI SQL 的方式来定义 CHAR 列应该使用缺省字符集。这是 MySQL 的缺省。CHAR 是 CHARACTER 的一个缩写。

[NATIONAL] VARCHAR(M) [BINARY]

一个变长字符串。注意:当值被存储时,尾部的空格被删除(这不同于 ANSI SQL 规范)。M 的范围是 1 ～ 255 个字符。 VARCHAR 值根据缺省字符集以大小写不区分的方式排序和比较，除非给出 BINARY 关键词值。见 1.1.1 隐式列指定变化。 VARCHAR 是 CHARACTER VARYING 一个缩写。

TINYBLOB

TINYTEXT

一个 BLOB 或 TEXT 列，最大长度为 255(2^8-1)个字符。见 1.1.1 隐式列指定变化。

BLOB

TEXT

一个 BLOB 或 TEXT 列，最大长度为 65535(2^16-1)个字符。见 1.1.1 隐式列指定变化。

MEDIUMBLOB

MEDIUMTEXT

一个 BLOB 或 TEXT 列，最大长度为 16777215(2^24-1)个字符。见 1.1.1 隐式列指定变化。

LONGBLOB

LONGTEXT

一个 BLOB 或 TEXT 列，最大长度为 4294967295(2^32-1)个字符。见 1.1.1 隐式列指定变化

ENUM('value1','value2',...)

枚举。一个仅有一个值的字符串对象，这个值式选自与值列表'value1'、'value2'，...,或 NULL。一个 ENUM 最多能有 65535 不同的值。

SET('value1','value2',...)

一个集合。能有零个或多个值的一个字符串对象，其中每一个必须从值列表'value1', 'value2', ...选出。一个 SET 最多能有 64 个成员。

1.3.1 列类型存储需求

对于每个由 MySQL 支持的列类型的存储需求在下面按类列出。

数字类型

表附 1-3 数字类型的存储需求

列类型	需要的存储量
TINYINT	1 字节
SMALLINT	2 个字节
MEDIUMINT	3 个字节
INT	4 个字节
INTEGER	4 个字节
BIGINT	8 个字节
FLOAT(X)	4 如果 X < = 24 或 8 如果 25 < = X < = 53
FLOAT	4 个字节
DOUBLE	8 个字节
DOUBLE PRECISION	8 个字节
REAL	8 个字节

DECIMAL(M,D)	M 字节(D+2，如果 M < D)
NUMERIC(M,D)	M 字节(D+2，如果 M < D)

日期和时间类型

表附 1-4　日期和时间类型的存储需求

列类型	需要的存储量
DATE	3 字节
DATETIME	8 字节
TIMESTAMP	4 字节
TIME	3 字节
YEAR	1 字节

串类型

表附 1-5　串类型的存储要求

列类型	需要的存储量
CHAR(M)	M 字节，1<=M<=255
VARCHAR(M)	L+1 字节,在此 L<=M 和 1<=M<=255
TINYBLOB,TINYTEXT	L+1 字节,在此 L<2^8
BLOB,TEXT	L+2 字节,在此 L<2^16
MEDIUMBLOB,MEDIUMTEXT	L+3 字节,在此 L<2^24
LONGBLOB,LONGTEXT	L+4 字节,在此 L<2^32
ENUM('value1','value2',...)	1 或 2 个字节,取决于枚举值的数目(最大值 65535）
SET('value1','value2',...)	1，2，3，4 或 8 个字节,取决于集合成员的数量(最多 64 个成员）

VARCHAR 和 BLOB 和 TEXT 类型是变长类型，对于其存储需求取决于列值的实际长度(在前面的表格中用 L 表示)，而不是取决于类型的最大可能尺寸。例如，一个

VARCHAR(10)列能保存最大长度为 10 个字符的一个字符串，实际的存储需要是字符串的长度(L)，加上 1 个字节以记录字符串的长度。对于字符串'abcd'，L 是 4 而存储要求是 5 个字节。

BLOB 和 TEXT 类型需要 1，2，3 或 4 个字节来记录列值的长度，这取决于类型的最大可能长度。

如果一个表包括任何变长的列类型，记录格式将也是变长的。注意，当一个表被创建时，MySQL 可能在某些条件下将一个列从一个变长类型改变为一个定长类型或相反。见 1.1.1 隐式列指定变化。

一个 ENUM 对象的大小由不同枚举值的数量决定。1 字节被用于枚举，最大到 255 个可能的值；2 个字节用于枚举，最大到 65535 值。

一个 SET 对象的大小由不同的集合成员的数量决定。如果集合大小是 N，对象占据(N+7)/8 个字节，四舍五入为 1，2，3，4 或 8 个字节。一个 SET 最多能有 64 个成员。

1.3.2 数字类型

MySQL 支持所有的 ANSI/ISO SQL92 的数字类型。这些类型包括准确数字的数据类型(NUMERIC, DECIMAL, INTEGER,和 SMALLINT)，也包括近似数字的数据类型(FLOAT, REAL,和 DOUBLE PRECISION)。关键词 INT 是 INTEGER 的一个同义词，而关键词 DEC 是 DECIMAL 一个同义词。

NUMERIC 和 DECIMAL 类型被 MySQL 实现为同样的类型，这在 SQL92 标准允许。他们被用于保存值，该值的准确精度是极其重要的值，例如与金钱有关的数据。当声明一个类是这些类型之一时，精度和规模的能被(并且通常是)指定；例如：

salary DECIMAL(9,2)

在这个例子中，9(precision)代表将被用于存储值的总的小数位数，而 2(scale)代表将被用于存储小数点后的位数。因此，在这种情况下，能被存储在 salary 列中的值的范围是从-9999999.99 到 9999999.99。在 ANSI/ISO SQL92 中，句法 DECIMAL(p)等价于 DECIMAL(p,0)。同样，句法 DECIMAL 等价于 DECIMAL(p,0)，这里实现被允许决定值 p。MySQL 当前不支持 DECIMAL/NUMERIC 数据类型的这些变种形式的任一种。这一般说来不是一个严重的问题，因为这些类型的主要益处得自于明显地控制精度和规模的能力。

DECIMAL 和 NUMERIC 值作为字符串存储，而不是作为二进制浮点数，以便保存那些值的小数精度。一个字符用于值的每一位、小数点(如果 scale>0)和“-”符号(对于负值)。如果 scale 是 0，DECIMAL 和 NUMERIC 值不包含小数点或小数部分。

DECIMAL 和 NUMERIC 值得最大的范围与 DOUBLE 一样，但是对于一个给定的 DECIMAL 或 NUMERIC 列，实际的范围可由制由给定列的 precision 或 scale 限制。当这样的列赋给了小数点后面的位超过指定 scale 所允许的位的值，该值根据 scale 四舍五入。当一个 DECIMAL 或 NUMERIC 列被赋给了其大小超过指定(或缺省的）precision 和 scale 隐含的范围的值，MySQL 存储表示那个范围的相应的端点值。

作为对 ANSI/ISO SQL92 标准的扩展，MySQL 也支持上表所列的整型类型 TINYINT、MEDIUMINT 和 BIGINT。另一个扩展是 MySQL 支持可选地指定一个整型值显示的宽度，

用括号跟在基本关键词之后(例如，INT(4))。这个可选的宽度指定被用于其宽度小于列指定宽度的值得左填补显示，但是不限制能在列中被存储的值的范围，也不限制值将被显示的位数，其宽度超过列指定的宽度。当与可选的扩展属性 ZEROFILL 一起使用时，缺省的空格填补用零代替。例如，对于声明为 INT(5) ZEROFILL 的列，一个为 4 的值作为 00004 被检索。注意，如果你在一个整型列存储超过显示宽度的更大值，当 MySQL 对于某些复杂的联结(join)生成临时表时，你可能会遇到问题，因为在这些情况下，MySQL 相信数据确实适合原来的列宽度。

所有的整型类型可以有一个可选(非标准的)属性 UNSIGNED。当你想要在列中仅允许正数并且你需要一个稍大一点的列范围，可以使用无符号值。

FLOAT 类型被用来标示近似数字的数据类型。ANSI/ISO SQL92 标准允许一个可选的精度说明(但不是指数的范围)，跟在关键词 FLOAT 后面的括号内位数。MySQL 实现也支持这个可选的精度说明。当关键词 FLOAT 被用于一个列类型而没有精度说明时，MySQL 使用 4 个字节存储值。一个变种的句法也被支持，在 FLOAT 关键词后面的括号给出 2 个数字。用这个选项，第一个数字继续表示在字节计算的值存储需求，而第二个数字指定要被存储的和显示跟随小数点后的位数(就象 DECIMAL 和 NUMERIC)。当 MySQL 要求为这样一个列，一个小数点后的小数位超过列指定的值，存储值时，该值被四舍五入，去掉额外的位。

REAL 和 DOUBLE PRECISION 类型不接受精度说明。作为对 ANSI/ISO SQL92 标准的扩展，MySQL 识别出 DOUBLE 作为 DOUBLE PRECISION 类型的一个同义词。与 REAL 精度比用于 DOUBLE PRECISION 的更小的标准要求相反，MySQL 实现了两种，作为 8 字节双精度浮点值(当运行不是“Ansi 模式”时)。为了最大的移植性，近似数字的数据值的存储所需代码应该使用没有精度或小数位数说明的 FLOAT 或 DOUBLE PRECISION。

当要求在数字的列存储超出该列类型允许的范围的值时，MySQL 剪切该值到范围内的正确端点值并且存储剪切后的结果值。

例如，一个 INT 列的范围是-2147483648 到 2147483647。如果你试图插入-9999999999 到一个 INT 列中，值被剪切到范围的低部端点，并存储-2147483648。同样，如果你试图插入 9999999999，2147483647 被存储。

如果 INT 列是 UNSIGNED，列的范围的大小是相同的，但是它的端点移到了 0 和 4294967295。如果你试图存储-9999999999 和 9999999999，在列被存储的值变为 0 和 4294967296。

对于 ALTER TABLE、LOAD DATA INFILE、UPDATE 和多行 INSERT 语句，由于剪切所发生的变换作为“警告”被报告。

1.3.3 日期和时间类型

日期和时间类型是 DATETIME、DATE、TIMESTAMP、TIME 和 YEAR。这些的每一个都有合法值的一个范围，而“零”当你指定确实不合法的值时被使用。注意，MySQL 允许你存储某个“不严格地”合法的日期值，例如 1999-11-31，原因我们认为它是应用程序的责任来处理日期检查，而不是 SQL 服务器。为了使日期检查更“快”，MySQL 仅检查

月份在 0-12 的范围，天在 0-31 的范围。上述范围这样被定义是因为 MySQL 允许你在一个 DATE 或 DATETIME 列中存储日期，这里的天或月是零。这对存储你不知道准确的日期的一个生日的应用程序来说是极其有用的，在这种情况下，你简单地存储日期象 1999-00-00 或 1999-01-00。（当然你不能期望从函数如 DATE_SUB()或 DATE_ADD()得到类似以这些日期的正确值）。

当用日期和时间工作时，这里是的一些要记住的一般考虑：

- MySQL 对一个给定的日期或时间类型以标准的格式检索，但是它试图为你提供的值解释成许多格式(例如，当你指定一个值被赋给或与比较一个日期或时间类型时)，但是只支持有在下列小节描述的格式。期望你提供合法的值，并且如果你以其他格式使用这些值，可能造成无法预料的结果。
- 尽管 MySQL 试图以多种格式解释值，但它总是期望日期值的年份部分在最左面，日期必须以年-月-日的顺序给出(例如，'98-09-04')，而不是以其他地方常用的月-日-年或日-月-年的次序(例如，'09-04-98'、'04-09-98')。
- 如果一个值在数字的上下文环境中被使用，MySQL 自动变换一个日期或时间类型值到一个数字，反过来也如此。
- 当 MySQL 遇到一个日期或时间类型的值超出范围或对给类型不合法(见本节的开始)时，它将该类型的值变换到“零”值。（例外的是超出范围的 TIME 值被剪切为适当的 TIME 范围端点值。）下表显示对每种类型的“零”值的格式：

表附 1-6　时间和日期类型的零值

列类型	“零”值
DATETIME	'0000-00-00
DATE	'0000-00-00'
TIMESTAMP	00000000000000（长度取决于显示尺寸）
TIME	'00:00:00'
YEAR	0000

- “零”值是特殊的，但是你能使用在表中显示的值来明显地存储或引用他们。你也可以使用值'0'或 0 做到，这更容易写。
- 在 MyODBC 2.50.12 和以上版本中，由 MyODBC 使用的“零”日期或时间值被自动变换到 NULL，因为 ODBC 不能处理这样的值。

1.3.3.1 Y2K 问题和日期类型

MySQL 本身 Y2K 安全的(见 1.6 2000 年一致性)，但是呈交给 MySQL 的输入值可能不是。一个包含 2 位年份值的任何输入是由二义性的，因为世纪是未知的。这样的值必须被解释成 4 位形式，因为 MySQL 内部使用 4 位存储年份。

对于 DATETIME, DATE, TIMESTAMP 和 YEAR 类型，MySQL 使用下列规则的解释

二义性的年份值：

- 在范围 00-69 的年值被变换到 2000-2069。
- 在范围 70-99 的年值被变换到 1970-1999。

记得这些规则仅仅提供对于你数据的含义的合理猜测。如果 MySQL 使用的启发规则不产生正确的值，你应该提供无二义的包含 4 位年值的输入。

1.3.3.2 DATETIME, DATE 和 TIMESTAMP 类型

DATETIME, DATE 和 TIMESTAMP 类型是相关的。本节描述他们的特征，他们是如何类似的而又不同的。

DATETIME 类型用在你需要同时包含日期和时间信息的值时。MySQL 检索并且以'YYYY-MM-DD HH:MM:SS'格式显示 DATETIME 值，支持的范围是'1000-01-01 00:00:00'到'9999-12-31 23:59:59'。（“支持”意味着尽管更早的值可能工作，但不能保证他们可以。）

DATE 类型用在你仅需要日期值时，没有时间部分。MySQL 检索并且以'YYYY-MM-DD'格式显示 DATE 值，支持的范围是'1000-01-01'到'9999-12-31'。

TIMESTAMP 列类型提供一种类型，你可以使用它自动地用当前的日期和时间标记 INSERT 或 UPDATE 的操作。如果你有多个 TIMESTAMP 列，只有第一个自动更新。

自动更新第一个 TIMESTAMP 列在下列任何条件下发生：

- 列没有明确地在一个 INSERT 或 LOAD DATA INFILE 语句中指定。
- 列没有明确地在一个 UPDATE 语句中指定且一些另外的列改变值。（注意一个 UPDATE 设置一个列为它已经有的值，这将不引起 TIMESTAMP 列被更新，因为如果你设置一个列为它当前的值，MySQL 为了效率而忽略更改。）
- 你明确地设定 TIMESTAMP 列为 NULL.

除第一个以外的 TIMESTAMP 列也可以设置到当前的日期和时间，只要将列设为 NULL，或 NOW()。

通过明确地设置希望的值，你可以设置任何 TIMESTAMP 列为不同于当前日期和时间的值，即使对第一个 TIMESTAMP 列也是这样。例如，如果，当你创建一个行时，你想要一个 TIMESTAMP 被设置到当前的日期和时间，但在以后无论何时行被更新时都不改变，你可以使用这个属性：

- 让 MySQL 在行被创建时设置列，这将初始化它为当前的日期和时间。
- 当你执行随后的对该行中其他列的更改时，明确设定 TIMESTAMP 列为它的当前值。

另一方面，你可能发现，当行被创建并且远离随后的更改时，很容易用一个你用 NOW() 初始化的 DATETIME 列。

TIMESTAMP 值可以从 1970 的某时的开始一直到 2037 年，精度为一秒，其值作为数字显示。

在 MySQL 检索并且显示 TIMESTAMP 值取决于显示尺寸的格式如下表。“完整”TIMESTAMP 格式是 14 位，但是 TIMESTAMP 列可以用更短的显示尺寸创造：

表附 1-7 TIMESTAMP 列的显示格式

列类型	显示格式
TIMESTAMP(14)	YYYYMMDDHHMMSS
TIMESTAMP(12)	YYMMDDHHMMSS
TIMESTAMP(10)	YYMMDDHHMM
TIMESTAMP(8)	YYYYMMDD
TIMESTAMP(6)	YYMMDD
TIMESTAMP(4)	YYMM
TIMESTAMP(2)	YY

所有的 TIMESTAMP 列都有同样的存储大小，不考虑显示尺寸。最常见的显示尺寸是 6、8、12、和 14。你可以在表创建时间指定一个任意的显示尺寸，但是值 0 或比 14 大被强制到 14。在从 1～13 范围的奇数值尺寸被强制为下一个更大的偶数。

使用一个常用的格式集的任何一个，你可以指定 DATETIME、DATE 和 TIMESTAMP 值：

- 'YYYY-MM-DD HH:MM:SS'或'YY-MM-DD HH:MM:SS'格式的一个字符串。允许一种“宽松”的语法--任何标点可用作在日期部分和时间部分之间的分隔符。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'格式的一个字符串。允许一种“宽松”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'格式的没有任何分隔符的一个字符串，例如，'19970523091528'和'970523091528'被解释为'1997-05-23 09:15:28'，但是'971122459015'是不合法的(它有毫无意义的分钟部分)且变成'0000-00-00 00:00:00'。
- 'YYYYMMDD'或'YYMMDD'格式的没有任何分隔符的一个字符串，如果字符串认为是一个日期。例如，'19970523'和'970523'被解释作为'1997-05-23'，但是'971332'是不合法的(它有无意义的月和天部分)且变成'0000-00-00'。
- YYYYMMDDHHMMSS 或 YYMMDDHHMMSS 格式的一个数字，如果数字认为是一个日期。例如，19830905132800 和 830905132800 被解释作为'1983-09-05 13:28:00'。
- YYYYMMDD 或 YYMMDD 格式的一个数字，如果数字认为是一个日期。例如，19830905 和 830905 被解释作为'1983-09-05'。
- 一个返回值可以在一个 DATETIME, DATE 或 TIMESTAMP 上下文环境中接受的函数，例如 NOW()或 CURRENT_DATE。

不合法 DATETIME, DATE 或 TIMESTAMP 值被变换到适当类型的“零”值('0000-00-00 00:00:00', '0000-00-00'或 00000000000000)。

对于包括的日期部分分隔符的指定为字符串的值，不必要为小于 10 的月或天的值指定 2 位数字，'1979-6-9'与'1979-06-09'是一样的。同样，对于包括的时间部分分隔符的指定为字符串的值，不必为小于 10 的小时、月或秒指定 2 位数字，'1979-10-30 1:2:3'与'1979-10-30 01:02:03'是一样的。

指定为数字应该是 6、8、12 或 14 位长。如果数字是 8 或 14 位长，它被假定以 YYYYMMDD 或 YYYYMMDDHHMMSS 格式并且年份由头 4 位数字给出。如果数字是 6 或 12 位长，它被假定是以 YYMMDD 或 YYMMDDHHMMSS 格式且年份由头 2 位数字给出。不是这些长度之一的数字通过填补前头的零到最接近的长度来解释。

指定为无分隔符的字符串用它们给定的长度来解释。如果字符串长度是 8 或 14 个字符，年份被假定头 4 个字符给出，否则年份被假定由头 2 个字符给出。对于字符串中呈现的多个部分，字符串从左到右边被解释，以找出年、月、日、小时、分钟和秒值，这意味着，你不应该使用少于 6 个字符的字符串。例如，如果你指定'9903'，认为将代表 1999 年 3 月，你会发现 MySQL 把一个“零”日期插入到你的表中，这是因为年份和月份值 99 和 03，但是日期部分丢失(零)，因此该值不是一个合法的日期。

TIMESTAMP 列使用被指定的值的完整精度的存储合法的值，不考虑显示大小。这有几个含意：

- 总是指定年，月，和日，即使你的列类型是 TIMESTAMP(4)或 TIMESTAMP(2)。否则，值将不是一个合法的日期并且 0 将被存储。
- 如果你使用 ALTER TABLE 拓宽一个狭窄的 TIMESTAMP 列，以前被“隐蔽”的信息将被显示。
- 同样，缩小一个 TIMESTAMP 列不会导致信息失去，除了感觉上值在显示时，较少的信息被显示出。
- 尽管 TIMESTAMP 值被存储为完整精度，直接操作存储值的唯一函数是 UNIX_TIMESTAMP()，其他函数操作在格式化了的检索的值上，这意味着你不能使用函数例如 HOUR()或 SECOND()，除非 TIMESTAMP 值的相关部分被包含在格式化的值中。例如，一个 TIMESTAMP 列的 HH 部分部被显示，除非显示大小至少是 10，因此在更短的 TIMESTAMP 值上试试使用 HOUR()产生一个无意义的结果。

在某种程度上，你可以把一种日期类型的值赋给一个不同的日期类型的对象。然而，这可能值有一些改变或信息的损失：

- 如果你将一个 DATE 值赋给一个 DATETIME 或 TIMESTAMP 对象，结果值的时间部分被设置为'00:00:00'，因为 DATE 值不包含时间信息。
- 如果你将一个 DATETIME 或 TIMESTAMP 值赋给一个 DATE 对象，结果值的时间部分被删除，因为 DATE 类型不存储时间信息。
- 记住，尽管 DATETIME, DATE 和 TIMESTAMP 值全都可以用同样的格式集来指定，但所有类型不都有同样的值范围。例如，TIMESTAMP 值不能比 1970 早或比 2037 网晚，这意味着，一个日期例如'1968-01-01'，当作为一个 DATETIME 或 DATE 值合法时，它不是一个正确 TIMESTAMP 值，并且如果赋值给这样一个对象，它

将被变换到 0。

当指定日期值时，当心某些缺陷：

- 允许作为字符串指定值的宽松格式能被欺骗。例如，值例如'10:11:12'可能看起来像时间值，因为“:”分隔符，但是如果在一个日期中使用，上下文将作为年份被解释成'2010-11-12'。值'10:45:15'将被变换到'0000-00-00'，因为'45'不是一个合法的月份。
- 以 2 位数字指定的年值是模糊的，因为世纪是未知的。MySQL 使用下列规则解释 2 位年值：
- 在 00-69 范围的年值被变换到 2000-2069。
- 在范 70-99 围的年值被变换到 1970-1999。

1.3.3.3 TIME 类型

MySQL 检索并以'HH:MM:SS'格式显示 TIME 值(或对大小时值，'HHH:MM:SS'格式)。TIME 值的范围可以从'-838:59:59'到'838:59:59'。小时部分可能很大的的原因是 TIME 类型不仅可以被使用在表示一天的时间(它必须是不到 24 个小时)，而且用在表示在 2 个事件之间经过的时间或时间间隔(它可以是比 24 个小时大些，或甚至是负值)。

你能用多中格式指定 TIME 值：

- 作为'HH:MM:SS'格式的一个字符串。“宽松”的语法被允许–任何标点符号可用作时间部分的分隔符，例如，'10:11:12'和'10.11.12'是等价的。
- 作为没有分隔符的'HHMMSS'格式的一个字符串，如果它作为一个时间解释。例如，'101112'被理解为'10:11:12'，但是'109712'是不合法的(它有无意义的分钟部分)并变成'00:00:00'。
- 作为 HHMMSS 格式的一个数字，如果它能解释为一个时间。例如，101112 被理解为'10:11:12'。
- 返回值可在一个 TIME 上下文接受的函数，例如 CURRENT_TIME。

对于作为包括一个时间分隔符的字符串被指定的 TIME 值，不必为小于 10 的小时、分钟或秒值指定 2 位数字，'8:3:2'与'08:03:02'是一样的。

将“短的”TIME 值赋值给一个 TIME 行列是要格外小心。MySQL 使用最右位代表秒的假设来解释值。(MySQL 将 TIME 值解释为经过的时间，而非作为一天的时间)例如，你可能想到'11:12'、'1112'和 1112 意味着'11:12:00'（11 点 12 分)，但是 MySQL 解释他们为'00:11:12'（11 分 12 秒)。同样，'12'和 12 被解释为'00:00:12'。

但是超出 TIME 范围之外的值是样合法的，它被剪切到范围适当的端点值。例如，'-850:00:00'和'850:00:00'被变换到'-838:59:59'和'838:59:59'。

不合法的 TIME 值被变换到'00:00:00'。注意，既然'00:00:00'本身是一个合法的 TIME 值，没有其他方法区分表中存储的一个'00:00:00'值，原来的值是否被指定为'00:00:00'或它是否是不合法的。

1.3.3.4 YEAR 类型

YEAR 类型是一个 1 字节类型用于表示年份。

MySQL 检索并且以 YYYY 格式显示 YEAR 值，其范围是 1901 到 2155。

你能用多种格式指定 YEAR 值：

- 作为在'1901'到'2155'范围的一个 4 位字符串。
- 作为在 1901 到 2155 范围的一个 4 位数字。
- 作为在'00'到'99'范围的一个 2 位字符串.在'00'到'69'和'70'到'99'范围的值被变换到在 2000 到 2069 范围和 1970 到 1999 的 YEAR 值。
- 作为在 1 到 99 范围的一个 2 位数字。在范围 1 到 69 和 70 到 99 的值被变换到在范围 2001 到 2069 和 1970 到 1999 的 YEAR 的值。注意对于 2 位数字的范围略微不同于 2 位数字字符串的范围，因为你不能直接指定零作为一个数字并且把它解释为 2000。你必须作为一个字符串'0'或'00'指定它，它将被解释为 0000。
- 其返回值可在一个 YEAR 上下文环境中接受的函数，例如 NOW()。

不合法 YEAR 值被变换到 0000。

1.3.4 字符串类型

字符串类型是 CHAR、VARCHAR、BLOB、TEXT、ENUM 和 SET。

1.3.4.1 CHAR 和 VARCHAR 类型

CHAR 和 VARCHAR 类型是类似的，但是在他们被存储和检索的方式不同。

一个 CHAR 列的长度被修正为在你创造表时你所声明的长度。长度可以是 1 和 255 之间的任何值。（在 MySQL 3.23 中，CHAR 长度可以是 0～255。）当 CHAR 值被存储时，他们被用空格在右边填补到指定的长度。当 CHAR 值被检索时，拖后的空格被删去。

在 VARCHAR 列中的值是变长字符串。你可以声明一个 VARCHAR 列是在 1 和 255 之间的任何长度，就像对 CHAR 列。然而，与 CHAR 相反，VARCHAR 值只存储所需的字符，外加一个字节记录长度，值不被填补；相反，当值被存储时，拖后的空格被删去。（这个空格删除不同于 ANSI SQL 规范。）

如果你把一个超过列最大长度的值赋给一个 CHAR 或 VARCHAR 列，值被截断以适合它。

下表显示了两种类型的列的不同，通过演示存储变长字符串值到 CHAR(4) 和 VARCHAR(4)列：

表附 1-8 CHAR 和 VARCHAR 类型的存储

值	CHAR(4)	存储需求	VARCHAR(4)	存储需求
''	''	4 个字节	''	1 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

从 CHAR(4)和 VARCHAR(4)列检索的值在每种情况下都是一样的，因为拖后的空格从检索的 CHAR 列上被删除。

在 CHAR 和 VARCHAR 列中存储和比较值是以大小写不区分的方式进行的，除非当表被创建时，BINARY 属性被指定。BINARY 属性意味着该列的值根据 MySQL 服务器正在运行的机器的 ASCII 顺序以大小写区分的方式存储和比较。

BINARY 属性是“粘性”的。这意味着，如果标记了 BINARY 的列用于一个表达式中，整个的表达式作为一个 BINARY 值被比较。

MySQL 在表创建时可以隐含地改变一个 CHAR 或 VARCHAR 列的类型。见 1.1.1 隐含的的列说明改变。

1.3.4.2 BLOB 和 TEXT 类型

一个 BLOB 是一个能保存可变数量的数据的二进制的大对象。4 个 BLOB 类型 TINYBLOB、BLOB、MEDIUMBLOB 和 LONGBLOB 仅仅在他们能保存值的最大长度方面有所不同。见 1.3.1 列类型存储需求。

4 个 TEXT 类型 TINYTEXT、TEXT、MEDIUMTEXT 和 LONGTEXT 对应于 4 个 BLOB 类型，并且有同样的最大长度和存储需求。在 BLOB 和 TEXT 类型之间的唯一差别是对 BLOB 值的排序和比较以大小写敏感方式执行，而对 TEXT 值是大小写不敏感的。换句话说，一个 TEXT 是一个大小写不敏感的 BLOB。

如果你把一个超过列类型最大长度的值赋给一个 BLOB 或 TEXT 列，值被截断以适合它。

在大多数方面，你可以认为一个 TEXT 行列是你所希望大的一个 VARCHAR 列。同样，你可以认为一个 BLOB 列是一个 VARCHAR BINARY 列。差别是：

- 用 MySQL 版本 3.23.2 和更新，你能在 BLOB 和 TEXT 列上索引。更旧的 MySQL 版本不支持这个。
- 当值被存储时，对 BLOB 和 TEXT 列没有拖后空格的删除，因为对 VARCHAR 列有删除。
- BLOB 和 TEXT 列不能有 DEFAULT 值。

MyODBC 定义 BLOB 为 LONGVARBINARY，TEXT 值为 LONGVARCHAR。

因为 BLOB 和 TEXT 值可以是非常长的，当使用他们时，你可能遇到一些限制：

- 如果你想要在一个 BLOB 或 TEXT 列上使用 GROUP BY 或 ORDER BY，你必须将列值变换成一个定长对象。这样做的标准方法是用 SUBSTRING 函数。例如：

```
mysql> select comment from tbl_name,substring(comment,20) as substr
    -> ORDER BY substr;
```

如果你不这样做，在排序时，只有列的首 max_sort_length 个字节被使用，缺省的 max_sort_length 是 1024；这个值能在启动 mysqld 服务器时使用-O 选择改变。你可以在包含 BLOB 或 TEXT 值得一个表达式上分组(group)，通过指定列的位置或使用一个

别名：

```
mysql> select id,substring(blob_col,1,100) from tbl_name GROUP BY 2;
mysql> select id,substring(blob_col,1,100) as b from tbl_name GROUP BY b;
```

- 一个 BLOB 或 TEXT 对象的最大尺寸由其类型决定，但是你能在客户与服务器之间是实际传输的最大值由可用的内存数量和通讯缓冲区的大小来决定。你能改变消息缓冲区大小，但是你必须在服务器和客户两端做。

1.3.4.3 ENUM 类型

一个 ENUM 是一个字符对象，其值通常从一个在表创建时明确被列举的允许值的一张表中选择。

在下列的某个情形下，值也可以空字符串("")或 NULL：

- 如果你把一个无效值插入到一个 ENUM（即，一个不在允许的值列表中的字符串），空字符串作为一个特殊错误的值被插入。
- 如果一个 ENUM 被声明为 NULL，NULL 也是列的合法值，并且缺省值是 NULL。如果一个 ENUM 被声明为 NOT NULL，缺省值是允许值的列表的第一成员。

每枚举值有一个编号：

- 在列说明中来自允许成员值列表值用从 1 开始编号。
- 空字符串错误值的编号值是 0。这意味着，你能使用下列 SELECT 语句找出被赋给无效 ENUM 值的行：

```
mysql> SELECT * FROM tbl_name WHERE enum_col=0;
```

- NULL 值的编号是 NULL。

例如，指定为 ENUM("one", "two", "three")的列可以有显示在下面的值的任何一个。每个值的编号也被显示：

表附 1-9 ENUM 类型的编号

值	编号
NULL	NULL
""	0
"one"	1
"two"	2

枚举可以有最大 65535 个成员。

当你把值赋给一个 ENUM 列时，字母的大小写是无关紧要的。然而，以后从列中检索的值大小写匹配在表创建时用来指定允许值的值的大小写。

如果你在一个数字的上下文环境中检索一个 ENUM，列值的编号被返回。如果你存储一个数字到一个 ENUM 中，数字被当作一个标号，并且存储的值是该编号的枚举成员。

ENUM 值根据列说明列举的枚举成员的次序被排序。（换句话说，ENUM 值根据他们

的编号数字被排序) 例如，对 ENUM("a", "b")，"a"排在"b"前面，但是对 ENUM("b", "a")，"b"排在"a"前面。空字符串排序非空字符串之前，并且 NULL 排在所有其他枚举值之前。

如果你想要得到一个 ENUM 列的所有可能的值，你应该使用：SHOW COLUMNS FROM table_name LIKE enum_column_name 并且分析在第二列的 ENUM 定义。

1.3.4.4 SET 类型

一个 SET 是可以有零或多个值的一个字符串对象，其每一个必须从表创建造被指定了的允许值的一张列表中被选择。由多个集合成员组成的 SET 列通过由由逗号分隔(“,”)的成员被指定，其推论是该 SET 成员值不能包含逗号本身。

例如，一个指定为 SET("one", "two") NOT NULL 的列可以有这些值的任何一个：

""

"one"

"two"

"one,two"

一个 SET 能有最多 64 个不同的成员。

MySQL 用数字值存储 SET 值，存储值的低阶位对应于第一个集合成员。如果你在数字上下文中检索一个 SET 值，检索的值把位设置位对应组成列值的集合成员。如果一个数字被存储进一个 SET 列，在数字的二进制表示中设置的位决定了在列中的集合成员。假定一个列被指定为 SET("a","b","c","d")，那么成员有下列位值：

表附 1-10 SET 成员的位值

SET	成员	十进制的值
a	1	0001
b	2	0010
c	4	0100
d	8	1000

如果你给该列赋值 9，即二进制的 1001，这样第一个和第四个 SET 值成员"a"和"d"被选择并且结果值是"a,d"。

对于包含超过一个 SET 成员的值，当你插入值时，无所谓以什么顺序列举值，也无所谓给定的值列举了多少次。当以后检索值时，在值中的每个成员将出现一次，根据他们在表创建时被指定的顺序列出成员。例如，如果列指定为 SET("a","b","c","d")，那么"a,d"、"d,a"和"d,a,a,d,d"在检索时将均作为"a,d"出现。

SET 值以数字次序被排序。NULL 指排在非 NULL SET 值之前。

通常，你使用 LIKE 操作符或 FIND_IN_SET()函数执行在一个 SET 上的一个 SELECT：

```
mysql> SELECT * FROM tbl_name WHERE set_col LIKE '%value%';
mysql> SELECT * FROM tbl_name WHERE FIND_IN_SET('value',set_col)>0;
```

但是下列也会工作：

```
mysql> SELECT * FROM tbl_name WHERE set_col = 'val1,val2';
mysql> SELECT * FROM tbl_name WHERE set_col & 1;
```

这些语句的第一个语句寻找一个精确的匹配。第二个寻找包含第一个集合成员的值。

如果你想要得到一个 SET 列的所有可能的值，你应该使用：SHOW COLUMNS FROM table_name LIKE set_column_name 并且分析在第二列的 SET 定义。

1.3.5 为列选择正确的类型

为了最有效地使用存储空间，试着在所有的情况下使用最精确的类型。例如，如果一个整数列被用于在之间 1 和 99999 的值， MEDIUMINT UNSIGNED 是最好的类型。

货币值的精确表示是一个常见的问题。在 MySQL，你应该使用 DECIMAL 类型，它作为一个字符串被存储，不会发生精确性的损失。如果精确性不是太重要，DOUBLE 类型也是足够好的。

对高精度，你总是能变换到以一个 BIGINT 存储的定点类型。这允许你用整数做所有的计算，并且仅在必要时将结果转换回浮点值。

1.3.6 列索引

所有的 MySQL 列类型能被索引。在相关的列上的使用索引是改进 SELECT 操作性能的最好方法。

一个表最多可有 16 个索引。最大索引长度是 256 个字节，尽管这可以在编译 MySQL 时被改变。

对于 CHAR 和 VARCHAR 列，你可以索引列的前缀。这更快并且比索引整个列需要较少的磁盘空间。在 CREATE TABLE 语句中索引列前缀的语法看起来像这样：

```
KEY index_name (col_name(length))
```

下面的例子为 name 列的头 10 个字符创建一个索引：

```
mysql> CREATE TABLE test (
            name CHAR(200) NOT NULL,
            KEY index_name (name(10)));
```

对于 BLOB 和 TEXT 列，你必须索引列的前缀，你不能索引列的全部。

1.3.7 多列索引

MySQL 能在多个列上创建索引。一个索引可以由最多 15 个列组成。（在 CHAR 和 VARCHAR 列上，你也可以使用列的前缀作为一个索引的部分)。

一个多重列索引可以认为是包含通过合并(concatenate)索引列值创建的值的一个排序数组。

当你为在一个 WHERE 子句索引的第一列指定已知的数量时，MySQL 以这种方式使用多重列索引使得查询非常快速，即使你不为其他列指定值。

假定一张表使用下列说明创建：

```
mysql> CREATE TABLE test (
        id INT NOT NULL,
        last_name CHAR(30) NOT NULL,
        first_name CHAR(30) NOT NULL,
        PRIMARY KEY (id),
        INDEX name (last_name,first_name));
```

那么索引 name 是一个在 last_name 和 first_name 上的索引，这个索引将被用于在 last_name 或 last_name 和 first_name 的一个已知范围内指定值的查询，因此，name 索引将使用在下列查询中：

```
mysql> SELECT * FROM test WHERE last_name="Widenius";
mysql> SELECT * FROM test WHERE last_name="Widenius"
                              AND first_name="Michael";
mysql> SELECT * FROM test WHERE last_name="Widenius"
                              AND (first_name="Michael" OR first_name="Monty");
mysql> SELECT * FROM test WHERE last_name="Widenius"
                              AND first_name >="M" AND first_name < "N";
```

然而，name 索引将不用在下列询问中：

```
mysql> SELECT * FROM test WHERE first_name="Michael";
mysql> SELECT * FROM test WHERE last_name="Widenius"
                              OR first_name="Michael";
```

关于 MySQL 使用索引改进性能的方式的更多的信息，

1.3.8 使用来自其他数据库引擎的列类型

为了跟容易地使用为其他供应商的 SQL 实现编写的代码，下表显示了 MySQL 映射的列类型。这些映射使得从其他数据库引擎移动表定义到 MySQL 更容易：

表附 1-11 外来类型与 MySQL 类型的对应

其他供应商类型	MySQL 类型
BINARY(NUM)	BINARY
CHAR VARYING(NUM)	VARCHAR(NUM)
FLOAT4	FLOAT
FLOAT8	DOUBLE
INT1	TINYINT
INT2	SMALLINT
INT3	MEDIUMINT
INT4	INT

INT8	BIGINT
LONG VARBINARY	MEDIUMBLOB
LONG VARCHAR	MEDIUMTEXT
MIDDLEINT	MEDIUMINT
VARBINARY(NUM)	VARCHAR(NUM) BINARY

其列类型映射发生在表创建时。如果你用其他供应商使用的类型创建表，那么发出一个 DESCRIBE tbl_name 语句，MySQL 使用等价的 MySQL 类型报告表结构。

1.4 用在 SELECT 和 WHERE 子句中的函数

在一个 SQL 语句中的 select_expression 或 where_definition 可由使用下面描述的函数的任何表达式组成。

包含 NULL 的一个表达式总是产生一个 NULL 值，否则除非表达式所包含的操作符和函数在文档中说明。

注意：在一个函数名和跟随它的括号之间不许没有空格。这帮助 MySQL 分析器区分函数调用和具有相同名字的对表或列的引用，尽管允许在参数周围有空格。

为了简洁，例子以缩写形式显示从 mysql 程序输出。因此：

```
mysql> select MOD(29,9);
+-----------+
| MOD(29,9) |
+-----------+
|         2 |
+-----------+
1 row in set (0.10 sec)
```

被显示为这样：

```
mysql> select MOD(29,9);
        -> 2
```

1.4.1 分组函数

(...)

括号。使用它们来强制在一个表达式的计算顺序。

```
mysql> select 1+2*3;
        -> 7
mysql> select (1+2)*3;
        -> 9
```

1.4.2 常用的算术操作

一般的算术操作符是可用的。注意在-、+和*情况下，如果两个参数是整数，结果用

BIGINT（64 位）精度计算！

+

加法

```
mysql> select 3+5;
    -> 8
```

-

减法

```
mysql> select 3-5;
    -> -2
```

*

乘法

```
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，因为整数乘积的结果超过用 BIGINT 计算的 64 位范围。

/

除法

```
mysql> select 3/5;
    -> 0.60
```

被零除产生一个 NULL 结果：

```
mysql> select 102/(1-1);
    -> NULL
```

一个除法用 BIGINT 算术计算，只要在它的结果被转换到一个整数的上下文中执行！

1.4.3 位函数

MySQL 为位操作使用 BIGINT（64 位)算法，因此这些操作符有最大 64 位的一个范围。

|

位或

```
mysql> select 29 | 15;
    -> 31
```

&

位与

```
mysql> select 29 & 15;
    -> 13
```

<<
左移位一个长(BIGINT)数字。
mysql> select 1 << 2
-> 4
>>
右移位一个长(BIGINT)数字。
mysql> select 4 >> 2
-> 1
~
颠倒所有的位。
mysql> select 5 & ~1
-> 4
BIT_COUNT(N)
返回在参数 N 设定的位的数量。
mysql> select BIT_COUNT(29);
-> 4

1.4.4 逻辑运算

所有的逻辑函数返回 1（TRUE）或 0（FALSE）。
NOT
!
逻辑非。如果参数是 0，返回 1，否则返回 0。例外： NOT NULL 返回 NULL。
mysql> select NOT 1;
-> 0
mysql> select NOT NULL;
-> NULL
mysql> select ! (1+1);
-> 0
mysql> select ! 1+1;
-> 1
最后的例子返回 1，因为表达式作为(!1)+1 计算。
OR
||
逻辑或。如果任何一个参数不是 0 并且不 NULL，返回 1。
mysql> select 1 || 0;
-> 1
mysql> select 0 || 0;
-> 0

```
mysql> select 1 || NULL;
        -> 1
```

AND

&&

逻辑与。如果任何一个参数是 0 或 NULL，返回 0，否则返回 1。

```
mysql> select 1 && NULL;
        -> 0
mysql> select 1 && 0;
        -> 0
```

1.4.5 比较运算符

比较操作得出值 1（TRUE）、0（FALSE）或 NULL 等结果。这些函数工作运用在数字和字符串上。当需要时，字符串自动地被变换到数字且数字到字符串(如在 Perl)。

MySQL 使用下列规则执行比较：

如果一个或两个参数是 NULL，比较的结果是 NULL，除了<=>操作符。

如果在比较中操作的两个参数是字符串，他们作为字符串被比较。

如果两个参数是整数，他们作为整数被比较。

十六进制的值如果不与一个数字比较，则被当作二进制字符串。

如果参数之一是一个 TIMESTAMP 或 DATETIME 列而其他参数是一个常数，在比较执行前，常数被转换为一个时间标记。这样做是为了对 ODBC 更友好。

在所有其他的情况下，参数作为浮点(实数)数字被比较。

缺省地，字符串使用当前的字符集以大小写敏感的方式进行(缺省为 ISO-8859-1 Latin1，它对英语运用得很出色)。

下面的例子演示了对于比较操作字符串到数字的转换：

```
mysql> SELECT 1 > '6x';
        -> 0
mysql> SELECT 7 > '6x';
        -> 1
mysql> SELECT 0 > 'x6';
        -> 0
mysql> SELECT 0 = 'x6';
        -> 1
```

=

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
```

<>

!=

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;
        -> 1
```

>=

大于或等于

```
mysql> select 2 >= 2;
        -> 1
```

>

大于

```
mysql> select 2 > 2;
        -> 0
```

<=>

安全等于 Null

```
mysql> select 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
        -> 1 1 0
```

IS NULL

IS NOT NULL

测试值是否是或不是 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;
```

expr BETWEEN min AND max

如果 expr 对大于或等于 min 且 expr 是小于或等于 max，BETWEEN 返回 1，否则它返回 0。如果所有的参数类型是一样得，这等价于表达式(min <= expr AND expr <= max)。第一个参数(expr)决定比较如何被执行。如果 expr 是一个大小写不敏感的字符串表达式，进行一个大小写不敏感的字符串比较。如果 expr 是一个大小写敏感的字符串表达式，进行一个大小写敏感的字符串比较。如果 expr 是一个整数表达式，进行整数比较。否则，进行一个浮点(实数)比较。

```
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 IN (value,...)

如果 expr 是在 IN 表中的任何值，返回 1，否则返回 0。如果所有的值是常数，那么所有的值根据 expr 类型被计算和排序，然后项目的搜索是用二进制的搜索完成。这意味着如果 IN 值表全部由常数组成，IN 是很快的。如果 expr 是一个大小写敏感的字符串表达式，字符串比较以大小写敏感方式执行。

```
mysql> select 2 IN (0,3,5,'wefwf');
    -> 0
mysql> select 'wefwf' IN (0,3,5,'wefwf');
    -> 1
```

expr NOT IN (value,...)

与 NOT (expr IN (value,...))相同。

ISNULL(expr)

如果 expr 是 NULL，ISNULL()返回 1，否则它返回 0。

```
mysql> select ISNULL(1+1);
    -> 0
mysql> select ISNULL(1/0);
    -> 1
```

注意，使用=的 NULL 的值比较总为假！

COALESCE(list)

回来 list 中第一个非 NULL 的单元。

```
mysql> select COALESCE(NULL,1);
        -> 1
mysql> select COALESCE(NULL,NULL,NULL);
        -> NULL
```

INTERVAL(N,N1,N2,N3,...)

如果 N< N1，返回 0，如果 N< N2，返回 1 等等。所有的参数被当作整数。为了函数能正确地工作，它要求 N1<N2<N3< ...<Nn。这是因为使用二进制搜索(很快)。

```
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
```

1.4.6 字符串比较函数

通常，如果在字符串比较中的任何表达式是区分大小写的，比较以大小写敏感的方式执行。

expr LIKE pat [ESCAPE 'escape-char']

使用 SQL 的简单的正规表达式比较的模式匹配。返回 1（TRUE）或 0（FALSE）。用 LIKE，你可以在模式中使用下列 2 个通配符字符：

- % 匹配任何数目的字符，甚至零个字符
- _ 精确匹配一个字符

为了测试一个通配符的文字实例，用转义字符的加在字符前面。如果你不指定 ESCAPE 字符，假定为“\”：

- \% 匹配一%字符
- _ 匹配一_字符

```
mysql> select 'David!' LIKE 'David\_';
        -> 0
mysql> select 'David_' LIKE 'David\_';
        -> 1
```

为了指定一个不同的转义字符，使用 ESCAPE 子句：

```
mysql> select 'David_' LIKE 'David|_' ESCAPE '|';
        -> 1
```

LIKE 允许用在数字的表达式上！（这是 MySQL 对 ANSI SQL LIKE 的一个扩充。）

```
mysql> select 10 LIKE '1%';
        -> 1
```

注意：因为 MySQL 在字符串中使用 C 转义语法(例如，“\n”)，你必须在你的 LIKE

字符串中重复任何“\”。例如，为了查找“\n”，指定它为“ \\n”，为了查找“\”，指定它为“\\\\”（反斜线被分析器剥去一次，另一次是在模式匹配完成时，留下一条单独的反斜线被匹配)。

expr NOT LIKE pat [ESCAPE 'escape-char']

与 NOT (expr LIKE pat [ESCAPE 'escape-char'])相同。

expr REGEXP pat

expr RLIKE pat

执行一个字符串表达式 expr 对一个模式 pat 的模式匹配。模式可以是一个扩充的正则表达式。如果 expr 匹配 pat，返回 1，否则返回 0。RLIKE 是 REGEXP 的一个同义词，提供了与 mSQL 的兼容性。注意：因为 MySQL 在字符串中使用 C 转义语法(例如，“\n”)，你必须在你的 REGEXP 字符串重复任何“\”。在 MySQL3.23.4 中，REGEXP 对于正常的(不是二进制)字符串是忽略大小写。

```
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
```

当决定一个字符的类型时，REGEXP 和 RLIKE 使用当前的字符集(缺省为 ISO-8859-1 Latin1)。

expr NOT REGEXP pat

expr NOT RLIKE pat

与 NOT (expr REGEXP pat)相同。

STRCMP(expr1,expr2)

如果字符串相同，STRCMP()回来 0，如果第一参数根据当前的排序次序小于第二个，返回-1，否则返回 1。

```
mysql> select STRCMP('text', 'text2');
        -> -1
mysql> select STRCMP('text2', 'text');
        -> 1
mysql> select STRCMP('text', 'text');
        -> 0
```

1.4.7 类型转换运算符

BINARY

BINARY 操作符强制跟随它后面的字符串为一个二进制字符串。即使列没被定义为 BINARY 或 BLOB，这是一个强制列比较区分大小写的简易方法。

```
mysql> select "a" = "A";
        -> 1
mysql> select BINARY "a" = "A";
        -> 0
```

BINARY 在 MySQL 3.23.0 中被引入。

1.4.8 控制流函数

IFNULL(expr1,expr2)

如果 expr1 不是 NULL，IFNULL()返回 expr1，否则它返回 expr2。IFNULL()返回一个数字或字符串值，取决于它被使用的上下文环境。

```
mysql> select IFNULL(1,0);
        -> 1
mysql> select IFNULL(0,10);
        -> 0
mysql> select IFNULL(1/0,10);
        -> 10
mysql> select IFNULL(1/0,'yes');
        -> 'yes'
```

IF(expr1,expr2,expr3)

如果 expr1 是 TRUE(expr1<>0 且 expr1<>NULL)，那么 IF()返回 expr2，否则它返回 expr3。IF()返回一个数字或字符串值，取决于它被使用的上下文。

```
mysql> select IF(1>2,2,3);
        -> 3
mysql> select IF(1<2,'yes','no');
        -> 'yes'
mysql> select IF(strcmp('test','test1'),'yes','no');
        -> 'no'
```

expr1 作为整数值被计算，它意味着如果你正在测试浮点或字符串值，你应该使用一个比较操作来做。

```
mysql> select IF(0.1,1,0);
        -> 0
mysql> select IF(0.1<>0,1,0);
        -> 1
```

在上面的第一种情况中，IF(0.1)返回 0，因为 0.1 被变换到整数值，导致测试 IF(0)。这可能不是你期望的。在第二种情况中，比较测试原来的浮点值看它是否是非零，比较的结

果被用作一个整数。

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

第一个版本返回result，其中 value=compare-value。第二个版本中如果第一个条件为真，返回 result。如果没有匹配的 result 值，那么结果在 ELSE 后的 result 被返回。如果没有 ELSE 部分，那么 NULL 被返回。

```
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
```

1.4.9 数学函数

所有的数学函数在一个出错的情况下返回 NULL。

-

单目减。改变参数的符号。

```
mysql> select - 2;
```

注意，如果这个操作符与一个 BIGINT 使用，返回值是一个 BIGINT！这意味着你应该避免在整数上使用-，那可能有值-2^63！

ABS(X)

返回 X 的绝对值。

```
mysql> select ABS(2);
        -> 2
mysql> select ABS(-32);
        -> 32
```

该功能可安全用于 BIGINT 值。

SIGN(X)

返回参数的符号，为-1、0 或 1，取决于 X 是否是负数、零或正数。

```
mysql> select SIGN(-32);
        -> -1
mysql> select SIGN(0);
        -> 0
mysql> select SIGN(234);
```

-> 1

MOD(N,M)

%

模 (类似 C 中的%操作符)。返回 N 被 M 除的余数。

mysql> select MOD(234, 10);

-> 4

mysql> select 253 % 7;

-> 1

mysql> select MOD(29,9);

-> 2

这个函数可安全用于 BIGINT 值。

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

注意返回值被变换为一个 BIGINT!

CEILING(X)

返回不小于 X 的最小整数值。

mysql> select CEILING(1.23);

-> 2

mysql> select CEILING(-1.23);

-> -1

注意返回值被变换为一个 BIGINT!

ROUND(X)

返回参数 X 的四舍五入的一个整数。

mysql> select ROUND(-1.23);

-> -1

mysql> select ROUND(-1.58);

-> -2

mysql> select ROUND(1.58);

-> 2

注意返回值被变换为一个 BIGINT!

ROUND(X,D)

返回参数 X 的四舍五入的有 D 为小数的一个数字。如果 D 为 0，结果将没有小数点或小数部分。

mysql> select ROUND(1.298, 1);

```
        -> 1.3
mysql> select ROUND(1.298, 0);
        -> 1
```

注意返回值被变换为一个 BIGINT!

EXP(X)

返回值 e（自然对数的底）的 X 次方。

```
mysql> select EXP(2);
        -> 1.389056
mysql> select EXP(-2);
        -> 0.135335
```

LOG(X)

返回 X 的自然对数。

```
mysql> select LOG(2);
        -> 0.693147
mysql> select LOG(-2);
        -> NULL
```

如果你想要一个数字 X 的任意底 B 的对数，使用公式 LOG(X)/LOG(B)。

LOG10(X)

返回 X 的以 10 为底的对数。

```
mysql> select LOG10(2);
        -> 0.301030
mysql> select LOG10(100);
        -> 2.000000
mysql> select LOG10(-100);
        -> NULL
```

POW(X,Y)

POWER(X,Y)

返回值 X 的 Y 次幂。

```
mysql> select POW(2,2);
        -> 4.000000
mysql> select POW(2,-2);
        -> 0.250000
```

SQRT(X)

返回非负数 X 的平方根。

```
mysql> select SQRT(4);
        -> 2.000000
mysql> select SQRT(20);
        -> 4.472136
```

PI()
返回 PI 的值（圆周率）。
```
mysql> select PI();
        -> 3.141593
```
COS(X)
返回 X 的余弦，在这里 X 以弧度给出。
```
mysql> select COS(PI());
        -> -1.000000
```
SIN(X)
返回 X 的正弦值，在此 X 以弧度给出。
```
mysql> select SIN(PI());
        -> 0.000000
```
TAN(X)
返回 X 的正切值，在此 X 以弧度给出。
```
mysql> select TAN(PI()+1);
        -> 1.557408
```
ACOS(X)
返回 X 反余弦，即其余弦值是 X。如果 X 不在-1 到 1 的范围，返回 NULL。
```
mysql> select ACOS(1);
        -> 0.000000
mysql> select ACOS(1.0001);
        -> NULL
mysql> select ACOS(0);
        -> 1.570796
```
ASIN(X)
返回 X 反正弦值，即其正弦值是 X。L 如果 X 不在-1 到 1 的范围，返回 NULL。
```
mysql> select ASIN(0.2);
        -> 0.201358
mysql> select ASIN('foo');
        -> 0.000000
```

ATAN(X)
返回 X 的反正切值，即其正切值是 X。
```
mysql> select ATAN(2);
        -> 1.107149
mysql> select ATAN(-2);
        -> -1.107149
```
ATAN2(X,Y)

返回 2 个变量 X 和 Y 的反正切。它类似于计算 Y/X 的反正切，除了两个参数的符号被用来决定结果的象限。

```
mysql> select ATAN(-2,2);
        -> -0.785398
mysql> select ATAN(PI(),0);
        -> 1.570796
```

COT(X)

返回 X 的余切。

```
mysql> select COT(12);
        -> -1.57267341
mysql> select COT(0);
        -> NULL
```

RAND()

RAND(N)

返回在范围 0 到 1.0 内的随机浮点值。如果一个整数参数 N 被指定，它被用作种子值。

```
mysql> select RAND();
        -> 0.5925
mysql> select RAND(20);
        -> 0.1811
mysql> select RAND(20);
        -> 0.1811
mysql> select RAND();
        -> 0.2079
mysql> select RAND();
        -> 0.7888
```

你不能在一个 ORDER BY 子句用 RAND()值使用列，因为 ORDER BY 将重复计算列多次。然而在 MySQL3.23 中，你可以做：SELECT * FROM table_name ORDER BY RAND()，这是有利于得到一个来自 SELECT * FROM table1,table2 WHERE a=b AND c<d ORDER BY RAND() LIMIT 1000 的集合的随机样本。注意在一个 WHERE 子句里的一个 RAND()将在每次 WHERE 被执行时重新评估。

LEAST(X,Y,...)

有 2 和 2 个以上的参数，返回最小(最小值)的参数。参数使用下列规则进行比较：

如果返回值被使用在一个 INTEGER 上下文，或所有的参数都是整数值，他们作为整数比较。

如果返回值被使用在一个 REAL 上下文，或所有的参数是实数值，他们作为实数比较。

如果任何参数是一个大小敏感的字符串，参数作为大小写敏感的字符串被比较。

在其他的情况下，参数作为大小写无关的字符串被比较。

```
mysql> select LEAST(2,0);
        -> 0
mysql> select LEAST(34.0,3.0,5.0,761.0);
        -> 3.0
mysql> select LEAST("B","A","C");
        -> "A"
```

在 MySQL 3.22.5 以前的版本，你可以使用 MIN()而不是 LEAST。

GREATEST(X,Y,...)

返回最大(最大值)的参数。参数使用与 LEAST 一样的规则进行比较。

```
mysql> select GREATEST(2,0);
        -> 2
mysql> select GREATEST(34.0,3.0,5.0,761.0);
        -> 761.0
mysql> select GREATEST("B","A","C");
        -> "C"
```

在 MySQL 在 3.22.5 以前的版本，你能使用 MAX()而不是 GREATEST.

DEGREES(X)

返回参数 X，从弧度变换为角度。

```
mysql> select DEGREES(PI());
        -> 180.000000
```

RADIANS(X)

返回参数 X，从角度变换为弧度。

```
mysql> select RADIANS(90);
        -> 1.570796
```

TRUNCATE(X,D)

返回数字 X，截断为 D 位小数。如果 D 为 0，结果将没有小数点或小数部分。

```
mysql> select TRUNCATE(1.223,1);
        -> 1.2
mysql> select TRUNCATE(1.999,1);
        -> 1.9
mysql> select TRUNCATE(1.999,0);
        -> 1
```

1.4.10 字符串函数

如果结果的长度大于服务器参数 max_allowed_packet，字符串值函数返回 NULL。

对于针对字符串位置的操作，第一个位置被标记为 1。

ASCII(str)

返回字符串 str 的最左面字符的 ASCII 代码值。如果 str 是空字符串，返回 0。如果 str

是 NULL，返回 NULL。

```
mysql> select ASCII('2');
        -> 50
mysql> select ASCII(2);
        -> 50
mysql> select ASCII('dx');
        -> 100
```

也可参见 ORD()函数。

ORD(str)

如果字符串 str 最左面字符是一个多字节字符，通过以格式((first byte ASCII code)*256+(second byte ASCII code))[*256+third byte ASCII code...]返回字符的 ASCII 代码值来返回多字节字符代码。如果最左面的字符不是一个多字节字符。返回与 ASCII()函数返回的相同值。

```
mysql> select ORD('2');
        -> 50
```

CONV(N,from_base,to_base)

在不同的数字基之间变换数字。返回数字 N 的字符串数字，从 from_base 基变换为 to_base 基，如果任何参数是 NULL，返回 NULL。参数 N 解释为一个整数，但是可以指定为一个整数或一个字符串。最小基是 2 且最大的基是 36。如果 to_base 是一个负数，N 被认为是一个有符号数，否则，N 被当作无符号数。 CONV 以 64 位点精度工作。

```
mysql> select CONV("a",16,2);
        -> '1010'
mysql> select CONV("6E",18,8);
        -> '172'
mysql> select CONV(-17,10,-18);
        -> '-H'
mysql> select CONV(10+"10"+'10'+0xa,10,10);
        -> '40'
```

BIN(N)

返回二进制值 N 的一个字符串表示，在此 N 是一个长整数(BIGINT)数字，这等价于 CONV(N,10,2)。如果 N 是 NULL，返回 NULL。

```
mysql> select BIN(12);
        -> '1100'
```

OCT(N)

返回八进制值 N 的一个字符串的表示，在此 N 是一个长整型数字，这等价于 CONV(N,10,8)。如果 N 是 NULL，返回 NULL。

```
mysql> select OCT(12);
        -> '14'
```

HEX(N)

返回十六进制值 N 一个字符串的表示，在此 N 是一个长整型(BIGINT)数字，这等价于 CONV(N,10,16)。如果 N 是 NULL，返回 NULL。

```
mysql> select HEX(255);
        -> 'FF'
```

CHAR(N,...)

CHAR()将参数解释为整数并且返回由这些整数的 ASCII 代码字符组成的一个字符串。NULL 值被跳过。

```
mysql> select CHAR(77,121,83,81,'76');
        -> 'MySQL'
mysql> select CHAR(77,71.3,'71.3');
        -> 'MMM'
```

CONCAT(str1,str2,...)

返回来自于参数连结的字符串。如果任何参数是 NULL，返回 NULL。可以有超过 2 个的参数。一个数字参数被变换为等价的字符串形式。

```
mysql> select CONCAT('My', 'S', 'QL');
        -> 'MySQL'
mysql> select CONCAT('My', NULL, 'QL');
        -> NULL
mysql> select CONCAT(14.3);
        -> '14.3'
```

LENGTH(str)

OCTET_LENGTH(str)

CHAR_LENGTH(str)

CHARACTER_LENGTH(str)

返回字符串 str 的长度。

```
mysql> select LENGTH('text');
        -> 4
mysql> select OCTET_LENGTH('text');
        -> 4
```

注意，对于多字节字符，其 CHAR_LENGTH()仅计算一次。

LOCATE(substr,str)

POSITION(substr IN str)

返回子串 substr 在字符串 str 第一个出现的位置，如果 substr 不是在 str 里面，返回 0.

```
mysql> select LOCATE('bar', 'foobarbar');
        -> 4
mysql> select LOCATE('xbar', 'foobar');
        -> 0
```

该函数是多字节可靠的。

LOCATE(substr,str,pos)

返回子串 substr 在字符串 str 第一个出现的位置，从位置 pos 开始。如果 substr 不是在 str 里面，返回 0。

```
mysql> select LOCATE('bar', 'foobarbar',5);
        -> 7
```

这函数是多字节可靠的。

INSTR(str,substr)

返回子串 substr 在字符串 str 中的第一个出现的位置。这与有 2 个参数形式的 LOCATE() 相同，除了参数被颠倒。

```
mysql> select INSTR('foobarbar', 'bar');
        -> 4
mysql> select INSTR('xbar', 'foobar');
        -> 0
```

这函数是多字节可靠的。

LPAD(str,len,padstr)

返回字符串 str，左面用字符串 padstr 填补直到 str 是 len 个字符长。

```
mysql> select LPAD('hi',4,'??');
        -> '??hi'
```

RPAD(str,len,padstr)

返回字符串 str，右面用字符串 padstr 填补直到 str 是 len 个字符长。

```
mysql> select RPAD('hi',5,'?');
        -> 'hi???'
```

LEFT(str,len)

返回字符串 str 的最左面 len 个字符。

```
mysql> select LEFT('foobarbar', 5);
        -> 'fooba'
```

该函数是多字节可靠的。

RIGHT(str,len)

返回字符串 str 的最右面 len 个字符。

```
mysql> select RIGHT('foobarbar', 4);
        -> 'rbar'
```

该函数是多字节可靠的。

SUBSTRING(str,pos,len)

SUBSTRING(str FROM pos FOR len)

MID(str,pos,len)

从字符串 str 返回一个 len 个字符的子串，从位置 pos 开始。使用 FROM 的变种形式是 ANSI SQL92 语法。

```
mysql> select SUBSTRING('Quadratically',5,6);
    -> 'ratica'
```

该函数是多字节可靠的。

SUBSTRING(str,pos)

SUBSTRING(str FROM pos)

从字符串 str 的起始位置 pos 返回一个子串。

```
mysql> select SUBSTRING('Quadratically',5);
    -> 'ratically'
mysql> select SUBSTRING('foobarbar' FROM 4);
    -> 'barbar'
```

该函数是多字节可靠的。

SUBSTRING_INDEX(str,delim,count)

返回从字符串 str 的第 count 个出现的分隔符 delim 之后的子串。如果 count 是正数，返回最后的分隔符到左边(从左边数) 的所有字符。如果 count 是负数，返回最后的分隔符到右边的所有字符(从右边数)。

```
mysql> select SUBSTRING_INDEX('www.mysql.com', '.', 2);
    -> 'www.mysql'
mysql> select SUBSTRING_INDEX('www.mysql.com', '.', -2);
    -> 'mysql.com'
```

该函数对多字节是可靠的。

LTRIM(str)

返回删除了其前置空格字符的字符串 str。

```
mysql> select LTRIM('    barbar');
    -> 'barbar'
```

RTRIM(str)

返回删除了其拖后空格字符的字符串 str。

```
mysql> select RTRIM('barbar     ');
    -> 'barbar'
```

该函数对多字节是可靠的。

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

返回字符串 str，其所有 remstr 前缀或后缀被删除了。如果没有修饰符 BOTH、LEADING 或 TRAILING 给出，BOTH 被假定。如果 remstr 没被指定，空格被删除。

```
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'
```

该函数对多字节是可靠的。

SOUNDEX(str)

返回 str 的一个同音字符串。听起来“大致相同”的 2 个字符串应该有相同的同音字符串。一个“标准”的同音字符串长是 4 个字符，但是 SOUNDEX()函数返回一个任意长的字符串。你可以在结果上使用 SUBSTRING()得到一个“标准”的 同音串。所有非数字字母字符在给定的字符串中被忽略。所有在 A-Z 之外的字符国际字母被当作元音。

```
mysql> select SOUNDEX('Hello');
        -> 'H400'
mysql> select SOUNDEX('Quadratically');
        -> 'Q36324'
```

SPACE(N)

返回由 N 个空格字符组成的一个字符串。

```
mysql> select SPACE(6);
        -> '      '
```

REPLACE(str,from_str,to_str)

返回字符串 str，其字符串 from_str 的所有出现由字符串 to_str 代替。

```
mysql> select REPLACE('www.mysql.com', 'w', 'Ww');
        -> 'WwWwWw.mysql.com'
```

该函数对多字节是可靠的。

REPEAT(str,count)

返回由重复 countTimes 次的字符串 str 组成的一个字符串。如果 count <= 0，返回一个空字符串。如果 str 或 count 是 NULL，返回 NULL。

```
mysql> select REPEAT('MySQL', 3);
        -> 'MySQLMySQLMySQL'
```

REVERSE(str)

返回颠倒字符顺序的字符串 str。

```
mysql> select REVERSE('abc');
        -> 'cba'
```

该函数对多字节可靠的。

INSERT(str,pos,len,newstr)

返回字符串 str，在位置 pos 起始的子串且 len 个字符长得子串由字符串 newstr 代替。

```
mysql> select INSERT('Quadratic', 3, 4, 'What');
        -> 'QuWhattic'
```

该函数对多字节是可靠的。

ELT(N,str1,str2,str3,...)

如果 N= 1，返回 str1，如果 N= 2，返回 str2，等等。如果 N 小于 1 或大于参数个数，

返回 NULL。ELT()是 FIELD()反运算。

```
mysql> select ELT(1, 'ej', 'Heja', 'hej', 'foo');
        -> 'ej'
mysql> select ELT(4, 'ej', 'Heja', 'hej', 'foo');
        -> 'foo'
```

FIELD(str,str1,str2,str3,...)

返回 str 在 str1, str2, str3, ...清单的索引。如果 str 没找到，返回 0。FIELD()是 ELT()反运算。

```
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)

如果字符串 str 在由 N 子串组成的表 strlist 之中，返回一个 1 到 N 的值。一个字符串表是被“,”分隔的子串组成的一个字符串。如果第一个参数是一个常数字符串并且第二个参数是一种类型为 SET 的列，FIND_IN_SET()函数被优化而使用位运算！如果 str 不是在 strlist 里面或如果 strlist 是空字符串，返回 0。如果任何一个参数是 NULL，返回 NULL。如果第一个参数包含一个“,”，该函数将工作不正常。

```
mysql> SELECT FIND_IN_SET('b','a,b,c,d');
        -> 2
```

MAKE_SET(bits,str1,str2,...)

返回一个集合 (包含由“,”字符分隔的子串组成的一个字符串)，由相应的位在 bits 集合中的的字符串组成。str1 对应于位 0，str2 对应位 1，等等。在 str1, str2, ...中的 NULL 串不添加到结果中。

```
mysql> SELECT MAKE_SET(1,'a','b','c');
        -> 'a'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice','world');
        -> 'hello,world'
mysql> SELECT MAKE_SET(0,'a','b','c');
        -> ''
```

EXPORT_SET(bits,on,off,[separator,[number_of_bits]])

返回一个字符串，在这里对于在“bits”中设定每一位，你得到一个“on”字符串，并且对于每个复位(reset)的位，你得到一个“off”字符串。每个字符串用“separator”分隔(缺省“,”)，并且只有“bits”的“number_of_bits” (缺省 64)位被使用。

```
mysql> select EXPORT_SET(5,'Y','N',',',4)
        -> Y,N,Y,N
```

LCASE(str)

LOWER(str)

返回字符串 str，根据当前字符集映射(缺省是ISO-8859-1 Latin1)把所有的字符改变成小写。该函数对多字节是可靠的。

```
mysql> select LCASE('QUADRATICALLY');
        -> 'quadratically'
```

UCASE(str)

UPPER(str)

返回字符串 str，根据当前字符集映射(缺省是ISO-8859-1 Latin1)把所有的字符改变成大写。该函数对多字节是可靠的。

```
mysql> select UCASE('Hej');
        -> 'HEJ'
```

该函数对多字节是可靠的。

LOAD_FILE(file_name)

读入文件并且作为一个字符串返回文件内容。文件必须在服务器上，你必须指定到文件的完整路径名，而且你必须有 file 权限。文件必须所有内容都是可读的并且小于 max_allowed_packet。如果文件不存在或由于上面原因之一不能被读出，函数返回 NULL。

```
mysql> UPDATE table_name
          SET blob_column=LOAD_FILE("/tmp/picture")
          WHERE id=1;
```

MySQL 必要时自动变换数字为字符串，并且反过来也如此：

```
mysql> SELECT 1+"1";
        -> 2
mysql> SELECT CONCAT(2,' test');
        -> '2 test'
```

如果你想要明确地变换一个数字到一个字符串，把它作为参数传递到 CONCAT()。

如果字符串函数提供一个二进制字符串作为参数，结果字符串也是一个二进制字符串。被变换到一个字符串的数字被当作是一个二进制字符串。这仅影响比较。

1.4.11 日期和时间函数

对于每个类型拥有的值范围以及并且指定日期何时间值的有效格式的描述见 1.3.6 日期和时间类型。

这里是一个使用日期函数的例子。下面的查询选择了所有记录，其 date_col 的值是在最后 30 天以内：

```
mysql> SELECT something FROM table
          WHERE TO_DAYS(NOW()) - TO_DAYS(date_col) <= 30;
```

DAYOFWEEK(date)

返回日期 date 的星期索引(1=星期天，2=星期一，……7=星期六)。这些索引值对应于 ODBC 标准。

```
mysql> select DAYOFWEEK('1998-02-03');
        -> 3
```

WEEKDAY(date)

返回 date 的星期索引(0=星期一，1=星期二，……6= 星期天)。

```
mysql> select WEEKDAY('1997-10-04 22:23:00');
        -> 5
mysql> select WEEKDAY('1997-11-05');
        -> 2
```

DAYOFMONTH(date)

返回 date 的月份中日期，在 1 到 31 范围内。

```
mysql> select DAYOFMONTH('1998-02-03');
        -> 3
```

DAYOFYEAR(date)

返回 date 在一年中的日数，在 1 到 366 范围内。

```
mysql> select DAYOFYEAR('1998-02-03');
        -> 34
```

MONTH(date)

返回 date 的月份，范围 1 到 12。

```
mysql> select MONTH('1998-02-03');
        -> 2
```

DAYNAME(date)

返回 date 的星期名字。

```
mysql> select DAYNAME("1998-02-05");
        -> 'Thursday'
```

MONTHNAME(date)

返回 date 的月份名字。

```
mysql> select MONTHNAME("1998-02-05");
        -> 'February'
```

QUARTER(date)

返回 date 一年中的季度，范围 1 到 4。

```
mysql> select QUARTER('98-04-01');
        -> 2
```

WEEK(date)

WEEK(date,first)

对于星期天是一周的第一天的地方，有一个单个参数，返回 date 的周数，范围在 0 到 52。2 个参数形式 WEEK()允许你指定星期是否开始于星期天或星期一。如果第二个参数是 0，星期从星期天开始，如果第二个参数是 1，从星期一开始。

```
mysql> select WEEK('1998-02-20');
```

-> 7

mysql> select WEEK('1998-02-20',0);

-> 7

mysql> select WEEK('1998-02-20',1);

-> 8

YEAR(date)

返回 date 的年份，范围在 1000 到 9999。

mysql> select YEAR('98-02-03');

-> 1998

HOUR(time)

返回 time 的小时，范围是 0 到 23。

mysql> select HOUR('10:05:03');

-> 10

MINUTE(time)

返回 time 的分钟，范围是 0 到 59。

mysql> select MINUTE('98-02-03 10:05:03');

-> 5

SECOND(time)

回来 time 的秒数，范围是 0 到 59。

mysql> select SECOND('10:05:03');

-> 3

PERIOD_ADD(P,N)

增加 N 个月到阶段 P（以格式 YYMM 或 YYYYMM)。以格式 YYYYMM 返回值。注意阶段参数 P 不是日期值。

mysql> select PERIOD_ADD(9801,2);

-> 199803

PERIOD_DIFF(P1,P2)

返回在时期 P1 和 P2 之间月数，P1 和 P2 应该以格式 YYMM 或 YYYYMM。注意，时期参数 P1 和 P2 不是日期值。

mysql> select PERIOD_DIFF(9802,199703);

-> 11

DATE_ADD(date,INTERVAL expr type)

DATE_SUB(date,INTERVAL expr type)

ADDDATE(date,INTERVAL expr type)

SUBDATE(date,INTERVAL expr type)

这些功能执行日期运算。对于 MySQL 3.22，他们是新的。ADDDATE()和 SUBDATE()是 DATE_ADD()和 DATE_SUB()的同义词。在 MySQL 3.23 中，你可以使用+和-而不是 DATE_ADD()和 DATE_SUB()。（见例子）date 是一个指定开始日期的 DATETIME 或 DATE

值，expr 是指定加到开始日期或从开始日期减去的间隔值一个表达式，expr 是一个字符串；它可以以一个“-”开始表示负间隔。type 是一个关键词，指明表达式应该如何被解释。EXTRACT(type FROM date)函数从日期中返回“type”间隔。下表显示了 type 和 expr 参数怎样被关联：

表附 1-12 type 的值

type 值	含义	期望的 expr 格式
SECOND	秒	SECONDS
MINUTE	分钟	MINUTES
HOUR	时间	HOURS
DAY	天	DAYS
MONTH	月	MONTHS
YEAR	年	YEARS
MINUTE_SECOND	分钟和秒	"MINUTES:SECONDS"
HOUR_MINUTE	小时和分钟	"HOURS:MINUTES"
DAY_HOUR	天和小时	"DAYS
YEAR_MONTH	年和月	"YEARS-MONTHS"
HOUR_SECOND	小时,	分钟,
DAY_MINUTE	天,	小时,
DAY_SECOND	天,	小时,

MySQL 在 expr 格式中允许任何标点分隔符。表示显示的是建议的分隔符。如果 date 参数是一个 DATE 值并且你的计算仅仅包含 YEAR、MONTH 和 DAY 部分(即，没有时间部分)，结果是一个 DATE 值。否则结果是一个 DATETIME 值。

```
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 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
```

如果你指定太短的间隔值(不包括 type 关键词期望的间隔部分)，MySQL 假设你省掉了间隔值的最左面部分。例如，如果你指定一个 type 是 DAY_SECOND，值 expr 被希望有天、小时、分钟和秒部分。如果你象"1:10"这样指定值，MySQL 假设日子和小时部分是丢失的并且值代表分钟和秒。换句话说，"1:10" DAY_SECOND 以它等价于"1:10" MINUTE_SECOND 的方式解释，这对那 MySQL 解释 TIME 值表示经过的时间而非作为一天的时间的方式有二义性。如果你使用确实不正确的日期，结果是 NULL。如果你增加 MONTH、YEAR_MONTH 或 YEAR 并且结果日期大于新月份的最大值天数，日子在新月用最大的天调整。

```
mysql> select DATE_ADD('1998-01-30', Interval 1 month);
    -> 1998-02-28
```

注意，从前面的例子中词 INTERVAL 和 type 关键词不是区分大小写的。

TO_DAYS(date)

给出一个日期 date，返回一个天数(从 0 年的天数)。

```
mysql> select TO_DAYS(950501);
    -> 728779
mysql> select TO_DAYS('1997-10-07');
    -> 729669
```

TO_DAYS()不打算用于使用格列高里历(1582)出现前的值。

FROM_DAYS(N)

给出一个天数 N，返回一个 DATE 值。

```
mysql> select FROM_DAYS(729669);
```

-> '1997-10-07'

TO_DAYS()不打算用于使用格列高里历(1582)出现前的值。

DATE_FORMAT(date,format)

根据 format 字符串格式化 date 值。下列修饰符可以被用在 format 字符串中：

- %M 月名字(January……December)
- %W 星期名字(Sunday……Saturday)
- %D 有英语前缀的月份的日期(1st, 2nd, 3rd, 等等。)
- %Y 年, 数字, 4 位
- %y 年, 数字, 2 位
- %a 缩写的星期名字(Sun……Sat)
- %d 月份中的天数, 数字(00……31)
- %e 月份中的天数, 数字(0……31)
- %m 月, 数字(01……12)
- %c 月, 数字(1……12)
- %b 缩写的月份名字(Jan……Dec)
- %j 一年中的天数(001……366)
- %H 小时(00……23)
- %k 小时(0……23)
- %h 小时(01……12)
- %I 小时(01……12)
- %l 小时(1……12)
- %i 分钟, 数字(00……59)
- %r 时间,12 小时(hh:mm:ss [AP]M)
- %T 时间,24 小时(hh:mm:ss)
- %S 秒(00……59)
- %s 秒(00……59)
- %p AM 或 PM
- %w 一个星期中的天数(0=Sunday ……6=Saturday)
- %U 星期(0……52), 这里星期天是星期的第一天
- %u 星期(0……52), 这里星期一是星期的第一天
- %% 一个文字“%”。

所有的其他字符不做解释被复制到结果中。

```
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'

MySQL3.23 中，在格式修饰符字符前需要%。在 MySQL 更早的版本中，%是可选的。

TIME_FORMAT(time,format)

这象上面的 DATE_FORMAT()函数一样使用，但是 format 字符串只能包含处理小时、分钟和秒的那些格式修饰符。其他修饰符产生一个 NULL 值或 0。

CURDATE()

CURRENT_DATE

以'YYYY-MM-DD'或 YYYYMMDD 格式返回今天日期值，取决于函数是在一个字符串还是数字上下文被使用。

mysql> select CURDATE();

-> '1997-12-15'

mysql> select CURDATE() + 0;

-> 19971215

CURTIME()

CURRENT_TIME

以'HH:MM:SS'或 HHMMSS 格式返回当前时间值，取决于函数是在一个字符串还是在数字的上下文被使用。

mysql> select CURTIME();

-> '23:50:26'

mysql> select CURTIME() + 0;

-> 235026

NOW()

SYSDATE()

CURRENT_TIMESTAMP

以'YYYY-MM-DD HH:MM:SS'或 YYYYMMDDHHMMSS 格式返回当前的日期和时间，取决于函数是在一个字符串还是在数字的上下文被使用。

mysql> select NOW();

-> '1997-12-15 23:50:26'

mysql> select NOW() + 0;

-> 19971215235026

UNIX_TIMESTAMP()

UNIX_TIMESTAMP(date)

如果没有参数调用，返回一个 Unix 时间戳记(从'1970-01-01 00:00:00'GMT 开始的秒数)。如果 UNIX_TIMESTAMP()用一个 date 参数被调用，它返回从'1970-01-01 00:00:00' GMT 开始的秒数值。date 可以是一个 DATE 字符串、一个 DATETIME 字符串、一个 TIMESTAMP

或以 YYMMDD 或 YYYYMMDD 格式的本地时间的一个数字。

```
mysql> select UNIX_TIMESTAMP();
        -> 882226357
mysql> select UNIX_TIMESTAMP('1997-10-04 22:23:00');
        -> 875996580
```

当 UNIX_TIMESTAMP 被用于一个 TIMESTAMP 列，函数将直接接受值，没有隐含的“string-to-unix-timestamp”变换。

FROM_UNIXTIME(unix_timestamp)

以'YYYY-MM-DD HH:MM:SS'或 YYYYMMDDHHMMSS 格式返回 unix_timestamp 参数所表示的值，取决于函数是在一个字符串还是或数字上下文中被使用。

```
mysql> select FROM_UNIXTIME(875996580);
        -> '1997-10-04 22:23:00'
mysql> select FROM_UNIXTIME(875996580) + 0;
        -> 19971004222300
```

FROM_UNIXTIME(unix_timestamp,format)

返回表示 Unix 时间标记的一个字符串，根据 format 字符串格式化。format 可以包含与 DATE_FORMAT()函数列出的条目同样的修饰符。

```
mysql> select FROM_UNIXTIME(UNIX_TIMESTAMP(),
                            '%Y %D %M %h:%i:%s %x');
        -> '1997 23rd December 03:43:30 x'
```

SEC_TO_TIME(seconds)

返回 seconds 参数，变换成小时、分钟和秒，值以'HH:MM:SS'或 HHMMSS 格式化，取决于函数是在一个字符串还是在数字上下文中被使用。

```
mysql> select SEC_TO_TIME(2378);
        -> '00:39:38'
mysql> select SEC_TO_TIME(2378) + 0;
        -> 3938
```

TIME_TO_SEC(time)

返回 time 参数，转换成秒。

```
mysql> select TIME_TO_SEC('22:23:00');
        -> 80580
mysql> select TIME_TO_SEC('00:39:38');
        -> 2378
```

1.4.12 其他函数

DATABASE()

返回当前的数据库名字。

```
mysql> select DATABASE();
```

-> 'test'

如果没有当前的数据库，DATABASE()返回空字符串。

USER()

SYSTEM_USER()

SESSION_USER()

返回当前 MySQL 用户名。

mysql> select USER();

-> 'davida@localhost'

在 MySQL 3.22.11 或以后版本中，这包括用户名和客户主机名。你可以象这样只提取用户名部分(值是否包括主机名部分均可工作)：

mysql> select substring_index(USER(),"@",1);

-> 'davida'

PASSWORD(str)

从纯文本口令 str 计算一个口令字符串。该函数被用于为了在 user 授权表的 Password 列中存储口令而加密 MySQL 口令。

mysql> select PASSWORD('badpwd');

-> '7f84554057dd964b'

PASSWORD()加密是非可逆的。PASSWORD()不以与 Unix 口令加密的相同的方法执行口令加密。你不应该假定如果你的 Unix 口令和你的 MySQL 口令是一样的，PASSWORD()将导致与在 Unix 口令文件存储的相同的加密值。见 ENCRYPT()。

ENCRYPT(str[,salt])

使用 Unix crypt()系统调用加密 str。salt 参数应该是一个有 2 个字符的字符串。(MySQL 3.22.16 中，salt 可以长于 2 个字符。)

mysql> select ENCRYPT("hello");

-> 'VxuFAJXVARROc'

如果 crypt()在你的系统上不可用，ENCRYPT()总是返回 NULL。ENCRYPT()只保留 str 起始 8 个字符而忽略所有其他，至少在某些系统上是这样。这将由底层的 crypt()系统调用的行为决定。

ENCODE(str,pass_str)

使用 pass_str 作为口令加密 str。为了解密结果，使用 DECODE()。结果是一个二进制字符串，如果你想要在列中保存它，使用一个 BLOB 列类型。

DECODE(crypt_str,pass_str)

使用 pass_str 作为口令解密加密的字符串 crypt_str。crypt_str 应该是一个由 ENCODE()返回的字符串。

MD5(string)

对字符串计算 MD5 校验和。值作为一个 32 长的十六进制数字被返回可以，例如用作哈希(hash)键。

mysql> select MD5("testing")

-> 'ae2b1fca515949e5d54fb22b8ed95575'

这是一个“RSA 数据安全公司的 MD5 消息摘要算法”。

LAST_INSERT_ID([expr])

返回被插入一个 AUTO_INCREMENT 列的最后一个自动产生的值。

mysql> select LAST_INSERT_ID();

-> 195

产生的最后 ID 以每个连接为基础在服务器被维护，它不会被其他客户改变。如果你更新另外一个有非魔术值(即，不是 NULL 和不是 0 的一个值)的 AUTO_INCREMENT 列，它甚至不会被改变。如果 expr 作为一个参数在一个 UPDATE 子句的 LAST_INSERT_ID()里面给出，那么参数值作为一个 LAST_INSERT_ID()值被返回。这可以用来模仿顺序：首先创建表：

mysql> create table sequence (id int not null);

mysql> insert into sequence values (0);

然后表能被用来产生顺序号，象这样：

mysql> update sequence set id=LAST_INSERT_ID(id+1);

你可以不调用 LAST_INSERT_ID()而产生顺序，但是这样使用函数的实用程序在服务器上自动维护 ID 值作为最后自动产生的值。你可以检索新的 ID 值，就像你能读入正常 MySQL 中的任何正常的 AUTO_INCREMENT 值一样。例如，LAST_INSERT_ID()（没有一个参数)将返回新 ID。C API 函数 mysql_insert_id()也可被用来得到值。

FORMAT(X,D)

格式化数字 X 为类似于格式'#,###,###.##'，四舍五入到 D 为小数。如果 D 为 0，结果将没有小数点和小数部分。

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'

VERSION()

返回表明 MySQL 服务器版本的一个字符串。

mysql> select VERSION();

-> '3.22.19b-log'

GET_LOCK(str,timeout)

试图获得由字符串 str 给定的一个名字的锁定，第二个 timeout 为超时。如果锁定成功获得，返回 1，如果尝试超时了，返回 0，或如果发生一个错误，返回 NULL(例如从存储器溢出或线程用 mysqladmin kill 被杀死)。当你执行 RELEASE_LOCK()时、执行一个新的 GET_LOCK()或线程终止时，一个锁定被释放。该函数可以用来实现应用锁或模拟记录锁，它阻止其他客户用同样名字的锁定请求；赞成一个给定的锁定字符串名字的客户可以使用

字符串执行子协作建议的锁定。

```
mysql> select GET_LOCK("lock1",10);
        -> 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，因为锁"lock1"自动地被第二个 GET_LOCK()调用释放。

RELEASE_LOCK(str)

释放字符串 str 命名的通过 GET_LOCK()获得的锁。如果锁被释放，返回 1，如果锁没被这个线程锁定(在此情况下锁没被释放)返回 0，并且如果命名的锁不存在，返回 NULL。如果锁从来没有通过调用 GET_LOCK()获得或如果它已经被释放了，锁将不存在。

BENCHMARK(count,expr)

BENCHMARK()函数重复 countTimes 次执行表达式 expr，它可以用于计时 MySQL 处理表达式有多快。结果值总是 0。意欲用于 mysql 客户，它报告查询的执行时间。

```
mysql> select BENCHMARK(1000000,encode("hello","goodbye"));
+----------------------------------------+
| BENCHMARK(1000000,encode("hello","goodbye")) |
+----------------------------------------+
|                                      0 |
+----------------------------------------+
1 row in set (4.74 sec)
```

报告的时间是客户端的经过时间，不是在服务器端的 CPU 时间。执行 BENCHMARK()若干次可能是明智的，并且注意服务器机器的负载有多重来解释结果。

1.4.13 与 GROUP BY 子句一起使用的函数

如果你在不包含 GROUP BY 子句的一个语句中使用聚合函数，它等价于聚合所有行。

COUNT(expr)

返回由一个 SELECT 语句检索出来的行的非 NULL 值的数目。

```
mysql> select student.student_name,COUNT(*)
        from student,course
        where student.student_id=course.student_id
        GROUP BY student_name;
```

COUNT(*)在它返回的检索出来的行数目上有些不同，不管他们是否包含 NULL 值。如果 SELECT 从一个表检索，或没有检索出其他列并且没有 WHERE 子句，COUNT(*)被

优化以便快速地返回。例如：

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

COUNT(DISTINCT expr,[expr...])

返回一个不同值的数目。

```
mysql> select COUNT(DISTINCT results) from student;
```

在 MySQL 中，你可以通过给出一个表达式列表以得到不同的表达式组合的数目。在 ANSI SQL 中，你可能必须在 CODE(DISTINCT ..)内进行所有表达式的连接。

AVG(expr)

返回 expr 的平均值。

```
mysql> select student_name, AVG(test_score)
            from student
            GROUP BY student_name;
```

MIN(expr)

MAX(expr)

返回 expr 的最小或最大值。MIN()和 MAX()可以有一个字符串参数;在这种的情况下，他们返回最小或最大的字符串值。

```
mysql> select student_name, MIN(test_score), MAX(test_score)
            from student
            GROUP BY student_name;
```

SUM(expr)

返回 expr 的和。注意，如果返回的集合没有行，它返回 NULL！

STD(expr)

STDDEV(expr)

返回 expr 标准差(deviation)。这是对 ANSI SQL 的扩展。该函数的形式 STDDEV()是提供与 Oracle 的兼容性。

BIT_OR(expr)

返回 expr 里所有位的位或。计算用 64 位(BIGINT)精度进行。

BIT_AND(expr)

返回 expr 里所有位的位与。计算用 64 位(BIGINT)精度进行。

MySQL 扩展了 GROUP BY 的用法。你可以不出现在的 GROUP BY 部分的 SELECT 表达式中使用列或计算，这表示这个组的任何可能值。你可以使用它是性能更好，避免在不必要的项目上排序和分组。例如，你在下列查询中不需要在 customer.name 上聚合：

```
mysql> select order.custid,customer.name,max(payments)
        from order,customer
        where order.custid = customer.custid
        GROUP BY order.custid;
```

在 ANSI SQL 中，你将必须将 customer.name 加到 GROUP BY 子句。在 MySQL 中，名字是冗余的。

如果你从 GROUP BY 部分省略的列在组中不是唯一的，不要使用这个功能。

在某些情况下，你可以使用 MIN()和 MAX()获得一个特定的列值，即使它不是唯一的。下例给出从包含 sort 列中最小值的行的 column 值：

substr(MIN(concat(sort,space(6-length(sort)),column),7,length(column)))

注意，如果你正在使用 MySQL 3.22(或更早)或如果你正在试图遵从 ANSI SQL，你不能在 GROUP BY 或 ORDER BY 子句中使用表达式。你可以通过使用表达式的一个别名解决此限制：

```
mysql> SELECT id,FLOOR(value/100) AS val FROM tbl_name
          GROUP BY id,val ORDER BY val;
```

在 MySQL3.23 中，你可以这样做：

```
mysql> SELECT id,FLOOR(value/100) FROM tbl_name ORDER BY RAND();
```

1.5 CREATE DATABASE 句法

CREATE DATABASE db_name

CREATE DATABASE 用给定的名字创建一个数据库。允许的数据库名字规则在 1.1.5 数据库、表、索引、列和别名命名中给出。如果数据库已经存在，发生一个错误。

在 MySQL 中的数据库实现成包含对应数据库中表的文件的目录。因为数据库在初始创建时没有任何表，CREATE DATABASE 语句只是在 MySQL 数据目录下面创建一个目录。

你也可以用 mysqladmin 创建数据库。

1.6 DROP DATABASE 句法

DROP DATABASE [IF EXISTS] db_name

DROP DATABASE 删除数据库中的所有表和数据库。要小心地使用这个命令！

DROP DATABASE 返回从数据库目录被删除的文件的数目。通常，这3倍于表的数量，因为每张表对应于一个“.MYD”文件、一个“.MYI”文件和一个“.frm”文件。

在 MySQL 3.22 或以后版本中，你可以使用关键词 IF EXISTS 阻止一个错误的发生，如果数据库不存在。

你也可以用 mysqladmin 丢弃数据库。

1.7 CREATE TABLE 句法

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name [(create_definition,...)]
[table_options] [select_statement]
create_definition:
  col_name type [NOT NULL | NULL] [DEFAULT default_value]
[AUTO_INCREMENT]
            [PRIMARY KEY] [reference_definition]
  or    PRIMARY KEY (index_col_name,...)
```

```
    or      KEY [index_name] (index_col_name,...)
    or      INDEX [index_name] (index_col_name,...)
    or      UNIQUE [INDEX] [index_name] (index_col_name,...)
    or      [CONSTRAINT symbol] FOREIGN KEY index_name (index_col_name,...)
            [reference_definition]
    or      CHECK (expr)
type:
            TINYINT[(length)] [UNSIGNED] [ZEROFILL]
    or      SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
    or      MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
    or      INT[(length)] [UNSIGNED] [ZEROFILL]
    or      INTEGER[(length)] [UNSIGNED] [ZEROFILL]
    or      BIGINT[(length)] [UNSIGNED] [ZEROFILL]
    or      REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
    or      DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
    or      FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
    or      DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]
    or      NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]
    or      CHAR(length) [BINARY]
    or      VARCHAR(length) [BINARY]
    or      DATE
    or      TIME
    or      TIMESTAMP
    or      DATETIME
    or      TINYBLOB
    or      BLOB
    or      MEDIUMBLOB
    or      LONGBLOB
    or      TINYTEXT
    or      TEXT
    or      MEDIUMTEXT
    or      LONGTEXT
    or      ENUM(value1,value2,value3,...)
    or      SET(value1,value2,value3,...)
index_col_name:
            col_name [(length)]
reference_definition:
            REFERENCES tbl_name [(index_col_name,...)]
```

```
                [MATCH FULL | MATCH PARTIAL]
                [ON DELETE reference_option]
                [ON UPDATE reference_option]
reference_option:
        RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
table_options:
    TYPE = {ISAM | MYISAM | HEAP}
or  AUTO_INCREMENT = #
or  AVG_ROW_LENGTH = #
or  CHECKSUM = {0 | 1}
or  COMMENT = "string"
or  MAX_ROWS = #
or  MIN_ROWS = #
or  PACK_KEYS = {0 | 1}
or  PASSWORD = "string"
or  DELAY_KEY_WRITE = {0 | 1}
or      ROW_FORMAT= { default | dynamic | static | compressed }
select_statement:
    [IGNORE | REPLACE] SELECT ...  (Some legal select statement)
```

CREATE TABLE 在当前数据库中用给出的名字创建一个数据库表。允许的表名的规则在 1.1.5 数据库，表，索引，列和别名命名中给出。如果当前数据库不存在或如果表已经存在，出现一个错误。

在 MySQL3.22 或以后版本中，表名可以被指定为 db_name.tbl_name，不管有没有当前的数据库都可以。

在 MySQL3.23 中，当你创建一张表时，你可以使用 TEMPORARY 关键词。如果一个连接死掉，临时表将自动被删除，并且其名字是按连接命名。这意味着，2 个不同的连接能使用相同的暂时表的名字而不会彼此冲突或与相同名字的现有数据库表冲突。(现有的表被隐蔽直到临时表被删除)。

在 MySQL3.23 或以后版本中，你可以使用关键词 IF NOT EXISTS 以便如果表已经存在不发生一个错误。注意，无法证实表结构是相同的。

每张表 tbl_name 由在数据库目录的一些文件表示。在 MyISAM 类型的表的情况下，你将得到：

表附 1-13　表的存储文件

文件	目的
tbl_name.frm	表定义(表格)文件
tbl_name.MYD	数据文件

tbl_name.MYI	索引文件

对于各种列类型的性质的更多信息，见 1.3 列类型。

如果既不指定 NULL 也不指定 NOT NULL，列被视为指定了 NULL。

整型列可以有附加的属性 AUTO_INCREMENT。当你插入 NULL 值（推荐)或 0 到一个 AUTO_INCREMENT 列中时，列被设置为 value+1，在此 value 是当前表中的列的最大值。AUTO_INCREMENT 顺序从 1 开始。见 20.4.29 mysql_insert_id()。如果你删除了包含一个 AUTO_INCREMENT 列的最大值的行，值将被重新使用。如果你删除表中所有的行，顺序重新开始。注意：每个表只能有一个 AUTO_INCREMENT 列，并且它必须被索引。为了使做 MySQL 兼容一些 ODBC 应用程序，用下列查询你可以找出最后插入的行：

SELECT * FROM tbl_name WHERE auto_col IS NULL

NULL 值对于 TIMESTAMP 列的处理不同于其他列类型。你不能在一个 TIMESTAMP 列中存储一个文字 NULL；设置列为 NULL 将把它设成当前的日期和时间。因为 TIMESTAMP 列表现就这样，NULL 和 NOT NULL 属性不以一般方式运用并且如果你指定它们，将被忽略。在另一方面，为了使它 MySQL 客户更容易地使用 TIMESTAMP 列，服务器报告这样的列可以被赋值 NULL(它是对的)，尽管 TIMESTAMP 实际上绝不包含一个 NULL 值。当你使用 DESCRIBE tbl_name 得到有关你的表的描述时，你就会明白。注意，设置一个 TIMESTAMP 列为 0 不同于将它设置为 NULL，因为 0 是一个有效的 TIMESTAMP 值。

如果没有为列指定 DEFAULT 值，MySQL 自动地分配一个。如果列可以取 NULL 作为值，缺省值是 NULL。如果列被声明为 NOT NULL，缺省值取决于列类型：

- 对于没有声明 AUTO_INCREMENT 属性的数字类型，缺省值是 0。对于一个 AUTO_INCREMENT 列，缺省值是在顺序中的下一个值。
- 对于除 TIMESTAMP 的日期和时间类型，缺省值是该类型适当的“零”值。对于表中第一个 TIMESTAMP 列，缺省值是当前的日期和时间。见 1.3.6 日期和时间类型。
- 对于除 ENUM 的字符串类型，缺省是空字符串。对于 ENUM，缺省值是第一个枚举值。

KEY 是 INDEX 的一个同义词。

在 MySQL 中，一个 UNIQUE 键只能有不同的值。如果你试图用匹配现有行的键来增加新行，发生一个错误。

A PRIMARY KEY 是一个唯一 KEY，它有额外的限制，即所有的关键列必须被定义为 NOT NULL。在 MySQL 中，键被命名为 PRIMARY。一张表只能有一个 PRIMARY KEY。如果在表中你没有一个 PRIMARY KEY 并且一些应用程序要求 PRIMARY KEY，MySQL 将返回第一个 UNIQUE 键，它没有任何 NULL 列，作为 PRIMARY KEY。

一个 PRIMARY KEY 可以是一个多列索引。然而，你不能在一个列说明中使用 PRIMARY KEY 的关键字属性创建一个多列索引。这样做将仅仅标记单个列作为主键。你必须使用 PRIMARY KEY(index_col_name, ...)语法。

如果你不能给索引赋予一个名字，这个索引将赋予与第一个 index_col_name 相同的名字，用一个可选的 suffix(_2, _3, ...)使它唯一。你能使用 SHOW INDEX FROM tbl_name 看到一张表的索引名字。见 1.21 SHOW 句法(得到表、列等的信息)。

只有 MyISAM 表类型支持可以有 NULL 值的列上的索引。在其他情况下，你必须声明这样的列为 NOT NULL，否则导致一个错。

用 col_name(length)语法，你可以指定仅使用部分的 CHAR 或 VARCHAR 列的一个索引。这能使索引文件变得更小。见 1.3.9 列索引。

只有 MyISAM 表类型支持 BLOB 和 TEXT 列的索引。当在一个 BLOB 或 TEXT 列上放置索引时，你必须总是指定索引的长度：

CREATE TABLE test (blob_col BLOB, index(blob_col(10)));

当你与 TEXT 或 BLOB 列一起使用 ORDER BY 或 GROUP BY 时，只使用头 max_sort_length 个字节。见 1.3.4.2 BLOB 和 TEXT 类型。

FOREIGN KEY、CHECK 和 REFERENCES 子句实际上不做任何事情，其语法仅仅提供兼容性，使得它更容易从其他的 SQL 服务器移植代码并运行借助引用创建表的应用。

每个 NULL 列占据额外一位，取舍到最接近的字节。

最大记录长度以字节计可以如下计算：

row length = 1
+ (sum of column lengths)
+ (number of NULL columns + 7)/8
+ (number of variable-length columns)

table_options 和 SELECT 选项只在 MySQL 3.23 和以后版本中被实现。不同的表类型是：

- ISAM 原来的表处理器
- MyISAM 全新二进制可移植的表处理器
- HEAP 用于该表的数据仅仅存储在内存中

见 3.4 有关数据表的操作。其他表选项被用来优化表的行为。在大多数情况下，你不必指定他们任何一个。选项对所有表都适用，如果不是则说明。

- AUTO_INCREMENT 你想要为你的表设定的下一个 auto_increment 值(MyISAM)
- AVG_ROW_LENGTH 你的表的平均行长度的近似值。你只需要为有变长记录的表设置它。
- CHECKSUM 如果你想要 MySQL 对每行维持一个校验和(使表变得更慢以更新但是使它更容易找出损坏的表)设置它为 1 (MyISAM)
- COMMENT 对于你的表的一篇 60 个字符的注释

MAX_ROWS 你计划在表中存储的行的最大数目

- MIN_ROWS 你计划在表中存储的行的最小数目
- PACK_KEYS 如果你想要有更小的索引，将它设为 1。这通常使的更新更慢并且读取更快(MyISAM，ISAM)。

- PASSWORD　用一个口令加密.frm 文件。该选项在标准 MySQL 版本中不做任何事情。
- DELAY_KEY_WRITE　如果想要推迟关键表的更新直到表被关闭(MyISAM)，将它设置为 1。
- ROW_FORMAT　定义行应该如何被存储(为了将来)。

当你使用一个 MyISAM 表时，MySQL 使用 max_rows * avg_row_length 的乘积决定最终的表将有多大。如果你不指定上面的任何选项，对一个表的最大尺寸将是 4G(或 2G，如果你的操作系统仅支持 2G 的表)。

如果你在 CREATE 语句后指定一个 SELECT，MySQL 将为在 SELECT 中所有的单元创键新字段。例如：

```
mysql> CREATE TABLE test (a int not null auto_increment,
            primary key (a), key(b))
            TYPE=HEAP SELECT b,c from test2;
```

这将创建一个有 3 个列的 HEAP 表。注意如果在拷贝数据进表时发生任何错误，表将自动被删除。

1.1.1 隐含的列说明改变

在某些情况下，MySQL 隐含地改变在一个 CREATE TABLE 语句给出的一个列说明。（这也可能在 ALTER TABLE。）

长度小于 4 的 VARCHAR 被改变为 CHAR。

如果在一个表中的任何列有可变长度，结果是整个行是变长的。因此，如果一张表包含任何变长的列(VARCHAR、TEXT 或 BLOB)，所有大于 3 个字符的 CHAR 列被改变为 VARCHAR 列。这在任何方面都不影响你如何使用列；在 MySQL 中，VARCHAR 只是存储字符的一个不同方法。MySQL 实施这种改变，是因为它节省空间并且使表操作更快捷。见 10.6 选择一种表格类型。

TIMESTAMP 的显示尺寸必须是偶数且在 2 ～ 14 的范围内。如果你指定 0 显示尺寸或比 14 大，尺寸被强制为 14。从 1～13 范围内的奇数值尺寸被强制为下一个更大的偶数。

你不能在一个 TIMESTAMP 列里面存储一个文字 NULL；将它设为 NULL 将设置为当前的日期和时间。因为 TIMESTAMP 列表现就是这样，NULL 和 NOT NULL 属性不以一般的方式运用并且如果你指定他们，将被忽略。DESCRIBE tbl_name 总是报告该 TIMESTAMP 列可能赋予了 NULL 值。

MySQL 将其他 SQL 数据库供应商使用的某个列类型映射到 MySQL 类型。见 1.3.8 只用其他数据库引擎的类型。

如果你想要知道 MySQL 是否使用了除你指定的以外的一种列类型，在创建或改变你的表之后，发出一个 DESCRIBE tbl_name 语句即可。

如果你使用 myisampack 压缩一个表，可能会发生改变某些其他的列类型。

1.8 ALTER TABLE 句法

```
ALTER [IGNORE] TABLE tbl_name alter_spec [, alter_spec ...]
alter_specification:
        ADD [COLUMN] create_definition [FIRST | AFTER column_name ]
  or    ADD INDEX [index_name] (index_col_name,...)
  or    ADD PRIMARY KEY (index_col_name,...)
  or    ADD UNIQUE [index_name] (index_col_name,...)
  or    ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
  or    CHANGE [COLUMN] old_col_name create_definition
  or    MODIFY [COLUMN] create_definition
  or    DROP [COLUMN] col_name
  or    DROP PRIMARY KEY
  or    DROP INDEX index_name
  or    RENAME [AS] new_tbl_name
  or    table_options
```

ALTER TABLE 允许你修改一个现有表的结构。例如，你可以增加或删除列、创造或消去索引、改变现有列的类型、或重新命名列或表本身。你也能改变表的注释和表的类型。见 1.7 CREATE TABLE 句法。

如果你使用 ALTER TABLE 修改一个列说明但是 DESCRIBE tbl_name 显示你的列并没有被修改，这可能是 MySQL 因为在 1.1.1 隐含的列说明改变中描述的原因之一而忽略了你的修改。例如，如果你试图将一个 VARCHAR 改为 CHAR，MySQL 将仍然使用 VARCHAR，如果表包含其他变长的列。

ALTER TABLE 通过制作原来表的一个临时副本来工作。修改在副本上施行，然后原来的表被删除并且重新命名一个新的。这样做使得所有的修改自动地转向到新表，没有任何失败的修改。当 ALTER TABLE 正在执行时，原来的表可被其他客户读取。更新和写入表被延迟到新表准备好了为止。

为了使用 ALTER TABLE，你需要在表上的 select、insert、delete、update、create 和 drop 的权限。

IGNORE 是 MySQL 对 ANSI SQL92 的一个扩充，如果在新表中的唯一键上有重复，它控制 ALTER TABLE 如何工作。如果 IGNORE 没被指定，副本被放弃并且恢复原状。如果 IGNORE 被指定，那么对唯一键有重复的行，只有使用第一行；其余被删除。

你可以在单个 ALTER TABLE 语句中发出多个 ADD、ALTER、DROP 和 CHANGE 子句。这是 MySQL 对 ANSI SQL92 的一个扩充，SQL92 在每个 ALTER TABLE 语句中只允许一个子句。

CHANGE col_name、DROP col_name 和 DROP INDEX 是 MySQL 对 ANSI SQL92 的扩充。

MODIFY 是 Oracle 对 ALTER TABLE 的扩充。

可选的词 COLUMN 是一个纯粹的噪音且可以省略。

如果你使用 ALTER TABLE tbl_name RENAME AS new_name 而没有任何其他选项，

MySQL 简单地重命名对应于表 tbl_name 的文件。没有必要创建临时表。

create_definition 子句使用 CREATE TABLE 相同的 ADD 和 CHANGE 语法。注意语法包括列名字，不只列类型。见 1.7 CREATE TABLE 句法。

你可以使用 CHANGE old_col_name create_definition 子句重命名一个列。为了这样做，指定旧的和新的列名字和列当前有的类型。例如，重命名一个 INTEGER 列，从 a 到 b，你可以这样做：

```
mysql> ALTER TABLE t1 CHANGE a b INTEGER;
```

如果你想要改变列的类型而非名字，就算他们是一样的，CHANGE 语法仍然需要 2 个列名。例如：

```
mysql> ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
```

然而，在 MySQL3.22.16a，你也可以使用 MODIFY 来改变列的类型而不是重命名它：

```
mysql> ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
```

如果你使用 CHANGE 或 MODIFY 缩短一个列，一个索引存在于该列的部分(例如，如果你有一个 VARCHAR 列的头 10 个字符的索引)，你不能使列短于被索引的字符数目。

当你使用 CHANGE 或 MODIFY 改变一个列类型时，MySQL 尽可能试图很好地变换数据到新类型。

在 MySQL3.22 或以后，你能使用 FIRST 或 ADD ... AFTER col_name 在一个表的行内在一个特定的位置增加列。缺省是增加到最后一列。

ALTER COLUMN 为列指定新的缺省值或删除老的缺省值。如果老的缺省值被删除且列可以是 NULL，新缺省值是 NULL。如果列不能是 NULL，MySQL 赋予一个缺省值。缺省值赋值在 1.7 CREATE TABLE 句法中描述。

DROP INDEX 删除一个索引。这是 MySQL 对 ANSI SQL92 的一个扩充。

如果列从一张表中被丢弃，列也从他们是组成部分的任何索引中被删除。如果组成一个索引的所有列被丢弃，该索引也被丢弃。

DROP PRIMARY KEY 丢弃主索引。如果这样的索引不存在，它丢弃表中第一个 UNIQUE 索引。(如果没有明确地指定 PRIMARY KEY，MySQL 标记第一个 UNIQUE 键为 PRIMARY KEY。)

用 C API 函数 mysql_info()，你能找出多少记录被拷贝， 和(当使用 IGNORE 时)由于唯一键值的重复多少记录被删除。

FOREIGN KEY、CHECK 和 REFERENCES 子句实际上不做任何事情，他们的句法仅仅提供兼容性，使得更容易地从其他 SQL 服务器移植代码并且运行借助引用来创建表的应用程序。

这里是一个例子，显示了一些 ALTER TABLE 用法。我们以一个如下创建的表 t1 开始：

```
mysql> CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

重命名表，从 t1 到 t2:

```
mysql> ALTER TABLE t1 RENAME t2;
```

为了改变列 a，从 INTEGER 改为 TINYINT NOT NULL(名字一样)，并且改变列 b，从 CHAR(10)改为 CHAR(20)，同时重命名它，从 b 改为 c：

```
mysql> ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

增加一个新 TIMESTAMP 列，名为 d：

```
mysql> ALTER TABLE t2 ADD d TIMESTAMP;
```

在列 d 上增加一个索引，并且使列 a 为主键：

```
mysql> ALTER TABLE t2 ADD INDEX (d), ADD PRIMARY KEY (a);
```

删出列 c：

```
mysql> ALTER TABLE t2 DROP COLUMN c;
```

增加一个新的 AUTO_INCREMENT 整数列，命名为 c：

```
mysql> ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
            ADD INDEX (c);
```

注意，我们索引了 c，因为 AUTO_INCREMENT 柱必须被索引，并且另外我们声明 c 为 NOT NULL，因为索引了的列不能是 NULL。

当你增加一个 AUTO_INCREMENT 列时，自动地用顺序数字填入列值。

1.9 OPTIMIZE TABLE 句法

```
OPTIMIZE TABLE tbl_name
```

如果你删除了一个表的大部分或如果你用变长的行对一个表(有 VARCHAR、BLOB 或 TEXT 列的表)做了改变，应该使用 OPTIMZE TABLE。删除的记录以一个链接表维持并且随后的 INSERT 操作再次使用老记录的位置。你可以使用 OPTIMIZE TABLE 回收闲置的空间。

OPTIMIZE TABLE 通过制作原来的表的一个临时副本来工作。老的表子被拷贝到新表中(没有闲置的行)，然后原来的表被删除并且重命名一个新的。这样做使得所有更新自动转向新的表，没有任何失败的更新。当时 OPTIMIZE TABLE 正在执行时，原来的表可被另外的客户读取。对表的更新和写入延迟到新表是准备好为止。

1.10 DROP TABLE 句法

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

DROP TABLE 删除一个或多个数据库表。所有表中的数据和表定义均被删除，故小心使用这个命令!

在 MySQL 3.22 或以后版本，你可以使用关键词 IF EXISTS 类避免不存在表的一个错误发生。

1.11 DELETE 句法

```
DELETE [LOW_PRIORITY] FROM tbl_name
    [WHERE where_definition] [LIMIT rows]
```

DELETE 从 tbl_name 表中删除满足由 where_definition 给出的条件的行，并且返回删除记录的个数。

如果你发出一个没有 WHERE 子句的 DELETE，所有行都被删除。MySQL 通过创建

一个空表来完成，它比删除每行要快。在这种情况下，DELETE 返回零作为受影响记录的数目。(MySQL 不能返回实际上被删除的行数，因为进行再创建而不是打开数据文件。只要表定义文件“tbl_name.frm”是有效的，表才能这样被再创建，即使数据或索引文件破坏了)。

如果你确实想要知道在你正在删除所有行时究竟有对少记录被删除，并且愿意承受速度上的惩罚，你可以这种形式的一个 ELETE 语句：

mysql> DELETE FROM tbl_name WHERE 1>0;

注意这比没有 WHERE 子句的 DELETE FROM tbl_name 慢的多了，因为它一次删除一行。

如果你指定关键词 LOW_PRIORITY，DELETE 的执行被推迟到没有其他客户读取表后。

删除的记录以一个链接表维持并且随后的 INSERT 操作再次使用老的记录位置。为了回收闲置的空间并减小文件大小，使用 OPTIMIZE TABLE 语句或 myisamchk 实用程序重新组织表。OPTIMIZE TABLE 较容易，但是 myisamchk 更快。见 1.9 OPTIMIZE TABLE 句法和 13.4.3 表优化。

MySQL 对 DELETE 特定的 LIMIT rows 选项告诉服务器在控制被返回到客户之前，将要删除的最大行数，这可以用来保证一个特定 DELETE 命令不会花太多的时间。你可以简单地重复 DELETE 命令直到受影响的行数小于 LIMIT 值。

1.12 SELECT 句法

```
SELECT [STRAIGHT_JOIN] [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [HIGH_PRIORITY]
    [DISTINCT | DISTINCTROW | ALL]
    select_expression,...
    [INTO {OUTFILE | DUMPFILE} 'file_name' export_options]
    [FROM table_references
        [WHERE where_definition]
        [GROUP BY col_name,...]
        [HAVING where_definition]
        [ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC] ,...]
        [LIMIT [offset,] rows]
        [PROCEDURE procedure_name] ]
```

SELECT 被用来检索从一个或多个表中精选的行。select_expression 指出你想要检索的列。SELECT 也可以用来检索不引用任何表的计算行。例如：

```
mysql> SELECT 1 + 1;
    -> 2
```

所有使用的关键词必须精确地以上面的顺序给出。例如，一个 HAVING 子句必须跟在 GROUP BY 子句之后和 ORDER BY 子句之前。

一个 SELECT 表达式可以用一个 AS 给定一个别名，别名被用作表达式的列名并且能使用在 ORDER BY 或 HAVING 子句中。例如：

```
mysql> select concat(last_name,', ',first_name) AS full_name
    from mytable ORDER BY full_name;
```

FROM table_references 子句指出从哪个表中检索行。如果你命名多个表，你正在执行一个联结(join)。对于联结的句法信息，见 1.13 JOIN 句法。

你可以引用一个列为 col_name、tbl_name.col_name 或 db_name.tbl_name.col_name，你不必在一个 SELECT 语句中指定一个 tbl_name 或 db_name.tbl_name 是一个列引用的前缀，除非引用有二义性。见 1.1.5 数据库、表、索引、列和别名命名。对于二义性的例子要求更加显式的列引用格式。

一个表引用可以使用 tbl_name [AS] alias_name 起一个别名。

```
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;
```

精选输出的列可以用列名、列别名或列位置在 ORDER BY 和 GROUP BY 子句引用，列位置从 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;
```

为了以降序排列，把 DESC（下降)关键词加到 ORDER BY 子句中你要排序的列名前。缺省是升序；这也可以用 ASC 关键词明确指定。

HAVING 子句能引用任何列或在 select_expression 中命名的别名，它最后运用，就在项目被送到客户之前，没有优化。不要对因该在 WHERE 子句中的项目使用 HAVING。例如，不能写成这样：

```
mysql> select col_name from tbl_name HAVING col_name > 0;
```

相反写成这样：

```
mysql> select col_name from tbl_name WHERE col_name > 0;
```

在 MySQL 3.22.5 或以后，你也能这样写查询：

```
mysql> select user,max(salary) from users group by user HAVING max(salary)>10;
```

在里面更老的 MySQL 版本中，你能这样写：

```
mysql> select user,max(salary) AS sum from users
            group by user HAVING sum>10;
```

SQL_SMALL_RESULT、SQL_BIG_RESULT、STRAIGHT_JOIN 和 HIGH_PRIORITY 是 MySQL 对 ANSI SQL92 的扩展。

STRAIGHT_JOIN 强制优化器以其列在 FROM 子句的次序联结(join)表。如果优化器以非最佳次序联结表，你能使用它加速查询。见 1.22 EXPLAIN 句法(得到关于 SELECT 的信息)。

SQL_SMALL_RESULT 能与 GROUP BY 或 DISTINCT 一起使用告诉优化器结果集将很小。在这种情况下，MySQL 将使用快速临时表存储最终的表而不是使用排序。SQL_SMALL_RESULT 是一个 MySQL 扩展。

SQL_BIG_RESULT 能与 GROUP BY 或 DISTINCT 一起使用以告诉优化器结果集合将有很多行。在这种情况下，如果需要，MySQL 将直接使用基于磁盘的临时表。 MySQL 在这种情况下将选择用 GROUP BY 单元上的键值进行排序而不是做一个临时表。

HIGH_PRIORITY 将赋予 SELECT 比一个更新表的语句更高的优先级，你应该仅对非常快的并且必须一次完成的查询使用它。如果表为读而锁定或甚至有一个等待表释放的更新语句，一个 SELECT HIGH_PRIORITY 将运行。

LIMIT 子句可以被用来限制 SELECT 语句返回的行数。LIMIT 取 1 个或 2 个数字参数，如果给定 2 个参数，第一个指定要返回的第一行的偏移量，第二个指定返回行的最大数目。初始行的偏移量是 0(不是 1)。

```
mysql> select * from table LIMIT 5,10;  # Retrieve rows 6-15
```

如果给定一个参数，它指出返回行的最大数目。

```
mysql> select * from table LIMIT 5;      # Retrieve first 5 rows
```

换句话说，LIMIT n 等价于 LIMIT 0,n。

SELECT ... INTO OUTFILE 'file_name'格式的 SELECT 语句将选择的行写入一个文件。文件在服务器主机上被创建，并且不能是已经存在的(不管别的，这可阻止数据库表和文件例如“/etc/passwd”被破坏)。在服务器主机上你必须有 file 权限以使用这种 SELECT。SELECT ... INTO OUTFILE 是 LOAD DATA INFILE 逆操作；语句的 export_options 部分的语法与用在 LOAD DATA INFILE 语句中的 FIELDS 和 LINES 子句的相同。见 1.16 LOAD DATA INFILE 句法。在最终的文本文件中，只有下列字符由 ESCAPED BY 字符转义：

- ESCAPED BY 字符
- 在 FIELDS TERMINATED BY 中的第一个字符
- 在 LINES TERMINATED BY 中的第一个字符

另外，ASCII 0 被变换到 ESCAPED BY 后跟 0(ASCII 48)。上述的原因是你必须转义任何 FIELDS TERMINATED BY、ESCAPED BY 或 LINES TERMINATED BY 字符以便能可靠地能读回文件。ASCII 0 被转义使它更容易用分页器观看。因为最终的文件不必须遵循 SQL 句法，没有别的东西需要转义。

如果你使用 INTO DUMPFILE 而不是 INTO OUTFILE，MySQL 将只写一行到文件中，没有任何列或行结束并且没有任何转义。如果你想要在一个文件存储一个 blob，这是很有用的。

1.13 JOIN 句法

MySQL 支持下列用于 SELECT 语句的 JOIN 句法：

table_reference, table_reference

table_reference [CROSS] JOIN table_reference

table_reference INNER JOIN table_reference

table_reference STRAIGHT_JOIN table_reference

table_reference LEFT [OUTER] JOIN table_reference ON conditional_expr

table_reference LEFT [OUTER] JOIN table_reference USING (column_list)

table_reference NATURAL LEFT [OUTER] JOIN table_reference

{ oj table_reference LEFT OUTER JOIN table_reference ON conditional_expr }

上述最后的 LEFT OUTER JOIN 的句法只是为了与 ODBC 兼容而存在的。

一个表可以是使用 aliasedtbl_name AS alias_name 或 tbl_name alias_name 的起的别名。

```
mysql> select t1.name, t2.salary from employee AS t1, info AS t2
            where t1.name = t2.name;
```

INNER JOIN 和,（逗号)在语义上是等价的，都是进行一个在使用的表之间的全联结。通常，你指定表应该如何用 WHERE 条件联结起来。

ON 条件是可以用在一个 WHERE 子句形式的任何条件。

如果在一个 LEFT JOIN 中没有右表的匹配记录，一个所有列设置为 NULL 的行被用于右表。你可以使用这个事实指出表中在另一个表中没有对应记录的记录：

```
mysql> select table1.* from table1
            LEFT JOIN table2 ON table1.id=table2.id
            where table2.id is NULL;
```

这个例子找出在 table1 中所有的行，其 id 值在 table2 中不存在（即，所有 table1 中的在 table2 中没有对应行的行)。当然这假定 table2.id 被声明为 NOT NULL。

USING (column_list)子句命名一系列必须存在于两个表中的列。 例如一个 USING 子句：

A LEFT JOIN B USING (C1,C2,C3,...)

被定义成在语义上等同一个这样的 ON 表达式：

A.C1=B.C1 AND A.C2=B.C2 AND A.C3=B.C3,...

2 个表的 NATURAL LEFT JOIN 被定义为在语义上等同于一个有 USING 子句命名在两表中存在的所有列的一个 LEFT JOIN。

STRAIGHT_JOIN 等同于 JOIN，除了左表在右表之前被读入，这能用于这些情况，联结优化器将表的顺序放错了。

一些例子：

```
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;
```

1.14 INSERT 句法

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    VALUES (expression,...),(...),...
或  INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
或  INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] tbl_name
    SET col_name=expression, col_name=expression, ...
```

INSERT 把新行插入到一个存在的表中，INSERT ... VALUES 形式的语句基于明确指定的值插入行，INSERT ... SELECT 形式插入从其他表选择的行，有多个值表的 INSERT ... VALUES 的形式在 MySQL 3.22.5 或以后版本中支持，col_name=expression 语法在 MySQL 3.22.10 或以后版本中支持。

tbl_name 是行应该被插入其中的表。列名表或 SET 子句指出语句为那一列指定值。

如果你为 INSERT ... VALUES 或 INSERT ... SELECT 不指定列表，所有列的值必须在 VALUES()表或由 SELECT 提供。如果你不知道表中列的顺序，使用 DESCRIBE tbl_name 来找出。

任何没有明确地给出值的列被设置为它的缺省值。例如，如果你指定一个列表并没命名表中所有列，未命名的列被设置为它们的缺省值。缺省值赋值在 1.7 CREATE TABLE 句法中描述。

一个 expression 可以引用在一个值表先前设置的任何列。例如，你能这样：

```
mysql> INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
```

但不能这样：

```
mysql> INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
```

如果你指定关键词 LOW_PRIORITY，INSERT 的执行被推迟到没有其他客户正在读取表。在这种情况下，客户必须等到插入语句完成后，如果表频繁使用，它可能花很长时间。这与 INSERT DELAYED 让客马上继续正好相反。

如果你在一个有许多值行的 INSERT 中指定关键词 IGNORE，表中任何复制一个现有 PRIMARY 或 UNIQUE 键的行被忽略并且不被插入。如果你不指定 IGNORE，插入如果有任何复制现有关键值的行被放弃。你可用 C API 函数 mysql_info()检查多少行被插入到表中。

如果 MySQL 用 DONT_USE_DEFAULT_FIELDS 选项配置，INSERT 语句产生一个错误，除非你明确对需要一个非 NULL 值的所有列指定值。

- INSERT INTO ... SELECT 语句满足下列条件：
- 查询不能包含一个 ORDER BY 子句。
- INSERT 语句的目的表不能出现在 SELECT 查询部分的 FROM 子句，因为这在 ANSI SQL 中被禁止让从你正在插入的表中 SELECT。（问题是 SELECT 将可能发

现在同一个运行期间内先前被插入的记录。当使用子选择子句时，情况能很容易混淆）

- AUTO_INCREMENT 列象往常一样工作。

如果你使用 INSERT ... SELECT 或 INSERT ... VALUES 语句有多个值列表，你可以使用 C API 函数 mysql_info()得到查询的信息。信息字符串的格式如下：

Records: 100 Duplicates: 0 Warnings: 0

Duplicates 指出不能被插入的行的数量，因为他们与现有的唯一的索引值重复。Warnings 指出在出现某些问题时尝试插入列值的次数。在下列任何条件下都可能发生错误：

- 插入 NULL 到被声明了 NOT NULL 的列，列被设置为它的缺省值。
- 将超出列范围的值设置给一个数字列，值被剪切为范围内适当的端点值。
- 将数字列设成例如'10.34 a'的值，拖尾的垃圾被剥去并仍然是数字部分被插入。如果值根本不是一个数字，列被设置到 0。
- 把一个字符串插入到超过列的最大长度的一个 CHAR、VARCHAR、TEXT 或 BLOB 列中。值被截断为列的最大长度。
- 把一个对列类型不合法的值插入到一个日期或时间列。列被设置为该列类型适当的“零”值。

对于 INSERT 语句的 DELAYED 选项是 MySQL 专属的选项-如果你客户有不能等到 INSERT 完成，它是很有用的。当你为日记登录使用 MySQL 时，而且你也周期性地运行花很长时间完成的 SELECT 语句，这是一个常见的问题。DELAYED 在面 MySQL 3.22.15 中被引入，它是 MySQL 对 ANSI SQL92 的一个扩展。

当你使用 INSERT DELAYED 时，客户将马上准备好，并且当表不被任何其他的线程使用时，行将被插入。

另一个使用 INSERT DELAYED 的主要好处是从很多客户插入被捆绑在一起并且写进一个块。这比做很多单独的插入要来的快。

注意，当前排队的行只是存储在内存中，直到他们被插入到表中。这意味着，如果你硬要杀死 mysqld(kill -9)或如果 mysqld 出人意料地死掉，没被写进磁盘的任何排队的行被丢失！

下列详细描述当你为 INSERT 或 REPLACE 使用 DELAYED 选项时，发生什么。在这个描述中，“线程”是收到一个 INSERT DELAYED 命令的线程并且“处理器”是处理所有对于一个特定表的 INSERT DELAYED 语句：

- 当一个线程对一个表执行一个 DELAYED 语句时，如果不存在这样的处理程序，一个处理器线程被创建以处理对于该表的所有 DELAYED 语句。
- 线程检查处理程序是否已经获得了一个 DELAYED 锁；如果没有，它告诉处理程序去获得。即使其他的线程有在表上的一个 READ 或 WRITE 锁，也能获得 DELAYED 锁。然而，处理程序将等待所有 ALTER TABLE 锁或 FLUSH TABLES 以保证表结构是最新的。
- 线程执行 INSERT 语句，但不是将行写入表，它把最后一行的副本放进被处理器线程管理的一个队列。任何语法错误都能被线程发觉并报告给客户程序。

- 顾客不能报告结果行的重复次数或 AUTO_INCREMENT 值;它不能从服务器获得它们，因为 INSERT 在插入操作完成前返回。如果你使用 C API，同样原因，mysql_info()函数不返回任何有意义的东西。
- 当行被插入到表中时，更新日志有处理器线程更新。在多行插入的情况下，当第一行被插入时，更新日志被更新。
- 在每写入 delayed_insert_limit 行后，处理器检查是否任何 SELECT 语句仍然是未完成，如果这样，在继续之前允许执行这些语句。
- 当处理器在它的队列中没有更多行时，表被解锁。如果在 delayed_insert_timeout 秒内没有收到新的 INSERT DELAYED 命令，处理器终止。
- 如果已经有多于 delayed_queue_size 行在一个特定的处理器队列中未解决，线程等待直到队列有空间。这有助于保证 mysqld 服务器对延迟的内存队列不使用所有内存。
- 处理器线程将在 Command 列的 MySQL 进程表中显示 delayed_insert。如果你执行一个 FLUSH TABLES 命令或以 KILL thread_id 杀死它，它将被杀死，然而，它在退出前首先将所有排队的行存进表中。在这期间，这次它将不从其他线程接受任何新的 INSERT 命令。如果你在它之后执行一个 INSERT DELAYED，将创建一个新的处理器线程。
- 注意，上述意味着，如果有一个 INSERT DELAYED 处理器已经运行，INSERT DELAYED 命令有比正常 INSERT 更高的优先级！其他更新命令将必须等到 INSERT DELAY 排队变空、杀死处理器线程(用 KILL thread_id)或执行 FLUSH TABLES。
- 下列状态变量提供了关于 INSERT DELAYED 命令的信息：

  Delayed_insert_threads　处理器线程数量

  Delayed_writes　用 INSERT DELAYED 被写入的行的数量

  Not_flushed_delayed_rows　等待被写入的行数字

  你能通过发出一个 SHOW STATUS 语句或通过执行一个 mysqladmin extended-status 命令察看这些变量。

注意如果表不在使用，INSERT DELAYED 比一个正常的 INSERT 慢。对服务器也有额外开销来处理你对它使用 INSERT DELAYED 的每个表的一个单独线程。这意味着，你应该只在你确实肯定需要它的时候才使用 INSERT DELAYED！

1.15 REPLACE 句法

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name [(col_name,...)]
    VALUES (expression,...)
或  REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
```

或 REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name
SET col_name=expression, col_name=expression,...

REPLACE 功能与 INSERT 完全一样，除了如果在表中的一个老记录具有在一个唯一索引上的新记录有相同的值，在新记录被插入之前，老记录被删除。见 1.14 INSERT 句法。

1.16 LOAD DATA INFILE 句法

LOAD DATA [LOW_PRIORITY] [LOCAL] INFILE 'file_name.txt' [REPLACE | IGNORE]
INTO TABLE tbl_name
[FIELDS
[TERMINATED BY '\t']
[OPTIONALLY] ENCLOSED BY '']
[ESCAPED BY '\\']]
[LINES TERMINATED BY '\n']
[IGNORE number LINES]
[(col_name,...)]

LOAD DATA INFILE 语句从一个文本文件中以很高的速度读入一个表中。如果指定 LOCAL 关键词，从客户主机读文件。如果 LOCAL 没指定，文件必须位于服务器上。(LOCAL 在 MySQL3.22.6 或以后版本中可用。)

为了安全原因，当读取位于服务器上的文本文件时，文件必须处于数据库目录或可被所有人读取。另外，为了对服务器上文件使用 LOAD DATA INFILE，在服务器主机上你必须有 file 的权限。见 1.1.2 用户的权限。

如果你指定关键词 LOW_PRIORITY，LOAD DATA 语句的执行被推迟到没有其他客户读取表后。

使用 LOCAL 将比让服务器直接存取文件慢些，因为文件的内容必须从客户主机传送到服务器主机。在另一方面，你不需要 file 权限装载本地文件。

你也可以使用 mysqlimport 实用程序装载数据文件；它由发送一个 LOAD DATA INFILE 命令到服务器来运作。 --local 选项使得 mysqlimport 从客户主机上读取数据。如果客户和服务器支持压缩协议，你能指定--compress 在较慢的网络上获得更好的性能。

当在服务器主机上寻找文件时，服务器使用下列规则：

- 如果给出一个绝对路径名，服务器使用该路径名。
- 如果给出一个有一个或多个前置部件的相对路径名，服务器相对服务器的数据目录搜索文件。
- 如果给出一个没有前置部件的一个文件名，服务器在当前数据库的数据库目录寻找文件。

注意这些规则意味着一个像“./myfile.txt”给出的文件是从服务器的数据目录读取，而作为“myfile.txt”给出的一个文件是从当前数据库的数据库目录下读取。也要注意，对于

下列哪些语句，对 db1 文件从数据库目录读取，而不是 db2：

```
mysql> USE db1;
mysql> LOAD DATA INFILE "./data.txt" INTO TABLE db2.my_table;
```

REPLACE 和 IGNORE 关键词控制对现有的唯一键记录的重复的处理。如果你指定 REPLACE，新行将代替有相同的唯一键值的现有行。如果你指定 IGNORE，跳过有唯一键的现有行的重复行的输入。如果你不指定任何一个选项，当找到重复键键时，出现一个错误，并且文本文件的余下部分被忽略时。

如果你使用 LOCAL 关键词从一个本地文件装载数据，服务器没有办法在操作的当中停止文件的传输，因此缺省的行为好像 IGNORE 被指定一样。

LOAD DATA INFILE 是 SELECT ... INTO OUTFILE 的逆操作，见 1.12 SELECT 句法。为了将一个数据库的数据写入一个文件，使用 SELECT ... INTO OUTFILE，为了将文件读回数据库，使用 LOAD DATA INFILE。两个命令的 FIELDS 和 LINES 子句的语法是相同的。两个子句是可选的，但是如果指定两个，FIELDS 必须在 LINES 之前。

如果你指定一个 FIELDS 子句，它的每一个子句(TERMINATED BY, [OPTIONALLY] ENCLOSED BY 和 ESCAPED BY)也是可选的，除了你必须至少指定他们之一。

如果你不指定一个 FIELDS 子句，缺省值与如果你这样写的相同：

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
```

如果你不指定一个 LINES 子句，缺省值与如果你这样写的相同：

```
LINES TERMINATED BY '\n'
```

换句话说，缺省值导致读取输入时，LOAD DATA INFILE 表现如下：

- 在换行符处寻找行边界
- 在定位符处将行分进字段
- 不要期望字段由任何引号字符封装
- 将由“\”开头的定位符、换行符或“\”解释是字段值的部分字面字符

相反，缺省值导致在写入输出时，SELECT ... INTO OUTFILE 表现如下：

- 在字段之间写定位符
- 不用任何引号字符封装字段
- 使用“\”转义出现在字段中的定位符、换行符或“\”字符
- 在行尾处写换行符

注意，为了写入 FIELDS ESCAPED BY '\\'，对作为一条单个的反斜线被读取的值，你必须指定 2 条反斜线值。

IGNORE number LINES 选项可被用来忽略在文件开始的一个列名字的头：

```
mysql> LOAD DATA INFILE "/tmp/file_name" into table test IGNORE 1 LINES;
```

当你与 LOAD DATA INFILE 一起使用 SELECT ... INTO OUTFILE 将一个数据库的数据写进一个文件并且随后马上将文件读回数据库时，两个命令的字段和处理选项必须匹配，否则，LOAD DATA INFILE 将不能正确解释文件的内容。假定你使用 SELECT ... INTO OUTFILE 将由逗号分隔的字段写入一个文件：

```
mysql> SELECT * FROM table1 INTO OUTFILE 'data.txt'
```

```
        FIELDS TERMINATED BY ','
        FROM ...
```

为了将由逗号分隔的文件读回来，正确的语句将是：

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE table2
        FIELDS TERMINATED BY ',';
```

相反，如果你试图用下面显示的语句读取文件，它不会工作，因为它命令 LOAD DATA INFILE 在字段之间寻找定位符：

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE table2
        FIELDS TERMINATED BY '\t';
```

可能的结果是每个输入行将被解释为单个的字段。

LOAD DATA INFILE 能被用来读取从外部来源获得的文件。例如，以 dBASE 格式的文件将有由逗号分隔并用双引号包围的字段。如果文件中的行由换行符终止，下面显示的命令说明你将用来装载文件的字段和行处理选项：

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE tbl_name
        FIELDS TERMINATED BY ',' ENCLOSED BY '"'
        LINES TERMINATED BY '\n';
```

任何字段或行处理选项可以指定一个空字符串('')。如果不是空，FIELDS [OPTIONALLY] ENCLOSED BY 和 FIELDS ESCAPED BY 值必须是一个单个字符。FIELDS TERMINATED BY 和 LINES TERMINATED BY 值可以是超过一个字符。例如，写入由回车换行符对（CR+LF）终止的行，或读取包含这样行的一个文件，指定一个 LINES TERMINATED BY '\r\n'子句。

FIELDS [OPTIONALLY] ENCLOSED BY 控制字段的包围字符。对于输出(SELECT ... INTO OUTFILE)，如果你省略 OPTIONALLY，所有的字段由 ENCLOSED BY 字符包围。对于这样的输出的一个例子(使用一个逗号作为字段分隔符)显示在下面：

```
"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"
```

如果你指定 OPTIONALLY，ENCLOSED BY 字符仅被用于包围 CHAR 和 VARCHAR 字段：

```
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，一个字段值中的 ENCLOSED BY 字符的出现通过用 ESCAPED BY 字符作为其前缀来转义。也要注意，如果你指定一个空 ESCAPED BY 值，可能产生不能被 LOAD DATA INFILE 正确读出的输出。例如，如果转义字符为空，上面显示的输出显示如下。注意到在第四行的第二个字段包含跟随引号的一个逗号，它(错误地)好象要终止字段：

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，ENCLOSED BY 字符如果存在，它从字段值的尾部被剥去。（不管是否指定 OPTIONALLY 都是这样；OPTIONALLY 对于输入解释不起作用)由 ENCLOSED BY 字符领先的 ESCAPED BY 字符出现被解释为当前字段值的一部分。另外，出现在字段中重复的 ENCLOSED BY 被解释为单个 ENCLOSED BY 字符，如果字段本身以该字符开始。例如，如果 ENCLOSED BY ""被指定，引号如下处理：

"The ""BIG"" boss" -> The "BIG" boss

The "BIG" boss -> The "BIG" boss

The ""BIG"" boss -> The ""BIG"" boss

FIELDS ESCAPED BY 控制如何写入或读出特殊字符。如果 FIELDS ESCAPED BY 字符不是空的，它被用于前缀在输出上的下列字符：

- 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（实际上将后续转义字符写成 ASCII'0'，而不是一个零值字节）

如果 FIELDS ESCAPED BY 字符是空的，没有字符被转义。指定一个空转义字符可能不是一个好主意，特别是如果在你数据中的字段值包含刚才给出的表中的任何字符。

对于输入，如果 FIELDS ESCAPED BY 字符不是空的，该字符的出现被剥去并且后续字符在字面上作为字段值的一个部分。例外是一个转义的“0”或“N”（即，\0 或\N，如果转义字符是“\”)。这些序列被解释为 ASCII 0（一个零值字节）和 NULL。见下面关于 NULL 处理的规则。

对于更多关于“\”- 转义句法的信息，见 1.1 文字：怎样写字符串和数字。

在某些情况下，字段和行处理选项相互作用：

- 如果 LINES TERMINATED BY 是一个空字符串并且 FIELDS TERMINATED BY 是非空的，行也用 FIELDS TERMINATED BY 终止。
- 如果 FIELDS TERMINATED BY 和 FIELDS ENCLOSED BY 值都是空的(")，一个固定行(非限定的)格式被使用。用固定行格式，在字段之间不使用分隔符。相反，列值只用列的“显示”宽度被写入和读出。例如，如果列被声明为 INT(7)，列的值使用 7 个字符的字段被写入。对于输入，列值通过读取 7 个字符获得。固定行格式也影响 NULL 值的处理；见下面。注意如果你正在使用一个多字节字符集，固定长度格式将不工作。

NULL 值的处理有多种，取决于你使用的 FIELDS 和 LINES 选项：

- 对于缺省 FIELDS 和 LINES 值，对输出，NULL 被写成\N，对输入，\N 被作为 NULL 读入(假定 ESCAPED BY 字符是“\”)。
- 如果 FIELDS ENCLOSED BY 不是空的，包含以文字词的 NULL 作为它的值的字

段作为一个 NULL 值被读入(这不同于包围在 FIELDS ENCLOSED BY 字符中的字 NULL，它作为字符串'NULL'读入)。

- 如果 FIELDS ESCAPED BY 是空的，NULL 作为字 NULL 被写入。
- 用固定行格式(它发生在 FIELDS TERMINATED BY 和 FIELDS ENCLOSED BY 都是空的时候)，NULL 作为一个空字符串被写入。注意，在写入文件时，这导致 NULL 和空字符串在表中不能区分，因为他们都作为空字符串被写入。如果在读回文件时需要能区分这两者，你应该不使用固定行格式。

一些不被 LOAD DATA INFILE 支持的情况：

- 固定长度的行(FIELDS TERMINATED BY 和 FIELDS ENCLOSED BY 都为空)和 BLOB 或 TEXT 列。
- 如果你指定一个分隔符与另一个相同，或是另一个的前缀，LOAD DATA INFILE 不能正确地解释输入。例如，下列 FIELDS 子句将导致问题：

  FIELDS TERMINATED BY '"' ENCLOSED BY '"'
- 如果 FIELDS ESCAPED BY 是空的，一个包含跟随 FIELDS TERMINATED BY 值之后的 FIELDS ENCLOSED BY 或 LINES TERMINATED BY 的字段值将使得 LOAD DATA INFILE 过早地终止读取一个字段或行。这是因为 LOAD DATA INFILE 不能正确地决定字段或行值在哪儿结束。

下列例子装载所有 persondata 表的行：

```
mysql> LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

没有指定字段表，所以 LOAD DATA INFILE 期望输入行对每个表列包含一个字段。使用缺省 FIELDS 和 LINES 值。

如果你希望仅仅装载一张表的某些列，指定一个字段表：

```
mysql> LOAD DATA INFILE 'persondata.txt'
            INTO TABLE persondata (col1,col2,...);
```

如果在输入文件中的字段顺序不同于表中列的顺序，你也必须指定一个字段表。否则，MySQL 不能知道如何匹配输入字段和表中的列。

如果一个行有很少的字段，对于不存在输入字段的列被设置为缺省值。缺省值赋值在 1.7 CREATE TABLE 句法中描述。

如果字段值缺省，空字段值有不同的解释：

- 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，列被设置为该类型的适当“零”值。见 1.3.6 日期和时间类型。

如果列有一个 NULL，或(只对第一个 TIMESTAMP 列)在指定一个字段表时，如果 TIMESTAMP 列从字段表省掉，TIMESTAMP 列只被设置为当前的日期和时间。

如果输入行有太多的字段，多余的字段被忽略并且警告数字加 1。

LOAD DATA INFILE 认为所有的输入是字符串，因此你不能像你能用 INSERT 语句的 ENUM 或 SET 列的方式使用数字值。所有的 ENUM 和 SET 值必须作为字符串被指定！

如果你正在使用 C API，当 LOAD DATA INFILE 查询完成时，你可通过调用 API 函数 mysql_info()得到有关查询的信息。信息字符串的格式显示在下面：

Records: 1 Deleted: 0 Skipped: 0 Warnings: 0

当值通过 INSERT 语句插入时，在某些情况下出现警告(见 1.14 INSERT 句法)，除了在输入行中有太少或太多的字段时，LOAD DATA INFILE 也产生警告。警告没被存储在任何地方；警告数字仅能用于表明一切是否顺利。如果你得到警告并且想要确切知道你为什么得到他们，一个方法是使用 SELECT ... INTO OUTFILE 到另外一个文件并且把它与你的原版输入文件比较。

对于有关 INSERT 相对 LOAD DATA INFILE 的效率和加快 LOAD DATA INFILE 的更多信息。

1.17 UPDATE 句法

```
UPDATE [LOW_PRIORITY] tbl_name SET col_name1=expr1,col_name2=expr2,...
    [WHERE where_definition] [LIMIT #]
```

UPDATE 用新值更新现存表中行的列，SET 子句指出哪个列要修改和他们应该被给定的值，WHERE 子句，如果给出，指定哪个行应该被更新，否则所有行被更新。

如果你指定关键词 LOW_PRIORITY，执行 UPDATE 被推迟到没有其他客户正在读取表时。

如果你从一个表达式的 tbl_name 存取列，UPDATE 使用列的当前值。例如，下列语句设置 age 为它的当前值加 1：

```
mysql> UPDATE persondata SET age=age+1;
```

UPDATE 赋值是从左到右计算。例如，下列语句两倍 age 列，然后加 1：

```
mysql> UPDATE persondata SET age=age*2, age=age+1;
```

如果你设置列为其它当前有的值，MySQL 注意到这点并且不更新它。

UPDATE 返回实际上被改变的行的数量。在 MySQL 3.22 或以后版本中，C API 函数 mysql_info()返回被匹配并且更新的行数和在 UPDATE 期间发生警告的数量。

在 MySQL3.23 中，你可使用 LIMIT #来保证只有一个给定数量的行被改变。

1.18 USE 句法

```
USE db_name
```

USE db_name 语句告诉 MySQL 使用 db_name 数据库作为随后的查询的缺省数据库。数据库保持到会话结束，或发出另外一个 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
```

利用 USE 语句使得一个特定的数据库称为当前数据库并不阻止你访问在另外的数据库中的表。下面的例子访问 db1 数据库中的 author 表和 db2 数据库中的 editor 表：

```
mysql> USE db1;
mysql> SELECT author_name,editor_name FROM author,db2.editor
            WHERE author.editor_id = db2.editor.editor_id;
```

USE 语句提供了 Sybase 的兼容性。

1.19 FLUSH 句法(清除缓存）

FLUSH flush_option [,flush_option]

如果你想要清除一些 MySQL 使用内部缓存，你应该使用 FLUSH 命令。为了执行 FLUSH，你必须有 reload 权限。

flush_option 可以是下列任何东西：

- HOSTS　清空主机缓存表。如果你的某些主机改变 IP 数字，或如果你得到错误消息 Host ... is blocked，你应该清空主机表。当在连接 MySQL 服务器时，对一台给定的主机有多于 max_connect_errors 个错误连续不断地发生，MySQL 认定某些东西错了并且阻止主机进一步的连接请求。清空主机表允许主机再尝试连接。见 18.2.3 Host '...' is blocked 错误)。你可用-O max_connection_errors=999999999 启动 mysqld 来避免这条错误消息。
- LOGS　关闭并且再打开标准和更新记录文件。如果你指定了一个没有扩展名的更新记录文件，新的更新记录文件的扩展数字将相对先前的文件加 1。
- PRIVILEGES　从 mysql 数据库授权表中重新装载权限。
- TABLES　关闭所有打开的表。
- STATUS　重置大多数状态变量到 0。

你也可以用 mysqladmin 实用程序，使用 flush-hosts, flush-logs, reload 或 flush-tables 命令来访问上述的每一个命令。

1.20 KILL 句法

KILL thread_id

每个对 mysqld 的连接以一个单独的线程运行。你可以用看 SHOW PROCESSLIST 命令察看哪个线程正在运行，并且用 KILL thread_id 命令杀死一个线程。

如果你有 process 权限，你能看到并且杀死所有线程。否则，你只能看到并且杀死你自己的线程。

你也可以使用 mysqladmin processlist 和 mysqladmin kill 命令检查并杀死线程。

1.21 SHOW 句法 (得到表，列等的信息）

```
SHOW DATABASES [LIKE wild]
or SHOW TABLES [FROM db_name] [LIKE wild]
or SHOW COLUMNS FROM tbl_name [FROM db_name] [LIKE wild]
or SHOW INDEX FROM tbl_name [FROM db_name]
or SHOW STATUS
or SHOW VARIABLES [LIKE wild]
or SHOW [FULL] PROCESSLIST
```

or SHOW TABLE STATUS [FROM db_name] [LIKE wild]

or SHOW GRANTS FOR user

SHOW 提供关于数据库、表、列或服务器的信息。如果使用 LIKE wild 部分，wild 字符串可以是一个使用 SQL 的“%”和“_”通配符的字符串。

你能使用 db_name.tbl_name 作为 tbl_name FROM db_name 句法的另一种选择。这两个语句是相等的：

mysql> SHOW INDEX FROM mytable FROM mydb;

mysql> SHOW INDEX FROM mydb.mytable;

SHOW DATABASES 列出在 MySQL 服务器主机上的数据库。你也可以用 mysqlshow 命令得到这张表。

SHOW TABLES 列出在一个给定的数据库中的表。你也可以用 mysqlshow db_name 命令得到这张表。

注意：如果一个用户没有一个表的任何权限，表将不在 SHOW TABLES 或 mysqlshow db_name 中的输出中显示。

SHOW COLUMNS 列出在一个给定表中的列。如果列类型不同于你期望的是基于 CREATE TABLE 语句的那样，注意，MySQL 有时改变列类型。见 1.1.1 隐含的列说明变化。

DESCRIBE 语句提供了类似 SHOW COLUMNS 的信息。见 1.23 DESCRIBE 句法 (得到列的信息)。

SHOW TABLE STATUS（在版本 3.23 引入）运行类似 SHOW STATUS，但是提供每个表的更多信息。你也可以使用 mysqlshow --status db_name 命令得到这张表。下面的列被返回：

- Name 表名
- Type 表的类型 (ISAM，MyISAM 或 HEAP)
- Row_format 行存储格式 (固定, 动态, 或压缩)
- Rows 行数量
- Avg_row_length 平均行长度
- Data_length 数据文件的长度
- Max_data_length 数据文件的最大长度
- Index_length 索引文件的长度
- Data_free 已分配但未使用了字节数
- Auto_increment 下一个 autoincrement(自动加 1)值
- Create_time 表被创造的时间
- Update_time 数据文件最后更新的时间
- Check_time 最后对表运行一个检查的时间
- Create_options 与 CREATE TABLE 一起使用的额外选项
- Comment 当创造表时，使用的注释 (或为什么 MySQL 不能存取表信息的一些信息)。

SHOW FIELDS 是 SHOW COLUMNS 一个同义词，SHOW KEYS 是 SHOW INDEX 一个同义词。你也可以用 mysqlshow db_name tbl_name 或 mysqlshow -k db_name tbl_name 列出一张表的列或索引。

SHOW INDEX 以非常相似于 ODBC 的 SQLStatistics 调用的格式返回索引信息。下面的列被返回：

- Table 表名
- Non_unique 0，如果索引不能包含重复。
- Key_name 索引名
- Seq_in_index 索引中的列顺序号，从 1 开始。
- Column_name 列名。
- Collation 列怎样在索引中被排序。在 MySQL 中，这可以有值 A（升序）或 NULL（不排序）。
- Cardinality 索引中唯一值的数量。这可通过运行 isamchk -a 更改.
- Sub_part 如果列只是部分被索引，索引字符的数量。NULL，如果整个键被索引。

SHOW STATUS 提供服务器的状态信息(象 mysqladmin extended-status 一样)。输出类似于下面的显示，尽管格式和数字可以有点不同：

```
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Aborted_clients       | 0     |
| Aborted_connects      | 0     |
| Connections           | 17    |
| Created_tmp_tables    | 0     |
| Delayed_insert_threads| 0     |
| Delayed_writes        | 0     |
| Delayed_errors        | 0     |
| Flush_commands        | 2     |
| Handler_delete        | 2     |
| Handler_read_first    | 0     |
| Handler_read_key      | 1     |
| Handler_read_next     | 0     |
| Handler_read_rnd      | 35    |
| Handler_update        | 0     |
| Handler_write         | 2     |
| Key_blocks_used       | 0     |
| Key_read_requests     | 0     |
| Key_reads             | 0     |
```

```
| Key_write_requests       | 0      |
| Key_writes               | 0      |
| Max_used_connections     | 1      |
| Not_flushed_key_blocks   | 0      |
| Not_flushed_delayed_rows | 0      |
| Open_tables              | 1      |
| Open_files               | 2      |
| Open_streams             | 0      |
| Opened_tables            | 11     |
| Questions                | 14     |
| Slow_queries             | 0      |
| Threads_connected        | 1      |
| Threads_running          | 1      |
| Uptime                   | 149111 |
+--------------------------+--------+
```

上面列出的状态变量有下列含义：

Aborted_clients　由于客户没有正确关闭连接已经死掉，已经放弃的连接数量。

Aborted_connects　尝试已经失败的 MySQL 服务器的连接的次数。

Connections　试图连接 MySQL 服务器的次数。

Created_tmp_tables　当执行语句时，已经被创造了的隐含临时表的数量。

Delayed_insert_threads　正在使用的延迟插入处理器线程的数量。

Delayed_writes　用 INSERT DELAYED 写入的行数。

Delayed_errors　用 INSERT DELAYED 写入的发生某些错误(可能重复键值)的行数。

Flush_commands　执行 FLUSH 命令的次数。

Handler_delete　请求从一张表中删除行的次数。

Handler_read_first　请求读入表中第一行的次数。

Handler_read_key　请求数字基于键读行。

Handler_read_next　请求读入基于一个键的一行的次数。

Handler_read_rnd　请求读入基于一个固定位置的一行的次数。

Handler_update　请求更新表中一行的次数。

Handler_write　请求向表中插入一行的次数。

Key_blocks_used　用于关键字缓存的块的数量。

Key_read_requests　请求从缓存读入一个键值的次数。

Key_reads　从磁盘物理读入一个键值的次数。

Key_write_requests　请求将一个关键字块写入缓存次数。

Key_writes　将一个键值块物理写入磁盘的次数。

Max_used_connections　同时使用的连接的最大数目。

Not_flushed_key_blocks　在键缓存中已经改变但是还没被清空到磁盘上的键块。
Not_flushed_delayed_rows　在 INSERT DELAY 队列中等待写入的行的数量。
Open_tables　打开表的数量。
Open_files　打开文件的数量。
Open_streams　打开流的数量(主要用于日志记载）
Opened_tables　已经打开的表的数量。
Questions　发往服务器的查询的数量。
Slow_queries　要花超过 long_query_time 时间的查询数量。
Threads_connected　当前打开的连接的数量。
Threads_running　不在睡眠的线程数量。
Uptime　服务器工作了多少秒。

关于上面的一些注释：

- 如果 Opened_tables 太大，那么你的 table_cache 变量可能太小。
- 如果 key_reads 太大，那么你的 key_cache 可能太小。缓存命中率可以用 key_reads/key_read_requests 计算。
- 如果 Handler_read_rnd 太大，那么你很可能有大量的查询需要 MySQL 扫描整个表或你有没正确使用键值的联结(join)。

SHOW VARIABLES 显示出一些 MySQL 系统变量的值，你也能使用 mysqladmin variables 命令得到这个信息。如果缺省值不合适，你能在 mysqld 启动时使用命令行选项来设置这些变量的大多数。输出类似于下面的显示，尽管格式和数字可以有点不同：

```
+--------------------+----------------------+
| Variable_name      | Value                |
+--------------------+----------------------+
| back_log           | 5                    |
| connect_timeout    | 5                    |
| basedir            | /my/monty/           |
| datadir            | /my/monty/data/      |
| delayed_insert_limit   | 100              |
| delayed_insert_timeout | 300              |
| delayed_queue_size     | 1000             |
| join_buffer_size       | 131072           |
| flush_time             | 0                |
| interactive_timeout    | 28800            |
| key_buffer_size        | 1048540          |
| language               | /my/monty/share/english/ |
| log                    | OFF              |
```

```
| log_update              | OFF                  |
| long_query_time         | 10                   |
| low_priority_updates    | OFF                  |
| max_allowed_packet      | 1048576              |
| max_connections         | 100                  |
| max_connect_errors      | 10                   |
| max_delayed_threads     | 20                   |
| max_heap_table_size     | 16777216             |
| max_join_size           | 4294967295           |
| max_sort_length         | 1024                 |
| max_tmp_tables          | 32                   |
| net_buffer_length       | 16384                |
| port                    | 3306                 |
| protocol-version        | 10                   |
| record_buffer           | 131072               |
| skip_locking            | ON                   |
| socket                  | /tmp/mysql.sock      |
| sort_buffer             | 2097116              |
| table_cache             | 64                   |
| thread_stack            | 131072               |
| tmp_table_size          | 1048576              |
| tmpdir                  | /machine/tmp/        |
| version                 | 3.23.0-alpha-debug   |
| wait_timeout            | 28800                |
+-------------------------+----------------------+
```

SHOW PROCESSLIST 显示哪个线程正在运行，你也能使用 mysqladmin processlist 命令得到这个信息。如果你有 process 权限， 你能看见所有的线程，否则，你仅能看见你自己的线程。见 1.20 KILL 句法。如果你不使用 FULL 选项，那么每个查询只有头 100 字符被显示出来。

SHOW GRANTS FOR user 列出对一个用户必须发出以重复授权的授权命令。

```
mysql> SHOW GRANTS FOR root@localhost;
+-----------------------------------------------------------------+
| Grants for root@localhost                                       |
+-----------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root''localhost' WITH GRANT OPTION  |
+-----------------------------------------------------------------+
```

1.22 EXPLAIN 句法(得到关于 SELECT 的信息)

EXPLAIN tbl_name

or EXPLAIN SELECT select_options

EXPLAIN tbl_name 是 DESCRIBE tbl_name 或 SHOW COLUMNS FROM tbl_name 的一个同义词。

当你在一条 SELECT 语句前放上关键词 EXPLAIN，MySQL 解释它将如何处理 SELECT，提供有关表如何联结和以什么次序联结的信息。

借助于 EXPLAIN，你可以知道你什么时候必须为表加入索引以得到一个使用索引找到记录的更快的 SELECT。你也能知道优化器是否以一个最佳次序联结表。为了强制优化器对一个 SELECT 语句使用一个特定联结次序，增加一个 STRAIGHT_JOIN 子句。

对于非简单的联结，EXPLAIN 为用于 SELECT 语句中的每个表返回一行信息。表以他们将被读入的顺序被列出。MySQL 用一边扫描多次联结的方式解决所有联结，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。当所有的表被处理完，它输出选择的列并且回溯表列表直到找到一个表有更多的匹配行，从该表读入下一行并继续处理下一个表。

从 EXPLAIN 的输出包括下面列：

table

输出的行所引用的表。

type

联结类型。各种类型的信息在下面给出。

possible_keys

possible_keys 列指出 MySQL 能使用哪个索引在该表中找到行。注意，该列完全独立于表的次序。这意味着在 possible_keys 中的某些键实际上不能以生成的表次序使用。如果该列是空的，没有相关的索引。在这种情况下，你也许能通过检验 WHERE 子句看是否它引用某些列或列不是适合索引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在用 EXPLAIN 检查查询。见 1.8 ALTER TABLE 句法。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。如果没有索引被选择，键是 NULL。

key_len

key_len 列显示 MySQL 决定使用的键长度。如果键是 NULL，长度是 NULL。注意这告诉我们 MySQL 将实际使用一个多部键值的几个部分。

ref

ref 列显示哪个列或常数与 key 一起用于从表中选择行。

rows

rows 列显示 MySQL 相信它必须检验以执行查询的行数。

Extra

如果 Extra 列包括文字 Only index，这意味着信息只用索引树中的信息检索出的。通常，这比扫描整个表要快。如果 Extra 列包括文字 where used，它意味着一个 WHERE 子句将

被用来限制哪些行与下一个表匹配或发向客户。

不同的联结类型列在下面，以最好到最差类型的次序：

system

表仅有一行(=系统表)。这是 const 联结类型的一个特例。

const

表有最多一个匹配行，它将在查询开始时被读取。因为仅有一行，在这行的列值可被剩下的优化器认为是常数。 const 表很快，因为它们只读取一次!

eq_ref

对于每个来自于先前的表的行组合，从该表中读取一行。这可能是最好的联结类型，除了 const 类型。它用在一个索引的所有部分被联结使用并且索引是 UNIQUE 或 PRIMARY KEY。

ref

对于每个来自于先前的表的行组合，所有有匹配索引值的行将从这张表中读取。如果联结只使用键的最左面前缀，或如果键不是 UNIQUE 或 PRIMARY KEY（换句话说，如果联结不能基于键值选择单个行的话)，使用 ref。如果被使用的键仅仅匹配一些行，该联结类型是不错的。

range

只有在一个给定范围的行将被检索，使用一个索引选择行。ref 列显示哪个索引被使用。

index

这与 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。

ALL

对于每个来自于先前的表的行组合，将要做一个完整的表扫描。如果表格是第一个没标记 const 的表，这通常不好，并且通常在所有的其他情况下很差。你通常可以通过增加更多的索引来避免 ALL，使得行能从早先的表中基于常数值或列值被检索出。

通过相乘 EXPLAIN 输出的 rows 行的所有值，你能得到一个关于一个联结要多好的提示。这应该粗略地告诉你 MySQL 必须检验多少行以执行查询。当你使用 max_join_size 变量限制查询时，也用这个数字。见 10.2.3 调节服务器参数。

下列例子显示出一个 JOIN 如何能使用 EXPLAIN 提供的信息逐步被优化。

假定你有显示在下面的 SELECT 语句，你使用 EXPLAIN 检验：

```
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：

1、被比较的列被声明如下：

表附 1-14　被比较的表

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)

2、表有显示在下面的索引：

表附 1-15　被比较表的索引

表	索引
tt	ActualPC
tt	AssignedPC
tt	ClientID
et	EMPLOYID（主键）
do	CUSTNMBR（主键）

3、tt.ActualPC 值不是均匀分布的。

开始，在任何优化被施行前，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,ClientID,ActualPC     NULL  NULL    NULL   3872
     range checked for each record (key map: 35)
```

因为 type 对每张表是 ALL，这个输出显示 MySQL 正在对所有表进行一个完整联结！这将花相当长的时间，因为必须检验每张表的行数的乘积次数！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……

如果列声明不同，这里的一个问题是 MySQL(还)不能高效地在列上使用索引。在本文中，VARCHAR 和 CHAR 是相同的，除非他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)，有一个长度失配。

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,ClientID,ActualPC NULL NULL NULL 3872  where used
do  ALL  PRIMARY  NULL  NULL  NULL  2135
    range checked for each record (key map: 1)
et_1  ALL  PRIMARY  NULL  NULL  NULL  74
    range checked for each record (key map: 1)
et  eq_ref PRIMARY  PRIMARY 15  tt.ActualPC 1
```

这不是完美的，但是是好一些了(rows 值的乘积少了一个 74 一个因子)，这个版本在几秒内执行。

第 2 种改变能消除 tt.AssignedPC = et_1.EMPLOYID 和 tt.ClientID = do.CUSTNMBR 比较的列的长度失配：

```
mysql> ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),
                      MODIFY ClientID   VARCHAR(15);
```

现在 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,ClientID,ActualPC ActualPC 15 et.EMPLOYID 52 where used
et_1  eq_ref PRIMARY  PRIMARY 15  tt.AssignedPC  1
do  eq_ref PRIMARY  PRIMARY 15  tt.ClientID  1
```

这“几乎”象它能得到的一样好。

剩下的问题是，缺省地，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 关于这些：

```
shell> myisamchk --analyze PATH_TO_MYSQL_DATABASE/tt
shell> mysqladmin refresh
```

现在联结是“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,ClientID,ActualPC  NULL  NULL  NULL  3872  where
used
et  eq_ref PRIMARY  PRIMARY 15  tt.ActualPC  1
et_1  eq_ref PRIMARY  PRIMARY 15  tt.AssignedPC  1
```

```
do      eq_ref PRIMARY          PRIMARY 15      tt.ClientID     1
```

注意在从 EXPLAIN 输出的 rows 列是一个来自 MySQL 联结优化器的“教育猜测”；为了优化查询，你应该检查数字是否接近事实。如果不是，你可以通过在你的 SELECT 语句里面使用 STRAIGHT_JOIN 并且试着在在 FROM 子句以不同的次序列出表，可能得到更好的性能。

1.23 DESCRIBE 句法 (得到列的信息）

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

DESCRIBE 提供关于一张表的列的信息。col_name 可以是一个列名字或包含 SQL 的“%”和“_”通配符的一个字符串。

如果列类型不同于你期望的是基于一个 CREATE TABLE 语句，注意 MySQL 有时改变列类型。见 1.1.1 隐含的列说明变化。

这个语句为了与 Oracle 兼容而提供的。

SHOW 语句提供类似的信息。见 1.21 SHOW 句法(得到表，列的信息)。

1.24 LOCK TABLES/UNLOCK TABLES 句法

```
LOCK TABLES tbl_name [AS alias] {READ | [LOW_PRIORITY] WRITE}
    [, tbl_name {READ | [LOW_PRIORITY] WRITE} ...]
...
UNLOCK TABLES
```

LOCK TABLES 为当前线程锁定表。UNLOCK TABLES 释放被当前线程持有的任何锁。当线程发出另外一个 LOCK TABLES 时，或当服务器的连接被关闭时，当前线程锁定的所有表自动被解锁。

如果一个线程获得在一个表上的一个 READ 锁，该线程(和所有其他线程)只能从表中读。如果一个线程获得一个表上的一个 WRITE 锁，那么只有持锁的线程 READ 或 WRITE 表，其他线程被阻止。

每个线程等待(没有超时)直到它获得它请求的所有锁。

WRITE 锁通常比 READ 锁有更高的优先级，以确保更改尽快被处理。这意味着，如果一个线程获得 READ 锁，并且然后另外一个线程请求一个 WRITE 锁，随后的 READ 锁请求将等待直到 WRITE 线程得到了锁并且释放了它。当线程正在等待 WRITE 锁时，你可以使用 LOW_PRIORITY WRITE 允许其他线程获得 READ 锁。如果你肯定终于有个时刻没有线程将有一个 READ 锁，你应该只使用 LOW_PRIORITY WRITE。

当你使用 LOCK TABLES 时，你必须锁定你将使用的所有表！如果你正在一个查询中多次使用一张表(用别名)，你必须对每个别名得到一把锁！这条政策保证表锁定不会死锁。

注意你应该不锁定任何你正在用 INSERT DELAYED 使用的表，这是因为在这种情况下，INSERT 被一个不同的线程执行。

通常，你不必锁定表，因为所有单个 UPDATE 语句是原语；没有其他线程能防碍任何其它正在执行 SQL 语句的线程。当你想锁定表，有一些情况：

- 如果你将在一堆表上运行许多操作，锁定你将使用的表是较快的。当然缺点是，没有其他线程能更新一个 READ 锁定的表并且没有其他线程能读一个 WRITE-锁定的表。
- MySQL 不支持事务环境，所以如果你想要保证在一个 SELECT 和一个 UPDATE 之间没有其他线程到来，你必须使用 LOCK TABLES。下面显示的例子要求 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;
```

没有 LOCK TABLES，另外一个线程可能有一个机会在执行 SELECT 和 UPDATE 语句之间往 trans 表中插入一个新行。

通过使用渐增更改(UPDATE customer SET value=value+new_value) 或 LAST_INSERT_ID()函数，在很多情况下你能使用 LOCK TABLES 来避免。

你也可以使用用户级锁定函数 GET_LOCK()和 RELEASE_LOCK()解决一些情况，这些锁保存在服务器的一张哈希表中并且用 pthread_mutex_lock()和 pthread_mutex_unlock()实现以获得高速度。见 1.4.12 其他函数。

1.25 SET OPTION 句法

SET [OPTION] SQL_VALUE_OPTION= value, ...

SET OPTION 设置影响服务器或你的客户操作的各种选项。你设置的任何选择保持有效直到当前会话结束，或直到你设置选项为不同的值。

CHARACTER SET character_set_name | DEFAULT

这用给定的映射表从/到客户映射所有字符串。对 character_set_name 当前唯一的选项是 cp1251_koi8，但是你能容易通过编辑在 MySQL 源代码分发的“sql/convert.cc”文件增加新的映射。缺省映射能用 character_set_name 的 DEFAULT 值恢复。注意设置 CHARACTER SET 选项的语法不同于设置其他选项目的语法。

PASSWORD = PASSWORD('some password')

设置当前用户的口令。任何非匿名的用户能改变他自己的口令！

PASSWORD FOR user = PASSWORD('some password')

设置当前服务器主机上的一个特定用户的口令。只有具备存取 mysql 数据库的用户可以这样做。用户应该以 user@hostname 格式给出，这里 user 和 hostname 完全与他们列在 mysql.user 表条目的 User 和 Host 列一样。例如，如果你有一个条目其 User 和 Host 字段是'bob'和'%.loc.gov'，你将写成：

```
mysql> SET PASSWORD FOR bob@"%.loc.gov" = PASSWORD("newpass");
```

或

```
mysql> UPDATE mysql.user SET password=PASSWORD("newpass") where user="bob'
```

and host="%.loc.gov";

SQL_AUTO_IS_NULL = 0 | 1

如果设置为 1（缺省 ），那么对于一个具有一个自动加 1 的行的表，用下列构件能找出最后插入的行：WHERE auto_increment_column IS NULL。这被一些 ODBC 程序入 Access 使用。

SQL_BIG_TABLES = 0 | 1

如果设置为 1，所有临时表存在在磁盘上而非内存中。这将更慢一些，但是对需要大的临时表的大 SELECT 操作，你将不会得到 The table tbl_name is full 的错误。对于一个新连接的缺省值是 0（即，使用内存中的临时表)。

SQL_BIG_SELECTS = 0 | 1

如果设置为 0，如果一个 SELECT 尝试可能花很长的时间，MySQL 将放弃。这在一个不妥当的 WHERE 语句发出时是有用的。一个大的查询被定义为一个将可能必须检验多于 max_join_size 行的 SELECT。对一个新连接的缺省值是1(它将允许所有 SELECT 语句)。

SQL_LOW_PRIORITY_UPDATES = 0 | 1

如果设置为 1，所有 INSERT、UPDATE、DELETE 和 LOCK TABLE WRITE 语句等待，直到在受影响的表上没有未解决的 SELECT 或 LOCK TABLE READ。

SQL_SELECT_LIMIT = value | DEFAULT

从 SELECT 语句返回的记录的最大数量。如果一个 SELECT 有一个 LIMIT 子句，LIMIT 优先与 SQL_SELECT_LIMIT 值。对一个新连接的缺省值是“无限”的。如果你改变了限制，缺省值能用 SQL_SELECT_LIMIT 的一个 DEFAULT 值恢复。

SQL_LOG_OFF = 0 | 1

如果设置为 1，如果客户有 process 权限，对该客户没有日志记载到标准的日志文件中。这不影响更新日志记录!

SQL_LOG_UPDATE = 0 | 1

如果设置为 0, 如果客户有 process 权限，对该客户没有日志记载到更新日志中。这不影响标准日志文件!

TIMESTAMP = timestamp_value | DEFAULT

为该客户设置时间。如果你使用更新日志恢复行，这被用来得到原来的时间标记。

LAST_INSERT_ID = #

设置从 LAST_INSERT_ID() 返回的值。当你在更新一个表的命令中使用 LAST_INSERT_ID()时，它存储在更新日志中。

INSERT_ID = #

设置当插入一个 AUTO_INCREMENT 值时，由 INSERT 命令使用的值。这主要与更新日志一起使用。

1.26 GRANT 和 REVOKE 句法

GRANT priv_type [(column_list)] [, priv_type [(column_list)] ...]

ON {tbl_name | * | *.* | db_name.*}

```
    TO user_name [IDENTIFIED BY 'password']
        [, user_name [IDENTIFIED BY 'password'] ...]
    [WITH GRANT OPTION]
REVOKE priv_type [(column_list)] [, priv_type [(column_list)] ...]
    ON {tbl_name | * | *.* | db_name.*}
    FROM user_name [, user_name ...]
```

GRANT 在 MySQL 3.22.11 或以后版本中实现。对于更早 MySQL 版本，GRANT 语句不做任何事情。

GRANT 和 REVOKE 命令允许系统主管在 4 个权限级别上授权和撤回赋予 MySQL 用户的权利：

全局级别

全局权限作用于一个给定服务器上的所有数据库。这些权限存储在 mysql.user 表中。

数据库级别

数据库权限作用于一个给定数据库的所有表。这些权限存储在 mysql.db 和 mysql.host 表中。

表级别

表权限作用于一个给定表的所有列。这些权限存储在 mysql.tables_priv 表中。

列级别

列权限作用于在一个给定表的单个列。这些权限存储在 mysql.columns_priv 表中。

对于 GRANT 如何工作的例子，见第七章。

对于 GRANT 和 REVOKE 语句，priv_type 可以指定下列的任何一个：

ALL PRIVILEGES	FILE	RELOAD
ALTER	INDEX	SELECT
CREATE	INSERT	SHUTDOWN
DELETE	PROCESS	UPDATE
DROP	REFERENCES	USAGE

ALL 是 ALL PRIVILEGES 的一个同义词，REFERENCES 还没被实现，USAGE 当前是“没有权限”的一个同义词。它能用在你想要创建一个没有权限用户的时候。

为了从一个用户撤回 grant 的权限，使用 GRANT OPTION 的一个 priv_type 值：

```
REVOKE GRANT OPTION ON ... FROM ...;
```

对于表，你能指定的唯一 priv_type 值是 SELECT、INSERT、UPDATE、DELETE、CREATE、DROP、GRANT、INDEX 和 ALTER。

对于列，你能指定的唯一 priv_type 值是(即，当你使用一个 column_list 子句时)是 SELECT、INSERT 和 UPDATE。

你能通过使用 ON *.*语法设置全局权限，你能通过使用 ON db_name.*语法设置数据库权限。如果你指定 ON *并且你有一个当前数据库，你将为该数据库设置权限。(警告：

如果你指定 ON *而你没有一个当前数据库，你将影响全局权限!）

为了容纳对任意主机的用户授予的权利，MySQL 支持以user@host 格式指定 user_name 值。如果你想要指定一个特殊字符的一个 user 字符串(例如“-”)，或一个包含特殊字符或通配符的 host 字符串(例如“%”)，你可以用括号括起能用户或主机名字 (例如，'test-user'@'test-hostname')。

你能在主机名中指定通配符。例如，user@"%.loc.gov"适用于在 loc.gov 域中任何主机的 user，并且 user@"144.155.166.%"适用于在 144.155.166 类 C 子网中任何主机的 user。

简单形式的 user 是 user@"%"的一个同义词。注意：如果你允许匿名用户连接 MySQL 服务器(它是缺省的)，你也应该增加所有本地用户如 user@localhost，因为否则，当用户试图从本地机器上登录到 MySQL 服务器时，对于 mysql.user 表中的本地主机的匿名用户条目将被使用！匿名用户通过插入有 User="的条目到 mysql.user 表中来定义。通过执行这个查询，你可以检验它是否作用于你：

mysql> SELECT Host,User FROM mysql.user WHERE User='';

目前，GRANT 仅支持最长 60 个字符的主机、表、数据库和列名。一个用户名字能最多到 16 个字符。

对与一个表或列的权限是由 4 个权限级别的逻辑或形成的。例如，如果 mysql.user 表指定一个用户有一个全局 select 权限，它不能被数据库、表或列的一个条目否认。

对于一个列的权限能如下计算：

global privileges

OR (database privileges AND host privileges)

OR table privileges

OR column privileges

在大多数情况下，你只授予用户一个权限级别上的权限，因此现实通常不象上面所说的那样复杂。:) 权限检查过程的细节在 6 MySQL 存取权限系统中给出。

如果你为一个在 mysql.user 表中不存在的用户/主机名组合授权，一个条目被增加并且保留直到用一个 DELETE 命令删除。换句话说，GRANT 可以创建 user 表的条目，但是 REVOKE 将不删除；你必须明确地使用 DELETE 删除.

在 MySQL 3.22.12 或以后，如果创建一个新用户或如果你有全局授予权限，用户的口令将被设置为由 IDENTIFIED BY 子句指定的口令，如果给出一个。如果用户已经有了一个口令，它被一个新的代替。

警告：如果你创造一个新用户但是不指定一个 IDENTIFIED BY 子句，用户没有口令。这是不安全的。

口令也能用 SET PASSWORD 命令设置。见 1.25 SET OPTION 句法。

如果你为一个数据库授权，如果需要在 mysql.db 表中创建一个条目。当所有为数据库的授权用 REVOKE 删除时，这个条目被删除。

如果一个用户没有在一个表上的任何权限，当用户请求一系列表时，表不被显示(例如，用一个 SHOW TABLES 语句)。

WITH GRANT OPTION 子句给与用户有授予其他用户在指定的权限水平上的任何权

限的能力。你应该谨慎对待你授予他 grant 权限的用户，因为具有不同权限的两个用户也许能合并权限!

你不能授予其他用户你自己不具备的权限； agree 权限允许你放弃你仅仅拥有的那些权限。

要知道，当你将一个特定权限级别上的 grant 授予其他用户，用户已经拥有(或在未来被授予!)的在该级别上的任何权限也可由该用户授权。假定你授权一个用户在一个数据库上的 insert 权限，那么如果你授权在数据库上 select 权限并且指定 WITH GRANT OPTION，用户能不仅放弃 select 权限，还有 insert。如果你授权用户在数据库上的 update 权限，用户能放弃 insert、select 和 update。

你不应该将 alter 权限授予一个一般用户。如果你这样做，用户可以通过重命名表试图颠覆权限系统!

注意，如果你正在使用即使一个用户的表或列的权限，服务器要检查所有用户的表和列权限并且这将使 MySQL 慢下来一点。

当 mysqld 启动时，所有的权限被读入存储器。数据库、表和列权限马上生效，而用户级权限在下一次用户连接时生效。你用 GRANT 或 REVOKE 对受权表执行的更改立即被服务器知晓。如果你手工修改授权表(使用 INSERT、UPDATE 等等)，你应该执行一个 FLUSH PRIVILEGES 语句或运行 mysqladmin flush-privileges 告诉服务器再次装载授权表。见 1.5 权限修改何时生效。

- ANSI SQL 版本的 GRANT 与 MySQL 版本之间的最大差别：
- ANSI SQL 没有全局或数据库级别权限，并且 ANSI SQL 不支持所有 MySQL 支持的权限。
- 当你在 ANSI SQL 抛弃一张表时，表的所有权限均被撤消。如果你在 ANSI SQL 撤销权限，所有基于该权限的授权也被也被撤消。在 MySQL 中，权限只能用明确的 REVOKE 命令或操作 MySQL 授权表抛弃。

1.27 CREATE INDEX 句法

CREATE [UNIQUE] INDEX index_name ON tbl_name (col_name[(length)],...)

CREATE INDEX 语句在 MySQL 版本 3.22 以前不做任何事情。在 3.22 或以后版本中，CREATE INDEX 被映射到一个 ALTER TABLE 语句来创建索引。见 1.8 ALTER TABLE 句法。

通常，你在用 CREATE TABLE 创建表本身时创建表的所有索引。见 1.7 CREATE TABLE 句法。CREATE INDEX 允许你把索引加到现有表中。

一个(col1,col2,...)形式的列表创造一个多列索引。索引值有给定列的值串联而成。

对于 CHAR 和 VARCHAR 列，索引可以只用一个列的部分来创建，使用 col_name(length) 句法。(在 BLOB 和 TEXT 列上需要长度)。下面显示的语句使用 name 列的头 10 个字符创建一个索引：

mysql> CREATE INDEX part_of_name ON customer (name(10));

因为大多数名字通常在头 10 个字符不同，这个索引应该不比从整个 name 列的创建的索引慢多少。另外，在索引使用部分的列能使这个索引成为更小的文件大部分，它能保存很多磁盘空格并且可能也加快 INSERT 操作！

注意，如果你正在使用 MySQL 版本 3.23.2 或更新并且正在使用 MyISAM 表类型，你只能在可以有 NULL 值的列或一个 BLOB/TEXT 列上增加一个索引，

关于 MySQL 如何使用索引的更多信息，见 8.1 索引的使用。

1.28 DROP INDEX 句法

DROP INDEX index_name ON tbl_name

DROP INDEX 从 tbl_name 表抛弃名为 index_name 的索引。DROP INDEX 在 MySQL 3.22 以前的版本中不做任何事情。在 3.22 或以后，DROP INDEX 被映射到一个 ALTER TABLE 语句来抛弃索引。见 1.8 ALTER TABLE 句法。

1.29 注释句法

MySQL 服务器支持# to end of line、-- to end of line 和/* in-line or multiple-line */注释风格：

```
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;
```

注意--注释风格要求你在--以后至少有一个空格！

尽管服务者理解刚才描述的注释句法，mysql 客户分析/* ... */注释的方式上有一些限制：

- 单引号和双引号字符被用来标志一个括起来的字符串的开始，即使在一篇注释内。如果引号在主室内没被第 2 个引号匹配，分析器不知道注释已经结束。如果你正在交互式运行 mysql，你能告知有些混乱，因为提示符从 mysql>变为'>或">。
- 一个分号被用来指出当前的 SQL 语句结束并且跟随它的任何东西表示下一行的开始

当你交互式运行 mysql 时和当你把命令放在一个文件中并用 mysql < some-file 告诉 mysql 从那个文件读它的输入时，这些限制都适用。

MySQL 不支持‘--’的 ANSI SQL 注释风格。

1.30 CREATE FUNCTION/DROP FUNCTION

句法

CREATE [AGGREGATE] FUNCTION function_name RETURNS {STRING|REAL|INTEGER}

SONAME shared_library_name

DROP FUNCTION function_name

一个用户可定义函数(UDF)是用一个像 MySQL 的原生（内置）函数如 ABS()和 CONCAT()的新函数来扩展 MySQL 的方法。

AGGREGATE 是 MySQL 3.23 的一个新选项。一个 AGGREGATE 函数功能就像一个原生 MySQL GROUP 函数如 SUM 或 COUNT()。

CREATE FUNCTION 在 mysql.func 系统表中保存函数名、类型和共享库名。你必须对 mysql 数据库有 insert 和 delete 权限以创建和抛弃函数。

所有活跃的函数在每次服务器启动时被重新装载，除非你使用--skip-grant-tables 选项启动 mysqld，在这种情况下，UDF 初始化被跳过并且 UDF 是无法获得的。（一个活跃函数是一个已经用 CREATE FUNCTION 装载并且没用 DROP FUNCTION 删除的函数。）

你也可以通过用户定义函数(UDF)接口加入函数。关于编写用户可定义函数的指令，。对于 UDF 的工作机制，函数必须用 C 或 C++ 编写，你的操作系统必须支持动态装载并且你必须动态编译了 mysqld(不是静态)。

附录二

MySQL 实用程序

各种 MySQL 程序概述

MYSQLADMIN

MYSQLDUMP

MYSQLIMPORT

MYISAMPACK

myisamchk

2.1 各种 MySQL 程序概述

所有使用 mysqlclient 客户库与服务器通信的 MySQL 客户使用下列环境变量：÷

表 2-1 MySQL 客户程序使用的变量

名字	说明
MYSQL_UNIX_PORT	缺省套接字；用于连接 localhost
MYSQL_TCP_PORT	缺省
MYSQL_PWD	缺省口令
MYSQL_DEBUG	调试时调试-踪迹选项
TMPDIR	临时表/文件被创建的目录

使用 MYSQL_PWD 是不安全的。

“mysql”客户使用 MYSQL_HISTFILE 环境变量中命名的文件来保存命令行历史，历史文件的缺省值是“$HOME/.mysql_history”，这里$HOME 是 HOME 环境变量的值。

所有 MySQL 程序取许多不同的选项，然而，每个 MySQL 程序提供一个--help 选项，你可以使用它得到程序不同选项的完整描述。例如，试一试 mysql --help。

你能用一个选项文件覆盖所有的标准客户程序的缺省选项。

下表简单地描述 MySQL 程序：

myisamchk

描述、检查、优化和修复 MySQL 表的使用程序。

make_binary_release

制作一个编译 MySQL 的一个二进制的版本。这能用 FTP 传送到在 ftp.tcx.se 网站的“/pub/mysql/Incoming”以方便其它 MySQL 用户。

msql2mysql

一个外壳脚本，转换 mSQL 程序到 MySQL。它不能处理所有的情况，但是当转换时，它给出一个好起点。

mysql

mysql 是一个简单的 SQL 外壳(具有 GNU readline 能力)，它支持交互式和非交互式使用。当交互地使用时，查询结果以 ASCII 表的格式被表示。当非交互地使用时(例如，作为一个过滤器)，结果是以定位符分隔的格式表示。(输出格式可以使用命令行选项改变)你可以简单地象这样运行脚本：

```
shell> mysql database < script.sql > output.tab
```

如果你在客户中由于内存不足造成问题，使用--quick 选项！这迫使 mysql 使用

mysql_use_result()而非 mysql_store_result()来检索结果集合。

mysqlaccess

一个脚本，检查对主机、用户和数据库组合的存取权限。

mysqladmin

执行管理操作的实用程序，例如创建或抛弃数据库，再装载授权表，清洗表到磁盘中和再打开日志文件。mysqladmin 也可以被用来从服务器检索版本，进程和状态信息。

mysqlbug

MySQL 错误报告脚本。当填写一份错误报告到 MySQL 邮件列表时，应该总是使用该脚本。

mysqld

SQL 守护进程。它应该一直在运行。

mysqldump

以 SQL 语句或定位符分隔的文本文件将一个 MySQL 数据库倾倒一个文件中。这是最早由 Igor Romanenko 编写的自由软件的增强版本。

mysqlimport

使用 LOAD DATA INFILE 将文本文件倒入其各自的表中。

mysqlshow

显示数据库，表，列和索引的信息。

mysql_install_db

以缺省权限创建 MySQL 授权表。这通常仅被执行一次。就是在系统上第一次安装 MySQL 时。

replace

一个实用程序，由 msql2mysql 使用，但是有更一般的适用性。replace 改变文件中或标准输入上的字符串。使用一台有限状态机首先匹配更长的字符串，能被用来交换字符串。例如，这个命令在给定的文件中交换 a 和 b：

```
shell> replace a b b a -- file1 file2 ...
```

safe_mysqld

一个脚本，用某些更安全的特征启动 mysqld 守护进程，例如当一个错误发生时，重启服务器并且记载运行时刻信息到一个日志文件中。

2.2 mysqladmin

用于执行管理性操作。语法是：

```
shell> mysqladmin [OPTIONS] command [command-option] command ...
```

通过执行 mysqladmin --help，你可以得到你mysqladmin 的版本所支持的一个选项列表。

目前 mysqladmin 支持下列命令：

create databasename　创建一个新数据库

drop databasename　删除一个数据库及其所有表

extended-status　给出服务器的一个扩展状态消息

flush-hosts　洗掉所有缓存的主机
flush-logs　洗掉所有日志
flush-tables　洗掉所有表
flush-privileges　再次装载授权表(同 reload)
kill id,id,...　杀死 mysql 线程
password　新口令，将老口令改为新口令
ping　检查 mysqld 是否活着
processlist　显示服务其中活跃线程列表
reload　重载授权表
refresh　洗掉所有表并关闭和打开日志文件
shutdown　关掉服务器
status　给出服务器的简短状态消息
variables　打印出可用变量
version　得到服务器的版本信息

所有命令可以被缩短为其唯一的前缀。例如：

```
shell> mysqladmin proc stat
+----+------+---------+----+-----------+-----+------+-----+
| Id | User | Host    | db | Command   | Time | State | Info |
+----+------+---------+----+-----------+-----+------+-----+
| 6  | monty | localhost |  | Processlist | 0  |      |     |
+----+------+---------+----+-----------+-----+------+-----+
Uptime: 10077  Threads: 1  Questions: 9  Slow queries: 0  Opens: 6  Flush tables: 1
Open tables: 2  Memory in use: 1092K  Max memory used: 1116K
```

2.3 mysqldump

实用程序，为备份或为把数据转移到另外的 SQL 服务器上倾倒一个数据库或许多数据库。倾倒将包含 创建表或充实表的 SQL 语句。

shell> mysqldump [OPTIONS] database [tables]

如果你不给定任何表，整个数据库将被倾倒。

通过执行 mysqldump --help，你能得到你 mysqldump 的版本支持的选项表。

注意，如果你运行 mysqldump 没有--quick 或--opt 选项，mysqldump 将在倾倒结果前装载整个结果集到内存中，如果你正在倾倒一个大的数据库，这将可能是一个问题。

mysqldump 支持下列选项：

--add-locks

在每个表倾倒之前增加 LOCK TABLES 并且之后 UNLOCK TABLE。(为了使得更快地插入到 MySQL)。

--add-drop-table

在每个 create 语句之前增加一个 drop table。

--allow-keywords

允许创建是关键词的列名字。这由表名前缀于每个列名做到。

-c, --complete-insert

使用完整的 insert 语句(用列名字)。

-C, --compress

如果客户和服务器均支持压缩，压缩两者间所有的信息。

--delayed

用 INSERT DELAYED 命令插入行。

-e, --extended-insert

使用全新多行 INSERT 语法。（给出更紧缩并且更快的插入语句）

-#, --debug[=option_string]

跟踪程序的使用(为了调试)。

--help

显示一条帮助消息并且退出。

--fields-terminated-by=...

--fields-enclosed-by=...

--fields-optionally-enclosed-by=...

--fields-escaped-by=...

--fields-terminated-by=...

这些选择与-T 选择一起使用，并且有相应的 LOAD DATA INFILE 子句相同的含义。见附录 1.16 LOAD DATA INFILE 语法。

-F, --flush-logs

在开始倾倒前，洗掉在 MySQL 服务器中的日志文件。

-f, --force,

即使我们在一个表倾倒期间得到一个 SQL 错误，继续。

-h, --host=..

从命名的主机上的 MySQL 服务器倾倒数据。缺省主机是 localhost。

-l, --lock-tables.

为开始倾倒锁定所有表。

-t, --no-create-info

不写入表创建信息(CREATE TABLE 语句）

-d, --no-data

不写入表的任何行信息。如果你只想得到一个表的结构的倾倒，这是很有用的!

--opt

同--quick --add-drop-table --add-locks --extended-insert --lock-tables。应该给你为读入一个 MySQL 服务器的尽可能最快的倾倒。

-pyour_pass, --password[=your_pass]

与服务器连接时使用的口令。如果你不指定“=your_pass”部分，mysqldump 需要来自终端的口令。

-P port_num, --port=port_num

与一台主机连接时使用的TCP/IP端口号。(这用于连接到localhost以外的主机，因为它使用 Unix 套接字。)

-q, --quick

不缓冲查询，直接倾倒至stdout；使用mysql_use_result()做它。

-S /path/to/socket, --socket=/path/to/socket

与localhost连接时（它是缺省主机)使用的套接字文件。

-T, --tab=path-to-some-directory

对于每个给定的表，创建一个table_name.sql文件，它包含SQL CREATE 命令，和一个table_name.txt文件，它包含数据。 注意：这只有在mysqldump运行在mysqld守护进程运行的同一台机器上的时候才工作。.txt文件的格式根据--fields-xxx和--lines--xxx选项来定。

-u user_name, --user=user_name

与服务器连接时，MySQL使用的用户名。缺省值是你的Unix登录名。

-O var=option, --set-variable var=option

设置一个变量的值。可能的变量被列在下面。

-v, --verbose

冗长模式。打印出程序所做的更多的信息。

-V, --version

打印版本信息并且退出。

-w, --where='where-condition'

只倾倒被选择了的记录；注意引号是强制的!

"--where=user='jimf'" "-wuserid>1" "-wuserid<1"

最常见的mysqldump使用可能制作整个数据库的一个备份：

```
mysqldump --opt database > backup-file.sql
```

但是它对用来自于一个数据库的信息充实另外一个MySQL数据库也是有用的：

```
mysqldump --opt database | mysql --host=remote-host -C database
```

2.4 mysqlimport

mysqlimport提供一个到LOAD DATA INFILESQL语句的命令行接口。mysqlimport的大多数选项直接对应于LOAD DATA INFILE的相同选项。见附录1.16 LOAD DATA INFILE语法。

mysqlimport象这样调用：

```
shell> mysqlimport [options] filename ...
```

对于在命令行上命名的每个文本文件，mysqlimport剥去文件名的扩展名并且使用它决定哪个表导入文件的内容。例如，名为“patient.txt”、“patient.text”和“patient”将全部被

导入名为 patient 的一个表中。

mysqlimport 支持下列选项：

-C, --compress

如果客户和服务器均支持压缩，压缩两者之间的所有信息。

-#, --debug[=option_string]

跟踪程序的使用(为调试)。

-d, --delete

在导入文本文件前倒空表格。

--fields-terminated-by=...

--fields-enclosed-by=...

--fields-optionally-enclosed-by=...

--fields-escaped-by=...

--fields-terminated-by=...

这些选项与对应于 LOAD DATA INFILE 的子句相同的含义。见 7.16 LOAD DATA INFILE 语法。

-f, --force

忽略错误。例如，如果对于一个文本文件的一个表不存在，继续处理任何余下的文件。没有--force，如果表不存在，mysqlimport 退出。

--help

显示一条帮助消息并且退出。

-h host_name, --host=host_name

导入数据到命名的主机上的 MySQL 服务器。缺省主机是 localhost。

-i, --ignore

见为--replace 选项的描述。

-l, --lock-tables

在处理任何文本文件前为写入所定所有的表。这保证所有的表在服务器上被同步。

-L, --local

从客户读取输入文件。缺省地，如果你连接 localhost(它是缺省主机)，文本文件被假定在服务器上。

-pyour_pass, --password[=your_pass]

与服务器连接时使用的口令。如果你不指定“=your_pass”部分，mysqlimport 要求来自终端的口令。

-P port_num, --port=port_num

与一台主机连接时使用的 TCP/IP 端口号。(这被用于连接到除 localhost 以外的主机，因为它使用 Unix 套接字。)

-r, --replace

--replace 和--ignore 选项控制对输入在唯一键值上有重复的现有记录的输入处理。如果你指定--replace，新行将代替有相同唯一键的存在的行。如果你指定--ignore，跳过输入在

唯一键值上有重复的现有记录。如果你不指定任何一个选项，当找到一个重复的键值，出现一个错误，并且文本文件余下部分被忽略。

-s, --silent

安静模式。只有在错误发生时，写出输出。

-S /path/to/socket, –socket=/path/to/socket

与 localhost(它是缺省主机)连接时使用的套接字文件。

-u user_name, --user=user_name

MySQL 使用的用户名字当与服务者联接时。缺省价值是你的 Unix 登录名字。

-v, --verbose

冗长模式。打印程序所做的更多信息。

-V, --version

打印版本信息并且退出。

2.5 myisampack

myisampack 被用来压缩 MyISAM 表，而 pack_isam 被用来压缩 ISAM 表。由于 ISAM 表被淘汰，这里我们将只讨论 myisampack。

myisampack 是当你订购超过 10 个许可证或扩展的支持时，你得到的一个额外的实用程序。因为这些仅以二进制形式被分发，他们仅在某些平台上可用。

下面我们仅谈论 myisampack, 但是每件事情对 pack_isam 也是持有的。

myisampack 通过单独压缩表中的每个列来工作。当表被打开时，需要加压缩的信息被读进内存，这使得在存取单个记录时能得到更好的性能，因为你只需要解压缩一个记录，不是更大的磁盘块，象在 MSDOS 上使用 Stacker 时一样。通常，myisampack 压缩数据文件 40%-70%。

MySQL 使用内存映射(mmap())在压缩表上而如果 mmap()的使用不工作，倒回到正常的读/写文件。

当前 myisampack 有 2 个限制：

在压缩后，表只能读。

myisampack 也能压缩 BLOB 或 TEXT 列。较老的 pack_isam 不能做到。

修正这些限制以在我们的 TODO 表上，但是具有低优先级。

myisampack 象这样调用：

shell> myisampack [options] filename ...

每个文件名应该是一个索引(“.MYI”) 文件名。如果你不在数据库目录下，你应该指定文件的路径名。允许省略“.MYI”扩展名。

myisampack 支持下列选项：

-b, --backup

制作表的一个备份，为 tbl_name.OLD。

-#, --debug=debug_options

输出调试日志。debug_options 串经常是'd:t:o,filename'。

-f, --force

即使它变得更大或如果临时文件存在，强制表的压缩。(myisampack 在压缩表时创建一个名位“tbl_name.TMD”的临时文件。如果你杀死 myisampack，“.TMD”文件不能被删除。通常，如果 myisampack 发现“tbl_name.TMD”存在，它以一个错误退出。用--force，myisampack 不管怎样都压缩表。

-?, --help

显示一条帮助消息并且退出。

-j big_tbl_name, --join=big_tbl_name

联结所有在命令行上被命名的表到一个单独的表 big_tbl_name 中。所有要被合并的表必须是相同的(同样的列名字和类型，同样的索引，等等。）

-p #, --packlength=#

指定记录长度存储尺寸，按字节。值应该是 1、2 或 3。(myisampack 用 1、2 或 3 字节的长度指针存储所有行。在最一般的情况下，myisampack 在它开始包装文件以前，能确定正确的长度值，但是它可能注意到在包装过程期间，它能使用了更短的长度。在这种情况下，myisampack 在下一次你包装同样文件时间打印出一条提示，你可以使用更短的记录长度。）

-s, --silent

安静模式。只有当错误发生时，写出输出。

-t, --test

不压缩表，仅仅测试压缩它。

-T dir_name, --tmp_dir=dir_name

使用命名的目录作为写入临时表的位置。

-v, --verbose

冗长模式。写出有关进展和包装结果的信息。

-V, --version

显示版本信息和出口。

-w, --wait

如果表正在使用，等待并且再试。如果 mysqld 服务器以--skip-locking 选项被调用，如果表可能在包装过程中被更新，调用 myisampack 不是一个好主意。

下面显示的命令顺序说明了一个典型的表压缩表压缩过程：

```
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
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
```

（某些过长的输出已经省略）

由 myisampack 打印的信息在下面描述：

normal

不是用额外压缩的列数。

empty-space

仅包含空格值的列数；这些将占据 1 位。

empty-zero

只包含二进制 0 值的列数；这些将占据 1 位。

empty-fill

不占据其类型全部字节范围的整形列数；这些被改变为一种更小的类型(例如，一个 INTEGER 列可以被改变为 MEDIUMINT)。

pre-space

用前导空间存储的小数的列数。在这种情况下，每个值将包含一个前导空格的数量的计数。

end-space

有很多拖后空格的列数。在这种情况下，每个值将包含一个拖后空格的数量的计数。

table-lookup

列只有少数不同的值，并且它在哈夫曼压缩前被变换一个 ENUM。

zero

所有值为零的列数。

Original trees

哈夫曼树的初始数目。

After join

在联结哈夫曼树以节省一些表头空间后余下的不同树的数量。

在一张表被压缩以后，myisamchk -dvv 打印出每个字段的额外信息：

Type

字段类型可以包含下列描述符：

constant

所有行有相同的值。

no endspace

不存储尾空格。

no endspace, not_always

不存储尾空格而且不对所有值做尾空格压缩。

no endspace, no empty

不存储尾空格。不存储空值。

table-lookup
列被变换到一个 ENUM。
zerofill(n)
值中最高 n 位总是 0 并且不被存储。
no zeros
不存储零。
always zero
0 值以 1 位被存储。
Huff tree
与字段相关的哈夫曼树
Bits
在哈夫曼树里使用的位数。

2.6 myisamchk

myisamchk 这样调用：

shell> myisamchk [options] tbl_name

options 指定你想要 myisamchk 做什么。他们在下面描述。（你也可以通过调用 myisamchk --help 得到一张选项表。) 没有选项，myisamchk 简单地检查你的表。为了得到更多的信息或告诉 myisamchk 执行校正操作，指定在下面和下小节描述的选项择。

tbl_name 是你想要检查的数据库表。如果你不在数据库目录的某处运行 myisamchk，你必须指定到文件的路径，因为 myisamchk 不知道你的数据库位于哪儿。实际上，myisamchk 别在乎你正在操作的文件是否位于一个数据库目录；你可以拷贝对应于一张数据库表的文件到别处并且在那里执行恢复操作。

如果你愿意，你可以 myisamchk 命令行命名几个表。你也能指定一个名字作为一个索引文件(用“ .MYI”后缀)，它允许你通过使用模式“*.MYI”指定在一个目录所有的表。例如，如果你在一个数据库目录，你可以这样在目录下检查所有的表：

shell> myisamchk *.MYI

如果你不在数据库目录下，你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

myisamchk 支持下列选项：

-a, --analyze

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。

-#, --debug=debug_options

输出调试记录文件。debug_options 字符串经常是'd:t:o,filename'。

-d, --description

打印出关于表的一些信息。

-e, --extend-check

非常彻底地检查表。这仅在极端情况下是必要的。通常，myisamchk 应该找出所有错误，即使没有改选项。

-f, --force

覆盖老的临时文件。如果你在检查表时使用-f (运行 myisamchk 没有-r)，myisamchk 在检查期间将自动为出现一个错误的表用-r 重启。

--help

显示一条帮助消息并且退出。

-i, --information

打印有关被检查的表的信息统计。

-k #, --keys-used=#

与-r 一起使用。告诉ISAM 表处理器仅更新头#个索引。较高编号的索引被撤销。这能用来使插入变得更快！撤销的索引能通过使用 myisamchk -r 被重新激活。

-l, --no-symlinks

在修复时，不跟随符号连接。通常 myisamchk 修复一个符号连接所指的表。

-q, --quick

与-r 一起使用使得一个修复更快。通常，原来的数据文件没被接触；你能指定第二个-q 强制使用原来的数据文件。

-r, --recover

恢复模式。可以修复几乎所有一切，除非唯一的键不是唯一。

-o, --safe-recover

恢复模式。使用一个老的恢复方法；这比-r 慢些，但是能处理一-r 不能处理的情况。

-O var=option, --set-variable var=option

设置一个变量的值。可能的变量列在下面。

-s, --silent

沉默模式。当错误发生时，仅写输出。你能使用-s 两次(-ss)非常沉默地做 myisamchk。

-S, --sort-index

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。

-R index_num, --sort-records=index_num

根据一个索引排序记录。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。

-u, --unpack

解开一个用 myisampack 压缩的表。

-v, --verbose

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)！

-V, --version

打印 myisamchk 版本并退出。

-w, --wait

如果表被锁定，等待。

对--set-variable(-O)选项，可能的变量是：

```
key_buffer_size       当前值: 16776192
read_buffer_size      当前值: 262136
write_buffer_size     当前值: 262136
sort_buffer_size      当前值: 2097144
sort_key_blocks       当前值: 16
decode_bits           当前值: 9
```