Mastering MySQL 4

MySQL 4 从入门到精通

〔美〕 Ian Gilfillan 著
王 军 等译

- 内容全面，可供用户、管理员和开发人员使用
- 掌握MySQL 4的新增功能
- 附有对常用编程API的参考，如MySQL、Syntax、Perl DBI和PHP

SYBEX
電子工業出版社
Publishing House of Electronics Industry
http://www.phei.com.cn

Mastering
MySQL 4

MySQL 4 从入门到精通

MySQL已经证明了它完全有能力与数据库管理领域的重量级产品竞争，如SQL Server和Oracle，而MySQL 4则更为出色。本书包括了所有的信息资源，使读者可以创建、维护、使用和扩展MySQL数据库。对于在MySQL环境中开发数据库应用程序的人员，本书也是一本很重要的读物。如果你是一个MySQL新手，本书让你尽快上手。如果你是升级用户，本书帮助你迅速掌握新的功能。内容包括：

- 安装MySQL
- 添加、删除和更新记录
- 选择适当的表类型
- 选择适当的数据类型
- 优化MySQL查询
- 设计数据库
- 对表结构进行一般化处理
- 维护和修复数据库
- 备份和复制数据
- 管理用户权限和安全
- 优化MySQL服务器
- 扩展MySQL
- 开发数据库应用程序

作者简介：

Ian Gilfillan是南非第一新闻门户网站Independent Online的首席开发者。他也开发和教授过MySQL、数据库、编程和网络开发等课程。作为许多数据库和编程方面文章的作者和南非第一个在线商店的开发者，Ian从1997年就开始使用MySQL了。

最新推出SYBEX子公司图书中译本：

UML与Rational Rose 2002从入门到精通
Delphi 6从入门到精通
Java 2从入门到精通(J2SE 1.4版)
JavaScript从入门到精通(黄金版)
JSP从入门到精通
Visual Basic.NET从入门到精通
Visual C#.NET从入门到精通
Visual Basic.NET数据库编程从入门到精通
Visual Basic 6从入门到精通
Visual C++ 6从入门到精通
XML从入门到精通(黄金版)
ASP.NET与VB.NET从入门到精通
ASP.NET与C#从入门到精通
XSLT从入门到精通
PHP 4.1从入门到精通
Kylix 2从入门到精通
Cisco路由器从入门到精通（第二版）
网络布线从入门到精通
网络安全从入门到精通(第二版)
网络安全积极防御从入门到精通
SQL Server 2000从入门到精通
SBS 2000服务器从入门到精通
Windows 2000 Server从入门到精通（第四版）
Windows Server 2003活动目录从入门到精通(第三版)
Windows XP家庭版从入门到精通(中文版)
Windows XP专业版从入门到精通(中文版)
Windows XP注册表从入门到精通
Red Hat Linux 7从入门到精通
AutoCAD 2002从入门到精通
Dreamweaver MX数据库从入门到精通
Autodesk VIZ 4从入门到精通
水晶报表Crystal Reports 9从入门到精通

ISBN 7-5053-8673-5

9 787505 386730 >

SYBEX

PHEI

美迪亚

责任编辑：春 丽

ISBN 7-5053-8673-5/TP · 5035

定价：66.00元

Mastering MySQL 4

MySQL 4从入门到精通

〔美〕 Ian Gilfillan 著

王 军 等译

電子工業出版社

Publishing House of Electronics Industry

北京 · BEIJING

内 容 提 要

MySQL已经是一个成熟的、能够完成重要任务的数据管理解决方案。而本书包括了成为熟练的MySQL DBA或开发人员所需要的知识。其内容从MySQL的运行方法及其数据和表的类型开始，过渡到高级SQL、索引和优化以及MySQL编程和扩展；接下来的第二部分则介绍了设计数据库的方法和一些容易被忽略的问题；第三部分讲述了优化高性能数据库、备份、复制、安全和安装等MySQL管理方面的问题；最后的附录部分则提供了读者所需的重要参考。

本书适用于应用程序开发人员、数据库管理员和普通MySQL用户。

版权贸易合同登记号：01-2002-6040

图书在版编目（CIP）数据

MySQL 4从入门到精通/（美）吉尔费伦（Gilfillan, L.）著；王军等译.—北京：电子工业出版社，2003.6
书名原文：Mastering MySQL 4
ISBN 7-5053-8673-5

Ⅰ. M… Ⅱ. ①吉… ②王… Ⅲ. 关系数据库－数据库管理系统，MySQL Ⅳ. TP311.138

中国版本图书馆CIP数据核字（2003）第030646号

责任编辑：春 丽
印 刷：北京天竺颖华印刷厂
出版发行：电子工业出版社 http://www.phei.com.cn
北京市海淀区万寿路173信箱 邮编：100036
北京市海淀区翠微东里甲2号 邮编：100036
经 销：各地新华书店
开 本：787×1092 1/16 印张：40.625 字数：1030千字
版 次：2003年6月第1版 2003年6月第1次印刷
定 价：66.00元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换，若书店售缺，请与本社发行部联系。联系电话：（010）68279077

致　谢

这是我第一次写书，以前只是写过很多文章，但我认为一本书实际上就是一篇大的文章。现在，我深刻地理解了：离开那些帮助过我的人们，这本书无疑是不可能成形的。首先，要感谢Anique van der Vlugt，他安装了相关软件、测试代码并在孤独的长夜中校对了其中的一些章节。第二，感谢Bob Meredith，他建立了一个测试服务器，并经常对此书提出问题。还要感谢Rushdi Salie和Web Factory，是他们将我引入了MySQL世界；在Linux、MySQL、Python和Perl广为人知之前，Pieter Claassen就已经对它们产生了强烈的热情，在此表示感谢。感谢我的雇主IOL，在我写书期间没有过多地监视我的工作时间。感谢技术编辑Charlie Hornberger提供了一些极好的注释，以及Sybex中所有为我提供过帮助的工作人员。开发编辑Tom Cirtin是我的第一个读者，制作编辑Donna Crossman、编辑Kim Wimpsett、副总编Joel Fugazzotto、插图提供人Tony Jonick使我的手稿形成一本书，还有合成人员Rozi Harris、排版人员Bill Clark、校对人员Amey Garber、Emily Hsuan、Dave Nash、Laurie O'Connell、Yariv Rabinovitch、Nancy Riddiough、Monique van den Berg、索引员Nancy Guenther，在此都要表示衷心的感谢。

当然还要感谢MySQL AB提供了如此伟大的产品。

译 者 的 话

在本书的翻译过程中，得到了魏巍、赵菁、赵宇、涂明华、蒙小斌、孙永强、张吉祥、郭颖浩、陈旌、何文、陈雪松、刘体争等同志的协作，许萍、张雯静完成了本书的录入工作，方勇、龚涛、宋爱华、周小暄帮助进行了书稿与打印稿的校对，在此深表感谢。

译者

简　介

MySQL已经是一个成熟的产品了。MySQL曾经是作为初级用户上网的玩具发布的，但它现在已经成为一个能够完成重要任务的数据管理解决方案。在成为网站的一个理想选择之前，它现在所包含的许多功能都是其他环境所需要的，尤其是在保持其令人惊讶的速度方面。它已经在速度方面胜过许多商业解决方案，并且有着一个精巧和强大的许可系统，但是现在的版本4具有ACID兼容的InnoDB事务存储引擎。

MySQL 4比较快，具有联机备份工具，且具有许多新的功能。没有理由不考虑将MySQL作为你的数据库解决方案。MySQL AB是推出MySQL的公司，提供高效且低成本的支持，与大多数开放源组织一样，你可以在网络上发现许多免费的支持。还没有被包含在MySQL中的那些标准功能（如视图和存储程序）现在正在进行开发，也许在你阅读本书的时候就完成了。

有很多理由可以使你选择MySQL作为重要任务的数据管理解决方案：

成本　MySQL是免费的，并且它的技术支持也很便宜。

支持　MySQL AB提供便宜的技术支持，并且还有一个庞大的活跃的MySQL组织。

速度　MySQL胜过它的大多数竞争对手。

功能　MySQL提供了开发人员所需要的大多数功能，如完全的ACID兼容、支持大多数ANSI SQL、联机备份、复制、安全套接层（Secure Sockets Layer，SSL）支持、与几乎所有编程环境的集成。同时，MySQL的开发和更新比大多数竞争者要快，其尚未包括的几乎所有标准功能都在开发的过程中。

可移植　MySQL可以在绝大多数操作系统中运行，在大多数情况下，数据可以从一个系统传送到另一个系统，没有任何困难。

易用　MySQL易于使用和管理。许多老的数据库受到遗留问题的影响，给管理带来不必要的麻烦。MySQL的工具非常强大灵活，但这并不改变其易用性。

读者对象

本书针对开发人员、数据库管理人员（DBA）和MySQL用户。书中包含如下主题：

- 探讨MySQL所实现的结构化查询语言（SQL）。
- 理解并使用数据和表类型。
- 优化查询和索引。
- 备份数据库。
- 管理用户和安全。
- 管理并配置MySQL（以及为了性能而优化配置）。
- 在多服务器上复制MySQL。

- 理解数据库设计和数据库标准化，并介绍一个完整的实例。如果要将MySQL用于很重要的应用中，关于此主题的知识点很重要。
- 用MySQL进行编程。
- 编写MySQL的扩展名。
- 安装MySQL。

本书的附录包括如下内容：

- 一份完整的MySQL参考。
- 参考包括PHP、Perl、C、Java、Python、ODBC功能和与MySQL进行互操作的方法。

本书不包括什么

MySQL是一个很宽泛的主题，本书包括成为熟练的MySQL DBA或开发人员所需要的知识。然而，一本书不可能解释所有的情况，因此书中不包括下列内容：

- 如何编程。本书在用MySQL进行编程方面提供帮助，但并不会从头教你如何编程。
- 嵌入式MySQL。
- 彻底讲解如何编译和安装库。编写自己的扩展名要求对在特定平台上编译和安装库有所了解。尽管也涉及了这个内容，但本书无法覆盖所有平台上的所有可能配置，因此，如果你想达到这个水平，就需要特定平台的好的信息源。

你需要什么

你将需要下列软件，以实践本书中的例子：

- MySQL客户机和服务器。你可以从MySQL的站点www.mysql.com下载当前版本。
- 一个可以在其上安装MySQL的系统（如果还没有的话）。MySQL可以安装在台式机上，但如果用于重要的应用中，最好运行在一台专门的服务器上。
- 如果要用MySQL开发应用程序，需要为自己的开发环境下载最新的驱动程序或API（应用编程接口）。MySQL与PHP、Perl、Java、C、C++和Python结合得最好，但你也可以在其他任何编程环境中使用它，如.NET开放数据库连接（Open Database Connectivity，ODBC）。访问MySQL的网站www.mysql.com，可以下载驱动程序的当前版本。

如何使用本书

本书分为四大部分。如果你是一个数据库新手，最好从第一部分“使用MySQL”开始。第1章“认识MySQL”使初级用户轻松进入SQL的世界。已经用过其他数据库管理系统的读者也可以浏览一下第1章，以熟悉MySQL的运行方法，之后就可以到第2章“数据类型和表类型”学习MySQL数据和表的类型。中级水平的读者可以从第3章“高级SQL”和第4章“索引和查询优化”开始，这两章介绍高级SQL、索引和优化。想要将某种编程语言和MySQL结合使用的读者应当阅读第5章“MySQL编程”和关于该种语言的附录。第6章“扩展

MySQL”针对已经理解了MySQL并想添加自己的功能的读者。

不具有正式数据库设计知识的读者将从第二部分“设计一个数据库”获益，其中包括了经常容易被忽略的数据库设计问题，而这些问题是开发大型数据库时所需要的。

需要管理MySQL的读者将从第三部分“MySQL管理”获益，该部分从新用户的基本知识发展到了一些高级问题，如优化高性能数据库。其中还解释了备份、复制、安全和安装。

最后是附录，在你需要MySQL SQL、函数和操作者参考时，以及需要参考数据库功能和用于大多数编程语言的方法时使用。

书中的示例代码可以在www.sybex.com上下载。

联系作者

首先需要查看Sybex的网站www.sybex.com。可以随时联系作者，地址是mysql4@greenman.co.za，我将尽可能地提供帮助或者指引正确的方向。

目　录

第一部分　使用MySQL

第1章　认识MySQL

现在，本书已经呈现在你的面前。有些读者可能已经很熟悉MySQL，希望更多地了解数据库复制和优化服务器变量。如果你是一个高级用户，请跳过本章。本章不适合你。但是，初学者不要着急。本书包括初学者入门所需的所有内容，使之最终成为高级用户。

MySQL是当今世界最流行的开放源（open-source）数据库。开放源的意思是，源代码（即组成MySQL的编程代码）对任何人都是免费的。世界各地的人们可以向MySQL添加内容、修复其漏洞、改进MySQL或者提出优化建议。确实是这样。MySQL由几年前的一个“玩具”数据库发展到成熟的版本4，已经赶上了许多商业数据库，威胁到了其他的数据库生产厂商。MySQL成长得相当快，这是因为有无数人在此项目中做出贡献，MySQL小组的成员也做出了很大贡献。

私有项目中的源代码由几个人来编写，并被谨慎地监测；与此不同，开放源项目不排斥任何对此感兴趣的人，只要他有足够的能力。2000年，MySQL才刚刚出现4年，其创始人Michael“Monty”Widenius就在参加开放源代码数据库大会时预测了其美好的前景。许多当时的数据库厂商对此不以为然，他们当中的一部分后来就再也没有音信了。

到了第3个版本，MySQL就控制了因特网市场上的低端数据库。到了第4个版本，该产品吸引了更大范围的客户。随着开放源Apache吸引了网络服务器市场，以及各种开放源操作系统（如Linux和FreeBSD）在服务器市场的强劲趋势，数据库市场中的MySQL时代已经到来。

本章内容包括：

- 基本的数据库概念和术语
- 连接MySQL Server和断开MySQL Server的连接
- 创建和关闭数据库
- 创建、更新和关闭表
- 向表中添加数据
- 从表返回数据和删除数据
- 理解基本的统计和日期功能
- 连接一个以上的表

理解MySQL基础

MySQL是一个关系数据库管理系统（RDBMS）。它是一个程序，可以存储大量的种类繁多的数据，并且提供服务以满足任何组织的需要，包括零售商店、大的商业企业和政府实体。MySQL的竞争对手都是有名的商业RDBMS，如Oracle、SQL Server和DB2。

MySQL包括安装该程序、建立不同级别的用户访问、管理该系统、保护并备份数据所需要的一切。可以用当今使用的大多数编程语言来开发数据库应用程序，并在大多数操作系统上运行它们，包括一些你可能从来没有听说过的操作系统。MySQL使用结构化查询语言（SQL，Structured Query Language），所有的关系数据库都使用该语言，本章后面将详细介绍（参见“创建并使用你的第一个数据库”一节）。SQL可以创建数据库，并根据特定标准来添加、操作和检索数据。

本章介绍了关系数据库的概念。你将学习什么是关系数据库、它是如何工作的，以及一些关键的术语。获取了这些信息，你就可以开始创建一个简单的数据库并处理其中的数据了。

什么是数据库

理解数据库的最简单方法是将它理解为一个相关文件的集合。我们考虑一个商店的销售订单文件（纸上的或电子的）。我们还有另外的产品文件，包括库存记录。要完成一个订单，需要先在订单文件中查找产品，然后在产品文件中查找该产品的库存。数据库和控制数据库的软件，即数据库管理系统（DBMS，database management system），帮助完成此类任务。现在的大多数数据库都是关系数据库，如此命名是因为它们所处理的数据表是通过一个共有的字段相关的。例如，表1.1显示了产品表，表1.2显示了发票表。我们看到，两个表之间的关系基于它们共有的字段stock_code。任何两个表都可以简单地通过拥有共有的字段相互建立关系。

表1.1 产品表

STOCK_CODE	描述	价格
A416	钉子，盒	0.14美元
C923	图钉，盒	0.08美元

表1.2 发票表

INVOICE_CODE	INVOICE_LINE	STOCK_CODE	数量
3804	1	A416	10
3804	2	C923	15

数据库术语

我们详细介绍前面的两个表，看它们是如何组织的：

- 每个表由多个行和列组成。
- 每一行包括一个单独实体（如一个产品或一个订单）的数据。这叫做记录。例如，表1.1中的第一行是一条记录，它描述产品A416，这是一盒钉子，价值14美分。也就是说，行和记录是可互换的两个概念。
- 每一列包括与该记录相关的一项数据，叫做属性。卖出产品的数量或产品的价格都是属性的例子。在介绍数据库表的时候，属性叫做字段。例如，在表1.1的“描述”列中的数据就是字段。也就是说，属性和字段是可互换的两个概念。

有了这种结构，数据库将提供处理数据的方法：SQL。SQL是搜索记录和更改记录的强大方法。尽管有些数据库管理系统已经增强了自身功能，但是它们中的大多数都使用SQL。这意味着，在本章和后续章节介绍SQL时，并不仅针对MySQL。其中的大多数内容可以用在任何其他的关系数据库中，如PostgreSQL、Oracle、Sybase或SQL Server。但了解了MySQL的优点之后，你就不会再想改变了！

连接到MySQL Server

运行MySQL并存储数据的机器叫做MySQL Server。要连接该服务器，你有几种选择。第一，你可以在台式机上安装MySQL客户端和MySQL服务器，如图1.1所示。第二，可以将MySQL客户端安装在台式机上，将MySQL服务器安装在另一台已连接的机器上，如图1.2所示。最后，你的台式机可以是连接到MySQL客户端的任何一台计算机，它反过来与MySQL服务器相连，既可以是同一台机器，也可以是另一台机器，如图1.3所示。

MySQL客户端和服务器

图1.1 你的机器上既有MySQL客户端也有MySQL服务器

台式机和MySQL客户端　　MySQL服务器

图1.2 你的机器上有MySQL客户端。MySQL服务器在另一台已经连接的机器上

台式机　　MySQL客户端　　MySQL服务器

图1.3 在这种情况下，你的终端可以是有能力相互连接的任何一台机器，它甚至不运行MySQL客户端

如果MySQL客户端不在你的台式机上，而你又需要与第二台机器连接以使用MySQL客户端，那么可能就需要使用Telnet或Secure Shell（SSH）客户端来操作。使用它们时，先打开Telnet，然后输入主机名、用户名和口令。如果你对此不确定，就向系统管理员寻求帮助。

登录到一台安装了MySQL客户端的机器后，再连接服务器就很容易了：

- 在UNIX机器上（例如，Linux或FreeBSD），从命令行运行如下命令：

```
% mysql -h hostname -u username -ppassword databasename
```

- 在Windows机器上，从命令提示符处运行同一命令：

```
% mysql -h hostname -u username -ppassword databasename
```

%是指命令提示符。它在你的机器上可能不太一样，例如，有些Windows机器上是c:\>，而有些UNIX机器上是$。-h和-u后面可以有一个空格（你也可以去掉空格），但是-p后面必须立刻加上口令，中间没有空格。

连接上之后，就会看到mysql>提示，它在你连接之后出现在大多数命令行上。不需要输入，它会自动出现。即使出现稍微不同的提示符，也不要着急，只需输入黑体显示的文本。这是本书中使用的惯例。

提示： 输入口令时有一个比较安全的方法，在多用户环境中，我推荐使用它。只输入-p，然后省略口令。当MySQL启动时，会提示你输入口令；此时输入口令，它不会出现在屏幕上。这就避免了有人看到你以纯文本输入的口令。

主机名是宿主服务器的机器（例如，www.sybex.com或196.30.168.20）。如果已经登录服务器（也就是说，MySQL客户端和服务器在同一台机器上），就不需要主机名。管理员为你分配用户名和口令（就是你的MySQL口令和用户名，与登录客户机的不一样）。有些不安全的系统不要求任何用户名或口令。

提示： 有些系统管理员不把MySQL放到默认路径中，这会给用户造成不便。那么，输入mysql命令，就会得到command not found错误（UNIX）或bad command or file name错误（Windows），尽管你知道已经安装了MySQL。在这种情况下，需要输入到MySQL客户端的完整路径（例如，/usr/local/build/mysql/bin/mysql或C:\mysql\bin\mysql）。如果有此问题，向管理员询问正确的路径。

要想断开连接，只输入QUIT，如下：

```
mysql> QUIT
Bye
```

也可以输入EXIT或按Ctrl+D。

说明： MySQL不区分大写字母和小写字母。如果愿意，你可以输入QUIT、quit或qUIt。

创建并使用你的第一个数据库

下面的小节介绍如何创建一个数据库并在上面执行查询。假设你已经连接到MySQL服务器，并且有权限使用数据库。否则，向管理员请求权限。可以将数据库命名为firstdb，请

管理员创建数据库并将该数据库的完全权限赋予你。这样，你就不会在后面遇到权限问题，但千万不要破坏现有数据库。如果管理员糊涂了，或者你是自己刚刚安装的MySQL，那么你或者管理员就需要执行下面两组命令之一，以便能够开始。记住，只输入黑体的文本。

如果刚刚安装MySQL

首先，作为根连接到MySQL。因为你才刚刚开始，所以还没有根口令，首先要做的是向根用户分配一个口令（参见第14章“数据库安全”）。

```
% mysql -u root mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15 to server version: 4.0.2-alpha-Max

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> SET PASSWORD=PASSWORD('g00r002b');
Query OK, 0 rows affected (0.00 sec)
```

为了方便使用，为根用户使用同一口令g00r002b，作为将要创建的用户guru2b。

接下来，就要创建firstdb数据库，后面将要处理它：

```
mysql> CREATE DATABASE firstdb;
Query OK, 1 row affected (0.01 sec)
```

最后，用户guru2b的口令为g00r002b，其需要被创建并被赋予数据库firstdb的完全访问权限：

```
mysql> GRANT ALL ON firstdb.* to guru2b@localhost
 IDENTIFIED BY 'g00r002b';
Query OK, 0 rows affected (0.01 sec)
mysql> exit
Bye
```

如果由管理员赋予权限

首先，管理员要作为根用户（或任何其他有权向新用户赋权的用户）先连接MySQL数据库：

```
% mysql -u root -p mysql
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15 to server version: 4.0.2-alpha-Max
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

接下来，管理员要创建firstdb数据库，后面将会处理它：

```
mysql> CREATE DATABASE firstdb;
Query OK, 1 row affected (0.01 sec)
```

最后，用户guru2b的口令是g00r002b，需要先创建他，然后赋予其对firstdb数据库的完全权限。注意，我们假设你是从localhost连接到数据库的（数据库客户端和数据库服务器在同一台机器上）。如果不是这样，管理员就要用正确的主机名替代localhost：

```
mysql> GRANT ALL ON firstdb.* to guru2b@localhost
 IDENTIFIED BY 'g00r002b';
Query OK, 0 rows affected (0.01 sec)
mysql> exit
Bye
```

guru2b是你的MySQL用户名，本书后面将一直用到它，g00r002b是你的口令。你也可以选择或被分配另外一个用户名。第14章介绍如何授权。

使用你的数据库

如果你是SQL或MySQL的初学者，这是一个好机会。我建议你按照顺序完成后面几个小节的练习。但是，只有在你超越本书并开始编写自己的查询的时候，才是真正的学习。所以，在学习的过程中要不断地实践，要尝试所能想到的各种情况。在这个阶段不要害怕犯错误！错误是也一种学习方式。数据是没有价值的。最好是现在让意外事故删除你的示例数据库，而不是在将来处理重要的数据库时发生这种意外。

我们首先创建示例数据库中的表，然后向表中添加数据。得到一些带有数据的表之后，将学习如何在这些表中执行查询。首先，用下列命令连接到新创建的表中：

```
% mysql -u guru2b -pg00r002b firstdb
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15 to server version: 4.0.2-alpha-Max

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

如果还没有正确地设置权限，会看到一条如下的错误消息：

```
ERROR 1044: Access denied for user: 'guru2be@localhost' to database 'firstdb'
```

如果是这样，你或者管理员需要返回前面两个小节所介绍的步骤。

所有这些权限的问题现在看起来比较麻烦，但它们却是一项非常有用的功能。有时候，你不能让任何人都访问你的数据，设置权限就是一种方法。

也可以不指定数据库而进行连接，如下所示：

```
% mysql -u guru2b -pg00r002b guru2b
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15 to server version: 4.0.2-alpha-Max

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

现在要确认你使用的是正确的数据库，需要让MySQL知道使用哪一个数据库。要确认你所做的任何工作都会影响正确的数据库，输入下面的黑体语句：

```
mysql> USE firstdb
Database changed
```

可以用两种方法连接数据库，在连接时指定数据库，或者连接后指定数据库。将来，如果系统中有多于一个的数据库要使用，你会发现，使用use语句在数据库之间切换非常容易。

创建一个表

连接到数据库之后，可以向它添加内容。为了方便使用，创建一个跟踪销售小组的数据库。我们知道，数据库包括多个表，创建一个包含销售代表数据的表。可以存储他们的姓名、雇员号和佣金。创建此表，要使用CREATE命令，但需要指定TABLE而不是DATABASE。输入如下的CREATE语句：

```
mysql> CREATE TABLE sales_rep(
  employee_number INT,
  surname VARCHAR(40),
  first_name VARCHAR(30),
  commission TINYINT
);
Query OK, 0 rows affected (0.00 sec)
```

警告： 不要忘记行末尾的分号。所有的MySQL命令必须以分号结束。对于初学者，会经常忘记它。还要注意，如果忘记了分号，又按了回车键，只需要在再一次按回车键之前添加分号。MySQL可以接受多行的命令。

不必像印刷的一样来输入语句；我经常用多行语句，以使其易读，但是在同一行输入它会使自己很方便。同时，如果使用了不同的形式，它也可以工作。在本书中，用大写字母代表MySQL关键字，小写字母代表你自己可以选择的名称。例如，如下输入：

```
mysql> create table SALES_REPRESENTATIVE(
  EMPLOYEE_NO int,
  SURNAME varchar(40),
  FIRST_NAME varchar(30),
  COMMISSION tinyint
);
```

没有问题。然而，如下输入：

```
mysql> CREATE TABLES sales_rep(
  employee_number INT,
  surname VARCHAR(40),
  first_name VARCHAR(30),
  commission TINYINT
);
```

会看到如下的错误消息：

```
ERROR 1064: You have an error in your SQL syntax near
 'TABLES sales_reps(employee_number INT,surname
 VARCHAR(40),first_name VARCHAR(30)  at line 1
```

这是因为错误地拼写了TABLE。要小心地正确输入大写的文本；你可以重命名小写字母的文本，没有任何问题（只要使用相同的名称）。

你也许会对出现在字段名后面的INT、VARCHAR和TINYINT术语产生疑问。它们就是数据类型和列类型。INT表示整数，通常在－2 147 483 648到2 147 483 647的范围内。这相当于地球人口的三分之一，所以无论销售小组发展到多大，它都可以覆盖。VARCHAR表示可变长字符。括号中的数字是字符串的最大长度。30个和40个字符可以分别满足人们的名和姓的要求。TINYINT表示小整数，通常在－128到127范围内。Commission字段是一个百分比值，不可能超过100%，所以用小整数就足够了。第2章“数据类型和表类型”会详细介绍各种列类型，以及什么时候使用它们。

用SHOW TABLES在数据库中列出现有的表

你已经有了一个表，就可以用SHOW TABLES来确认它的存在：

```
mysql> SHOW TABLES;
+-------------------+
| Tables_in_firstdb |
+-------------------+
| sales_rep         |
+-------------------+
1 row in set (0.00 sec)
```

SHOW TABLES列出当前数据库中所有的表。在前面创建的firstdb中，只有一个表：sales_rep。所以除非你确实忘记了，此命令不常使用。在具有许多表的大数据库中，有些表的名称确实可能会被忘记。或者，你遇到了一个新的数据库。在这些情况中，就要使用SHOW TABLES。

用DESCRIBE来检查表结构

DESCRIBE命令显示表的结构。要想查看MySQL已经正确创建了你的表，按下列语句输入：

```
mysql> DESCRIBE sales_rep;
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| employee_number | int(11)     | YES  |     | NULL    |       |
| surname         | varchar(40) | YES  |     | NULL    |       |
| first_name      | varchar(30) | YES  |     | NULL    |       |
| commission      | tinyint(4)  | YES  |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

这个表中有各种类型的列，你也许还不熟悉。现在，你就会对字段和类型有兴趣了。第2章将介绍其他的表头。字段名与你输入的完全一样，两个VARCHAR字段具有你所分配给它们的大小。注意，INT和TINYINT也分配了一个大小，尽管在创建它们的时候没有指定。记住，TINYINT在默认时的范围是－128到127（包括负号，有4个字符），INT的范围是－2 147 483 648到2 147 483 647（包括负号，有11个字符），所以实际上分配的大小是指显

示的宽度。

向表中添加新记录

现在已经有了一个表，要向其中放入一些数据。我们假设有三个销售代表（如表1.3所示）。

表1.3 销售代表

雇员代码	姓	名	代理百分比
1	Rive	Sol	10
2	Gordimer	Charlene	15
3	Serote	Mike	10

要想将这些数据放到表中，使用SQL语句INSERT来创建一个记录，如下所示：

```
mysql> INSERT INTO sales_rep(employee_number,surname,first_name,commission)
    VALUES(1,'Rive','Sol',10);
mysql> INSERT INTO sales_rep(employee_number,surname,first_name,commission)
    VALUES(2,'Gordimer','Charlene',15);
mysql> INSERT INTO sales_rep(employee_number,surname,first_name,commission)
    VALUES(3,'Serote','Mike',10);
```

说明： 字符串字段（VARCHAR字符字段）的值要用单引号括起来，但数字字段（commission和employee_number）不需要。确认你已经在单引号中放入了正确的字段值，并且每个单引号之间要匹配（有一个前引号，就要有一个后引号），这些是SQL初学者容易忽略的。

还有一个便捷的INSERT语句，用于输入数据。你可能用过下列语句：

```
mysql> INSERT INTO sales_rep VALUES(1,'Rive','Sol',10);
mysql> INSERT INTO sales_rep VALUES(2,'Gordimer','Charlene',15);
mysql> INSERT INTO sales_rep VALUES(3,'Serote','Mike',10);
```

以这种方式输入命令，必须以数据库中定义的顺序来输入字段。下列语句是不正确的：

```
mysql> INSERT INTO sales_rep VALUES(1,'Sol','Rive',10);
Query OK, 1 row affected (0.00 sec)
```

这看上去是正确的，但数据是以错误的顺序输入的，Sol成为了姓，而Rive成为了名。

提示： 我建议最好现在就形成使用完整INSERT语句的习惯，特别是如果你想通过编程语言运行查询的话。首先，它降低了出错的机会（你不能由语句看出名和姓的顺序是否错了）；第二，这可以使程序更加灵活。第5章“MySQL编程”详细介绍了这个问题。

在一个INSERT语句中添加数据

你也许还用过另一个便捷方法，就是在一个INSERT语句中输入所有的数据，每一条记录用逗号分隔，如下所示：

```
mysql> INSERT INTO sales_rep (employee_number,surname,first_name,commission)
VALUES
(1,'Rive','Sol',10),
 (2,'Gordimer','Charlene',15),
(3,'Serote','Mike',10);
Query OK, 3 rows affected (0.05 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

这种方法避免了大量的输入，服务器也能够较快地处理它。

用LOAD DATA从文本文件插入大量数据

插入数据的最后一种方法是用LOAD DATA语句，如果要将大量数据一次插入，这也是最好的方法，如下所示：

```
mysql> LOAD DATA LOCAL INFILE "sales_rep.sql" INTO TABLE sales_rep;
```

数据文件的格式一定要正确，没有异常。在这种情况下，如果使用默认设置，文本文件中的每条记录占一行，每个字段由TAB键来分隔。假设\t字符表示TAB键，每一行以换行符结束，那么文本文件应当如下所示：

```
1\tRive\tSol\t10
2\tGordimer\tCharlene\t15
3\tSerote\tMike\t10
```

这种应用也可以有所变化，用于存储备份，将在第11章“数据库备份”中讨论。这甚至比多记录INSERT语句更加有效率。LOCAL关键字指出了客户机找到文件的那台服务器（你所连接的那台机器）。如果将它省略，意味着MySQL在数据库服务器上寻找文件。默认情况下，LOAD DATA假设：值的顺序与表中定义的顺序一致，每个字段由TAB键分隔，每条记录以换行符分隔。

从表中检索信息

在MySQL中，很容易从表中获取信息。你可以使用强大的SELECT语句，例如：

```
mysql> SELECT commission FROM sales_rep WHERE surname='Gordimer';
+------------+
| commission |
+------------+
|         15 |
+------------+
1 row in set (0.01 sec)
```

SELECT语句有几个部分。SELECT后面的第一部分是字段列表。你可能已经返回了一些其他字段，来代替代理字段，如下所示：

```
mysql> SELECT commission,employee_number FROM
 sales_rep WHERE surname='Gordimer';
+------------+-----------------+
```

```
| commission | employee_number |
+------------+-----------------+
|         15 |               2 |
+------------+-----------------+
1 row in set (0.00 sec)
```

你也可能使用通配符（*）来返回所有的字段，如下所示：

```
mysql> SELECT * FROM sales_rep WHERE surname='Gordimer';
+-----------------+----------+------------+------------+
| employee_number | surname  | first_name | commission |
+-----------------+----------+------------+------------+
|               2 | Gordimer | Charlene   |         15 |
+-----------------+----------+------------+------------+
1 row in set (0.00 sec)
```

该通配符表示表中的所有字段。在前面的例子中，所有这四个字段以它们在表结构中的顺序返回了。

在SELECT语句中，WHERE后面的部分叫做WHERE从句。从句是非常灵活的，可以包含许多不同的条件。看下面的例子：

```
mysql> SELECT * FROM sales_rep WHERE commission>10
 OR surname='Rive' AND first_name='Sol';
+-----------------+----------+------------+------------+
| employee_number | surname  | first_name | commission |
+-----------------+----------+------------+------------+
|               1 | Rive     | Sol        |         10 |
|               2 | Gordimer | Charlene   |         15 |
+-----------------+----------+------------+------------+
2 rows in set (0.00 sec)
```

理解AND和OR是正确使用SQL的基础。只有AND关键字两边的条件都为真，结果才为真。只要OR语句中的条件有一个为真，结果就为真。表1.4显示了AND/OR的真值表。

表1.4 AND/OR真值表

关键字	气温高	湿度大	结果
AND	真	真	真，气温高且湿度大
AND	真	假	假，因为湿度不大，所以不能说是气温高且湿度大
AND	假	真	假，因为气温不高，所以不能说是气温高且湿度大
AND	假	假	假，因为两项都不是真的，所以不能说是气温高且湿度大
OR	真	真	真，两项都是真的，所以气温高或湿度大
OR	真	假	真，气温高
OR	假	真	真，湿度大
OR	假	假	假，两项都不是真的

MySQL处理条件的顺序

下面的例子示范了一个圈套，许多人都会落入该圈套。我们假设，一位经理让你提供一份清单，其中包括姓为Rive且名为Sol或者代理任务超过10%的所有雇员。你很可能要尝试如下的查询：

```
mysql> SELECT * FROM sales_rep WHERE surname='Rive'
 AND first_name='Sol' OR commission>10;
+-----------------+----------+------------+------------+
| employee_number | surname  | first_name | commission |
+-----------------+----------+------------+------------+
|               1 | Rive     | Sol        |         10 |
|               2 | Gordimer | Charlene   |         15 |
+-----------------+----------+------------+------------+
2 rows in set (0.00 sec)
```

这个结果就是你想要的。但是如果该经理的意思稍微有一点点不同呢？其中的雇员必须是姓Rive，然后可以名为Sol或者代理任务超过10%。第二个结果就不对了，因为尽管该雇员的代理任务超过10%，他的名字却不是Sol。AND结构的意思是，两个条件必须都为真。你必须使用如下的查询：

```
mysql> SELECT * FROM sales_rep WHERE surname='Rive'
 AND (first_name='Sol' OR commission>10);
+-----------------+---------+------------+------------+
| employee_number | surname | first_name | commission |
+-----------------+---------+------------+------------+
|               1 | Rive    | Sol        |         10 |
+-----------------+---------+------------+------------+
1 row in set (0.00 sec)
```

注意查询中的圆括号。如果有多个条件，就一定要知道处理条件的顺序。先执行OR部分，还是先执行AND部分呢？通常，你得到的是口头的模糊指令，但这个例子表明了，在实现查询之前必须要先把要求弄清楚。有时，这种错误永远不会被发现。这就是所谓的计算机错误，但它最终会落到某个人身上，通常会是设计出错误查询的人。

附录B“MySQL函数和运算符索引”列出了运算符及其过程。我建议读者使用圆括号来确定查询中的过程。有些人发誓要记住所有的内置优先级。例如，我们在学校里学过1+1*3=4，而不等于6，这是因为乘法运算要在加法运算的前面执行。同样，AND运算要在OR运算的前面执行。但并不是每个人都知道这个顺序，所以要使用括号，让每个人都了解到1+（1*3）的含义。我已经从事了许多年的编程工作，仍然无法知道所有的运算符优先级，也许这本来就是不可能的。

模式匹配：LIKE和%

让我们更多地了解SELECT语句。如果想返回Mike Serote的详细资料，该怎么办呢？你会说，这很简单，使用下面的查询：

```
mysql> SELECT * FROM sales_rep WHERE surname='Serote' and first_name='Mike';
+-----------------+---------+------------+------------+
| employee_number | surname | first_name | commission |
+-----------------+---------+------------+------------+
|               3 | Serote  | Mike       |         10 |
+-----------------+---------+------------+------------+
1 row in set (0.00 sec)
```

但是，如果你忘记了如何拼写Serote，该怎么办呢？是Serotte，还是Serota？你不得不尝试多次查询，才能成功，并且如果你碰巧记不起正确的拼写，就永远不会成功。你也许想试着只使用Mike，但要记住，许多数据库都包含成千上万的记录。很幸运，我们有一个更好的方法。MySQL允许使用LIKE语句。如果你还记得，姓是以Sero开头的，就可以使用下面的语句：

```
mysql> SELECT * FROM sales_rep WHERE surname LIKE 'Sero%';
+-----------------+---------+------------+------------+
| employee_number | surname | first_name | commission |
+-----------------+---------+------------+------------+
|               3 | Serote  | Mike       |         10 |
+-----------------+---------+------------+------------+
1 row in set (0.00 sec)
```

注意其中的%。它是一个通配符，类似于*，但是只能用于SELECT条件中。它的含义是：0个或多个字符。所以会返回你考虑到的所有可能的拼写。通配符可以使用任意多次，查询可以像下面这样：

```
mysql> SELECT * FROM sales_rep WHERE surname LIKE '%e%';
+-----------------+----------+------------+------------+
| employee_number | surname  | first_name | commission |
+-----------------+----------+------------+------------+
|               1 | Rive     | Sol        |         10 |
|               2 | Gordimer | Charlene   |         15 |
|               3 | Serote   | Mike       |         10 |
+-----------------+----------+------------+------------+
3 rows in set (0.00 sec)
```

这将返回所有的记录，因为它寻找所有包括字母e的姓。这与下面的查询不同，下面的查询只寻找以字母e开头的姓：

```
mysql> SELECT * FROM sales_rep WHERE surname LIKE 'e%';
Empty set (0.00 sec)
```

你也可以使用如下所示的查询，它寻找所有符合这样条件的姓：在任何位置有一个字母e，并且以字母e结束。

```
mysql> SELECT * FROM sales_rep WHERE surname LIKE '%e%e';
+-----------------+---------+------------+------------+
| employee_number | surname | first_name | commission |
```

```
+-----------------+---------+-----------+------------+
|               3 | Serote  | Mike      |         10 |
+-----------------+---------+-----------+------------+
1 row in set (0.00 sec)
```

我们再向表中添加几条记录，这样可以尝试一些更复杂的查询。添加如下两条记录：

```
mysql> INSERT INTO sales_rep values(4,'Rive','Mongane',10);
mysql> INSERT INTO sales_rep values(5,'Smith','Mike',12);
```

分类

另一个有用且常用的从句允许将结果**分**类。一份按字母顺序列出的雇员表是十分有用的，你可以使用**ORDER BY**从句来生成该表：

```
mysql> SELECT * FROM sales_rep ORDER BY surname;
+-----------------+----------+-----------+------------+
| employee_number | surname  | first_name | commission |
+-----------------+----------+-----------+------------+
|               2 | Gordimer | Charlene  |         15 |
|               1 | Rive     | Sol       |         10 |
|               4 | Rive     | Mongane   |         10 |
|               3 | Serote   | Mike      |         10 |
|               5 | Smith    | Mike      |         12 |
+-----------------+----------+-----------+------------+
5 rows in set (0.00 sec)
```

如果想按照姓名分类，你也许已经注意到此列表并不完全正确，因为Sol Rive出现在了Mongane Rive的前面。要想纠正此问题，需要在姓一样的时候按照名**分**类。为此，使用如下语句：

```
mysql> SELECT * FROM sales_rep ORDER BY surname,first_name;
+-----------------+----------+-----------+------------+
| employee_number | surname  | first_name | commission |
+-----------------+----------+-----------+------------+
|               2 | Gordimer | Charlene  |         15 |
|               4 | Rive     | Mongane   |         10 |
|               1 | Rive     | Sol       |         10 |
|               3 | Serote   | Mike      |         10 |
|               5 | Smith    | Mike      |         12 |
+-----------------+----------+-----------+------------+
5 rows in set (0.00 sec)
```

现在的顺序是正确的。要想以相反的顺序分类（递减的顺序），使用DESC关键字。下面的查询返回所有记录，以挣得佣金的多少进行排序，从高到低：

```
mysql> SELECT * FROM sales_rep ORDER BY commission DESC;
+-----------------+----------+-----------+------------+
```

```
| employee_number | surname  | first_name | commission |
+-----------------+----------+------------+------------+
|               2 | Gordimer | Charlene   |         15 |
|               5 | Smith    | Mike       |         12 |
|               1 | Rive     | Sol        |         10 |
|               4 | Rive     | Mongane    |         10 |
|               3 | Serote   | Mike       |         10 |
+-----------------+----------+------------+------------+
5 rows in set (0.00 sec)
```

同时，你也许需要进一步分类挣得10%佣金的三位雇员。为此，可以使用ASC关键字。这是默认的分类顺序，所以该关键字不是严格必需的，但它可以使查询更加清晰：

```
mysql> SELECT * FROM sales_rep ORDER BY commission DESC,
 surname ASC,first_name ASC;
+-----------------+----------+------------+------------+
| employee_number | surname  | first_name | commission |
+-----------------+----------+------------+------------+
|               2 | Gordimer | Charlene   |         15 |
|               5 | Smith    | Mike       |         12 |
|               4 | Rive     | Mongane    |         10 |
|               1 | Rive     | Sol        |         10 |
|               3 | Serote   | Mike       |         10 |
+-----------------+----------+------------+------------+
5 rows in set (0.01 sec)
```

限制结果的数量

迄今为止，你总是得到满足条件的所有结果。然而，在现实的数据库中也许包括成千上万条记录，你不想一次看到所有的记录。因此，MySQL允许使用LIMIT从句。LIMIT是非标准的SQL，也就是说，你不能对所有其他的数据库使用它，但它对于MySQL是一个强有力的增强。如果你只想找到佣金最高的雇员（像此例一样，假设只有一个），可以运行此查询：

```
mysql> SELECT first_name,surname,commission FROM sales_rep
 ORDER BY commission DESC;
+------------+----------+------------+
| first_name | surname  | commission |
+------------+----------+------------+
| Charlene   | Gordimer |         15 |
| Mike       | Smith    |         12 |
| Sol        | Rive     |         10 |
| Mike       | Serote   |         10 |
| Mongane    | Rive     |         10 |
+------------+----------+------------+
5 rows in set (0.00 sec)
```

很清楚，你要找的雇员是Charlene Gordimer。但是，LIMIT允许你只返回该记录，如下所示：

```
mysql> SELECT first_name,surname,commission FROM sales_rep
 ORDER BY commission DESC LIMIT 1;
+------------+----------+------------+
| first_name | surname  | commission |
+------------+----------+------------+
| Charlene   | Gordimer |         15 |
+------------+----------+------------+
1 row in set (0.00 sec)
```

如果LIMIT从句的后面只有一个数字，它就决定了返回的行数。

说明： LIMIT 0不返回任何记录。这好像没有什么用处，但是在大数据库上测试查询而不实际运行查询时，它是一个有用的方法。

LIMIT从句并不是只允许返回数据集开始或结束处的有限数量的记录。也可以告诉MySQL一个偏移量，也就是说，从哪个结果记录开始。如果LIMIT从句的后面有两个数字，第一个就是偏移量，第二个是所限制的行数。下面的例子按递减的顺序返回第二条记录：

```
mysql> SELECT first_name,surname,commission FROM
 sales_rep ORDER BY commission DESC LIMIT 1,1;
+------------+---------+------------+
| first_name | surname | commission |
+------------+---------+------------+
| Mike       | Smith   |         12 |
+------------+---------+------------+
1 row in set (0.00 sec)
```

默认的偏移量是0（计算机通常是从0开始计数），所以如果指定偏移量是1，就是从第二条记录开始。运行下面的查询，可以验证这一点：

```
mysql> SELECT first_name,surname,commission FROM sales_rep
 ORDER BY commission DESC LIMIT 0,1;
+------------+----------+------------+
| first_name | surname  | commission |
+------------+----------+------------+
| Charlene   | Gordimer |         15 |
+------------+----------+------------+
1 row in set (0.00 sec)
```

LIMIT 1与LIMIT 0, 1是一样的，因为如果不指定的话，就假设是默认值0。

现在来看另一个例子。如何以佣金字段的降序，来返回第三、第四和第五个记录？

```
mysql> SELECT first_name,surname,commission FROM sales_rep
 ORDER BY commission DESC LIMIT 2,3;
+------------+----------+------------+
```

```
| first_name | surname | commission |
+------------+---------+------------+
| Sol        | Rive    |         10 |
| Mike       | Serote  |         10 |
| Mongane    | Rive    |         10 |
+------------+---------+------------+
3 rows in set (0.00 sec)
```

2是偏移量（记住，偏移量是从0开始，所以2是实际的第三个记录），3是返回的记录数。

说明： LIMIT通常用于运行MySQL的搜索引擎，例如，每页显示10条结果。第一页的结果使用LIMIT 0, 10，第二页的结果使用LIMIT 10, 10，依此类推。

用MAX()返回最大值

MySQL有许多函数，用于调整查询。如果你碰巧已经看到了附录B，不要就认为必须得学习那么多页的术语。没有人能记住所有的术语，但只要你知道它们的存在，就可以查阅是否有完成特定任务的术语。一旦用得多了，就会发现你已经记住了大多数的常用函数。这比起记住所有函数来，要省力得多。首先要学习的函数是MAX()。要想返回最高的佣金，使用此查询：

```
mysql> SELECT MAX(commission) from sales_rep;
+-----------------+
| MAX(commission) |
+-----------------+
|              15 |
+-----------------+
1 row in set (0.00 sec)
```

在使用函数时，注意其中的括号。函数是应用于括号中的参数的。在本书中，我写的函数都带括号，以提醒读者，它是一个函数以及如何使用它。例如，MAX()。

警告： 小心查询中的空格。在大多数情况下，它们不会有什么影响；但是在处理函数的时候（函数通常由括号中的参数来识别），需要特别小心。如果在单词COUNT后面加上一个空格，就会得到一条MySQL语法错误。

返回惟一的记录

有些时候，你不想返回重复的结果。看下面的查询：

```
mysql> SELECT surname FROM sales_rep ORDER BY surname;
+----------+
| surname  |
+----------+
| Gordimer |
| Rive     |
| Rive     |
```

```
| Serote    |
| Smith     |
+-----------+
5 rows in set (0.00 sec)
```

此查询没有错误，但如果你只希望返回姓的清单，每个姓只出现一次呢？你不想让Rive为Mongane和Sol都出现，只需要出现一次。解决的办法是用DISTINCT，如下所示：

```
mysql> SELECT DISTINCT surname FROM sales_rep ORDER BY surname;
+-----------+
| surname   |
+-----------+
| Gordimer  |
| Rive      |
| Serote    |
| Smith     |
+-----------+
4 rows in set (0.00 sec)
```

计数

在查询结果的下面，我们可以看到，MySQL显示了返回的行数，如4 rows in set。有时，你实际上只需要返回结果的数目，而不是记录的内容本身。在这种情况下，使用COUNT()函数：

```
mysql> SELECT COUNT(surname) FROM sales_rep;
+-----------------+
| COUNT(surname)  |
+-----------------+
|               5 |
+-----------------+
1 row in set (0.01 sec)
```

这与前面的例子没有什么区别，因为表中姓的数目与名的数目是一样的。使用下面的查询可以得到同样的结果：

```
mysql> SELECT COUNT(*) FROM sales_rep;
+----------+
| COUNT(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
```

要想计数不重复的姓的数目，将COUNT()与DISTINCT结合使用，如下：

```
mysql> SELECT COUNT(DISTINCT surname) FROM sales_rep;
+------------------------+
```

```
| COUNT(DISTINCT surname) |
+-------------------------+
|                       4 |
+-------------------------+
1 row in set (0.00 sec)
```

用AVG()、MIN()和SUM()返回平均数、最小值和总数

这些函数与MAX()的工作方式是一样的。将要处理的字段放到括号中，这样，使用下面的查询就可以返回平均佣金数：

```
mysql> SELECT AVG(commission) FROM sales_rep;
+-----------------+
| AVG(commission) |
+-----------------+
|         11.4000 |
+-----------------+
1 row in set (0.00 sec)
```

要想找到销售人员挣得的最少的佣金数，使用下面的查询：

```
mysql> SELECT MIN(commission) FROM sales_rep;
+-----------------+
| MIN(commission) |
+-----------------+
|              10 |
+-----------------+
1 row in set (0.00 sec)
```

SUM()的工作方式也是一样的。在这里计算佣金的总数是不可能的，但可以对其工作方式有所理解：

```
mysql> SELECT SUM(commission) from sales_rep;
+-----------------+
| SUM(commission) |
+-----------------+
|              57 |
+-----------------+
1 row in set (0.00 sec)
```

在查询中进行计算

SQL允许在查询中执行计算。看下面的简单例子：

```
mysql> SELECT 1+1;
+-----+
| 1+1 |
+-----+
|   2 |
```

```
+-----+
1 row in set (0.00 sec)
```

显然，这不是你使用MySQL的原因。没有任何一所学校让学生在数学考试中使用MySQL。但它在查询中确实是一个有用的功能。例如，将每个人的佣金增加1%，如果再想看到销售人员的佣金数，使用下面的查询：

```
mysql> SELECT first_name,surname,commission + 1 FROM sales_rep;
+------------+----------+----------------+
| first_name | surname  | commission + 1 |
+------------+----------+----------------+
| Sol        | Rive     |             11 |
| Charlene   | Gordimer |             16 |
| Mike       | Serote   |             11 |
| Mongane    | Rive     |             11 |
| Mike       | Smith    |             12 |
+------------+----------+----------------+
5 rows in set (0.00 sec)
```

删除记录

MySQL使用DELETE语句删除记录。类似于SELECT，由于要删除整个记录，所以不需要指定任何的列。只需要表名和条件。例如，Mike Smith辞职了，要删除他，使用下面的查询：

```
mysql> DELETE FROM sales_rep WHERE employee_number = 5;
Query OK, 1 row affected (0.00 sec)
```

你也可以使用名和姓作为删除的条件，在这种情况下，查询也可以工作。但是，对于现实中的数据库，最好使用惟一的字段来识别正确的人选。第3章将介绍索引，但现在要记住employee_number是惟一的字段，最好使用它（Mike Smith是个比较常见的名字）。在专门介绍索引的章节中，你将认识到，将employee_number作为惟一的字段是很应该的。

警告： 在DELETE语句中使用条件时，要十分小心。简单地输入DELETE FROM sales_rep；这会将表中的所有记录删除。此处没有撤消选项，所以在学习备份数据之前，这样做会带来一些麻烦。

在表中更改记录

我们已经学习了用INSERT添加记录，用DELETE删除记录，用SELECT返回结果。现在只剩下学习如何改变现有记录中的数据了。我们假设，Sol Rive刚刚向纳米比亚沙漠中的居民卖了一大船的沙子，他的佣金增长到12%。要想正确地反映出这个事实，使用UPDATE语句，如下所示：

```
mysql> UPDATE sales_rep SET commission = 12 WHERE
 employee_number=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

警告：再一次强调，小心地使用条件。没有WHERE从句，就可能将每个人的佣金改成12%。

INSERT、SELECT、UPDATE和DELETE组成了处理数据的四个标准语句。它们是SQL数据处理语言（Data Manipulation Language，DML）的一部分。用它们，你可以按自己的需要改变记录中的数据。第2章将介绍更高级的查询。

删除表和数据库

还有定义数据结构的语句，是SQL数据定义语言（Data Definition Language，DDL）的一部分。你已经看到过一个了，就是CREATE语句，用于创建数据库、表和数据库的结构。与数据处理一样，你也可能要删除或改变这些表。我们创建一个表，然后再删除它：

```
mysql> CREATE TABLE commission (id INT);
Query OK, 0 rows affected (0.01 sec)
mysql> DROP TABLE commission;
Query OK, 0 rows affected (0.00 sec)d
```

警告：没有警告，就不会得到重视。表及其中的所有数据已经被删除了，一定要小心使用这个语句。

也可以对数据库执行此操作：

```
mysql> CREATE DATABASE shortlived;
Query OK, 1 row affected (0.01 sec)
mysql> DROP DATABASE shortlived;
Query OK, 0 rows affected (0.00 sec)
```

现在你已经了解，为什么授权是如此重要。为可以连接到MySQL的每个人授予此权利，是很危险的。第14章将学习如何使用权限来避免这种灾难。

改变表结构

最后一个DDL语句是ALTER，允许改变表的结构。你可以添加列、改变列的定义、重命名表和删除列。

添加列

假设你要在sales_reps表中创建一个列，用于存储该员工加入公司的日期。UPDATE不能用于此处，因为它只改变数据，而不改变结构。做此更改，使用ALTER语句：

```
mysql> ALTER TABLE sales_rep ADD date_joined DATE;
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

提示：DATE是一个列类型，以年-月-日的格式存储数据（YYYY-MM-DD）。如果你习惯于使用其他的日期格式，如美国格式（MM/DD/YYYY），需要做一些调整。

你的经理对数据库提出了另一个要求（尽管所有的更改都很容易执行，但还是最好从一开始就做好正确的设计，因为有些改变会产生令人不快的结果。第9章“数据库设计”介绍了数据库设计的内容）。你需要存储销售代表出生的年份，以分析该员工的年龄。MySQL有一个YEAR列类型，可以使用。添加如下的列：

```
mysql> ALTER TABLE sales_rep ADD year_born YEAR;
Query OK, 4 rows affected (0.02 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

更改列定义

添加完出生年份，你的经理又有了一个更好的主意。为什么不存储销售代表的出生日期呢？这样，年份和前面一样，但公司可以在员工生日的时候给他一个小小的惊喜。要改变列定义，使用如下查询：

```
mysql> ALTER TABLE sales_rep CHANGE year_born birthday
 DATE;
Query OK, 4 rows affected (0.03 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

CHANGE从句的后面是旧的列名称，然后是新的列名称及其定义。要改变定义，而不改变列名，只需简单地使名称与前面一样，如下所示：

```
mysql> ALTER TABLE tablename CHANGE oldname oldname new_column_definition;
```

也可以使用MODIFY从句，它不要求重复名称，如下所示：

```
mysql> ALTER TABLE tablename MODIFY oldname new_column_definition;
```

为表重新命名

一天早晨，你的经理又有了新的要求，认为销售代表这个术语不应该再使用。今后，雇员就是现金流的增加者，记录需要被存储为增加值（enhancement value）。经理的视野很开阔，你决定满足他的要求，首先要添加新的字段：

```
mysql> ALTER TABLE sales_rep ADD enhancement_value int;
Query OK, 4 rows affected (0.05 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

然后为表进行重命名。为此，要在ALTER语句中使用RENAME，如下所示：

```
mysql> ALTER TABLE sales_rep RENAME cash_flow_specialist;
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

第二天，经理看上去有点不那么自信，认为这种改变可能不太合适。你又决定在别人看到之前，将表名恢复原样，并删除新列。

```
mysql> ALTER TABLE cash_flow_specialist RENAME TO
 sales_rep;
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

说明： 注意两个ALTER RENAME语句的区别：第二个RENAME语句中包含了TO。然而，这两个语句在功能上是一致的。这种情况在MySQL中有不少，用不只一种方法执行任务。事实上，还有第三种方法来为表进行重命名，可以使用RENAME old tablename TO new_tablename。这些功能用于与其他数据库兼容，或者符合ANSI SQL标准。

删除列

要想删除不想要的列enhancement_value（是什么使我们认为这个字段应该是INT呢？），使用ALTER... DROP，如下：

```
mysql> ALTER TABLE sales_rep DROP enhancement_value;
Query OK, 4 rows affected (0.06 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

使用日期函数

现在已经有了两个日期列，我们来看MySQL的日期函数。现在，表的结构如下所示：

```
mysql> DESCRIBE sales_rep;
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| employee_number | int(11)     | YES  |     | NULL    |       |
| surname         | varchar(40) | YES  |     | NULL    |       |
| first_name      | varchar(30) | YES  |     | NULL    |       |
| commission      | tinyint(4)  | YES  |     | NULL    |       |
| date_joined     | date        | YES  |     | NULL    |       |
| birthday        | date        | YES  |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

如果执行一个查询，返回date_joined和birthday，得到如下结果：

```
mysql> SELECT date_joined,birthday FROM sales_rep;
+-------------+----------+
| date_joined | birthday |
+-------------+----------+
| NULL        | NULL     |
| NULL        | NULL     |
| NULL        | NULL     |
| NULL        | NULL     |
+-------------+----------+
4 rows in set (0.00 sec)
```

NULL值表明，你还没有向这些字段输入任何值。你会注意到，在DESCRIBE（说明）一个表时有Null表头。YES是默认值，表示该字段中可以没有任何值。有时，要指定某个字段永远不能是NULL值。第3章将介绍如何这样做。NULL值通常影响查询的结果，并且有其自身的一些注意事项，将在第2章和第3章介绍。为了确认没有NULL值，更新销售代表的记录：

```
mysql> UPDATE sales_rep SET date_joined =
 '2000-02-15', birthday='1976-03-18'
 WHERE employee_number=1;
mysql> UPDATE sales_rep SET date_joined =
 '1998-07-09', birthday='1958-11-30'
```

```
 WHERE employee_number=2;
mysql> UPDATE sales_rep SET date_joined =
 '2001-05-14', birthday='1971-06-18'
 WHERE employee_number=3;
mysql> UPDATE sales_rep SET date_joined =
 '2002-11-23', birthday='1982-01-04'
 WHERE employee_number=4;
```

有不少有用的日期函数，这些例子只显示了一部分；附录A和附录B有更多的信息。

指定日期格式

如果你想以自己选择的格式显示日期，而不是用标准的YYYY-MM-DD格式，那么要记住一些日期格式。要想以MM/DD/YYYY格式返回所有雇员的生日，使用DATE_FORMAT()函数，如下：

```
mysql> SELECT DATE_FORMAT(date_joined,'%m/%d/%Y')
 FROM sales_rep WHERE employee_number=1;
+-----------------------------------+
| date_format(date_joined,'%m/%d/%Y') |
+-----------------------------------+
| 02/15/2000                        |
+-----------------------------------+
```

date_joined后面的单引号内的部分叫做格式字符串。在格式字符串中，使用特定的字符串来指定要返回的格式。%m返回月份（01～12），%d返回日期（01～31），%Y返回四位数的年份。特定的字符串有许多；附录B有完整的清单。同时，看下面的例子：

```
mysql> SELECT DATE_FORMAT(date_joined,'%W %M %e %y')
 FROM sales_rep WHERE employee_number=1;
+--------------------------------------+
| DATE_FORMAT(date_joined,'%W %M %e %y') |
+--------------------------------------+
| Tuesday February 15 00               |
+--------------------------------------+
```

%W返回星期日期，%M返回月份的名字，%e返回日期（1～31），%y返回两位数的年份。注意，%d也返回日期（01～31），但与%e不一样，因为它包括了前面的零。

在下面的查询中，%a是缩写的星期日期，%D是带后缀的日期，%b是缩写的月份名字，%Y是四位数的年份：

```
mysql> SELECT DATE_FORMAT(date_joined,'%a %D %b, %Y')
 FROM sales_rep WHERE employee_number=1;
+----------------------------------------+
| DATE_FORMAT(date_joined,'%a %D %b, %Y') |
+----------------------------------------+
| Tue 15th Feb, 2000                     |
+----------------------------------------+
```

说明： 可以在格式字符串中加入自己的字符。前面的例子用到了一个斜线（/）和一个逗号（,）。你可以添加任何用于格式化日期的文本。

返回当前日期和时间

要想通过服务器知道当前的日期，使用CURRENT_DATE()函数。还有另外一个函数NOW()，它除了日期外还返回时间：

```
mysql> SELECT NOW(),CURRENT_DATE();
+---------------------+----------------+
| NOW()               | CURRENT_DATE() |
+---------------------+----------------+
| 2002-04-07 18:32:31 | 2002-04-07     |
+---------------------+----------------+
1 row in set (0.00 sec)
```

说明： NOW()返回日期和时间。有一个列类型叫做DATETIME，允许在表中以同样的格式（YYYY-MM-DD HH:MM:SS）存储数据。

你可以在返回它的时候，对birthday字段执行其他的转化。如果你担心因为已经用出生日期代替了年份字段而不能返回年份，那么可以使用YEAR()函数，如下所示：

```
mysql> SELECT YEAR(birthday) FROM  sales_rep;
+----------------+
| YEAR(birthday) |
+----------------+
|           1976 |
|           1958 |
|           1982 |
|           1971 |
+----------------+
4 rows in set (0.00 sec)
```

MySQL有一些其他的函数，用于返回日期的一部分：MONTH()和DAYOFMONTH()：

```
mysql> SELECT MONTH(birthday),DAYOFMONTH(birthday) FROM sales_rep;
+-----------------+----------------------+
| MONTH(birthday) | DAYOFMONTH(birthday) |
+-----------------+----------------------+
|               3 |                   18 |
|              11 |                   30 |
|               1 |                    4 |
|               6 |                   18 |
+-----------------+----------------------+
4 rows in set (0.00 sec)
```

创建高级查询

现在，你已经比较了解基本的查询了。在现实世界中，大多数查询都是很简单的，和前面的例子差不多。数据库设计得越好（参考第二部分“设计一个数据库”），查询就越简单。但在有的情况下，还需要更多的知识，最常见的就是将两个或多个表进行关联（这类查询称为关联）。

用AS给出新的列标题

前面的查询还没有容易到无法读或无法理解。我们通过按月份分类并返回销售代表的名字，来整理一下前面的查询。我们还会遇到用AS关键字来引入别名，也就是说，为列起另一个名字：

```
mysql> SELECT surname,first_name,MONTH(birthday)
 AS month,DAYOFMONTH(birthday) AS day FROM sales_rep
 ORDER BY month;
+----------+------------+-------+------+
| surname  | first_name | month | day  |
+----------+------------+-------+------+
| Rive     | Mongane    |     1 |    4 |
| Rive     | Sol        |     3 |   18 |
| Serote   | Mike       |     6 |   18 |
| Gordimer | Charlene   |    11 |   30 |
+----------+------------+-------+------+
4 rows in set (0.01 sec)
```

用CONCAT连接列

有时你要将人名显示为一个结果字段，而不是把姓和名的字段分开。可以将列的结果连接在一起，使用CONCAT()函数（表示连接的意思），如下：

```
mysql> SELECT CONCAT(first_name, ' ',surname)
 AS name,MONTH(birthday) AS month,DAYOFMONTH(birthday)
 AS day FROM sales_rep ORDER BY month;
+------------------+-------+------+
| name             | month | day  |
+------------------+-------+------+
| Mongane Rive     |     1 |    4 |
| Sol Rive         |     3 |   18 |
| Mike Serote      |     6 |   18 |
| Charlene Gordimer |    11 |   30 |
+------------------+-------+------+
4 rows in set (0.00 sec)
```

说明： 注意CONCAT()中的空格。与日期指定字符串一样，可以用任何字符来格式化CONCAT()。

找到一年中的一天

要想知道Sol Rive是在该年中的哪一天（1～366）加入公司的，使用下面的查询：

```
mysql> SELECT DAYOFYEAR(date_joined) FROM sales_rep
 WHERE employee_number=1;
+-------------------------+
| DAYOFYEAR(date_joined) |
+-------------------------+
|                      46 |
+-------------------------+
```

处理多个表

关系数据库真正的力量来自于多个表之间的关系。到现在为止，我们一直在处理一个表，以熟悉SQL的语法。但在现实世界的应用中，都有多于一个的表，所以你需要知道如何处理这些情况。首先，我们向数据库中添加两个新表。表1.5包含客户数据（客户ID、名和姓），表1.6包含销售数据（客户ID、销售代表ID、销售额和惟一的销售代码）。

表1.5 客户表

ID	FIRST_NAME	SURNAME
1	Yvonne	Clegg
2	Johnny	Chaka-Chaka
3	Winston	Powers
4	Patricia	Mankunku

表1.6 销售表

CODE	SALES_REP	CUSTOMER	VALUE
1	1	1	2000
2	4	3	250
3	2	3	500
4	1	4	450
5	3	1	3800
6	1	2	500

你能创建这些表吗？这里是我使用的语句：

```
mysql> CREATE TABLE customer(
  id int,
  first_name varchar(30),
  surname varchar(40)
);
Query OK, 0 rows affected (0.00 sec)
mysql> CREATE TABLE sales(
```

```
  code int,
  sales_rep int,
  customer int,
  value int
);
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO customer(id,first_name,surname) VALUES
(1,'Yvonne','Clegg'),
(2,'Johnny','Chaka-Chaka'),
(3,'Winston','Powers'),
(4,'Patricia','Mankunku');
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0
mysql> INSERT INTO sales(code,sales_rep,customer,value) VALUES
(1,1,1,2000),
(2,4,3,250),
(3,2,3,500),
(4,1,4,450),
(5,3,1,3800),
(6,1,2,500);
Query OK, 6 rows affected (0.00 sec)
Records: 6  Duplicates: 0  Warnings: 0
```

连接两个或多个表

可以看到，这里使用了销售代表的雇员号和销售表中的客户ID。如果检查第一个销售记录，可以看到它是sales_rep 1的，查看sales_rep表，他是Sol Rive。查看两表关系的手工过程与MySQL的执行过程一样，只需要你告诉它使用什么关系。我们写一个查询，返回第一条销售记录的所有信息，以及销售代表的名字：

```
mysql> SELECT sales_rep,customer,value,first_name,surname
 FROM sales,sales_rep WHERE code=1 AND
 sales_rep.employee_number=sales.sales_rep;
+-----------+----------+-------+------------+---------+
| sales_rep | customer | value | first_name | surname |
+-----------+----------+-------+------------+---------+
|         1 |        1 |  2000 | Sol        | Rive    |
+-----------+----------+-------+------------+---------+
1 row in set (0.00 sec)
```

该查询中，SELECT后面的第一部分列出了要返回的字段。这很容易，只要列出来自于两个表的想要的字段。

FROM后面的第二部分告诉MySQL使用哪些表。在此例中，是两个表：sales表和sales_rep表。

WHERE后面的第三部分包含了条件code=1，返回销售表的第一条记录。后面的部分使这个查询成为关联。此时，是在告诉MySQL，要关联哪些字段，或者表之间的关系存在于哪些字段上。Sales表和sales_rep表之间的关系在sales_rep的employee_number字段和sales表的sales_rep字段之间。所以，因为你在sales_rep字段发现了1，所以必须在sales_rep表中寻找employee_number 1。

再试一下。这次，要返回Sol Rive（employee_number 1）的所有销售记录。我们来看这个查询背后的过程：

- 你需要哪些表？显然是sales_rep和sales。它们已经是查询的一部分：FROM sales_rep, sales。
- 需要哪些字段？你需要所有的销售信息。所以，字段清单是SELECT code, customer, value。
- 最后，你的条件是什么？第一个是你只需要Sol Rive的结果，第二个是指定关系，该关系在sales表的sales_rep和sales_rep表的employee_number字段之间。所以，条件是这样的：WHERE first_name='Sol' and surname='Rive' AND sales.sales_rep = sales_rep.employee_number。

最终的查询应该是这样：

```
mysql> SELECT code,customer,value FROM sales_rep,sales
 WHERE first_name='Sol' AND surname='Rive' AND
 sales.sales_rep = sales_rep.employee_number;
+------+----------+-------+
| code | customer | value |
+------+----------+-------+
|    1 |        1 |  2000 |
|    4 |        4 |   450 |
|    6 |        2 |   500 |
+------+----------+-------+
3 rows in set (0.00 sec)
```

注意在关系的条件中使用的符号：sales.sales_rep或者sales_rep.employee_number。先指定表名，然后是一个句点，接着是字段名，使得查询更加清晰，当不同的表具有同样的字段名时，查询是强制性的。也可以在字段清单中使用这个符号。例如，前面的查询也可以写成：

```
mysql> SELECT code,customer,value FROM sales,
 sales_rep WHERE first_name='Sol' AND surname='Rive'
 AND sales_rep = employee_number;
+------+----------+-------+
| code | customer | value |
+------+----------+-------+
|    1 |        1 |  2000 |
|    4 |        4 |   450 |
|    6 |        2 |   500 |
+------+----------+-------+
3 rows in set (0.00 sec)
```

字段名的前面没有表名，是因为不同的表中没有同名的字段。或者也可以将它写成：

```
mysql> SELECT sales.code,sales.customer,sales.value
 FROM sales,sales_rep WHERE sales_rep.first_name='Sol'
 AND sales_rep.surname='Rive' AND sales.sales_rep =
 sales_rep.employee_number;
+------+----------+-------+
| code | customer | value |
+------+----------+-------+
|    1 |        1 |  2000 |
|    4 |        4 |   450 |
|    6 |        2 |   500 |
+------+----------+-------+
3 rows in set (0.00 sec)
```

这两个查询的结果是一样的。

在有同样的字段名时，要想声明发生了什么，将销售表中的sales_rep字段改变成employee_number。不要着急；后面还要将其改回来：

```
mysql> ALTER TABLE sales CHANGE sales_rep
 employee_number int;
Query OK, 6 rows affected (0.00 sec)
Records: 6  Duplicates: 0  Warnings: 0
```

我们再试一下关联，将名字更正，但没有使用逗点来指定表名：

```
mysql> SELECT code,customer,value FROM sales_rep,sales
 WHERE first_name='Sol' AND surname='Rive'AND employee_number = employee_number;
ERROR 1052: Column: 'employee_number' in where clause is ambiguous
```

通过读该查询，你可以看到它不是很清晰。所以，现在必须在每次涉及employee_number字段时都要使用表名：

```
mysql> SELECT code,customer,value FROM sales_rep,sales
 WHERE sales_rep.employee_number=1 AND sales_rep.employee_number =
 sales.employee_number;
+------+----------+-------+
| code | customer | value |
+------+----------+-------+
|    1 |        1 |  2000 |
|    4 |        4 |   450 |
|    6 |        2 |   500 |
+------+----------+-------+
3 rows in set (0.00 sec)
```

提示： 你可能已经使用了sales.employee_number，来代替WHERE从句中的sales_rep.employee_number，但是最好使用小一些的表，因为这会使MySQL执行更少的工作，即可返回查询。第4章将介绍优化查询。

我们把字段名改回去，然后再继续深入：

```
mysql> ALTER TABLE sales CHANGE employee_number sales_rep INT;
Query OK, 6 rows affected (0.00 sec)
Records: 6  Duplicates: 0  Warnings: 0
```

执行日期计算

执行日期计算是相对容易的。在下一小节，我们将根据生日计算出某个人的年龄，但是首先，我们将执行一个比较简单的计算。要计算出当前日期和出生日期之间的年数，使用两个函数YEAR()和NOW()：

```
mysql> SELECT YEAR(NOW()) - YEAR(birthday) FROM sales_rep;
+-------------------------------+
| YEAR(NOW()) - YEAR(birthday) |
+-------------------------------+
|                            26 |
|                            44 |
|                            31 |
|                            20 |
+-------------------------------+
4 rows in set (0.00 sec)
```

说明：也可以用CURRENT_DATE()来代替NOW()，将给出同样的结果。

前面的查询结果不返回年龄，只返回年份的差值。这里没有计算日期和月份。本节讲解年龄的计算；这对于初学者是比较麻烦的。不要担心。一旦实践了基本的查询，这类查询将会是很简单的。

你需要像前面那样减去年份，如果还没有过去一个整年，就再减去一年。如果某人出生于2001年12月10日，那么在2002年1月的时候，他不是一岁，要等到过了2002年12月10日。比较好的办法是，读取两个日期字段（当前日期和出生日期）中的MM-DD部分并对它们进行比较。如果当前的是大的，就是过去了一个整年，计算可以结束。如果当前的MM-DD部分小于出生日期的该部分，就表明还没有过去一整年，需要再减去一年。听起来有点麻烦，但MySQL使其变得简单，因为它将真表达式的值置为1，假表达式的值置为0：

```
mysql> SELECT YEAR(NOW()) > YEAR(birthday) FROM
 sales_rep WHERE employee_number=1;
+-------------------------------+
| YEAR(NOW()) > YEAR(birthday) |
+-------------------------------+
|                             1 |
+-------------------------------+
1 row in set (0.00 sec)
mysql> SELECT YEAR(NOW()) < YEAR(birthday) FROM
 sales_rep WHERE employee_number=1;
+-------------------------------+
```

```
| YEAR(NOW()) < YEAR(birthday) |
+------------------------------+
|                            0 |
+------------------------------+
1 row in set (0.00 sec)
```

当前的年份大于雇员1的出生年份。那是真的，值为1。当前年份不小于出生的年份。那是假的，值为0。

现在需要快速返回日期的MM-DD部分。快捷方式是使用RIGHT()字符串函数：

```
mysql> SELECT RIGHT(CURRENT_DATE,5),RIGHT(birthday,5) FROM sales_rep;
+----------------------+-------------------+
| RIGHT(CURRENT_DATE,5) | RIGHT(birthday,5) |
+----------------------+-------------------+
| 04-06                | 03-18             |
| 04-06                | 11-30             |
| 04-06                | 01-04             |
| 04-06                | 06-18             |
+----------------------+-------------------+
4 rows in set (0.00 sec)
```

RIGHT()函数中的5是指从字符串右端算起的函数返回的字符数。完整的字符串是2002-04-06，最右边的5个字符是04-06（包括中间的符号）。所以，现在就有了执行日期计算的所有元素：

```
mysql> SELECT surname, first_name, (YEAR(CURRENT_DATE) -
 YEAR(birthday)) -(RIGHT(CURRENT_DATE,5)<RIGHT(birthday,5))
 AS age FROM sales_rep;
+----------+------------+------+
| surname  | first_name | age  |
+----------+------------+------+
| Rive     | Sol        |   26 |
| Gordimer | Charlene   |   43 |
| Rive     | Mongane    |   20 |
| Serote   | Mike       |   30 |
+----------+------------+------+
4 rows in set (0.00 sec)
```

你的结果可能与这些结果不完全匹配，因为我们的时间不一样，你的运行时间肯定比我的要晚。

警告： *在执行这种复杂计算时，一定要注意括号的匹配。对于每个前括号，都需要一个后括号，而且要在正确的位置。*

能不能找到在某一种情况下，前面的年龄查询不能执行？当当前年份与出生年份一样时，会得到－1作为答案。认真阅读附录，尝试用自己的方法计算年龄。有许多可能发生的情况，深入学习MySQL，开发一个AGE()函数。

在查询中分组

现在已经有了一个销售表，我们加入SUM()函数，就可以比前面更好地使用它，只需返回销售总值：

```
mysql> SELECT SUM(value) FROM sales;
+------------+
| SUM(value) |
+------------+
|       7500 |
+------------+
1 row in set (0.00 sec)
```

现在要得到每个销售代表的总销售额。要想手工完成此任务，需要通过sales_rep将销售表分组。先将第一个销售代表的所有销售额放到一起，计算总数，然后对第二个销售代表重复此操作。SQL有GROUP BY从句，MySQL也以同样的方式使用它：

```
mysql> SELECT sales_rep,SUM(value) FROM sales GROUP BY
 sales_rep;
+-----------+------------+
| sales_rep | SUM(value) |
+-----------+------------+
|         1 |       2950 |
|         2 |        500 |
|         3 |       3800 |
|         4 |        250 |
+-----------+------------+
4 rows in set (0.00 sec)
```

如果不用分组而进行同样的查询，就会得到一个错误消息：

```
mysql> SELECT sales_rep,SUM(value) FROM sales;
ERROR 1140: Mixing of GROUP columns
 (MIN(),MAX(),COUNT()...) with no GROUP columns
 is illegal if there is no GROUP BY clause
```

此查询没有任何作用，因为它试图将摘要字段SUM()与普通字段混合。你期待些什么呢？让所有值的总数对每个sales_rep进行重复？

也可以将分组查询的输出分类。要想从高到低返回每位销售代表的总销售额，添加ORDER BY从句：

```
mysql> SELECT sales_rep,SUM(value) AS sum FROM sales
 GROUP BY sales_rep ORDER BY sum desc;
+-----------+------+
| sales_rep | sum  |
+-----------+------+
|         3 | 3800 |
```

```
|          1 | 2950 |
|          2 |  500 |
|          4 |  250 |
+------------+------+
```

现在尝试一个更复杂的查询，它使用前面学过的一些概念。我们要返回销售额最少的销售代表的名字。首先，必须要返回一个雇员号。在运行查询时，会得到不同的数字，因为有三个人只有一次销售。现在返回哪一个是无关紧要的。像下面这样执行：

```
mysql> SELECT sales_rep,COUNT(*) as count from sales
 GROUP BY sales_rep ORDER BY count LIMIT 1;
+------------+-------+
| sales_rep | count |
+------------+-------+
|          4 |     1 |
+------------+-------+
1 row in set (0.00 sec)
```

还可以更进一步吗？可以再执行一个关联，返回销售代表4的名字吗？如果你能做到，就是相当不错的初学者了。下面的查询可以做到：

```
mysql> SELECT first_name,surname,sales_rep,COUNT(*) AS
 count from sales,sales_rep WHERE sales_rep=employee_number
 GROUP BY sales_rep,first_name,surname ORDER BY count
 LIMIT 1;
+-------------+---------+-----------+-------+
| first_name | surname | sales_rep | count |
+-------------+---------+-----------+-------+
| Mongane     | Rive    |         4 |     1 |
+-------------+---------+-----------+-------+
1 row in set (0.00 sec)
```

小结

MySQL是一个关系数据库管理系统。逻辑上，数据被组织到表中，表通过一个共同的字段而相互关联。表由行（记录）组成，记录由列（字段）组成。字段可以是不同的类型：数字类型、字符串类型或者日期类型（本章只是引入了SQL，后面章节将详细介绍这种语言）。

MySQL服务器用于存储数据和运行查询。要想连接到MySQL服务器，需要一台MySQL客户机。它既可以和服务器是同一台计算机，也可以是一台远程计算机。

一个数据库管理系统的能力来自于其组织数据和检索数据的能力。SQL是处理数据和定义数据的行业标准，其最重要的命令如下：

- CREATE语句创建数据库及其中的表
- INSERT语句向表中放置记录

- SELECT语句返回查询的结果
- DELETE语句从表中删除记录
- UPDATE语句修改表中的数据
- ALTER语句更改表的结构，利用ADD添加新的列、CHANGE改变已有列的名称和定义、RENAME为一个表进行重命名、DROP删除一个表

函数增强了MySQL的能力。你可以通过圆括号迅速识别一个函数。MySQL中有许多函数：数学函数（如SUM()用于计算结果集的总数）；日期和时间函数（如YEAR()用于提取日期的年份部分）；字符串函数（如RIGHT()用于提取字符串中从右侧开始的一部分）。

有了这些基本的信息，你就可以开始学习如何组织数据，深入到高级的SQL，并且会遇到MySQL使用的各种类型的表。

第2章 数据类型和表类型

我们知道，MySQL使用了几种表类型。默认情况下，它使用MyISAM表类型，其为SELECT的速度而优化。大多数的网站使用这种表，因为网站很少使用INSERT或UPDATE语句，而经常会用到SELECT语句。本章将详细介绍所有的表类型。在第1章中，我们已经简单介绍了几种数据类型。本章将继续介绍所有可用的数据类型，以及什么时候使用它们。

本章内容包括：

- 数字、字符串和日期列类型
- MySQL命令行选项
- 逻辑、算术、比较和位运算符
- 连接MySQL的选项
- 理解表类型

列类型

要想有效地使用MySQL，首先要理解它的基本构造元素。MySQL清单有时会有一些问题，其解决方案通常很简单，只需使用另外一种列类型或表类型，或对其功能有更好的理解。本章首先讨论各种列类型，然后讨论MySQL中的表类型。

MySQL中有三种主要的列类型：数字、字符串和日期。尽管还有许多的类型我们也将简要介绍，但我们可以将它们分类到三种主要类型当中的一种。通常，你应该选择尽可能小的列类型，这样可以节省空间，并且可以更快地进行访问和更新。然而，如果选择的类太小，列的结果可能会在插入时丢失数据，所以一定要选择一个覆盖所有可能性的类型。下面的内容将详细介绍每个类型。

说明： 列名称是大小写不相关的，所以SELECT field1 FROM tablename与SELECT FieLD1 FROM tablename是一样的。但是要注意，表名称和数据库名称可以是大小写相关的。默认情况下，它们在Windows中是大小写不相关的，但在UNIX的大多数版本中是大小写相关的。

数字列类型

数字类型用于存储各种类型的数字数据，如价格、年龄或者数量。数字类型主要分为两种：整型（完全是数字，没有任何的小数或分数位）和浮点型。所有的数字类型允许两个选项：UNSIGNED和ZEROFILL。UNSIGNED不允许有负数（将正数的范围扩大），而ZEROFILL为该值添加上零，而不是常用的空格，并且自动将它变为UNSIGNED。例如：

```
mysql> CREATE TABLE test1(id TINYINT ZEROFILL);
Query OK, 0 rows affected (0.32 sec)
```

```
mysql> INSERT INTO test1 VALUES(3);
Query OK, 1 row affected (0.16 sec)

mysql> INSERT INTO test1 VALUES (-1)
Query OK, 1 row affected (0.16 sec)

mysql> INSERT INTO test1 VALUES (256)
Query OK, 1 row affected (0.16 sec)

mysql> SELECT * from test1;
+------+
| id   |
+------+
|  003 |
|  000 |
|  255 |
+------+
3 rows in set (0.00 sec)
```

注意，因为字段是UNSIGNED，负数被调整为适合该范围，又因为256超过了范围的最大值，它被调整为255，也就是所允许的最大正值。

说明：在数字列类型上执行查询时，不需要对数字值使用引号。

表2.1列出了MySQL中可用的数字类型。

表2.1 数字类型

类型	描述
TINYINT[(M)] [UNSIGNED] [ZEROFILL]	一个微小整数，－128～127（SIGNED），0～255（UNSIGNED），需要1字节的存储
BIT	同TINYINT(1)
BOOL	同TINYINT(1)
SMALLINT[(M)] [UNSIGNED] [ZEROFILL]	一个小整数，－32 768～32 767（SIGNED），0～65 535（UNSIGNED），需要2字节的存储
MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]	一个中等整数，－8 388 608～8 388 607（SIGNED），0～16 777 215（UNSIGNED），需要3字节的存储
INT[(M)] [UNSIGNED] [ZEROFILL]	一个整数，－2 147 483 648～2 147 483 647（SIGNED），0～4 294 967 295（UNSIGNED），需要4字节的存储
INTEGER	同INT
BIGINT[(M)] [UNSIGNED] [ZEROFILL]	一个大整数，－9 223 372 036 854 775 808～9 223 372 036 854 775 807（SIGNED），0～18 446 744 073 709 551 615（UNSIGNED），需要8字节的存储。参见此表后面的规则，其中有一些使用BIGINTS的注意事项

（续表）

类型	描述
FLOAT(precision) [UNSIGNED] [ZEROFILL]	一个浮点数。precision <=24用于单精度浮点数；precision在25和53之间，用于双精度浮点数。FLOAT(X)与相应的FLOAT和DOUBLE类型有着相同的范围，但是没有定义显示尺寸和小数位数。在MySQL 3.23之前，这不是一个真的浮点值，且总是有两位小数。MySQL中的所有计算都用双精度，所以这会带来一些意想不到的问题
FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]	一个小的或单精度浮点数。－3.402823466E+38到－1.175494351E－38、0和1.175494351E－38到3.402823466E+38。如果是UNSIGNED，正数范围保持一样，但负数是不允许的。M是指总的显示宽度，D是指小数的位数。不带参数的FLOAT或者FLOAT(X)中的X <= 24表示一个单精度浮点数。FLOAT(X)中的X在25到53之间，表示一个双精度浮点数。需要4字节的存储（单精度）
DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]	一个双精度浮点数。－1.7976931348623157E+308到－2.2250738585072014E－308、0和2.2250738585072014E－308到1.7976931348623157E+308。如果是FLOAT，UNSIGNED不会改变正数范围，但不允许负数。M是指总的显示宽度，D是指小数位数。没有参数的DOUBLE或者FLOAT(X)中的X在25和53之间，表示一个双精度浮点数。需要8字节的存储
DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL]	同DOUBLE
REAL[(M,D)] [UNSIGNED] [ZEROFILL]	同DOUBLE
DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]	将一个数像字符串那样存储，每个字符占一个字节（这叫做unpacked，所有其他的数字类型被打包）。－1.7976931348623157E+308到－2.2250738585072014E－308、0和2.2250738585072014E－308到1.7976931348623157E+308。M是指总的位数（在版本3.23以后，不包括符号和小数点）。D是指小数点后面的位数。它应该总是小于M。如果被省略，D就默认为0。与其他的数字类型不同，M和D可以约束允许值的范围。如果用UNSIGNED，就不允许负值
DEC[(M[,D])] [UNSIGNED] [ZEROFILL]	同DECIMAL
NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL]	同DECIMAL

在决定选择哪种数字类型时，可以遵循如下的原则：

- 选择最小的可用类型（如果值永远不超过127，则TINYINT强于INT）。
- 对于完全都是数字的，选择整数类型（记住，钱也可以存储为完全的数字。例如，可以存储为美分，而不是美元，这就是整数）。它也可以存储成小数。
- 对于高精度，使用整数类型而不使用浮点类型（舍入的错误会影响浮点数）。

表2.1中的M值通常会导致混淆。将M的值设置得比该类型允许的高，这将不允许超出该范围。例如：

```
mysql> CREATE TABLE test2(id TINYINT(10));
Query OK, 0 rows affected (0.32 sec)

mysql> INSERT INTO test2(id) VALUES(100000000);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT id FROM test2;
+------+
| id   |
+------+
|  127 |
+------+
1 row in set (0.00 sec)
```

尽管插入的数字比10位小，但是因为它被分配了TINYINT，所以它被限制在最大正值为127。

对于宽度小于指定列宽度的值，可选择的宽度规范在左边补齐该值的显示；但DECIMAL字段是个例外，它既不约束列中存储的值的范围，也不约束宽度超过指定值所显示的位数。

然而，如果要将一个类型限制为小于所允许的范围，值不会丢失。它不约束可以存储的范围，也不约束显示的位数，例如：

```
mysql> CREATE TABLE test3(id INT(1));
Query OK, 0 rows affected (0.32 sec)

mysql> INSERT INTO test3(id) VALUES(42432432);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT id FROM test3;
+----------+
| id       |
+----------+
| 42432432 |
+----------+
1 row in set (0.16 sec)
```

宽度规范最常见的是与zerofill一同使用，因为可以很容易地看到结果：

```
mysql> CREATE TABLE test4(id INT(3) ZEROFILL,id2 INT ZEROFILL);
Query OK, 0 rows affected (0.32 sec)
```

```
mysql> INSERT INTO test4(id,id2) VALUES (22,22);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM test4;
+------+------------+
| id   | id2        |
+------+------------+
|  022 | 0000000022 |
+------+------------+
1 row in set (0.22 sec)
```

在id中，宽度规范的影响是将显示限制为三个字符，同时，id2字段使用常规无符号的INT默认值（10）。

字符串列类型

字符串列用于存储任何类型的字符数据，如名字、地址或者报纸文章。表2.2描述了MySQL中可用的字符串类型。

表2.2 字符串类型

类型	描述
[NATIONAL] CHAR(M) [BINARY]	字符。固定长度的串，在右边补齐空格，达到指定的长度。从0到255个字符（MySQL 3.23版本之前是1～255）。检索值时，后缀的空格被删除
CHAR	与CHAR（1）同义
[NATIONAL] VARCHAR(M) [BINARY]	可变长字符。一个可变长度的串，其中的后缀空格在存储值时被删除（这是一个小问题，可以发现来自另一个DBMS的结果）。从0到255个字符（MySQL 4.0.2版本之前是1～255）
TINYBLOB	微小的二进制大对象。最多255个字符（2^8-1）。要求长度+1字节的存储。与TINYTEXT一样，只不过搜索是大小写相关的。大多数情况下要使用它，而不是VARCHAR，因为它更快一些
TINYTEXT	最大255个字符（2^8-1）。要求长度+1字节的存储。与TINYBLOB一样，只不过搜索是大小写不相关的。大多数情况下要使用VARCHAR，因为它更快一些
BLOB	二进制大对象。最大65 535个字符（$2^{16}-1$）。要求长度+2字节的存储。与TEXT一样，只不过搜索是大小写相关的
TEXT	最大65 535个字符（$2^{16}-1$）。要求长度+2字节的存储。与BLOB一样，只不过搜索是大小写不相关的
MEDIUMBLOB	中等大小的二进制大对象。最大16 777 215个字符（$2^{24}-1$）。要求长度+3字节的存储。与MEDIUMTEXT一样，只不过搜索是大小写相关的

（续表）

类型	描述
MEDIUMTEXT	最大16 777 215个字符（$2^{24}-1$）。要求长度+3字节的存储。与MEDIUMBLOB一样，只不过搜索是大小写不相关的
LONGBLOB	大的二进制大对象。最大4 294 967 295个字符（$2^{32}-1$）。要求长度+4字节的存储。与LONGTEXT一样，只不过搜索是大小写相关的。注意，由于外部限制，这里也有一个限制，即16MB每通信包/表行
LONGTEXT	最大4 294 967 295个字符（$2^{32}-1$）。要求长度+4字节的存储。与LONGBLOB一样，只不过搜索是大小写不相关的。注意，由于外部限制，这里也有一个限制，即16MB每通信包/表行
ENUM('value1','value2',...)	枚举。只能有一个指定值，即NULL或“”。最大有65 535个值
SET('value1','value2',...)	一个集合。可以包含0到64个值，均来自于指定清单

在决定使用哪种字符串类型的时候，要遵守如下的指导方针：

- 不要在字符串列中存储数字。将数字存储在数字列中，效率能够高得多。字符串字段的每一位要占一个整个的字节，而在数字字段中，它是以位存储的。同时，如果在字符串中排序数字，会产生矛盾的结果。
- 从速度方面考虑，要选择固定的列，如CHAR。
- 为了节省空间，使用动态的列，如VARCHAR。
- 为了将列中的内容限制在一种选择，使用ENUM。
- 为了允许在一个列中有多于一个的条目，选择SET。
- 对于想搜索的大小写不相关文本，使用TEXT。
- 对于想搜索的大小写相关文本，使用BLOB。
- 对于图像和其他二进制的对象，将它们存储在文件系统中，而不要直接存储在数据库中。

默认情况下，CHAR和VARCHAR类型在搜索时是大小写不相关的，除非你使用了BINARY关键字。例如：

```
mysql> CREATE TABLE test5(first_name CHAR(10));
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO test5(first_name) VALUES ('Nkosi');
Query OK, 1 row affected (0.06 sec)

mysql> SELECT first_name FROM test5 WHERE first_name='nkosi';
+------------+
| first_name |
+------------+
| Nkosi      |
+------------+
1 row in set (0.17 sec)
```

即使指定了nkosi，而不是Nkosi，这个搜索也会返回一个结果。如果你改变了表，指定first_name列为BINARY，就找不到结果，如下：

```
mysql> ALTER TABLE test5 CHANGE first_name first_name CHAR(10) BINARY;
Query OK, 1 row affected (0.16 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> SELECT first_name FROM test5 WHERE first_name='nkosi';
Empty set (0.17 sec)
```

说明： 在大多数DBMS中，大小写不相关地搜索CHAR和VARCHAR字段是不常见的，所以如果你是从其他的DBMS转到MySQL的，一定要小心。

NATIONAL关键字只是为了满足ANSI SQL（ANSI就是美国国家标准化组织，他们开发了“标准化”的SQL。大多数数据库管理系统（DBMS）都在某种程度上遵守它，但都没有完全遵守，都有自己的特点）的要求。它告诉DBMS使用MySQL默认的字符集（MySQL标准）。

说明： 使用CHAR而不是VARCHAR会导致很大的表，但通常处理得很快，因为MySQL完全了解每个记录从哪里开始。在“MyISAM表”一节可以看到对此问题的讨论。

ENUM列有一些有趣的功能。如果添加了一个无效的值，就会插入一个空串（“”），如下所示：

```
mysql> CREATE TABLE test6(bool ENUM("true","false"));
Query OK, 0 rows affected (0.17 sec)

mysql> INSERT INTO test6(bool) VALUES ('true');
Query OK, 1 row affected (0.17 sec)

mysql> INSERT INTO test6(bool) VALUES ('troo');
Query OK, 1 row affected (0.06 sec)

mysql> SELECT bool from test6;
+------+
| bool |
+------+
| true |
|      |
+------+
2 rows in set (0.11 sec)
```

也可以在枚举字段上根据索引（从1开始的第一个值）执行查询。在前面的例子中，true反映了索引1，false反映了索引2，NULL反映了索引NULL，任何其他的值反映了索引0。例如：

```
mysql> SELECT * FROM test6 WHERE bool=0;
+------+
| bool |
+------+
|      |
+------+
```

```
1 row in set (0.17 sec)

mysql> SELECT * FROM test6 WHERE bool=1;
+------+
| bool |
+------+
| true |
+------+
1 row in set (0.16 sec)
```

下面的例子显示了该查询。如果直接插入一个索引，返回结果时可以看到完整的枚举值：

```
mysql> INSERT INTO test6(bool) VALUES(2);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM test6;
+-------+
| bool  |
+-------+
| true  |
|       |
| false |
+-------+
3 rows in set (0.16 sec)
```

警告：LOAD DATA不允许使用索引向枚举字段添加记录，因为它将所有的输入作为字符串来处理。

枚举字段以索引值来分类，而不是按照字母顺序。也就是说，它们按照定义值的顺序来分类：

```
mysql> SELECT * FROM test6 ORDER BY bool ASC;
+-------+
| bool  |
+-------+
|       |
| true  |
| false |
+-------+
3 rows in set (0.22 sec)
```

集合的工作方式类似于枚举字段：

```
mysql> CREATE TABLE test7 (fruit SET('apple','mango','litchi','banana'));
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO test7 VALUES('banana');
Query OK, 1 row affected (0.17 sec)

mysql> INSERT INTO test7 VALUES('litchi');
Query OK, 1 row affected (0.05 sec)
```

```
mysql> INSERT INTO test7 VALUES('paw-paw');
Query OK, 1 row affected (0.00 sec)
```

SET类型的不同之处在于，可以添加多个实例：

```
mysql> INSERT INTO test7 VALUES('apple,mango');
Query OK, 1 row affected (0.06 sec)

mysql> SELECT * FROM test7;
+-------------+
| fruit       |
+-------------+
| banana      |
| litchi      |
|             |
| apple,mango |
+-------------+
4 rows in set (0.17 sec)
```

如果是枚举，分类是按照索引进行的：

```
mysql> INSERT INTO test7 VALUES('mango,apple');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM test7 ORDER BY fruit;
+-------------+
| fruit       |
+-------------+
|             |
| apple,mango |
| apple,mango |
| litchi      |
| banana      |
+-------------+
5 rows in set (0.11 sec)
```

注意，元素的顺序总是与CREATE TABLE语句指定的一致，所以mango, apple存储为apple, mango，在分类结果中也是这样显示。

说明： 你可以创建一个CHAR(0)类型的列。这看上去没有用，但在使用依赖于字段存在的旧的应用程序，此程序又不实际进行存储时，它会很有用。如果需要一个只包含两个值NULL和“”的字段，也可以使用它。

日期和时间列类型

日期和时间列类型用于处理时间数据，可以存储当日的时间或出生日期这样的数据。表2.3描述了MySQL中可用的日期列类型。

表2.3 日期类型

类型	描述
DATETIME	YYYY-MM-DD HH:MM:SS，从1000-01-01 00:00:00到9999-12-31 23:59:59
DATE	YYYY-MM-DD，从1000-01-01到9999-12-31
TIMESTAMP	YYYYMMDDHHMMSS
TIME	HH:MM:SS
YEAR	YYYY

TIMESTAMP列类型可以由不同的方式来显示，如表2.4所示。

表2.4 TIMESTAMP类型

类型	描述
TIMESTAMP(14)	YYYYMMDDHHMMSS
TIMESTAMP(12)	YYMMDDHHMMSS
TIMESTAMP(10)	YYMMDDHHMM
TIMESTAMP(8)	YYYYMMDD
TIMESTAMP(6)	YYMMDD
TIMESTAMP(4)	YYMM
TIMESTAMP(2)	YY

这并不意味着数据丢失。数字只是影响显示，即使是在定义为TIMESTAMP(2)的列中，完整的14位数被存储，所以如果后面要改变表的定义，完整的TIMESTAMP将会正确显示。

警告： 除UNIX_TIMESTAMP()以外，函数处理显示值。所以，DAYOFWEEK()函数不会处理TIMESTAMP(2)或TIMESTAMP(4)。

MySQL在接收数据格式时是很宽容的。你可以用任何其他的标点符号来代替连字符（-）或冒号（:），而不影响其有效性。例如：

```
mysql> CREATE TABLE tt(ts DATETIME);

mysql> INSERT INTO tt(ts) VALUES('1999+11+11 23-24');
Query OK, 1 row affected (0.06 sec)
```

你甚至可以用另一个字符来代替空格。下面的例子用一个等号代替了它：

```
mysql> INSERT INTO tt(ts) VALUES('1999+12=12-12'12');
Query OK, 1 row affected (0.05 sec)
```

如果输入的值是无效的，不会得到错误消息，而是结果被设置为0（0000是YEAR类型，00:00:00是TIME类型，等等）。

MySQL选项

在运行mysql命令连接到MySQL时，可以使用表2.5中所列的任何选项。

表2.5 MySQL选项

选项	描述
-?, --help	显示帮助和退出
-A, --no-auto-rehash	允许快速启动。自动重复允许按Tab键；MySQL来完成表或字段。MySQL在启动时推敲字段和表的名字，但有时，如果有许多的表和字段，启动会变得很慢。用此选项会改变这种情况。要想在指定了此选项时使用hashing，在命令行上输入rehash
-b, --no-beep	关闭出错时的嘟嘟声
-B, --batch	在批处理模式中接受SQL语句。用制表符分隔显示结果。不使用历史文件
--character-sets-dir=...	告诉MySQL，字符集位于哪个目录
-C, --compress	在服务器/客户机协议中使用压缩
-#, --debug[=...]	创建一个调试日志。默认时是d:t:o:，/tmp/mysql.trace，其激活调试，打开函数调用入口和退出跟踪，输出到/tmp/mysql.trace。你可以指定另一个文件来覆盖它
-D, --database=...	表明使用哪个数据库。一般情况下，你可以不指定此选项而选择一个数据库，但在配置文件中，这是有用的
--default-character-set=...	设置默认的字符集
-e, --execute=...	执行命令和退出。其输出与-B选项是一样的
-E, --vertical	垂直地打印查询的输出，每个字段占一行。没有这个选项，你也可以通过以\G结束语句来强制这种输出
-f, --force	强制MySQL继续处理，即使得到了一个SQL错误。在处理文件时，用于批处理模式
-g, --no-named-commands	禁用命名的命令。只使用*形式，或者只在以分号结尾的行的开头使用命名的命令。自从版本10.9以来，客户端默认以此选项开始。然而，用-g选项，长格式命令仍然可以从第一行工作
-G, --enable-named-commands	允许命名的命令。长格式命令是允许的，还有缩短的*命令
-h, --host=...	连接到指定的主机
-H, --html	将查询结果格式化为HTML。一般情况下，你会使用一个编程语言来格式化它，但这个选项可以用于快速生成HTML
-i, --ignore-space	忽略函数名后面的空格

（续表）

选项	描述
-L, --skip-line-numbers	使MySQL不为错误写出行号。用于输出到结果文件后要搜索错误或进行比较的情况下
--no-pager	禁用分页器，导致输出成为标准输出。参见-pager选项
--no-tee	禁止输出文件。参见交互式的帮助（\h）
-n, --unbuffered	在每个查询之后清空缓冲器
-N, --skip-column-names	使MySQL不写出结果中的列名
-O, --set-variable var=option	给变量赋一个值。--help列出变量
-o, --one-database	只更新默认的数据库。这用于在更新日志中越过更新到其他数据库时
--pager[=...]	长结果输出将会卷到屏幕以外。可以将结果输出到一个分页器。默认的分页器是你的ENV变量PAGER。有效的分页器包括less、more、cat [>filename]等等。这个选项在批处理模式中不工作。Pager只用于UNIX
-p[password], --password[=...]	连接到服务器时使用的密码。如果没有在命令行上给出密码，你会得到提示。如果在命令行上输入密码，选项和密码之间不能有空格
-P, --port=...	默认情况下，你是通过端口3306连接到MySQL服务器。你可以通过指定TCP/IP端口号来改变它
-q, --quick	使结果一行一行地显示。如果结果很大，这将加快输出的速度。如果输出被延缓，这会降低服务器的速度。不使用历史文件
-r, --raw	写出不带转义符转换的列值。与--batch一同使用
-s, --silent	不显示过多的输出
-S, --socket=...	连接所用的套接字文件
-t, --table	以表的格式输出。在非批处理模式中，这是默认的
-T, --debug-info	在退出时打印一些调试信息
--tee=...	将每个数据添加到输出文件。参见交互式的帮助（\h）。在批处理模式中不能工作
-u, --user=#	指定一个注册用户。如果没有指定该用户，MySQL就假设是当前用户（如果有的话）
-U, --safe-updates[=#], --i-am-a-dummy[=#]	只允许UPDATE和DELETE使用键。如果此选项被默认设置，你可以使用--safe-updates=0重新设置
-v, --verbose	使MySQL给出更详细的输出（-v-v-v给出表的输出格式，-t）
-V, --version	输出版本信息并退出
-w, --wait	如果连接断开，此选项将等待并试图再连接，而不是中断

自动重复功能允许按Tab键来完成表或字段。MySQL在你连接的时候创建了它，但有时，如果有许多表和字段，启动会变得很慢。使用-A或-- no-auto-rehash选项来改变这种情况。

-E选项垂直地打印结果。如果在查询结尾使用\G，即使没有激活此选项而连接到了MySQL，也可以得到此类输出：

```
mysql> SELECT * FROM customer\G;
*************************** 1. row ***************************
        id: 1
first_name: Yvonne
   surname: Clegg
*************************** 2. row ***************************
        id: 2
first_name: Johnny
   surname: Chaka-Chaka
*************************** 3. row ***************************
        id: 3
first_name: Winston
   surname: Powers
*************************** 4. row ***************************
        id: 4
first_name: Patricia
   surname: Mankunku
*************************** 5. row ***************************
        id: 5
first_name: Francois
   surname: Papo
*************************** 6. row ***************************
        id: 7
first_name: Winnie
   surname: Dlamini
*************************** 7. row ***************************
        id: 6
first_name: Neil
   surname: Beneke
7 rows in set (0.00 sec)
```

忽略空格的选项（-i）使你在使用函数时更加轻松。例如，下面的语句通常会导致错误（注意MAX后面的空格）：

```
mysql> SELECT MAX (value) FROM sales;
ERROR 1064: You have an error in your SQL syntax near '(value) from
 sales' at line 1
```

但如果用了-I选项来连接，就没有问题了：

```
mysql> SELECT MAX(value) FROM sales;
+--------------+
| MAX (value) |
+--------------+
|         3800 |
+--------------+
```

-H（或--html）选项将查询结果放到一个HTML表中。如果用此选项进行连接，会看到下面的输出分类：

```
mysql> SELECT * FROM customer;
<TABLE BORDER=1><TR><TH>id</TH><TH>first_name</TH><TH>surname</TH></TR>
<TR><TD>1</TD><TD>Yvonne</TD><TD>Clegg</TD></TR>
<TR><TD>2</TD><TD>Johnny</TD><TD>Chaka-Chaka</TD></TR>
<TR><TD>3</TD><TD>Winston</TD><TD>Powers</TD></TR>
<TR><TD>4</TD><TD>Patricia</TD><TD>Mankunku</TD></TR>
<TR><TD>5</TD><TD>Francois</TD><TD>Papo</TD></TR>
<TR><TD>7</TD><TD>Winnie</TD><TD>Dlamini</TD></TR>
<TR><TD>6</TD><TD>Neil</TD><TD>Beneke</TD></TR></TABLE>
7 rows in set (0.00 sec)
```

-o选项只允许更新到默认的数据库。如果用此选项连接，就不能更新firstdb数据库中的任何表：

```
mysql> UPDATE customer SET first_name='John' WHERE first_name='Johnny';
Ignoring query to other database
```

-U选项帮助避免不允许未使用键的UPDATE或DELETE的情况（第4章“索引和查询优化”将介绍键）。如果用此选项进行连接，下列命令将不能工作：

```
mysql> DELETE FROM customer;
ERROR 1175: You are using safe update mode and you tried to update a
 table without a WHERE that uses a KEY column
```

表类型

有两个事务安全的表类型（InnoDB和BDB），但其他（ISAM、MyISAM、MERGE和HEAP）都不是事务安全的。选择正确的表类型能够极大地影响性能速度。

ISAM表

ISAM（索引顺序存储方法）表类型是旧的MySQL标准。在版本3.23.0中，MyISAM表类型代替了它（尽管ISAM类型仍然能用到版本4.1）。所以，如果你使用的是过时的数据库，可能只能遇到ISAM表类型。这两种类型的主要区别是，MyISAM表的索引比ISAM表的索引要小得多，所以在MyISAM表中带有索引的SELECT使用更少的系统资源。只不过MyISAM使用更强的处理器能力，来向压缩的索引插入记录。

ISAM表有如下特点：

- ISAM用.ISD来存储数据文件，索引文件的扩展名为.ISM。
- 不能在不同的机器或操作系统上互相访问。也就是说，你不能只拷贝ISD和ISM文件。必须使用其他的备份方法，如mysqldump（参见第11章“数据库备份”）。

如果你确实是在运行ISAM表类型，就应该将它改成更有效率的MyISAM类型。MyISAM表也允许使用MySQL的内置功能。要想将ISAM表转换成MyISAM表，用下面的语句：

```
ALTER TABLE tablename TYPE = MYISAM;
```

MyISAM表

在版本3.23.0中，MyISAM表类型代替了ISAM。MyISAM索引比ISAM索引小得多，所以系统将使用更少的资源执行带有索引的SELECT。然而，MyISAM使用更多的处理器能力来将记录插入压缩的索引。

MyISAM数据文件的扩展名是.MYD，索引的扩展名是.MYI。MyISAM数据库存储在一个目录里，所以如果你从第1章起一直跟着做练习，并且有权查看目录firstdb，会看到下面的文件：

- sales_rep.MYI
- sales_rep.MYD
- sales.MYD
- sales.MYI
- customer.MYD
- customer.MYI

数据文件总是比索引文件大。在第4章，我们将学习如何正确地使用索引，以及它们包含什么。

MyISAM表有三个子类型：静态、动态或压缩。

在创建表的时候，MySQL决定使用动态或静态的表。静态表是默认格式，只要没有VARCHAR、BLOB或TEXT列，它就存在。如果这些列类型存在一种，表类型就成了动态的。

静态表

静态表（也叫做定长表）有固定的长度。图2.1显示了一个小表中存储在一个字段中的字符。该字段是一个名字，设置为CHAR(10)。

I	A	N							
V	I	N	C	E	N	T			
M	I	R	I	A	M				

图2.1 以静态格式存储的数据

每个记录正好存储了10个字节。如果实际的名字占用较少的字节，列的其余部分添加空格，以达到10个字符。

静态表的特点如下：

- 很快（因为MySQL知道名字总是在第11个字符处开始）
- 容易缓存
- 出现问题后容易重建（也是因为记录的位置是固定的，MySQL知道每个记录在哪里，所以只有出问题时写的那个记录会丢失）
- 要求更多的磁盘空间（3个记录需要30个字符，即使用于名字的只有16个字符）
- 不必用myisamchk进行重组（参见第10章“基本管理”）

动态表

动态表中的列是不同长度的。如果将静态表中使用的数据放入动态表，它将像图2.2那样存储。

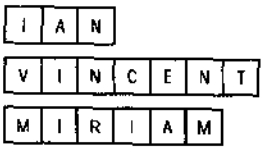

图2.2 以动态格式存储的数据

虽然这种数据格式节省空间，但是它更复杂。每条记录有一个头，来表明该记录有多长。

动态表类型的特点如下：

- 所有的字符串列都是动态的（除非它们少于4字节。在这种情况下，节省的空间可以忽略，而增加的复杂性会导致性能丢失）。
- 通常比定长表占用少得多的磁盘空间。
- 表需要经常维护，以避免碎片（例如，将Ian更新为Iane，字母e不能马上出现在字母n的后面，因为这个空间被后面的列或记录占用了）。参见第10章对维护的介绍。
- 对于有碎片的列，每个新链接损失6个字节，大小至少有20字节（如果进一步的更新超过了该大小，也许会有其自己的链接）。
- 不容易在出现问题后重建，特别是如果碎片问题严重的话。
- 不包括链接，动态记录的尺寸可以用这个公式计算：

  3
  + （列数 + 7）/ 8
  + （字符列数）
  + 数字列的打包尺寸
  + 字符串长度
  + （空列的数目 + 7）/ 8

- 每条记录有一个头，表明哪个字符串列是空的，哪个数字列包含0（非空记录），在哪种情况下不向磁盘存储。非空字符串包括一个长度字节，加上字符串内容。

压缩表

压缩表是只读表类型，使用很少的磁盘空间。它们用于不会改变的存档数据是最理想的（因为它们是只读的，不能写入），或者用于没有太多空间的时候，如CD-ROM。

压缩表的特点如下：

- 用myisampack工具创建（注意，CREATE TABLE的选项ROW_FORMAT= "compressed"只在服务器添加了myisampack代码后才能工作）。
- 表要少得多。
- 因为每条记录是分开压缩的，所以不能同时访问。
- 每个列可以以不同方式压缩，只需使用不同的压缩算法。
- 可以压缩定长或动态表格式。
- 要想用myisampack创建压缩表，运行下列语句：

```
myisampack [options] filename
```

表2.6显示了压缩表的选项。

表2.6 压缩表选项

选项	描述
-b, --backup	创建表的备份，叫做tablename.OLD
-#, --debug=debug_options	输出调试日志。debug_options串通常是d:t:o, filename
-f, --force	在压缩时，MySQL创建一个暂时的文件，叫做tablename.TMD。如果压缩过程失败，此文件不会被删除。如果压缩导致表变大，或者如果表太小以致于不能压缩，此选项强制MySQL打包表，即使存在暂时文件
-?, --help	显示帮助消息并退出
-j big_tablename, --join=big_tablename	将命令行上列出的所有表连接成一个大表。要连接的所有表必须是一致的（如列和索引）
-p #, --packlength=#	通常，只在第二次运行myisampack时使用此选项。myisampack存储所有的行，并带有1～3的长度指示。有时，myisampack注意到处理过程中应该使用一个短一些的指示。下一次再打包表的时候，你可以让myisampack使用可选的长度存储尺寸
-s, --silent	无记载模式。只输出错误
-t, --test	此选项不会实际打包表，它只是测试打包过程
-T dir_name, --tmp_dir=dir_name	将暂时的表写入指定的目录
-v, --verbose	详细模式。写出过程信息和打包结果
-V, --version	显示版本信息并退出
-w, --wait	如果正在使用表，此选项等待并重试。如果在打包时有可能更新表，就不推荐将此选项与--skip-external-locking结合使用

我们来压缩前面用过的一个表。因为该表很小以致于不能正常压缩，所以必须使用-f选项：

```
C:\Program Files\MySQL\bin>myisampack -v -f  ..\data\firstdb\sales_~1
Compressing ..\data\firstdb\sales_~1.MYD: (5 records)
- Calculating statistics

normal:      3  empty-space:       0  empty-zero:         2  empty-fill:   1
pre-space:   0  end-space:         2  intervall-fields:   0  zero:         0
Original trees:  7  After join: 1
- Compressing file
Min record length:     10   Max length:     17   Mean total length:     40
-35.81%
```

为了对表进行拆包，运行myisamchk --unpack filename:

```
C:\Program Files\MySQL\bin>myisamchk --unpack ..\data\firstdb\sales_~1
- recovering (with keycache) MyISAM-table '..\data\firstdb\sales_~1'
Data records: 5
```

MERGE表

MERGE表是相同MyISAM表的合并。它们是在版本3.23.25中引入的。只有当MyISAM表变得太大时，才使用它们。

MERGE表的优点如下：

- 在有些情况下速度更快（你可以将不同的表分离到不同的磁盘，并用MERGE表将它们作为一个表来访问）。
- 表的尺寸较小。有些操作系统有文件尺寸限制，拆分表并创建一个MERGE表可以避免此问题。同时，文件更容易传输，如将它们拷贝到CD上。
- 你可以使大多数的原表成为只读的，并允许在最近的表中插入数据。这意味着，你所冒的风险是在UPDATE和INSERT的过程中一个小表被破坏，并且该表的修补比较快。

MERGE表的缺点如下：

- 对于eq_ref搜索，它们很慢。
- 在更改表的时候要特别小心，因为这将破坏MERGE表（没有实际的损害，只是MERGE表可能难以获得）。
- REPLACE不能工作。
- 表使用比较多的文件描述符。

我们来创建一个MERGE表。首先，需要创建两个同样的表：

```
CREATE TABLE sales_rep1 (
id INT AUTO_INCREMENT PRIMARY KEY,
employee_number INT(11),
surname VARCHAR(40),
first_name VARCHAR(30),
commission TINYINT(4),
```

```
date_joined DATE,
birthday DATE
) TYPE=MyISAM;

CREATE TABLE sales_rep2 (
id INT AUTO_INCREMENT PRIMARY KEY,
employee_number INT(11),
surname VARCHAR(40),
first_name VARCHAR(30),
commission TINYINT(4),
date_joined DATE,
birthday DATE
) TYPE=MyISAM;

CREATE TABLE sales_rep1_2 (
id INT AUTO_INCREMENT PRIMARY KEY,
employee_number INT(11),
surname VARCHAR(40),
first_name VARCHAR(30),
commission TINYINT(4),
date_joined DATE,
birthday DATE
) TYPE=MERGE
UNION=(sales_rep1,sales_rep2);
```

向表中插入一些数据，这样，后面就可以对它进行测试：

```
INSERT INTO sales_rep1 ('employee_number', 'surname',
 'first_name', 'commission', 'date_joined', 'birthday')
 VALUES (1,'Tshwete','Paul',15,'1999-01-03','1970-03-04');

INSERT INTO sales_rep2 ('employee_number', 'surname',
 'first_name', 'commission', 'date_joined', 'birthday')
 VALUES (2,'Grobler','Peggy-Sue',12,'2001-11-19','1956-08-25');
```

现在，如果对合并表进行查询，可以得到sales_rep1和sales_rep2中的所有记录：

```
mysql> SELECT first_name,surname FROM sales_rep1_2;
+------------+---------+
| first_name | surname |
+------------+---------+
| Paul       | Tshwete |
| Peggy-Sue  | Grobler |
+------------+---------+
2 rows in set (0.00 sec)
```

基于前面的结果，你不知道任何记录在哪个表中。还好，如果更新记录，不必知道这些。下面的语句

```
mysql> UPDATE sales_rep1_2 set first_name = "Peggy"
 WHERE first_name="Peggy-Sue";
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

可以正确地更新记录。因为记录只是物理地存在于基础级上，对MERGE表和MyISAM表的查询都将反映出正确数据，如下所示：

```
mysql> SELECT first_name,surname FROM sales_rep1_2;
+------------+---------+
| first_name | surname |
+------------+---------+
| Paul       | Tshwete |
| Peggy      | Grobler |
+------------+---------+
2 rows in set (0.00 sec

mysql> SELECT first_name,surname FROM sales_rep2;
+------------+---------+
| first_name | surname |
+------------+---------+
| Peggy      | Grobler |
+------------+---------+
1 row in set (0.00 sec)
```

对DELETE语句也是一样的：

```
mysql> DELETE FROM sales_rep1_2 WHERE first_name='Peggy';
Query OK, 1 row affected (0.00 sec)
```

记录在基础级上被删除，所以它从对MERGE表和基础表的查询中消失：

```
mysql> SELECT first_name,surname FROM sales_rep1_2;
+------------+---------+
| first_name | surname |
+------------+---------+
| Paul       | Tshwete |
+------------+---------+
1 row in set (0.00 sec)

mysql> SELECT first_name,surname FROM sales_rep2;
Empty set (0.00 sec)
```

然而，如果你试图执行一个INSERT，MySQL将不知道把记录插入到哪个基础表，所以它会返回一个错误：

```
mysql> INSERT INTO sales_rep1_2 ('surname', 'first_name',
 'commission', 'date_joined', 'birthday')
 VALUES ('Shephard','Earl',11,'2002-12-15','1961-05-31');
ERROR 1031: Table handler for 'sales_rep1_2' doesn't have this option
```

还好，版本4中引入了一个解决办法（以前，根本就不能插入到MERGE表）。在创建MERGE表时，可以指定插入到哪个表。看下面CREATE语句的最后一个从句：

```
CREATE TABLE sales_rep1_2 (
id INT AUTO_INCREMENT PRIMARY KEY,
employee_number INT(11),
surname VARCHAR(40),
first_name VARCHAR(30),
commission TINYINT(4),
date_joined DATE,
birthday DATE
) TYPE=MERGE
UNION=(sales_rep1,sales_rep2)
INSERT_METHOD = LAST
```

INSERT_METHOD可以是NO、FIRST或者LAST。然后，插入的记录被放入合并清单中的第一个表、最后一个表或根本不放入表。默认值是NO。

警告： 如果对基础表执行结构上的改变，如重命名或重建索引，就需要重建MERGE表。删除MERGE表，然后执行改变，最后重建MERGE表。如果执行改变时忘记了删除MERGE表，就会发现无法正确地访问MERGE表。删除并重建MERGE表可以解决此问题。

HEAP表

HEAP表是最快的表类型，因为它们存储在内存里，并使用散列的索引。其缺点是：由于存储在内存中，所有的数据会在出现问题时丢失。它们也不能保留太多的数据（除非你对RAM有很大的预算）。

你可以根据一个表的内容创建另一个表。HEAP表常用于对现有表提供更快的访问——对原表进行插入和更新，然后生成新表以进行快速读取。让我们从sales_rep表创建一个。如果你还没有创建sales_rep表，就创建它并对其使用下列语句：

```
CREATE TABLE sales_rep (
  employee_number int(11) default NULL,
  surname varchar(40) default NULL,
  first_name varchar(30) default NULL,
  commission tinyint(4) default NULL,
  date_joined date default NULL,
  birthday date default NULL
) TYPE=MyISAM;

INSERT INTO sales_rep VALUES (1, 'Rive', 'Sol', 10,
 '2000-02-15', '1976-03-18');
INSERT INTO sales_rep VALUES (2, 'Gordimer', 'Charlene', 15,
 '1998-07-09', '1958-11-30');
INSERT INTO sales_rep VALUES (3, 'Serote', 'Mike', 10,
 '2001-05-14', '1971-06-18');
INSERT INTO sales_rep VALUES (4, 'Rive', 'Mongane', 10,
 '2002-11-23', '1982-01-04');
```

现在就创建了一个HEAP表，它获得sales_rep的一个子集，并将其放入内存以得到快速访问：

```
mysql> CREATE TABLE heaptest TYPE=HEAP SELECT first_name,surname
 FROM sales_rep;
Query OK, 4 rows affected (0.02 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM heaptest;
+------------+----------+
| first_name | surname  |
+------------+----------+
| Sol        | Rive     |
| Charlene   | Gordimer |
| Mike       | Serote   |
| Mongane    | Rive     |
+------------+----------+
4 rows in set (0.00 sec)
```

HEAP表的特点如下：

- 因为HEAP表使用内存，所以不要让它太大。通常，表受限于mysqld变量max_heap_table_size。
- 不能像MyISAM表那样使用键。它们不能用于ORDER BY。
- 它们只使用完整的键来搜索记录，而不是键的一部分。
- 在搜索索引时，它们只使用=和<=>。
- MySQL的范围优化器不能找到两个值之间有多少条记录。
- 然而，如果在HEAP表中正确地使用了键，它们是很快的！
- 与许多其他的散列表不同，HEAP表允许非惟一的键。
- 它们不支持NULL列上的索引。
- 它们不支持AUTO_INCREMENT列。
- 不支持BLOB或TEXT列。

我们看到，MyISAM索引和HEAP索引有很多的不同。如果你依赖于HEAP表不使用的索引，它就会很慢。第4章介绍了使用键的更多信息。

说明： 除了max_heap_table_size限制和计算机的内存限制以外，可以在有些安装上达到每个表4GB的限制，因为这个限制是由32位计算机的地址空间强加的。

InnoDB表

InnoDB表是事务安全的表类型，也就是说，你具有COMMIT和ROLLBACK的能力。在MyISAM表中，执行插入时整个表被锁定。在那一瞬间，不能对表执行其他的语句。InnoDB使用记录级的锁定，以便只锁定该记录，而不是整个表，语句仍然可以对其他记录执行。

如果你的数据执行大量的INSERT或UPDATES，出于性能方面的考虑，应该使用InnoDB表。如果你的数据库执行大量的SELECT，MyISAM是更好的选择。

要想使用InnoDB表，MySQL要用InnoDB支持来进行编译（参见第15章“安装MySQL”），如mysqld-max distribution。还应该安装一些配置参数，之后再享受这种表类型的优秀性能，所以一定要阅读第13章“配置并优化MySQL”。

启动用InnoDB选项编译过的MySQL，并只使用默认值，可以看到如下的内容：

```
C:\MySQL\bin>mysqld-max
InnoDB: The first specified data file .\ibdata1 did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file .\ibdata1 size to 64 MB
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file .\ib_logfile0 did not exist: new to be created
InnoDB: Setting log file .\ib_logfile0 size to 5 MB
InnoDB: Log file .\ib_logfile1 did not exist: new to be created
InnoDB: Setting log file .\ib_logfile1 size to 5 MB
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: Creating foreign key constraint system tables
InnoDB: Foreign key constraint system tables created
020504 12:42:52  InnoDB: Started
C:\MYSQL\BIN\MYSQLD-2.EXE: ready for connections
```

警告： *在版本4以前，不能只启动MySQL。至少要在配置文件中设置innodb_data_file_path。这个配置文件将在第13章讨论。*

默认情况下，MySQL在默认的数据目录中创建一个文件ibdata1（通常，Windows中是C:\MSQL\data，UNIX中是/usr/local/var/）。

InnoDB表与MyISAM表不同，数据库不是存储在一个目录中，将表作为文件。所有的表和索引都存储在InnoDB表空间（它可以由一个或多个表组成，前面的例子中是ibdata1）。

这时，表数据限制不再限于操作系统的文件尺寸。

说明： *在MyISAM表中，有着2GB限制的操作系统允许表的最大尺寸是2GB。InnoDB没有这种限制，尽管优化表空间的责任落到了管理员身上。*

该表的最初尺寸设置为16MB。在MySQL 4之前的版本，设置为64MB，但是不可扩展（一旦用完空间，就没有办法了）。后面的版本默认地将其设置为自动扩展，也就是说，随着数据的增长，表空间也增长。为了优化性能，你要对此有所控制。此处只是简单介绍一下，第13章将详细讨论。

要创建一个InnoDB表，使用下列语句：

```
mysql> CREATE TABLE innotest (f1 INT,f2 CHAR(10),INDEX (f1)) TYPE=InnoDB;
Query OK, 0 rows affected (0.10 sec)
```

BDB表

BDB就是伯克利数据库（Berkeley Database，最早是在加州大学伯克利分校创建的）。它也是一个事务安全的表类型。和InnoDB表一样，BDB支持也需要被编译进MySQL，使其能够工作（mysql-max版本自带BDB支持）。

要想创建一个BDB表，在CREATE TABLE语句的后面使用TYPE=BDB：

```
mysql> CREATE TABLE bdbtest(f1 INT,f2 CHAR(10)) TYPE=BDB;
Query OK, 0 rows affected (0.28 sec)
```

当前，MySQL和BDB之间的接口正在测试的过程中（它独立于MySQL而存在）。MySQL和BDB已经用了很多年，都很稳定，但它们之间的接口却不是这样。查看最新的文档，了解最新的情况。

小结

如果你想得到MySQL的最好性能，那么为字段选择类型是十分重要的。数字类型可以进行计算，并且通常比字符串类型小。日期类型可以很容易地存储日期和时间数据。

类似地，要想正确地使用MySQL，你需要理解它的运算符。逻辑运算符（如，AND和OR）和比较运算符（如，=和LIKE）帮助将结果的范围从你的查询缩小到真正所需要的结果。位运算符是一个有用的功能，用于二进制数字数据。

MySQL有很多表类型，适合于不同的环境。MyISAM表类型（默认）对于有大量与更新有关的查询的系统是十分理想的（如网站）。MERGE表是同一MyISAM表的融合，允许将大表拆分成多个小表，易于更新，且在某些情况下提高速度。

HEAP表是速度最快的，存储在内存中。InnoDB和BDB表都是事务安全的，允许语句分组，以使数据完整。InnoDB表利用了一致读，这意味着表的结果按照它们完成处理之后出现的样子来显示。这并不总是理想的，你可以不考虑这种行为，用读取块来进行更新和共享。

第3章 高级SQL

在第1章“认识MySQL”中，已经体会到了SQL的一些滋味，但还有更多的东西需要学习。为了体会和掌握MySQL的优势，需要了解MySQL的各种逻辑、算法、比较和位运算。通过这些知识，可以构建比第1章学到的更复杂的查询。同样，会发现有时执行查询时需要多张表中的数据，即连接。奇特的左连接，右连接，外连接，内连接和自然连接似乎很容易混淆。本章将介绍它们，并将说明何时和如何使用它们。

本章的主要内容：

- 逻辑、算法、比较和位运算
- 高级连接（内、外、左、右和自然连接）
- 使用UNION的连接结果
- 用连接重写子选择
- 用DELETE和TRUNCATE删除记录
- 用户变量
- 以批处理模式运行MySQL
- 用BEGIN和COMMIT执行事务
- 一致读
- 表级锁
- 为更新和共享提供的读锁

运算符

运算符是复杂查询的构件块。逻辑运算符（比如AND和OR）可以用来把相关的条件用各种方式组织起来。通过算术运算符（比如+或*）可以在查询中进行基本的算术运算。比较运算符（比如>或<）可以用来比较值，并可以用这种方法缩小结果集。最后，不经常使用的位运算符，可以用来在查询中进行位级的操作。

逻辑运算符

逻辑运算符归纳为true（1）或false（0）。比如，如果一个人是男的，问他是男还是女（假设要一个yes/no的答案），答案就会是yes，也即true（真）。如果问他是否既是男又是女，答案就会是no，也即false（假）。问题中的AND和OR就是逻辑运算符。表3.1更详细地描述了这种运算符。

表3.1 逻辑运算符

运算符	语法	描述
AND, &&	c1 AND c2, c1 && c2	当两个条件c1和c2都为真时结果才为真
OR, ||	c1 OR c2, c1 || c2	当c1或c2为真时，结果就为真
!, NOT	! c1, NOT c1	c1为假结果为真，c1为真结果为假

与上面的组装表并执行查询相反的是，下面的例子将产生0或者1。查询表中的每一行也会产生一个0或者1。1会返回，而0则不会。如果理解了这点，就可以在自己的表中使用这个原则。如果是第一次看到上面这些操作符，看看能否根据表3.1来预测结果。

```
mysql> SELECT 1 AND 0;
+----------+
| 1 AND 0 |
+----------+
|        0 |
+----------+

mysql> SELECT NOT(1 AND 0);
+--------------+
| NOT(1 AND 0) |
+--------------+
|            1 |
+--------------+

mysql> SELECT !((1 OR 0) AND (0 OR 1));
+--------------------------+
| !((1 OR 0) AND (0 OR 1)) |
+--------------------------+
|                        0 |
+--------------------------+
```

记住：在最里边的括号中的条件最先进行。因此，MySQL会把前面例子中的复杂语句简化成如下形式：

```
!((1 OR 0) AND (0 OR 1))
!((1) AND (1))
!(1)
0
```

算术运算符

算术运算符被用来执行基本的算术运算。比如，在2 + 3 = 5中的加号（+）就是一个算术运算符。表3.2中描述的就是MySQL中可用的算术运算符。

表3.2 算术运算符

运算符	语法	描述
+	a + b	把a和b相加，返回两者之和
-	a - b	从a中减去b，返回两者的差
*	a * b	a和b相乘，返回两者的积
/	a / b	a除以b，返回商
%	a % b	a模b，返回 a / b的余数

比如，两个类型为INT的列相加产生的也是一个INT：

```
mysql> SELECT 2+1;
+-----+
| 2+1 |
+-----+
|   3 |
+-----+

mysql> SELECT 4-2/4;
+-------+
| 4-2/4 |
+-------+
|  3.50 |
+-------+
```

这返回3.5，而不是0.5，因为首先进行除（还记得在学校学习的优先级规则吗？）。但是，应该尽量使用括号来表明优先进行的运算以使不知道这些规则的人能清楚。为了前面的查询简单起见，重写如下：

```
mysql> SELECT 4-(2/4);
+---------+
| 4-(2/4) |
+---------+
|    3.50 |
+---------+
```

说明：这个查询中的值都是整数，但因为结果为一个非整数元素，因此它的返回值是一个浮点数。

下面的例子演示模运算符：

```
mysql> SELECT 5 % 3;
+-------+
| 5 % 3 |
+-------+
|     2 |
+-------+
```

模运算返回除的余数。在前面的例子中，5除以3得1，余数为2。

比较运算符

当值之间需要进行比较时使用比较运算符。比如，可以做一个论断，34大于2。大于就是一个比较运算符。表3.3列举和描述了在MySQL中可以使用的比较运算符。

表3.3 比较运算符

运算符	语法	描述
=	a = b	如果a和b相等为真（不包括NULL）
!=, <>	a != b, a <> b	a不等于b时为真
>	a > b	a大于b时为真
<	a < b	a小于b时为真
>=	a >= b	a大于或等于b时为真
<=	a <= b	a小于或等于b时为真
<=>	a <=> b	a和b相等时为真（包括NULL）
IS NULL	a is NULL	a包含一个NULL值时为真
IS NOT NULL	a IS NOT NULL	a不包含一个NULL值时为真
BETWEEN	a BETWEEN b and c	a为b和c之间的值时为真，包含b和c
NOT BETWEEN	a NOT BETWEEN b and c	a不是b和c之间的值时为真，包含b和c
LIKE	a LIKE b	a依据SQL模式匹配上b时为真
NOT LIKE	a NOT LIKE b	a依据SQL模式不能匹配上b时为真。两个可以接受的匹配符为%（意为任意个字符）和 _（意为一个字符）
IN	a IN (b1, b2, b3...)	a等于列表中的任意一个时为真
NOT IN	a NOT IN (b1, b2, b3...)	a不等于列表中的任意一个时为真
REGEXP, RLIKE	a REGEXP b, a RLIKE b	a用普通表达式的方式匹配b时为真
NOT REGEXP, NOT RLIKE	a NOT REGEXP b, a NOT RLIKE B	a不能用普通表达式的方式匹配b时为真

在比较运算符中，也是用1和0来表示真和假的。记住应该用自己的常数和数据库表中的域来替换它们。比如，考虑一个如图3.1所示的有两列的表。

FIELD1
15
13

图3.1 TABLE1

下面的代码演示了=比较运算符：

```
SELECT * FROM TABLE1 WHERE FIELD1 = 13.
```

表中的每一行都会进行比较以查看条件是真还是假。对第一行的表达式如下：

```
15 = 13
```

这是错误的，因此这一行不会返回。对第二行，表达式如下：

```
13 = 13
```

这是正确的，因此这次这一行会返回。
一旦理解了，就可以在自己的表中使用它。
另一个例子：

```
mysql> SELECT 13=11;
+-------+
| 13=11 |
+-------+
|   0   |
+-------+
```

如果有编程背景的话，就可能会知道不同的类型（比如字符串和数字）进行比较的复杂性。比如，如果问字符串‘thirty’是否小于数29，答案会是那种呢？MySQL在进行不同类型的值的比较时，提供了尽量好的帮助（尽量进行转换类型，即类型转换）。如果比较字符串和数字，或者是浮点数与整数进行比较，MySQL对它们进行比较就像它们是相同的类型一样，比如：

```
mysql> SELECT '4200' = 4200.0;
+-----------------+
| '4200' = 4200.0 |
+-----------------+
|               1 |
+-----------------+
```

字符串‘4200’被转换为一个与4200.0相等的一个数。但是，两个字符串‘4200’和‘4200.0’却是不一样的：

```
mysql> SELECT '4200' = '4200.0';
+-------------------+
| '4200' = '4200.0' |
+-------------------+
|                 0 |
+-------------------+
```

下面的例子演示了字符串比较的大小写不相关性：

```
mysql> SELECT 'abc' = 'ABC';
+---------------+
```

```
| 'abc' = 'ABC' |
+---------------+
|             1 |
+---------------+
```

在下面的例子中，在大小写不相关的查询中，结尾的空格被忽略了：

```
mysql> SELECT 'abc' = 'ABC ';
+---------------+
| 'abc' = 'ABC' |
+---------------+
|             1 |
+---------------+
1 row in set (0.00 sec)
```

下面的例子是一个需要注意的重要的例子，它的结果不是0（假）而是NULL：

```
mysql> SELECT NULL=0;
+--------+
| NULL=0 |
+--------+
|   NULL |
+--------+
```

为了对NULL行进行运算，应使用如下的方式：

```
mysql> SELECT NULL<=>0;
+----------+
| NULL<=>0 |
+----------+
|        0 |
+----------+
```

NULL基本上是一个运算的第三种可能的结果：即真、假和NULL。下面没有一个查询在与NULL值比较时提供了有效的结果：

```
mysql> SELECT 200 = NULL, 200 <> NULL, 200 < NULL, 200 > NULL;
+------------+-------------+------------+------------+
| 200 = NULL | 200 <> NULL | 200 < NULL | 200 > NULL |
+------------+-------------+------------+------------+
|       NULL |        NULL |       NULL |       NULL |
+------------+-------------+------------+------------+
```

应该使用IS NULL（或IS NOT NULL）比较来代替：

```
mysql> SELECT NULL IS NULL;
+--------------+
| NULL IS NULL |
+--------------+
```

```
|            1 |
+--------------+

mysql> SELECT 4.5 BETWEEN 4 and 5;
+---------------------+
| 4.5 BETWEEN 4 and 5 |
+---------------------+
|                   1 |
+---------------------+
```

下面的例子演示了一个在使用BETWEEN时经常犯的一个错误。

```
mysql> SELECT 5 BETWEEN 6 and 4;
+--------------------+
| 5 BETWEEN 6 and 4 |
+--------------------+
|                  0 |
+--------------------+
```

警告： MySQL不对BETWEEN后面的两个值进行排序，因此，如果顺序错误的话，全部行的结果都会是错误的。应该确保前面的数小。

因为在字母表中，a在b之前，下面例子的结果为真。字符串比较顺序是从左到右进行，一次一个字符：

```
mysql> SELECT 'abc' < 'b';
+-------------+
| 'abc' < 'b' |
+-------------+
|           1 |
+-------------+
```

在下面的例子中，b小于等于b；但是，下一个b却不小于等于空（右字符串的第二个字符）：

```
mysql> SELECT 'bbc' <= 'b';
+--------------+
| 'bbc' <= 'b' |
+--------------+
|            0 |
+--------------+
```

函数IN()可以用来测试一个值是否与一系列值匹配。这个域可以和括号中用逗号分隔的一列值中的任意一个进行匹配，如同下面的例子所示：

```
mysql> SELECT 'a' IN ('b','c','a');
+----------------------+
| 'a' in ('b','c','a') |
+----------------------+
|                    1 |
+----------------------+
```

在SQL模式匹配中使用LIKE

在比较运算符中经常一起使用的通配符列在表3.4中。通过它们可以用来比较用户不太确定的某个字符（或系列字符）。

表3.4 通配符

字符	描述
%	任意个数的字符
_	一个字符

下面的例子演示了通配符%的用法：

```
mysql> SELECT 'abcd' LIKE '%bc%';
+--------------------+
| 'abcd' LIKE '%bc%' |
+--------------------+
|                  1 |
+--------------------+
```

通配符%可以返回任意个数的字符，因此如下的例子也可匹配：

```
mysql> SELECT 'abcd' LIKE '%b%';
+-------------------+
| 'abcd' LIKE '%b%' |
+-------------------+
|                 1 |
+-------------------+
mysql> SELECT 'abcd' LIKE 'a_ _ _';
+----------------------+
| 'abcd' LIKE 'a_ _ _' |
+----------------------+
|                    1 |
+----------------------+
```

一个下划线（_）只能匹配一个字符，因此，在上面的例子中，如果只有两个下划线，而不是三个的话，就不能匹配成功，如下：

```
mysql> SELECT 'abcd' LIKE 'a_ _';
+--------------------+
| 'abcd' LIKE 'a_ _' |
+--------------------+
|                  0 |
+--------------------+
```

规则表达式

通过规则表达式可以在MySQL中进行复杂的比较。对于很多人来说，听到规则表达式这个术语，就像中世纪的医生听到鼠疫一样——立即皱起眉头，准备好了借口，也失去了信心。它们确实很复杂——整本书都可以只讨论这个主题。但在MySQL中使用它们却不困难，而且在MySQL中对比较运算提供了一些灵活性。表3.5描述了可以在MySQL中使用的规则表达式。

表3.5 规则表达式（REGEXP, RLIKE）

字符	描述
*	与其之前的零或多个字符实例匹配
+	与之前的一个或多个字符串实例匹配
?	与其之前的零或一个字符串实例匹配
.	与任意的单个字符匹配
[xyz]	与x, y或z（括号中的字符）中的任一个匹配
[A-Z]	与任意的大写字符匹配
[a-z]	与任意的小写字符匹配
[0-9]	与任意的数字匹配
^	从开始固定此匹配
$	固定此匹配到结束
|	在规则表达式中分割字符串
{n,m}	字符串至少应该出现n次，但不能超过m次
{n}	字符串只能出现n次
{n,}	字符串至少应该出现n次

规则表达式匹配（REGEXP）可以产生与SQL匹配（LIKE）一样的结果，但其中也存在一些重要的区别。对于规则表达式，除非用户指明，会在字符串中的任意部分进行匹配。因此不需要像LIKE一样在两边使用通配符。注意下面两个结果的区别：

```
mysql> SELECT 'abcdef' REGEXP 'abc';
+-----------------------+
| 'abcdef' REGEXP 'abc' |
+-----------------------+
|                     1 |
+-----------------------+

mysql> SELECT 'abcdef' LIKE 'abc';
+---------------------+
| 'abcdef' LIKE 'abc' |
+---------------------+
|                   0 |
+---------------------+
```

为了得到与LIKE一样的结果，需要在结尾使用%通配符：

```
mysql> SELECT 'abcdef' LIKE 'abc%';
+----------------------+
| 'abcdef' LIKE 'abc%' |
+----------------------+
|                    1 |
+----------------------+
```

下面的例子与以a为第一个字符的字符串匹配：

```
mysql> SELECT 'abc' REGEXP '^a';
+--------------------+
| 'abc' REGEXP '^a' |
+--------------------+
|                  1 |
+--------------------+
```

但是下面的例子却不能匹配上，因为加号（+）表示g至少应该出现一次或多次：

```
mysql> SELECT 'abcdef' REGEXP 'g+';
+----------------------+
| 'abcdef' REGEXP 'g+' |
+----------------------+
|                    0 |
+----------------------+
```

下面的查询能够匹配上，因为星号（*）表示字符串中可以包含零个或多个g。

```
mysql> SELECT 'abcdef' REGEXP 'g*';
+----------------------+
| 'abcdef' REGEXP 'g*' |
+----------------------+
|                    1 |
+----------------------+
```

也可以使用星号来匹配ian或其替代，iain。在a后放置任意字符都将导致匹配失败。比如：

```
mysql> SELECT 'ian' REGEXP 'ia*n';
+----------------------+
| 'ian' REGEXP 'ia*n' |
+----------------------+
|                    1 |
+----------------------+
```

但问题是星号可以匹配任意个的字符，因此也可以匹配‘iaiiiin’，如下：

```
mysql> SELECT 'iaiiiiin' REGEXP 'ia*n';
+--------------------------+
| 'iaiiiiin' REGEXP 'ia*n' |
+--------------------------+
|                        1 |
+--------------------------+
```

为了防止这一点，应该限制‘i’的匹配个数为零或者是一。只要把星号改为问号就可以达到这个目的。下面的例子就只匹配‘ian’和‘iain,’而不对‘iaiiin’匹配。

```
mysql> SELECT 'iaiiiiin' REGEXP 'iai?n';
+--------------------------+
| 'iaiiiiin' REGEXP 'iai?n' |
+--------------------------+
|                        0 |
+--------------------------+
```

第一眼看到下面的例子，可能认为不能匹配，因为a匹配了四次，而{3}却表示只能匹配三次。但实际上，可以匹配三次，以及两次，一次和四次：

```
mysql> SELECT 'aaaa' REGEXP 'a{3}';
+---------------------+
| 'aaaa' REGEXP 'a{3}' |
+---------------------+
|                   1 |
+---------------------+
```

如果只想匹配aaa，应该使用下面的方法：

```
mysql> SELECT 'aaaa' REGEXP '^aaa$';
+----------------------+
| 'aaaa' REGEXP '^aaa$' |
+----------------------+
|                    0 |
+----------------------+
```

插字符（^）表示开始，美元符（$）表示结束，省略其中的任意一个，匹配也都能成功。

下面的匹配，因为{3}只针对c而不是abc，因此会失败：

```
mysql> SELECT 'abcabcabc' REGEXP 'abc{3}';
+-----------------------------+

| 'abcabcabc' REGEXP 'abc{3}' |
+-----------------------------+
|                           0 |
+-----------------------------+
```

而下面的例子却会成功：

```
mysql> SELECT 'abccc' REGEXP 'abc{3}';
+-------------------------+
| 'abccc' REGEXP 'abc{3}' |
+-------------------------+
|                       1 |
+-------------------------+
```

为了匹配abcabcabc，需要使用圆括号，如下：

```
mysql> SELECT 'abcabcabc' REGEXP '(abc){3}';
+-----------------------------+
| 'abcabcabc' REGEXP '(abc){3}' |
+-----------------------------+
|                           1 |
+-----------------------------+
```

注意下面例子中的中括号与大括号的区别。大括号把abc作为一组，当做一个整体，而中括号却容许a，或b，或c中的任意一个以及它们的任意一个组合进行匹配，例示如下：

```
mysql> SELECT 'abcbbcccc' REGEXP '[abc]{3}';
+-----------------------------+
| 'abcbbcccc' REGEXP '[abc]{3}' |
+-----------------------------+
|                           1 |
+-----------------------------+
```

下面的例子使用圆括号来达到同样的效果，其中使用竖线来分组可替换的子串：

```
mysql> SELECT 'abcbbcccc' REGEXP '(a|b|c){3}';
+-------------------------------+
| 'abcbbcccc' REGEXP '(a|b|c){3}' |
+-------------------------------+
|                             1 |
+-------------------------------+
```

位运算符

为了理解位运算是如何工作的，需要了解一些布尔数和布尔运算的知识。这些类型的查询不太经常使用，但在任意的“guru2be”中都把它们作为保留节目。表3.6描述了位运算符。

表3.6 位运算

运算符	语法	描述
&	A & b	位与
|	a | b	位或
<<	a << b	a左 b位移
>>	a >> b	a右移b位

普通的数位系统，即十进制系统，以10作为基。人有10个指头，因此它听起来很合理。从0数到9，然后就是10，移到10那一列，然后又从0开始：

00 01 02 03 04 05 06 07 08 09 10

十进制系统包含0到9共10个数。但在实际中，人们发现基于两个数0和1的二进制系统对于计算机系统来说更有用。它们代表了电子连接的两个状态，on和off：

00 01 10 11

在十进制中，用完全部的数字之后（9之后为10），进入10进位，而在二进制中，1之后为10（发音为1-0，以避免与十进制中的数混淆）。

在十进制中，高位的大小按照10的冥次的方式增加，如图3.2所示。

10^6	10^5	10^4	10^3	10^2	10^1	10^0
百万	十万	万	千	百	十	个
4	3	9	2	4	2	1

图3.2 10的冥次

图中的数，读做4百3十9万2千4百2十1也可以显示为：

4 * 100 000 000 +
3 * 100 000 +
9 * 10 000 +
2 * 1000 +
4 * 100 +
2 * 10 +
1 * 1

如果遵从上面的方式（假设是一个孩子学习十进制记数），就可以把上面的概念应用到二进制数上。

在二进制中，高位大小按照2的冥次增加，如图3.3所示。

2^6	2^5	2^4	2^3	2^2	2^1	2^0
64s	32s	16s	8s	4s	2s	1s
1	1	1	1	1	1	1

图3.3 2的冥次

如果把前面的二进制数——1111111——转换为十进制，如下：

1 * 64 +
1 * 32 +
1 * 16 +
1 * 8 +
1 * 4 +
1 * 2 +
1 * 1

等同于64 + 32 + 16 + 8 + 4 + 2 + 1，即127。

同样，二进制数101001即为1 * 1 + 1 * 8 + 1 * 32 = 41。

因此，把二进制数转换为十进制数非常容易，但反过来，把十进制数转换为二进制数又怎么样呢？同样很简单。把18转换为二进制数，可以从图3.4开始。

2^6	2^5	2^4	2^3	2^2	2^1	2^0
64s	32s	16s	8s	4s	2s	1s

图3.4 第一步，画出列

从左开始，显然在18中不存在64和32。但在18中却存在一个16。因此在标明为16的列写上1，如图3.5.所示。

2^6	2^5	2^4	2^3	2^2	2^1	2^0
64s	32s	16s	8s	4s	2s	1s
0	0	1				

图3.5 第二步，填值

因为已经记下了18中的16，因此需要从18中减去16，剩下2。继续右边的位，在2中没有8，4，但有一个2。因为2减去2后为0，在2的列写下1后就可以停止了，如图3.6所示。

2^6	2^5	2^4	2^3	2^2	2^1	2^0
64s	32s	16s	8s	4s	2s	1s
0	0	1	0	0	1	0

图3.6 第三步，十进制转为二进制

在二进制中，18就是10010。对于更大的数，只需要在左边放更多的列即可（表示128，256等等）。二进制数很可能很快就变得很长，而这就是不用二进制来存储数的原因。八进制（以8为基）和十六进制（以16为基）是其他的两种更方便的数制系统。

再回到位运算中，考虑两个数，9和7。在二进制中，它们表示为1001和111。位运算作用于二进制数的单个位上。

对于位与（AND）运算，当两个位都是1的时候，结果才为1（如同普通的AND）。图3.7显示了这两个二进制数：

1	0	0	1
0	1	1	1
0	**0**	**0**	**1**

图3.7 位与运算：9&7

从左开始，1 AND 0为0，因此最左列（第八列）为0。右移，0 AND 1为0，再右移，0 AND 1为0。只在最右列，1 AND 1，得到1。

因此，7和9位与的结果为1！如下：

```
mysql> SELECT 9&7;
+-----+
```

```
| 9&7 |
+-----+
|   1 |
+-----+
```

说明：位与与数的前后顺序没有关系——换句话说，9&7等同于7&9。

对于位或，只要一位为1，结果就是1。图3.8显示的是在同样的9和7之间的位或：

1	0	0	1
0	1	1	1
1	1	1	1

图3.8 位或运算9|7

全部的列中都至少有一个1，因此结果为全部1。二进制的1111等同于15。

```
mysql> SELECT 9|7;
+-----+
| 9|7 |
+-----+
|  15 |
+-----+
```

<<为左移运算，因此，<< b表示a的位左移b列。比如，2 << 1。在二进制中2即10。如果左移一位，得到100，即4。如下：

```
mysql> SELECT 2 << 1;
+--------+
| 2 << 1 |
+--------+
|      4 |
+--------+

mysql> SELECT 15 << 4;
+---------+
| 15 << 4 |
+---------+
|     240 |
+---------+
```

15即为1111，左移4位后，得到11110000。用一般的方法转换为十进制，如图3.9所示。

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128s	64s	32s	16s	8s	4s	2s	1s
1	1	1	1	0	0	0	0

图3.9 把二进制数11110000转换为十进制数

总数就是：

128 + 64 + 32 + 16 = 240

位运算按照BIG INT进行，就是数位限制为64位。超过64位的移动，或者是对负数的移动，返回0。如下：

```
mysql> SELECT 3 << 64;
+---------+
| 3 << 64 |
+---------+
|       0 |
+---------+
```

>>为右移运算符，a >> b表示把a右移b列。一列以外的位丢失。同样，对负数进行移动，也返回0。比如：

```
mysql> SELECT 3 >> 1;
+--------+
| 3 >> 1 |
+--------+
|      1 |
+--------+
```

在二进制中，3就是11，右移1位，有一个1超过了一列（也可以说是1.1，尽管在二进制的表述中不存在小数点）。因为处理的是整数，小数点右边的数就被丢去了（可能应该称为二进制点，可能在好莱坞电影中会有这样的名字），最后剩下一个1（二进制和十进制都是）。比如：

```
mysql> SELECT 19 >> 3;
+---------+
| 19 >> 3 |
+---------+
|       2 |
+---------+
```

在上面的例子中，19即10011，右移3位变为10，011被丢弃了，10就是十进制的2。

```
mysql> SELECT 4 >> 3;
+--------+
| 4 >> 3 |
+--------+
|      0 |
+--------+
```

这个例子右移太多，丢失了全部的位。

高级连接

在第1章中已经看到一个基本的两张表连接的例子。但实际上，连接可以比那个例子复杂的多，而写得不好的连接会导致严重的效率问题。

回到前面一章创建的表中。如果略过了前面一章，可以用如下的语法重新创建一个：

```
CREATE TABLE customer (
  id int(11) default NULL,
  first_name varchar(30) default NULL,
  surname varchar(40) default NULL
) TYPE=MyISAM;

INSERT INTO customer VALUES (1, 'Yvonne', 'Clegg');
INSERT INTO customer VALUES (2, 'Johnny', 'Chaka-Chaka');
INSERT INTO customer VALUES (3, 'Winston', 'Powers');
INSERT INTO customer VALUES (4, 'Patricia', 'Mankunku');

CREATE TABLE sales (
  code int(11) default NULL,
  sales_rep int(11) default NULL,
  customer int(11) default NULL,
  value int(11) default NULL
) TYPE=MyISAM;

INSERT INTO sales VALUES (1, 1, 1, 2000);
INSERT INTO sales VALUES (2, 4, 3, 250);
INSERT INTO sales VALUES (3, 2, 3, 500);
INSERT INTO sales VALUES (4, 1, 4, 450);
INSERT INTO sales VALUES (5, 3, 1, 3800);
INSERT INTO sales VALUES (6, 1, 2, 500);

CREATE TABLE sales_rep (
  employee_number int(11) default NULL,
  surname varchar(40) default NULL,
  first_name varchar(30) default NULL,
  commission tinyint(4) default NULL,
  date_joined date default NULL,
  birthday date default NULL
) TYPE=MyISAM;

INSERT INTO sales_rep VALUES (1, 'Rive', 'Sol', 10,
 '2000-02-15', '1976-03-18');
INSERT INTO sales_rep VALUES (2, 'Gordimer', 'Charlene', 15,
 '1998-07-09', '1958-11-30');
INSERT INTO sales_rep VALUES (3, 'Serote', 'Mike', 10,
 '2001-05-14', '1971-06-18');
```

```
INSERT INTO sales_rep VALUES (4, 'Rive', 'Mongane', 10,
 '2002-11-23', '1982-01-04');
```

从一个基本的连接开始：

```
mysql> SELECT sales_rep, customer,value, first_name,surname
 FROM sales, sales_rep WHERE code=1 AND
 sales_rep.employee_number=sales.sales_rep;
+-----------+----------+-------+------------+---------+
| sales_rep | customer | value | first_name | surname |
+-----------+----------+-------+------------+---------+
|         1 |        1 |  2000 | Sol        | Rive    |
+-----------+----------+-------+------------+---------+
```

因为表sales_rep和sales之间的连接处是域employee_number或sales_rep，这两个域形成了WHERE从句的连接条件。

通过三张表来形成一个复杂的连接也不是特别困难。如果想返回sales rep和customer两者的first names和surnames，以及sale的值，可以使用下面的查询：

```
mysql> SELECT sales_rep.first_name,sales_rep.surname,
 value,customer.first_name, customer.surname FROM
 sales,sales_rep,customer WHERE sales_rep.employee_number =
 sales.sales_rep AND customer.id = sales.customer;
+------------+----------+-------+------------+-------------+
| first_name | surname  | value | first_name | surname     |
+------------+----------+-------+------------+-------------+
| Sol        | Rive     |  2000 | Yvonne     | Clegg       |
| Mike       | Serote   |  3800 | Yvonne     | Clegg       |
| Sol        | Rive     |   500 | Johnny     | Chaka-Chaka |
| Charlene   | Gordimer |   500 | Winston    | Powers      |
| Mongane    | Rive     |   250 | Winston    | Powers      |
| Sol        | Rive     |   450 | Patricia   | Mankunku    |
+------------+----------+-------+------------+-------------+
```

sales_rep表的employee_number域与sales表的sales_rep域相关。

而customer表的id域与sales表的customer域相关。因此，这些构成了WHERE从句的两个连接条件。由于没有其他的条件，这个查询就列举了在表sales_rep和customer中与sales相关的全部行。

内连接

内连接就是前面学到的连接的另一种描述。下面的两个查询是等同的：

```
mysql> SELECT first_name,surname,value FROM customer,sales WHERE
 id=customer;
+------------+-------------+-------+
| first_name | surname     | value |
+------------+-------------+-------+
```

```
| Yvonne     | Clegg       |  2000 |
| Winston    | Powers      |   250 |
| Winston    | Powers      |   500 |
| Patricia   | Mankunku    |   450 |
| Yvonne     | Clegg       |  3800 |
| Johnny     | Chaka-Chaka |   500 |
+------------+-------------+-------+
6 rows in set (0.00 sec)

mysql> SELECT first_name,surname,value FROM customer INNER JOIN sales
 ON id=customer;
+------------+-------------+-------+
| first_name | surname     | value |
+------------+-------------+-------+
| Yvonne     | Clegg       |  2000 |
| Winston    | Powers      |   250 |
| Winston    | Powers      |   500 |
| Patricia   | Mankunku    |   450 |
| Yvonne     | Clegg       |  3800 |
| Johnny     | Chaka-Chaka |   500 |
+------------+-------------+-------+
```

左连接（左外连接）

假设完成了另外一次销售。但这次是一个现金销售，而自己忘记了获取用户的详细信息，而且在自己意识到这点之前，用户已经拿着货物离开了。这也不会有问题，因为还可以往sales表中给用户添加一个NULL值。

```
mysql> INSERT INTO sales(code,sales_rep,customer,value) VALUES
 (7, 2,NULL,670);
```

执行查询，就返回sales reps和customers的value和names：

```
mysql> SELECT sales_rep.first_name, sales_rep.surname, value,
 customer.first_name, customer.surname FROM sales,sales_rep,
 customer WHERE sales_rep.employee_number = sales.sales_rep
 AND customer.id = sales.customer;
+------------+----------+-------+------------+-------------+
| first_name | surname  | value | first_name | surname     |
+------------+----------+-------+------------+-------------+
| Sol        | Rive     |  2000 | Yvonne     | Clegg       |
| Mike       | Serote   |  3800 | Yvonne     | Clegg       |
| Sol        | Rive     |   500 | Johnny     | Chaka-Chaka |
| Charlene   | Gordimer |   500 | Winston    | Powers      |
| Mongane    | Rive     |   250 | Winston    | Powers      |
| Sol        | Rive     |   450 | Patricia   | Mankunku    |
+------------+----------+-------+------------+-------------+
```

发生了什么？新的销售在哪里？问题出在这里，sales表中的customer为空，连接条件不完整。还记得吗，在本章的前面，学习运算符时，已经见到=运算符不包括NULL值。因为在customer表中没有NULL记录，<=>运算符也没有帮助，因此，用对null友好的等于来检查也没有用。

解决这个问题的办法就是使用外连接。这会返回表中匹配的每一个结果，而不管另一个表中是否有相关的记录。因此，即使sales表中的customer域为空并且与customer表没有关系，也会返回一个记录。左外连接就是返回左边的匹配行，不考虑右边的表是否有相应的行。左连接（左外连接的简称）的语法如下：

```
SELECT field1, field2 FROM table1 LEFT JOIN table2 ON field1=field2
```

先在customer和sales表中试一个简单的例子，来进行下面左连接。

```
mysql> SELECT first_name,surname,value FROM sales LEFT JOIN customer
 ON id=customer;
+------------+-------------+-------+
| first_name | surname     | value |
+------------+-------------+-------+
| Yvonne     | Clegg       |  2000 |
| Winston    | Powers      |   250 |
| Winston    | Powers      |   500 |
| Patricia   | Mankunku    |   450 |
| Yvonne     | Clegg       |  3800 |
| Johnny     | Chaka-Chaka |   500 |
| NULL       | NULL        |   670 |
+------------+-------------+-------+
```

七个记录都如同期望的那样返回来了。

在左连接中，表的顺序非常重要。返回匹配的全部行的表一定是左表（LEFT JOIN关键字之前）。如果颠倒了顺序，如下：

```
mysql> SELECT first_name,surname,value FROM customer LEFT JOIN sales
 ON id=customer;
+------------+-------------+-------+
| first_name | surname     | value |
+------------+-------------+-------+
| Yvonne     | Clegg       |  2000 |
| Yvonne     | Clegg       |  3800 |
| Johnny     | Chaka-Chaka |   500 |
| Winston    | Powers      |   250 |
| Winston    | Powers      |   500 |
| Patricia   | Mankunku    |   450 |
+------------+-------------+-------+
```

只看到六个记录。因为在这个查询中，左表为customer，而连接只匹配存在于左表中的记录，NULL客户（表示与customer表没有关系）的sales记录是不被返回的。

说明：左连接在过去经常被叫做左外连接。为熟悉起见，MySQL中也采用了这样的术语。

当然，可以把这个扩展到第三张表，作为前面查询的答案（customers和sales reps的名字，以及每一个销售的sales值）。看一看自己能不能做，下面是作者的建议：

```
mysql> SELECT sales_rep.first_name, sales_rep.surname, value,
 customer.first_name, customer.surname FROM sales LEFT JOIN
 sales_rep ON sales_rep.employee_number = sales.sales_rep
 LEFT JOIN customer ON customer.id = sales.customer;
+------------+----------+-------+------------+-------------+
| first_name | surname  | value | first_name | surname     |
+------------+----------+-------+------------+-------------+
| Sol        | Rive     |  2000 | Yvonne     | Clegg       |
| Mongane    | Rive     |   250 | Winston    | Powers      |
| Charlene   | Gordimer |   500 | Winston    | Powers      |
| Sol        | Rive     |   450 | Patricia   | Mankunku    |
| Mike       | Serote   |  3800 | Yvonne     | Clegg       |
| Sol        | Rive     |   500 | Johnny     | Chaka-Chaka |
| Charlene   | Gordimer |   670 | NULL       | NULL        |
+------------+----------+-------+------------+-------------+
```

右连接（右外连接）

右连接与左连接除了连接的顺序相反以外，其他的完全相同。为了返回sale中全部用户的名字，包括没有相应的客户数据的那些sales，可以把sales表放在连接的右边：

```
mysql> SELECT first_name,surname,value FROM customer RIGHT JOIN
 sales ON id=customer;
+------------+-------------+-------+
| first_name | surname     | value |
+------------+-------------+-------+
| Yvonne     | Clegg       |  2000 |
| Winston    | Powers      |   250 |
| Winston    | Powers      |   500 |
| Patricia   | Mankunku    |   450 |
| Yvonne     | Clegg       |  3800 |
| Johnny     | Chaka-Chaka |   500 |
| NULL       | NULL        |   670 |
+------------+-------------+-------+
```

提示：如果混淆了应该把哪张表放在左或右连接的哪一边，只要记住右连接读取右边的表的全部记录，包括null，而左连接读取左边的表的全部记录，包括null，就可以了。

全外连接

在写本书的时候，MySQL还不支持全外连接。所谓的那种连接，就是第一个表中的每一个记录，包括与第二个表中没有匹配的记录以及第二个表中的每一个记录，包括与第一个表中没有匹配的记录都会返回的连接。这和左连接和右连接等同。注意最近的文档，MySQL

很快就会支持这个。全外连接的语法与其他的连接一样。

```
SELECT field1,field2 FROM table1 FULL OUTER JOIN table2
```

自然连接和USING关键字

customer表中的id域与sales表中的customer域是相关的。如果它们的名字相同，SQL有一些快捷键可以使JOIN语句更简洁。现在把sales.customer改为sales.id来演示：

```
mysql> ALTER TABLE sales CHANGE customer id INT;
```

因为这两张表有相同名字的域，可以进行自然连接，它们在进行连接时查找相同名字的域：

```
mysql> SELECT first_name,surname,value FROM customer NATURAL JOIN
 sales;
+------------+-------------+-------+
| first_name | surname     | value |
+------------+-------------+-------+
| Yvonne     | Clegg       |  2000 |
| Winston    | Powers      |   250 |
| Winston    | Powers      |   500 |
| Patricia   | Mankunku    |   450 |
| Yvonne     | Clegg       |  3800 |
| Johnny     | Chaka-Chaka |   500 |
+------------+-------------+-------+
```

这与下面的语句完全相同：

```
mysql> SELECT first_name,surname,value FROM customer INNER JOIN
 sales ON customer.id=sales.id;
+------------+-------------+-------+
| first_name | surname     | value |
+------------+-------------+-------+
| Yvonne     | Clegg       |  2000 |
| Winston    | Powers      |   250 |
| Winston    | Powers      |   500 |
| Patricia   | Mankunku    |   450 |
| Yvonne     | Clegg       |  3800 |
| Johnny     | Chaka-Chaka |   500 |
+------------+-------------+-------+
```

这里在两张表中只有一个相同的域，如果有更多的话，每一个都会成为连接条件的一部分。

自然连接也可以是左或右连接，下面的两个语句是相同的：

```
mysql> SELECT first_name,surname,value FROM customer LEFT JOIN sales
 ON customer.id=sales.id;
+------------+-------------+-------+
```

```
| first_name | surname      | value |
+------------+--------------+-------+
| Yvonne     | Clegg        |  2000 |
| Yvonne     | Clegg        |  3800 |
| Johnny     | Chaka-Chaka  |   500 |
| Winston    | Powers       |   250 |
| Winston    | Powers       |   500 |
| Patricia   | Mankunku     |   450 |
+------------+--------------+-------+

mysql> SELECT first_name,surname,value FROM customer NATURAL LEFT JOIN sales;
+------------+--------------+-------+
| first_name | surname      | value |
+------------+--------------+-------+
| Yvonne     | Clegg        |  2000 |
| Yvonne     | Clegg        |  3800 |
| Johnny     | Chaka-Chaka  |   500 |
| Winston    | Powers       |   250 |
| Winston    | Powers       |   500 |
| Patricia   | Mankunku     |   450 |
+------------+--------------+-------+
```

关键字USING需要比自然连接更多的控制。如果在两张表中有超过一个的相同域，这个关键字就可以用来指定连接条件中使用哪个域。比如，考虑两张表A和B，它们有相同的域a, b, c, d, 下面的语句是相同的：

```
SELECT * FROM A LEFT JOIN B USING (a,b,c,d)
SELECT * FROM A NATURAL LEFT JOIN B
```

关键字USING可以更灵活，因为它可以用来选择四个相同域中的某些域。比如：

```
SELECT * FROM A LEFT JOIN B USING (a,d)
```

说明： 为了自然连接，相同是指名字相同，而不是类型。两个id域可以是INT和DECIMAL，或者甚至是INT和VARCHAR，只要它们有相同的名字就行。

返回在一张表中而不在另一张表中存在的结果

迄今为止，通过内连接，可以返回在两张表中都出现了的行。通过外连接，也可以返回在另一张表中不存在对应匹配记录的记录。但经常也需要返回在一张表中存在而在另一张表中不存在的记录。为了演示这一点，先添加一个新的sales_rep：

```
mysql> INSERT INTO sales_rep VALUES(5, 'Jomo', 'Igneasund', 10,
 '2002-11-29', '1968-12-01');
```

现在，做一个内连接，就会返回完成一次销售的全部sales reps：

```
mysql> SELECT DISTINCT first_name,surname FROM sales_rep
 INNER JOIN sales ON sales_rep=employee_number;
+------------+----------+
```

```
| first_name | surname   |
+------------+-----------+
| Sol        | Rive      |
| Mongane    | Rive      |
| Charlene   | Gordimer  |
| Mike       | Serote    |
+------------+-----------+
```

在查询中使用了DISTINCT关键字以避免重复行，因为有sales reps完成了不止一次销售。

但反过来也很有用。老板正在发火，头儿也很担心。哪个sales reps没有完成任何销售？可以通过查看出现在sales_rep表中而在sales表中却没有相应项的sales reps来了解这个信息。

```
mysql> SELECT first_name,surname FROM sales_rep LEFT JOIN sales
 ON sales_rep=employee_number WHERE sales_rep IS NULL;
+------------+---------+
| first_name | surname |
+------------+---------+
| Ignesund   | Jomo    |
+------------+---------+
```

必须使用左连接（外连接，而不是内连接），因为外连接只返回在第一张表中不存在相应记录（或null值）的记录。

使用UNION连接结果

MySQL 4介绍了ANSI SQL中长等待的UNION语句，它是用来把不同的SELECT的结果连接成一个的。每一个语句都必须有相同个数的列。

为了了解这个语句的使用，再创建另一张表，它包含从存储的前一个拥有者传递过来的用户列表。

```
mysql> CREATE TABLE old_customer(id int, first_name varchar(30),
 surname varchar(40));
mysql> INSERT INTO old_customer VALUES (5432, 'Thulani', 'Salie'),
 (2342, 'Shahiem', 'Papo');
```

现在，为了得到旧的和新的用户列表，可以使用下面的方法：

```
mysql> SELECT id, first_name, surname FROM old_customer UNION SELECT
 id, first_name,surname FROM customer;
+------+------------+-------------+
| id   | first_name | surname     |
+------+------------+-------------+
| 5432 | Thulani    | Salie       |
| 2342 | Shahiem    | Papo        |
|    1 | Yvonne     | Clegg       |
|    2 | Johnny     | Chaka-Chaka |
|    3 | Winston    | Powers      |
|    4 | Patricia   | Mankunku    |
+------+------------+-------------+
```

也可以用一般的方式组织输出。只是需要注意ORDER BY从句是用在整个UNION上还是仅仅用在了SELECT上：

```
mysql> SELECT id, first_name, surname FROM old_customer UNION SELECT
 id, first_name, surname FROM customer ORDER BY surname,first_name;
+------+------------+-------------+
| id   | first_name | surname     |
+------+------------+-------------+
|    2 | Johnny     | Chaka-Chaka |
|    1 | Yvonne     | Clegg       |
|    4 | Patricia   | Mankunku    |
| 2342 | Shahiem    | Papo        |
|    3 | Winston    | Powers      |
| 5432 | Thulani    | Salie       |
+------+------------+-------------+
```

排序是在整个UNION上进行的。如果只想在第二个SELECT上进行，需要使用括号：

```
mysql> SELECT id, first_name, surname FROM old_customer UNION
 (SELECT id, first_name, surname FROM customer ORDER BY surname,
 first_name);
+------+------------+-------------+
| id   | first_name | surname     |
+------+------------+-------------+
| 5432 | Thulani    | Salie       |
| 2342 | Shahiem    | Papo        |
|    2 | Johnny     | Chaka-Chaka |
|    1 | Yvonne     | Clegg       |
|    4 | Patricia   | Mankunku    |
|    3 | Winston    | Powers      |
+------+------------+-------------+
```

提示： 就像在排序的时候，在可能存在两义性的时候，都应该使用括号。它可以保证对正确的部分进行排序，同时也使其余的人能够比较容易地理解这条语句。不要以为别人都和自己一样了解情况！

默认情况下，UNION不返回重复的记录（与DISTINCT关键字相似）。可以通过使用ALL关键字来重载这个特性，以使其返回全部的结果：

```
mysql> SELECT id FROM customer UNION ALL SELECT id FROM sales;
+------+
| id   |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    1 |
```

```
|    3 |
|    3 |
|    4 |
|    1 |
|    2 |
| NULL |
+------+
```

还需要思考一下UNION。只要每个SELECT返回的域的个数和数据类型相同，就可以很容易地把毫不相关的域组织在一起。MySQL会非常乐意返回这些值，即使它们毫无意义：

```
mysql> SELECT id, surname FROM customer UNION ALL SELECT value,
 sales_rep FROM sales;
+------+-------------+
| id   | surname     |
+------+-------------+
|    1 | Clegg       |
|    2 | Chaka-Chaka |
|    3 | Powers      |
|    4 | Mankunku    |
| 2000 | 1           |
|  250 | 4           |
|  500 | 2           |
|  450 | 1           |
| 3800 | 3           |
|  500 | 1           |
|  670 | 2           |
+------+-------------+
```

子选择

很多查询都是在SELECT中使用SELECT。子选择预计在版本4.1中实现。到现在为止，MySQL还不允许子选择，部分是设计的原因（一般都比后面介绍的替代方法效率低），部分是因为在1001表中处于低位，还有其他的更重要事情需要实现。讲到MySQL的实现，需要看看它们是如何工作的。

通过连接来重写子选择

下面考虑一个需要返回销售额大于1000美元的全部sales reps的查询。如果能够执行子选择，试一试下面的语句：

```
mysql> SELECT first_name,surname FROM sales_rep WHERE
 sales_rep.employee_number IN (SELECT code FROM sales WHERE
 value>1000);
+------------+---------+
| first_name | surname |
+------------+---------+
```

```
| Sol         | Rive     |
+-------------+----------+
```

只有Sol Rive满足这个条件。

查询先分解为内选择——就是先执行下面的一步：

```
mysql> SELECT id FROM sales WHERE value>1000;
+------+
| id   |
+------+
|    1 |
|    1 |
+------+
```

然后是剩余的部分：

```
mysql> SELECT first_name, surname FROM sales_rep WHERE
 sales_rep.employee_number IN (1);
+-------------+----------+
| first_name | surname |
+-------------+----------+
| Sol         | Rive     |
+-------------+----------+
```

但实际上我们已经知道有一个更好的方法来完成这个任务，就是使用连接：

```
mysql> SELECT DISTINCT first_name,surname FROM sales_rep INNER JOIN sales ON
 employee_number=id WHERE value>1000;
+-------------+----------+
| first_name | surname |
+-------------+----------+
| Sol         | Rive     |
| Sol         | Rive     |
+-------------+----------+
```

或者是：

```
mysql> SELECT DISTINCT first_name,surname FROM sales_rep,sales WHERE
 sales.id=sales_rep.employee_number AND value>1000;
+-------------+----------+
| first_name | surname |
+-------------+----------+
| Sol         | Rive     |
| Sol         | Rive     |
+-------------+----------+
```

之所以说更好是因为连接在查询中效率更高，能够更快地返回结果。对一个极小的数据库来说，似乎没有什么差别，但对于一个大的、使用频繁的表来说，效率是致命的问题，可能需要减少额外的任意一个微秒的时间。

返回还没有销售业绩的全部sales reps，如果DBMS容许的话，可以使用下面的子选择：

```
mysql> SELECT first_name,surname FROM sales_rep WHERE employee_number
 NOT IN (SELECT DISTINCT code from sales);
+------------+---------+
| first_name | surname |
+------------+---------+
| Ignesund   | Jomo    |
+------------+---------+
```

但下面的方法更好：

```
mysql> SELECT DISTINCT first_name,surname FROM sales_rep LEFT JOIN sales ON
 sales_rep=employee_number WHERE sales_rep IS NULL;
+------------+---------+
| first_name | surname |
+------------+---------+
| Ignesund   | Jomo    |
+------------+---------+
```

使用INSERT SELECT从其他表中添加记录到一个表中

INSERT语句也可以用来添加记录或者是部分记录到一个已经存在的表中。比如，假设需要创建一个包含用户名和他们所购货物的价值的新表。返回需要的结果的查询如下：

```
mysql> SELECT first_name,surname,SUM(value) FROM sales NATURAL JOIN
 customer GROUP BY first_name, surname;
+------------+-------------+------------+
| first_name | surname     | SUM(value) |
+------------+-------------+------------+
| Johnny     | Chaka-Chaka |        500 |
| Patricia   | Mankunku    |        450 |
| Winston    | Powers      |        750 |
| Yvonne     | Clegg       |       5800 |
+------------+-------------+------------+
```

首先需要创建一个存储结果的新表：

```
mysql> CREATE TABLE customer_sales_values(first_name
 VARCHAR(30), surname VARCHAR(40), value INT);
```

现在把结果插入这个表中：

```
mysql> INSERT INTO customer_sales_values(first_name,surname,value)
 SELECT first_name,surname, SUM(value) FROM sales NATURAL JOIN
 customer GROUP BY first_name, surname;
```

现在，customer_sales_values表中包含下列信息：

```
mysql> SELECT * FROM customer_sales_values;
+------------+-------------+-------+
| first_name | surname     | value |
+------------+-------------+-------+
| Johnny     | Chaka-Chaka |   500 |
| Patricia   | Mankunku    |   450 |
| Winston    | Powers      |   750 |
| Yvonne     | Clegg       |  5800 |
+------------+-------------+-------+
```

有关添加记录的更多信息

INSERT也接受在UPDATE语句中使用的相似的语法。下面的语句：

```
mysql> INSERT INTO customer_sales_values(first_name, surname, value)
 VALUES('Charles', 'Dube', 0);
```

等同于如下的语句：

```
mysql> INSERT INTO customer_sales_values SET first_name =
 'Charles', surname='Dube', value=0;
```

在添加记录的时候，也可以进行一些有限的计算。为了演示这一点，在customer_sales_value表中增加另一个域：

```
mysql> ALTER TABLE customer_sales_values ADD value2 INT;
```

现在可以在这个表中添加记录，把value2的值增加2倍：

```
mysql> INSERT INTO customer_sales_values(first_name, surname, value,
 value2) VALUES('Gladys', 'Malherbe', 5, value*2);
```

现在这个记录包含下面的内容：

```
mysql> SELECT * FROM customer_sales_values WHERE first_name='Gladys';
+------------+----------+-------+--------+
| first_name | surname  | value | value2 |
+------------+----------+-------+--------+
| Gladys     | Malherbe |     5 |     10 |
+------------+----------+-------+--------+
```

有关删除记录的更多信息（DELETE和TRUNCATE）

前面已经知道如何使用DELETE语句来删除一个记录。可能已经注意到在不使用WHERE从句的情况下，有一个警告，即将删除全部记录。用那种方式删除全部记录的一个问题就是对于一个大表来说，会非常慢。幸运的是，还有更好的方式。

我们先使用DELETE语句来删除customer_sales_value表中的全部记录：

```
mysql> DELETE FROM customer_sales_values;
Query OK, 7 rows affected (0.00 sec)
```

更快的方法是使用TRUNCATE。先把记录加进去，然后使用TRUNCATE语句：

```
mysql> INSERT INTO customer_sales_values(first_name, surname, value,
 value2) VALUES('Johnny', 'Chaka-Chaka', 500, NULL),('Patricia',
 'Mankunku', 450, NULL), ('Winston', 'Powers', 750, NULL),('Yvonne',
 'Clegg', 5800, NULL), ('Charles', 'Dube', 0, NULL), ('Charles',
 'Dube', 0, NULL), ('Gladys', 'Malherbe', 5, 10);

mysql> TRUNCATE customer_sales_values;
Query OK, 0 rows affected (0.00 sec)
```

注意两个语句输出的区别。DELETE会告诉删除了多少行，而TRUNCATE却没有；TRUNCATE只删除而不统计它们。实际上，它是删除表后再重建它。

用户变量

MySQL有一个特性，即容许用户把一个值存储为临时变量，以供后面的语句使用。在大多数情况下，都需要使用编程语言来做这种事（见第5章“MySQL编程”），但MySQL变量在MySQL命令行下也非常有用。

变量的值通过SET语句来设置或者是在SELECT语句中使用:=设置。查看佣金数大于平均值的全部sales reps，可以使用如下的语句：

```
mysql> SELECT @avg := AVG(commission) FROM sales_rep;
+-------------------------+
| @avg := AVG(commission) |
+-------------------------+
|                 11.0000 |
+-------------------------+

mysql> SELECT surname,first_name FROM sales_rep WHERE commission>@avg;
+----------+------------+
| surname  | first_name |
+----------+------------+
| Gordimer | Charlene   |
+----------+------------+
```

at (@)符号标明一个MySQL变量。平均佣金存储在变量@avg中，它可以在后面的存储中存取。

也可以特地设定一个变量。比如，为了在需要的时候不必重复计算四个复杂的计算，可以在前面设置一个变量：

```
mysql> SET @result = 22/7*33.23;

mysql> SELECT @result;
+-----------------+
```

```
| @result          |
+------------------+
| 104.43714285714 |
+------------------+
```

用户变量可以是字符串，整数或者是浮点数。它们可以被设置为一个表达式（不包括需要文字值的地方，比如在LIMIT从句中）。它们还不能用来代替查询的某部分，比如代替表名。下面是一个例子：

```
mysql> SET @t = 'sales';
mysql> SELECT * FROM @t;
ERROR 1064: You have an error in your SQL syntax near '@t' at line 1

mysql> SET @v=2;
mysql> SELECT * FROM sales LIMIT 0,@v;
ERROR 1064: You have an error in your SQL syntax near '@v' at line 1
```

用户变量被设置在一个特殊的线程中（或到服务器的连接中），并且不能被其余的线程存取。当线程关闭或连接中断的时候，变量也就被重置了。

从第一个线程Window1中执行下面的语句：

```
mysql> SET @a = 1;

mysql> SELECT @a;
+------+
| @a   |
+------+
|    1 |
+------+
```

不能在另一个线程中访问这个变量。在Window2中执行下面的语句：

```
mysql> SELECT @a;
+------+
| @a   |
+------+
| NULL |
+------+
```

如果关闭连接并重新在Window1中进行连接，会发现MySQL已经从Window1中删除了这个变量，如下：

```
mysql> exit

% mysql firstdb
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14 to server version: 4.0.1-alpha-max

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT @a;
```

```
+------+
| @a   |
+------+
| NULL |
+------+
```

注意在SELECT语句中，首先进行WHERE从句的计算然后才是域列表。如果没有返回记录，那个语句就没有设置用户变量。比如，因为下面的语句中没有返回记录，所以也就没有被设置用户变量：

```
mysql> SELECT @a:=2 FROM sales WHERE value>10000;
Empty set (0.00 sec)

mysql> SELECT @a;
+------+
| @a   |
+------+
| NULL |
+------+
```

但如果返回了至少一个记录，用户变量就会被正确设置：

```
mysql> SELECT @a:=2 FROM sales WHERE value>2000;
+--------+
| @a:=2  |
+--------+
|      2 |
+--------+

mysql> SELECT @a;
+------+
| @a   |
+------+
| 2    |
+------+
```

同样，在域列表中设置的用户变量也不能作为条件。下面的语句因为在使用条件的时候用户变量还没有设置，因此是不能正常工作的：

```
mysql> SELECT @d:=2000,value FROM sales WHERE value>@d;
Empty set (0.00 sec)
```

应该在查询之前先设置变量，如下：

```
mysql> SET @d=2000;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @d,value FROM sales WHERE value>@d;
+------+-------+
| @d   | value |
```

```
+------+-------+
| 2000 |  3800 |
+------+-------+
```

也可以在WHERE从句中设置变量。注意如果那样的话，就不能在域列表中正确反映结果，除非重新设置变量！例如：

```
mysql> SELECT @e,value FROM sales WHERE value>(@e:=2000);
+------+-------+
| @e   | value |
+------+-------+
| NULL |  3800 |
+------+-------+
```

为了正确反映结果，应该在域列表中重置合格变量：

```
mysql> SELECT @f:=2000,value FROM sales WHERE value>(@f:=2000);
+----------+-------+
| @f:=2000 | value |
+----------+-------+
|     2000 |  3800 |
+----------+-------+
```

这种实现用户变量的方式不太简洁，应该分开设置它们。

警告：用户变量的生命周期等同与线程。如果忘记了用户变量的初始化，就会得到不正确的结果。

执行存储在文件中的SQL语句

SQL语句组通常存储在文件中以便重复使用。这样就可以在操作系统的命令行中容易地执行它们。这被称为用批处理模式执行MySQL（与交互式的连接到服务器，然后自己执行命令相对应）。

创建一个包含如下两行的文本文件test.sql ：

```
INSERT INTO customer(id,first_name,surname) VALUES(5,'Francois','Papo');
INSERT INTO customer(id,first_name,surname) VALUES(6,'Neil','Beneke');
```

可以用如下的方式在操作系统命令行下执行这两个语句：

```
% mysql firstdb < test.sql
```

记住：如果需要的话，加上主机名、用户名和口令（这个例子为了易读起见，显示的是简化版本）。

现在连接上MySQL服务器，会发现下面的两个记录已经被加上了：

```
mysql> SELECT * FROM customer;
+------+------------+-------------+
| id   | first_name | surname     |
+------+------------+-------------+
```

```
|    1 | Yvonne     | Clegg        |
|    2 | Johnny     | Chaka-Chaka  |
|    3 | Winston    | Powers       |
|    4 | Patricia   | Mankunku     |
|    5 | Francois   | Papo         |
|    6 | Neil       | Beneke       |
+------+------------+--------------+
```

如果文件中任意行存在一个SQL错误，MySQL都会立即停止执行文件的其余行。把test.sql改为如下内容，在最开始处添加一个DELETE语句以便在重复执行这些语句组多遍后，也不会产生重复的记录：

```
DELETE FROM customer WHERE id>=6;
INSERT INTO customer(id,first_name,surname) VALUES(6,'Neil','Beneke');
INSERT INTO customer(id,first_name,surname) VALUES(,'Sandile','Cohen');
INSERT INTO customer(id,first_name,surname) VALUES(7,'Winnie','Dlamini');
```

在命令行执行这个文件，将看到MySQL返回的错误：

```
% mysql firstdb < test.sql
ERROR 1064 at line 2: You have an error in your SQL syntax near
 ''Sandile','Cohen')' at line 1
```

如果现在查看customer表包含的内容，就会发现第一个记录已经正确地插入了，但因为第二行包含一个错误（id域没有设定），MySQL从这一行开始停止处理：

```
mysql> SELECT * FROM customer;
+------+------------+--------------+
| id   | first_name | surname      |
+------+------------+--------------+
|    1 | Yvonne     | Clegg        |
|    2 | Johnny     | Chaka-Chaka  |
|    3 | Winston    | Powers       |
|    4 | Patricia   | Mankunku     |
|    5 | Francois   | Papo         |
|    6 | Neil       | Beneke       |
+------+------------+--------------+
```

也可以通过force选项（全部的MySQL选项列表可见第2章“数据类型和表类型”）来强制MySQL在出现错误的情况下也继续执行：

```
% mysql -f firstdb < test.sql
ERROR 1064 at line 2: You have an error in your SQL syntax near
 ''Sandile','Cohen')' at line 1
```

尽管还是在报告错误，但全部的有效记录还是插入了，这可以从表中看到：

```
mysql> SELECT * FROM customer;
+------+------------+--------------+
```

```
| id   | first_name | surname     |
+------+------------+-------------+
|    1 | Yvonne     | Clegg       |
|    2 | Johnny     | Chaka-Chaka |
|    3 | Winston    | Powers      |
|    4 | Patricia   | Mankunku    |
|    5 | Francois   | Papo        |
|    7 | Winnie     | Dlamini     |
|    6 | Neil       | Beneke      |
+------+------------+-------------+
```

重定向输出到文件

可以把获得的输出放入另一个文件中。比如，可以把选择语句加入一个原始文件中，而把查询的输出结果放入第三个文件中，而不是直接从MySQL的命令行执行选择语句。改变test.sql文件如下：

```
DELETE FROM customer WHERE id>=6;
INSERT INTO customer(id,first_name,surname) VALUES(6,'Neil','Beneke');
INSERT INTO customer(id,first_name,surname) VALUES(7,'Winnie','Dlamini');
SELECT * FROM customer;
```

这样就可以把结果输出到一个文件test_output.txt中，如下：

```
% mysql firstdb < test.sql > test_output.txt
```

现在文件test_output.txt包含如下内容：

```
id      first_name      surname
1       Yvonne          Clegg
2       Johnny          Chaka-Chaka
3       Winston         Powers
4       Patricia        Mankunku
5       Francois        Papo
7       Winnie          Dlamini
6       Neil            Beneke
```

注意，输出结果并不是同交互式模式下的输出完全一样。数据用制表符分界了，并且没有了格式化的行。如果想要交互式格式的输出文件，可以使用表选项-t，比如：

```
% mysql -t firstdb < test.sql > test_output.txt
```

在MySQL命令行中使用文件

可以在MySQL命令行中通过SOURCE命令来执行存储在文件中的SQL语句：

```
mysql> SOURCE test.sql
Query OK, 2 rows affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)
```

```
Query OK, 1 row affected (0.00 sec)

+------+------------+-------------+
| id   | first_name | surname     |
+------+------------+-------------+
|    1 | Yvonne     | Clegg       |
|    2 | Johnny     | Chaka-Chaka |
|    3 | Winston    | Powers      |
|    4 | Patricia   | Mankunku    |
|    5 | Francois   | Papo        |
|    7 | Winnie     | Dlamini     |
|    6 | Neil       | Beneke      |
+------+------------+-------------+
7 rows in set (0.00 sec)
```

在稍后如果不需要的话，可以删除通过文本文件添加的记录：

```
mysql> DELETE FROM customer WHERE id > 4;
```

使用批处理模式的原因有如下几条：

- 可以在以后需要的时候重新执行SQL语句。
- 可以拷贝和发送文件给其他人。
- 在有错误的时候，比较容易改正。
- 有时必须用批处理模式，比如在每天的一个固定时间需要重复执行某个SQL命令（比如，与UNIX的cron）。

事务和锁

数据库查询一个接一个。对于一个网站的服务页，只要服务得足够快，并不关心数据库执行查询的顺序。但有些查询可能需要特定的顺序，比如需要前面查询的结果，又如需要同时完成的一组更新。全部的表类型都可以使用锁，但只有InnoDB和BDB表才有内置的事物功能。本节讨论各种事务和锁的机理。

InnoDb表中的事务

InnoDb表类型的能力在于使用事务，或者说是一组SQL语句。典型的例子是银行事务。比如，钱从一个人的账号转到另一个账号，至少需要两个查询：

```
UPDATE person1 SET balance = balance-transfer_amount;
UPDATE person2 SET balance = balance+transfer_amount;
```

这很好，但如果发生了什么问题，比如在第一个查询完成以后，第二个查询完成之前，系统崩溃了，会如何？第一个人钱已经从账号上划走了，账已经支付了，但第二个人却会发怒，他会认为账没有支付。在这种情况下，保证两个查询同时执行或同时不执行非常重要。为了做到这一点，需要把查询包在事务中，用BEGIN来表明事务的开始，用COMMIT语句来表明事务的结束。如果在这之间发生了错误，可以使用ROLLBACK命令来回滚事务的不

完全部分。

现在执行一些查询来看看这是如何工作的。如果在第2章没有创建下面的表，就需要重新创建：

```
mysql> CREATE TABLE innotest (f1 INT,f2 CHAR(10),INDEX (f1))TYPE=InnoDB;
Query OK,0 rows affected (0.10 sec)

mysql> INSERT INTO innotest(f1) VALUES(1);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT f1 FROM innotest;
+------+
| f1   |
+------+
|    1 |
+------+
1 row in set (0.21 sec)
```

迄今为止没有发生奇迹！现在把查询包在一个BEGIN/COMMIT中：

```
mysql> BEGIN;
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO innotest(f1) VALUES(2);
Query OK, 1 row affected (0.05 sec)

mysql> SELECT f1 FROM innotest;
+------+
| f1   |
+------+
|    1 |
|    2 |
+------+
2 rows in set (0.16 sec)
```

如果现在执行ROLLBACK，事务就会被回滚，因为它还没有被提交：

```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT f1 FROM innotest;
+------+
| f1   |
+------+
|    1 |
+------+
1 row in set (0.17 sec)
```

现在看一下在完成事务之前中断了连接会发生什么事：

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO innotest(f1) VALUES(2);
Query OK, 1 row affected (0.00 sec)

mysql> EXIT
Bye

C:\MySQL\bin> mysql firstdb
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8 to server version: 4.0.1-alpha-max

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT f1 FROM innotest;
+------+
| f1   |
+------+
|    1 |
+------+
1 row in set (0.11 sec)
```

重复前面的语句，这一次在离开之前先进行COMMIT。COMMIT执行以后，事务就完成了，因此重新连接的话，就会看到新记录：

```
mysql> BEGIN;
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO innotest(f1) VALUES(2);
Query OK, 1 row affected (0.06 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.05 sec)

mysql> EXIT
Bye

C:\Program Files\MySQL\bin> mysql firstdb
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9 to server version: 4.0.1-alpha-max

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT f1 FROM innotest;
+------+
| f1   |
+------+
|    1 |
|    2 |
+------+
2 rows in set (0.11 sec)
```

一致读

在默认情况下，InnoDB表进行一致读。这句话的意思就是当执行SELECT时，MySQL会一直返回最近完成的事务的数据库中的值。如果有事务正在进行，UPDATE或INSERT语句也不会受到影响。但有一个例外：打开的事务可以看到变化（你可能已经注意到了这点：执行BEGIN-INSERT-SELECT时，显示了插入的结果）。为了演示这一点，需要打开两个窗口，并连接到数据库上。

首先在Window1中从事务中添加一个记录：

```
mysql> BEGIN;
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO innotest(f1) VALUES(3);
Query OK, 1 row affected (0.05 sec)
```

现在切换到Window2：

```
mysql> SELECT f1 FROM innotest;
+------+
| f1   |
+------+
|    1 |
|    2 |
+------+
2 rows in set (0.16 sec)
```

3是插入的，因为它是一个未完成的事务的一部分，因此没有返回。从一个未完成事务返回的结果是不一致读。

现在返回Window1：

```
mysql> SELECT f1 FROM innotest;
+------+
| f1   |
+------+
|    1 |
|    2 |
|    3 |
+------+
```

因为还在事务中，所以3是可见的。

现在还是在Window1中提交这个事务：

```
mysql> COMMIT;
```

现在在Window2中的查询会反映完成的事务：

```
mysql> SELECT f1 FROM innotest;
+------+
```

```
| f1   |
+------+
|    1 |
|    2 |
|    3 |
+------+
```

更新的读锁

一致读并不总是我们需要的东西。如果有超过一个用户往innotest表中增加新记录会如何呢？每个新记录插入一个惟一的升序数字。在这个例子中，f1域既不是PRIMARY KEY也不是AUTO_INCREMENT域，因此在表结构中不会有任何东西阻止重复的发生。但实际上应该保证重复永远不会发生。此时希望读取f1中存在的值然后再插入新的加1的值。但这并不能保证惟一值。看一下在Window1中的例子：

```
mysql> BEGIN;

mysql> SELECT MAX(f1) FROM innotest;
+----------+
| MAX(f1) |
+----------+
|        3 |
+----------+
```

同时在Window2中，有一个用户在做同样的事：

```
mysql> BEGIN;

mysql> SELECT MAX(f1) FROM innotest;
+----------+
| MAX(f1) |
+----------+
|        3 |
+----------+
1 row in set (0.11 sec)
```

现在，两个用户（Window1和Window2）都添加了一个新记录并且提交了它们中的事务：

```
mysql> INSERT INTO innotest(f1) VALUES(4);
Query OK, 1 row affected (0.11 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

现在他们中任一个执行SELECT，都会看到如下结果：

```
mysql> SELECT f1 FROM innotest;
+------+
| f1   |
```

```
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    4 |
+------+
```

一致读并没有保证产生希望的结果：记录值为4和5。避免这个问题的方式是在SELECT中使用一个更新锁。通过告诉MySQL正在为了更新进行读，它就会在事务完成之前阻止其他人读取这个值。

为了这次正确执行，先从表中删除不正确的4：

```
mysql> DELETE FROM innotest WHERE f1=4;
Query OK, 2 rows affected (0.00 sec)
```

现在在Window1中设置一个更新锁：

```
mysql> BEGIN;
mysql> SELECT MAX(f1) FROM innotest FOR UPDATE;
+---------+
| MAX(f1) |
+---------+
|       3 |
+---------+
mysql> INSERT INTO innotest(f1) VALUES(4);
Query OK, 1 row affected (0.05 sec)
```

同时，在Window2中也设置一个更新锁：

```
mysql> BEGIN;
mysql> SELECT MAX(f1) FROM innotest FOR UPDATE;
```

注意为什么没有结果返回。MySQL正在等待Window1中的事务完成。

在Window1中完成这个事务：

```
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

Window2在等待INSERT完成之后，返回了查询的结果。

```
mysql> SELECT MAX(f1) FROM innotest FOR UPDATE;
+---------+
| MAX(f1) |
+---------+
|       4 |
+---------+
1 row in set (4 min 32.65 sec)
```

现在，保证4是最后的值，可以在Window2中往表中加入5：

```
mysql> INSERT INTO innotest(f1) VALUES(5);
Query OK, 1 row affected (0.06 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

共享的读锁

还有一种只有在读取一个正在被一个未完成的事务改变的值才不返回的读锁。它只返回最新的数据，但它自己并不是改变那个值的事务的一部分。比如，使用在innotest表中创建的f2域。假设f1域先被计算了，但只在事务的后一阶段才给f2加值。当SELECT的时候，自己希望看到的既不是给f1域值的记录，也不是给f2域值的记录，而应该是最近的记录。在这种情况下，在返回结果之前，需要等待事务完成。比如，事务从Window1开始：

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO innotest(f1) VALUES(6);
Query OK, 1 row affected (0.00 sec)

mysql> UPDATE innotest set f2='Sebastian' WHERE f1=6;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

如果在Window2中执行一个普通的SELECT，不会得到最新的值（因为前面的事务还没有完成，而InnoDB默认情况下为一致读）。但是，如果在SELECT中使用LOCK IN SHARE MODE，则只有在Window1中的事务完成后才会得到结果。

在Window2中执行普通的查询，返回如下：

```
mysql> SELECT MAX(f1) FROM innotest;
+----------+
| MAX(f1) |
+----------+
|        5 |
+----------+
1 row in set (0.17 sec)
```

还在Window2中，用LOCK IN SHARE MODE的方式执行同样的查询也不会返回结果：

```
mysql> SELECT MAX(f1) FROM INNOTEST LOCK IN SHARE MODE;
```

在Window1中完成事务：

```
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

现在Window2中会返回正确的结果：

```
mysql> SELECT MAX(f1) FROM innotest LOCK IN SHARE MODE;
+---------+
| MAX(f1) |
+---------+
|       6 |
+---------+
1 row in set (4 min 32.98 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

自动提交

默认情况下，除非用BEGIN标明了一个事务，MySQL会自动提交语句。比如，在Window1中的查询返回如下：

```
mysql> SELECT f1 FROM innotest;
+------+
| f1   |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
+------+
6 rows in set (0.11 sec)
```

现在Window2中的用户插入一个记录：

```
mysql> INSERT INTO innotest(f1) VALUES (7);
Query OK, 1 row affected (0.00 sec)
```

在Window1中立即可见（记住在前面的例子中用COMMIT完成了事务）：

```
mysql> SELECT f1 FROM innotest;
+------+
| f1   |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
|    7 |
```

```
+------+
7 rows in set (0.11 sec)
```

由于默认设置了AUTOCOMMIT，来自Window2中的INSERT在其他窗口中会立即可见。但是，对于事务安全的表（InnoDB, BDB），你可以通过把AUTOCOMMIT设为0来改变这个方式。

首先，在Window1中把AUTOCOMMIT设为0：

```
mysql> SET AUTOCOMMIT=0;
Query OK, 0 rows affected (0.00 sec)
```

然后在Window2中执行查询：

```
mysql> SELECT f1 FROM innotest;
+------+
| f1   |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
|    7 |
+------+
7 rows in set (0.22 sec)
```

现在Window1插入一个记录：

```
mysql> INSERT INTO innotest(f1) VALUES(8);
Query OK, 1 row affected (0.00 sec)
```

这次在Window2中不能立即可见：

```
mysql> SELECT f1 FROM innotest;
+------+
| f1   |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
|    7 |
+------+
7 rows in set (0.16 sec)
```

因为已经取消了AUTOCOMMIT，来自Window1中的INSERT一直要到特定的COMMIT执行后才会执行。

从Window1中提交这个事务：

```
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

现在在Window2中，新记录已经可用：

```
mysql> SELECT f1 FROM innotest;
+------+
| f1   |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
|    7 |
|    8 |
+------+
8 rows in set (0.11 sec)
```

AUTOCOMMIT=0的设置并不在整个服务器中起作用，而只是针对特定的会话。如果Window2也把AUTOCOMMIT设为0，就会体会到不同。

首先，在Window1和2中都设置AUTOCOMMIT为0：

```
mysql> SET AUTOCOMMIT=0;
Query OK, 0 rows affected (0.00 sec)
```

现在在Window1中执行下面的语句看看会出现什么：

```
mysql> SELECT f1 FROM innotest;
+------+
| f1   |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
|    7 |
|    8 |
+------+
8 rows in set (0.17 sec)
```

在Window2中添加一个记录，然后提交这个事务：

```
mysql> INSERT INTO innotest(f1) VALUES(9);
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

现在看在Window1中它是否出现：

```
mysql> SELECT f1 FROM innotest;
+------+
| f1   |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
|    7 |
|    8 |
+------+
8 rows in set (0.11 sec)
```

尽管已经提交了结果，新记录中的9也没有出现！原因是Window1中的SELECT也是事务的一部分。一致读赋了一个时间点，当设置的事务完成后，这个时间点才会前移。

在Window1中提交事务：

```
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT f1 FROM innotest;
+------+
| f1   |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
|    7 |
|    8 |
|    9 |
+------+
9 rows in set (0.22 sec)
```

如同在前面看到的，看到最新结果的惟一方式是使用带LOCK IN SHARE MODE的SELECT。它一直会等到插入的事务执行完提交后才返回。

BDB表中的事务

BDB表处理事务与InnoDB表稍微有所不同。先创建表（如果在第2章中没有创建），然后插入一个来自Window1的记录：

```
mysql> CREATE TABLE bdbtest(f1 INT,f2 CHAR(10))TYPE=BDB;
Query OK, 0 rows affected (0.28 sec)

mysql> BEGIN;
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO bdbtest(f1) VALUES(1);
Query OK, 1 row affected (0.00 sec)
```

现在在Window2中执行如下：

```
mysql> SELECT f1 FROM bdbtest;
```

Window2等待Window1中的事务完成（它一直要到Window1中的事务开始，才会返回基于条件的结果集，就像InnoDB）。

只有在Window1提交后，Window2才会得到结果。完成Window1中的事务：

```
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

Window2中的查询完成（不需要重新键入）：

```
mysql> SELECT f1 FROM bdbtest;
+------+
| f1   |
+------+
|    1 |
+------+
1 row in set (3 min 13.99 sec)
```

注意查询花费了很长时间。在BDB表中没有“快速”的SELECT的事实，表明任何一个延迟的事务都可能导致严重的效率问题。

对InnoDB表，默认模式为AUTOCOMMIT=1。这表明除非把改变放在一个事务（用BEGIN开始）中，否则它们会立即被执行。

从Window1中执行下面的查询：

```
mysql> SELECT f1 FROM bdbtest;
+------+
| f1   |
+------+
|    1 |
+------+
1 row in set (0.17 sec)
```

现在从Window2中执行一个INSERT：

```
mysql> INSERT INTO bdbtest(f1) VALUES(2);
Query OK, 1 row affected (0.06 sec)
```

可以立即从Window1中得到结果：

```
mysql> SELECT f1 FROM bdbtest;
+------+
| f1   |
+------+
|    1 |
|    2 |
+------+
2 rows in set (0.16 sec)
```

把AUTOCOMMIT设置为0，效果与把全部的语句包在一个BEGIN中一样。
在Window1中把AUTOCOMMIT设置为0，然后插入一个记录：

```
mysql> SET OPTION AUTOCOMMIT=0;
Query OK, 0 rows affected (0.11 sec)
mysql> INSERT INTO bdbtest(f1) VALUES(3);
Query OK, 1 row affected (0.11 sec)
```

当事务还在活动的时候，Window2中的查询一直会等待：

```
mysql> SELECT f1 FROM bdbtest;
```

只有当事务提交以后结果才会出现。
在Window1中提交事务：

```
mysql> COMMIT;
Query OK, 0 rows affected (0.05 sec)
```

现在在Window2中得到了查询结果（不需重新键入查询）：

```
mysql> SELECT f1 FROM bdbtest;
+------+
| f1   |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (2 min 8.14 sec)
```

深入事务行为

还有许多其他的命令用来自动结束一个事务（换句话说，就像自己执行了提交命令）：

- BEGIN

- ALTER TABLE
- CREATE INDEX
- RENAME TABLE（是ALTER TABLE x RENAME的同义词）
- TRUNCATE
- DROP TABLE
- DROP DATABASE

即使命令不成功，也会执行提交。比如，在Window1中开始下面的事务：

```
mysql> BEGIN;
mysql> SELECT MAX(f1) FROM innotest FOR UPDATE;
+---------+
| MAX(f1) |
+---------+
|       9 |
+---------+
```

在Window2开始另一个事务：

```
mysql> BEGIN;
mysql> SELECT MAX(f1) FROM innotest FOR UPDATE;
```

结果没有显示，因为Window1已经锁住了更新的行。但是，Window1中的用户改变了主意，决定先改变表的结构。

- 在Window1中执行ALTER TABLE:

```
mysql> ALTER TABLE innotest add f1 INT;
ERROR 1060: Duplicate column name 'f1'
```

尽管ALTER失败了，锁还是被释放了，事务也被提交了，在Window2中等待的查询也完成了（不需重新键入）。

```
mysql> SELECT MAX(f1) FROM innotest FOR UPDATE;
+---------+
| MAX(f1) |
+---------+
|       9 |
+---------+
1 row in set (2 min 23.52 sec)
```

锁表

在讨论InnoDB和BDB表的时候，已经遇到了行级锁的概念，也就是某行被锁定一段时间。行级锁在大量的插入和更新时效率更高。但行级锁只对事务安全的表类型（BDB和InnoDB）有效。MySQL也有表级锁，这是对所有的表类型都有效的。

有两种类型的表级锁：读锁和写锁。读锁表示只对表进行读操作，写操作被锁。写锁表示在上锁的那段时间没有读或写操作。锁一张表的语法如下：

```
LOCK TABLE tablename {READ|WRITE}
```

解锁一张表，只需简单地执行UNLOCK TABLES语句，如下：

```
UNLOCK TABLES
```

下面演示了一个实际的表级锁，对任何类型的表都可以。先从Window1中锁表：

```
mysql> LOCK TABLE customer READ;
Query OK, 0 rows affected (0.01 sec)
```

其他的线程可以读，但不能写，这可以从Window2中执行下面的语句看出：

```
mysql> SELECT * FROM customer;
+------+------------+-------------+
| id   | first_name | surname     |
+------+------------+-------------+
|    1 | Yvonne     | Clegg       |
|    2 | Johnny     | Chaka-Chaka |
|    3 | Winston    | Powers      |
|    4 | Patricia   | Mankunku    |
+------+------------+-------------+

mysql> INSERT INTO customer(id,first_name,surname) VALUES(5,'Francois','Papo');
```

一直到锁被Window1释放后，INSERT语句才会执行。

```
mysql> UNLOCK TABLES;
```

Window2中的INSERT现在也完成了（不需重新键入）：

```
mysql> INSERT INTO customer(id,first_name,surname) VALUES(5,'Francois','Papo');
Query OK, 1 row affected (7 min 0.74 sec)
```

也可以同时锁多张表。从Window1中放置如下的锁：

```
mysql> LOCK TABLE customer READ,sales WRITE;
```

其他的线程可以读customer表，但不能读sales表。从Window2中执行如下的SELECT：

```
mysql> SELECT * FROM sales;
```

如果创建锁的线程想在customer表中添加记录，就会失败。它不会等待锁被释放（因为它创建了这个锁，如果等待的话，锁就不能释放）；仅仅是INSERT失败。在Window1中试一下：

```
mysql> INSERT INTO customer VALUES (1,'a','b');
ERROR 1099: Table 'customer' was locked with a READ lock and can't be
updated
```

但是，它可以在写锁的表上进行读，如下，还是在Window1中：

```
mysql> SELECT * FROM sales;
+------+-----------+------+-------+
```

```
| code | sales_rep | id   | value |
+------+-----------+------+-------+
|    1 |         1 |    1 |  2000 |
|    2 |         4 |    3 |   250 |
|    3 |         2 |    3 |   500 |
|    4 |         1 |    4 |   450 |
|    5 |         3 |    1 |  3800 |
|    6 |         1 |    2 |   500 |
|    7 |         2 | NULL |   670 |
+------+-----------+------+-------+
```

```
mysql> UNLOCK TABLES;
```

写锁释放后，Window2现在开始执行SELECT（不需重新键入）：

```
mysql> SELECT * FROM sales;
+------+-----------+------+-------+
| code | sales_rep | id   | value |
+------+-----------+------+-------+
|    1 |         1 |    1 |  2000 |
|    2 |         4 |    3 |   250 |
|    3 |         2 |    3 |   500 |
|    4 |         1 |    4 |   450 |
|    5 |         3 |    1 |  3800 |
|    6 |         1 |    2 |   500 |
|    7 |         2 | NULL |   670 |
+------+-----------+------+-------+
7 rows in set (5 min 59.35 sec)
```

说明：可以使用单数或复数的形式。[UN]LOCK TABLE和[UN]LOCK TABLES都是有效的，无论想锁多少表，MySQL都不关心语法！

写锁的优先级高于读锁，如果一个线程正在等一个读锁，而请求的写锁来了，在获得读锁之前会等待到写锁获得，释放，如下：

从Window1中放置一个写锁：

```
mysql> LOCK TABLE customer WRITE;
Query OK, 0 rows affected (0.00 sec)
```

在Window2中放置一个读锁：

```
mysql> LOCK TABLE customer READ;
```

只有在写锁被释放以后，才能获得读锁。同时，另一个等待写锁的请求，也得等到第一个写锁释放。

在第三个窗口Window3中放置另外一个写锁：

```
mysql> LOCK TABLE customer WRITE;
```

现在从Window1中释放锁：

```
mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.00 sec)
```

现在Window 3中的写锁可以得到了，尽管它是在读锁之后请求的，如下（不需重新键入LOCK语句）：

```
mysql> LOCK TABLE customer WRITE;
Query OK, 0 rows affected (33.93 sec)
mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.00 sec)
```

只有当Window 3中的写锁释放以后，Window2中的读锁才能获得（不需重新键入）：

```
mysql> LOCK TABLE customer READ;
Query OK, 0 rows affected (4 min 2.46 sec)
mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.00 sec)
```

可以通过用LOW_PRIORITY关键字来指定写锁为低优先级以便覆盖。如果用写锁的低优先级请求重新运行前面的例子，前面的读锁会首先获得。

先在Window1中放置写锁：

```
mysql> LOCK TABLE customer WRITE;
Query OK, 0 rows affected (0.00 sec)
```

接着在Window2中建立读锁：

```
mysql> LOCK TABLE customer READ;
```

在Window3中创建一个低优先级的写锁：

```
mysql> LOCK TABLE customer LOW_PRIORITY WRITE;
```

现在在Window1中释放锁：

```
mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.00 sec)
```

这次Window2先获得了锁（不需重新键入LOCK语句）：

```
mysql> LOCK TABLE customer READ;
Query OK, 0 rows affected (20.88 sec)

mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.00 sec)
```

最后Window3中获得了写锁（不需重新键入）：

```
mysql> LOCK TABLE customer LOW_PRIORITY WRITE;
Query OK, 0 rows affected (1 min 25.94 sec)
```

释放锁以便后面可以继续使用这张表：

```
mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.00 sec)
```

警告： LOCK TABLES语句不是事务安全的。它在锁表之前会提交全部的活动事务。

表级锁一般用在不支持事务的表上。如果使用InnoDB或BDB表，就使用BEGIN和COMMIT来避免数据的异常。下面是一个使用它的场合的例子。如果customer_sales_values表为空，用一些记录来填充它：

```
mysql> INSERT INTO customer_sales_values(first_name, surname, value,
 value2) VALUES('Johnny', 'Chaka-Chaka', 500, NULL), ('Patricia',
 'Mankunku', 450, NULL), ('Winston', 'Powers', 750, NULL), ('Yvonne',
 'Clegg', 5800, NULL), ('Charles', 'Dube', 0, NULL), ('Charles',
 'Dube', 0, NULL), ('Gladys', 'Malherbe', 5, 10);
```

假设Johnny Chaka-Chaka完成了两次销售，两者都被不同的职员处理。一个销售额为100美元，另一个为300美元。两个职员都经过了读取已存在的值，然后把100或300与其相加，然后更新记录的过程。问题来了：如果两者都在更新之前先执行了SELECT，那么，其中的一个更新会覆盖另一个，而有一个值丢失了，如下：

先在Window1中执行查询：

```
mysql> SELECT value from customer_sales_values WHERE
 first_name='Johnny' and surname='Chaka-Chaka';
+-------+
| value |
+-------+
|   500 |
+-------+
```

然后在Window2中执行查询：

```
mysql> SELECT value from customer_sales_values WHERE
 first_name='Johnny' and surname='Chaka-Chaka';
+-------+
| value |
+-------+
|   500 |
+-------+
```

这是Window1：

```
mysql> UPDATE customer_sales_values SET value=500+100 WHERE
 first_name='Johnny' and surname='Chaka-Chaka';
Query OK, 1 row affected (0.01 sec)
```

这是Window2：

```
mysql> UPDATE customer_sales_values SET value=500+300 WHERE
 first_name='Johnny' and surname='Chaka-Chaka';
Query OK, 1 row affected (0.01 sec)
```

两个销售额都获得后，Johnny的销售额总数是800美元，少了100美元！如果仔细锁表的话，可以避免这个问题。

重置数据以后，重新开始，执行下面的更新：

```
mysql> UPDATE customer_sales_values SET value=500 WHERE
 first_name='Johnny' and surname='Chaka-Chaka';
Query OK, 1 row affected (0.00 sec)
```

现在在Window1中设置一个写锁：

```
mysql> LOCK TABLE customer_sales_values WRITE;
mysql> SELECT value from customer_sales_values WHERE
 first_name='Johnny' and surname='Chaka-Chaka';
+-------+
| value |
+-------+
|   500 |
+-------+
```

Window2也要设置一个写锁：

```
mysql> LOCK TABLE customer_sales_values WRITE;
```

没有获得锁，因为Window1已经获得了一个写锁。现在在Window1中可以更新记录，但要在Window1释放锁容许Window2继续之前。

在Window1中执行下面的更新语句并释放锁：

```
mysql> UPDATE customer_sales_values SET value=500+100 WHERE
 first_name='Johnny' and surname='Chaka-Chaka';
Query OK, 1 row affected (0.00 sec)
mysql> UNLOCK TABLES;
```

Window2获得了锁（不需重新键入），可以像下面一样完成其余的事务：

```
mysql> LOCK TABLE customer_sales_values WRITE;
Query OK, 0 rows affected (1 min 35.87 sec)
mysql> SELECT value from customer_sales_values WHERE
 first_name='Johnny' and surname='Chaka-Chaka';
+-------+
| value |
+-------+
|   600 |
+-------+
1 row in set (0.00 sec)
```

```
mysql> UPDATE customer_sales_values SET value=600+300 WHERE
 first_name='Johnny' and surname='Chaka-Chaka';
Query OK, 1 row affected (0.01 sec)

mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.00 sec)
```

Johnny得到了他应该得到的钱，表正确地反映了他的销售额900美元。

避免表级锁

在更新率高的表中，应该尽可能地避免使用表级锁。比如，在写锁的情况下，在锁存在期间，表中的记录都不能读或写。并且因为在默认情况下，写锁的优先级高于读锁，只有在更新或插入完全完成后，才可以进行读记录操作，这会导致潜在的可怕的MySQL拥塞。但还是有办法避免表级锁的。一个方法就是在同一个语句中执行读和更新操作（称做增量更新）。

在Window1中执行下面的增量更新：

```
mysql> UPDATE customer_sales_values SET value=value+300 WHERE
 first_name='Johnny' and surname='Chaka-Chaka';
```

Window2也可以执行自己的更新：

```
mysql> UPDATE customer_sales_values SET value=value+100 WHERE
 first_name='Johnny' and surname='Chaka-Chaka';
```

现在，无论语句的顺序如何，更新都会在最新的值上进行。

事务级别

在处理事务时可以通过设置事务级别来改变默认的动作方式。MySQL中有许多事务级别，它支持下面的事务分离级别：

READ UNCOMMITTED 这个级别的事务容许从其他事务中读取未提交的数据（称为脏读）。

READ COMMITTED 这个级别不容许脏读。

REPEATABLE READ 这个级别容许可重复的读（数据即使已经被另一个提交过的事务改变）。

SERIALIZABLE 这个级别不容许幻影读——就是另一个事务已经提交了与查询的结果匹配的新行。每次读取的数据是一样的。

使用下面的语法来改变事务的级别：

```
SET [scope] TRANSACTION ISOLATION LEVEL
{ isolation_level }
```

scope选项可以是GLOBAL或SESSION之一。这个选项覆盖了通常的语句范围，而这个语句是为下一个开始的事务设置事务分离级别的。GLOBAL为新事务设置级别，而SESSION是针对那个线程的全部新事务的。isolation_level是前面的四个分离级别中的其中之一。

小结

连接可能比第1章“认识MySQL”中的简单的两个表连接复杂的多。内连接忽略被连接表的NULL值（或者是没有关联记录的行），而外连接包括NULL值。左外连接返回指定的第一张表（在左边）中的全部数据，包括在右边表中没有关联记录的数据，右外连接返回指定连接的右边的表的全部数据。全外连接结合了左和右连接的特点，但MySQL还不支持全外连接。

自然连接利用了公共域有相同的名字的特点，在这种情况下，可以简化语法。

UNION命令把一个以上的查询结果组合成一个结果。

子选择是查询之中的查询。一般用连接重写之后效率会提高。

像DELETE语句那样一个一个地删除记录，如果想用它来删除表中的全部数据，效率会很低。在这种情况下，可以使用TRUNCATE语句，它是一个快速的删除方法，但它不像DELETE那样返回被删除的记录个数。

用户变量可以用来存储在后面的查询中使用的值。在使用的时候需要注意，用户变量应该在使用之前设置。在SELECT语句中，在域列表之前（在SELECT和用户变量设置之后）先执行条件（WHERE从句）。

MySQL也可以用批处理模式执行，即SQL语句存储在文件中以便编辑和重用。可以重定向输出到文件中，就可以在后面容易地检查查询的结果了。

全部的表类型都可以使用表级锁，这样整个表都被锁住了，与此对应的是针对事务安全的表的行级锁。

在下一章，读者将加强自己的技能，看看如何优化数据库的性能。还将探究创建索引，编写更有效率的查询语句，以及提高服务器的性能。

第4章 索引和查询优化

查询能正确工作是一件事，而在客户比较多时，能让查询快速进行则是另外一件事。可以用几种基本的方法来加速查询。恰当地使用索引能够使效率得到很大地提高，仔细地调整系统也能显著地提高性能。

本章的主要内容：

- 创建和使用索引
- 主键、惟一索引、全文索引和普通索引
- 全文检索
- 删除和改变索引
- 自动增加的域
- 使用EXPLAIN分析查询
- 优化SELECT语句

理解索引

迄今为止，前面章节创建的表没有一个有索引。添加新记录的时候，一般是加在表的结尾，但在删除了一个记录并且有空间的情况下，记录也可以加在表的中间。换句话说，就是记录的储存没有顺序。比如，考虑以下第3章“高级SQL”中的客户表，假设采用了那章中的例子，就应该包含如下顺序的记录：

```
+------+------------+-------------+
| id   | first_name | surname     |
+------+------------+-------------+
| 1    | Yvonne     | Clegg       |
| 2    | Johnny     | Chaka-Chaka |
| 3    | Winston    | Powers      |
| 4    | Patricia   | Mankunku    |
| 5    | Francois   | Papo        |
| 7    | Winnie     | Dlamini     |
| 6    | Neil       | Beneke      |
+------+------------+-------------+
```

现在，假设自己在做MySQL的工作。如果想返回姓为Beneke的任意记录，就要从开始检查每一个记录。如果没有进一步的信息，对于自己或者是对MySQL来说，没有其他的办法知道到哪里可以找到满足这个标准的记录。用这种方式扫描表（从始到终检查全部的记录）叫全表扫描。当表很大的时候，这样的效率很低下；对包含成千上万记录的表进行全表扫描会很慢。

如果记录排序了的话，就会解决这个问题。还是看前面的同样的记录，但表按照姓排了序：

```
+------+------------+-------------+
| id   | first_name | surname     |
+------+------------+-------------+
| 6    | Neil       | Beneke      |
| 2    | Johnny     | Chaka-Chaka |
| 1    | Yvonne     | Clegg       |
| 7    | Winnie     | Dlamini     |
| 4    | Patricia   | Mankunku    |
| 5    | Francois   | Papo        |
| 3    | Winston    | Powers      |
+------+------------+-------------+
```

现在，检索这张表可以更快了。因为已经知道记录是按照姓的字母顺序排序的，一旦检查到以C开头的姓Chaka-Chaka，就知道不会再有Beneke记录了。这样，就只需要检查一个记录，而不是前面未排序的表中的七个记录。这很节约时间，而且在大表中好处会更大。

因此，排序表看起来是问题的答案。但不幸的是，也可能需要用其他的方式查询表。比如，可能需要返回一个id为3的记录。如果是按姓排序的表，还是需要检查全部的记录，这样就再一次遇到效率低下的问题。

解决的方法是为需要排序的每个域创建分离的列表。它们不用包含全部的域，只要有需要排序的域和一个指向全表的记录的指针即可。这些列表叫索引，它们是关系型数据库中最常用和最容易误用的部分（见图4.1）。在一些情况下（MyISAM表），索引被存储为分离的文件，在另一些情况下（InnoDB表），索引存储为表空间的一部分。

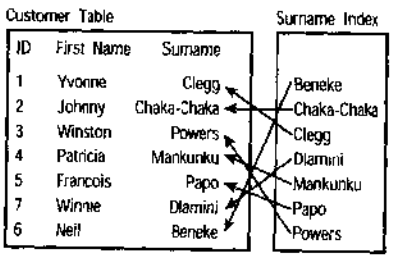

图4.1 指向关联的customer表记录的索引记录

创建索引

在MySQL中，有四种类型的索引：主键、惟一索引、全文索引和普通索引。

创建主键

主键就是值惟一并且没有值为NULL的域的索引。

说明： 术语“主键”严格地说，是一个逻辑术语，但MySQL使用它来表示一个物理索引。当MySQL表示的主键存在时，就总是存在一个关联的索引。贯穿本文，术语键都表示一个物理索引。

在创建表的时候创建主键，需要在域定义的结尾使用PRIMARY KEY以及域的列表：

```
CREATE TABLE tablename(fieldname columntype NOT NULL,
   [fieldname2...,] PRIMARY KEY(fieldname1 [,fieldname2...]));
```

注意关键字NOT NULL在创建主键时是强制性的，主键不能包含null值。如果忘记了指明这一点，MySQL就会警告：

```
mysql> CREATE TABLE pk_test(f1 INT, PRIMARY KEY(f1));
ERROR 1171: All parts of a PRIMARY KEY must be NOT NULL;
If you need NULL in a key, use UNIQUE instead
```

在一个已经存在的表上创建主键，可以使用ALTER关键字：

```
ALTER TABLE tablename ADD PRIMARY KEY(fieldname1 [,fieldname2...]);
```

为customer表选择一个主键非常容易。id域就可以担当这种功能，因为每个客户都有不同的id，并且不能为null。哪个名字域都不理想，因为在某些阶段它们都可能会重复。为了给customer表的id域增加一个主键，需要改变域以使其不能为null，然后增加主键。可以用一个语句来完成这个任务，如下：

```
mysql> ALTER TABLE customer MODIFY id INT NOT NULL, ADD PRIMARY KEY(id);
Query OK, 7 rows affected (0.00 sec)
Records: 7  Duplicates: 0  Warnings: 0
```

可以通过下面的语句来检查列以查看对表所做的改变：

```
mysql> DESCRIBE customer;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     |      | PRI | 0       |       |
| first_name | varchar(30) | YES  |     | NULL    |       |
| surname    | varchar(40) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

id域在Null列没有YES标志，表示它不再接受null值。在Key列有一个PRI，表示这是主键。

主键也可以由多个域组成。有时不能用一个域来惟一地标示一个记录。在这种情况下需要用逗号分割域来添加主键：

```
mysql> CREATE TABLE pk2(id INT NOT NULL, id2 INT NOT NULL, PRIMARY KEY(id,id2));
Query OK, 0 rows affected (0.00 sec)
```

或者是在表存在时，用如下的方式：

```
mysql> ALTER TABLE pk2 ADD PRIMARY KEY(id,id2);
Query OK, 0 rows affected (0.01 sec)
```

前面一章的sales表还没有主键：

```
mysql> SHOW COLUMNS FROM sales;
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| code      | int(11) | YES  |     | NULL    |       |
| sales_rep | int(11) | YES  |     | NULL    |       |
| id        | int(11) | YES  |     | NULL    |       |
| value     | int(11) | YES  |     | NULL    |       |
+-----------+---------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

假设增加了一个与已存在记录的代码相同的新记录：

```
mysql> SELECT * FROM sales;
+------+-----------+------+-------+
| code | sales_rep | id   | value |
+------+-----------+------+-------+
|    1 |         1 |    1 |  2000 |
|    2 |         4 |    3 |   250 |
|    3 |         2 |    3 |   500 |
|    4 |         1 |    4 |   450 |
|    5 |         3 |    1 |  3800 |
|    6 |         1 |    2 |   500 |
|    7 |         2 | NULL |   670 |
+------+-----------+------+-------+
7 rows in set (0.00 sec)
mysql> INSERT INTO sales VALUES(7,3,3,1000);
Query OK, 1 row affected (0.00 sec)
```

现在还没有出现问题。尽管现在有两个代码为7的记录，但在表的结构中容许这样做。现在，由于使用主键的原因，决定把代码作为主键：

```
mysql> ALTER TABLE sales MODIFY code INT NOT NULL,ADD PRIMARY KEY(code);
ERROR 1062: Duplicate entry '7' for key 1
```

代码域有重复的值，但根据主键的定义它应该是惟一的。这里，将不得不删除和更新重复的值或者使用容许重复的普通索引。大多数情况下，使用主键的情况下，表工作得更好。这种情况下，更新相应的记录非常容易。

```
mysql> UPDATE sales SET code=8 WHERE code=7 AND sales_rep=3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> ALTER TABLE sales MODIFY code INT NOT NULL,ADD PRIMARY KEY(code);
Query OK, 8 rows affected (0.01 sec)
Records: 8  Duplicates: 0  Warnings: 0
```

提示： 作者工作在一个因为没有主键和不存在键的组合而变得有成千重复的“惟一”域系统中。在创建表的时候，增加一个键，特别是主键是很好的习惯。

创建普通索引

不是主键的索引容许重复的值（除非域标明为惟一的）。在创建表的时候，同时创建索引是最好的方法：

```
CREATE TABLE tablename(fieldname columntype, fieldname2
   columntype, INDEX [indexname] (fieldname1 [,fieldname2...]));
```

只要使用逗号分割，就可以在创建表的时候创建多个索引：

```
CREATE TABLE tablename(fieldname columntype, fieldname2
   columntype, INDEX [indexname1] (fieldname1,fieldname2),INDEX
   [indexname2] (fieldname1 [,fieldname2...]));
```

也可以使用下面的代码在以后创建索引：

```
ALTER TABLE tablename ADD INDEX [indexname] (fieldname1 [,fieldname2...]);
```

或者是如下的代码：

```
mysql> CREATE INDEX indexname on tablename(fieldname1 [,fieldname2...]);
```

尽管使用CREATE INDEX语句时索引名是强制性的，这两个语句还是要求索引名。如果在ALTER TABLE ADD... INDEX...语句中没有对索引命名，MySQL会根据域名自己赋一个名。如果在索引中有多于一个的域的话，MySQL就会采用第一个域作为索引名。如果对于同样的域有第二个索引，MySQL就在索引名后加_2，然后是_3，等等。

下面的sales表有一个主键，但在value域上还可以创建一个索引。经常需要查询比某个值大或小的记录，即按值排序：

```
mysql> SHOW COLUMNS FROM sales;
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| code      | int(11) |      | PRI | 0       |       |
| sales_rep | int(11) | YES  |     | NULL    |       |
| id        | int(11) | YES  |     | NULL    |       |
| value     | int(11) | YES  |     | NULL    |       |
+-----------+---------+------+-----+---------+-------+
4 rows in set (0.00 sec)
mysql> ALTER TABLE sales ADD INDEX(value);
Query OK, 8 rows affected (0.02 sec)
Records: 8  Duplicates: 0  Warnings: 0
```

说明： 在MySQL语句中，可以使用KEY关键字来代替INDEX。作者一般使用INDEX，因为在磁盘中，KEY一般是指逻辑结构，而INDEX一般指实际的物理结构。

创建全文索引

可以在表MyISAM中针对任意的CHAR、VARCHAR或TEXT域创建全文索引。全文索引是用来对大表中的文本域进行索引的。

使用下面的语法在创建表的时候创建全文索引：

```
CREATE TABLE tablename (fieldname columntype, fieldname2
   columntype, FULLTEXT(fieldname [,fieldname2...]));
```

在这个语法中，可以加上可选的INDEX关键字：

```
CREATE TABLE tablename (fieldname columntype, fieldname2
   columntype, FULLTEXT INDEX(fieldname [,fieldname2...]));
```

在一个已经存在的表中创建全文索引，则使用下面的语法：

```
ALTER TABLE tablename ADD FULLTEXT [indexname] (fieldname [,fieldname2...]);
```

或者是用下面的代码：

```
CREATE FULLTEXT INDEX indexname ON tablename(fieldname [,fieldname2...]);
```

下面的例子创建了一张表并对其中的一些域创建了全文索引：

```
mysql> CREATE TABLE ft(f1 VARCHAR(255),f2 TEXT,f3 BLOB,f4 INT);
Query OK, 0 rows affected (0.01 sec)
mysql> ALTER TABLE ft ADD FULLTEXT (f1,f2);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

f1和f2域分别是VARCHAR和TEXT，因此可以查全文索引：

```
mysql> ALTER TABLE ft ADD FULLTEXT (f1,f4);
ERROR 1005: Can't create table './firstdb/#sql-52eb_4f.frm' (errno: 140)
mysql> ALTER TABLE ft ADD FULLTEXT (f2,f3);
ERROR 1005: Can't create table './firstdb/#sql-52eb_4f.frm' (errno: 140)
```

在这个例子中f4域是INT类型，而f3是BLOB类型，因此不能在它们上面创建全文索引。

使用全文索引

先创建一张带全文索引的表，然后插入一些书名来测试它：

```
mysql> CREATE TABLE ft2(f1 VARCHAR(255),FULLTEXT(f1));
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO ft2 VALUES('Waiting for the
 Barbarians'),
 ('In the Heart of the Country'),
 ('The Master of Petersburg'),
 ('Writing and Being'),
 ('Heart of the Beast'),
```

```
('Heart of the Beest'),
('The Beginning and the End'),
('Master Master'),
('A Barbarian at my Door');
Query OK, 9 rows affected (0.00 sec)
Records: 9  Duplicates: 0  Warnings: 0
```

可以使用MATCH()函数匹配域，AGAINST()匹配值，来返回全文检索的结果，就像在下面的例子中，查找单词Master出现的次数：

```
mysql> SELECT * FROM ft2 WHERE MATCH(f1) AGAINST ('Master');
+--------------------------+
| f1                       |
+--------------------------+
| Master Master            |
| The Master of Petersburg |
+--------------------------+
2 rows in set (0.01 sec)
```

尽管Master是第二个加入的，但它还是先出现了，这是不一致的。MySQL为每个匹配都计算一个适当的结果并按照这个顺序返回结果。

说明： 记住在文本域上的查询与大小写无关。如果对于VARCHAR或CHAR域没有使用BINARY关键字的话，结果也是一样的。

噪音单词

现在执行另一个查询：

```
mysql> SELECT * FROM ft2 WHERE MATCH(f1) AGAINST ('The Master');
+--------------------------+
| f1                       |
+--------------------------+
| Master Master            |
| The Master of Petersburg |
+--------------------------+
2 rows in set (0.00 sec)
```

这些结果可能与你希望的不同。大部分名称上都包含单词the，The Beginning and the End包含两个。这样做有许多原因：

- MySQL有一个叫百分之五十的门限的东西。任何在百分之五十以上的域出现的单词就被认为是噪音单词，意即它们会被忽略。
- 少于3个的单词会从索引中排除。
- 有一个预定义的噪音单词列表，其中包括the。

因此，The Beginning and the End没有机会了！

警告： 如果有一张只有一个记录的表，全部单词都会是噪音单词，而全文索引不会返回任何东西。表的记录比较少的话，也会增加被当做噪音单词的可能性。

下面的查询不会返回任何内容，尽管for在数据中确实出现了，但因为for小于等于3个字符，因此默认情况下从索引中排除了：

```
mysql> SELECT * FROM ft2 WHERE MATCH(f1) AGAINST ('for');
Empty set (0.00 sec)
```

相关性

可以在WHERE条件中使用MATCH()函数，也可以如下一样返回结果：

```
mysql> SELECT f1, (MATCH(f1) AGAINST ('Master')) FROM ft2;
+---------------------------+-------------------------------+
| f1                        | (MATCH(f1) AGAINST ('Master')) |
+---------------------------+-------------------------------+
| Waiting for the Barbarians |                             0 |
| In the Heart of the Country |                             0 |
| The Master of Petersburg  |             1.2245972156525 |
| Writing and Being         |                             0 |
| Heart of the Beast        |                             0 |
| Heart of the Beest        |                             0 |
| A Barbarian at my Door    |                             0 |
| Master Master             |              1.238520026207 |
| The Beginning and the End |                             0 |
+---------------------------+-------------------------------+
9 rows in set (0.00 sec)
```

相关性分数可能不能匹配这个例子，因为MySQL有时会改变权重的计算方式。

相关性计算非常智能。它基于行中的索引域的单词个数，行中惟一单词的个数，结果中的单词总数，包含特定单词的记录数，以及单词的权重。少的单词权重大，包含单词的记录越多，权重越小。

MySQL可以在返回需要的域的同时返回相关性，基本不需要额外的时间，因为二者调用的MATCH()函数是相同的。

```
mysql> SELECT f1,(MATCH(f1) AGAINST ('Master')) FROM ft2
 WHERE MATCH(f1) AGAINST ('Master');
+---------------------------+-------------------------------+
| f1                        | (MATCH(f1) AGAINST ('Master')) |
+---------------------------+-------------------------------+
| Master Master             |              1.238520026207 |
| The Master of Petersburg  |             1.2245972156525 |
+---------------------------+-------------------------------+
```

布尔全文查找

MySQL 4增强的最有用的功能是进行布尔全文查找。它利用了全集特征来查找单词、单词组合以及单词的部分，等等（见表4.1）。

表4.1 布尔全文查找

运算符	描述
+	接下来的单词是强制性的，在返回的行中必须出现
-	接下来的单词是禁止的，在返回的行中不能出现
<	接下来的单词的相关性小于其他单词
>	接下来的单词的相关性大于其他单词
( )	在子表达式中用来分组单词
~	接下来的单词对行相关性提供相反的贡献（与-运算符不同，如果发现了单词，整个行都被排除；也与<运算符不同，尽管提供的是一个非常低的但还是正的单词相关性）
*	通配符，表示0或多个字符。只能出现在单词的结尾
"	包含在双引号中的内容作为一个整体对待

布尔全文查找不把50%的门限考虑在内。为了进行布尔全文查找，需要使用IN BOOLEAN MODE从句：

```
mysql> SELECT * FROM ft2 WHERE MATCH(f1) AGAINST
 ('+Master -Petersburg' IN BOOLEAN MODE);
+---------------+
| f1            |
+---------------+
| Master Master |
+---------------+
1 row in set (0.00 sec)
```

在这个例子中，排除了单词Petersburg，因此尽管Master出现在标题上，但并没有返回The Master of Petersburg。

注意这两个结果集的差别：

```
mysql> SELECT * FROM ft2 WHERE MATCH(f1) AGAINST
 ('Country Master' IN BOOLEAN MODE);
+-----------------------------+
| f1                          |
+-----------------------------+
| In the Heart of the Country |
| The Master of Petersburg    |
| Master Master               |
+-----------------------------+
3 rows in set (0.00 sec)
mysql> SELECT * FROM ft2 WHERE MATCH(f1) AGAINST
 ('+Country Master' IN BOOLEAN MODE);
+-----------------------------+
```

```
| f1                          |
+-----------------------------+
| In the Heart of the Country |
+-----------------------------+
1 row in set (0.00 sec)
```

在第二个查找中，单词Country是强制性的（默认情况下，单词是可选的），因此The Master of Petersburg和Master Master都没有返回。

下面的例子演示了导致混淆的一般原因：

```
mysql> SELECT * FROM ft2 WHERE MATCH(f1) AGAINST
 ('+Dog Master' IN BOOLEAN MODE);
+--------------------------+
| f1                       |
+--------------------------+
| The Master of Petersburg |
| Master Master            |
+--------------------------+
2 rows in set (0.00 sec)
```

如果把这个结果与前面例子的结果比较的话，你可能很感到奇怪，但因为单词Dog小于等于3个字符，为了查找的目的而排除掉了。

下面两个例子演示了查找一个单词和单词的一部分（使用*运算符）的区别：

```
mysql> SELECT * FROM ft2 WHERE MATCH(f1) AGAINST
 ('Barbarian' IN BOOLEAN MODE);
+-----------------------+
| f1                    |
+-----------------------+
| A Barbarian at my Door |
+-----------------------+
1 row in set (0.00 sec)
mysql> SELECT * FROM ft2 WHERE MATCH(f1) AGAINST
 ('Barbarian*' IN BOOLEAN MODE);
+---------------------------+
| f1                        |
+---------------------------+
| A Barbarian at my Door    |
| Waiting for the Barbarians |
+---------------------------+
2 rows in set (0.01 sec)
```

默认情况下，除非使用了*运算符，只匹配整个单词。

下面的三个例子演示了使用>和<运算符来分别增加和减少权重：

```
mysql> SELECT f1,MATCH(f1) AGAINST ('Heart Beest Beast'
 IN BOOLEAN MODE) AS m FROM ft2 WHERE MATCH(f1)
```

```
 AGAINST ('Heart Beest Beast' IN BOOLEAN MODE);
+-----------------------------+------+
| f1                          | m    |
+-----------------------------+------+
| In the Heart of the Country |    1 |
| Heart of the Beast          |    2 |
| Heart of the Beest          |    2 |
+-----------------------------+------+
3 rows in set (0.00 sec)
mysql> SELECT f1,MATCH(f1) AGAINST ('Heart Beest >Beast'
 IN BOOLEAN MODE) AS m FROM ft2 WHERE MATCH(f1)
 AGAINST ('Heart Beest >Beast' IN BOOLEAN MODE);
+-----------------------------+------+
| f1                          | m    |
+-----------------------------+------+
| In the Heart of the Country |    1 |
| Heart of the Beast          |  2.5 |
| Heart of the Beest          |    2 |
+-----------------------------+------+
3 rows in set (0.00 sec)
```

>运算符增加了Heart of the Beast的相关性。

```
mysql> SELECT f1,MATCH(f1) AGAINST ('Heart <Beest Beast'
 IN BOOLEAN MODE) As m FROM ft2 WHERE MATCH(f1)
 AGAINST ('Heart <Beest Beast' IN BOOLEAN MODE);
+-----------------------------+-----------------+
| f1                          | m               |
+-----------------------------+-----------------+
| In the Heart of the Country |               1 |
| Heart of the Beast          |               2 |
| Heart of the Beest          | 1.6666667461395 |
+-----------------------------+-----------------+
3 rows in set (0.00 sec)
```

<运算符减少了Heart of the Beest的相关性。

下面的五个例子演示了——<运算符，增加一个递减的、正的匹配的权重；~运算符，对匹配提供了一个负的权重；-运算符，禁止匹配——之间的区别。第一个例子是一个基本的布尔查找，使用权重1来匹配：

```
mysql> SELECT *,MATCH(f1) AGAINST ('Door' IN BOOLEAN MODE)
   AS m FROM ft2 WHERE MATCH(f1) AGAINST ('Door' IN BOOLEAN MODE);
+-----------------------+------+
| f1                    | m    |
+-----------------------+------+
| A Barbarian at my Door |    1 |
+-----------------------+------+
1 row in set (0.00 sec)
```

下面，<运算符把权重大约减少到2/3，但还是一个正的权重：

```
mysql> SELECT *,MATCH(f1) AGAINST ('<Door' IN BOOLEAN MODE)
   AS m FROM ft2 WHERE MATCH(f1) AGAINST ('<Door' IN BOOLEAN MODE);
+-----------------------+-----------------+
| f1                    | m               |
+-----------------------+-----------------+
| A Barbarian at my Door | 0.66666668653488 |
+-----------------------+-----------------+
1 row in set (0.00 sec)
```

~运算符把权重减少到一个负值，因此，因为在与A Barbarian at my Door匹配时，结果小于0，所以不会返回这行：

```
mysql> SELECT *,MATCH(f1) AGAINST ('~Door' IN BOOLEAN MODE)
   AS m FROM ft2 WHERE MATCH(f1) AGAINST ('~Door' IN BOOLEAN MODE);
Empty set (0.00 sec)
```

在一个普通的匹配联结中使用~运算符，就可以看到权重是如何减少的，在这里是0.5：

```
mysql> SELECT *,MATCH(f1) AGAINST ('~Door Barbarian*' IN BOOLEAN MODE)
  AS m FROM ft2 WHERE MATCH(f1) AGAINST ('~Door Barbarian*' IN BOOLEAN MODE);
+-----------------------------+------+
| f1                          | m    |
+-----------------------------+------+
| A Barbarian at my Door      |  0.5 |
| Waiting for the  Barbarians |    1 |
+-----------------------------+------+
2 rows in set (0.01 sec)
```

最后，下面的例子显示了~和-运算符之间的差别，在这个例子中，找到Door的时候，-运算符会禁止进行匹配：

```
mysql> SELECT *,MATCH(f1) AGAINST ('-Door Barbarian*' IN BOOLEAN MODE)
  AS m FROM ft2 WHERE MATCH(f1) AGAINST ('-Door Barbarian*' IN BOOLEAN MODE);
+-----------------------------+------+
| f1                          | m    |
+-----------------------------+------+
| Waiting for the  Barbarians |    1 |
+-----------------------------+------+
1 row in set (0.00 sec)
```

下面的例子演示在一个子表达式中分组单词的方法：

```
mysql> SELECT f1,MATCH(f1) AGAINST ('+Heart +(<Beast >Beast)'
 IN BOOLEAN MODE) As m FROM ft2 WHERE MATCH(f1)
 AGAINST ('+Heart +(<Beast >Beast)' IN BOOLEAN MODE);
+--------------------+-----------------+
| f1                 | m               |
```

```
+----------------------+------------------+
| Heart of the Beast   |             1.25 |
| Heart of the Beest   | 0.83333337306976 |
+----------------------+------------------+
2 rows in set (0.00 sec)
```

+运算符作用在括号中的整个子串中，这意味着至少Beest和Beast其中之一出现在字符串中。In the Heart of the Countr不会出现，因为Beest或Beast都没有出现。用下面的代码比较这一点。

下面的例子演示了一般使用的查找形式，其中提供的单词都是强制性的：

```
mysql> SELECT f1,MATCH(f1) AGAINST ('+Heart +<Beest +>Beast)' IN BOOLEAN MODE)
AS m FROM ft2 WHERE MATCH(f1) AGAINST ('+Heart +<Beest +>Beast)' IN BOOLEAN MODE);
Empty set (0.00 sec)
```

因为没有任何行同时包含有Heart、Beest和Beast，因此不会返回任何行。

下面的两个例子演示在查找中使用""运算符和不使用它的区别。""运算符可以用来对短语进行完全匹配：

```
mysql> SELECT * FROM ft2 WHERE MATCH(f1)
 AGAINST ('the Heart of the' IN BOOLEAN MODE);
+----------------------------+
| f1                         |
+----------------------------+
| In the Heart of the Country |
| Heart of the Beast         |
| Heart of the Beest         |
+----------------------------+
3 rows in set (0.01 sec)

mysql> SELECT * FROM ft2 WHERE MATCH(f1)
 AGAINST ('"the Heart of the"' IN BOOLEAN MODE);
+----------------------------+
| f1                         |
+----------------------------+
| In the Heart of the Country |
+----------------------------+
1 row in set (0.00 sec)
```

注意在使用双引号时，不要忘了开始的单引号。如果忘了，实际上就不会有任何运算符了。比如：

```
mysql> SELECT * FROM ft2 WHERE MATCH(f1)
 AGAINST ("the Heart of the" IN BOOLEAN MODE);
+----------------------------+
| f1                         |
+----------------------------+
| In the Heart of the Country |
```

```
| Heart of the Beast          |
| Heart of the Beest          |
+-----------------------------+
3 rows in set (0.00 sec)
```

警告：全文索引要花费很长时间来产生，因此就会导致优化语句也花费很长的时间。

创建惟一索引

惟一索引除了不容许有重复的记录以外，其他与普通索引一样。

使用下面的语句在创建表的时候创建惟一索引：

```
CREATE TABLE tablename (fieldname columntype, fieldname2
   columntype, UNIQUE(fieldname [,fieldname2...]));
```

或者，在表已经存在的情况下，可以使用下面的语法：

```
ALTER TABLE tablename ADD UNIQUE [indexname ] (fieldname [,fieldname2...]);
```

或者是这个语法：

```
CREATE UNIQUE INDEX indexname ON tablename(fieldname [,fieldname2...]);
```

如果索引只包含一个域，那这个域不能包含重复值：

```
mysql> CREATE TABLE ui_test(f1 INT,f2 INT,UNIQUE(f1));
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO ui_test VALUES(1,2);
Query OK, 1 row affected (0.01 sec)
mysql> INSERT INTO ui_test VALUES(1,3);
ERROR 1062: Duplicate entry '1' for key 1
```

尽管f1域并不是作为惟一创建的，惟一索引也防止了重复的产生。如果索引包含多个域，但单个域值是可以重复的，而组成整个索引的域的组合值是不能重复的：

```
mysql> CREATE TABLE ui_test2(f1 INT,f2 INT,UNIQUE(f1,f2));
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO ui_test2 VALUES(1,2);
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO ui_test2 VALUES(1,3);
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO ui_test2 VALUES(1,3);
ERROR 1062: Duplicate entry '1-3' for key 1
```

从域的部分创建索引

对VARCHAR、CHAR、BLOB和TEXT列，MySQL容许不使用全部的域来创建索引。比如，尽管客户名可以到40个字符，但一般名中的前10个字符就有可能不同。通过只使用前10个字符作为索引，索引会小很多。这会使更新和插入更快（将只有使用全部列的四分之一），并且只要不使索引太短，就不会影响SELECT的速度。如果只把名中的一个字母作为索引，就失去了索引的目的。

用域的部分创建索引，只需在列名后的括号中加上大小即可。比如，用下面的代码在客户表中对姓氏域创建10个字符的索引：

```
mysql> ALTER TABLE customer ADD INDEX (surname(10));
Query OK, 8 rows affected (0.00 sec)
Records: 8  Duplicates: 0  Warnings: 0
```

说明：不能在整个BLOB或TEXT域上创建一个索引（除非是一个全文检索），因此在这种情况下，应指定索引的大小。

使用自动增加域

自动增加域是一个很有用的功能，它容许在每次插入新记录的时候自动增加一个域的值。记录中只有一个域自动增加，而且这个域必须是数字主键或数字惟一索引。

创建自动增加域

在创建新表的时候创建自动增加域的语法如下：

```
CREATE TABLE tablename(fieldname INT AUTO_INCREMENT, [fieldname2...,]
   PRIMARY KEY(fieldname));
Query OK, 0 rows affected (0.00 sec)
```

在一个已经存在的表中创建自动增加域，使用如下的语法：

```
ALTER TABLE tablename MODIFY fieldname columntype AUTO_INCREMENT;
Query OK, 0 rows affected (0.00 sec)
```

customer表中有一个理想的自动增加域，即id域：

```
mysql> SHOW COLUMNS FROM customer;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     |      | PRI | 0       |       |
| first_name | varchar(30) | YES  |     | NULL    |       |
| surname    | varchar(40) | YES  | MUL | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

id域是一个数字域并且已经是一个主键，因为已经把id按顺序赋给日期，把id域改为自动增加域将能使MySQL自动进行这个过程。下面的代码把id域变为了自动增加域：

```
mysql> ALTER TABLE customer MODIFY id INT AUTO_INCREMENT;
Query OK, 7 rows affected (0.01 sec)
Records: 7  Duplicates: 0  Warnings: 0
mysql> SHOW COLUMNS FROM customer;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
```

```
+------------+-------------+------+-----+---------+----------------+
| id         | int(11)     |      | PRI | NULL    | auto_increment |
| first_name | varchar(30) | YES  |     | NULL    |                |
| surname    | varchar(40) | YES  | MUL | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

在包含自动增加域的表中插入记录

现在，如果加入记录，不需要指定id的值，MySQL将自动增加下一最大值：

```
mysql> SELECT * FROM customer;
+----+------------+-------------+
| id | first_name | surname     |
+----+------------+-------------+
|  1 | Yvonne     | Clegg       |
|  2 | Johnny     | Chaka-Chaka |
|  3 | Winston    | Powers      |
|  4 | Patricia   | Mankunku    |
|  5 | Francois   | Papo        |
|  7 | Winnie     | Dlamini     |
|  6 | Neil       | Beneke      |
+----+------------+-------------+
7 rows in set (0.00 sec)

mysql> INSERT INTO customer(first_name,surname) VALUES('Breyton','Tshbalala');
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM customer;
+----+------------+-------------+
| id | first_name | surname     |
+----+------------+-------------+
|  1 | Yvonne     | Clegg       |
|  2 | Johnny     | Chaka-Chaka |
|  3 | Winston    | Powers      |
|  4 | Patricia   | Mankunku    |
|  5 | Francois   | Papo        |
|  7 | Winnie     | Dlamini     |
|  6 | Neil       | Beneke      |
|  8 | Breyton    | Tshbalala   |
+----+------------+-------------+
8 rows in set (0.00 sec)
```

一个最重要的特征是即使记录被删除了，MySQL的自动增加计数器也会记住最近增加的数。这可以保证新插入的记录有一个新的id值并且不和旧的项的任何记录冲突。

```
mysql> DELETE FROM customer WHERE id=8;
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO customer(first_name,surname)
```

```
    VALUES('Breyton','Tshbalala');
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM customer;
+----+------------+-------------+
| id | first_name | surname     |
+----+------------+-------------+
|  1 | Yvonne     | Clegg       |
|  2 | Johnny     | Chaka-Chaka |
|  3 | Winston    | Powers      |
|  4 | Patricia   | Mankunku    |
|  5 | Francois   | Papo        |
|  7 | Winnie     | Dlamini     |
|  6 | Neil       | Beneke      |
|  9 | Breyton    | Tshbalala   |
+----+------------+-------------+
8 rows in set (0.01 sec)
```

id现在为9。尽管下一个最高的保留记录是7，最近插入的值也是8。

返回和重置自动增加值

可以使用LAST_INSERT_ID()函数返回最近插入的自动增加值：

```
mysql> SELECT LAST_INSERT_ID() FROM customer LIMIT 1;
+------------------+
| last_insert_id() |
+------------------+
|                9 |
+------------------+
1 row in set (0.00 sec)
```

这在需要创建新的自动增加值的地方如更新时很有用。比如，接着的代码查找最近插入的自动增加值，为了给Breyton Tshbalala设置新的id值而增加了1：

```
mysql> UPDATE customer set id=LAST_INSERT_ID()+1 WHERE
 first_name='Breyton' AND surname='Tshbalala';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> SELECT * FROM customer;
+----+------------+-------------+
| id | first_name | surname     |
+----+------------+-------------+
|  1 | Yvonne     | Clegg       |
|  2 | Johnny     | Chaka-Chaka |
|  3 | Winston    | Powers      |
|  4 | Patricia   | Mankunku    |
|  5 | Francois   | Papo        |
```

```
|  7 | Winnie     | Dlamini     |
|  6 | Neil       | Beneke      |
| 10 | Breyton    | Tshbalala   |
+----+------------+-------------+
8 rows in set (0.00 sec)
```

警告： LAST_INSERT_ID()在自动增加计数器时存在问题。 参看本章后面的“LAST_INSERT_ID()的问题”小节。

如果想把自动增加计数器重置为特定的值，比如在删除全部记录后，重置为1，就可以使用如下代码：

```
ALTER TABLE tablename AUTO_INCREMENT=auto_inc_value;
```

创建一个测试例子来检查：

```
mysql> CREATE TABLE ai_test(id INT NOT NULL AUTO_INCREMENT,
 f1 VARCHAR(10),PRIMARY KEY (id));
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO ai_test(f1) VALUES('one'),('two');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
mysql> SELECT * FROM ai_test;
+----+------+
| id | f1   |
+----+------+
|  1 | one  |
|  2 | two  |
+----+------+
2 rows in set (0.00 sec)
mysql> DELETE FROM ai_test;
Query OK, 2 rows affected (0.00 sec)
mysql> INSERT INTO ai_test(f1) VALUES('three');
Query OK, 1 row affected (0.01 sec)
mysql> SELECT * FROM ai_test;
+----+-------+
| id | f1    |
+----+-------+
|  3 | three |
+----+-------+
1 row in set (0.00 sec)
```

尽管表已经空了，自动增加计数器还是保留了原值。可以使用TRUNCATE来清除表，这也会重置自动增加计数器：

```
mysql> DELETE FROM ai_test;
Query OK, 1 row affected (0.00 sec)
mysql> ALTER TABLE ai_test AUTO_INCREMENT=1;
```

```
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

这条语句把自动增加计数器重置为了1。

```
mysql> INSERT INTO ai_test(f1) VALUES('four');
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM ai_test;
+----+------+
| id | f1   |
+----+------+
|  1 | four |
+----+------+
1 row in set (0.00 sec)
mysql> TRUNCATE ai_test;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO ai_test(f1) VALUES('five');
Query OK, 1 row affected (0.01 sec)
mysql> SELECT * FROM ai_test;
+----+------+
| id | f1   |
+----+------+
|  1 | five |
+----+------+
1 row in set (0.00 sec)
```

通过使用TRUNCATE而不是DELETE，自动增加计数器被重置了。

警告：现在，这只作用于MyISAM表上。对于其他的表类型需要手动重置自动增加计数器。

把自动增加计数器设置为1以外的值也很容易。比如，可以在创建表的时候设置：

```
mysql> CREATE TABLE ai_test2(id INT NOT NULL AUTO_INCREMENT,
 f1 VARCHAR(5),PRIMARY KEY(id)) AUTO_INCREMENT=50;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO ai_test2(f1) VALUES('one');
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM ai_test2;
+----+------+
| id | f1   |
+----+------+
| 50 | one  |
+----+------+
1 row in set (0.00 sec)
```

也可以在表存在后设置计数器：

```
mysql> DELETE FROM ai_test;
Query OK, 3 rows affected (0.00 sec)
```

```
mysql> ALTER TABLE ai_test AUTO_INCREMENT=1000;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> INSERT INTO ai_test(f1) VALUES('one');
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM ai_test;
+------+------+
| id   | f1   |
+------+------+
| 1000 | one  |
+------+------+
1 row in set (0.01 sec)
```

在大多数情况下当表为空时，应该使用这个特点，但这并不是一定的；即使表中存在记录也可以重置计数器：

```
mysql> ALTER TABLE ai_test2 AUTO_INCREMENT=500;
Query OK, 1 row affected (0.01 sec)
Records: 1  Duplicates: 0  Warnings: 0
mysql> INSERT INTO ai_test2(f1) VALUES('two');
Query OK, 1 row affected (0.01 sec)
mysql> SELECT * FROM ai_test2;
+-----+------+
| id  | f1   |
+-----+------+
|  50 | one  |
| 500 | two  |
+-----+------+
2 rows in set (0.00 sec)
```

说明：当前，这也只作用于MyISAM表。比如，对于InnoDB表，只能把自动增加计数器设置为1。

前面的例子在没有指定id域的情况下插入了记录。如果愿意使用自动增加域的值指定的替代语法，就可以把自动增加域的值设为NULL或0：

```
mysql> INSERT INTO ai_test VALUES(NULL,'two');
Query OK, 1 row affected (0.01 sec)
mysql> INSERT INTO ai_test VALUES(0,'three');
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM ai_test;
+------+-------+
| id   | f1    |
+------+-------+
| 1000 | one   |
| 1001 | two   |
| 1002 | three |
+------+-------+
3 rows in set (0.00 sec)
```

越界

注意即使自动增加计数器关联的域是一个有符号的域，它也只能是一个正数。如果想设置为一个负数，就会遇到奇怪的现象：

```
mysql> ALTER TABLE ai_test2 AUTO_INCREMENT=-500;
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0
mysql> INSERT INTO ai_test2(f1) VALUES('three');
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM ai_test2;
+------------+-------+
| id         | f1    |
+------------+-------+
|         50 | one   |
|        500 | two   |
| 2147483647 | three |
+------------+-------+
3 rows in set (0.00 sec)
```

因为-500超出了自动增加计数器容许值的正数范围，MySQL会把它设置为容许的最大的整数：2147483647。如果想增加另一个记录，会得到一个重复键值的错误，因为MySQL不能再产生更大的值：

```
mysql> INSERT INTO ai_test2(f1) VALUES('four');
ERROR 1062: Duplicate entry '2147483647' for key 1
```

警告： *注意为记录保留足够的空间。如果在SIGNED TINYINT域上创建自动增加计数器，一旦到127，就会产生重复键值的错误。*

LAST_INSERT_ID()的问题

在使用LAST_INSERT_ID()函数的时候，有许多情况会导致问题：

- LAST_INSERT_ID()返回值不是被重置的自动增加计数器的值，而是返回1。
- 数字是以每个连接为基础保存的，因此如果插入是从另外的连接增加的，则这个函数返回的数字不会更新。

下面是一些例子：

```
mysql> SELECT * FROM ai_test2;
+------------+-------+
| id         | f1    |
+------------+-------+
|         50 | one   |
|        500 | two   |
| 2147483647 | three |
+------------+-------+
3 rows in set (0.00 sec)
```

```
mysql> ALTER TABLE ai_test2 AUTO_INCREMENT=501;
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> UPDATE ai_test2 SET id=LAST_INSERT_ID()+1 WHERE f1='three';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

这里id的值应该为501。但是，看下面的例子，就会感到吃惊：

```
mysql> SELECT * FROM ai_test2;
+-----+-------+
| id  | f1    |
+-----+-------+
|  50 | one   |
| 500 | two   |
|   1 | three |
+-----+-------+
3 rows in set (0.00 sec)
```

在重置自动增加计数器时，LAST_INSERT_ID已被重置为1。更糟糕的在下面：

```
mysql> ALTER TABLE ai_test2 AUTO_INCREMENT=501;
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> UPDATE ai_test2 SET id=LAST_INSERT_ID()+1 WHERE f1='two';
ERROR 1062: Duplicate entry '1' for key 1
```

现在有了重复键值，UPDATE失败了。当有多个连接时，会产生其他的错误。打开两个窗口，对数据库建立两个连接。

在Window1中：

```
mysql> TRUNCATE ai_test2;
Query OK, 0 rows affected (0.01 sec)
mysql> INSERT INTO ai_test2(f1) VALUES('one');
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM ai_test2;
+----+------+
| id | f1   |
+----+------+
|  1 | one  |
+----+------+
1 row in set (0.01 sec)
mysql> SELECT LAST_INSERT_ID() FROM ai_test2;
+------------------+
| last_insert_id() |
+------------------+
|                1 |
+------------------+
1 row in set (0.00 sec)
```

现在还没有问题。现在转到第二个窗口，并插入另一个记录。在Window2中：

```
mysql> INSERT INTO ai_test2(f1) VALUES('two');
Query OK, 1 row affected (0.00 sec)
Window1:
mysql> SELECT LAST_INSERT_ID() FROM ai_test2;
+------------------+
| last_insert_id() |
+------------------+
|                1 |
|                1 |
+------------------+
2 rows in set (0.00 sec)
```

应该是2的时候，返回值还是1。现在，如果想用这个值来更新，就会得到熟悉的键值重复的错误：

```
mysql> UPDATE ai_test2 SET id=LAST_INSERT_ID()+1 WHERE f1='one';
ERROR 1062: Duplicate entry '2' for key 1
```

多列索引和自动增加域

对于MyISAM和BDB表，也可以使多列索引的第二个索引域成为一个自动增加域。这在创建数据组的时候很有用。对这个例子，将为职员创建一张表，职员可以根据是否为经理、雇员或合同工来分级，然后在其最上面分配一个指定的职位：

```
mysql> CREATE TABLE staff(rank ENUM('Employee','Manager',
 'Contractor') NOT NULL,position VARCHAR(100),
 id INT NOT NULL AUTO_INCREMENT,PRIMARY KEY(rank,id));
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO staff(rank,position) VALUES
 ('Employee','Cleaner'),
 ('Contractor','Network maintenance'),
 ('Manager','Sales manager');
Query OK, 3 rows affected (0.01 sec)
mysql> SELECT * FROM staff;
+------------+---------------------+----+
| rank       | position            | id |
+------------+---------------------+----+
| Employee   | Cleaner             |  1 |
| Contractor | Network maintenance |  1 |
| Manager    | Sales manager       |  1 |
+------------+---------------------+----+
3 rows in set (0.00 sec)
```

三个记录全部都有相同的id值，主键包含两个域：rank和id。如果开始对每个级别添加其他的记录，就会看到熟悉的自动增加的现象：

```
mysql> INSERT INTO staff(rank,position) VALUES
 ('Employee','Security guard'),
 ('Employee', 'Receptionist'),
 ('Manager','Head of security');
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> SELECT * FROM staff;
+------------+--------------------+----+
| rank       | position           | id |
+------------+--------------------+----+
| Employee   | Cleaner            |  1 |
| Contractor | Network maintenance |  1 |
| Manager    | Sales manager      |  1 |
| Employee   | Security guard     |  2 |
| Employee   | Receptionist       |  3 |
| Manager    | Head of security   |  2 |
+------------+--------------------+----+
6 rows in set (0.01 sec)
```

在这个例子中，有职员1、2和3；经理1和2；合同工1。自动增加计数器对每组都保持了正确的值。

但是，在下面的情况下，不能重置自动增加计数器：

```
mysql> ALTER TABLE staff AUTO_INCREMENT=500;
Query OK, 6 rows affected (0.01 sec)
Records: 6  Duplicates: 0  Warnings: 0

mysql> INSERT INTO staff(rank,position) VALUES
 ('Employee','Stationary administrator'),
 ('Manager','Personnel manager'),
 ('Contractor','Programmer');
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> SELECT * FROM staff;
+------------+--------------------------+----+
| rank       | position                 | id |
+------------+--------------------------+----+
| Employee   | Cleaner                  |  1 |
| Contractor | Network maintenance      |  1 |
| Manager    | Sales manager            |  1 |
| Employee   | Security guard           |  2 |
| Employee   | Receptionist             |  3 |
| Manager    | Head of security         |  2 |
| Employee   | Stationary administrator |  4 |
| Manager    | Personnel manager        |  3 |
| Contractor | Programmer               |  2 |
```

```
+------------+-------------------------+----+
9 rows in set (0.00 sec)
```

自动增加计数器还是从它们停下的地方继续，而没有考虑ALTER语句。

删除或改变索引

有时，超过生命期的索引是没有用的，因此需要改变或删除它。不管对索引进行何种改变，都需要先删除然后用新定义重新构建。

使用下面的语法删除主键：

```
ALTER TABLE tablename DROP PRIMARY KEY;
```

删除普通、惟一或全文索引，则需要指定索引名：

```
ALTER TABLE tablename DROP INDEX indexname;
```

或者是这样：

```
DROP INDEX indexname ON tablename;
```

如果不能确定索引名，可以使用SHOW KEYS来显示全部的索引名：

```
SHOW KEYS FROM tablename;
```

理解表类型和索引

出现在索引中的表类型都有自己的行为特征，处理方式都是不同的。并不是每一个表类型都有合适的索引类型。在把索引提交给表类型之前，理解如何使用一个表以及需要何种索引是很重要的。有时因为某种类型的索引不可用而使看起来理想的表类型也变得没用了。下面的列表标出了针对每个表类型的索引的特点和区别。

MyISAM表具有下面的属性：

- 索引存在一个扩展名为.MYI的文件中。
- 数字索引是按照高字节为先的方式来存储的，以方便索引的压缩。
- 可以存在BLOB和TEXT索引。
- 在索引中，可以容许Null值（非主键）。
- 数据和索引可以在不同的路径（速度更快）中。

MERGE表有下面的属性：

- MERGE表不包含自己的索引。
- MRG文件包含从MyISAM表的部件得到的.MYI索引文件列表。
- 在创建MERGE表时，还需要指定索引。

HEAP表有下面的属性：

- 使用存储在内存中的散列索引，速度非常快。
- 只能使用带=和<=>运算符的索引。
- 不能使用某列可以带NULL值的索引。

- 索引不能使用ORDER BY从句。
- MySQL不能近似计算两个值之间的行数（这个结果由查询优化器使用来判断哪个索引使用效率更高）。对于这个问题，可以参看本章后面小节的“使用ANALYZE帮助MySQL查询优化器”。

ISAM表使用的B-Tree索引存储在扩展名为.ism.的文件中。

InnoDB表不能使用全文索引。

高效使用索引

索引非常少的表返回结果非常慢。但增加太多的索引，尽管很少见，也会有问题——占用空间，而且，因为索引是排序的，每次进行插入或更新时，索引都必须针对变化重新排序，这会导致很多额外负担。接下来的小节解释何时使用索引，何时不用。同时要注意索引的效率也依赖于配置；关于这一点，可参看第13章“配置并优化MySQL”。

何处使用索引

使用索引的最普遍之处是获取与WHERE从句中条件匹配的行：

```
mysql> SELECT first_name FROM customer WHERE surname>'C';
+------------+
| first_name |
+------------+
| Yvonne     |
| Johnny     |

| Winston    |
| Patricia   |
| Francois   |
| Winnie     |
| Breyton    |
+------------+
7 rows in set (0.00 sec)
```

在这个例子中，surname域的索引很有用。在这个查询中first_name域的索引没有用，因为它是条件的一部分。只出现在域列表（紧接着SELECT的）中的任何域都不能利用索引。

查找MAX()或MIN()值时，MySQL只需要在排序的索引表中查找第一个和最后一个值，这非常快。如果需要经常查找MAX()或MIN()值，建立针对适当域的索引会非常有用。

```
mysql> SELECT MAX(id) FROM customer;
+---------+
| MAX(id) |
+---------+
|      10 |
+---------+
```

针对id域的索引会对查询非常有用。

还有一种情况，就是当返回的域都是索引的一部分时，MySQL不需要查看全表而只需要看索引。考虑这个例子：

```
mysql> SELECT id FROM customer;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
|  4 |
|  5 |
|  6 |
|  7 |
| 10 |
+----+
8 rows in set (0.01 sec)
```

如果这里的id域是被索引的，MySQL甚至不需要查看数据文件。但如果索引只包含全部列数据的一部分（比如，域是20个字符的VARCHAR，但索引是以前10个字符为基础创建的）时，就无效了。

另一种索引非常有用的情况是对域使用ORDER BY的地方，就像这个例子：

```
mysql> SELECT * FROM customer ORDER BY surname;
+----+------------+-------------+
| id | first_name | surname     |
+----+------------+-------------+
|  6 | Neil       | Beneke      |
|  2 | Johnny     | Chaka-Chaka |
|  1 | Yvonne     | Clegg       |
|  7 | Winnie     | Dlamini     |
|  4 | Patricia   | Mankunku    |
|  5 | Francois   | Papo        |
|  3 | Winston    | Powers      |
| 10 | Breyton    | Tshbalala   |
+----+------------+-------------+
8 rows in set (0.01 sec)
```

因为记录是按照排序的方式返回的，刚好与索引完全相同，surname域上的索引对这个查询就非常有用。如果ORDER BY是DESC，索引只需要简单地反序读即可。

警告：当前，索引不能使用在HEAP表中的ORDER BY从句中。

索引也可以被用来加速连接，如下例子所示：

```
mysql> SELECT first_name,surname,commission FROM sales,sales_rep
 WHERE code=8 AND sales.sales_rep=sales_rep.employee_number;
+------------+---------+------------+
| first_name | surname | commission |
+------------+---------+------------+
| Mike       | Serote  |         10 |
+------------+---------+------------+
1 row in set (0.00 sec)
```

索引可以用来进行连接条件——换句话说，查找sales.sales_rep和sales_rep.employee_number域。这甚至可以在使用替代的语法时也使用。

```
mysql> SELECT first_name,surname,commission FROM sales INNER
 JOIN sales_rep ON sales.sales_rep=sales_rep.employee_number
 WHERE code=8;
+------------+---------+------------+
| first_name | surname | commission |
+------------+---------+------------+
| Mike       | Serote  |         10 |
+------------+---------+------------+
1 row in set (0.00 sec)
```

索引也可以在查询条件包含通配符的情况下使用，比如：

```
mysql> SELECT * FROM sales_rep WHERE surname LIKE 'Ser%';
+------------+---------+------------+
| first_name | surname | commission |
+------------+---------+------------+
| Mike       | Serote  |         10 |
+------------+---------+------------+
1 row in set (0.00 sec)
```

但索引不能在下面的情况中使用：

```
mysql> SELECT * FROM sales_rep WHERE surname LIKE '%Ser%';
+------------+---------+------------+
| first_name | surname | commission |
+------------+---------+------------+
| Mike       | Serote  |         10 |
+------------+---------+------------+
1 row in set (0.00 sec)
```

与前面的区别是后者有一个通配符作为第一个字符。因为索引是按照第一个字符的字母顺序排序的，通配符的出现致使索引无效。

选择索引

现在已经知道在何处使用MySQL了，下面是几个帮助选择索引的技巧。

- 不用说，有查询需要使用索引（比如，WHERE条件中的域）的时候，需要创建索引，不要针对不使用的域（比如第一个字符的条件是一个通配符）来创建索引。
- 创建的索引返回的行越少越好。主键是最好的，因为主键只和惟一的记录相关联。同样，枚举域上的索引不是特别有用（比如，包含值yes或no的域的索引只能减少一半的查询，却带来了对索引的大量维护）。
- 使用短索引（比如，名字的头10个字符的索引，而不是全部域）。
- 不要创建太多的索引。索引增加了更新和添加记录的时间，因此如果索引在查询中很少使用，而没有索引只是轻微地降低速度，就不要创建这个索引。

使用最左边的前缀

已经知道可以对一个以上的域创建索引。surnames和first names是说明这点的非常好的例子。尽管有可能有很多的重复surnames、first name都会使索引近似惟一。实际上，这里对MySQL有两个索引可用。第一个是surname和first_name，而第二个仅仅是surname。从索引中的域列表的最左边开始，MySQL可以顺序使用它们，只要它们保持从左开始的顺序。下面的例子会使这点更清楚。添加一个customer表的初始域并增加一些值，以及一个索引：

```
mysql> ALTER TABLE customer ADD initial VARCHAR(5);
Query OK, 8 rows affected (0.01 sec)
Records: 8  Duplicates: 0  Warnings: 0
mysql> ALTER TABLE customer ADD INDEX (surname,initial, first_name);
Query OK, 8 rows affected (0.01 sec)
Records: 8  Duplicates: 0  Warnings: 0
mysql> UPDATE customer SET initial='X' WHERE id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> UPDATE customer SET initial='B' WHERE id=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> UPDATE customer SET initial='M' WHERE id=3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> UPDATE customer SET initial='C' WHERE id=4;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> UPDATE customer SET initial='P' WHERE id=5;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> UPDATE customer SET initial='B' WHERE id=10;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

如果在查询条件中使用了这些域中的全部三个，就会最大限度地利用索引：

```
mysql> SELECT * FROM customer WHERE surname='Clegg' AND
 initial='X' AND first_name='Yvonne';
```

如果查询surname和initial，就可以利用索引的大部分：

```
mysql> SELECT * FROM customer WHERE surname='Clegg' AND initial='X';
```

或者仅仅是surname：

```
mysql> SELECT * FROM customer WHERE surname='Clegg';
```

但是，如果打破了最左顺序并查找first name或initial之一，或first name和initial两者，MySQL就不会使用索引。比如，下面的例子不会利用索引：

```
mysql> SELECT * FROM customer WHERE initial='X' AND first_name='Yvonne';
mysql> SELECT * FROM customer WHERE first_name='Yvonne';
mysql> SELECT * FROM customer WHERE initial='X';
```

如果对索引中的第一和第三个域（surname和first_name）进行查找，也会破坏顺序，因此索引也不会被完全利用。但是，因为surname是索引的第一部分，索引的这一部分还会被利用：

```
mysql> SELECT * FROM customer WHERE surname='Clegg' AND first_name='Yvonne';
+----+------------+---------+---------+
| id | first_name | surname | initial |
+----+------------+---------+---------+
|  1 | Yvonne     | Clegg   | X       |
+----+------------+---------+---------+
1 row in set (0.00 sec)
```

可以在任何索引被使用的时候，都使用最左前缀，比如在ORDER BY从句中。

使用EXPLAIN分析MySQL如何使用索引

MySQL中保守的最好的秘密之一就是EXPLAIN语句。即使使用了多年MySQL的人也似乎忽略了这条语句，而它可以使生活更容易。

EXPLAIN显示（解释！）了MySQL如何使用索引来处理SELECT语句以及连接表。它可以帮助选择更好的索引和写出更优化的查询语句。

要使用它，只要在SELECT语句前加上它就可以了：

```
mysql> EXPLAIN SELECT surname,first_name FROM customer WHERE id=1;
+----------+-------+---------------+---------+---------+-------+------+-------+
| table    | type  | possible_keys | key     | key_len | ref   | rows | Extra |
+----------+-------+---------------+---------+---------+-------+------+-------+
| customer | const | PRIMARY       | PRIMARY |       4 | const |    1 |       |
+----------+-------+---------------+---------+---------+-------+------+-------+
1 row in set (0.00 sec)
```

这些行的意思是什么？见表4.2的解释。

表4.2 EXPLAIN列的意义

列	描述
table	显示其余的行是关于哪张表的（在这个例子中显得很多余，但在查询中连接了一个以上的表时很有用）
type	这是重要的列，显示连接使用了何种类型。从最好到最差的连接类型为const、eq_ref、ref、range、index和ALL
possible_keys	显示可能应用在这张表中的索引。如果为空，没有可能的索引。可以为相关的域从WHERE语句中选择一个合适的
Key	实际使用的索引。如果为NULL，则没有使用索引。很少的情况下，MySQL会选择优化不足的索引。这种情况下，可以在SELECT语句中使用USE INDEX (indexname)来强制使用一个索引或者用IGNORE INDEX(indexname)来强制MySQL忽略索引
key_len	使用的索引的长度。在不损失精确性的情况下，长度越短越好
ref	显示索引的哪一列被使用了，如果可能的话，是一个常数
rows	MySQL认为必须检查的用来返回请求数据的行数
extra	关于MySQL如何解析查询的额外信息。将在表4.3中讨论，但这里可以看到的坏的例子是Using temporary和Using filesort，意思是MySQL根本不能使用索引，结果是检索会很慢

表4.3分析extra列返回的描述的意义。

表4.3 extra EXPLAIN列返回的描述的意义

EXTRA COLUMN描述	描述
Distinct	一旦MySQL找到了与行联合匹配的行，就不再搜索了
Not exists	MySQL优化LEFT JOIN，一旦它找到了匹配LEFT JOIN标准的行，就不再搜索了
Range checked for each record (index map: #)	没有找到理想的索引，因此对于从前面表中来的每一个行组合，MySQL检查使用哪个索引，并用它来从表中返回行。这是使用索引的最慢的连接之一
Using filesort	看到这个的时候，查询就需要优化了。MySQL需要进行额外的步骤来发现如何对返回的行排序。它根据连接类型以及存储排序键值和匹配条件的全部行的行指针来排序全部行
Using index	列数据是从仅仅使用了索引中的信息而没有读取实际的行的表返回的，这发生在对表的全部的请求列都是同一个索引的部分的时候

（续表）

EXTRA COLUMN描述	描述
Using temporary	看到这个的时候，查询需要优化了。这里，MySQL需要创建一个临时表来存储结果，这通常发生在对不同的列集进行ORDER BY上，而不是GROUP BY上
Where used	使用了WHERE从句来限制哪些行将与下一张表匹配或者是返回给用户。如果不想返回表中的全部行，并且连接类型是ALL或index，这就会发生，或者是查询有问题

EXPLAIN返回的类型列显示了使用的连接类型。表4.4解释了这些连接类型，并按照效率高低的顺序排序。

表4.4 不同的连接类型

连接	描述
system	表只有一行：system表。这是const连接类型的特殊情况
const	表中的一个记录的最大值能够匹配这个查询（索引可以是主键或惟一索引）。因为只有一行，这个值实际就是常数，因为MySQL先读这个值然后把它当做常数来对待
eq_ref	在连接中，MySQL在查询时，从前面的表中，对每一个记录的联合都从表中读取一个记录，它在查询使用了索引为主键或惟一键的全部时使用
ref	这个连接类型只有在查询使用了不是惟一或主键的键或者是这些类型的部分（比如，利用最左前缀）时发生。对于之前的表的每一个行联合，全部记录都将从表中读出。这个连接类型严重依赖于根据索引匹配的记录多少——越少越好
range	这个连接类型使用索引返回一个范围中的行，比如使用>或<查找东西时发生的情况
index	这个连接类型对前面的表中的每一个记录联合进行完全扫描（比ALL更好，因为索引一般小于表数据）
ALL	这个连接类型对前面的表中的每一个记录联合进行完全扫描，这一般比较糟糕，应该尽量避免

返回到例子：

```
mysql> EXPLAIN SELECT surname,first_name FROM customer WHERE id=1;
+----------+-------+---------------+---------+---------+-------+------+-------+
| table    | type  | possible_keys | key     | key_len | ref   | rows | Extra |
+----------+-------+---------------+---------+---------+-------+------+-------+
| customer | const | PRIMARY       | PRIMARY |       4 | const |    1 |       |
+----------+-------+---------------+---------+---------+-------+------+-------+
1 row in set (0.00 sec)
```

在customer表的id域上已经有了一个主键，并且，因为在查询中只有一个条件，id域等同于一个常数，查询已尽可能地优化了。rows列显示MySQL只需要查找一列来返回结果。

不可能比这更好了。同样，连接（这种情况，不是真正的连接）的类型是const，即代表常数，也是最好的类型。看一下如果在没有索引的表上执行同样的查询会发生什么：

```
mysql> EXPLAIN SELECT * FROM sales_rep WHERE employee_number=2;
+-----------+------+---------------+------+---------+------+------+------------+
| table     | type | possible_keys | key  | key_len | ref  | rows | Extra      |
+-----------+------+---------------+------+---------+------+------+------------+
| sales_rep | ALL  | NULL          | NULL |    NULL | NULL |    5 | where used |
+-----------+------+---------------+------+---------+------+------+------------+
1 row in set (0.00 sec)
```

这个查询非常坏。连接类型是ALL，即最坏的情况，没有可能的键值，并且检查了5行来返回结果（现在在sales_rep表中只有5个记录）。看看如何改善这种情况：

```
mysql> SHOW COLUMNS FROM sales_rep;
+-----------------+------------+------+-----+---------+-------+
| Field           | Type       | Null | Key | Default | Extra |
+-----------------+------------+------+-----+---------+-------+
| employee_number | int(11)    | YES  |     | NULL    |       |
| surname         | varchar(40)| YES  |     | NULL    |       |
| first_name      | varchar(30)| YES  |     | NULL    |       |
| commission      | tinyint(4) | YES  |     | NULL    |       |
| date_joined     | date       | YES  |     | NULL    |       |
| birthday        | date       | YES  |     | NULL    |       |
+-----------------+------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
mysql> SELECT * FROM sales_rep;
+-----------------+----------+------------+------------+-------------+------------+
| employee_number | surname  | first_name | commission | date_joined | birthday   |
+-----------------+----------+------------+------------+-------------+------------+
|               1 | Rive     | Sol        |         10 | 2000-02-15  | 1976-03-18 |
|               2 | Gordimer | Charlene   |         15 | 1998-07-09  | 1958-11-30 |
|               3 | Serote   | Mike       |         10 | 2001-05-14  | 1971-06-18 |
|               4 | Rive     | Mongane    |         10 | 2002-11-23  | 1982-01-04 |
|               5 | Jomo     | Ignesund   |         10 | 2002-11-29  | 1968-12-01 |
+-----------------+----------+------------+------------+-------------+------------+
5 rows in set (0.01 sec)
```

当然对主键的选择是employee_number域。它没有重复值，也不需要，因此没有任何原因不使用它做主键。

```
mysql> ALTER TABLE sales_rep MODIFY employee_number INT NOT NULL
    PRIMARY KEY;
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

如果重新在查询中执行EXPLAIN就立即会看见结果。

```
mysql> EXPLAIN SELECT * FROM sales_rep WHERE employee_number=2;
+-----------+-------+---------------+---------+---------+-------+------+-------+
| table     | type  | possible_keys | key     | key_len | ref   | rows | Extra |
+-----------+-------+---------------+---------+---------+-------+------+-------+
| sales_rep | const | PRIMARY       | PRIMARY |       4 | const |    1 |       |
+-----------+-------+---------------+---------+---------+-------+------+-------+
1 row in set (0.00 sec)
```

这是巨大的提高。

如果在表名前使用EXPLAIN，它就是DESCRIBE tablename或SHOW COLUMNS FROM tablename的同意词。

在查询中计算

看一下更复杂的情形。比如，需要返回对佣金小于百分之二十的销售代表的佣金增加百分之五后的结果。下面是使用EXPLAIN的可能查询：

```
mysql> EXPLAIN SELECT * FROM sales_rep WHERE (commission+5)<20;
+-----------+------+---------------+------+---------+------+------+-------------+
| table     | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+-----------+------+---------------+------+---------+------+------+-------------+
| sales_rep | ALL  | NULL          | NULL |    NULL | NULL |    5 | where used  |
+-----------+------+---------------+------+---------+------+------+-------------+
```

还不太好。这个查询没有利用任何索引。这不奇怪，因为只有一个索引，即针对employee_number的主键。似乎对commission域增加一个索引将使事情有很大改观。因为commission域有重复值，它不能是主键（每张表只容许有一个）或者是惟一键。因为它不是字符域，这个惟一的键只是一个普通索引：

```
mysql> ALTER TABLE sales_rep ADD INDEX(commission);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

现在，如果重新执行查询，会得到如下结果：

```
mysql> EXPLAIN SELECT * FROM sales_rep WHERE (commission+5)<20;
+-----------+------+---------------+------+---------+------+------+-------------+
| table     | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+-----------+------+---------------+------+---------+------+------+-------------+
| sales_rep | ALL  | NULL          | NULL |    NULL | NULL |    5 | where used  |
+-----------+------+---------------+------+---------+------+------+-------------+
1 row in set (0.00 sec)
```

没有改进！MySQL还是在检查每个记录。原因是每一个记录都计算了commission+5。为了给commission域加5然后与常数20比较，要从每个记录中读出commission域。不要尝试在索引域中进行任何计算。解决方法是在常数上进行计算，不要在索引域上进行。在代数式上，（x + 5 < y）等同于（x < y - 5），因此可以重写查询：

```
mysql> EXPLAIN SELECT * FROM sales_rep WHERE commission<20-5;
+-----------+-------+---------------+------------+---------+------+------+-------------+
| table     | type  | possible_keys | key        | key_len | ref  | rows | Extra       |
+-----------+-------+---------------+------------+---------+------+------+-------------+
| sales_rep | range | commission    | commission |       2 | NULL |    3 | where used  |
+-----------+-------+---------------+------------+---------+------+------+-------------+
1 row in set (0.00 sec)
```

这要好一些。MySQL只进行一次运算，得到常数15，并能够用索引查找比此值小的数。可以输入如下的查询：

```
mysql> EXPLAIN SELECT * FROM sales_rep WHERE commission<15;
+-----------+-------+---------------+------------+---------+------+------+-------------+
| table     | type  | possible_keys | key        | key_len | ref  | rows | Extra       |
+-----------+-------+---------------+------------+---------+------+------+-------------+
| sales_rep | range | commission    | commission |       2 | NULL |    3 | where used  |
+-----------+-------+---------------+------------+---------+------+------+-------------+
1 row in set (0.00 sec)
```

这里自己计算出这个常数，但速度并没有显著改变，从20减去5花费MySQL很少的时间（否则你至少要测量它一次！）。注意如果返回在增加百分之五佣金后佣金刚好是百分之二十的销售代表时，会发生什么：

```
mysql> EXPLAIN SELECT * FROM sales_rep WHERE commission=15;
+-----------+------+---------------+------------+---------+-------+------+-------------+
| table     | type | possible_keys | key        | key_len | ref   | rows | Extra       |
+-----------+------+---------------+------------+---------+-------+------+-------------+
| sales_rep | ref  | commission    | commission |       2 | const |    1 | where used  |
+-----------+------+---------------+------------+---------+-------+------+-------------+
1 row in set (0.00 sec)
```

查询类型从range改变为ref，这更好使用。这很容易理解，返回一个准确值比返回一个范围的值（或许多准确值）花的时间要少。

在最左前缀中使用EXPLAIN

重新讨论一下最左前缀的话题，看看EXPLAIN能如何帮助更好地理解它。考虑下面的查询：

```
mysql> EXPLAIN SELECT * FROM customer WHERE first_name='Yvonne';
+----------+------+---------------+------+---------+------+------+-------------+
| table    | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+----------+------+---------------+------+---------+------+------+-------------+
| customer | ALL  | NULL          | NULL | NULL    | NULL |    8 | where used  |
+----------+------+---------------+------+---------+------+------+-------------+
1 row in set (0.01 sec)
```

因为first_name域不是索引的最左部分，对索引来说是没用的，连接类型ALL清楚地说明了这点。下面的例子显示了EXPLAIN怎么构成了最左前缀：

```
mysql> EXPLAIN SELECT * FROM customer WHERE surname='Clegg'
 AND initial='X' AND first_name='Yvonne';
+----------+------+---------------+---------+---------+-------------------+------+-------------+
| table    | type | possible_keys | key     | key_len | ref               | rows | Extra       |
+----------+------+---------------+---------+---------+-------------------+------+-------------+
| customer | ref  | surname       | surname |      78 | const,const,const |    1 | where used |
+----------+------+---------------+---------+---------+-------------------+------+-------------+
1 row in set (0.01 sec)
```

这个例子中，索引中的全部三个域都被用了，因为问题中的索引容许重复，所以使用的连接类型是ref。如果表结构排除了surname、initial和first_name的重复组合，连接类型就应该是eq_ref。注意ref列，const,const,const，表明索引的全部三个部分都与一个常数比较。下面的例子演示了相同的情况：

```
mysql> EXPLAIN SELECT * FROM customer WHERE
   surname='Clegg' AND initial='X';
+----------+------+---------------+---------+---------+-------------+------+-------------+
| table    | type | possible_keys | key     | key_len | ref         | rows | Extra       |
+----------+------+---------------+---------+---------+-------------+------+-------------+
| customer | ref  | surname       | surname |      47 | const,const |    1 | where used |
+----------+------+---------------+---------+---------+-------------+------+-------------+
1 row in set (0.00 sec)
```

这里，索引也使用正确，但只使用了前两个域。键值的长度很短（即MySQL扫描的内容较少因此快点）。下面的情况没有利用最左前缀：

```
mysql> EXPLAIN SELECT * FROM customer WHERE initial='X';
+----------+------+---------------+------+---------+------+------+-------------+
| table    | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+----------+------+---------------+------+---------+------+------+-------------+
| customer | ALL  | NULL          | NULL |    NULL | NULL |    8 | where used |
+----------+------+---------------+------+---------+------+------+-------------+
1 row in set (0.00 sec)
```

这个例子没有遵循最左前缀的原则而且没有利用索引。下面的例子也没有利用最左前缀，但利用了索引：

```
mysql> EXPLAIN SELECT * FROM customer WHERE surname='Clegg'
AND first_name='Yvonne';
+----------+------+---------------+---------+---------+-------+------+-------------+
| table    | type | possible_keys | key     | key_len | ref   | rows | Extra       |
+----------+------+---------------+---------+---------+-------+------+-------------+
| customer | ref  | surname       | surname |      41 | const |    1 | where used |
+----------+------+---------------+---------+---------+-------+------+-------------+
1 row in set (0.00 sec)
```

尽管这里没有使用最左前缀，但first_name域不在顺序之中，surname也能够利用好索引。在这种情况下，它把MySQL需要扫描的行数限制为1，因为surname Clegg是惟一的，不用考虑任何first names或initials。

优化Selects

在连接中，可以通过把全部行相乘来计算MySQL需要查询的行数。在下面的例子中，MySQL需要检查5×8×1行，总数为40：

```
mysql> EXPLAIN SELECT * FROM customer,sales_rep,sales
 WHERE sales.sales_rep=sales_rep.employee_number
 AND customer.id=sales.id;
+----------+--------+---------------+---------+---------+----------+------+------------+
| table    | type   | possible_keys | key     | key_len | ref      | rows | Extra      |
+----------+--------+---------------+---------+---------+----------+------+------------+
| sales_rep | ALL    | PRIMARY       | NULL    |    NULL | NULL     |    5 |            |
| sales    | ALL    | NULL          | NULL    |    NULL | NULL     |    8 | where used |
| customer | eq_ref | PRIMARY       | PRIMARY |       4 | sales.id |    1 |            |
+----------+--------+---------------+---------+---------+----------+------+------------+
3 rows in set (0.00 sec)
```

可以看到，连接的表越多，查询的行数也越多。好的数据库设计就要在需要更多连接的小数据库表与难以维护的大表之间的平衡。第8章“范式化数据库”介绍了一些有用的技巧来达到这个目的。

为了优化，我们将集中在前面查询中的sales_rep和sales之间的连接上。重新创建那个连接（使用替代的LEFT JOIN语法）：

```
mysql> EXPLAIN SELECT * FROM sales_rep LEFT JOIN sales
  ON sales.sales_rep = sales_rep.employee_number;
+----------+------+---------------+------+---------+------+------+-------+
| table    | type | possible_keys | key  | key_len | ref  | rows | Extra |
+----------+------+---------------+------+---------+------+------+-------+
| sales_rep | ALL  | NULL          | NULL |    NULL | NULL |    5 |       |
| sales    | ALL  | NULL          | NULL |    NULL | NULL |    8 |       |
+----------+------+---------------+------+---------+------+------+-------+
2 rows in set (0.00 sec)
```

这个例子中要检查的行数为5乘8（从行列），即40。这里没有使用索引。如果检查前两个表的结构，就会明白这是为什么：

```
mysql> DESCRIBE sales;
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| code      | int(11) |      | PRI | 0       |       |
| sales_rep | int(11) | YES  |     | NULL    |       |
| id        | int(11) | YES  |     | NULL    |       |
| value     | int(11) | YES  |     | NULL    |       |
+-----------+---------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

```
mysql> DESCRIBE sales_rep;
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| employee_number | int(11)     |      | PRI | 0       |       |
| surname         | varchar(40) | YES  |     | NULL    |       |
| first_name      | varchar(30) | YES  |     | NULL    |       |
| commission      | tinyint(4)  | YES  | MUL | NULL    |       |
| date_joined     | date        | YES  |     | NULL    |       |
| birthday        | date        | YES  |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

因为没有WHERE条件，查询将返回第一张表sales_rep的全部记录。连接在sales_rep表（因为返回全部记录，因此不能使用索引）的employee_number域与sales表（没有建立索引）的sales_rep域之间进行。问题是连接条件没有利用索引。如果给sales_rep域增加一个索引，就能提高效率：

```
mysql> CREATE INDEX sales_rep ON sales(sales_rep);
Query OK, 8 rows affected (0.01 sec)
Records: 8  Duplicates: 0  Warnings: 0
mysql> EXPLAIN SELECT * FROM sales_rep LEFT JOIN sales
 ON sales.sales_rep = sales_rep.employee_number;
+-----------+------+---------------+-----------+---------+-----------------------------+------+-------+
| table     | type | possible_keys | key       | key_len | ref                         | rows | Extra |
+-----------+------+---------------+-----------+---------+-----------------------------+------+-------+
| sales_rep | ALL  | NULL          | NULL      |    NULL | NULL                        |    5 |       |
| sales     | ref  | sales_rep     | sales_rep |       5 | sales_rep.employee_number   |    2 |       |
+-----------+------+---------------+-----------+---------+-----------------------------+------+-------+
2 rows in set (0.00 sec)
```

已经把MySQL需要读取的行数从40减少到了10，提高了4倍。

如果从另一个角度执行LEFT JOIN（让sales表包含可能的nulls，而不是sales_rep表），用EXPLAIN就会得到不同的结果：

```
mysql> EXPLAIN SELECT * FROM sales LEFT JOIN sales_rep ON
 sales.sales_rep = sales_rep.employee_number;
+-----------+--------+---------------+---------+---------+-----------------+------+-------+
| table     | type   | possible_keys | key     | key_len | ref             | rows | Extra |
+-----------+--------+---------------+---------+---------+-----------------+------+-------+
| sales     | ALL    | NULL          | NULL    |    NULL | NULL            |    8 |       |
| sales_rep | eq_ref | PRIMARY       | PRIMARY |       4 | sales.sales_rep |    1 |       |
+-----------+--------+---------------+---------+---------+-----------------+------+-------+
2 rows in set (0.00 sec)
```

只需要检查8行，因为尽管sales表中的行都返回了，但sales_rep表（employee_number）中的主键被用来进行了连接。

进一步看下面的例子：

```
mysql> EXPLAIN SELECT * FROM sales_rep LEFT JOIN sales
 ON sales.sales_rep = sales_rep.employee_number
 WHERE sales.sales_rep IS NULL;
```

```
+-----------+------+---------------+-----------+---------+--------------------------+------+-------------+
| table     | type | possible_keys | key       | key_len | ref                      | rows | Extra       |
+-----------+------+---------------+-----------+---------+--------------------------+------+-------------+
| sales_rep | ALL  | NULL          | NULL      |    NULL | NULL                     |    5 |             |
| sales     | ref  | sales_rep     | sales_rep |       5 | sales_rep.employee_number |    2 | where used |
+-----------+------+---------------+-----------+---------+--------------------------+------+-------------+
2 rows in set (0.00 sec)
```

现在，如果改变sales_rep域，不允许其出现null值，就会看到一些变化：

```
mysql> ALTER TABLE sales CHANGE sales_rep sales_rep INT NOT NULL;
Query OK, 8 rows affected (0.00 sec)
Records: 8  Duplicates: 0  Warnings: 0
mysql> EXPLAIN SELECT * FROM sales_rep LEFT JOIN sales ON
 sales.sales_rep = sales_rep.employee_number WHERE
 sales.sales_rep IS NULL;
+-----------+------+---------------+-----------+---------+--------------------------+------+--------------------------+
| table     | type | possible_keys | key       | key_len | ref                      | rows | Extra                    |
+-----------+------+---------------+-----------+---------+--------------------------+------+--------------------------+
| sales_rep | ALL  | NULL          | NULL      | NULL    | NULL                     |    5 |                          |
| sales     | ref  | sales_rep     | sales_rep |       4 | sales_rep.employee_number |    2 | where used; Not exists |
+-----------+------+---------------+-----------+---------+--------------------------+------+--------------------------+
2 rows in set (0.00 sec)
```

注意索引长度（key_len显示的）现在是4，而不是5。因为不再容许null，每个记录都不必存储，不论域是否为空，因此长度少1字节。同样，注意extra列的Not exists注释。因为sales_rep域不能再包含null，一旦MySQL发现了与LEFT JOIN标准匹配的记录，就不需要进行任何的查询了。

在某些情况下，表在MySQL中出现的顺序也会影响查询的速度。MySQL尽量选择最好的选项，但它并不能总是知道最快的路径。下面的小节将解释如何使MySQL尽量多地掌握索引组合的信息，但是，就像下面的例子演示的一样，有时这还不够：

首先，创建四个相同的表：

```
mysql> CREATE TABLE t1 (f1 int unique not null, primary key(f1));
Query OK, 0 rows affected (0.15 sec)
mysql> CREATE TABLE t2 (f2 int unique not null, primary key(f2));
Query OK, 0 rows affected (0.15 sec)
mysql> CREATE TABLE t3 (f3 int unique not null, primary key(f3));
Query OK, 0 rows affected (0.15 sec)
mysql> CREATE TABLE t4 (f4 int unique not null, primary key(f4));
Query OK, 0 rows affected (0.15 sec)
```

接着，每个表加两个记录：

```
mysql> INSERT INTO t1 VALUES(1),(2);
Query OK, 2 rows affected (0.12 sec)
Records: 2  Duplicates: 0  Warnings: 0
mysql> INSERT INTO t2 VALUES(1),(2);
Query OK, 2 rows affected (0.12 sec)
Records: 2  Duplicates: 0  Warnings: 0
mysql> INSERT INTO t3 VALUES(1),(2);
Query OK, 2 rows affected (0.12 sec)
```

```
Records: 2  Duplicates: 0  Warnings: 0
mysql> INSERT INTO t4 VALUES(1),(2);
Query OK, 2 rows affected (0.12 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

现在，假设需要在这些表上进行连接。下面的例子会给出需要的结果：

```
mysql> SELECT * FROM t1,t2 LEFT JOIN t3 ON (t3.f3=t1.f1)
 LEFT JOIN t4 ON (t4.f4=t1.f1) WHERE t2.f2=t4.f4;
+----+----+------+------+
| f1 | f2 | f3   | f4   |
+----+----+------+------+
|  1 |  1 |    1 |    1 |
|  2 |  2 |    2 |    2 |
+----+----+------+------+
2 rows in set (0.02 sec)
```

如果使用EXPLAIN来检查记录，就会注意到：

```
mysql> EXPLAIN SELECT * FROM t1,t2 LEFT JOIN t3 ON
 (t3.f3=t1.f1) LEFT JOIN t4 ON (t4.f4=t1.f1)
 WHERE t2.f2=t4.f4;
+-------+--------+---------------+---------+---------+------+------+--------------------------+
| table | type   | possible_keys | key     | key_len | ref  | rows | Extra                    |
+-------+--------+---------------+---------+---------+------+------+--------------------------+
| t1    | index  | NULL          | PRIMARY |       4 | NULL |    2 | Using index              |
| t2    | index  | PRIMARY,b,f2  | PRIMARY |       4 | NULL |    2 | Using index              |
| t3    | eq_ref | PRIMARY,c,f3  | PRIMARY |       4 | t1.f1|    1 | Using index              |
| t4    | eq_ref | PRIMARY,d,f4  | PRIMARY |       4 | t1.f1|    1 | where used; Using index  |
+-------+--------+---------------+---------+---------+------+------+--------------------------+
4 rows in set (0.00 sec)
```

有两个索引扫描，一个在t1上，另一个在t2上，即MySQL需要扫描全部的索引。仔细查看索引，会注意到LEFT JOIN需要t2在t4之前读。可以改变表的顺序和把t2从LEFT JOIN中分开来体会到这点：

```
mysql> EXPLAIN SELECT * FROM t2,t1 LEFT JOIN t3
 ON (t3.f3=t1.f1) LEFT JOIN t4 ON (t4.f4=t1.f1)
 WHERE t2.f2=t4.f4;
+-------+--------+---------------+---------+---------+------+------+-------------+
| table | type   | possible_keys | key     | key_len | ref  | rows | Extra       |
+-------+--------+---------------+---------+---------+------+------+-------------+
| t1    | index  | NULL          | PRIMARY |       4 | NULL |    2 | Using index |
| t3    | eq_ref | PRIMARY,c,f3  | PRIMARY |       4 | t1.f1|    1 | Using index |
| t4    | eq_ref | PRIMARY,d,f4  | PRIMARY |       4 | t1.f1|    1 | Using index |
| t2    | eq_ref | PRIMARY,b,f2  | PRIMARY |       4 | t4.f4|    1 | Using index |
+-------+--------+---------------+---------+---------+------+------+-------------+
4 rows in set (0.01 sec)
```

注意区别！根据行列数，只需要读2×1×1×1行（总数为2），与前面的查询的2×2×1×1（总数4）对应。当然，结果是一样的：

```
mysql> SELECT * FROM t2,t1 LEFT JOIN t3 ON (t3.f3=t1.f1)
 LEFT JOIN t4 ON (t4.f4=t1.f1) WHERE t2.f2=t4.f4;
+----+----+------+------+
| f2 | f1 | f3   | f4   |
+----+----+------+------+
|  1 |  1 |    1 |    1 |
|  2 |  2 |    2 |    2 |
+----+----+------+------+
2 rows in set (0.00 sec)
```

这再一次显示了使用EXPLAIN测试查询的重要性。没有对MySQL的内部工作的好的理解，永远也不会知道前面的查询哪个更快，尽管可能知道应该有差别。EXPLAIN量化了我们的估计，而这会随着经验的增加而增加。

用ANALYZE帮助优化MySQL的查询

MySQL中决定使用哪个键值的重要部分叫做查询优化器。它对索引进行快速查询以决定使用哪个索引。人在找书的时候也会做相似的事。比如，假设查找作者为Zakes Mda书名为“Ways”的一本书，而且只知道有两个索引。如果其中的一个是包含4000项的作者名的字母顺序，另一个是包含12000项的书名的字母顺序，读者可能会使用作者索引。但如果知道Zakes Mda写了200本书但只有一本叫“Ways”，就可能会使用另一个索引。如果MySQL知道索引包含的信息，它也会做得更好。可以通过运行如下的命令ANALYZE TABLE tablename来提供这种信息（叫基数，或者是惟一值）。在例表中执行这点没有太多可说的，但对于有许多插入、更新和删除的大表，经常分析表对性能有帮助。

如果键值分布不是最新的，ANALYZE TABLE就会更新它（执行ANALYZE 等同于执行myisamchk -a或myismachk --analyze. 更多信息见第12章“数据库复制”）。

警告：这只在MyISAM和BDB表中可用，并且在处理的过程中表被读锁住了。因此，不要在数据库忙时执行。

可以通过运行命令SHOW INDEX看到对MySQL可用的信息。

```
mysql> SHOW INDEX FROM customer;
+----------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table    | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+----------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| customer |          0 | PRIMARY  |            1 | id          | A         |           8 |     NULL | NULL   |      | BTREE      |         |               |
| customer |          1 | surname  |            1 | surname     | A         |           8 |     NULL | NULL   |      | BTREE      |         |               |
| customer |          1 | surname  |            2 | initial     | A         |           8 |     NULL | NULL   |      | BTREE      |         |               |
| customer |          1 | surname  |            3 | first_name  | A         |           8 |     NULL | NULL   |      | BTREE      |         |               |
+----------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
4 rows in set (0.01 sec)
```

表4.5解释了由Show Index语句返回的列的含义。

表4.5 SHOW INDEX返回的列的意义

列	描述
Table	正在查看的表名
Non_unique	0或1.0表示索引不能包含重复值（主键和惟一索引），1表示可以
Key_name	索引名
Seq_in_index	索引中的列的顺序，以1开始
Column_name	列名
Collation	A或NULL。A表示索引以升序排列，NULL表示不排序
Cardinality	索引中惟一值的个数。这通过运行ANALYZE TABLE或myisamchk -a来更新
Sub_part	如果整个列为索引的话，为NULL，否则为以字符表示的索引的大小
Packed	索引是否打包
Null	如果列能包含NULL值，则为YES
Comment	各种注释

删除和更新都会在表中留下缝隙（特别是在表包含TEXT、BLOB或VARCHAR域时）。这将导致额外的工作，因为在读取操作时需要跳过这些缝隙。

OPTIMIZE TABLE解决了这个问题，通过连接分割的记录来删除缝隙，与整理表数据是同样的作用。

提示： 关于OPTIMIZE语句的更多内容见第10章“基本管理”。注意在处理的过程中表是锁定的，因此不要在处理高峰时运行。

优化SELECT语句和安全

容许的权限（见第14章“数据库安全”）越复杂，执行查询的开支越大。但这不是少用安全的原因，如果有一个高容量的查询，还有一个低容量的复杂权限的查询，那么保留一个尽量分开的低容量集是有用的。

基准函数

BENCHMARK()函数返回MySQL执行一个函数若干次所花的时间。它能够给出两部机器性能之间的基本差别。语法如下：

```
SELECT BENCHMARK(number_of_repetitions,expression)
```

比较下面两个基准之间的差别，其中MySQL计算999的平方根1千万次的时间。第一个运行在一个运行Windows 98的不太常用的1GB Duron机器上，第二个运行在稍微有点忙的运行Linux Red Hat 7的Pentium III 850MHz机器上。为了比较，在运行FreeBSD 4.6的老点的Cyrix 200MMX的机器上执行了第三次：

```
mysql> SELECT BENCHMARK(10000000,SQRT(999));
+-------------------------------+
| BENCHMARK(10000000,SQRT(999)) |
+-------------------------------+
|                             0 |
+-------------------------------+
1 row in set (0.66 sec)
mysql> SELECT BENCHMARK(10000000,SQRT(999));
+-------------------------------+
| BENCHMARK(10000000,SQRT(999)) |
+-------------------------------+
|                             0 |
+-------------------------------+
1 row in set (2.73 sec)
mysql> SELECT BENCHMARK(10000000,SQRT(999));
+-------------------------------+
| BENCHMARK(10000000,SQRT(999)) |
+-------------------------------+
|                             0 |
+-------------------------------+
1 row in set (13.24 sec)
```

警告： 使用BENCHMARK()来比较机器时要小心。对活动的数据库，还有许多其他的因素，比如硬盘速度，在这种测试中就不会考虑。它的主要目的是优化函数。

优化更新、删除和插入

UPDATE基本上与SELECT相同，但其后执行了一个写操作。比如为了优化，下面的代码：

```
UPDATE fieldname FROM tablename WHERE condition
```

与这个代码一样：

```
SELECT fieldname FROM tablename WHERE condition
```

可以采用与SELECT语句等同的优化方法优化UPDATE语句。也要注意使用的索引越少，数据越小，操作就会越快。不要使用多余的索引或使域或索引大于需要的。

DELETE语句的速度依赖于索引的个数。删除时，每个记录都需要从关联的索引以及主数据文件中删除。这就是为什么TRUNCATE tablename比DELETE tablename要快，因为表是一次删除的，不需要删除单个的索引和数据记录。

插入数据的最好办法是使用LOAD DATA而不是INSERT，因为前者要快20倍。

通过在增加数据时禁用键值可以加速处理速度。MySQL此时只需考虑增加数据，而不需要担心同时要增加索引文件。这样，数据会增加得更快，而当索引被分开建立后，处理也

会更优化了。可以使用下面的过程：

```
ALTER TABLE tbl_name DISABLE KEYS;
LOAD DATA INFILE filename INTO TABLE tablename
ALTER TABLE tbl_name ENABLE KEYS;
```

但并不总是可以从文本文件中插入。如果能分组插入，同时插入多个记录会比单个语句要快。比如，下面的查询：

```
INSERT INTO tablename VALUES(record1),(record2) (recordn);
```

就比下面的替代语句要快：

```
INSERT INTO tablename VALUES(record1);
INSERT INTO tablename VALUES(record1);
...
INSERT INTO tablename VALUES(recordn);
```

原因是每个INSERT语句都要进行索引刷新。如果需要多个INSERT语句，可以使用锁来达到同样的目的。对于非事务表，使用这样的语句：

```
LOCK TABLES tablename WRITE;
INSERT INTO tablename VALUES (record1),(record2),(record3);
INSERT INTO tablename VALUES (record4),(record5),(record6);
UNLOCK TABLES;
```

注意上面的例子在运行中时，任何人都不能读表。

对事务表想达到同样的目的，可使用下面的语句：

```
BEGIN;
INSERT INTO tablename VALUES (record1),(record2),(record3);
INSERT INTO tablename VALUES (record4),(record5),(record6);
COMMIT;
```

当从多个线程中增加记录时，事情变得更复杂一些。考虑下面的情节，第一个线程增加10 000个记录，第二个线程增加1个记录。如果使用锁的话，整个的速度可能快很多，但第二个线程只在第一个线程之后结束。没有锁的话，第二个线程结束得快的多，但整体的速度会慢一点。第二个线程相对于第一个线程的重要性将决定采用的方法。

但是，应用程序可能需要连续做很多不相关的插入。如果不使用行级锁（对InnoDB表是可能的），就可能出现因为几个人执行插入，大部分查询表的人等待超长的时间。但不要失望——有办法减小这个影响。

首先是使用INSERT LOW PRIORITY。这会导致插入操作失去强制性，并等到队列中不再有读查询为止。但有个问题，就是如果数据库非常忙，进行INSERT LOW PRIORITY的客户可能会用很长的时间才能找到间隙。

替代的是INSERT DELAYED。客户立即被释放，插入被放入队列（其他的INSERT DELAYED语句还是在等待队列结束）。这样做的缺点就是没有有意义的信息返回给客户（比如auto_increment值），因为在客户被释放时，INSERT还没有被处理。但有些事情不值得等待！注意，如果在这种情况下发生了灾难（比如掉电），队列中的插入可能会全部丢失。

警告： 对于INSERT LOW PRIORITY或INSERT DELAYED，在数据插入时，我们都没有办法，因此要小心使用。

小结

错误地使用索引可能是单一的最重要的导致效率出现问题的原因。索引是一个指向主要数据文件的小的、排序的文件。因为只需要检索小的索引文件，因此查找特定的记录会更快。

索引可以是一个主键（不能包含null值的惟一索引）、惟一索引、普通索引（可以包含重复值），或者是全文索引。全文索引容许在文本域中针对某种关键字的组合进行一个高级别的复杂查询。

自动增加域与主键关联，它可以让MySQL自动考虑域的顺序。如果插入了一个记录，MySQL会在前面的自动增加值上加1并把它作为自动增加域的值。

EXPLAIN返回关于MySQL在某个特定的查询中使用索引的有用信息。可以使用它来查看MySQL是否使用了创建的索引，如果查询不是最优的，可以得到对哪个域创建索引以及如何改变查询使其优化的信息。

第5章 MySQL编程

本章不准备教授读者如何编程。很多书都想同时教授MySQL和编程语言，但最后两件事都没有完成好。本书假设读者已经是一个出色的程序员，只是想学习或者是只想成为数据库管理员（DBA）的角色，而对编程不感兴趣。

当数据库成为了大型信息系统的一部分并且功能齐全的应用也向系统中加入自己的值后，读者就会感到数据库的真正威力。比如，新的网站就需要工具来添加和排序新闻文章并显示在网站上，以及跟踪最流行的故事。但绝大部分记者都对学习结构化查询语言（SQL）不感兴趣，因此他们需要一个设计良好的界面连接到数据库上去。这个界面可以是一个带超文本标记语言（HTML）格式的网页，并带一个提交键来调用脚本执行INSERT语句。或者界面也可以是从报纸的QuarkXPress系统得到并自动加入到数据库中的文章的新闻反馈。

另一个关联的应用是针对财务顾问的信息系统，系统的信息是最近的股价和汇价，这样，财务顾问通过存取和分析信息来为客户监控股价和汇价的变化趋势。

信息系统的需求是无限的。这些方案都有一个共同点：都包括一个应用和MySQL不能提供的额外的逻辑。在理论上，可以使用任意的编程语言来开发应用。经常使用的语言有Java、C、PHP、Perl、C++、Visual Basic、Python和Tcl，它们大部分与MySQL都有开发得很好的应用编程接口（API）。本书的附录包含有大部分语言的API。

本章的全部例子使用的都是PHP——不是因为都应该使用PHP，而是因为大部分人都已经这样做了，因为有C相似背景（C、Perl或C++）的人对它的语法都很熟悉，而且对于其他的程序员它学习起来也很容易。不是语法而是编程的原则更重要。本章的全部代码都有详细注释以便读者无论熟悉哪种语言或水平如何都能跟着学。

本章的主要内容：

- 使用永久连接
- 使代码容易移植和维护
- 估算数据库与应用的负载
- 探讨应用的开发过程

使用好的数据库编程技术

下面的小节将介绍一些最普通的技术，数据库程序员使用它们来保证应用健壮（不容易瘫痪）、可移植（容易移植到新的环境或平台上），以及易于维护。当应用生成大量的连接请求并且大部分是在短时间内从同一个源而来时，永久连接是非常有用的技术。没有经验的、着急的或懒惰的程序员在没有考虑移植性和维护，或者是给应用带来更大的负载而不是把负载给数据库的情况下创建的代码很容易使后来的程序员（也经常是他们自己）感觉更困难。通过在开始时做一些简单的计划，就可以避免后面不必要的问题。

使用永久连接

MySQL与其他的数据库相比，在连接时已经相当快了。但是，连接数据库还是一个相当重的任务，并且，如果要在相当短的时间内（比如从Web服务器连接）进行大量的连接，就要在自己的资源上使其尽可能的容易。使用永久连接就会在脚本完成以后还能保持连接打开。当下一个连接请求来的时候，就会重新使用已存在的连接，节约负载。在PHP中，可以使用mysql_pconnect()函数来创建永久连接：

```
mysql_pconnect($host,$user,$pass);
  // the mysql_pconnect function creates a persistent connection to a
  // mysql database, taking host, user and password parameters
```

但并不需要在很长的时间中都让连接悬挂着。大多数情况下，Web服务器都会在之后清除连接。但是，作者也遇到过Web服务器有问题，导致不清除连接而重启自己的情况。Web服务器被设置为容许400个实例，而MySQL可以承受750个连接。在Web服务器出错并不能清除的情况下，对数据库的连接数会加倍，可能到800。数据库服务器会突然用完可用的连接数。可以通过减少wait_timeout mysqld变量（或interactive_timeout，取决于如何连接的）——它决定了MySQL中一个连接在关闭之前容许保持非活动的时间——来减少永久连接悬挂时间太长的危险（第10章“基本管理”和第13章“配置并优化MySQL”，解释了如何设置这些变量）。默认值是28 800秒（8小时）。通过把这个值减少到600秒，就防止了问题再出现。之后，就只需要关心Web服务器了。

使代码易于移植和维护

一些简单的步骤能够极大地提高代码的灵活性。这包括把连接的详细信息分开保留并放在单一的位置上，同时灵活地创建数据库查询以便在将来改变数据库结构时不影响到应用。

连接

通过本地函数，大部分编程语言在连接数据库时都非常容易。比如，PHP就有许多函数可以与MySQL一起使用，比如mysql_connect()、mysql_query()，等等。

当编写只有一个数据库连接的极小的应用时，可以使用本地类，它可以很简单地连接到MySQL（见清单5.1）。

清单5.1 totally_importable.php

```
$db = mysql_pconnect("dbhostname.co.za", "db_app", "g00r002b");
  // the mysql_pconnect function creates a persistent connection to a
  // mysql database, taking host, user and password parameters
  // where 'dbhostname' is the host, 'db_app' the user and
  // 'g00r002b' the password
if (!$db) {
    echo "There was a problem connecting to the database.";
    exit;
}
```

```
// basic error checking - if the connection is not successful, print
// an error message and exit the script
```

读者将要碰到的许多例子都使用了这种方式来连接，因为这很容易理解并且在小的条件下很好。但在更重要的使用数据库的情况下，应该使它尽可能的易于移植，容易维护和保证安全。

假设有10个脚本连接到数据库。如果全部的10个脚本都用同一种方式连接，读者有一天想把数据库移到新服务器上或者想改变密码，此时就不得不改变10个脚本。

现在假设有100个脚本。

作者在以前碰到过这种情况，面对着存在密码威胁安全的可能（如果密码在上百个地方，就更容易被发现），作者不得不做许多繁杂的工作！最好的解决方法是从开始就正确地构建应用。把数据库连接的详细信息分开保存，并把它们包括在连接数据库的脚本中。然后，在需要改变细节的时候，只需要在一个地方改变（并且知道不会有任何东西被忘记）即可。改变位于上百个地方的密码可能会有遗忘一个地方的危险，而且只有在功能失败的时候才能发现。

显示在清单5.2和清单5.3的方案更好一些。

清单5.2 db.inc

```
$host = "dbhostname.co.za";
$user = "db_app";
$pass = "g00r002b";
```

清单5.3 not_too_portable.php

```
require_once "$include_path/db.inc";
// includes the file containing the connection details, db.inc
// located in the path: $include_path, which should be a safe location
$db = mysql_pconnect($host, $user, $pass);
  // the mysql_pconnect function creates a persistent connection to a
  // mysql database, taking host, user and password parameters
if (!$db) {
    echo "There was a problem connecting to the database.";
    exit;
}
  // basic error checking - if the connection is not successful, print
  // an error message and exit the script
```

在这个例子中，密码、主机名和用户名都存储在一个文件中，因此改变细节只需在一个地方（db.inc）进行。

警告： 如果创建Web应用，确保db.inc不在Web树内（Web服务器一般不能使用.inc文件，但无论如何，应该把敏感信息被得越远越好）。

进一步的改进来自于高一点的抽象级别中。假设在清单5.2和清单5.3中，管理人员决定移到另一个DBMS中。碰巧，MySQL不是对每一种情况都是理想的，除此之外，管理人员经常做一些奇怪的决定——比如作者遇到的另一种情况，管理人员花费了上万的资金在另一种

数据库上，而全部实际需要的就是适当地配置MySQL（幸运的是作者设法说服了他们，代码又一次不太好移植！）。

为了使代码易于移植，我们只想在一个地方改变DBMS的细节。这包括创建处理连接细节的第二个函数，这显示在清单5.4和清单5.5中。db_pconnect()函数放在文件db.inc中，在portable.php中，这个函数代替了mysql_pconnect()用来连接数据库。

清单5.4 db.inc

```
// this function connects to the database, and returns the connection
function db_pconnect() {
  $host = "dbhostname.co.za";
  $user = "db_app";
  $pass = "g00r002b";
  return mysql_pconnect($host, $user, $pass);
}
```

清单5.5 portable.php

```
require_once "$include_path/db.inc";
 // includes the file containing the connection details, db.inc
 // located in the path: $include_path, which should be a safe location
$db = db_pconnect($host, $user, $pass);
if (!$db) {
  echo "There was a problem connecting to the database.";
  exit;
}
 // basic error checking - if the connection is not successful, print
 // an error message and exit the script
```

现在，如果想从MySQL换到另一个数据库，只需要用一个适当的函数比如odbc_pconnect()替换mysql_pconnect()。

说明： 本书不想影响读者的编程风格。每种语言都有自己的强项和弱项。比如，Java与PHP比，更多的是一个面向对象的语言，因此把前面的例子直接翻译为Java，并不能工作。最重要的原则是使应用尽可能地容易维护（把连接信息存储在一个地方）和移植（避免针对特定数据库的代码）。

数据库查询

在直接查询MySQL时，使用诸如SELECT *那样的快捷键是可以接受的。但在应用中应该避免这个，因为这会使代码更难移植。考虑在入门的数据库表中有三个域的情况，按顺序，它们是id、first_name和surname。编程代码可能有点像清单5.6。

清单5.6 totally_inflexible_select.php

```
 // assuming the connection $db has already been made
$result = mysql_query('SELECT * FROM entrants',$db);
  // run the query on the active connection
```

```
while ($row = mysql_fetch_array($result,MYSQL_NUM)) {
    // mysql_fetch_array when called with MYSQL_NUM
    // as a parameter, returns a numerically indexed
    // array, each element corresponding to a
    // field returned from a returned row
  $id = $row[0];
    // Because the id is the first field in the database,
    // it is returned as the first element of the array,
    // which of course starts at 0
  $first_name = $row[1];
  $surname = $row[2];
  // .. do some processing with the details
}
```

这开始还能工作。但现在，某人（总是某人）对数据库结构做了变动，在first_name和surname之间添加了一个叫initial的新域。读者自己的代码不需要initial。突然代码不能工作了，因为initial是数组的第三个元素（$row[2]）并存储为$surname。实际的surname现在在$row[3]中，却永远不会被存取。

在totally_inflexible_select.php脚本中需要注意的问题有很多。除了在数据库结构变化后它不能工作以外，用来返回域的函数返回的都是数字数组，而不是组合数组。这样，代码会很难读，不了解数据库结构的人不知道从数据库中返回的是什么。在PHP中，可以通过使用函数返回组合数组来纠正这一点，如清单5.7所示。

清单5.7 inflexible_select.php

```
 // assuming the connection $db has already been made
$result = mysql_query("SELECT * FROM entrants",$db);
while ($row = mysql_fetch_array($result,MYSQL_ASSOC)) {
    // mysql_fetch_array when called with MYSQL_ASSOC
    // as a parameter, returns an associative array,
    // with each key of the array being the name of a
    // field returned from the row
  $id = $row["id"];
  $first_name = $row["first_name"];
  $surname = $row["surname"];
  // .. do some processing with the details
}
```

这要好一点，因为现在即使在数据库表中加了一个initial后，代码也能工作。它能够处理一些数据库结构的变化，并且它更易读。对数据库结构没有任何了解的程序员也能够知道返回的域是什么。但还可以做进一步改进，通过执行SELECT *查询，可以要求MySQL从表中返回全部域。读者自己的代码只需要使用三个域，因此为什么要在返回全部域的时候浪费资源，并且使用额外的磁盘输入/输出和造成网络更大的压力？只需要指定要返回的域。这不仅减少资源浪费，而且能使代码更易读。实际上，在某些情况下，返回组合数组比返回数字数组更耗资源。这种情况下，通过指定如清单5.8所示的域，也可以在返回数字数组的

时候保持代码的可读性。

清单5.8 flexible_select.php

```
// assuming the connection $db has already been made
$result = mysql_query("SELECT id,first_name,surname FROM entrants",$db);
while ($row = mysql_fetch_array($result,MYSQL_NUM)) {
    // mysql_fetch_array when called with MYSQL_NUM
    // as a parameter, returns a numerically indexed
    // array, each element corresponding to a
    // field returned from a returned row
  $id = $row[0];
  $first_name = $row[1];
  $surname = $row[2];
  // .. do some processing with the details
}
```

同样的原则也可以应用在INSERT查询上。在应用中，不要使用不带域列表的INSERT查询。比如，考虑带三个域——id、first_name和surname的原始的表，可以使用清单5.9的代码。

清单5.9 inflexible_insert.php

```
// assuming the connection $db has already been made
$result = mysql_query("INSERT INTO entrants?
VALUES('$id','$first_name','$surname')",$db);
```

如果表结构改变了，代码又会失败。添加一个域initial，插入的域的个数与表中的域的个数不同，查询将失败。

解决这个问题的方法是在插入中指定数据库的域，如清单5.10所示。

清单5.10 flexible_insert.php

```
// assuming the connection $db has already been made
$result = mysql_query("INSERT INTO entrants(id, first_name,?
surname) VALUES('$id','$first_name','$surname')",$db);
```

这同样能够增加可读性，特别是考虑到域值不总是与域名匹配的情况下，如同上面的例子一样。

数据库服务器应该做多少工作

在架构师之间长期争论的事之一就是数据库服务器应该做多少工作和应用应该做多少工作。

MySQL开发人员开始也强烈支持把负担给应用，一部分理由是不支持诸如存储过程和触发器之类的功能，而另一部分是因为原则。因为这个他们遭到批评，缺乏这些功能使大家拒绝接受MySQL，忽略了MySQL是一个正规的数据库，而现在，从版本4开始，MySQL就要开始摆脱这一点了。

一般来说，数据库应该做尽量多的事。考虑下面产生同样输出但用不同的方式实现的例子。清单5.11返回没有排序的全部数据，然后使用PHP的sort()函数来排序。而清单5.12使用了MySQL的ORDER BY从句来排序数据。

清单5.11 work_the_script.php

```
// assuming the connection $db has already been made
$result = mysql_query("SELECT surname FROM entrants",$db);
while ($row = mysql_fetch_array($result,MYSQL_ASSOC)) {
    // mysql_fetch_array when called with MYSQL_ASSOC
    // as a parameter, returns an associative array,
    // with each key of the array being the name of a
    // field returned from the row
  $surname[] = $row["surname"];
    // add the surname as the next element of the
    // surname array (and creates the array if not yet done)
}
  sort($surname)
  // the sort() function sorts the array
  // ...continue processing the sorted data
```

清单5.12 work_the_db.php

```
// assuming the connection $db has already been made
$result = mysql_query("SELECT surname FROM entrants ORDER BY surname",$db);
while ($row = mysql_fetch_array($result,MYSQL_ASSOC)) {
    // mysql_fetch_array when called with MYSQL_ASSOC
    // as a parameter, returns an associative array,
    // with each key of the array being the name of a
    // field returned from the row
  $surname[] = $row["surname"];
    // add the surname as the next element of the
    // surname array (and creates the array if not yet done)
}
  // ...continue processing the sorted data
```

清单5.12更好一些。如果这是一个普遍进行的操作，MySQL可以（应该）在surname域建一个索引，从一个索引中读取有顺序的数据比从读取未排序的数据并使用应用来排序更快一些。

实际上，从数据库中读取排序的数据可能要比读取未排序的数据要快（即使考虑sort()函数），因为排序数据可能只需要从索引中而不是数据文件中读取。

也可能有例外（可能的情况是索引不存在而数据库服务器是主要的瓶颈），但大多数情况下，显示在清单5.12中的技术更好一些。

一个相似的、更极端的（但还是很普通）例子是应用完成WHERE从句的工作，如清单5.13所示。

清单5.13 work_the_script2.php

```
// assuming the connection $db has already been made
$result = mysql_query("SELECT surname FROM entrants",$db);
while ($row = mysql_fetch_array($result,MYSQL_ASSOC)) {
    // mysql_fetch_array when called with MYSQL_ASSOC
    // as a parameter, returns an associative array,
    // with each key of the array being the name of a
    // field returned from the row
  if ($row["surname"] == 'Johnson') {
    $johnson[] = $row["surname"];
      // add the surname as the next element of the
      // johnson array (and creates the array if not yet done)
  }
  elseif ($row["surname"] == 'Makeba') {
    $makeba[] = $row["surname"];
    // add the surname as the next element of the
    // makeba array (and creates the array if not yet done)

  }
}
  // ...process the makeba and johnson arrays
```

更好的解决办法是使用WHERE从句，如同清单5.14，并且不要浪费时间来获得不需要的额外数据。

清单5.14 work_the_db2.php

```
// assuming the connection $db has already been made
$result = mysql_query("SELECT surname FROM?
 entrants WHERE surname = 'Makeba' OR surname='Johnson'",$db);
while ($row = mysql_fetch_array($result,MYSQL_ASSOC)) {
    // mysql_fetch_array when called with MYSQL_ASSOC
    // as a parameter, returns an associative array,
    // with each key of the array being the name of a
    // field returned from the row
  if ($row["surname"] == 'Johnson') {
    $johnson[] = $row["surname"];
      // add the surname as the next element of the
      // johnson array (and creates the array if not yet done)
  }
  elseif ($row["surname"] == 'Makeba') {
    $makeba[] = $row["surname"];
      // add the surname as the next element of the
      // makeba array (and creates the array if not yet done)
  }
}
  // ...continue processing the sorted data
```

如果正在处理更多的名字，可以把这些代码片段写得更好，但关键是清单5.14效率更高，因为MySQL做了这个工作，限制了返回的结果，减少了使用的资源。

清单5.15演示的是由具有其他数据库背景的人提供的解决办法。因为MySQL版本4.0没有完全实现子选择（尽管这会在版本4.1中改正），人们总是以为除了使用应用以外没有其他的方法。比如，有两个customer表并且想知道哪个客户在一张表中出现了但没有在另一张表中出现的情况，下面的标准的ANSI查询在MySQL中就不起作用：

```
SELECT first_name,surname FROM entrants WHERE code NOT IN
 (SELECT code FROM referred_entrants);
```

因为这个原因，显示在清单5.15中的代码也经常被使用。

清单5.15 work_the_script3.php

```
    // assuming the connection $db has already been made
$result = mysql_query("SELECT code FROM entrants",$db);
$codelist = "";
  // initialise the list of codes
while ($row = mysql_fetch_array($result,MYSQL_ASSOC)) {
    // mysql_fetch_array when called with MYSQL_ASSOC
    // as a parameter, returns an associative array,
    // with each key of the array being the name of a
    // field returned from the row
  $codelist .= $row["code"].",";
    // add the code, followed by a comma to the $codelist variable
}
$codelist = substr($codelist, 0, -1);
  // removes the last comma, resulting in a list
  // such as "1,3,4,8,12";
$result = mysql_query("SELECT first_name,surname FROM?
 referred_entrants WHERE code NOT IN($codelist)",$db);
while ($row = mysql_fetch_array($result,MYSQL_ASSOC)) {
    // ...process the entrant details
}
```

清单5.15也可以工作，但它给应用留了太多的工作。通过思考，MySQL可以通过查询来返回结果，如清单5.16所示。

清单5.16 work_the_db3.php

```
    // assuming the connection $db has already been made
$result = mysql_query("SELECT entrants.first_name,?
  entrants.surname FROM entrants LEFT JOIN referred_entrants?
  ON entrants.code = referred_entrants.code WHERE?
  referred_entrants.code IS NULL",$db);
while ($row = mysql_fetch_array($result,MYSQL_ASSOC)) {
    // ...process the entrant details
}
```

当子选择被实现后（根据当前的计划，应该是版本4.1），显示在清单5.17中的代码更简单。

清单5.17 work_the_db3_2.php

```
        // assuming the connection $db has already been made
    $result = mysql_query("SELECT first_name,surname FROM
     entrants WHERE code NOT IN (SELECT code FROM
     referred_entrants", $db);
    while ($row = mysql_fetch_array($result,MYSQL_ASSOC)) {
        // ...process the entrant details
    }
```

应用开发的阶段

本章假设读者已经能够编程或者能够用另一本书学习。但很多新程序员，特别是那些没有经过很好培训的人，总会忘记在以后回去再查看一下程序开发过程并计划一下以后的情况。很多程序员还在重复别人的工作，不能充分利用自己的时间，从而责备项目经理、用户以及周围的人。当然，其他的人也可能需要责备，但本书的目标是MySQL开发人员，作者针对开发人员涉及的项目提出了一些建议。特别是Web开发人员，他们通常通过学习HTML和JavaScript才接触到MySQL和编程语言，因此不太知道大型项目的复杂性，当项目变大时，就不知所措了。

下面的小节将简单地讨论应用开发的阶段。它不是严格的规定；而只是可能的框架的一种。任何一种应用开发都会根据可用的资源以及项目的特殊情况容许一定的灵活性。尽管各种好的开发方法的原理是一样的，但还是应该选择一种适合自己需要的方法。与第9章“数据库设计”连接起来阅读本章，将更好地掌握这个原则。

阶段1：需求分析

项目的需求分析显然是开发应用的一个步骤。当面对糟糕的软件开发现状时，专家们一遍又一遍地重复着这个原则。很多情况下，“我们不知道你需要哪个”或“在开始的时候你没有告诉我们”经常成为项目糟糕状况的借口。项目的第一和可能是最重要的阶段就是决定需求。

确定用户需求

用户的大部分需求都是琐碎的，经常使他们自己都吃惊。作者曾经遇到一个用户，与一个能力稍差或者说是懒惰的开发人员讨论后，被吓得做出了一个不会比在表中加一个额外的域更复杂的需求。在大多数情况下，只要需求适当，任何事情都是可能的。主要的困难不是完成用户的需求，而是在框架已经建好后完成不同的需求。用户通常不知道自己需要什么。需要帮助他们形成需求。但是，本书是针对开发人员的，而不是业务决定者，因此作者把这个负担都放在了开发人员身上。应该确保用户的需求是清楚的，不仅开发队伍应该理解，而且用户自己也应理解。

项目团队在决定用户需求时，需要考虑下面几点：

- 让用户决定他们的需求。指导用户，解释为什么某个建议是不实际的或提供一个更好的替代。团队需要利用经验，把没有说出来的需求公开并形成文档。对用户显而易见的问题可能对开发人员并不明显，这样重要的需求就被丢掉了。比如，用户可能需要一个预定系统处理预定。但，这不够好。对预定来说，每个域都需要，后台的处理过程应该明细化。
- 一旦理解了用户需求，写下来并把它们返回给用户和项目拥有者（给项目付钱的人）以得到确认。用户应该确定他们能够得到的东西，而在以后不会感到奇怪。
- 让拥有者在需求上签字。这样，就不会产生使双方不愉快的可能。范围蔓延——在开发的过程中需求继续增加——是一个经常发生的危险问题，通常出现在需求没有达成正式一致的时候。或者项目拥有者需要的内容越来越多，或者是某一方的人发现了一个从来没有人想到的致命的新问题。

确定技术需求

确定技术需求与确定用户需求一样重要。在一个服务器上，一个月2千万的页面请求对一个增强的数据库网站是可能的，但这至少需要一台好的机器和特定的条件。团队不应该在项目需求上增加任何影响，比如，“应该运行于Linux和MySQL之上”。后面列举技术需求的原因是让技术更好地适应项目——如果有任何限制。需要回答的问题包括在如下：

- 多少机器？需要那种结构来连接它们？
- 需要的是哪种机器？
- 需要哪种操作系统、DBMS或其他的软件（比如Web服务器、邮件客户端，等等）？
- 开发应用的语言是哪种？它们是面向对象的吗？

阶段2：设计应用

一旦明确了需求，就应该开始设计应用了。

模型

模型简化了程序结构并把用户需求转换为了程序员能够很容易理解的形式。这些可以是正式的模型，比如统一建模语言（UML）需要的，一个数据流图，或者是简单地在纸上画的图。包括的基本内容是每个处理需要的数据和处理产生的数据。

使用伪码

伪码是能够帮助程序员更好和更容易地开发应用的另一个步骤。伪码可以处理逻辑需求，创建解决问题需要的算法而不需担心编程语言的准确语法。

阶段3：编码

这经常变成了惟一的一步。但如果在前面的步骤产生的文档存在的话，编码会变得非常容易。在编码阶段可以使用下面的小技巧：

- 代码要文档化。代码中要包括注释，并创建独立的文档以注明应用是如何组织在一起的以及是如何工作的。如果不为自己，也应该为自己以后的人做（在几个月以后，自己会吃惊地发现忘记一些“琐碎”细节是多么容易）。如果编码员没有为后面的人留下足够的文档，那他就是自私的人。确保及时地做这些事，不要在截止日期的压力下忽略这个方面的事。
- 使用清楚的文件名、函数名和变量名。比如，函数f3()不太直接，而calculate_interest()却很清楚。
- 不要重复别人的工作。已经有许多类和实例代码可用。只有很少的情况，需要编码员做一些独一无二的事情。大部分情况下工作都是重复的，编码员所做的就是一个图书管理员的工作，即为工作找到适当的代码。
- 即使在变量没有被初始化也能使用的语言中，也要在一个地方初始化和文档化变量。这不仅可读性好，而且更安全。
- 把代码分解成小的部分。这会使它更容易理解、调试和维护。比如，在Web开发中，一些源支持一个脚本处理全部事情这样的优点，但实际结果却通常是陷入混乱。
- 聪明地使用路径。应用的全部不能也不应该出现在同一个路径中。为了安全起见，逻辑化地分组它们。
- 重用自己的代码。已经创建的函数和类可以一遍一遍地使用。把它们写成能够针对稍微不同的任务。这会使稍后阶段改变的代码容易一些。这特别适用于数据库连接之类的事上。
- 把逻辑层和表示层分开（在Web环境中HTML和脚本语言经常混在一起）。
- 调试查询时，能够在MySQL中直接运行查询而不必通过应用（作者经常显示查询然后把查询粘贴到MySQL客户端）是很有帮助的。这能够缩小错误范围，以便确定是正在运行的查询的错误还是应用的错误。
- 即使工作在不需要关闭连接（在PJP中执行mysql_close()）和清除资源的情况下，也要养成这种关闭连接的习惯。Web开发人员在开发其他应用时，经常陷入混乱，因为他们习惯了在Web服务器过程完成后自动清除全部的Web环境。同样，PHP那样的语言也没有C那样的语言严格。
- 简单的代码就是好代码。写出一大段不可读的代码行，似乎使自己看起来聪明，但这却妨碍了后面想读代码的人。编写过于复杂的代码的代码员没有做到建议给代码员的事：把复杂简单化。简单的代码一般也快。它甚至能更快，比如用字符串函数来代替一般表达式。
- 在编码员超过一个的时候，使用编码标准。这样当编码团队的成员使用他人的代码时，就可因为在适应新的标准（或缺乏标准）上花的时间少而效率高一些。编码标准包括诸如在使用对齐和变量名习惯（比如是否使用大写，比如$totalConnects; 下画线，比如$total_connects; 或者是不要空格，比如$totalconnects）时空格（或tab键）的个数。标准甚至包括约定编辑器的使用，因为不同的编辑器对齐代码是不同的，这会使代码更难读。
- 大的项目还需要某种版本控制。这能够保证在多人操作同一部分代码时，工作不相互冲突。如果习惯在不同的机器上工作并在各个地方保存不同的版本，一个人的项

目也很容易失控。大项目可以使用诸如CVS或Visual SourceSafe（可以分别在http://www.cvshome.org/和http://msdn.microsoft.com/ssafe/找到）的东西，而小项目可以简单地使用编号方式。

- 项目许可的话（特别是在项目需求存在疑问的情况下）使用原型。原型就是最终开发的系统的模型，它没有全部的功能，但一直到最终的系统完成它都还在工作。原型需要更多的用户角色，也能使他们感到更多的参与度。

阶段4：测试和实现

永远不要忽略测试阶段！比如，截止日期的压力可能使你把三个星期的测试计划减少为1个星期，但这肯定是不对的。不应该把测试当做不需要的负担；它应该是一个仔细调节应用和保证应用正常运行的重要部分。

有许多种测试方式：

单元测试 这种测试保证每个类、方法和函数自己都工作正常。应该测试在各种可能的方式下，不管输入如何，结果都正确。

集成测试 这种测试保证当与其他单元组合时，每个单元还能正常工作。

系统测试 整个系统作为一个整体来测试。这包括在负载下测试系统的效率（压力测试）、多用户使用，等等。

回归测试 用来测试错误修改是否破坏了其他的功能。这在“快速修改”以应对无法预见的情况时非常普遍。

在完成需求分析、设计应用、编码和测试后，就要发布应用了。读者可能想先把应用提供给小组的用户或单个的办公室使用，以便处理一些不可预见的问题，或者在某一时间让大家进入新系统。然后，准备在用户提出新的需求时重新开始这个过程。

小结

有许多技术可以使数据库应用易于移植和维护，特别是数据查询。从脚本中把连接细节分离出去，并确保细节保存在一个地方以方便读者改变细节。确保在对数据库表结构进行改变时，如果与脚本无关，就不应影响到脚本。在SELECT和INSERT查询时指定域名。

在某些情况（特别是Web应用）下使用永久连接有好处。使用它能够减少从同一个地方来的频繁的连接而导致的负载。

MySQL在执行一个任务时，一般比编程语言效率要高。MySQL针对速度进行了优化，能够使用索引、缓存和内存快速地存取信息。比如，尽管能够使用应用语言代替MySQL来排序数据，但效率不高。一般说来，要尽量使用MySQL。

开发应用时，仔细计划是非常重要的。在决定技术需求、设计模型和编码应用之前，先收集和规范化从用户来的需求。

编码时，确保避免通常的缺陷以使代码尽量灵活、移植性好和容易维护。在实现应用之前，应该准备好全面地测试它，这与编码是一样重要的。

第6章 扩展MySQL

MySQL的一个最大优势就是能够相对容易地扩展。当前面的MySQL版本被指责为缺乏特点时，得到的反馈总是“自己写”。MySQL的这一点对于每个能够使用C或C++的人来说都很容易。

可以用两种方式为MySQL增加函数。创建一个用户定义的函数（UDF）或者是增加一个本地（内置）函数。可以在源码或者是二进制发布中增加UDF，并能在任何时候增加和删除它们，但只能在源码中增加一个内置函数（改变源码并编译增加的部分）。本章只讨论UDF（本章使用术语UDF来描述与单个MySQL函数相对应的C/C++函数的关联集。术语“函数”描述了一个单个的C/C++函数）。

本章的主要内容：

- 标准用户定义函数
- 集合用户定义函数
- UD函数和参数

用户定义函数

读者在C或C++中会经常写UDF。可以在二进制发布和已经与--with-mysqld-ldflags=-rdynamic配置的源码中使用它们。一旦添加以后，当服务器重启后，除非使用了--skip-grant-tables选项，UDF都会一直可用。

有两种类型的UDF：标准的和集合的。标准的UDF就像普通的内置函数，比如POW()和SIN()，并作用于单个行的数据上，而集合函数就像内置的作用于组的SUM()和AVG()函数。

为了实现UDF，需要做如下工作：

1. 写C或C++函数（只要能把它们编译为本地代码共享库的形式即可，也可以使用其他的语言）。
2. 编译和安装库。
3. 调UDF到MySQL中。

UDF被增加后，细节就存储在mysql数据库的func表中。func表看起来如下所示：

```
mysql> SHOW COLUMNS FROM func;
+-------+-----------------------------+------+-----+----------+-------+
| Field | Type                        | Null | Key | Default  | Extra |
+-------+-----------------------------+------+-----+----------+-------+
| name  | char(64) binary             |      | PRI |          |       |
| ret   | tinyint(1)                  |      |     | 0        |       |
| dl    | char(128)                   |      |     |          |       |
| type  | enum('function','aggregate') |      |     | function |       |
+-------+-----------------------------+------+-----+----------+-------+
4 rows in set (0.00 sec)
```

name域包含有UDF的名（以及主要的C/C++函数名）。ret域指明UDF是否能返回null，而dl域指明包含UDF（很多UDF可以放在同一个库中）库的名，type域指明UDF是标准形式的还是集合形式的。

说明： 如果是从旧版本的MySQL升级而来，可能不会包含有上面的全部列。执行mysql_fix_privilege_tables脚本（在bin路径中）在表中生成缺少的列。

在本节中，读者先学习编译和安装发布的MySQL带来的UDF例子，然后再写自己的UDF。在开始之前，先创建一张小表并添加记录，增加之后，此表就会被使用来测试UDF。

```
mysql> USE firstdb;
Database changed
mysql> CREATE TABLE words (id tinyint(4), word varchar(50));
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO words VALUES (1,'aeiou');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO words VALUES (2,'bro');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO words VALUES (3,'so');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO words VALUES (4,'kisso');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO words VALUES (5,'lassoo');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO words VALUES (2,'bro');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO words VALUES (3,'so');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO words VALUES (4,'kisso');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO words VALUES (4,'kisso');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO words VALUES (5,'lassoo');
Query OK, 1 row affected (0.00 sec)
```

MySQL带有五个UDF的例子，一起放在udf_example.cc文件中，一般存储在源码的mysql/sql路径中。需要把它编译为可共享的目标文件。在UNIX系统中，这些文件的扩展名一般为.so，在Windows中一般为.dll。使用的命令（在UNIX中）有点像下面的语句：

```
% gcc -shared -o udf_example.so udf_example.cc
```

本书不是编程和编译的指南，因此尽管都粗略地解释了原因，但并不能覆盖全部的组合。读者可能需要一些经验或者是为了对本章的大部分内容有更多了解而需要得到一些帮助。

为了得到自己系统的正确的编译命令，可以使用检查依赖性的make工具。每个系统可能有所不同，但运行make后都会得到下面类似的输出：

```
% make udf_example.o
g++ -DMYSQL_SERVER -DDEFAULT_MYSQL_HOME="\"/usr/local\""
-DDATADIR="\"/usr/local/var\"" -DSHAREDIR="\"/usr/local/share/mysql\""
-DHAVE_CONFIG_H -I../innobase/include -I./../include -I./../regex
-I. -I../include -I. -O3 -DDBUG_OFF   -fno-implicit-templates
-fno-exceptions -fno-rtti -c udf_example.cc
```

使用提供的选项来编译UDF。系统还是不一样的——一些需要-c选项，而另一些却不需要。自己应该了解自己的系统，或者是知道某人已经或者能在第一次（第二次，第三次）失败后有足够的耐心尝试。最后的命令可能像下面这样：

```
% gcc -shared -o udf_example.so udf_example.cc -I../innobase/include?
 -I./../include -I./../regex -I. -I../include -I.
```

说明：在有些系统中，应该用两步来完成：先把udf_example.cc编译为udf_example.o，然后从udf_example.o创建共享库（使用gcc -shared -o udf_example.so udf_example.o）。

一旦编译完UDF，就把它放在经常放共享库的地方。对于UNIX系统，可以是任何ld（大部分为/usr/lib或/lib）能搜索到的路径，也可以设置环境变量来指向存储库的路径。输入man dlopen能得到环境变量的名字（通常是LD_LIBRARY或LD_LIBRARY_PATH），应该在启动脚本（mysql.server或mysqld_safe）上设置它。对于Windows系统，一般要把UDF放在WINDOWS\System32或WINNT\System32路径下。把编译后的文件放到合适的地方，比如：

```
% cp udf_example.so /usr/lib
```

一旦放好了文件，有些系统可能需要创建连接（比如执行ldconfig）或在调用函数前重新启动MySQL。

为了从MySQL命令行中调用UDF，使用CREATE FUNCTION语句，语法如下：

```
CREATE [AGGREGATE] FUNCTION function_name RETURNS {STRING|REAL|INTEGER}
       SONAME shared_library_name
```

例子来自于多个UDF（可以把多个UDF绑定在单个的库中）。现在，仅仅调用三个函数，如下：

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME "udf_example.so";
Query OK, 0 rows affected (0.03 sec)
mysql> CREATE FUNCTION myfunc_double RETURNS REAL SONAME "udf_example.so";
Query OK, 0 rows affected (0.00 sec)
mysql> CREATE AGGREGATE FUNCTION avgcost RETURNS REAL SONAME "udf_example.so";
Query OK, 0 rows affected (0.00 sec)
```

现在可以测试新UDF了。为了明白UDF做了些什么，需要看一下udf_example.cc文件。这个扩音器UDF（这个算法的准确名字叫metaphone）接受一个字符串，返回基于字符串的发音方式的结果。它与著名的soundex算法相似，但针对英语做了许多变动。

```
mysql> SELECT METAPHON(word) FROM words;
+-----------------+
```

```
| METAPHON(word) |
+----------------+
| E              |
| BR             |
| S              |
| KS             |
| LS             |
| BR             |
| S              |
| KS             |
| KS             |
| LS             |
+----------------+
10 rows in set (0.00 sec)
```

这并不是特别有用，但至少读者现在知道如何增加函数了。使用下面的代码来测试集合函数是否正在工作：

```
mysql> SELECT AVGCOST(id,1.5) FROM words;
+------------------+
| AVGCOST(id,1.5) |
+------------------+
|           1.5000 |
+------------------+
1 row in set (0.00 sec)
```

只有一个结果，尽管函数工作于一个组中。没有使用GROUP BY从句，因此整个的结果集被当做一个单个的组。如果按照id的内容分组，因为有五个惟一的id值，就会得到五个结果：

```
mysql> SELECT id,AVGCOST(id,1.5) FROM words GROUP BY id;
+------+------------------+
| id   | AVGCOST(id,1.5) |
+------+------------------+
|    1 |           1.5000 |
|    2 |           1.5000 |
|    3 |           1.5000 |
|    4 |           1.5000 |
|    5 |           1.5000 |
+------+------------------+
```

可以使用DROP FUNCTION语句来删除UDF，比如：

```
mysql> DROP FUNCTION myfunc_double;
Query OK, 0 rows affected (0.01 sec)
```

可以通过查看mysql数据库的func表的内容来了解可以使用的UDF列表：

```
mysql> SELECT * FROM mysql.func;
+-----------+-----+----------------+-----------+
| name      | ret | dl             | type      |
+-----------+-----+----------------+-----------+
| metaphon  |   0 | udf_example.so | function  |
| avgcost   |   1 | udf_example.so | aggregate |
+-----------+-----+----------------+-----------+
2 rows in set (0.01 sec)
```

这暗示着用户增加或删除函数需要在func表或者mysql数据库中有INSERT或DELETE的权限。通常只把这个权限给管理员，因为除了存取mysql数据库的安全的危险外，UDF还有许多潜在的危险。

现在，我们从头开始创建UDF。先创建一个标准的（不是集合的）UDF，叫做count_vowels。

标准UDF

标准UDF有一个与UDF同名的主函数，这是必需的，还有两个可选的函数，这两个函数除了在其结尾增加了_init and _deinit以外，它们的名字相似。这些函数必须在同一个库中。

Init函数

Init函数是初始化函数，在处理UDF的开始处调用。它检查传递给UDF的参数（比如类型和个数是否正确）并指定结果的细节（是否可以为NULL，可以有多少个十进位，等等）。

它的定义如下：

```
my_bool function_name_init(UDF_INIT *initid, UDF_ARGS *args, char *message);
```

函数返回一个布尔类型值，没有错误时函数返回false，否则返回true。

initd参数

initd参数是UDF的主要数据结构。它会被传递给全部的三个函数。默认设置的改变都在这个函数中进行。这个结构包含下面的成员：

my-bool maybe_mull 这是一个布尔类型，指明UDF能返回一个NULL值（如果设置为true）或不能 （如果设置为false）。除非函数的任何一个参数能为NULL，默认设置都为false。

unsigned int decimals 指明能够返回的最大的小数个数。默认情况下，采用了传递给主函数的任何一个参数的最多的小数个数。因此，如果传递了203.2、219.12和341.456，小数位默认就是3（根据最后一个参数的.456）。但可以在init函数中设置最大个数限制。

unsigned int max_length 指明返回的结果的最大长度。对于字符串UDF，默认的是最长的字符串变元的长度。对于整数，默认为21个字符 （包括符号）。对于实数，默认的为13（包括符号和小数点） 加小数位的个数。

char *ptr 这是UDF可以使用的指针——比如，传递数据给全部三个函数。如果指针被新数据使用，就应该在init函数中分配内存。

args参数

第二个参数args是一个包含了从查询中传递的参数的结构。它包含下面的成员：

unsigned int arg_count 包含从查询中传递的参数个数。如果UDF包含有一个参数集，为了错误处理，可以使用这个值。

enum Item_result *arg_type 包含一个类型的数组。每个元素与其中的一个参数对应，因此总的元素个数与arg_count的值相同。可能的类型是STRING_RESULT、INT_RESULT和REAL_RESULT。用这个来进行错误检查，或者是把参数强制转换为需要的指定类型。

char **args 包含从查询传递的实际变元的数组。如果参数是常数，可以用args->args[i]的形式存取，这里I是参数的元素个数。对于非常数，args->args[i]是0，就像从行中传递给主函数的实际的值。这将在“Main函数”中进一步讨论。

unsigned long *lengths 一个包含了从查询中传递的每一个参数的可能的最大字符串长度的数组，同样参见“Main函数”中的讨论。

message参数

message参数包含一个字符指针，它用在初始化期间产生任何错误信息的时候。当init函数返回true，表示错误的时候，它就应该被赋予一个值。默认的字符缓存是200字节，但一般都需要一个比这短得多的错误信息（标准终端的宽度是80个字符）。必须用null字符结束。

Main函数

main函数是标准UDF惟一需要的，并且它会被查询中返回的每一行调用。这个函数的返回值与整个UDF的返回值是相同的，可以是字符串、实数或整数。根据返回值，函数应该声明为相应的类型。

如果UDF返回一个字符串：

```
char *function_name(UDF_INIT *initid, UDF_ARGS *args, char *result,
 unsigned long *length, char *is_null, char *error);
```

如果UDF返回一个实数：

```
double function_name(UDF_INIT *initid, UDF_ARGS *args,
 char *is_null, char *error);
```

如果UDF返回一个整数：

```
long long function_name(UDF_INIT *initid, UDF_ARGS *args,
 char *is_null, char *error);
```

对于数字类型，main函数的返回值就是那个值。如果是字符串类型，返回值是指向结果的指针，而长度存储在length参数中。默认的结果缓冲区是255字节，因此，如果结果小于这些，指针就是传递给main函数的结果的指针。如果大于的话，就应该是在init函数中分配的指针（需要使用malloc()来分配空间，然后在deinit函数中释放这个空间）。

initd参数

这个结构（前面讨论的）的全部属性对于main函数都是可用的。没有必要在main函数中修改任何值。

args参数

来自这个结构的属性都在前面讨论过。但在main函数中，args数组包含了从每一行传递给函数的实际参数。因为可能在类型上不同，因此必须强制转换为适当的类型。对于类型为INT_RESULT的参数，强制转换args->args[i]为long long，如下：

```
long long int_val;
int_val = *((long long*) args->args[i]);
```

对于REAL_RESULT类型的参数，强制转换为double，如下：

```
double   real_val;
real_val = *((double*) args->args[i]);
```

对于STRING_RESULT类型的参数，可用字符串类型如args->args[i]，字符串的长度如args->length[i]，结尾的null字符除外（对于数字类型，args->length[i]还是包含着init函数所给的最大长度）。

length参数

这是一个指向一个整数的指针，这个整数是读者设置的返回的值的长度（不包括结尾的null）。

is null参数

如果UDF返回null值，它被设为1；否则，为默认值0。

result参数

这是一个指向字符数组的指针，用来存储UDF的返回值；只分配了255字节，因此如果结果更长的话，就应该使用init函数中的ptr参数。需要分配和释放这个内存。

Deinit函数

这个函数释放init分配的内存并处理其他需要清除的内容，特别是可能在init函数中分配的指针。函数的声明如下：

```
void function_name_deinit(UDF_INIT *initid)
```

创建一个标准的UDF例子

全部介绍完以后，创建一个小的叫做count_vowels的UDF。它只用了一个参数（必须是字符串），并返回字符串中元音的个数。清单6.1包含了这个UDF。

清单6.1 count_vowels.cc

```
#ifdef STANDARD
#include <stdio.h>
#include <string.h>
#else
#include <my_global.h>
#include <my_sys.h>
#endif
#include <mysql.h>
#include <m_ctype.h>
#include <m_string.h>

/* These must be right or mysqld will not find the symbol! */

extern "C" {
my_bool count_vowels_init(UDF_INIT *initid, UDF_ARGS *args, char *message);
void count_vowels_deinit(UDF_INIT *initid);
long long count_vowels(UDF_INIT *initid, UDF_ARGS *args,
 char *is_null, char *error);
}

/* Makes sure there is one argument passed, and that it's a string. */
my_bool count_vowels_init(UDF_INIT *initid, UDF_ARGS *args, char *message) {
  if (args->arg_count != 1 || args->arg_type[0] != STRING_RESULT) {
    strcpy(message,"You can only pass one argument, and it must be a string");
    return 1;
  }
  return 0;
}

/* no need for a deinit function, as we don't allocate extra memory */
void count_vowels_deinit(UDF_INIT *initid){
}

/* count the number of vowels in the string */
long long count_vowels(UDF_INIT *initid, UDF_ARGS *args,char *is_null,➥
 char *error) {
 long long num_vowels = 0; /* the same type as the result of the function */
 char *word = args->args[0];  /* pointer to string */
 int i = 0; /* to loop through the word */
 char c;    /* to contain the letter  */
 while ( ( c = word[ i++ ] ) != '\0' ) {
  switch ( c ) {
   case 'a':
   case 'e':
   case 'i':
   case 'o':
   case 'u':
```

```
    num_vowels++; /* if the letter in c is a vowel, increment the counter */
   }
  }
  return num_vowels;
}
```

存储完这个文件后，编译它，然后把它拷贝到放置库的路径处，就像前面讨论的那样：

```
% make count_vowels.o
g++ -DMYSQL_SERVER -DDEFAULT_MYSQL_HOME="\"/usr/local/mysql\""
 -DDATADIR="\"/usr/local/mysql/var\""
 -DSHAREDIR="\"/usr/local/mysql/share/mysql\""
 -DHAVE_CONFIG_H -I../innobase/include -I./../include
 -I./../regex -I. -I../include -I. -O3 -DDBUG_OFF
 -fno-implicit-templates -fno-exceptions -fno-rtti -c repeat_str.cc
% gcc -shared -o count_vowels.so count_vowels.cc -I../innobase/include➥
 -I./../include -I./../regex -I. -I../include -I.
```

现在连接到MySQL，加载函数并运行一个测试：

```
mysql> CREATE FUNCTION count_vowels RETURNS INTEGER SONAME "count_vowels.so";
Query OK, 0 rows affected (0.02 sec)
mysql> SELECT id,word,count_vowels(word) FROM words;
+------+--------+--------------------+
| id   | word   | count_vowels(word) |
+------+--------+--------------------+
|    1 | aeiou  |                  5 |
|    2 | bro    |                  1 |
|    3 | so     |                  1 |
|    4 | kisso  |                  2 |
|    5 | lassoo |                  3 |
|    2 | bro    |                  1 |
|    3 | so     |                  1 |
|    4 | kisso  |                  2 |
|    4 | kisso  |                  2 |
|    5 | lassoo |                  3 |
+------+--------+--------------------+
10 rows in set (0.00 sec)
```

如果传递了一个非字符串，比如从id域来的参数，或者是一个以上的参数，就会得到自己指定的错误信息：

```
mysql> SELECT id,word,count_vowels(id) FROM words;
ERROR:
You can only pass one argument, and it must be a string
```

警告： 如果改变UDF，在重新上载之前，要确保从MySQL中删除了这个函数。如果不这样，就很可能使MySQL崩溃，并不得不重新启动。

理解集合函数

集合函数是那些可以被GROUP BY从句使用的像SUM()和AVG()那样的函数。为了创建集合UDF，使用与标准UDF一样的函数，除此之外，还有两个要求：reset和add函数。其他函数的作用也是不同的：

Reset函数 这个函数在每个新组的开始调用。用于组计算的数据也在这里设置。声明这个函数如下：

```
char *xxx_reset(UDF_INIT *initid, UDF_ARGS *args,
                char *is_null, char *error);
```

Add函数 除了组的第一行以外，对每一行它都会被调用。可能想要对每一行都调用它，这种情况下，就需要从reset函数中调用。

Main函数 main函数只在每组的数据中调用一次（在最后），因此被用来进行全组数据需要的计算（通常通过initd->ptr来存取）。

Init函数 与标准的UDF行为相同，只是ptr属性在集合函数中更重要。它存储每组在add函数中被添加的数据。之后main函数能够存取关于全组的数据。

Deinit函数 与在标准UDF中作用相同，只是它总是存在，就像需要清除ptr一样。

创建集合UDF的例子

对于集合UDF，应该对count_vowels UDF做一些改变，以便它能够统计元音的组数。下面将创建一个带元素count的叫做data的结构。在add函数中，只要遇到元音，就要增加data->count（在每一行中都会调用），并在reset函数中重置值（每组调用一次）。因为在第一行add函数不会被显式调用，应该在reset函数中调用，以确保每组的第一行也被统计了。清单6.2包含了这个集合UDF。

清单6.2 count_agg_vowels.cc

```
#ifdef STANDARD
#include <stdio.h>
#include <string.h>
#else
#include <my_global.h>
#include <my_sys.h>
#endif
#include <mysql.h>
#include <m_ctype.h>
#include <m_string.h>          // To get strmov()

#ifdef HAVE_DLOPEN

/* These must be right or mysqld will not find the symbol! */

extern "C" {
my_bool count_agg_vowels_init( UDF_INIT* initid, UDF_ARGS* args, char* message );
```

```
void count_agg_vowels_deinit( UDF_INIT* initid );
void count_agg_vowels_reset( UDF_INIT* initid, UDF_ARGS* args,➥
char* is_null, char *error );
void count_agg_vowels_add( UDF_INIT* initid, UDF_ARGS* args,➥
char* is_null, char *error );
long long count_agg_vowels( UDF_INIT* initid, UDF_ARGS* args,➥
char* is_null, char *error );
}

struct count_agg_vowels_data {
  unsigned long long count;
};

/* Count the number of vowels */
my_bool
count_agg_vowels_init( UDF_INIT* initid, UDF_ARGS* args, char* message ) {
  struct count_agg_vowels_data* data;

  if (args->arg_count != 1 || args->arg_type[0] != STRING_RESULT) {
    strcpy(message,"You can only pass one argument, and it must be a string");
    return 1;
  }

  initid->max_length = 20;
  data = new struct count_agg_vowels_data;
  data->count = 0;
  initid->ptr = (char*)data;

  return 0;
}

/* free the memory allocated to from ptr */
void
count_agg_vowels_deinit( UDF_INIT* initid ) {
  delete initid->ptr;
}

/* called once at the beginning of each group. Needs to call the add
function as well resets data->count to 0 for the new group */
 void
count_agg_vowels_reset( UDF_INIT* initid, UDF_ARGS* args,?
char* is_null, char* message ) {
  struct count_agg_vowels_data* data = (struct count_agg_vowels_data*)initid-
>ptr;
  data->count           = 0;

  *is_null = 0;
  count_agg_vowels_add( initid, args, is_null, message );
}
```

```
/* called for every row, add the number of vowels to data->count */
void
count_agg_vowels_add( UDF_INIT* initid, UDF_ARGS* args, char* is_nul,?
char* message ) {
  struct count_agg_vowels_data* data     =➥
(struct count_agg_vowels_data*)initid->ptr;
  char *word = args->args[0];  /* pointer to string */
  int I = 0;
  char c;

  while ( ( c = word[ I++ ] ) != '\0' ) {
   switch ( c ) {
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u':
    data->count++;
   }
 }
}
/* returns data->count, or a null if it cannot find anything */
long long
count_agg_vowels( UDF_INIT* initid, UDF_ARGS* args, char* is_null, char* error
)
{
  struct count_agg_vowels_data* data = (struct count_agg_vowels_data*)initid-
>ptr;
  if (!data->count)
  {
    *is_null = 1;
    return 0;
  }

  *is_null = 0;
  return data->count;
}

#endif /* HAVE_DLOPEN */
% make count_agg_vowels.o
g++ -DMYSQL_SERVER -DDEFAULT_MYSQL_HOME="\"/usr/local/mysql\""
 -DDATADIR="\"/usr/local/mysql/var\""
 -DSHAREDIR="\"/usr/local/mysql/share/mysql\""
 -DHAVE_CONFIG_H -I../innobase/include -I./../include
 -I./../regex -I. -I../include -I. -O3 -DDBUG_OFF
 -fno-implicit-templates -fno-exceptions -fno-rtti -c repeat_str.cc
```

```
% gcc -shared -o count_agg_vowels.so count_agg_vowels.cc -I../innobase/include
 -I./../include -I./../regex -I. -I../include -I.
```

现在连接到MySQL，加载函数并运行一个测试：

```
mysql> CREATE AGGREGATE FUNCTION count_agg_vowels RETURNS INTEGER
 SONAME "count_vowels.so";
Query OK, 0 rows affected (0.02 sec)

mysql> SELECT count_agg_vowels(word) FROM words;
+-------------------------+
| count_agg_vowels(word) |
+-------------------------+
|                      21 |
+-------------------------+
1 row in set (0.01 sec)
mysql> SELECT id,count_agg_vowels(word) FROM words GROUP BY id;
+------+-------------------------+
| id   | count_agg_vowels(word) |
+------+-------------------------+
|    1 |                       5 |
|    2 |                       2 |
|    3 |                       2 |
|    4 |                       6 |
|    5 |                       6 |
+------+-------------------------+
5 rows in set (0.00 sec)
```

UDF中问题的解决

有许多原因造成UDF不工作。如果UDF实现得不好，MySQL很容易崩溃，因此在一个运行的现实的系统中，要小心实现没有试用过的UDF。每个系统都有自己的复杂性，但下面是一些更常见的问题：

确保在上载或重新调用函数之前，已经删除了任何已存在的同名的函数（如果在更新函数）。如果不能用常规的方法删除之前增加的坏的UDF，就应该用手工的方式从func表中删除它。

- 尽管通常不需要，也可以在重新调用函数之前尝试停止和重启MySQL。
- 确保创建函数时返回的类型与在代码中的主函数返回的类型匹配（字符串、实数或整数）。
- 为了实现UDF，应该用--with-mysqld-ldflags=-rdynamic配置MySQL。

小结

MySQL容许增加用户定义的函数（UDF）。在查询中，可以像使用普通的函数一样使用UDF。有两种类型的UDF：集合和标准UDF。集合UDF工作在一组数据上并且能在GROUP BY从句中使用。 标准UDF工作在单个的数据行上。

标准UDF包括三个函数：必需的并被每一行调用的主函数； 分别在开始和结束时调用一次的初始化和结束函数。集合函数还使用增加函数（在主函数中被每一行调用，在每一组的最后只调用一次）和重置函数（在每一组的开始调用）。

第二部分　设计一个数据库

第7章　理解关系型数据库

就像在看到前一个时代的艺术状况后，我们才惊叹于电影特技的效果那样，我们也只有在了解了之前的数据库的状况后，才能更完全地欣赏关系数据库的威力。

关系数据库容许任何一张表通过普通域的方式与其他的表关联。它是一个高度灵活的系统，而大部分现代数据库都是关系型的。

本章的主要内容：

- 分层数据库模型
- 网络数据库模型
- 关系数据库模型
- 学习基本术语
- 表键值和外部键
- 视图

探讨早期的数据库模型

在数据库出现之前，存储数据的惟一方法是使用不相关的文件。程序员得花很大精力去取得数据，而程序则得进行复杂的分解工作。

像Perl那样的语言，由于拥有强大的处理文本的规则表达式的思想，使得这个工作比以前简单了一些；但从文件中获取数据还是一个有挑战性的工作。由于没有一个标准的方法存取数据，系统更容易出错，开发起来更慢，更难维护。数据冗余（数据不必要的重复）和糟糕的数据完整性（数据没有在全部需要的地方都修改，导致提供错误的或过期的数据）是在文件方式的数据存取方法中经常发生的事。因为这些原因，产生了数据管理系统（DBMS）用来提供标准的、可靠的存取和更新数据的方式。它们在应用和数据之间提供了一个中间层，这样，程序员就能集中精力于开发应用之上，而不用担心数据存取的问题。

数据库模型是一个关于如何表示数据的逻辑模型。数据库设计员不需要担心数据的物理存储，数据库模型容许他们站在一个更高的角度或者是更概念性的级别上，以减少应用开发针对的现实世界的问题与技术实现之间的鸿沟。

有多种数据库模型。我们先学习两种普通的模型：分层数据库模型和网络数据库模型。然后再探讨MySQL（以及大部分现代DBMS）使用的关系模型。

理解分层数据库模型

早期的模型是头朝下的树形的分层数据库模型。文件由父子方式关联，其中一个父亲可以与多个孩子关联，而一个孩子只能与一个父亲关联。大部分人都很熟悉这种结构——即大部分文件系统工作的方式。一般有一个包含各种其他路径和文件的根或顶级路径。每个子路径又可以包含许多文件和路径，等等。每个文件或路径自己只能存在于一个路径中——它只有一个父亲。从图7.1可以看到，A1是根路径，B1与B2是它的孩子。B1是C1、C2和C3的父亲，而它们又有自己的孩子。

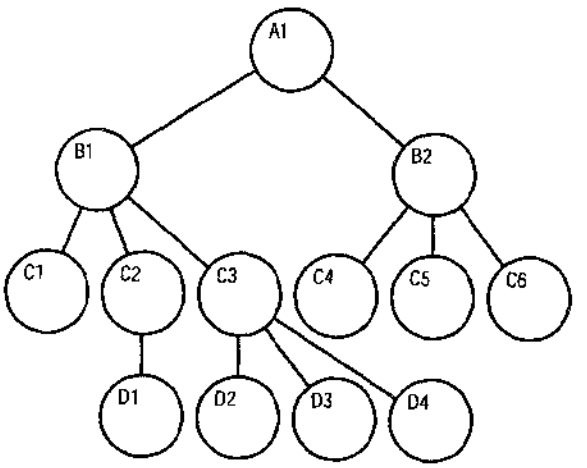

图7.1 分层数据库模型

尽管在处理不相关的文件时，这个模型已经有了巨大的提高，但还是有一些严重的不足。它可以很好地代表一对多的关系（一个父亲多个孩子；比如，一个分公司有多个职员），但在表示多对多的关系时存在问题。在分层模型中，很难实现诸如产品文件和定单文件之类的关系。特别是，一个定单包含许多产品，而一个产品能够出现在多个定单中。同样，分层模型也不太灵活，因为增加一个新关系会导致已存在的结构的整个改变，这样整个已存在的应用都需要改变。当有人在项目即将发布之前，忘记了一个文件类型又想把它加到结构中时，就不太有趣了。

程序员为了遍历模型来存取需要的数据，需要清楚地了解数据结构，这样开发应用就很复杂。就像在前面的章节看到的那样，在两张相关的表中存取数据，只需要知道来自这两张表中读者要求的域。在分层模型中，需要了解两者之间的整个链。比如，为了得到A1到D4的关联数据，需要路径：A1、B1、C3和D4。

理解网络数据库模型

网络数据库模型是在层数据库模型之上的一个进步，它被设计用来解决分层数据库模型的一些问题，特别是缺乏灵活性的问题。在这个模型中，容许一个孩子对应多个父亲（孩

子被称为成员，而父亲被称为拥有者），而不是一个孩子只能对应一个父亲。它声明了需要的更复杂的模型关系，比如前面提到的定单/产品的多对多关系。从图7.2可见，A1有两个成员，B1与B2。B1拥有C1、C2、C3和C4。但是，在这个模型中，C4有两个拥有者，B1和B2。

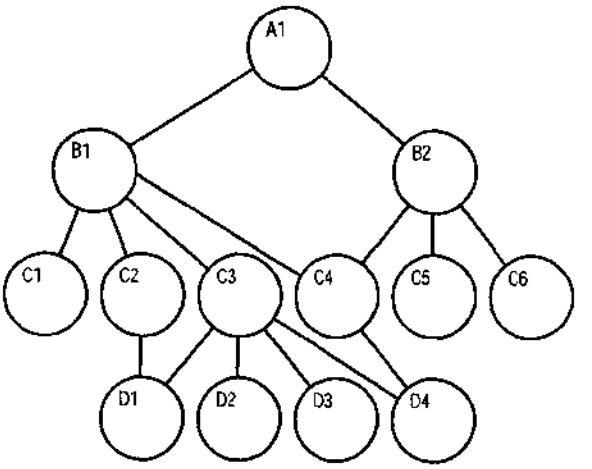

图7.2 网络数据库模型

当然，这个模型也有问题，否则，大家都还在使用它。它更难实现和维护，并且，尽管比分层模型灵活，但也还有灵活性问题。并不是任何关系都能设置另一个拥有者，为了更高效地利用这个模型，程序员还得很好地理解数据结构。

理解关系数据库模型

关系数据库模型是网络数据库模型的一个巨大飞跃。关系模型通过普通的域方法使一个文件与任何其他的文件关联，这代替了依赖于父子或拥有者－成员的关系。由于可以改变数据库方案而不影响系统存取数据的能力，设计的复杂性突然被大大地减小了。而且因为数据存取是通过文件之间的直接关系而不是通过路径的方式，文件之间的新关系能很容易地增加。

在1970年，当E. F. Codd开发出这个模型时，它被认为是一个不切实际的模型。增加的易用性来自于效率的极大损失，而且当时的硬件技术还不能实现这个模型。当然，从那时开始，硬件技术已经取得了巨大的进步，现在最简单的计算机都能运行复杂的关系数据库管理系统。

就像第一部分“使用MySQL”介绍的，关系数据库是与SQL共同发展的。SQL的简单——一个新手都能够在短时间内学会基本的查询——是关系模型流行的一个主要原因。

表7.1和表7.2通过stock_code域关联。任何两张表都能简单地通过创建它们共有的域来建立联系。

表7.1 Product表

STOCK_CODE	DESCRIPTION	PRICE
A416	钉子，盒	0.14美元
C923	图钉，盒	0.08美元

表7.2 Invoice表

INVOICE_CODE	INVOICE_LINE	STOCK_CODE	QUANTITY
3804	1	A416	10
3804	2	C923	15

基本术语介绍

关系模型使用某种术语来描述它的部件。如果阅读过第一部分“使用MySQL”，就会对其中的许多都很熟悉了：

- Data（数据）是保存在数据库中的值。就其自身来说，数据基本没有意义。CA 684-213是DMV（机动车部）数据库中数据的例子。
- Information（信息）是处理过的数据。比如，CA 684-213是Lyndon Manson在DMV数据库中的登记号。
- Database（数据库）是表的集合。
- 表由记录（表中的水平行，也叫tuples）构成。每个记录都必须是惟一的，可以在表中用任何顺序存储。
- 记录是由域（表的垂直列，也叫属性）构成的。记录基本上是一个事实（比如，一个客户或一个销售）。
- 域可以是各种各样的类型type。就像在第2章“数据类型与表类型”看到的，MySQL有许多类型，这些类型基本分成三类：字符、数字和日期。比如，客户的名字可以是字符域，客户的生日是日期域，而客户孩子的个数则是数字域。
- 一个域的容许值的范围叫域（也叫域的规格）。比如，credit_card域可能被限制为Mastercard，Visa和Amex。
- 如果域不包含任何东西，就说这个域包含null值。Null域会导致计算的复杂和对日期精确性的影响。因为这个原因，许多域规定不能包含null值。
- 键存取表中的指定记录。
- 索引index是提高数据库性能的机制。索引经常和键混淆。严格地说，索引是物理结构的一部分，而键是逻辑结构的一部分，但经常会把这些术语混用。
- 视图是由实际表的子集构成的虚表。
- 一对一（1:1）关系就是：在关系中的第一张表的实例，在第二张表中也只有一个实例对应。一个例子：连锁店的自动售货机。每台自动售货机只能在一个商店，而每个商店只有一个自动售货机（见图7.3）。

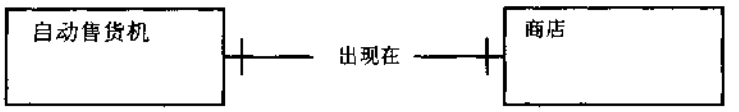

图7.3　一对一关系

- 一对多（1:M）关系：在关系中的第一张表的实例在第二张表中有多个实例对应。这是一个普通的关系种类。例子是雕塑家和他们的雕塑的关系。每个雕塑家可以创作多个雕塑，但每个雕塑只能由一个雕塑家创作（见图7.4）。

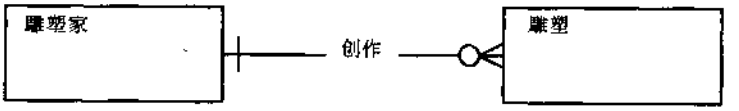

图7.4　一对多关系

- 多对多（M:N）关系：第一张表的实例在第二张表中有多个实例对应，而第二张表的实例在第一张表中有多个实例对应。比如，一个学生可以有多个老师，而一个老师，也可以有多个学生（见图7.5）。

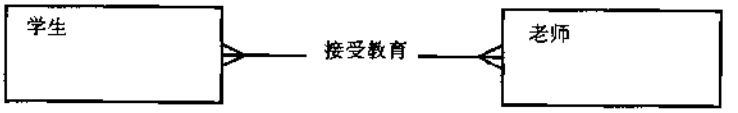

图7.5　多对多关系

- 强制关系是指在关系中，第一张表中的每个实例在第二张表中都必须有一个或多个实例存在。比如，一个音乐小组要成立，必须存在至少一个音乐师。
- 可选关系是指，第一张表中的每个实例在第二张表中可以有实例存在。比如，在数据库中没有写过书的作者（换句话说，潜在的作者），这个关系就是可选的。但反过来却不一定正确，比如，列出的书必须有一个作者。
- 数据一致性是指数据的准确性、有效性和一致性。不好的数据一致性的例子：客户的电话号码存储在两个不同的地方；另一个例子是包含的成绩记录指向的老师已经不在学校。在第8章“范式化化数据库”中，读者将学习帮助减少这种问题风险的技术：标准化数据库。

既然已经介绍了一些基本术语，下面的小节将更详细地讨论表键，它是关系数据库的一个基本方面。

表键介绍

键（key），顾名思义，是存取表的钥匙。如果知道这个键，就知道如何区分特定的记录和表之间的关系。

候选键是一个惟一区分记录的域或域的联合。它不能包含null值，而且必须是惟一的。（如果有重复值，就不能区分惟一记录了）。

主键是一个被指定为在数据库结构中区分惟一记录的候选键。作为例子，表7.3显示了Customer表。

表7.3 Customer表

CUSTOMER_CODE	FIRST_NAME	SURNAME	TELEPHONE_NUMBER
1	John	Smith	448-2143
2	Charlotte	Smith	448-2143
3	John	Smith	9231-5312

第一眼看过去，这张表有两个可能的候选键。customer_code或者是first_name、surname和telephone_number的组合都可以胜任。最好选择域个数少的候选键作为主键，因此，在这个例子中使用了customer_code。反思一下，第二个组合可能不是惟一的。从理论上说，first_name、surname和telephone_number的组合可能重复，比如，父亲有一个同名的儿子，他的联系电话也一样。如果把这三个域考虑为主键的状态，系统就必须排除这个可能。

在说英语的国家，可能有许多John Smiths，但可以通过分配一个惟一的号来避免混淆。一旦创建了主键，剩余的候选键就被标为选择键。

外部键介绍

已经知道两张表之间创建的关系是通过在表之间赋一个共同的域而实现的。这个共同域必须是一张表的主键。考虑用户表与销售表之间的关系。如果在销售表中使用另一个不惟一的域比如用户的first name来代替主键customer_code会不太好。这样将永远不能知道是销售给哪个客户的。因此，在图7.6中，在sale表中，customer_code被叫做外部键；换句话说，它是外表中的主键。

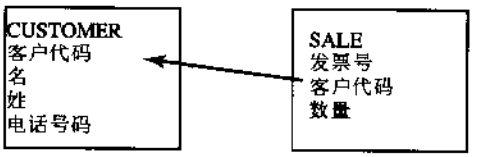

图7.6 设置外部键

外部键容许指向一致性。意思是如果外部键包含一个值，这个值指向一个在关联表中已经存在的记录。比如，考虑表7.4和表7.5。

表7.4 教师表

代码	名	姓
1	Anne	Cohen
2	Leonard	Clark
3	Vusi	Cave

表7.5 课程表

课程名称	教师
编程简介	1
信息系统	2
系统软件	3

这里存在指向一致性，因为全部存在于课程表中的老师都存在于教师表中。但是，假设Anne Cohen离开了学院，因此从教师表中删除了她。在没有实现指向一致性的情况下，她就会从教师表中被删除掉，但没有从课程表中删除，这些显示在表7.6和表7.7中。

表7.6 教师表

代码	名	姓
2	Leonard	Clark
3	Vusi	Cave

表7.7 课程表

课程名称	教师
编程简介	1
信息系统	2
系统软件	3

现在，当查看哪个人讲授编程简介，就会送出一个不存在的记录。这叫做数据一致性不好。

外部键容许层叠式的删除和更新。比如，Anne Cohen离开了，带走了她的编程简介，有关她的信息都应该可以用一条语句从教师和课程表中删除。从相关的表中“层叠式”删除，就删除了全部相关的记录。从版本3.23.44开始，MySQL已经支持用InnoDB表类型来检查外部键，从版本4.0.0开始支持“层叠式”删除。要记住的是，进行指向一致性会对效率有损失。如果没有它，就变成应用对维护数据一致性负责。

外部键可以包含null值，表示不存在关系。

介绍视图

视图是虚拟的表。它们只是一个结构而不包含数据。 它们的目的是让用户看到实际数据的一个子集。视图是MySQL最经常需要的特征，准备在版本5中实现。

视图可以包含一张表的子集。比如，表7.8是一个完全表，即表7.9的子集。

表7.8 学生视图

学生视图
名
姓
年级

表7.8 学生表

学生
Student_id
名
姓
年级
地址
电话

这张视图用来让其他的学生了解同学的成绩但不能存取个人信息。

视图也可以是许多表的组合，比如显示在表7.10中的视图。 它是表7.11、表7.12和表7.13的组合。

表7.9 学生年级视图

学生年级视图
名
姓
课程说明
年级

表7.10 学生表

学生
Student_id
名
姓
地址
电话

表7.11 课程表

课程
Course_id
课程说明

表7.12 年级表

年级
Student_id
Course_id
年级

视图对于安全也很有用。在大的机构里，可能有很多开发人员工作在一个项目上，视图可以让开发人员只存取相关的数据。他们不需要的，即使在同一张表上，都可以被隐藏，以防被看见或被操作。它还使针对开发人员的查询更简单了。比如，没有视图，开发人员必须使用下面的查询来获得视图中的域：

```
SELECT first_name,surname,course_description,grade FROM student,
 grade, course WHERE grade.student_id = student.student_id AND
 grade.course_id = course.course_id;
```

使用视图，开发人员可以用下面的方法做到同样的事：

```
SELECT first_name,surname,course_description,grade FROM student_grade_view;
```

这对一个没有学习过连接的初级程序员来说很简单，而且即使对一个高级程序员，烦恼也要少一些。

小结

在有数据库之前，程序员把数据存储在文件中。但是，对于程序员来说，存取文件数据效率低下，由此产生了数据库。

分层数据库用从上到下一对多的结构存储数据。它们不够灵活并给程序员增加了许多其他的工作。

网络数据库能够很容易地表现多对多的关系，但它们很难开发和维护。

关系数据库容许任何一张表通过普通域的方式与其他的表关联。它是一个高度灵活的系统，大部分现代数据库都是关系型的。

表键值容许存取数据库中的记录。主键是一个或者许多惟一表明一行的属性。外部键是一个或多个属性，这个或这些属性是另一张表中的主键。

表视图是已存在的表的逻辑子集。它们不包含任何数据，但能使开发人员更容易开发应用，增强安全性，等等。

第8章 范式化数据库

本章介绍优化数据库设计的有力工具：范式化。从E.F. Codd在20世纪70年代开发出范式化开始，范式化就是大部分数据库设计的基本要求。通过本章总结的几个步骤，就能够减少数据异常和使维护变得容易。

本章的主要内容：

- 第一范式
- 第二范式
- 第三范式
- Boyce-Codd范式
- 第四范式
- 第五范式
- 反范式化

理解范式化

在第一部分“使用MySQL”中，在MySQL中创建了一些表。读者可能已经在小的项目，即数据库包含一或两张表的项目中使用过MySQL。但随着经验的增加和开始处理大的项目，就会发现查询变得越来越复杂和笨拙，这时开始遇到效率问题，或者是数据异常也开始出现了。如果没有数据库设计和范式化的信息，这些问题可能就不可克服了，掌握MySQL的计划就可能不能进行下去了。数据库范式化是一个可以帮助避免数据异常和管理数据时出现的其他问题的技术。它由各种阶段的表的转换组成：第一范式，第二范式，第三范式，等等。它的目标是：

- 消除数据冗余（因此使用更少的空间）
- 使对数据的改变更容易，而且能避免这样做的时候出现异常
- 更容易执行一致性约束
- 产生一个与数据表示的情况更相似的更容易理解的结构，并容许结构的增长

下面通过创建一个简单的数据集来开始学习。先看一下范式化的过程，理解一下范式化的原因，而不要担心理论。做过了这些以后，再介绍理论和范式化的各种步骤，这样下次做的时候会使整个过程简单一些。

假设正在处理一个系统，此系统记录某个位置种的植物和与其关联的土质描述。

位置：
位置代码：11
位置名：Kirstenbosch Gardens

包含下面三种植物：

植物代码：431
植物名：Leucadendron
土质分类：A
土质描述：Sandstone

植物代码：446
植物名：Protea
土质分类：B
土质描述：Sandstone/Limestone

植物代码：482
植物名：Erica
土质分类：C
土质描述：Limestone

位置：
位置代码：12
位置名：Karbonkelberg Mountains

包含下面的两种植物：

植物代码：431
植物名：Leucadendron
土质分类：A
土质描述：Sandstone

植物代码：449
植物名：Restio
土质类型：B
土质描述：Sandstone/Limestone

前面的数据中有一个问题。关系数据库中表是网格或表格（MySQL与其他的现代数据库一样，是一个关系数据库）的形式，每一行都是一个惟一的记录。重新用列表报告的方式安排这个数据 （显示在表8.1中）。

表8.1 用列表报告的形式显示植物数据

位置代码	位置名	植物代码	植物名	土质分类	土质描述
11	Kirstenbosch Gardens	431	Leucadendron	A	Sandstone
		446	Protea	B	Sandstone/Limestone
		482	Erica	C	Limestone

（续表）

位置代码	位置名	植物代码	植物名	土质分类	土质描述
12	Karbonkelberg Mountains	431	Leucadendron	A	Sandstone
		449	Restio	B	Sandstone/Limestone

在数据库中，如何把这些数据输入表中？可以拷贝一个前面看到的报告的版式，产生一个类似表8.2的表。Null域表示在这个域中没有输入数据。

表8.2 尝试创建带植物数据的表

位置代码	位置名	植物代码	植物名	土质分类	土质描述
11	Kirstenbosch Gardens	431	Leucadendron	A	Sandstone
NULL	NULL	446	Protea	B	Sandstone/Limestone
NULL	NULL	482	Erica	C	Limestone
12	Karbonkelberg Mountains	431	Leucadendron	A	Sandstone
NULL	NULL	449	Restio	B	Sandstone/Limestone

这张表没有太多的用。前面的三行实际是一个组，属于同一个位置。如果仅操作第三行，数据就是不完整的，因为不能告诉别人发现Erica的地方。同样，根据表代表的内容，不能使用位置码或者是任何其他的域作为主键（记住，主键是一个惟一表示一个记录的域或域列）。如果不能惟一地区分其中的每个记录，那这张表就没什么用。

因此，问题的答案是保证表的每行都是单独的，而不是组或集的一部分。为了做到这点，删除数据组或集，并使每一行成为一个完整的记录，结果显示在表8.3中。

表8.3 每个记录代表惟一

位置代码	位置名	植物代码	植物名	土质分类	土质描述
11	Kirstenbosch Gardens	431	Leucadendron	A	Sandstone
11	Kirstenbosch Gardens	446	Protea	B	Sandstone/Limestone
11	Kirstenbosch Gardens	482	Erica	C	Limestone
12	Karbonkelberg Mountains	431	Leucadendron	A	Sandstone
12	Karbonkelberg Mountains	449	Restio	B	Sandstone/Limestone

说明： 主键用斜体字显示在表8.3和接下来的表中。

注意位置码不能是关于自己的主键。它并不惟一地区分一行数据，因此，主键必须是位置码和植物码的组合。这两个域一起惟一地区分开数据行。设想一下：永远不会在一个特定的地方多次增加同样的植物类型。一旦知道了在那个位置发生了那样的事实，就足够了。如果想记录一个位置的植物数量——对这个例子只对植物的传播感兴趣——不必对每种植物都增加一个完整的新类；而只需要加一个数量域。如果因为某种原因添加了超过一个的植物/位置组合的实例，就需要增加某些内容到键上，以使其惟一。

现在，数据可以放入到表格式中了，但还是有一些问题。表存储了代码为11指向Kirstenbosch Gardens的信息3次！除了浪费空间，还有另外一个严重的问题。仔细看一下表8.4中的全部数据。

表8.4 数据异常

位置代码	位置名	植物代码	植物名	土质分类	土质描述
11	Kirstenbosch Gardens	431	Leucadendron	A	Sandstone
11	Kirstenbosh Gardens	446	Protea	B	Sandstone/Limestone
11	Kirstenbosch Gardens	482	Erica	C	Limestone
12	Karbonkelberg Mountains	431	Leucadendron	A	Sandstone
12	Karbonkelberg Mountains	449	Restio	B	Sandstone/Limestone

注意到表8.4中数据的奇怪之处吗？如果注意到了，那非常好！第二个记录Kirstenbosch拼写错误。现在想像从表中成千的记录中指出这个错误的难度！通过使用表8.4中的结构，指出数据异常的机会就大大增加了。

答案很简单——删除重复记录。而需要做的就是查找部分依赖——换句话说，就是查找依赖于键的一部分而不是整个键的域。因为位置码和植物码两者构成了键值，即需要查找仅依赖于位置码或植物码的域。

在这个情况中，有许多域是这样的。位置名依赖于位置码（植物码与决定项目名没有关系），而植物名、土质码和土质名都依赖于植物数。因此，取出这些域，如表8.5所示。

显然不能完全删除数据而不考虑数据库。将其取出并放入新表中，包含部分依赖的域和它们依赖的域。对部分依赖的每个键值域创建一个新表（在这个例子中，两者都已经是主键的一部分，但情况并不总是这样）。这样，就区分了植物码的植物名、土质描述和土质分类。新表包含主键即植物码，同时有植物名、土质描述和土质分类，如表8.6所示。

表8.5 删除不依赖于整个键的域

位置码	植物码
11	431
11	446
11	482
12	431
12	449

表8.6 创建一个带植物数据的新表

植物码	植物名	土质分类	土质描述
431	Leucadendron	A	Sandstone
446	Protea	B	Sandstone/Limestone
482	Erica	C	Limestone
449	Restio	B	Sandstone/Limestone

对位置码也做同样的工作，如表8.7。

表8.7 创建带位置数据的新表

位置码	位置名
11	Kirstenbosch Gardens
12	Karbonkelberg Mountains

明白这些表是如何消除前面的重复问题的吗？只有一个记录包含有Kirstenbosch Gardens，因此发现拼写错误的概率就大的多了。并且在不同的记录中，没有浪费空间用来存储名字。注意位置码和植物码域在这两张表中重复了。这些是创建关系的域，允许将各种植物域与各种位置关联。显然除了彻底删除关联以外，没有其他办法来解决这些域的重复问题，好在重复存储小代码比大片文本的效率高的多。

但表还不是很完美。还是有机会碰到异常。仔细检查表8.8。

表8.8 另一个异常

植物码	植物名	土质分类	土质描述
431	Leucadendron	A	Sandstone
446	Protea	B	Sandstone/Limestone
482	Erica	C	Limestone
449	Restio	B	Sandstone/Limestone

表8.8的问题是Restio与sandstone关联了，但实际上，土质分类为B的应该是sandstone和limestone的混合（在这个例子中，土质分类决定了土质描述）。这里又一次存储了数据冗余：对每个植物，土质分类对土质描述的关系是相对于整体来存储的。像以前一样，解决的方案

是取出额外的数据并放在另外的表中。实际上在这个阶段需要找的是转换关系，或者是一个非键的域与另一个非键域的依赖关系。土质描述，尽管在某种程度上依赖于植物码（如果用前面的方式来看的话，可能是一个部分依赖），实际上却依赖于土质分类。因此，必须删除土质描述：再一次取出它并放到一个新表中，同时带着实际的键（土质分类），如表8.9和8.10所示。

表8.9 删除了土质描述的植物数据

植物码	植物名	土质分类
431	Leucadendron	A
446	Protea	B
482	Erica	C
449	Restio	B

表8.10 创建一个带土质描述的新表

土质分类	土质描述
A	Sandstone
B	Sandstone/Limestone
C	Limestone

这样，出现异常的概率再一次降低了。现在除了sandstone和limestone的组合之外，不可能再把土质分类B与任何另外的事物关联了。土质描述对土质分类的关系只存储在一个地方：新土质表中，而且能够保证它们是精确的。

为了指导读者，现在不带数据表再看一下这个例子。通常在设计系统的时候，还没有可用的完整的测试数据集，而且如果不一定明白数据之间的关系，这也就不必要了。作者已经用表演示了存储在表中的没有范式化的数据的结果，没有它们，就只能依赖于域之间的依赖关系，这就是范式化数据库的关键。

开始，数据结构如下：

位置码

位置名

1－n植物数（1－n是这个域有许多的简便写法——换句话说，它是一个重复的组）

1－n植物名

1－n土质分类

1－n土质描述

这是一个完全没有范式化的结构——换句话说，就是0范式。因此，开始范式化的过程，就是从0范式到第一范式。

第一范式

第一范式中的表遵循下面的规则：

- 没有重复的组。

- 全部的键属性都定义了。
- 全部的属性都依赖于主键。

上面的意思就是数据必须能填入表格的形式中，而且每个域只包含一个值。这就是主键定义的阶段。有些资料说在第一范式中，对一张表定义主键是不必要的，但在这个阶段一般都这样做，并且在进行到下一个阶段之前是必要的。除了理论上的争论以外，都必须定义主键。

提示： 原子性的原则在这个阶段也应该很好地得到应用，尽管它并不总是定义为第一范式的一部分。就是说全部的列都应该包含最小的部分，或者说不可分割。一个一般的例子是一个人创建了名字域，而后后悔没有创建为姓和名两个域。

到现在为止，植物的例子还没有键值，并且有重复的组。为了得到第一范式，需要定义一个主键并改变结构以便没有重复的组；也就是，每行/列的交互只包含一个值，且仅仅有一个值。如果没有这个，就不能把数据放入大部分数据库需要的二维表中。把位置码与植物码一起定义为主键（任何一个都不能单独区分惟一的记录），并用一个单一值属性替代重复的行。做完这个后，剩余的数据显示在表8.11中。

表8.11 第一范式

植物位置表
位置码
位置名
植物码
植物名
土质分类
土质描述

这个表现在是第一范式，它是第二范式吗？

第二范式

第二范式中的表遵循下面的规则：

- 是第一范式
- 不包含部分依赖（属性只依赖于主键的一部分）

提示： 如果属性只依赖于主键的一部分，这个主键一定由超过一个域组成。如果主键只包含一个域，这个表还是第一范式的话，它也自动成为了第二范式。

检查一下全部的域。位置名只依赖于位置码。植物名、土质分类和土质描述只依赖于植物码（假设每种植物只在一种土质类型处出现，这个例子就是这种情况）。删除这些域中的每一个，把它们放入分离的表中，而键就是原来它们依赖的键的一部分。比如，对植物名，键是植物码。产生的表为8.12、表8.13和表8.14。

表8.12　部分依赖删除后的植物位置表

植物位置表
植物码
位置码

表8.13　产生于依赖于植物码的域的表

植物表
植物码
植物名
土质分类
土质描述

表8.14　产生于依赖位置码的域的表

位置表
位置码
位置名

产生的表现在属于第二范式，它是第三范式吗？

第三范式

第三范式中的表遵循下面的规则：

- 是第二范式
- 不包含传递依赖（非键属性通过另一个非键属性依赖于主键）

提示： 如果表只包含一个非键属性，显然一个非键属性依赖于另一个非键属性是不可能的。任何像这样的表，如果是第二范式的，就自动是第三范式的。

只有植物表有超过一个的非键属性，其他的已经是第三范式了，不用再考虑了。因为表都是第二范式的，全部域都以同样的方式依赖于主键。但这是通过另一个非键域而依赖的吗？植物名既不依赖于土质分类也不依赖于土质描述。土质分类也既不依赖于土质描述也不依赖于植物名。但是，土质描述依赖于土质分类。使用与前面相同的过程删除它，带着作为键值依赖的属性放入表中。产生的表为8.15、表8.16、表8.17和表8.18。

表8.15　没有改变的植物位置表

植物位置表
植物码
位置码

表8.16 删除了土质描述的植物表

植物表
植物码
植物名
土质分类

表8.17 新土地表

土地表
土质分类
土质描述

表8.18 没改变的位置表

位置表
位置码
位置名

全部的这些表都是第三范式了。第三范式避免了绝大部分的数据异常，对大部分表都已经足够了。作者建议在实现表之前，就把要操作的表变为第三范式，这样在大部分情况下都能达到本章开始列出的范式化目标。这之上的范式，比如Boyce-Codd范式和第四范式，对于商业应用来说基本无用。但任何熟练的数据库实践者都应该了解到还有例外，而且能在需要的时候达到更高级别的范式。

Boyce-Codd范式

E.F. Codd和R.F. Boyce，这两个数据库模型的开发鼻祖获得了赋予范式其名字的荣誉。E.F. Codd开发和扩展了关系模型，并在20世纪70年代发明了关系模型的范式化，而R.F. Boyce是结构化查询语言的发明人之一（叫SEQUEL）。

与某些资料说的相反，Boyce-Codd范式与第四范式是不同的。在定义之前，看一些数据异常的例子，它们已经在第三范式中列出了，而在转换为Boyce-Codd范式的过程中得到了解决（见表8.19所示）。

表8.19 包含数据学生、课程和导师关系的表

学生课程导师表
学生
课程
导师

对表8.19假设下面是正确的：

- 每个导师只有一门课

- 每门课可以有一个或多个导师
- 每个学生的每门课只有一个导师
- 每个学生可以上一门或多门课

哪一个作为键？没有一个单一的域足够用来惟一区分一个记录，因此需要使用两个域，使用哪两个？

学生和导师似乎是最好的选择，因为通过它们可以决定课程。或者也可以使用学生和课程，它们可以决定导师。现在，使用学生和课程作为键（见表8.20）。

表8.20 使用学生和课程作为键

学生课程导师表
学生
课程
导师

这张表是什么范式？因为它有键并且没有重复的组，它属于第一范式。它也属于第二范式，因为导师依赖于其他两个域（学生有很多课程，因此有很多导师，而课程有很多导师）。最后，它也属于第三范式，因为只有一个非键属性。

但还是有一些数据异常。看一下表8.21中的数据例子。

表8.21 更多的数据异常

学生	课程	导师
Conrad Pienaar	生物学	Nkosizana Asmal
Dingaan Fortune	数学	Kader Dlamini
Gerrie Jantjies	理学	Helen Ginwala
Mark Thobela	生物学	Nkosizana Asmal
Conrad Pienaar	理学	Peter Leon
Alicia Ncita	理学	Peter Leon
Quinton Andrews	数学	Kader Dlamini

Peter Leon教授理学这个事实的存储是冗余的，同样Kader Dlamini教授数学和Nkosizana Asmal教授生物也是如此。问题是导师决定了课程。或换一个方式，课程由导师决定。这张表属于第三范式是因为没有非键属性依赖于另一个非键属性。但是，一个键属性依赖于非键属性！也可以使用熟悉的方式删除这个域并把它和键一起放入另一张表（见表8.22和表8.23所示）。

表8.22 删除课程之后的学生导师表

学生导师表
学生
导师

删除课程域后，主键需要包括剩下的两个域来惟一地区分记录。

表8.23 导师课程表结果

导师课程表
导师
课程

尽管在原始表中已经选择course作为主键的一部分，因为导师决定了课程，因此在表中使用了它作为主键。可以看到，冗余的问题已经得到解决。

因此，如果表满足下面的条件，就属于Boyce-Codd范式：

- 属于第三范式。
- 每个决定因素都是候选键。

这听起来很吓人！对大多数新学数据库设计的人来说，这些都是新术语。但是，如果跟着这个例子，就会很清楚这些术语了：

- 决定因素是一个决定另一个属性值的属性。
- 候选键是键或选择键 （也就是，这个属性可以是那张表的键）之一。

导师不是候选键（不能单独的惟一区分记录），但它决定了课程，因此表不属于Boyce-Codd范式。

再看一下这个例子，如果选择学生和导师作为键会发生什么呢？显示在表8.24中。这次表是什么范式？

表8.24 使用学生和导师作为键

学生课程导师表
学生
导师
课程

因为有一个主键而且没有重复组，它还是第一范式。但是这一次，因为课程只决定于键的一部分：导师，因此它不是第二范式。删除课程和它的键，导师，就得到了显示在表8.25和表8.26的数据。

表8.25 删除课程

学生导师表
学生
导师

表8.26 使用课程创建新表

导师课程表
导师
课程

不管使用哪种方法，只要确保把表范式化为Boyce-Codd范式，都会得到两张同样的表。通常有可以选择为键的替代域时，选择哪一个是没有关系的，因为在范式化以后，会得到同样的结果。

第四范式

考虑一种即使表已经是Boyce-Codd范式也还会产生数据冗余的情况。还是考虑前面的student/instructor/course例子，但改变其中的一个假设。这次，一个学生的一门课程可以有多个导师（见表8.27）。

表8.27 每个课程有几个导师的学生课程导师数据

学生	课程	导师
Conrad Pienaar	生物	Nkosizana Asmal
Dingaan Fortune	数学	Kader Dlamini
Gerrie Jantjies	理学	Helen Ginwala
Mark Thobela	生物	Nkosizana Asmal
Conrad Pienaar	理学	Peter Leon
Alicia Ncita	理学	Peter Leon
Quinton Andrews	数学	Kader Dlamini
Dingaan Fortune	数学	Helen Ginwala

除了Helen Ginwala教授Gerrie Jantjies的理学以及Dingaan Fortune的数学，以及Helen Ginwala和Kader Dlamini都教授Dingaan Fortune的数学以外，与以前的数据是一样的。

惟一可能的键是三个属性的组合，如表8.28中所示。没有其他的组合可以惟一区分特定的记录。

表8.28 三个属性作为键

学生课程导师表
学生
导师
课程

但这还是有潜在的异常现象。Kader Dlamini教授数学的事实存储超过一次，同样的还有Dingaan Fortune学习数学的情况。真正的问题是存储的事实超过一种：除了学生－课程

关系，还有学生－导师关系。通过把数据分布在两张表中，可以避免这个问题。这显示在表8.29和表8.30中。

表8.29 为学生-导师关系创建表

学生导师表
学生
导师

表8.30 为学生-课程关系创建表

学生课程表
学生
课程

当有多个多值依赖时，这种情况就会存在。两个属性之间，如果对于每个属性，在第二个属性中都存在一个或多个关联的值时，多值依赖的情况就会存在。对于每个学生，都有多个课程值。这是第一个多值依赖。对每一个学生的值，都存在一个或多个关联的导师值，这是第二个多值依赖。

一张表，如果满足下面的标准，就是第四范式：

- 属于Boyce-Codd范式
- 不包含超过一个的多值依赖

第五范式及以上

还有只存在于学术兴趣上的更进一步的范式。用它们来解决的问题在实际中非常少见，因此作者不准备详细介绍。对于感兴趣的读者，下面的例子提供了一些感觉（见表8.31所示）。

表8.31 销售代表例子

销售代表	公司	产品
Felicia Powers	Exclusive	书籍
Afzal Ignesund	Wordsworth	杂志
Felicia Powers	Exclusive	杂志

因为检查哪个组合是有效的需要全部三个记录，通常把这个数据存储在一张表中。Afzal Ignesund在Wordsworth销售杂志，但并不一定有书。而Felicia Powers碰巧在Exclusive销售书和杂志。但是，再加一个条件：如果一个销售代表销售某种产品，并且是为特定的公司销售它，那他们就必须为那家公司销售那个产品。现在来看一个满足这个条件的大的数据集（见表8.32所示）。

表8.32 大的数据集

销售代表	公司	产品
Felicia Powers	Exclusive	书籍
Felicia Powers	Exclusive	杂志
Afzal Ignesund	Wordsworth	书籍
Felicia Powers	Wordsworth	书籍
Felicia Powers	Wordsworth	杂志

现在，根据这个额外的依赖关系，可以在不丢失事实的前提下，把表8.32进一步范式化为三张分开的表，这显示在表8.33、表8.34和表8.35中。

表8.33 创建一个带销售代表和产品的表

销售代表	产品
Felicia Powers	书籍
Felicia Powers	杂志
Afzal Ignesund	书籍

表8.34 创建一张带销售代表和公司的表

销售代表	公司
Felicia Powers	Exclusive
Felicia Powers	Wordsworth
Afzal Ignesund	Wordsworth

表8.35 创建一张带公司和产品的表

公司	产品
Exclusive	书
Exclusive	杂志
Wordsworth	书
Wordsworth	杂志

基本上，如果一张表不能用不同的键（大部分表显然都能够用同样的键组成更小的表）组成更小的表，这张表就属于第五范式。

第五范式之上，就进入了使人陶醉的一种理论上理想的域键范式的领域。它对于数据库设计者的实际用处与簿记员的无穷概念相似——比如，只存在于理论上，但并不能在实际中使用。即使是最糟糕的执行者也不希望用这样的簿记员。

对于对进一步探询这个学术的和高理论性的主题感兴趣的人，作者建议他阅读C.J. Date的“An Introduction to Database Systems”一书（Addison-Wesley, 1999）。

理解逆范式化

逆范式化就是把因为性能原因而范式化的转变逆转的过程。它是一个在专家之间激起争论的主题；有些人说花销太高而从不逆范式化，而另一些人却吹捧它的好处并例行公事地逆范式化。

对于逆范式化的支持者，其想法如下：随着向高范式演进，范式化创建了越来越多的表，但越多的表在取得数据时需要越多的连接，这会降低查询的速度。因为这个原因，为了提高某些查询的速度，可以不考虑数据的一致性，而返回到使用低范式的数据结构上。

作者建议一个实际的处理方法，考虑SQL的限制，特别是MySQL，但注意不要无限制地逆范式化。下面的小技巧会帮助读者做决定：

- 如果用范式化结构的效率可以接受，就不应该逆范式化。
- 如果效率不能接受，确保逆范式化能够使效率可接受。研究可替代的选择，比如更好的硬件，就可能能够避免逆范式化。数据结构改变后，在以后就很难恢复。
- 确保愿意用降低数据一致性来获得效率的提高。
- 考虑将来可能的方式，即应用可能对数据有不同的需求的情况。逆范式化在增强特定应用的效率的同时，也使数据结构依赖于应用，而理想的情况是应用独立于数据。

表8.36介绍了一个可能不是最适宜逆范式化的一般结构。能区分这张表属于哪个范式吗？

表8.36 客户表

客户表
ID
名
姓
地址行1
地址行2
区县
邮政编码

因为这张表有一个主键并且没有重复的组，所以表8.36一定属于第一范式。因为只有一个键，因此也属于第二范式，故也不能有部分依赖。属于第三范式吗？存在转换依赖吗？似乎有。邮政编码可能决定于区县属性。为了使其属于第三范式，应该取出邮政编码，把它放入一个以区县为键的分开的表中。在大部分情况下，作者都不建议这样做。尽管这张表不属于第三范式，但并不值得麻烦地分开这张表。表越多，需要做的连接就多，系统效率就会降下来。范式化的根本原因就是通过删除数据冗余来减少表的大小（这一般会加快速度）。但也需要看表是如何使用的。区县和邮政编码作为地址，一般总是会一起返回。在大部分情况下，删除区县/邮政编码组合省下的空间并不能改变额外的连接导致的系统速度降低。在某些情况下，这可能有用，比如需要用邮政编码或区县对成千的客户排序地址，数据的分布意味着对新的、小的表的查询返回结果会足够快。最后，有经验的数据库设计员不会刚性地

遵守这些步骤，因为他们明白如何使用数据。而这只有通过经验才能获得。范式化只是一个在大多数情况下帮助产生一个更有效率的结构的步骤而不是数据库设计的规则。

提示： 作者看到过一些可怕的数据库设计，基本上是因为没有范式化，而不是太范式化。如果没有把握的话，就范式化！

小结

数据范式是一个进行在表上的用来减少各种各样的数据异常的过程：

- 第一范式包含非重复的组并保证全部属性都依赖于主键。
- 第二范式不包含部分依赖。
- 第三范式不包含传递依赖。
- 出于实际考虑，第三范式通常就够了；实际上，超范式可能导致性能问题，数据库要进行太多的连接。
- Boyce-Codd范式保证每个决定因素都是候选键。
- 第四范式包含不超过一个的多值依赖。
- 第五范式保证表不能通过不同的键值来变为小表。
- 域键范式是一个理论上的想法，超出了本书的范围。

第9章 数据库设计

数据库之所以存在是因为需要把数据转换为信息。数据就是原始的没有经过处理的事实。信息是通过把数据加工成有用事物的过程来获得的。比如，电话本中数百万的名字和电话号码是数据。当房子着火时，消防部门的电话号码就是信息。

数据库是一个事实的大仓库，是用一种把数据更容易处理为信息的方式设计出来的。如果电话本的结构是一个很不方便的形式，比如名字和号码是按照号码发布的时间顺序排列的，则把数据转换为信息会很困难。如果不知道消防部门的电话号码是何时发布的，查找它可能会花数个小时。当找到号码的时候，房子可能已经变成了灰烬。因此，电话本设计成现在的样子是一件好事。

数据库更灵活；与电话本中相同的数据集可以通过MySQL按名字、电话号码、地址或时间顺序排序。但数据库当然更复杂，它包含了许多不同种类的信息。人、工作职位和公司产品都能组合起来提供复杂的信息。但这个复杂性也使数据库的设计更复杂了。糟糕的设计可能是查询很慢，甚至可能使某种信息不可能得到。本章介绍好的数据库设计。

本章的主要内容：

- 数据库生命周期
- 实体-关系模型
- 数据库设计中的一般错误
- 现实世界的例子：创建一个出版跟踪系统
- 事务的并发控制

数据库生命周期

同任何事情一样，数据库也有有限的生命期。它们产生于乐观的气氛中，在生命中得到名誉、财富和平静，或者是声名狼藉，然后再一次消失。即使是最成功的数据库也会在某个时候被另一个更灵活、更新式的结构所替代，开始新的生命。尽管数据库生命周期的具体定义可能不同，但一般都有六个阶段：

分析 分析阶段就是会见用户，测试已存在的系统来找出问题、可能的解决方法和限制。决定新系统的目标和范围。

设计 设计阶段就是从前面决定的需求创建一个概念性的设计，也创建为数据库实现而做的逻辑和物理设计。

实现 实现阶段就是安装数据库管理系统（DBMS），创建数据库，调入或引进数据。

测试 测试阶段对数据库进行测试和调整，一般与关联的应用结合起来完成。

运行 运行阶段是让数据库正常工作，产生信息给用户。

维护 维护阶段就是针对新需求或变化的运行条件（比如负载加重）对数据库进行变动。

数据库开发不依赖于系统开发，它一般是一个大的系统开发过程的一个部件。系统开发的阶段基本上是数据库生命周期阶段的镜像和超集。数据库设计就是处理系统存储数据的设计，系统设计同时与影响数据的过程关联。

阶段1：分析

存在的系统不能再使用了，是更换的时候了。可能是存在的纸系统产生了太多的错误，或者是老的基于平面文件的Perl脚本不能适应负载了。又或者是针对网站的已存在的新闻数据库在压力下挣扎着，因此需要升级。这就进入了对现存的系统进行评估的阶段。

根据项目大小，负责数据库实现和编码的设计者可以是单个人，也可以是一个分析团队。现在，术语设计者提出了各种可能的情况。

下面是分析阶段的步骤：

1. 分析机构
2. 定义任何问题、可能的解决问题的方法和限制
3. 定义目标
4. 在范围上达成一致

在检查系统的时候，设计者需要检查整体——不仅仅是硬件或已存在的表结构，还有要求重新设计的机构的整个状况。比如，带中央管理的大银行与任何人都可以在网站上发布新闻的非中心管理的媒体组织的结构和处理方式就不同。这似乎很琐碎，但理解正在构建的数据库针对的机构对设计一个好的数据库非常重要。同样的来自于银行和媒体机构的要求会导致不同的设计，因为机构不同。换句话说，即使是在情况相似的时候，针对银行构建的方案也不能想当然地作为媒体机构的实现。银行内中心控制的文化可能意味着张贴在银行网站上的新闻需要管理中心节制和审查，或者是需要设计者保留详细的某人在何时和为什么修改的审记踪迹。从简单的方面说，媒体机构可能更放任和高兴任何获授权的编辑修改新闻。理解了机构的文化，就可以帮助设计者提出合适的问题。银行可能不需要问审记踪迹，它可能本来就需要它；到了实现的时候，需要附上审记踪迹，而这需要更多的时间和资源。

一旦理解了机构结构，就可以针对存在的系统对使用者提一些问题，比如他们的问题是什么，需要什么，现在存在的限制，数据库系统的目标是什么，以及将存在什么限制等。需要对不同的角色使用者提问，因为每个角色都可能对数据库需要什么有自己的理解。比如，媒体机构的市场部可能需要跟踪人们在网站上从一篇新闻文章移到另一篇上去，但编辑部却可能需要某个时间某篇文章的阅读统计数据。还可能被告知未来可能的需求。可能编辑部希望扩展网站，以给他们提供网上文章相互链节的标签。记住这个将来的需求，可能使将来需要增加超链接时容易一些。

限制可以包括硬件（“必须使用已存在的数据库服务器，即AMD Duron 900MHz”）和人（“在任何时候都只有一个数据管理员”）。限制也包括对值的限制。比如，在大学数据库中，一个学生的成绩不可能高于100分，或剧院数据库中的三种座位分类是小，中，大。

除了最小的组织外，依赖于一级管理或者是单独一个人提供目标和解决当前存在的问题都还是不够的。高级的管理可能需要对数据库设计付费，低级别的需要使用它，而且它们的输入可能对成功的设计更为重要。

当然，尽管任何事情都可以有无穷的时间和金钱的支持，但这基本上从未出现过。确定和规划范围是项目的重要部分。如果预算是一个月的工作，而理想的方案需要3个月，设计者必须清楚这些限制并要同项目拥有者在实现的方面上达成一致意见。

阶段2：设计

设计阶段就是以前面阶段确认的需求为基础来开发新系统。另一个说法就是把对数据结构的商业理解转换为技术理解。问题“什么”（“需要什么数据？需要解决什么问题？”）被问题“怎么”（“数据怎么组织？数据怎么存取？”）所替代。

这个阶段包含三个部分：概念设计，逻辑设计和物理设计。有些方法学把逻辑设计阶段与其他的两个阶段合并。注意本章的目的不是对数据库设计方法学的定义的讨论（有整本书讨论这个的！），而是介绍这个主题的。

概念设计

概念设计的目的是基于前面确认的需求来构建一个与最终的物理模型更接近的概念模型。最有用和最普通的概念模型叫实体-关系模型。

实体和属性

实体基本就是想保存的信息如人、地方或事情。比如，图书馆系统可能有书、库和客户实体。学习区分什么应该是实体，什么应该是一组实体以及什么应该是实体的属性需要实践，但有一些经验性的规则。下面的问题可以帮助区分某个内容是否是实体：

- 数目上的变化是否不依赖于其他的实体？比如，人的高度可能不是实体，因为它在数目上的变化依赖于人。它不是基础，因此在这种情况下，它不能是实体。
- 能够重要到保证维护的努力吗？比如，客户对一个杂货店来说可能并不重要，因此在这种情况下它不是实体，但可能对一个音像店很重要，因此它是实体。
- 它自己的内容不能再分为子类吗？比如，租车机构对不同种类的交通工具有不同的标准和存放要求。交通工具可能不是一个实体，因为它可以分成小车和船，而这些是实体。
- 列举的是事物的种类而不是实例吗？视频游戏blow-em-up 6不是实体而是game实体的一个实例。
- 有许多相关的事实吗？如果只包含一个属性，它就不太可能是实体。比如，在某些情况下，城市可能是实体，但如果它只包含一个属性——城市名的话，它就更可能是另一个实体——比如客户——的属性。

下面的实体例子是关于大学的，所带的可能属性在括号中：

- 课程（名，代码，必选课程）
- 学生（姓，名，地址，年龄）
- 书（书名，ISBN，价格，库存数量）

实体的实例是那个实体的特殊表现。比如，学生Rudolf Sono是学生实体的一个实例。也可能有多个实例。如果只有一个实例，就要考虑实体是否被授权了。最顶级的一般不授权一个实体。比如，如果系统是为一个特定的大学开发的，大学就不是实体，因为整个系统

都是针对那个大学的。但是，如果系统是开发用来跟踪国内全部大学的注册的话，大学就是一个有效的实体。

关系

实体通过某种方式关联。比如，客户可以是图书馆的职员并能带出书。一本书可能只在一个特定的图书馆里。理解存储的数据是什么以及数据的关系如何，将对数据库的物理实现有很大的帮助。

有许多可能的关系：

强制性关系 对实体A的每个实例，一定存在实体B的一个或多个实例。但这并不意味着对实体B的每个实例都一定存在实体A的一个或多个实例。关系是可选性的还是强制性的只是从一个方向说的，因此在B到A的关系是强制的时，A到B的关系可以是可选的。

可选性关系 对于实体A的每个实例，可能存在也可能不存在实体B的实例。

1对1关系（1:1） 就是对于实体A的每个实例，存在实体B的一个实例，反过来也一样。如果关系是可选的，存在的可以是零个或一个实例，而如果关系是强制的，就存在一个而且只有一个关联的实例。

1对多关系 （1:M） 就是对于实体A的每个实例，存在实体B的多个实例，而对于实体B的每个实例，只存在实体A的一个实例。当然，这可以是可选的或强制性的关系。

多对多关系 （M:N） 就是对于实体A的每个实例，存在实体B的多个实例，反之，也如此。这也可以是可选的或强制性的关系。

有多种方法显示这些关系。图9.1显示了学生和课程实体。在这个例子中，每个学生必须登记至少一门课程，但却不一定必须有学生登记某课程。学生－课程的关系是强制的，而课程－学生的关系是可选的。

图9.1 多对多关系

图9.2显示的是invoice line和product实体。每个发票行至少应该有一个产品（但不能超过一个）；但是，每个产品可以出现在许多发票行上或者根本没有。发票行－产品关系是强制性的，而产品－发票行关系则是可选的。

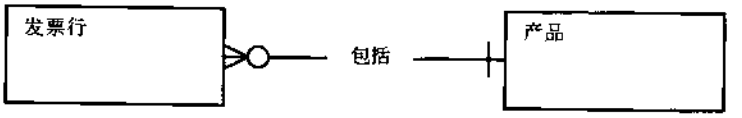

图9.2 一对多关系

图9.3显示丈夫和妻子实体。每个丈夫一定有一个而且只能有一个妻子，而每个妻子一定有而且只能有一个丈夫。两个关系都是强制性的。

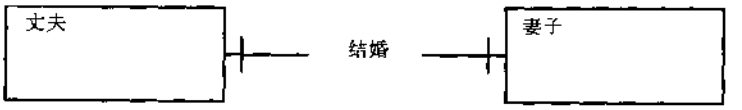

图9.3 一对一关系

实体也可以与自己有关系。这样的实体叫递归实体。考虑person实体：如果对哪些是兄弟的存储数据感兴趣，就可以有一个“是兄弟”关系。在这种情况下，关系是M:N的。

弱实体是没有另一个实体就不能存在的实体。比如，在学校，scholar实体与弱实体“父母/监护人”关联。没有学者，父母或监护人在这个系统中就不会存在。弱实体一般部分或全部地从关联实体中得到它们的主键。Parent/guardian可以把从学者表得到的主键作为自己的主键的部分（如果对每个scholar，只存储了一个parent/guardian，就可以是整个键）。

术语“连通性”是指关系的分类（1:1、1:M或M:N）。

术语“势”是指一个关系的特定的实体数。势限制列明了关联实体可能发生的最小和最大数。在丈夫和妻子的例子中，势限制是（1,1），而在学生选修了一到八门课程的例子中，势限制可以表示为（1,8）。

开发实体关系图

实体关系图显示了实体相互之间的关系。它由多种关系比如已经在图9.1、图9.2和图9.3所见的各种关系组成。一般来说，这些实体都会变成数据库的表。

开发关系图的第一步就是区分系统中的全部实体。在起始阶段，不需要区分属性，但如果设计者对其中的一些实体不太确定的话，属性可以帮着区分它们。一旦列出了实体，实体之间的关系就通过它们的类型而区分和模型化了：一对多、可选的，等等。有许多软件包可以用来辅助画实体关系图，但任何图形包都应该胜任这个工作。

一旦画完实体关系图，就提供给管理人员。非技术人员很容易理解实体关系图，特别是如果引导他们经过了整个过程的话。这也能帮助区分已经蔓延的错误。模型化的部分原因是因为理解模型比整页的文本更容易，因而更容易让管理人员检查，这样能够减少错误进入下个阶段的机会，错误在下一个阶段会更难改正。

提示： 认识到没有所谓的正确或错误的答案是很重要的。情况越复杂，能够工作的设计可能越多。尽管需要数据库设计技巧，但在过程开始之前，更需要有经验的设计人员一起讨论什么能够工作和后面阶段可能出现的问题。

一旦图被通过了，下阶段就是用两个一对多的关系替代多对多的关系。DBMS不能直接实现多对多的关系，因此需要被分解为两个小的关系。为了实现它，必须创建一个交集或实体的复合体。这个交集实体比普通的实体更不像“现实世界”，有时会很难命名。在这种情况下，可以根据相交的两个实体来命名。比如，可以把student和course之间的多对多关系叫student-course实体（见图9.4）。

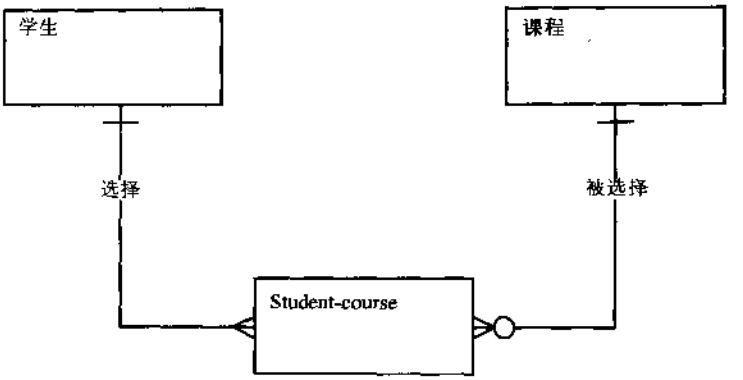

图9.4 创建student-course相交实体

即使实体是递归的也可以这样。有M:N关系“是兄弟”的人实体也需要交集实体。这个情况，可以给这个交集实体取一个好名： brother。这个实体包含两个域，对于兄弟关系的每个人——也就是第一个兄弟的主键和另一个兄弟的主键（见图9.5）。

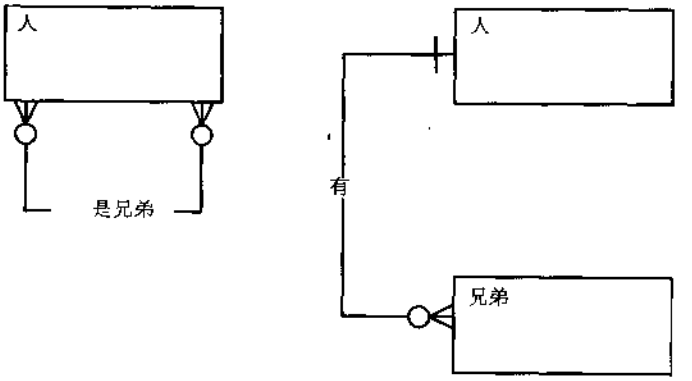

图9.5 创建兄弟交集实体

逻辑和物理设计

概念设计完成后，就应该进入逻辑和物理设计。一般的，在这个阶段依据需求和数据结构的复杂性选择DBMS。严格地说，逻辑设计和物理设计是两个不同的阶段，但经常合并为一个。它们是重叠的，因为大部分当前的DBMS（包括MySQL）在磁盘上以1:1作为基础来匹配逻辑记录与物理记录。

每个实体将变成数据库的一张表，而每个属性将变为这张表的一个域。如果DBMS支持外部键并且设计者决定实现它的话，就可以创建外部键。如果关系是强制性的，外部键就必须定义为NOT NULL，而如果关系是可选性的，外部键就可以容许null。比如，因为在前面的例子中发票行－产品的关系，在发票行表中，产品代码域是一个外部键。因为发票行必须包含产品，这个域必须定义为NOT NULL。当前，InnoDB支持外部键限制，而在版本4中，MyISAM表不支持外部键，但在版本4.1中，可能支持。支持外部键的DBMS在定义时需要使用ON DELETE CASCADE或ON DELETE RESTRICT从句。ON DELETE RESTRICT意思是除非与外部键关联的记录全部都删除了，记录才能被删除。在发票行－产品例子中，发票

行表中的ON DELETE RESTRICT意味着如果删除一个产品，除非与删除的产品关联的发票行都被删除了，否则删除不会发生。这避免了指向不存在的产品的发票行存在的可能。ON DELETE CASCADE除了更自动化（也更危险）外，也能达到同样的效果。如果外部键是用ON DELETE CASCADE声明的，产品被删除后，关联的发票行也都会自动被删除。ON UPDATE CASCADE与ON DELETE CASCADE相似，是指当主键更新时，指向主键的全部外部键也被更新。

在设计数据库的时候，范式化表是很重要的（见第8章“范式化数据库”）。这个过程可以帮助避免数据冗余和提高数据一致性。

数据库设计新手经常会犯一些普通的错误。如果已经仔细地区分了实体和属性，并且范式化了数据，就可能避免这些错误。但是，在设计者快速完成设计过程的时候，经常会导致大量的不相关的数据表。遵从下面的小技巧，将帮助读者避免许多经常犯的错误：

- 把不相关的数据放入不同的表中。习惯使用电子表单的人经常犯这个错误，因为他们习惯在一张二维的表中看到全部的数据。关系数据库是非常强大的，不要用这种方式使其“残废”。
- 不要存储可以计算的数据。假设读者喜欢三个数：A、B以及A与B（A * B）的积。不要存储积。它浪费空间，而且需要的话很容易计算。而且它会使数据库更难维护：如果改变了A，也得同时改变全部的积。为什么把数据库浪费在需要时可以自己计算的东西上呢？
- 设计的数据库满足已经分析的全部条件了吗？在创建实体关系图的冲击下，会很容易忽略某个条件。管理人员指出实体关系图中的错误比指出缺少的内容要更容易。事务逻辑与数据库逻辑同样重要，但更容易被忽略。比如，很容易指出如果没有关联的客户的话，不能有销售，但构建了如下的内容吗——如果被授权的客户没有推荐，另一客户的销售不能低于500美元？
- 即将变为域名的属性仔细选择过吗？域应该清楚地命名。比如，如果使用f1和f2代替surname和first_name，减少输入节省的时间将损失在查找域的正确拼写上，或者是开发人员认为f1是姓而f2是名的错误上。同样，要避免对不同的域使用同样的名。如果六个表都有同样的主键code，就会造成不必要的困难。应该使用尽可能多的描述术语，如sales_code或customer_code。
- 不要创建太多的关系。基本上系统中的表在某种程度上都能搭上关系，但并不需要这样做。比如，网球运动员属于运动俱乐部，运动俱乐部属于一个地区。因此，网球运动员也属于那个地区，但这个关系可以通过运动俱乐部得出，因此不需要增加一个外部键（除非对某些查询因为效率的原因需要）。范式化可以帮助避免这种问题（而且即使想为速度优化，也最好范式化然后有意识地逆范式化，也比不范式化要好）。
- 已经满足全部关系了吗？从实体关系图中得到的关系在表结构中都是作为普通域出现的吗？覆盖全部的关系了吗？全部的多对多关系都分成了带一个相交实体的一对多关系了吗？
- 列出了全部的限制了吗？限制包括性别只能是m或f，学龄儿童的年龄不可能超过20岁，E-mail地址必须有一个at符号（@）和至少一个句点（.）；对这些限制不要想当

然。在某个阶段系统就需要实现它们，或者是忘记它们，或者如果没有在之前列出的话，就必须回去收集更多的此类信息。

- 计划存储更多的数据吗？如果客户只是想简单地注册一个时事通讯，需要他们提供眼睛的颜色、最喜欢的鱼和他们祖父母的名字吗？有时管理人员想从客户处得到太多的信息。如果用户是设计机构之外的人，他们可能对设计过程没有发言权，但总是应该最先想到他们。再想一下获取全部数据的困难和花费的时间。如果电话接线员在销售之前需要记录下全部的这个信息，想想这会多慢。再想想数据对数据库速度的影响。大的表一般会减慢存取速度，而不必要的BLOB、TEXT和VARCHAR域会导致记录和表破碎。
- 应该分开的域组合了吗？把姓和名组合在一起是一个常见的错误。稍后就会意识到如果把他们存储为John Ellis和Alfred Ntombela的形式，在按字母顺序排序名字时会显示得多么麻烦。把不同的数据分散保存。
- 每张表都至少有一个主键吗？最好有一个省略主键的正当理由。如何快速区分一个惟一的记录？应该意识到索引会大大加速存取速度，而如果保存的数据量小的话，增加的负担是很小的。同样，最好为主键创建一个新域而不用已存在的域。在当前的数据集中，姓和名可能是惟一的，但它们不可能永远都是。创建一个系统定义的主键可以保证它总是惟一的。
- 还应该考虑其他的索引。在存取表的时候，什么域可能被使用？以后在测试系统的时候，总是可以创建更多的域，但在这个阶段如果需要就添加。
- 外部键存储得正确吗？在一个一对多的关系中，外部键出现在“多”表中，而关联的主键出现在“一”表中。混淆它们会导致错误。
- 保证引用一致性吗？外部键不应该与另一张表中不存在的主键关联。
- 已经覆盖了可能需要的全部字符集了吗？比如，德语字母中，有一个扩展字符集，如果数据库是针对德国用户的，就应该考虑到这点。同样，如果系统是国际化的，就应该考虑到日期和币种格式的问题。
- 安全性足够吗？记住应该赋予可能的最小的权限。如果没有必要，就不要让人查看那张表。让危险的用户查看数据，即使他们不能改动，也是他们进攻的第一步。

阶段3：实现

实现阶段就是在要求的硬件上安装DBMS，在硬件和软件平台上优化数据库并使其运行最好，创建数据库并调用数据。初始的数据可以是直接得到的新数据，也可以是从MySQL数据库或其他的DBMS引入的已存在的数据。在这个阶段需要建立数据库安全性并给各种用户满足要求的存取权限。最后，在这个阶段设置初始的备份计划。

下面是实现阶段的步骤：

1. 安装DBMS
2. 根据硬件、软件和使用条件调整设置变量
3. 创建数据库和表
4. 装载数据
5. 设置用户和安全权限

6. 实现备份计划

阶段4：测试

测试阶段就是测试数据的效率、安全性和一致性，一般和已开发的应用联合进行。在各种各样的负载条件下测试效率，以观察数据库处理多并发连接或高容量更新和读取时的效率。结果产生得是否足够快？比如，因为低估了更新的影响，针对MyISAM表设计的应用证明太慢。解决措施就是把表类型改变为InnoDB。

因为应用可能有逻辑缺陷从而导致事务丢失或其他的不准确，因此要测试数据一致性。安全性也需要测试，以保证用户只能存取和改变他们自己的数据。

可能不得不改变逻辑或物理设计。因为效率的原因（见第8章），可能需要新索引（通过测试者仔细使用第4章介绍的“索引和查询优化”中MySQL的EXPLAIN语句来发现）或者某张表需要逆范式化。

测试和性能调试是交互式的，进行的测试越多，实现改变的可能越多。

下面是测试阶段的步骤：

1. 测试性能
2. 测试安全性
3. 测试数据一致性
4. 根据测试结果仔细调整参数或修改逻辑或物理设计

阶段5：运行

当测试完成和数据库准备好日常使用后，就开始了运行阶段。系统的使用者开始操作系统，调用数据，读取结果，等等。这不可避免地会产生问题。在这个阶段，设计者需要仔细地管理数据库的范围，因为使用者希望他们的要求都得到考虑。糟糕的数据库设计者会发现项目已经超出了他们开始时估计的范围，而如果范围没有定义清楚并且不能达成一致，情况就会变得更糟糕。项目拥有者的需求没有满足的话，他们就会担心，而数据库设计者就会感到很劳累和失望。即使范围管理得很好，也还会有新需求。这样就会进入下一个阶段。

有各种策略来实现一个展示。低调处理——在开始阶段保持相对少的用户——就能很好地工作，这能使修改错误容易一些。高调展示一般会以煽动管理人员发怒而结束，因为即使是经过了最好的测试人员，用户也还是会发现没有预见的错误，而这些错误最好远离现场。发布也可以采取分布式的方式，就是选择分支机构或办公室，当已经证明系统稳定后，再发布到剩余的分支中去。

下面是运行阶段的步骤：

1. 把数据库的运行交给用户
2. 对用户发现的问题做最后的修改

阶段6：维护

数据库维护阶段就是合并一般的维护，比如维护索引、优化表、增加和删除用户、改变口令，以及备份和系统失败时的备份恢复（关于维护的更多信息，见第10章“基本管理”）。新需求也开始要求处理了，而这可能导致创建新域，或者是新表。

随着系统和机构的变化，现存的数据库可能变得越来越不能充分满足机构的需要了。比如，媒体机构可能与其他国家的媒体机构合并，这需要许多数据源的集成，或者是规模和人员的急剧扩大（或减少）。最后，都会有一个时间，不管是完成后10个月还是10年，数据库系统都会被替换。维护现存数据库需要的资源越来越多，创建新设计的努力就要和当前的维护努力匹配。到了这点，数据库就走到了生命周期的终点，新项目就开始了分析阶段。

下面是维护阶段的步骤：

1. 维护索引（比如，使用MySQL的ANALYZE）
2. 维护表（MySQL的OPTIMIZE）
3. 维护用户（MySQL的GRANT和REVOKE）
4. 改变口令
5. 备份
6. 恢复备份
7. 根据新需求改变设计

现实世界的例子：创建一个出版跟踪系统

现在将一步一步地开始一个数据库设计系统。Poet's Circle是一个出版诗和诗选的发行商。它想开发一个新系统来跟踪诗人、诗歌、诗选以及销售。下面的小节显示的是从开始的分析到最后的工作数据库。

Poet's Circle数据库阶段1：分析

下面的信息是在Poet's Circle同各种管理人员谈话时收集到的：他们想开发一个系统来跟踪记录过的诗人、他们写的诗歌和他们发表作品的出版物名，以及这些出版物对客户的销售情况。

设计者通过问各种问题来得到更多的信息，比如，“就系统而言，诗人是什么？Poet's Circle跟踪没有写过或发表过诗的诗人吗？出现相关的诗之前出版物也记录吗？出版物由一首诗还是多首诗构成？潜在的客户细节记录吗？”下面总结了不同问题的答案：

- Poet's Circle是一个基于在其网站上活动的诗人圈子来选择其出版物的发行人。如果有足够的人要求诗发表，Poet's Circle就会这样做。
- 诗人就是想成为诗人的任何人，不一定在系统中已经有诗或曾经写过诗。
- 诗可以通过Web界面、E-mail或者是纸的方式提交。
- 全部获得的诗都是由一个关联的诗人写的，其细节已存储在系统中。如果没有诗人的细节，就不能提交和储存诗。
- 出版物可以是一首诗、诗集或者诗文学批判集。
- 客户可以通过Web界面登录并可以实时 订购出版物或表达希望收到新出版物的兴趣，以便以后购买。
- 出版物只销售给细节已经存储在系统中的客户，而不进行匿名销售。
- 单个销售可以只针对一个出版物，但可以同时购买多种出版物。如果与这次销售相关的客户超过一个， Poet's Circle把其作为多个销售。每个客户都有自己的销售。

- 并不是全部出版物都销售——可能有一些是特别版本，而另一些则可能永远不会销售！

Poet's Circle数据库阶段2：设计

基于上面的信息，就可以开始逻辑设计，并且可以区分出初始实体了：

- Poet
- Poem
- Publication
- Sale
- Customer

Poet's Circle不是实体，甚至也不是publisher实体的实例。只有在系统是为许多出版商开发的时候，publisher才是一个有效的实体。

website和poetry community也不是实体。只有一个website，而且，无论如何，website只是一个处理存储在数据库中数据的方法。针对这个系统，只有一个诗歌社区，而且不会想存储太多的内容。

接下来，需要确定这些实体之间的关系。下面的内容需要区分：

- 一个诗人可以写许多诗。分析已经证明了即使没有关联的诗，诗人也可以存储在系统中。诗可以在以后获得，甚至诗人可以是一个潜在的诗人。当然，很多诗人都写过诗，尽管诗可能是不止一个诗人写的。
- 出版物可以包含许多诗（诗集）或仅仅一首。它也可以不包含诗（比如诗歌批判）。诗可以出现也可以不出现在出版物中。
- 销售应该至少是一份出版物，也可以是许多出版物。出版物可以有也可以没有任何一次销售。
- 客户可以有多个销售也可以根本就没有。一次销售只针对一个也只能是针对一个客户。

可以区分下面的属性：

- Poet: first name, surname, address, telephone number
- Poem: poem title, poem contents
- Publication: title, price
- Sales: date, amount
- Customer: first name, surname, address, telephone number

基于这些实体和关系，就可以构建图9.6中显示的实体关系图。

如图9.6显示的，有两个多对多关系。在DBMS中实现之前，需要把它们转换为一对多关系，而结果为带有交叉实体的poem-publication和sale-publication，如图9.7所示。

现在，开始逻辑和物理设计，需要增加能在实体之间创建的关系的属性，并指定主键。通常最好的方式是创建新的、惟一的主键。

表9.1到表9.7显示的是从每个实体创建的表的结构。

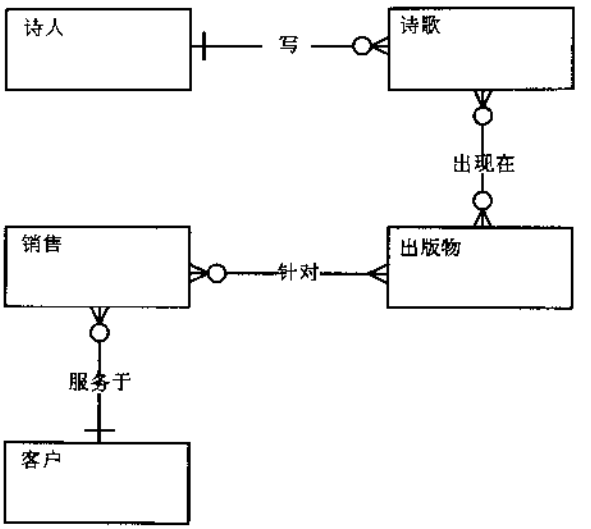

图9.6 Poet's Circle实体关系图

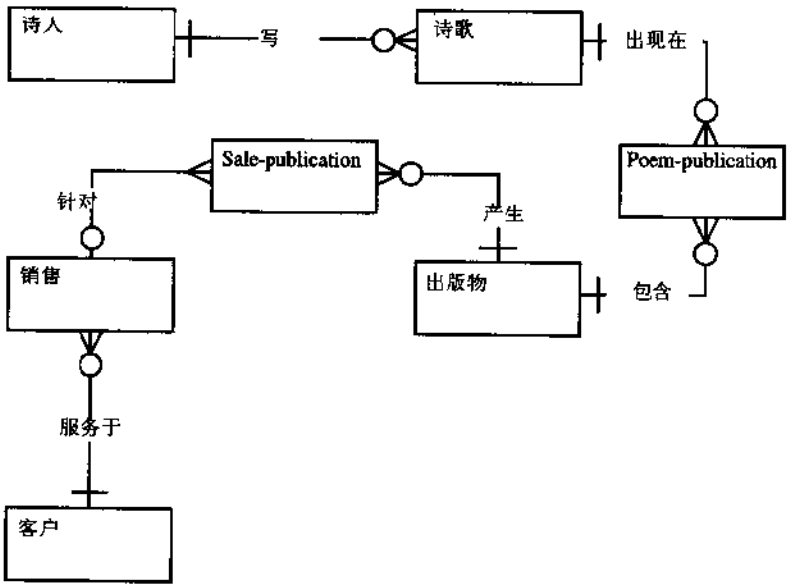

图9.7 删除了多对多关系的Poet's Circle实体关系图

表9.1 诗人表

域	定义
诗人代码	主键，整数
名	字符（30）
姓	字符（40）
地址	字符（100）
邮政编码	字符（20）
电话号码	字符（30）

表9.2 诗歌表

域	定义
诗歌代码	主键，整数
诗歌标题	字符（50）
诗歌内容	文本
诗人代码	外部键，整数

表9.3 Poem-Publication表

域	定义
诗歌代码	组合主键，外部键，整数
出版物代码	组合主键，外部键，整数

表9.4 出版物表

域	定义
出版物代码	主键，整数
标题	字符（100）
价格	数字（5.2）

表9.5 Sale-Publication表

域	定义
销售代码	组合主键，外部键，整数
出版物代码	组合主键，外部键，整数

表9.6 销售表

域	定义
销售代码	主键，整数
日期	日期
数量	数字（10.2）
客户代码	外部键，整数

表9.7 客户表

域	定义
客户代码	主键，整数
名	字符（30）
姓	字符（40）
地址	字符（100）
邮政编码	字符（20）
电话号码	字符（30）

MySQL对这个设计没有任何问题，因此选择它作为DBMS，同时还选择了现有的硬件和操作系统平台。

Poet's Circle数据库阶段3：实现

设计完成后，接着就应该安装MySQL并执行CREATE语句了，如下：

```
mysql> CREATE TABLE poet (poet_code INT NOT NULL, first_name VARCHAR(30),
 surname VARCHAR(40), address VARCHAR(100), postcode VARCHAR(20),
 telephone_number VARCHAR(30), PRIMARY KEY(poet_code));
Query OK, 0 rows affected (0.02 sec)
mysql> CREATE TABLE poem(poem_code INT NOT NULL, title VARCHAR(50),
 contents TEXT, poet_code INT NOT NULL, PRIMARY KEY(poem_code),
 INDEX(poet_code), FOREIGN KEY(poet_code) REFERENCES poem(poet_code))
 type=InnoDB;
Query OK, 0 rows affected (0.00 sec)
mysql> CREATE TABLE publication(publication_code INT NOT NULL,
 title VARCHAR(100),price MEDIUMINT UNSIGNED,
 PRIMARY KEY(publication_code)) type=InnoDB;
Query OK, 0 rows affected (0.05 sec)
mysql> CREATE TABLE poem_publication(poem_code INT NOT NULL,
 publication_code INT NOT NULL, PRIMARY KEY(poem_code,
 publication_code), INDEX(poem_code), INDEX(publication_code),
 FOREIGN KEY(poem_code) REFERENCES poem(poem_code),
 FOREIGN KEY(publication_code) REFERENCES
 publication(publication_code)) TYPE=InnoDB;
Query OK, 0 rows affected (0.09 sec)
mysql> CREATE TABLE sales_publication(sales_code INT NOT NULL,
 publication_code INT NOT NULL,PRIMARY KEY(sales_code,
 publication_code)) TYPE =InnoDB;
Query OK, 0 rows affected (0.07 sec)
mysql> CREATE TABLE customer(customer_code INT NOT NULL, first_name
 VARCHAR(30), surname VARCHAR(40), address VARCHAR(100), postcode
 VARCHAR(20), telephone_number VARCHAR(30), PRIMARY KEY(customer_code))
 TYPE=InnoDB;
Query OK, 0 rows affected (0.06 sec)
mysql> CREATE TABLE sale(sale_code INT NOT NULL, sale_date DATE,
 amount INT UNSIGNED, customer_code INT NOT NULL, PRIMARY
 KEY(sale_code), INDEX(customer_code), FOREIGN KEY(customer_code)
 REFERENCES customer(customer_code)) TYPE = InnoDB;
Query OK, 0 rows affected (0.08 sec)
```

Poet's Circle数据库阶段4～6：测试、运行和维护

一旦数据库准备好了，应用程序也开发出来了，就需要进行测试了。数据库生命周期中的其他阶段可以合理地独立于系统的开发过程之外，测试阶段的部分工作就是测试系统的各个部件协调运行的情况。

负载测试可能表明MySQL还未能准备好处理希望的600个并发连接的情况，配置文件还需要改变。其他的测试可能表明，在某种情况下，因为锁的机制没有用统一的方式实现，导致出现重复键值的错误，并且应用不能正确处理锁。应用也需要修改。备份也需要测试，以便能够在最小的宕机时间内顺利地从备份中恢复正常。

警告：测试是最容易被忽视的重要阶段。不能正确地考虑测试的设计者或经理是不合格的。无论系统如何小，都要确保分配时间进行彻底的测试和修改不可避免的问题。

一旦测试完成，系统就可以发布了。先进行小范围的发布，只让一些经过选择的诗人访问网站并上载他们的诗。假设发现了其他的问题：无名的浏览器有不兼容的问题，导致提交垃圾诗。严格地说，这不属于数据库程序员解决的问题，但这是一个一旦系统的各部分都工作得很好时测试反映的一个问题。坚持要求用户必须使用能正确处理开发页面的浏览器，不符合这些标准的浏览器禁止上载。

很快，系统就完全发布了。而维护是一个永远不会结束的任务，随着进行大量的更新和删除，数据库会变得零碎。管理人员必须经常执行OPTIMIZE语句，并且当然，不可避免的磁盘失败导致整夜的恢复和对使用mysqldump的感谢。

事务的一致性控制

数据库请求是按照线性，即一个接一个的方式发生的。当许多用户访问一个数据库，或者是一个用户有一个关联的请求集需要运行时，保证结果的一致性就很重要了。为了做到这点，就使用事务——一组数据库请求作为一个整体来处理。换句话说，它们是工作的逻辑单元。

为了保证数据一致性，事务需要满足四个条件：原子性、一致性、分离性和持久性（也即ACID）。

原子性

原子性意味着整个事务必须完成。如果不是这样，就放弃整个事务。这保证了数据库永远不会出现会导致糟糕的数据一致性的部分完成的事务。比如，从一个银行账户取钱，第二个请求失败了，系统就不能把钱存入另一个银行账户，这时，两个请求都必须失败。钱不能简单地丢掉或从一个账户取出了却没有存入另一个账户中。

一致性

一致性是指满足某种条件时，数据所处的状态。比如，每张发票必须与customer表中的一个客户关联，这是一个规则。如果关联的客户是在事务的后阶段才加入的话，插入的发票就没有关联的客户，在这种事务过程中，就可以打破这个规则。这种临时的冲突在事务之外是不可见的，而且会在事务完成的时候解决。

分离性

分离性是指在一个事务处理过程中使用的数据只有在第一个事务完成后才可以被第二个事务使用。比如，两个人存入100美元到一个900美元的账户上，第一个事务必须是把100美元加到900美元上，而第二个必须是把100美元加到1000美元上。如果第二个事务在第一个事务完成前读出了900美元，两个事务都似乎成功了，但丢失了100美元。第二个事务必须等到它能单独存取数据。

持久性

持久性是指事务中的数据一旦提交，即使之后系统失败了，结果也会保持。事务进行的时候，影响是非持久的。如果数据库崩溃了，备份总是能够恢复到事务开始之前的一致状态。事务完成的任何事都不能被改变。

小结

好的数据库设计保证了具有更长的生命和更有效率的数据库系统。通过花费时间仔细地设计，设计者能够避免绝大多数困扰着许多已有的数据库的一般的重复性错误。

数据库生命周期（DBLC）可以用许多方式来定义，但得到的是相同的主要步骤。首先，分析阶段收集信息，检查现存系统的问题，可能的解决方式，等等。然后，设计阶段，先是针对管理人员的概念性的内容，然后是针对实现的逻辑和物理内容来仔细地设计新系统。接着，在测试阶段出现问题之前，实现阶段实际地发布数据库。然后，测试阶段成功后，系统就投入日常的运行使用中。维护阶段几乎是接着就开始了。随着到来的变化的请求，需要进行常规的优化和备份。最后，一旦维护变得太密集，就要开始一个新的数据库周期来代替老化的系统。

事务保证数据库在存在的过程中都保持状态一致。为了做到这一点，有四个原则。原子性表示在一个事务中，请求作为一个整体而成功或失败。一致性保证数据库在事务之间返回的状态是一致的。分离性保证了在影响同样数据的事务开始处理之前，前一个事务的请求一定要完成。而持久性保证即使在失败的情况下，数据库也能保持一致。

第三部分　MySQL管理

第10章　基 本 管 理

虽然MySQL很容易维护和管理，但是它却不能自主完成这些工作。本章将讲述基本的管理工作，其中一些内容将在后续的章节中进行更细致的描述。读者将学到如何启动或关闭服务器，包括手工操作和自动操作。读者还将了解到一些MySQL管理员所必需的工具，并学习如何使用记录文件和如何配置MySQL。

本章的主要内容：

- MySQL实用工具
- 启动和关闭MySQL
- 在开机后自动启动mysqld
- 配置MySQL
- 日志记录
- 循环日志
- 优化、检查、分析和修复表

成为MySQL的管理员

要想成为一名管理员，需要对MySQL如何工作有更多的了解，而不仅是进行查询。读者需要熟悉MySQL产品所提供的一些功能，以及MySQL是如何安装的。管理员的主要工作包括下列一些内容：

mysqladmin　它可能是最有用的管理工具。读者可以利用它建立和删除数据库、停止服务器的运行、查看服务器的参数、查看并关闭MySQL进程、设置密码和刷新日志文件！

mysqldump　建立SQL数据库的备份。

mysqld　实际上它并不是MySQL工具，而是MySQL服务器。会有一些类似于mysqld variables的术语，它们实际上就是服务器参数。

mysqlimport　将文本文件引入数据库表。

mysqlcheck　检查、分析并修复数据库表。

mysqlhotcopy　它是Perl脚本文件，可以用来快速地备份数据库表。

myisampack 压缩MyISAM表。

对于一个新的系统，最基本的问题是：数据存在哪儿？

在UNIX中默认的位置总是/usr/local/var，在Windows中是C:\mysql\data。当编辑MySQL时，可以选择数据目录，在配置文件中指定新的数据目录可以设置不同的位置。本章稍后的章节“配置MySQL”将帮助读者了解更详细的配置文件内容，但现在，知道它在UNIX中称做my.cnf，在Windows中是my.ini就够了，并且其中包含MySQL配置数据。如果希望强制MySQL使用新的位置，相应的输入行类似下面这一行：

```
datadir = C:/mysqldata
```

为了找到当前安装的数据目录，读者可以利用mysqladmin的参数选项。对于新手来说，下面的输出样本有些吓人。它有许多许多参数（并且每一个新的MySQL更新似乎都会增加更多的内容），按照字母顺序排列：

```
% mysqladmin -uroot -pg00r002b variables;
+-----------------------+-----------------------------------------+
| Variable_name         | Value                                   |
+-----------------------+-----------------------------------------+
| back_log              | 50                                      |
| basedir               | /usr/local/mysql-max-4.0.1-alpha-pc-    |
|                       |   linux-gnu-i686/                       |
| bdb_cache_size        | 8388600                                 |
| bdb_log_buffer_size   | 32768                                   |
| bdb_home              | /usr/local/mysql/data/                  |
| bdb_max_lock          | 10000                                   |
| bdb_logdir            |                                         |
| bdb_shared_data       | OFF                                     |
| bdb_tmpdir            | /tmp/                                   |
| bdb_version           | Sleepycat Software: Berkeley DB 3.2.9a: |
|                       |  (December 23, 2001)                    |
| binlog_cache_size     | 32768                                   |
| character_set         | latin1                                  |
| character_sets        | latin1 big5 czech euc_kr gb2312 gbk     |
|                       |  latin1_de sjis tis620 ujis dec8 dos    |
|                       |  german1 hp8 koi8_ru latin2 swe7 usa7   |
|                       |  cp1251 danish hebrew win1251 estonia   |
|                       |  hungarian koi8_ukr win1251ukr greek    |
|                       |  win1250 croat cp1257 latin5            |
| concurrent_insert     | ON                                      |
| connect_timeout       | 5                                       |
| datadir               | /usr/local/mysql/data/                  |
```

读者还可以利用一些工具对感兴趣的参数进行筛选，如下面所示，首先在UNIX中可以使用grep，并且在Windows中可以使用find：

```
% mysqladmin -uroot -pg00r002b variables | grep 'datadir'
| datadir                 | /usr/local/mysql/data/                    |
```

或

```
C:\mysql\bin>mysqladmin variables | find "datadir"
| datadir                 | C:\mysql\data\                            |
```

在这个例子中，在大多数安装的UNIX系统中，默认的数据目录是/usr/local/mysql/data，在Windows系统中是c:\mysql\data。

数据目录通常包含日志文件（尽管我们可以改变它们的位置，但是默认配置下它们安装在这里），以及实际数据。在这个MyISAM表的例子中，每个数据库都有它自己的目录，并且在这个目录中，每个表都有三个相关的文件：一个数据文件.MYD、一个索引文件.MYI和一个目标文件.frm。BDB表也存放在相同的目录中，但是它们包括一个.db文件和一个.frm目标文件。在数据库目录中，InnoDB表有其自己的.frm目标文件，但是实际的数据存放在上一级目录中，与数据库属于同一级目录。

MySQL客户端可以通过三种方法访问服务器：

UNIX sockets（UNIX套接字） 当UNIX机器与同一台机器上的服务器进行连接时，使用这些套接字。除非特别制定，套接字文件放在默认的位置（通常是/tmp/mysql.sock或/var/lib/mysql.sock）。

Named pipes（命名管道） 在Windows NT/2000/XP机器上使用这种方法，这里选项--enable-named-pipe与可执行的、准许命名管道的命令一起使用（mysqld-max-nt或mysqld-nt）。

TCP/IP through a port（通过端口建立的TCP/IP会话） 这是最慢的方法，但它是连接到运行在Windows 95/98/Me或远程UNIX机器的惟一方法。

启动和关闭MySQL

MySQL可以运行在各种操作系统上，但是过程各有不同。下面的章节分成Windows和UNIX部分进行论述。

在UNIX中的启动和关闭

启动MySQL的最基本的方法是直接运行mysqld。然而大多数版本都带有一个被称为mysqld_safe的启动脚本（较老的MySQL版本称之为safe_mysqld），在手动启动MySQL时应该使用它。这个脚本具有一些非常安全的特点，如对错误进行日志记录和在错误发生时自动重启动服务器。为了启动MySQL，可以以root的身份登录，并且在MySQL所在的目录（通常是/usr/local/mysql）中运行下面的命令：

```
% bin/mysqld_safe  --user=mysql &
```

需要注意的是，用户被设置成mysql。以mysql用户运行MySQL可以避免许可权和安全问题。

警告： 如果读者不熟悉系统，特别是没有安装过它，那么在试图以这种方法启动服务器之前要向系统管理员说明。启动服务器的方法有很多种，以错误的方法启动可能会出问题！大多数实际运行的系统在系统启动后使用脚本来自动启动MySQL，并且如果已经有脚本，那么读者应该使用它。

一些产品带有脚本文件mysql.server，它甚至可能已经自动安装了（有时只是改名为mysql）。通常它所在的目录中，进程可以在系统启动后自动启动。如果是这样，那么你就应该使用这个脚本来启动。mysql.server具有启动（start）和关闭（stop）选项。下面是一个在Red Hat Linux上使用的通用启动方法：

```
% /etc/rc.d/init.d/mysql start
% Starting mysqld daemon with databases from /usr/local/mysql/data
```

在FreeBSD中，文件可能在目录/usr/local/etc/rc.d中，这时读者应该使用下面的命令：

```
% /usr/local/etc/rc.d/mysql.sh start
```

若要关闭，可以使用命令mysqladmin：

```
% mysqladmin shutdown -uroot -p
Enter password:
020706 16:56:02  mysqld ended
```

或者是启动脚本中等价的选项，如这里的：

```
% /etc/rc.d/init.d/mysql stop
Killing mysqld with pid 2985
Wait for mysqld to exit\c
.\c
.\c
.\c
.\c
.\c
.\c
020706 17:07:49  mysqld ended
```

在系统启动后自动开启MySQL

在实际运行的系统中，最棒的管理构想差不多要算是在系统启动的同时运行MySQL。如果还没有做到这一点，那么读者需要知道自己的UNIX版本如何在系统启动或关闭时开启和停止进程。不同的系统间有着显著的不同。如果读者尚不清楚这些，特别是没有亲自安装所持有版本的MySQL，那么最好是通知系统管理员。他们对系统细节的了解要比本书中所描述更准确。

这里有一个Red Hat Linux的例子：

```
% cp /usr/share/mysql/mysql.server /etc/rc.d/init.d
```

这条命令将脚本文件mysql.server拷贝到初始化目录。

下面两条命令确保了在系统启动并进入多用户模式（运行级3）时启动MySQL，在系统关闭时（运行级0）关闭MySQL。它们在适当的运行级中与脚本文件mysql.server建立一个链接：

```
% ln -s /etc/rc.d/init.d/mysql.server /etc/rc.d/rc3.d/S99mysql
% ln -s /etc/rc.d/init.d/mysql.server /etc/rc.d/rc0.d/S01mysql
```

下面这个例子运行于最近版本的FreeBSD，这时启动脚本只需要被拷贝到目录rc.d中并赋予.sh的扩展名：

```
% cp /usr/local/mysql/support-files/mysql.server /usr/local/etc/rc.d/mysql.sh
```

读者需要确保自己的脚本文件是可执行的，并且不可以被非授权的人访问，像下面这样：

```
% chmod 700 /usr/local/etc/rc.d/mysql.sh
```

一些较老的系统可能使用/etc/rc.local来启动脚本文件，这时读者应该向文件中加入一些与下面例子类似的内容：

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

在UNIX中启动MySQL时需避免的常见问题

问题最常见的原因是许可权。如果不是以root的身份登录并启动MySQL，那么你会碰到下面的问题：

```
% /usr/local/mysql/bin/mysqld_safe --user=mysql &
% The file /usr/local/mysql/libexec/mysqld doesn't exist or is not executable
Please do a cd to the mysql installation directory and restart
this script from there as follows:
./bin/mysqld_safe.
```

为了解决这种问题，需要以root身份登录：

```
% su
Password:
% /usr/local/mysql/bin/mysqld_safe --user=mysql &
[1] 24756
% Starting mysqld daemon with databases from
/usr/local/mysql-max-4.0.2-alpha-unknown-freebsdelf4.6-i386/data
```

至于其他问题，MySQL错误日志可以为读者提供一些帮助。读者可以参见本章稍后“错误日志”一节。

在Windows中启动和关闭MySQL

Windows版本中有许多可执行文件（见表10.1）。可以根据使用MySQL的情况来选择所需的内容。

表10.1 MySQL可执行文件

可执行文件	描述
mysqld	支持调试、自动内存分配检查、事务表（InnoDB和BDB）和符号链接的二进制文件
mysqld-opt	不支持事务表（InnoDB或BDB）的优化二进制文件
mysqld-nt	支持命名管道（在NT/2000/XP使用）的优化二进制文件。此文件可以在95/98/Me上运行，但由于这些系统不支持，所以不能生成命名管道
mysqld-max	支持事务表（InnoDB和BDB）和符号链接的优化二进制文件
mysqld-max-nt	支持事务表（InnoDB和BDB）、符号链接和运行在NT/2000/XP上的命名管道的优化二进制文件

向下面的例子这样，简单地执行所要使用的可执行文件，可以启动MySQL：

```
C:\> c:\mysql\bin\mysqld-max
020706 18:53:45  InnoDB: Started
```

如果目录不同，用安装MySQL时的目的目录代替\mysql\bin\。很多Windows用户喜欢使用下面的方式：

```
C:\> c:\progra~1\mysql\bin\mysqld-max
```

读者还可以使用Windows版本中的winmysqladmin工具来启动MySQL。

说明： 如果一直在使用Windows 95，需要确信安装了Winsock 2。较老的Windows 95不带Winsock 2，并且MySQL无法运行。读者可以从www.microsoft.com下载。

自动启动MySQL

在Windows 95/98/Me中，可以在StartUp文件夹中建立文件winmysqladmin的快捷方式。这个文件与其他可执行文件存放在相同的位置，换句话说，默认的位置是c:\mysql\bin。

确保文件my.ini包含要用的可执行文件。你可能手动运行mysqld-max，然后要利用winmysqladmin自动启动MySQL。然而，举例来说，如果文件包含下列内容：

```
[WinMySQLAdmin]
Server=C:/PROGRAM FILES/MYSQL/bin/mysqld-opt.exe
```

那么就不能使用所期望的事务能力。可以手动编辑my.ini文件，或使用命令winmysladmin进行修改，即选择my.ini Setup并改变文件mysqld（见图10.1）。

在NT/2000/XP中，像下面这样将MySQL作为服务进行安装：

```
C:\> c:\mysql\bin\mysqld-max-nt -install
```

如果不希望自动启动MySQL，但仍然希望它作为服务使用，那么运行相同的命令，并使用选项manual：

```
C:\mysql\bin> mysqld-max-nt --install-manual
```

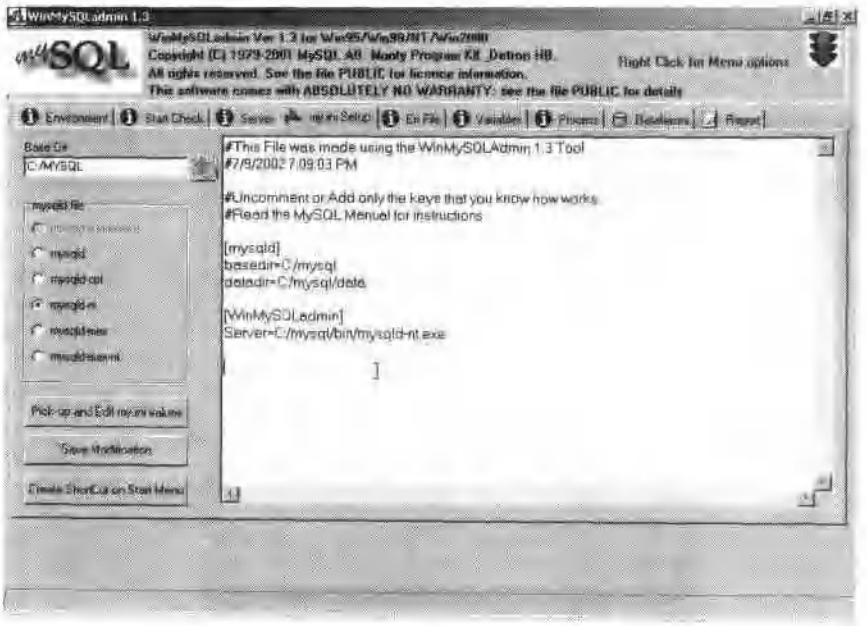

图10.1 使用winmysqladmin对配置文件my.ini进行更新

随后操作者可按下面的内容启动MySQL服务：

```
C:\> net start mysql
The MySql service is starting.
The MySql service was started successfully.
```

停止MySQL通常可使用命令mysqladmin shutdown或按照下面的操作：

```
C:\> net stop mysql
The MySql service is stopping.....
The MySql service was stopped successfully.
```

若要将其作为服务删除，可以运行命令mysqld，并使用参数remove，如下所示：

```
C:\> c:\mysql\bin\mysqld-max-nt -remove
```

操作者还可以使用控制面板（Control Panel）中的服务（Services）功能，并点击启动（Start）或停止（Stop）（见图10.2所示）。

避免在Windows中启动MySQL时常见的问题

在Windows中启动MySQL时，通常碰到的问题发生在MySQL被安装到了非默认的目录中（如c:\Program Files\MySQL\bin）。有时安装位置没有正确地出现在配置文件my.ini中。例如，如果已经将MySQL安装到目录中，那么my.ini包括的内容应该与下面内容相似：

```
[mysqld]
basedir=C:/Program Files/mysql
datadir=C:/Program Files/data
```

```
[WinMySQLAdmin]
Server=C:/Program Files/mysql/bin/mysqld-max-nt.exe
```

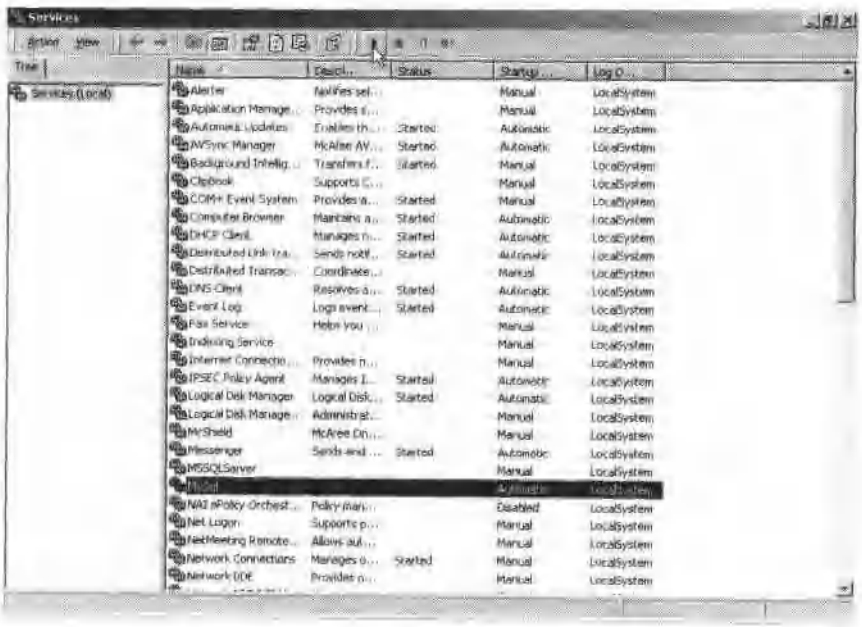

图10.2 在Windows 2000中作为服务启动MySQL

说明： 在选项文件中，Windows路径名由斜线（/）确定，而不是Windows通常用的反斜线（\）。如果操作者希望使用反斜线，那么操作者需要扩展它们（利用另一个反斜线），如同反斜线是一个特殊的MySQL字符，例如：Server=C:\\Program Files\\mysql\\bin\\mysqld-opt.exe。

操作者可能还希望在文件名中使用空格，像下面这样：

```
C:/program files/mysql
```

而不是：

```
C:/progra~1/mysql
```

启动winmysqladmin可能建立一个文件my.ini，干扰现有的文件。尝试删除这个新建立的my.ini文件（如果必要，那么恢复原先的那个文件）。

如果问题仍然存在，那么可以检查错误日志，查看是否存在明显的原因。错误日志默认地存放在文件C:\MySQL\data\mysql.err中。

操作者还可以在单机模式下启动MySQL（mysqld --standalone），这样可能会出现更有用的输出信息，或者孤注一掷，在debug模式中写一个跟踪文件（通常是C:\mysqld.trace），这样可能会有些用处。

配置MySQL

为了使MySQL以所希望的方式运行，操作者需要以某些形式进行配置，如将默认的表类型设为InnoDB，或以某种语言显示错误消息。操作者可以通过三种方式设置更多的选项：通过命令行、通过配置文件或通过预设置环境参数。通过命令行设置选项对于测试是很有用的，但是如果操作者希望保持那些选项很长一段时间，那么它就变得没有什么用了。环境参数几乎从来不用。最常用的，也是最有用的方法是通过配置文件。

在UNIX中，启动配置文件通常被称为my.cnf，并且可以放在下面的位置中。MySQL按照从上到下的顺序读取文件内容，因此较靠前的位置具有较高的优先级（见表10.2所示）。

表10.2 UNIX配置文件的优先级

文件	描述
/etc/my.cnf	全局选项，对所有的服务器和用户有效。如果操作者不清楚在哪里放置配置文件，那么可以将它放在这里
DATA_DIRECTORY/my.cnf	服务器选项，在指定的目录DATA_DIRECTORY中存储数据。对于二进制文件，它通常是/usr/local/mysql/data，对于源安装文件，它通常是/usr/local/var。需要注意的是，它不一定与为mysqld指定的--datadir选项相同。相反，它是在系统设置时指定的
defaults-extra-file	服务器选项，或以命令行选项--defaults-extra-file=filename开始的客户端应用的选项
~/.my.cnf	用户选项

在Windows中，启动配置文件通常称为my.ini或my.cnf，这取决于其所在的位置（见表10.3所示）。

表10.3 Windows配置文件的优先级

文件	描述
C:WINDOWS_SYSTEM_FOLDER\my.ini	全局选项，对所有的服务器和用户有效。Windows系统文件夹通常是C:WINNT\System32、C:\WINNT或C:\WINDOWS。如果操作者不清楚该使用哪个配置文件，那么我们建议使用这个文件
C:\my.cnf	全局选项，对所有的服务器和用户都有效（可能只使用前面的文件my.ini）
C:\DATA_DIRECTORY\my.cnf	服务器选项，在指定的目录DATA_DIRECTORY中存储数据（它通常是C:\mysql\data）
defaults-extra-file=filename	服务器选项，或以命令行选项--defaults-extra-file=filename开始的客户端应用的选项

在Windows中，如果C盘不是启动盘，或者你使用了winmysqladmin功能，那么你必须使用配置文件my.ini（在Windows系统文件夹中）。

说明：Windows没有为具体用户的选项所建立的配置文件。

下面是一份简单的配置文件：

```
# The following options will be passed to all MySQL clients
[client]
#password       = your_password
port            = 3306
socket          = /tmp/mysql.sock

# The MySQL server
[mysqld]
port            = 3306
socket          = /tmp/mysql.sock
skip-locking
set-variable    = key_buffer=16M
set-variable    = max_allowed_packet=1M
set-variable    = table_cache=64
set-variable    = sort_buffer=512K
set-variable    = net_buffer_length=8K
set-variable    = myisam_sort_buffer_size=8M
#set-variable   = ft_min_word_length=3
log-bin
server-id       = 1

[mysqldump]
quick
set-variable    = max_allowed_packet=16M

[mysql]
no-auto-rehash
# Remove the next comment character if you are not familiar with SQL
#safe-updates

[myisamchk]
set-variable    = key_buffer=20M
set-variable    = sort_buffer=20M
set-variable    = read_buffer=2M
set-variable    = write_buffer=2M

[mysqlhotcopy]
interactive-timeout
```

“井号（#）”表示注释，方括号（[]）是部分的标记。在方括号中的术语表示其中的设置将会影响的程序。在本例中，配置interactive-timeout将只在程序mysqlhotcopy运行时起作用。在部分标记下设置的选项应用于以前设置过的部分，直到有下一个部分标记出现。

在前面的例子中，第一个端口用于客户端，第二个端口用于MySQL服务器。通常这两个端口是相同的，但不一定必须这样（例如，在同一台机器上运行多个MySQL服务器的时候）。

目前有三种类型的选项：

- option=value（如port=3306）。
- option（如log-bin）。这些是布尔型的选项，如果它们不存在（就使用默认值），如果存在，那么会对它们进行配置。
- set-variable=variable=value（如set-variable=write_buffer=2M）。它允许操作者对MySQL服务器参数进行设置。

警告： 在配置的例子中，有一个对客户端的密码选项被注释掉了。这对于连接可能是方便了，但是在大多数情况下出于安全的考虑我不建议这样做。所有可能读到这个文件的人都可能对MySQL进行访问。

下面的程序支持一些选项文件：myisamchk、myisampack、mysql、mysql.server、mysqladmin、mysqlcheck、mysqld、mysqld_safe、mysqldump、mysqlimport和mysqlshow。

基本上，差不多所有MySQL程序从命令行可以使用的选项都可以在一个文件中进行设置。

大部分的原版MySQL都可以针对操作环境做出正确的配置。在本章稍后的部分中，读者将看到不同服务器参数的含义，以及如何对它们进行配置以达到MySQL的最大性能。大多数MySQL产品具有四种配置样本：

MY-HUGE.CNF 对于那些为MySQL提供了1GB的专用内存的系统。

my.large.cnf 对于那些为MySQL提供了至少512MB的专用内存的系统。

my-medium.cnf 对于那些为MySQL提供了至少32MB内存的系统，或者机器中配置128MB内存，为多种应用提供服务（如既是Web服务器，又是数据库服务器）。

my-small.cnf 对于那些少于64MB内存的系统，MySQL无法占用过多的资源。

有一些产品只带有一种样本：my-example.cnf。

在最近的文件中可以看到，512MB常常不会维持住一个大的系统。建议大家将其中最适合实际需求的一个文件拷贝到所希望的目录中，然后再对它进行进一步的修改。

提示： 最好还是对自己的配置文件进行备份。如果系统失败了，那么可能要花费很长的时间对服务器进行重新配置。

日志记录

检查日志文件听起来不像是一个好主意，但是它不仅对于发现已经出现的问题是无价的，而且对于测定那些尚未发现的、可能会造成比错过一次周末晚会更糟糕的情况，也是至关重要的。MySQL有许多不同的日志文件：

The error log 这里会有MySQL启动、运行或停止时出现的问题。

The query log 所有的连接和执行的查询都会被记录在这里。

The binary update log 所有改变数据的SQL语句都存在这里。

The slow query log 所有查询，如果执行时间超过long_query_time，或者没有利用任何索引，那么都会在这里进行记录。

The update log 已经对它提出了意见，认为它应该在所有情况下都由二进制更新日志所代替。改变数据的SQL语句都存在这里。

The ISAM log 记录所有对ISAM表的修改。只对调试ISAM代码起作用。

出错日志

在Windows中，MySQL出错日志称为mysql.err，并且在UNIX中它被称为hostname.err（例如test.mysqlhost.co.za.err）。它存放在数据目录中（通常在Windows中是C:\MySQL\data，在UNIX中是/usr/local/mysql/data，或者在Red Hat Linux的诸多版本中被存放在目录/var/lib/mysql中）。

它包括启动、关闭信息和运行中出现的关键错误。如果服务器死机并自动重新启动，或者如果MySQL注意到有一张表需要自动检查或修复，那么错误日志将进行记录。日志记录还可能包含在MySQL死机时的堆栈跟踪信息。下面是一份出错日志的样本：

```
010710 19:52:43  mysqld started
010710 19:52:43  Can't start server: Bind on TCP/IP port: Address already in use
010710 19:52:43  Do you already have another mysqld server running on port: 3306 ?
010710 19:52:43  Aborting

010710 19:52:43  /usr/local/mysql-3.23.39-pc-linux-gnu-i686/bin/mysqld:
 Shutdown Complete

010710 19:52:43  mysqld ended

010710 19:55:23  mysqld started
/usr/local/mysql-3.23.39-pc-linux-gnu-i686/bin/mysqld: ready for connections
010907 17:50:38  /usr/local/mysql-3.23.39-pc-linux-gnu-i686/bin/mysqld:
 Normal shutdown

010907 17:50:38  /usr/local/mysql-3.23.39-pc-linux-gnu-i686/bin/mysqld:
 Shutdown Complete

010907 17:50:38  mysqld ended
```

这份日志记录了在本章开始时出现的MySQL没有正常启动的信息。随后在你准备正常启动MySQL时，由于其他进程已经启动，因此你无法启动MySQL。为了解决这个问题，你必须终止这个故障进程（如在UNIX中通过执行kill s 9 PID或kill -9 PID，或在Windows中使用任务管理器）。

查询日志

利用下面my.cnf文件中的选项可以启动查询日志：

```
log =[query_log_filename]
```

如果没有指定query_log_filename，那么查询日志将被赋予主机的名字。

它将记录所有的连接和执行的请求。对于出于安全目的查看谁正在连接（何时连接），以及进行调试查看是否服务器在正确接收查询，这个日志可能是有用的。

这类日志会牵扯到性能问题，因此如果考虑到性能问题，操作者应该释放它。二进制更新日志（这种日志只记录更新查询）在这里可能有用。

下面是一个示例查询日志：

```
/usr/local/mysql-max-4.0.1-alpha-pc-linux-gnu-i686/bin/mysqld, Version:
 4.0.1-alpha-max-log, started with:
Tcp port: 3306  Unix socket: /tmp/mysql.sock
Time                 Id Command    Argument
020707  1:01:29       1 Connect    root@localhost on
020707  1:01:35       1 Init DB    firstdb
020707  1:01:38       1 Query      show tables
020707  1:01:51       1 Query      select * from innotest
020707  1:01:54       1 Quit
```

二进制更新日志

当参数log-bin在配置文件my.cnf或my.ini中使用时，二进制日志记录被激活，如下所示：

```
--log-bin[=binlog_filename]
```

所有扩展名都将被丢弃，MySQL为二进制日志添加其自己的扩展名。如果没有指定文件名，那么根据附加的-bin，二进制日志文件以主机名进行命名。MySQL还建立了一个二进制索引文件，它具有相同的名字和扩展名.index。利用下面的参数，该索引可以赋予不同的名字（和位置）：

```
--log-bin-index=binlog_index_filename
```

二进制更新日志包含所有更新数据库的SQL语句，以及执行查询所需的时间和查询何时执行的时间戳。语句以它们执行的顺序进行排列记录（在查询完成后，但是要在事务完成前或锁被解开前）。还没有被提交的更新先被放在缓存之中。

二进制更新日志还对备份恢复（见第11章“数据库备份”）以及从主数据库向从属数据库进行备份（见第12章“数据库复制”）很有用。

二进制更新日志文件从扩展名001开始。每次服务器重新启动或mysqladmin refresh、mysqladmin flush-logs或FLUSH LOGS的其中一个运行时，如果生成新的文件，数字便加一。当二进制日志达到max_bin_log_size时，也生成（并增加）新的日志文件。

max_bin_log_size的值在文件my.cnf或my.ini中进行设置，如下所示：

```
set-variable    = max_binlog_size = 1000M
```

读者在查看参数值时可以看到其以字节为单位的默认空间大小：

```
% mysqladmin -u root -pg00r002b variables | grep 'max_binlog_size'
max_binlog_size                 | 1073741824
```

二进制更新索引文件包含了到目前所使用的所有二进制日志的列表。下面是一个简单的例子：

```
./test-bin.001
./test-bin.002
./test-bin.003
./test-bin.004
```

如果操作者现在刷新这些日志，那么二进制更新索引将被附加新的二进制日志：

```
% mysqladmin -u root -pg00r002b flush-logs
```

这个例子现在包含下列内容：

```
./test-bin.001
./test-bin.002
./test-bin.003
./test-bin.004
./test-bin.005
```

操作者可以利用RESET MASTER删除所有的未用二进制更新日志：

```
mysql> RESET MASTER;
Query OK, 0 rows affected (0.00 sec)
```

二进制更新索引现在显示只有一个二进制更新日志：

```
./test-bin.006
```

警告： 直到确认二进制更新日志将不再需要时再删除它们。如果正在进行复制，那么要特别小心（更多的有关内容参见第12章）。如果正在使用二进制日志恢复备份，那么应确保没有删除所有比最新的备份更新的日志。

并不是所有数据库的所有更新都需要日志记录。在很多情况下，操作者可能只希望存储某些数据库的更新。配置文件my.cnf和my.ini中的binlog-do-db和binlog-ignore-db参数使你能够对此进行控制。前一个参数指出哪个数据库将要被记录。例如下面的配置：

```
binlog-do-db = firstdb
```

将只记录数据库firstdb的更新，但是下面的：

```
binlog-ignore-db = test
```

将记录除test以外的所有数据库。如果操作者希望记录多个数据库，那么可以填加若干行：

```
binlog-do-db = test
binlog-do-db = firstdb
```

当更新作为事务的一部分将要被记录时，MySQL通过配置文件中binlog-cache-size指定的大小建立缓冲区（默认大小是32KB或32 768字节）。每个线程都可以建立其中一个缓冲区。为了避免同时使用过多的缓冲区，还需要进行参数max-binlog-cache-size的设置。默认的最大值是4GB或4 294 967 295字节。

由于二进制日志是一个二进制文件，因此数据比在旧的文本型更新日志中更有效，并且操作者不能通过文本编辑器查看其中的数据。工具mysqlbinlog可以使操作者看到这些数据：

```
C:\Program Files\MySQL\data>..\bin\mysqlbinlog test-bin.002
# at 4
#020602 18:40:02 server id  1   Start: binlog v 2, server v 4.0.1-alpha-max-log
created 020602 18:40:02
# at 79
#020602 18:41:27 server id  1   Query   thread_id=3     exec_time=0
 error_code=0
use firstdb;
SET TIMESTAMP=1023036087;
CREATE TABLE customer(id INT);
# at 146
#020602 18:41:40 server id  1   Query   thread_id=3     exec_time=0
 error_code=0
SET TIMESTAMP=1023036100;
INSERT INTO customer(id) VALUES(1);
# at 218
#020602 18:43:12 server id  1   Query   thread_id=5     exec_time=0
 error_code=0
SET TIMESTAMP=1023036192;
INSERT INTO customer VALUES(12);
# at 287
#020602 18:45:00 server id  1   Stop
```

为了使用二进制更新日志来更新MySQL服务器中的内容，需要向所要求的服务器完整地输送这些结果，举例来说：

```
% mysqlbinlog ..\data\test-bin.022 | mysql
```

说明：使用二进制更新日志的结果之一就是，并发的插入将不能与CREATE INSERT或INSERT SELECT一起运行。并发的插入是MySQL准许读和写同时发生在MyISAM表中，但是通过这两类语句启动它们将意味着，二进制更新日志不能可靠地用于备份恢复或复制。

低速查询日志

操作者可以通过配置文件中的下列参数开始低速查询日志：

```
log-slow-queries[=slow_query_log_filename]
```

如果没有提供slow_query_log_filename，那么低速查询日志将被赋予主机名，并被附加slow.log （例如，test.mysqlhost.co.za-slow.log）。

所有执行时间长于long_query_time的SQL语句都要被记录。

在配置文件my.cnf或my.ini中设置它，如下所示：

```
set-variable    = long_query_time = 20
```

这个参数以秒为单位进行衡量（因为MySQL计划很快以毫秒来衡量，所以读者需要查看最新的文档）。

如果设置了参数log-long-format，那么所有没有使用索引的查询也将被记录。在文件my.cnf或my.ini中加入下面这一行可以记录这些查询：

```
log-long-format
```

这是一个有用的日志。它对于性能的影响不大（假设所有查询都很快），并且强调了那些最需要注意的查询（丢失了索引或索引没有得到最佳应用）。

下面是一份示例低速查询日志：

```
/usr/local/mysql-max-4.0.1-alpha-pc-linux-gnu-i686/bin/mysqld, Version:
 4.0.1-alpha-max-log, started with:
Tcp port: 3306  Unix socket: /tmp/mysql.sock
Time                 Id Command    Argument
# Time: 020707 13:57:57
# User@Host: root[root] @ localhost []
# Query_time: 0  Lock_time: 0  Rows_sent: 8  Rows_examined: 8
use firstdb;
select id from sales;
# Time: 020707 13:58:47
# User@Host: root[root] @ localhost []
# Query_time: 0  Lock_time: 0  Rows_sent: 6  Rows_examined: 8
```

在这份日志中，由于查询select id from sales没有利用索引，因此它被记录在这里。这个查询可能已使用了字段id上的索引。在哪里使用索引可以参见第4章“索引和查询优化”。

操作者还可以使用工具mysqldumpslow来显示低速查询日志的内容：

```
% mysqldumpslow test-slow.log

Reading mysql slow query log from test-slow.log
Count: 1  Time=0.00s (0s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@localhost
  # Query_time: N  Lock_time: N  Rows_sent: N  Rows_examined: N
  use firstdb;
  select id from sales

Count: 1  Time=0.00s (0s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@localhost
  # Query_time: N  Lock_time: N  Rows_sent: N  Rows_examined: N
  DELETE FROM sales WHERE id>N

Count: 1  Time=0.00s (0s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@localhost
  # Query_time: N  Lock_time: N  Rows_sent: N  Rows_examined: N
  select id from sales where id<N
```

循环日志

尽管日志文件非常有用，但是本质上它们仍然存在弊病，它们会一直增长到没有任何空间为止。最后操作者需要删除多余的日志文件，利用一些脚本来自动执行这项任务是比较

好的办法。

对于那些并不重要的日志，下面的做法就足够了（假设操作者从包含日志文件的目录中开始）。下面的命令是在UNIX系统中执行的：

```
mv logfile backup_directory/logfile.old
mysqladmin flush-logs
```

下面的命令是在Windows系统中执行的：

```
move logfile backup_directory\logfile.old
mysqladmin flush-logs
```

刷新日志（在连接服务器时还可以通过SQL语句FLUSH LOGS来执行）会关闭并再次打开那些没有依序增长的日志（如低速查询日志）。或者在日志出现增长的情况下（二进制更新日志），刷新日志会建立一个新的带有扩展名的日志文件，并且一个接一个地增加，强制MySQL使用新的文件。

如果旧的日志文件不会再被使用，那么它可以被移走进行备份，也可以被删除。任何在两个语句之间处理的查询都不会进行日志记录，因此发生在那时的查询日志不存在。例如，假设查询日志称做querylog，下面这组命令展示了一个方法对日志进行循环。操作者需要打开两个窗口，窗口Window1连接到系统外壳或命令行，窗口Window2连接到MySQL。

首先，从Window1中：

```
% mv querylog querylog.old
```

现在从Window2中执行查询（连接到MySQL）：

```
mysql> SELECT * FROM sales;
```

从Window1中查看是否查询被记录在日志中：

```
% tail querylog
tail: querylog: No such file or directory
```

到操作者刷新日志时，日志文件才会出现，并且查询将被记录在日志中：

```
% mysqladmin -uroot -pg00r002b flush-logs
```

从Window2中执行另一个查询：

```
mysql> SELECT * FROM customer;
```

与你在Window1中看到的一样，这次查询被记录到查询日志中：

```
% tail querylog
/usr/local/mysql-max-4.0.1-alpha-pc-linux-gnu-i686/bin/mysqld, Version:
 4.0.1-alpha-max-log, started with:
Tcp port: 3306  Unix socket: /tmp/mysql.sock
Time                 Id Command    Argument
020707 20:45:23       5 Quit
020707 20:45:26       4 Query       select * from customer
```

这种方法不能与重要的日志文件（如二进制更新日志）一起使用，这是因为如果它们对复制或恢复备份有用，那么一些查询记录就不可能丢失。因此，只要日志被刷新，那么就会生成新的二机制更新日志，其扩展名每次加一。记录只会被添加到最近的日志中，这意味着操作者可以移动旧的日志，而不必担心查询会被丢失。假设有一个称为gmbinlog的二进制更新日志，它以一个二进制更新日志开始，那么可以尝试一下下面的操作：

```
C:\Program Files\MySQL\data>dir *-bin*

 Volume in drive C has no label
 Volume Serial Number is 2D20-1303
 Directory of C:\Program Files\MySQL\data

GMBINLOG 001           272  07-07-02  8:50p gmbinlog.001
GMBINL~1 IND             0  07-07-02  8:48p gmbinlog.index
         2 file(s)            398 bytes
         0 dir(s)      33,868.09 MB free
C:\Program Files\MySQL\data>..\bin\mysqladmin flush-logs
C:\Program Files\MySQL\data>dir *-bin*

 Volume in drive C has no label
 Volume Serial Number is 2D20-1303
 Directory of C:\Program Files\MySQL\data

GMBINLOG 001           272  07-07-02  8:50p gmbinlog.001
GMBINL~1 IND             0  07-07-02  8:48p gmbinlog.index
GMBINLOG 002             0  07-07-02  8:50p gmbinlog.001
         3 file(s)            398 bytes
         0 dir(s)      33,868.09 MB free
C:\Program Files\MySQL\data> move gmbinlog.001 D:\backup_directory\gmbinlog001.old
```

警告：如果在使用复制，那么直到确信没有从服务器再需要旧的二进制日志文件后，才能删除它们。更详细的内容请见第12章。

运行于Red Hat Linux上的MySQL带有一个日志循环脚本。如果你现在使用的这个软件没有这个脚本，那么你可以按下面的内容生成自己的脚本：

```
# This logname is set in mysql.server.sh that ends up in /etc/rc.d/init.d/mysql
#
# If the root user has a password you have to create a
# /root/.my.cnf configuration file with the following
# content:
#
# [mysqladmin]
# password = <secret>
# user= root
#
# where "<secret>" is the password.
#
# ATTENTION: This /root/.my.cnf should be readable ONLY
```

```
# for root !
/usr/local/var/mysqld.log {
        # create 600 mysql mysql
        notifempty
        daily
        rotate 3
        missingok
        compress
    postrotate
        # just if mysqld is really running
        if test -n "`ps acx|grep mysqld`"; then
                /usr/local/bin/mysqladmin flush-logs
        fi
    endscript
}
```

对表进行优化、分析、检查和修复

数据库管理员的工作中常规的部分是进行预防性的维护，以及修复那些出现问题的内容。即使付出最大的努力，数据错误还是可能发生，如断电所造成的写中断。通常操作者可以相当容易地纠正这些错误。

进行检查和修复通常具有四个主要的任务：

- 对表进行优化
- 对表进行分析（分析并存储MyISAM和BDB表中键的分布）
- 对表进行检查（检查表的错误，并且为MyISAM更新键的统计内容）
- 对表进行修复（修复被破坏的MyISAM表）

对表进行优化

随着时间的推移，包括BLOB和VARCHAR字节的表将变得比较繁冗。因为这些字段长度不同，当记录进行更新、插入或删除时，它们永远都不会占有相同大小的空间。记录将开始变成碎片，并且将留下空闲的空间。就好像一个具有碎片的磁盘，这种情况将降低性能，因此为了保持MySQL具有非常好的形式，操作者将要进行常规的碎片整理。优化表是一种方法，它可以通过很多方式实现。它具有OPTIMIZE TABLE语句、mysqlcheck工具（如果服务器在运行）或myisamchk工具（如果服务器没有运行或表中没有交互）。

只优化当前MyISAM的工作和部分B2B表。利用MyISAM表，优化过程如下面的内容：

- 对那些行被断开或已被删除的表进行碎片整理。
- 如果索引还没有进行分类，那么对它们进行分类。
- 如果索引统计还没有进行更新，那么对它们进行更新。

利用BDB表，优化可以对键的分布进行分析（与ANALYZE TABLE相同，可以参见本章稍后部分的“利用ANALYZE TABLE对表进行分析”）。

利用OPTIMIZE语句对表进行优化

语句OPTIMIZE是一条SQL语句，在连接到MySQL数据库时使用。下面是它的语法：

```
OPTIMIZE TABLE tablename
```

操作者还可以立刻对很多表进行优化，之间以逗号（,）进行分隔：

```
mysql> OPTIMIZE TABLE customer,sales;
+------------------+----------+----------+---------------------------+
| Table            | Op       | Msg_type | Msg_text                  |
+------------------+----------+----------+---------------------------+
| firstdb.customer | optimize | status   | Table is already up to date |
| firstdb.sales    | optimize | status   | OK                        |
+------------------+----------+----------+---------------------------+
2 rows in set (0.02 sec)
```

在这种情况下，customer表已经进行了优化。

利用mysqlcheck对表进行优化

mysqlcheck是一个命令行工具，除了优化之外，它可以执行大量的检查和修复任务。对mysqlcheck功能的全面描述可以在本章稍后的“使用mysqlcheck”部分看到。为了优化firstdb数据库中的customer表，需要使用mysqlcheck的参数-o，如下所示：

```
% mysqlcheck -o firstdb customer  -uroot -pg00r002b
firstdb.customer            Table is already up to date
```

mysqlcheck允许操作者一次对多个表进行优化，方法是在数据库的名字后列出所有的表：

```
% mysqlcheck -o firstdb customer sales  -uroot -pg00r002b
firstdb.customer            Table is already up to date
firstdb.sales               Table is already up to date
```

操作者还可以省去所有表的引用，对整个数据库进行优化，如下所示：

```
% mysqlcheck -o firstdb -uroot -pg00r002b
```

利用myisamchk对表进行优化

最后，在服务器关闭或没有与服务器互操作的时候，操作者可以使用myisamchk命令行工具（如果服务器正在运行，那么在运行这条语句之前利用mysqladmin flush-tables对表进行刷新。操作者还需要确保服务器没有与表进行互操作，否则会出现故障）。这是对表进行检查的最老的办法。操作者必须在表的正确位置上运行myisamchk，或者指定表所在的路径。对于myisamchk功能的全面描述可以在本章后面的“使用myisamchk”一节中看到。

下面是一条与其作用相同的优化语句：

```
myisamchk --quick --check-only-changed --sort-index --analyze tablename
```

或者：

```
myisamchk -q -C -S -a tablename
```

举例来说：

```
% myisamchk --quick --check-only-changed --sort-index --analyze customer
- check key delete-chain
- check record delete-chain
- Sorting index for MyISAM-table 'customer'
```

参数-r可以对表进行修复，同时也删去了浪费的空间：

```
% myisamchk -r sales
- recovering (with sort) MyISAM-table 'sales'
Data records: 8
- Fixing index 1
- Fixing index 2
```

如果操作者没有指定表索引文件的路径，并且操作者没有从正确的目录下开始，那么就会出现下面的错误：

```
% myisamchk -r customer
myisamchk: error: File 'customer' doesn't exist
```

指定.MYI文件的全路径可以避免这个错误：

```
% myisamchk -r /usr/local/mysql/data/firstdb/customer
- recovering (with keycache) MyISAM-table '/usr/local/mysql/data/firstdb/customer'
Data records: 0
```

警告：在优化操作的过程中，表会被锁住，因此不要在很忙的时候进行这个操作。同样，确信你在运行OPTIMIZE TABLE时有足够的空余空间。如果在你的系统几乎或已经没有磁盘空间的时候，MySQL将可能不能完成优化，你的表可能无法使用。

优化是对包含MyISAM表的数据库的常规管理事务中一个重要环节，应该定期进行。

对表进行分析

通过更新表的索引信息对表进行分析，可改善数据库性能，这样MySQL就能够就如何连接表做出更好的决定。不同索引项的分类被存储下来，准备稍后的应用（现在分析操作只对MyISAM和BDB表起作用）。

有三种方法可以对表进行分析：

- 当连接到MySQL时，利用ANALYZE TABLE语句
- 利用mysqlcheck命令行工具
- 利用myisamcheck命令行工具

对表的定期分析可以改善性能，并且应该成为常规维护工作中的一部分。

利用ANALYZE TABLE对表进行分析

ANALYZE TABLE是在连接到服务器上的数据库时使用的语句。下面是这条语句的语法：

```
ANALYZE TABLE tablename
```

例如：

```
mysql> ANALYZE TABLE sales;
+---------------+---------+----------+----------+
| Table         | Op      | Msg_type | Msg_text |
+---------------+---------+----------+----------+
| firstdb.sales | analyze | status   | OK       |
+---------------+---------+----------+----------+
1 row in set (0.00 sec)
```

Msg_type（消息类型）包括状态、错误、信息或告警中的一项。如果索引文件完全丢失，并且操作者试图对表进行分析，那么将发生什么事情呢：

```
mysql> ANALYZE TABLE zz;
+------------+---------+----------+----------------------------------+
| Table      | Op      | Msg_type | Msg_text                         |
+------------+---------+----------+----------------------------------+
| firstdb.zz | analyze | error    | Table 'firstdb.zz' doesn't exist |
+------------+---------+----------+----------------------------------+
1 row in set (0.00 sec)
```

如果表在上次分析操作后被修改了，那么它只会被再次进行分析：

```
mysql> ANALYZE TABLE sales;
+---------------+---------+----------+----------------------------+
| Table         | Op      | Msg_type | Msg_text                   |
+---------------+---------+----------+----------------------------+
| firstdb.sales | analyze | status   | Table is already up to date |
+---------------+---------+----------+----------------------------+
1 row in set (0.00 sec)
```

利用mysqlcheck对表进行分析

mysqlcheck命令行工具将在本章稍后章节“使用mysqlcheck”中进行全面的讨论。运行mysqlcheck需要服务器的运行，并且它只对MyISAM表起作用。为了用它对表进行分析，操作者可以使用-a参数：

```
% mysqlcheck -a firstdb sales -uroot -pg00r002b
firstdb.sales                                          OK
```

操作者还可以对数据库中的多个表进行分析，方法是在数据库的名字后列出多个表的名字：

```
% mysqlcheck -a firstdb sales customer -uroot -pg00r002b
firstdb.sales                  Table is already up to date
firstdb.customer               Table is already up to date
```

如果操作者试图对不支持分析操作的的表进行分析（如表InnoDB），那么操作将无法进行，但对表不会产生破坏。例如：

```
% mysqlcheck -a firstdb innotest -uroot -pg00r002b
firstdb.innotest
error    : The handler for the table doesn't support check/repair
```

不输入任何表的名字，操作者还可以分析数据库中的所有表：

```
% mysqlcheck -a firstdb innotest -uroot -pg00r002b
```

利用myisamchk对表进行分析

myisamchk命令行工具将在本章稍后的部分“使用myisamchk”中进行全面的讨论。服务器不应该运行，或者你必须确信没有与你所操作的表发生互操作。如果没有使用--skip-external-locking参数，即使服务器在运行，那么操作者仍然可以安全地使用myisamchk对表进行分类。如果使用了--skip-external-locking参数，那么操作者需要在开始分析之前对表进行刷新（利用命令mysqladmin flush-tables），并且确保没有对这个表的访问发生。如果在mysiamchk运行时，mysqld或其他任何人或事对表进行了访问操作，那么操作者可能得到无效的结果。为了对表进行分析，需要使用-a参数：

```
% myisamchk -a /usr/local/mysql/data/firstdb/sales
Checking MyISAM file: /usr/local/mysql/data/firstdb/sales
Data records:       9   Deleted blocks:       0
- check file-size
- check key delete-chain
- check record delete-chain
- check index reference
- check data record references index: 1
- check data record references index: 2
```

对表进行检查

当索引与数据不同步的时候，错误就会发生。系统崩溃或电源故障都可能导致表被破坏的情况发生。数据的破坏是相当少见的。大多数情况下，被破坏的是索引文件。尽管操作者可能注意到信息返回的速度慢或数据没有在其所在地被找到，但是破坏依然很难被发现。在操作者对错误有所怀疑时，首要的事情应该是对表的检查。表被破坏的一些征兆包括下面的一些错误：

- 文件意外的结束。
- 记录文件失效。
- tablename.frm被锁住，不能进行修改。
- 无法找到文件tablename.MYI（Errcode: ###）。
- 从表的处理程序得到错误###。perror工具给出了相关错误号的更多的信息。运行perror和错误号，例如：

```
% perror 126
126 = Index file is crashed / Wrong file format
```

其他常见错误还包括：

```
126 = Index file is crashed / Wrong file format
127 = Record-file is crashed
132 = Old database file
134 = Record was already deleted (or record file crashed)
135 = No more room in record file
136 = No more room in index file
141 = Duplicate unique key or constraint on write or update
144 = Table is crashed and last repair failed
145 = Table was marked as crashed and should be repaired
```

一旦连接到MySQL服务器，操作者可以使用CHECK TABLE命令、使用mysqlcheck工具（当服务器运行时），或者在服务器已经停止运行时使用myisamchk工具。检查会对索引统计进行更新，并检查错误。

如果找到了错误，那么需要对表进行修复（请见本章稍后的“修复表”部分）。严重的错误会导致表被破坏，这种情况下直到表被修复后才能再被使用。

提示： *在电源故障或系统崩溃后，永远都要对表进行检查。操作者通常会在用户注意到出现问题之前修复所有发生的损坏。*

利用CHECK TABLES对表进行检查

CHECK TABLE的语法如下：

```
CHECK TABLE tablename [option]
```

例如：

```
mysql> CHECK TABLE customer;
+------------------+-------+----------+----------+
| Table            | Op    | Msg_type | Msg_text |
+------------------+-------+----------+----------+
| firstdb.customer | check | status   | OK       |
+------------------+-------+----------+----------+
1 row in set (0.01 sec)
```

CHECK只能够检查表MyISAM和InnoDB。

根据操作者要进行的操作，有五个参数可以使用，如表10.4所示。

表10.4 CHECK TABLE参数

参数	描述
QUICK	这是最快的检查，并且没有对行进行扫描以发现错误链接
FAST	只检查那些没有正常关闭的表

（续表）

参数	描述
CHANGED	只检查那些没有正常关闭或在上次检查后被修改的表
MEDIUM	这是默认的参数。它对行进行扫描，以检查那些删除的链接是正确的。它还对行进行键的校验和计算，并且利用计算出的键的校验和进行验证
EXTENDED	最慢的方法，但是它可以通过对每一行相关的每个索引进行的全面的键查找，对表的一致性进行检查

QUICK参数对于检查那些没有错误疑问的表是很有用的。

如果错误或告警被返回，那么操作者应该试着修复这个表。

操作者一次可以检查多个表，方法是一个接一个列出这些表的名字，例如：

```
mysql> CHECK TABLE sales,customer;
+-----------------+-------+----------+----------+
| Table           | Op    | Msg_type | Msg_text |
+-----------------+-------+----------+----------+
| firstdb.sales   | check | status   | OK       |
| firstdb.customer| check | status   | OK       |
+-----------------+-------+----------+----------+
2 rows in set (0.01 sec)
```

利用mysqlcheck对表进行检查

在服务器运行时，可以使用mysqlcheck命令行工具，并且它只对MyISAM表起作用。本章稍后的部分“使用mysqlcheck”会对它进行全面的描述。

表10.5 在对表进行检查时使用的mysqlcheck参数

参数	描述
--auto-repair	与检查参数一起使用，在检查完成后，它将自动开始修复被破坏的表
-c, --check	对表进行检查
-C, --check-only-changed	对上次检查后修改了的表或没有正常关闭的表进行检查
-F, --fast	对那些没有正常关闭的表进行检查
-e, --extended	这是最慢的检查形式，但是它会确保表完全一致。操作者还可以利用这个参数进行修复操作，尽管通常并不必要
-m, --medium-check	这比扩展检查要快得多，并且它会找到大量的错误
-q, --quick	最快的检查，在进行检查时，它并没有对表中的行进行检查。当进行修复时，它只修复索引树

命令的语法如下：

```
mysqlcheck [options] databasename tablename[s]
```

例如：

```
% mysqlcheck -c firstdb customer -uroot -pg00r002b
firstdb.customer                                    OK
```

操作者可以在数据库的名字后列出许多表的名字从而检查多个表：

```
% mysqlcheck -c firstdb sales customer -uroot -pg00r002b
firstdb.sales                                       OK
firstdb.customer                                    OK
```

只给出数据库的名字，操作者可以检查数据库中所有的表。

```
% mysqlcheck -c firstdb -uroot -pg00r002b
```

利用myisamchk对表进行检查

当服务器被关闭或与正在检查的表没有交互操作时，操作者可以使用myisamchk命令行参数（在本章稍后的“使用mysiamchk”部分中进行全面的描述）。如果没有使用参数--skip-external-locking，即使服务器正在运行，操作者仍然可以安全地使用myisamchk对表进行检查。表将被锁住，访问将受到影响，但是不会出现错误报告。如果使用了--skip-external-locking，那么操作者需要在开始检查前对表进行刷新（利用mysqladmin flush-tables），并且确信没有任何访问发生。如果mysqld或任何其他的人或事对这个表进行了访问，那么操作者可能遇到错误的结果（即使在表正在被使用时，它们仍然被标记为失效）。

命令语法如下所示：

```
mysiamchk [options] tablename
```

与CHECK TABLE语句相同的是中间选项：

```
myisamchk -m table_name
```

myisamchk默认的参数是普通检查选项（-c）。还有快速检查（-F），它只检查那些没有正常关闭的表。小写的-f参数与它不同，是强制性参数，意思是即使发生错误仍然继续检查。同样还有中速检查（-m），稍微慢一点，并且更完整。最极端的参数是-e（执行扩充的检查），它是最彻底，也是最慢的参数。它通常也是绝望的标志，只有在其他所有的参数都失败时才使用它。提高key_buffer_size的参数值可以加速扩充检查（如果有足够多的资源）。表10.6中列出了进行检查时的参数。

表10.6 myisamchk检查操作参数

参数	描述
-c, --check	普通检查，默认参数
-e, --extend-check	最慢的，最全面的检查形式。如果操作者在使用extended-check，并且有很多内存，那么操作者应该大量增加key_buffer_size的值
-F, --fast	快速检查，只检查那些没有正常关闭的表

（续表）

参数	描述
-C, --check-only-changed	只对那些自上次检查后出现更改后的表进行检查
-f, --force	如果在表中出现错误，那么进行修复
-i, --information	显示被检查表的统计信息
-m, --medium-check	中速检查，比扩展检查更快，适用于大多数情况
-U, --update-state	记录表何时进行检查，以及是否表已失效，这对于-C参数很有用。当表正在使用，而且使用了--skip-external-locking参数时，不应该使用它
-T, --read-only	不将表标记为已检查（当服务器运行，并且使用了--skip-external-locking参数时，它对于运行myisamchk很有用）

下面是发现错误时的一个示例myisamchk输出内容：

```
% myisamchk largetable.MYI
Checking MyISAM file: Hits.MYI
Data records: 2960032   Deleted blocks:       0
myisamchk: warning: 1 clients is using or hasn't closed the table
properly
- check file-size
myisamchk: warning: Size of datafile is: 469968400 Should be: 469909252
- check key delete-chain
- check record delete-chain
- check index reference
- check data record references index: 1
- check data record references index: 2
- check data record references index: 3
myisamchk: error: Found 2959989 keys of 2960032
- check record links
myisamchk: error: Record-count is not ok; is 2960394 Should be: 2960032
myisamchk: warning: Found    2960394 parts   Should be: 2960032 parts
```

对表进行修复

如果你已经对表进行了检查，并且发现了错误，那么你需要对它们进行修复。根据所使用的方法，可用的修复参数会有多种，但是可能会失败。如果磁盘失效，或者没有表可以工作，那么惟一的选择是从备份中进行恢复。对表进行修复可能占用相当多的资源，包括磁盘和内存资源：

- 通常，修复一张表会占用两倍于原有文件大小的磁盘空间（在同一块盘上）。由于数据文件不被修改，因此快速修复（见下面部分中的参数）是一个例外。
- 新的索引文件所占用的空间（与原来的文件在相同的硬盘上）。旧的索引在开始时便被删掉了，这通常并不重要，但是如果硬盘已经趋于装满，这就变得很重要了。

- 根据标准和--sort-recover参数，分类缓冲区被建立起来。这会占用大小为(largest_key + row_pointer_length) * number_of_rows * 2的空间。如果操作者有可用的内存空间，那么通过增加mysqld变量sort_buffer_size的大小，可以将其中一些挪至内存中（可以加快处理的速度）。否则，它会由环境变量TMPDIR或myisamchk参数-t指定并建立。
- 内存的使用由mysqld参数或myisamchk命令行中设置的参数决定（见“使用myisamchk”部分）。

如果错误是由表的空间溢出造成的，并且表的类型是InnoDB，那么操作者将不得不扩大InnoDB的表空间。MyISAM表具有巨大的理论大小（8兆TB），但是默认情况下只有4GB。如果表的空间达到了这个限制，那么操作者可以通过使用MAX_ROWS和AVG_ROW_LENGTH ALTER TABLE参数对其进行扩展。为了修复名为“limited”的表（当前它只有三个记录），操作者可以按照下列的方法进行操作：

```
mysql> ALTER TABLE limited MAX_ROWS=999999999999 AVG_ROW_LENGTH=100;
Query OK, 3 rows affected (0.28 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

这将为更多的记录分配指针。当BLOB和TEXT字段出现时，AVG_ROW_LENGTH被使用，并且它为MySQL提出了记录的平均大小的概念，这可以用于优化的目的。

修复非MyISAM表的类型

在接下来的部分中讨论的三种方法只对MyISAM表起作用。一些参数据说有时也对BDB表起作用，但是它们没有为此而设计。目前，惟一的修复损坏BDB和InnoDB表的方法是从备份中恢复。

利用REPAIR TABLE修复表

当连接到MySQL服务器时，操作者可以运行REPAIR TABLE语句。目前它只对MyISAM起作用。表10.7中显示了它的参数。

语法如下：

```
REPAIR TABLE tablename[s] option[s]
```

表10.7 可用的REPAIR TABLE参数

参数	描述
QUICK	因为数据文件没有被修改，因此它是最快的修复。由于数据文件没有被修改，因此它使用的磁盘空间更少
EXTENDED	企图从数据文件中恢复每个可能的行。因为它会产生无用的数据行，所以除非作为最后的办法，此参数不应该被使用
USE_FRM	如果.MYI文件丢失或文件头（header）被损坏，那么要使用这个参数。它将对.frm表定义文件中发现的定义中的索引进行重建

下面是在丢失.MYI文件的情况下使用REPAIR的一个例子。我们可以删除表t4的.MYI文件。

```
% ls -l t4.*
-rw-rw----  1 mysql  mysql    10 Jun 14 02:00 t4.MYD
-rw-rw----  1 mysql  mysql  4096 Jun 14 02:00 t4.MYI
-rw-rw----  1 mysql  mysql  8550 Jun  8 10:46 t4.frm
% rm t4.MYI
% mysql -uguru2b -pg00r002b firstdb
```

正常的REPAIR不工作：

```
mysql> REPAIR TABLE t4;
+-------------+--------+----------+-----------------------------------+
| Table       | Op     | Msg_type | Msg_text                          |
+-------------+--------+----------+-----------------------------------+
| firstdb.t4 | repair | error    | Can't find file: 't4.MYD' (errno: 2) |
+-------------+--------+----------+-----------------------------------+
1 row in set (0.47 sec)
```

现在的错误消息（4.0.3）指出，当实际上是丢失了的.MYI文件时，.MYD文件无法找到。读到这里，好像错误消息已经得到了澄清。为了在这种情况下修复这张表，操作者需要使用USE_FRM参数，如同它的名字所建议的那样，利用.frm定义文件重建.MYI索引文件：

```
mysql> REPAIR TABLE t4 USE_FRM;
+-------------+--------+----------+-----------------------------------+
| Table       | Op     | Msg_type | Msg_text                          |
+-------------+--------+----------+-----------------------------------+
| firstdb.t4 | repair | warning  | Number of rows changed from 0 to 2 |
| firstdb.t4 | repair | status   | OK                                |
+-------------+--------+----------+-----------------------------------+
2 rows in set (0.46 sec)
```

利用mysqlcheck修复表

在服务器还在运行时，使用mysqlcheck命令行工具，并且它只对MyISAM表起作用。在本章“使用mysqlcheck”部分，我们会对它进行全面的描述。为了修复这些表，操作者可以使用-r参数：

```
% mysqlcheck -r firstdb customer -uroot -pg00r002b
firstdb.customer                                    OK
```

操作者可以在数据库的名字后列出表的名字对多个表同时进行修复：

```
% mysqlcheck -r firstdb customer sales -uroot -pg00r002b
firstdb.customer                                    OK
firstdb.sales                                       OK
```

如果出于某些原因，数据库中所有的表都损坏了，那么操作者可以只给出数据库的名字即可对所有的表进行修复：

```
% mysqlcheck -r firstdb -uroot -pg00r002b
```

利用myisamchk修复表

操作者可以使用myisamchk命令行工具（在本章稍后“使用myisamchk”部分进行全面的描述）对表进行修复（见表10.8）。

或者服务器没有运行，或者操作者必须确信与所操作的表没有互操作，就像当你利用--skip-external-locking参数启动MySQL时的情况。如果没有使用--skip-external-locking参数，操作者只能安全地使用myisamchk来修复那些确信没有发生同时访问的表。无论是否使用--skip-external-locking，操作者都需要在开始修复（利用mysqladmin flush-tables）之前对这些表进行刷新，并且确保没有访问发生。如果在myisamchk运行时，mysqld或其他人或事对表进行访问，那么操作者可能会得到错误的结果（即使在表没有损坏时，表还是被标记为损坏）。

命令的语法如下：

```
myisamchk [options] [tablenames]
```

表10.8 利用myisamchk修复表

参数	描述
-D #, --data-file-length=#	在重建数据文件时指定其最大长度
-e, --extend-check	试图从数据文件中恢复每个可能存在的行。由于它可能会产生无用行，因此除非没有办法，否则不使用这个参数
-f, --force	如果遇到旧的临时文件（其扩展名为.TMD），那么对其进行覆盖，而不是终止
-k #, keys-used=#	指定使用的键，该键能够使处理速度变得更快。每个二进制位代表了一个键，以0开始表示第一个键
-r, --recover	修复大多数损坏，并且应该是首先使用的参数。为了使恢复操作进行得更快，如果有足够的内存空间，那么操作者可以增加sort_buffer_size的大小。这个参数不会从惟一键不惟一的少数形式的损坏中进行恢复操作
-o, --safe-recover	比-r更全面也更慢的修复参数，只有在-r失效时才使用它。它从所有行中读取信息，并基于这些行重建索引。由于并没有建立分类缓冲区，因此它还比-r使用的磁盘空间更少。操作者可以增加key_buffer_size的大小来加快修复的速度
-n, --sort-recover	即使导致的临时文件很大，但仍强制MySQL使用分类来分析索引。
--character-sets-dir=...	包含字符集的目录
--set-character-set=name	为索引指定新的字符集

（续表）

参数	描述
-t, --tmpdir=path	如果操作者不希望使用环境变量指定的任何值，那么可以为临时文件指定新的存储路径
-q, --quick	由于没有对数据文件进行修改，因此它是最快的修复操作。如果具有重复的键，那么指定q两次（-q -q）将对数据文件进行修改。由于没有对数据文件进行修改，因此它还将使用更少的磁盘空间
-u, --unpack	打开已经被myisampack工具打包的文件

操作者必须从包含.MYI文件的目录中运行myisamchk，或提供相应的路径。下面是一次修复操作的例子，MySQL决定是否使用分类或keycache：

```
% myisamchk -r customer
- recovering (with keycache) MyISAM-table 'customer.MYI'
Data records: 0
% myisamchk -r sales
- recovering (with sort) MyISAM-table 'sales.MYI'
Data records: 9
- Fixing index 1
- Fixing index 2
```

为了使myisamchk执行得更快捷，如果有很多内存空间，那么除了前面所说的增加sort_buffer_size和key_buffer_size的大小之外，操作者还可以设置一些其他的变量。本章稍后的“使用myisamchk”部分将对myisamchk进行全面的描述。

使用mysqlcheck

由于很多修复和检查功能以前只能在服务器关闭后才能使用，因此mysqlcheck工具是为MySQL的最新用户提供的一种便利。幸运的是，这种对mysqlcheck工具的限制已经成为过去的事情了。

mysqlcheck使用CHECK、REPAIR、ANALYZE和OPTIMIZE语句从命令行执行这些功能，这对于自动维护数据库是有用的（见表10.9）。

语法如下：

```
mysqlcheck [options] databasename [tablenames]
```

或者：

```
mysqlcheck [options] --databases databasename1 [databasename2 databasename 3 ...]
```

或者：

```
mysqlcheck [options] --all-databases
```

表10.9 mysqlcheck参数

参数	描述
-A, --all-databases	检查所有可用的数据库
-1, --all-in-1	将对每个数据库的表的查询混合成一个查询（而不是每个表一个）。表之间用逗号分隔
-a, --analyze	分析列出的表
--auto-repair	如果表被损坏了，那么自动修复它们（在查询中的表都检查过后）
-#, --debug=...	输出debug日志
--character-sets-dir=...	指出字符集所在的目录
-c, --check	检查表
-C, --check-only-changed	对上次检查后出现修改的表或没有正确关闭的表进行检查
--compress	在客户端/服务器协议中使用压缩方式
-?, --help	显示帮助消息并退出
-B, --databases	列出很多数据库进行检查（检查数据库中的所有表）
--default-character-set=...	设置默认的字符集
-F, --fast	检查没有正确关闭的表
-f, --force	即使遇到错误，仍强制进行处理
-e, --extended	这是最慢的检查形式，但是将确保表是完全一致的。虽然通常不必要使用它，但是操作者还可以在修复操作中使用这个参数
-h, --host=...	连接的主机名
-m, --medium-check	比扩展（extended）检查的速度更快，并且能够发现大量的错误
-o, --optimize	对表进行优化
-p, --password[=...]	连接需要的密码
-P, --port=...	连接用到的端口
-q, --quick	最快的检查，在进行检查时不对表中的行进行检查。在修复时，它只修复索引树
-r, --repair	修复大多数错误，除了那些无故出现重复的惟一键
-s, --silent	除了错误消息以外，不显示任何输出信息
-S, --socket=...	指定在连接时使用的套接字文件
--tables	列出要检查的表。带有-B参数时，它会优先执行
-u, --user=#	指定连接时所用的用户账号
-v, --verbose	打印处理的大量输出信息
-V, --version	显示版本信息并退出

mysqlcheck工具还有一个特性，允许在它不指定参数的情况下，以不同的方式执行操作。通过利用下面的名字简单地建立mysqlcheck拷贝，它将执行默认的操作：

- mysqlrepair：默认参数为-r。
- mysqlanalyze：默认参数为-a。
- mysqloptimize：默认参数为-o。

当它成为指定的mysqlcheck时，默认的参数是-c。所有这些重命名的文件都可以具有全部的mysqlcheck功能——只是它们的默认操作发生了变化。

使用myisamchk

myisamchk是较老的工具，从MySQL面世一直至今一直都有。它还可以用于分析、检查和修复表，但是如果在服务器运行时希望使用它，那么操作者需要仔细操作。表10.10描述了myisamchk通常使用的一些参数，表10.11描述了检查参数，表10.12描述了修复参数，表10.13描述了其他的参数。

服务器应该停止运行，或者操作者要确信与所要操作的表没有任何互操作，如在启动MySQL时使用--skip-external-locking参数。如果使用了--skip-external-locking参数，并且确信没有同时进行访问操作，那么操作者之可以安全地使用myisamchk来修复表。无论是否使用了--skip-external-locking，操作者都需要在启动修复（利用mysqladmin flush-tables）之前刷新表，并要确保没有进行访问操作。

我建议操作者在服务器运行时使用其他的参数。

语法如下所示：

```
myisamchk [options] tablename[s]
```

除非指定.MYI索引文件所在的目录，否则操作者必须在它们所在的目录中运行myisamchk，不然就会得到下面的错误信息：

```
% myisamchk -r sales.MYI
myisamchk: error: File 'sales.MYI' doesn't exist
```

指定路径可以解决这个问题：

```
% myisamchk -r /usr/local/mysql/data/firstdb/sales.MYI
- recovering (with sort) MyISAM-table '/usr/local/mysql/data/firstdb/sales.MYI'
Data records: 9
- Fixing index 1
- Fixing index 2
```

表的名字可以指定带有或不带有扩展名.MYI。

```
% myisamchk -r sales
- recovering (with sort) MyISAM-table 'sales'
Data records: 9
- Fixing index 1
- Fixing index 2
% myisamchk -r sales.MYI
```

```
- recovering (with sort) MyISAM-table 'sales.MYI'
Data records: 9
- Fixing index 1
- Fixing index 2
```

警告：常见的错误是对一个.MYD数据文件试图运行myisamchk。永远使用.MYD索引文件！

操作者可以使用通配符来搜索数据库目录中（*.MYI）所有的表以至所有数据库的所有表：

```
% myisamchk -r /usr/local/mysql/data/*/*.MYI
```

表10.10 常规的myisamchk参数

参数	描述
-#, --debug=debug_options	输出一份debug日志。常见的debug_option串是d:t:o,filename
-?, --help	显示帮助消息并退出
-O var=option, --set-variable var=option	设置变量的值。myisamchk的可能的变量和它们默认的值可以通过myisamchk --help查找
-s, --silent	只输出错误消息。可以使用秒s参数使myisamchk操作不会频繁地输出信息
-v, --verbose	比平常显示更多的信息。在与silent一起使用时，多个v可以用来输出更多的信息（-vv或-vvv）
-V, --version	显示myisamchk版本信息并退出
-w, --wait	如果表被锁住了，-w将等到表被解锁而不是退出并显示出错信息。如果mysqld运行时带有参数--skip-external-locking，那么表只能被其他myisamchk命令锁住

通过运行myisamchk --help，除了常规的参数之外，操作者可以看到利用-O参数可以修改哪些变量，并且当前的设置是：

```
% myisamchk --help
..
Possible variables for option --set-variable (-O) are:
key_buffer_size       current value: 520192
myisam_block_size     current value: 1024
read_buffer_size      current value: 262136
write_buffer_size     current value: 262136
sort_buffer_size      current value: 2097144
sort_key_blocks       current value: 16
decode_bits           current value: 9
ft_min_word_len       current value: 4
ft_max_word_len       current value: 254
ft_max_word_len_for_sort  current value: 20
```

在做一个扩展的检查或当索引一次被插入一行时（使用safe-recover参数），使用由key_buffer_size分配的空间。当索引在修复中被分类时，sort_buffer_size被用于默认的修复。

为了获得更快的修复速度，可以将sort_buffer_size设置为可用内存总量的1/4。每次只能使用两个参数中的一个，因此当两个参数的值变大时，操作者不必担心内存会溢出。

说明： 在my.cnf文件（或my.ini）内，对于mysqld和myisamchk有不同的部分。操作者可以容易地将sort_buffer_size设置为高数值，进行修复。如果系统在每天的运行中有其他的需要，可以将它保持在较低的值。

表10.10 myisamchk的检查参数

参数	描述
-c, --check	常规检查，也是默认的参数
-e, --extend-check	最慢的、最全面的检查形式。如果在使用--extended-check并且没有很多内存，那么应该将key_buffer_size的值增加得大一些
-F, --fast	快速检查，只对那些没有正确关闭的表进行检查
-C, --check-only-changed	只对那些自上次检查后修改过的表进行检查
-f, --force	如果在表中发现错误，那么这个参数执行修复参数
-i, --information	显示被检查表的统计信息
-m, --medium-check	中等检查，比扩展检查更快并且大多数情况下足够好
-U, --update-state	保存有关何时表被检查以及表是否已失效的信息，这对于-C参数是有用的。当表正在使用并且使用了--skip-external-locking参数时，不应该使用这个参数
-T, --read-only	它并不将表标记为检查过的状态（对于在服务器运行并使用--skip-external-locking参数时运行myisamchk的情况有用）

表10.11 myisamchk修复参数

参数	描述
-D #, --data-file-length=#	在重建数据文件时，指定最大的数据文件长度
-e, --extend-check	企图从数据文件中恢复所有可能的行。因为它会产生无用的行，所以除非它是最后的办法，否则不使用这个参数
-f, --force	如果遇到以前存在的临时文件（具有.TMD扩展名），那么对其进行覆盖而不是终止
-k #, keys-used=#	指定要用的键，这可以使处理过程更快。每个二进制位代表一个键，以0开始表示第一个键（例如，1是第一个索引，10是第二个索引）
-r, --recover	它可以修复大多数损坏，并且应该作为第一个参数尝试使用。如果有足够的内存空间，为了加快恢复的进程，操作者可以增加sort_buffer_size的大小。这个参数不会从惟一键不惟一的少数形式的损坏中进行恢复操作

（续表）

参数	描述
-o, --safe-recover	比-r更全面的、更慢的修复参数，只有在-r失效时才应该使用它。它从所有的行中读取信息，并在行的基础上重建索引。由于没有建立分类缓存，因此它还比-r使用的磁盘空间要少。为了加快修复的速度，操作者可以增加key_buffer_size的大小
-n, --sort-recover	即使产生的临时文件很大，也强制MySQL使用分类来分析索引
--character-sets-dir=...	包含字符集的目录
--set-character-set=name	为索引指定新的字符集
-t, --tmpdir=path	如果不希望使用环境变量TMPDIR的内容，那么需要为存储临时文件指定新的目录
-q, --quick	最快的修复，运行时数据文件没有被修改。如果出现重复的键，那么第二次运行这个参数将修改数据文件。由于数据文件没有修改，因此使用较少的磁盘空间
-u, --unpack	将由myisampack工具打包的文件解包

表10.12 其他的myisamchk参数

参数	描述
-a, --analyze	通过修改表的索引信息以分析表来改善性能，这样MySQL可以就如何连接表做出更好的决定。不同索引项的分类为了以后的使用被存储下来。这个参数与ANALYZE TABLE相同
-d, --description	显示表的描述
-A, --set-auto-increment [=value]	将AUTO_INCREMENT的计数器设置为指定的值（或者如果没有提供数值，则加1）
-S, --sort-index	以降序对索引树块进行分类，这会改善按键搜寻和进行表扫描的性能
-R, --sort-records=#	根据指定的索引进行记录的分类（索引号码从1开始；操作者可以使用SHOW INDEX来查看排序的列表）。这可以加速按索引排序的查询，以及归类后的选择。如果操作者对一个大的表第一次进行分类，那么速度可能会很慢

运行带有-d参数的myisamchk会生成下面类型的输出：

```
% myisamchk -d customer
MyISAM file:         customer
Record format:       Packed
Character set:       latin1 (8)
Data records:        3  Deleted blocks:          0
Recordlength:        75
table description:
```

```
Key Start Len Index   Type
1      2         4      unique   long
```

小结

MySQL有许多工具可以使对数据库服务器的管理尽可能得像一份不费力的任务。但是数据越重要，表就越大，而且当问题发生时，能够适当地、快速地控制问题就越重要。操作者可以通过多种方法启动和关闭MySQL服务器，例如直接使用mysqld，但是强烈建议操作者使用包装脚本，如带有分类的mysqld_safe脚本。

Windows和UNIX在自动启动方面有着非常不同的方法，但是一旦操作者了解了在做什么，那么二者就变得相当的容易了。

配置文件是控制服务器操作的一种灵活方法，在第13章“配置并优化MySQL”中，操作者将了解到它们更多的用于优化方面的内容。

在灾难发生时，无论是查询或其他无法预料的问题，日志文件在帮助确定问题方面可能是非常有用的。它们还有助于从备份中进行恢复操作。它们还可以帮助我们做一些平凡的工作，如在优化方面确定速度慢的查询。当然它们很快就可能无法控制，并且进入循环，以至于避免它们无限增长也是一项重要的工作。

对表的常规维护是避免问题发生的最好办法。表的优化操作可以对它们进行碎片重整，对于更新键的信息，以协助MySQL基于数据的最新信息作出查询连接的决定，也是很有用的。但是尽管作出了最大的努力，无法预测的电力故障、硬件故障或人为错误还是可能导致数据的损坏（特别是在索引中），这时各种修复参数是非常有用的。在服务器关闭时，较老的myisamchk工具更有用，并且mysqlcheck工具能够在服务器还在运行时执行维护工作（如同相关的SQL语句可以做到得那样）。

接下来的章节将更详细地探讨这些题目：数据库安全性、复制和配置。

第11章 数据库备份

对数据库来说，最重要也是最容易被忽略的事情之一就是备份。由于不可预测性，偶然的事件可能会带来惨重的损失。许多人都有这样的痛苦体会：由于推迟备份那些看上去暂时不会用到的数据，结果付出了数据完全丢失的代价。数据越是重要，数据变化越是频繁，数据的备份就越要经常进行。对于一个新闻数据库而言，其内容变化是经常发生的，对每日的工作记录进行备份是明智的。对于小型网站，其数据每周发生变化，则每周对数据进行备份是合理的。但无论大型数据库或小型数据库，备份都是不能避免的工作。可以说，备份是任何数据存储系统中至关重要的组成部份。

建议保留自己的配置文件（my.cnf或my.ini），像你调整服务器这样的工作也值得保留。如果发现配置文件丢失了，就要从失败的磁盘中做一次恢复工作，哪怕这很轻松，也不会让人感到愉快。

本章向你展示了几种用MySQL进行备份和恢复的方法。一旦知道了有关备份工作的复杂性和潜在价值，你将会在你所处的环境中更加好地执行最佳策略了。

本章要点：

- 备份和恢复命令
- 通过直接拷贝文件的方法进行备份
- mysqldump
- mysqlhotcopy
- 使用SELECT INTO备份
- 使用LOAD DATA INFILE来恢复
- 使用LOAD DATA LOCAL的安全问题
- 使用二进制更新记录
- 备份InnoDB表
- 用复制的方法备份

用BACKUP备份MyISAM表

一种最容易的备份方法是用BACKUP命令。这只有在使用MyISAM表时才起作用。其语句结构如下：

```
BACKUP TABLE tablename TO '/db_backup_path';
```

备份的路径需要完整给出，包含想保存数据的目录，而不仅仅是个文件名。该命令将生成以.frm（定义）和.MYD（数据）结尾的文件，但不产生.MYI（索引）文件。你可以在数据库恢复后重建索引。

在处理文件时，要注意文件权限。备份时，如果没有全部文件和目录的恰当权限，MySQL多数不会给出友好的错误信息以示警告。

在UNIX环境下使用BACKUP

接下来的例子是运行在UNIX环境下的，执行操作的用户是一位根目录用户（Windows环境下的例子看下一节）：

```
% cd /
% mkdir db_backups
```

该命令在根目录下创建了一个目录，这种情况可以把备份存放在子目录里。接着的命令是连接firstdb数据库和运行BACKUP命令：

```
% mysql firstdb
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13 to server version: 4.0.1-alpha-max-log
mysql> BACKUP TABLE sales TO '/db_backups';
+---------------+--------+----------+-----------------------------------------+
| Table         | Op     | Msg_type | Msg_text                                |
+---------------+--------+----------+-----------------------------------------+
| firstdb.sales | backup | error    | Failed copying .frm file: errno = 13 |
| firstdb.sales | backup | status   | Operation failed                        |
+---------------+--------+----------+-----------------------------------------+
```

这个例子的问题是MySQL没有把文件写入/db_backups目录的权限。需要在退出MySQL后，在命令行中，设置mysql用户成为该目录的所有者：

```
mysql> exit
Bye
% chown mysql db_backups/
```

警告：需要有正确的权限才能做这个操作。确信作为用户你有这个恰当权限。在这个例子中，由于在根目录，所以没有问题。但如果不是根目录用户，可能要从系统管理员那里获得帮助。

现在，BACKUP命令可以正确运行了：

```
% mysql firstdb;
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15 to server version: 4.0.1-alpha-max-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> BACKUP TABLE sales TO '/db_backups';
+---------------+--------+----------+----------+
| Table         | Op     | Msg_type | Msg_text |
+---------------+--------+----------+----------+
| firstdb.sales | backup | status   | OK       |
+---------------+--------+----------+----------+
1 row in set (0.00 sec)
mysql> exit
Bye
```

这次备份操作获得了成功。再次退到命令行，便能看到新建立的文件。

```
% ls -l db_backups/
total 10
-rw-rw----   1 mysql    mysql         136 May 26 14:07 sales.MYD
-rw-rw----   1 mysql    mysql        8634 May 26 14:07 sales.frm
```

有两个新建的文件。

提示： 如果用这种方法备份遇到任何问题，那很可能是由于文件权限的原因。如果无法使用创建的文件或拿不准原因，要去请求系统管理员的帮助。另外，记住BACKUP目前只对MyISAM表起作用（查一下最新的文献，也许你读到此处时，情况已经不同了）。

为确保被备份的表格间的关联性，BACKUP命令会设置一个解读锁。

备份时，你可以列出多个表的名称，将这些表一次备份下来：

```
mysql> BACKUP TABLE sales,sales_rep,customer TO '/db_backups';
+--------------------+--------+----------+----------+
| Table              | Op     | Msg_type | Msg_text |
+--------------------+--------+----------+----------+
| firstdb.sales      | backup | status   | OK       |
| firstdb.sales_rep  | backup | status   | OK       |
| firstdb.customer   | backup | status   | OK       |
+--------------------+--------+----------+----------+
3 rows in set (0.05 sec)
```

一个表格一次被设置一把锁，如上所示，先设sales表，然后在sales表被备份后，再设sales_rep表，以此类推。这样做是考虑到那些相关联的表。但如果想同时也得到所有相关联表的简要描述，则必须为这些表设置你自己的锁：

```
mysql> LOCK TABLES customer READ,sales READ,sales_rep READ;
Query OK, 0 rows affected (0.00 sec)

mysql> BACKUP TABLE sales,sales_rep,customer TO '/db_backups';
+--------------------+--------+----------+----------+
| Table              | Op     | Msg_type | Msg_text |
+--------------------+--------+----------+----------+
| firstdb.sales      | backup | status   | OK       |
| firstdb.sales_rep  | backup | status   | OK       |
| firstdb.customer   | backup | status   | OK       |
+--------------------+--------+----------+----------+
3 rows in set (0.00 sec)
```

注意，不能单个地锁住这些表：

```
mysql> LOCK TABLE sales READ;
Query OK, 0 rows affected (0.00 sec)

mysql> LOCK TABLE sales_rep READ;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> LOCK TABLE customer READ;
Query OK, 0 rows affected (0.00 sec)

mysql> BACKUP TABLE sales,sales_rep,customer TO '/db_backups';
+--------------------+--------+----------
+-----------------------------------------------------+
| Table              | Op     | Msg_type | Msg_t      |
+--------------------+--------+----------
+-----------------------------------------------------+
   firstdb.sales     | backup | error
| Table 'sales' was not locked with LOCK TABLES       |
| firstdb.sales_rep | backup | error
| Table 'sales_rep' was not locked with LOCK TABLES |
| firstdb.customer  | backup | status
| OK                                                  |
+--------------------+--------+----------
+-----------------------------------------------------+
3 rows in set (0.00 sec)
```

LOCK TABLE自动释放所有被同一线索贯穿的锁，因此备份时，只有customer表的锁仍然保持。

说明：为了能锁住一个表，需要为你的表获得LOCK TABLES特权和SELECT特权。

```
mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.00 sec)
```

Windows环境下使用BACKUP

即使不使用UNIX，也应该先阅读前面的例子，因为它解释了BACKUP语句的一些概念。下面的例子处理的是专门针对Windows的问题。Windows虽没有有关文件权限带来的复杂性，但有其自己的问题。看一下这个例子，看能否发现是什么原因导致的错误：

```
C:\MySQL\bin>cd \
C:\>mkdir db_backups
C:\>mysql firstdb
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 4.0.1-alpha-max

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> BACKUP TABLE sales TO 'c:\db_backups';
+---------------+--------+----------+-----------------------------------+
  Table         | Op     | Msg_type | Msg_text                          |
+---------------+--------+----------+-----------------------------------+
| firstdb.sales | backup | error    | Failed copying .frm file: errno = 2 |
  firstdb.sales | backup | status   | Operation failed                  |
+---------------+--------+----------+-----------------------------------+
2 rows in set (0.33 sec)
```

不幸的是，这个错误信息给得不太明确。问题出在反斜线符（\），反斜线是MySQL的换码符，被用来避开类似单双引号一类的特殊字符。为了在Windows中用反斜线表示路径，需要加上另外一个换码符：

```
mysql> BACKUP TABLE sales TO 'c:\\db_backups';
+---------------+--------+----------+----------+
| Table         | Op     | Msg_type | Msg_text |
+---------------+--------+----------+----------+
| firstdb.sales | backup | status   | OK       |
+---------------+--------+----------+----------+
1 row in set (0.55 sec)
```

用RESTORE恢复MyISAM表

BACKUP的相反过程是RESTORE，用来恢复以前使用BACKUP创建的MyISAM表。该命令也重建索引，对于大型表而言，这会花些时间。语法如下：

```
RESTORE TABLE tablename FROM '/db_backup_path'
```

你不能恢复已存在的表，如果试图恢复早些时候备份的sales表，将得到以下信息：

```
mysql> RESTORE TABLE sales FROM '/db_backups';
+-------+---------+----------+-----------------------------------------+
| Table | Op      | Msg_type | Msg_text                                |
+-------+---------+----------+-----------------------------------------+
| sales | restore | error    | table exists, will not overwrite on restore |
+-------+---------+----------+-----------------------------------------+
1 row in set (0.01 sec)
```

这条失败信息至少清楚地表明前面的备份是成功的。

为了测试这个备份，必须采取冒险的行动，中途退出sales表。

警告： 这样做也许很直接，但请不要退出原始表去测试已激活的表。可以去试着恢复其他数据库或服务器。如果这里提及得太晚了，你的恢复工作失败了，请不要说你是从这本书学到的。

现在DROP并试着RESTORE这个表：

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

mysql> RESTORE TABLE sales FROM 'db_backups';
+-------+---------+----------+-------------------------+
| Table | Op      | Msg_type | Msg_text                |
+-------+---------+----------+-------------------------+
| sales | restore | error    | Failed copying .frm file |
+-------+---------+----------+-------------------------+
1 row in set (0.01 sec)
```

别慌。你在前面的代码中看到过这个错误吗？因为写的路径不对。当有些事情不对时，慌乱是你最大的敌人之一。在看到过去危机情况下的结果后，就不会惊叫着指责MySQL是个骗人的软件。有一个简单的原因可以解释为何不起作用，比如前面的打字错误。正像UNIX的例子所展示的，正确的路径将会正确地恢复这个表：

```
mysql> RESTORE TABLE sales FROM '/db_backups';
+---------------+---------+----------+----------+
| Table         | Op      | Msg_type | Msg_text |
+---------------+---------+----------+----------+
| firstdb.sales | restore | status   | OK       |
+---------------+---------+----------+----------+
1 row in set (0.00 sec)
```

下面是Windows下的正确表述：

```
mysql> RESTORE TABLE sales FROM 'c:\\db_backups';
+---------------+---------+----------+----------+
| Table         | Op      | Msg_type | Msg_text |
+---------------+---------+----------+----------+
| firstdb.sales | restore | status   | OK       |
+---------------+---------+----------+----------+
1 row in set (0.66 sec)
```

仅是为了安抚一下你的过度猜疑，让我们看看sales表是否真的正确恢复了：

```
mysql> SELECT * FROM sales;
+------+-----------+------+-------+
| code | sales_rep |   id | value |
+------+-----------+------+-------+
|    1 |         1 |    1 |  2000 |
|    2 |         4 |    3 |   250 |
|    3 |         2 |    3 |   500 |
|    4 |         1 |    4 |   450 |
|    5 |         3 |    1 |  3800 |
|    6 |         1 |    2 |   500 |
|    7 |         2 | NULL |   670 |
|    8 |         3 |    3 |  1000 |
+------+-----------+------+-------+
8 rows in set (0.00 sec)
```

通过直接拷贝文件的方法备份MyISAM表

MyISAM表被以文件的形式存在以数据库名字命名的目录中（.frm代表定义，.MYD代表数据，.MYI代表索引），所以备份数据的一个简便方法就是拷贝这些文件。不像BACKUP，直接拷贝文件不会自动给这些表上锁。因此，为了得到相关联的图形，你需要自己锁住这些表格。另一种选择是，在服务器关闭时直接拷贝。一旦表被锁住，需要清查这些

表，以确保任何未写下的索引都被写入了磁盘。对目前这个例子，需要打开两个窗口。

来自窗口1中的LOCK和FLUSH表：

```
mysql> LOCK TABLES sales READ,sales_rep READ,customer READ;
Query OK, 0 rows affected (0.00 sec)
mysql> FLUSH TABLES sales,sales_rep,customer;
Query OK, 0 rows affected (0.02 sec)
```

现在从窗口2中拷贝表（UNIX下）：

```
% cd /usr/local/mysql/data/firstdb
% cp sales.* /db_backups
% cp sales_rep.* /db_backups
% cp customer.* /db_backups
%
```

Windows下为（从窗口2）：

```
C:\MySQL\data\firstdb>copy customer.* c:\db_backup
customer.frm
customer.MYI
customer.MYD
        3 file(s) copied
C:\MySQL\data\firstdb>copy sales.* c:\db_backup
sales.frm
sales.MYI
sales.MYD
        3 file(s) copied
C:\MySQL\data\firstdb>copy sales_rep.* c:\db_backup
sales_rep.frm
sales_rep.MYI
sales_rep.MYD
        3 file(s) copied
```

一旦完成拷贝文件，就可以从窗口1中将锁释放，操作如下：

```
mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.00 sec)
```

警告： 在备份期间，锁存在时，不能在表中加入新的记录，而且还要经历读取时烦人的磨难。如果可能的话，不要在高峰时间进行备份。

再次重复，要测试备份，要退出你的表。

从窗口1退出表：

```
mysql> DROP TABLE sales;
Query OK, 0 rows affected (0.00 sec)
```

从窗口2拷贝表（这是UNIX下的例子）：

```
% cp /db_backups/sales.* .
```

我们的表被恢复了。可以在窗口2中用如下方法验证：

```
mysql> SELECT * FROM sales;
+------+-----------+------+-------+
| code | sales_rep |   id | value |
+------+-----------+------+-------+
|    1 |         1 |    1 |  2000 |
|    2 |         4 |    3 |   250 |
|    3 |         2 |    3 |   500 |
|    4 |         1 |    4 |   450 |
|    5 |         3 |    1 |  3800 |
|    6 |         1 |    2 |   500 |
|    7 |         2 | NULL |   670 |
|    8 |         3 |    3 |  1000 |
+------+-----------+------+-------+
8 rows in set (0.00 sec)
```

UNIX的权限问题也有可能再度困扰你。如果没有作为mysql用户来备份文件（通常要以根目录用户的身份完成这一工作），你可能看到下面的信息：

```
mysql> SELECT * FROM sales;
ERROR 1017: Can't find file: './firstdb/sales.frm' (errno: 13)
```

问题出在已经把文件拷贝回来，但mysql用户不能打开这个文件。下面窗口1中的片断说明是以根目录用户的身份备份了文件。

```
[root@test firstdb]# ls -l
total 183
...
-rw-r-----   1 root     root         153 May 27 22:27 sales.MYD
-rw-r-----   1 root     root        3072 May 27 22:27 sales.MYI
-rw-r-----   1 root     root        8634 May 27 22:27 sales.frm
-rw-rw----   1 mysql    mysql        156 May 22 21:50 sales_rep.MYD
-rw-rw----   1 mysql    mysql       3072 May 22 21:50 sales_rep.MYI
-rw-rw----   1 mysql    mysql       8748 May 22 21:50 sales_rep.frm
...
```

为了恢复对mysql的权限，需从窗口1中将sales表的控制权换到mysql：

```
% chown mysql sales.*
%
```

现在一切应该工作正常，在窗口2中运行查询，就能看见如下信息：

```
mysql> SELECT * FROM sales;
+------+-----------+------+-------+
| code | sales_rep |   id | value |
+------+-----------+------+-------+
|    1 |         1 |    1 |  2000 |
```

```
|    2 |          4 |    3 |   250 |
|    3 |          2 |    3 |   500 |
|    4 |          1 |    4 |   450 |
|    5 |          3 |    1 |  3800 |
|    6 |          1 |    2 |   500 |
|    7 |          2 |  NULL |   670 |
|    8 |          3 |    3 |  1000 |
+------+------------+------+-------+
8 rows in set (0.00 sec)
```

MyISAM表是独立于平台的，因此这些表可以转移到具有不同硬件或操作系统机器上的MySQL中。旧的MySQL第三版建立的MyISAM表也能用在第四版上，但反过来不行。为把数据从MySQL的第四版移到第三版，需要用mysqldump的参数，这会在下一节讨论。

用mysqldump备份

前两种直接拷贝文件的方法只在使用MyISAM表的时候起作用（对InnnoDB表来说，由于其不能以文件的形式保存，因此前面的方法无法使用）。Mysqldump是另外一种方法，它为那些用于创建表备份的SQL语句产生一个的转储的地方。这也潜在地允许数据库可以接口到其他数据库上（注意：不是所有MySQL的特点都是其他数据库可接受的标准。如果不得不转移到另一个DBMS上，会有一个漫长而费力的工作等着你）。

为了备份客户表，在UNIX集成器中运行下列指令：

```
% mysqldump firstdb customer > /db_backups/customer_2002_11_12.sql
```

或在Windows命令行中运行下列命令：

```
C:\MySQL\bin>mysqldump firstdb customer > c:\db_backups\customer_2002_11_12.sql

C:\MySQL\bin>
```

记住，如果需要的话，请指出路径、用户名和密码。这将在db_backups目录中创建一个文件，其中包含了重建客户表的SQL语句。可以在任何文本编辑器中，如NotePad或vi中看到这个文件。文件的第一部分包含如下内容：

```
# MySQL dump 8.14
#
# Host: localhost        Database: firstdb
#--------------------------------------------------------
# Server version         4.0.1-alpha-max-log
```

井号后的内容只是评论，有关版本的说明等杂乱信息。在这之后，文件中是重建不同表的重要SQL语句。片断显示的是哪些用户可以重建客户表：

```
#
# Table structure for table 'customer'
#
```

```
CREATE TABLE customer (
  id int(11) NOT NULL auto_increment,
  first_name varchar(30) default NULL,
  surname varchar(40) default NULL,
  initial varchar(5) default NULL,
  PRIMARY KEY  (id),
  KEY surname (surname,initial,first_name)
) TYPE=MyISAM;

#
# Dumping data for table 'customer'
#

INSERT INTO customer VALUES (1,'Yvonne','Clegg','X');
INSERT INTO customer VALUES (2,'Johnny','Chaka-Chaka','B');
INSERT INTO customer VALUES (3,'Winston','Powers','M');
INSERT INTO customer VALUES (4,'Patricia','Mankunku','C');
INSERT INTO customer VALUES (5,'Francois','Papo','P');
INSERT INTO customer VALUES (7,'Winnie','Dlamini',NULL);
INSERT INTO customer VALUES (6,'Neil','Beneke',NULL);
INSERT INTO customer VALUES (10,'Breyton','Tshabalala','B');
```

警告： 使用mysqldump的默认结果恢复数据库是很费时间的。因为索引键存器在每个INSERT语句后都被刷新，所以大的表要花很长时间才能恢复。想知道如何加快恢复速度，请看下一节的mysqldump参数。

恢复用mysqldump备份的数据库

能通过退出和重建数据的方法测试备份数据：

```
mysql> DROP TABLE customer;
Query OK, 0 rows affected (0.31 sec)
mysql> exit
Bye
```

警告： 再次警告，不要用这种方法测试已激活的数据库备份。这样做只能是一种模拟数据库受到损失的方法。

为了恢复UNIX机器上的表：

```
% mysql firstdb < /db_backups/customer_2002_11_12.sql
```

或从Windows中恢复：

```
C:\MySQL\bin>mysql firstdb < c:\db_backups\customer_2002_11_12.sql
```

这个表已被恢复了。

表11.1描述了mysqldump可用的不同参数。

表11.1 mysqldump参数

参数	描述
--add-locks	把LOCK TABLES语句放在每一个表转存处之前，而把UNLOCK TABLE语句放在每一个表转存处之后。这将使INSERT语句的执行速度大大加快（因为关键的缓存器只在UNLOCK TABLE语句之后刷新）
--add-drop-table	在每个CREATE TABLE语句之前，加入DROP TABLE语句。如果表已经存在，它会干扰恢复过程，因此要确保被恢复表的清白历史
-A, --all-databases	转储所有存在的数据库。它相当于所有列出数据库的-B或--databases参数
-a, --all	包括所有MySQL特有的CREATE参数
--allow-keywords	通常列的名称不能与关键字相同。此命令可以用关键字创建列名，方法是让每一个列名以表名开头
-c, --complete-insert	使用完整地插入语句——也就是使用INSERT INTO tablename (x,y,z) VALUES (a,b,c)，而不是INSERT INTO tablename VALUES (a,b,c)
-C, --compress	压缩在客户端和服务器之间传送的数据，如果两者支持压缩的话
-B, --databases	转储几个数据库。如果该参数被选中，无需指定表名。数据库中所有表都被转储。其输出是在每一个新数据库前放一个USE databasename语句
--delayed	用INSERT DELAYED而不是INSERT插入行
-e, --extended-insert	利用多行INSERT语法结构。其输出更紧凑，运行也更快，这是因为索引缓存器只在每个INSERT命令后刷新
-#, --debug[=option_string]	跟踪程序使用，用于调试
--help	显示帮助信息和退出
--fields-terminated-by=...	与LOAD DATA INFILE参数一样。参看介绍有关SELECT INTO和LOAD DATA INFILE的章节
--fields-enclosed-by=...	与LOAD DATA INFILE参数一样。参看介绍有关SELECT INTO和LOAD DATA INFILE的章节
--fields-optionally-enclosed-by=...	与LOAD DATA INFILE参数一样。参看介绍有关SELECTINTO和LOAD DATA INFILE的章节
--fields-escaped-by=...	与LOAD DATA INFILE参数一样。参看介绍有关SELECT INTO和LOAD DATA INFILE的章节
--lines-terminated-by=...	与LOAD DATA INFILE参数一样。参看介绍有关SELECT INTO和LOAD DATA INFILE的章节
-F, --flush-logs	在开始转储前，清除日志文件
-f, --force	即使转储期间MySQL出错，也继续进行

（续表）

参数	描述
-h, --host=...	在指定的主机上，从MySQL服务器中转储数据。默认的主机是localhost
-l, --lock-tables	在开始转储前锁定所有的表。表被READ LOCAL锁住，该命令允许对MyISAM表同时进行插入操作
-K, --disable-keys	在INSERT语句前使索引无效，之后有效。这使插入操作速度更快
-n, --no-create-db	CREATE DATABASE语句将不能被放在输出中。如果--databases或--all-databases参数被选用，则可以
-t, --no-create-info	不包括CREATE TABLE语句，因此表被认为已经退出
-d, --no-data	仅卸载表的结构，不包括表的任何INSERT语句
--opt	等效于--quick --add-drop-table --add-locks --extended-insert --lock-tables。这是最快的恢复
-ppassphrase, --password[=passphrase]	当连接服务器时，指明要用的密码。通常，如果不指明密码，你会得到提示，这是一个较安全的参数
-P portnumber, --port=portnumber	当连接主机时，要求指明TCP/IP端口。若连接的是localhost，则不需要指明。参看-s参数
-q, --quick	不对查询进行暂存，但直接转储到stdout。用mysql_use_result()去做。对于大的表，必须这样做
-Q, --quote-names	把表和列的名放入单引号里
-r, --result-file=...	输出到指定文件。这在DOS中有用，因为它可以阻止UNIX的新行符\n字符被转换为换行和回车\n\r
-S /path/to/socket, --socket=/path/to/socket	当连接到localhost（默认）时，指明要用的套接字文件
--tables	使-B或--databases参数无效
-T, --tab=path-to-some-directory	为每一个表创建两个文件：包含CREATE语句的tablename.sql和包含数据的tablename.txt文件。该参数只有mysqldump运行在服务器上时才起作用
-u username, --user= username	当连接服务器时，指明要用的MySQL用户名。默认值是UNIX登录名
-O var=option, --set-variable var=option	设置变量值
-v, --verbose	在进行mysqldump过程中，迫使MySQL显示更多的信息，使其变得更“健谈”
-V, --version	显示版本信息和退出方法
-w, --where='where-condition'	只转储满足where条件的记录。条件必须出现在引号里

（续表）

参数	描述
-X, --xml	以符合文法的XML形式转储数据库
-x, --first-slave	锁定所有数据库中的所有表
-O net_buffer_length=n	当创建多行插入语句时，该参数创建行数的大小最多为n（-e或--opt参数，n必须小于16MB，mysqld变量max_allowed_packet必须大于n）

有三种主要的方式来使用mysqldump。

```
% mysqldump [OPTIONS] database [tables]
```

或：

```
% mysqldump [OPTIONS] --databases [OPTIONS] DB1 [DB2 DB3...]
```

或：

```
% mysqldump [OPTIONS] --all-databases [OPTIONS]
```

下面的例子中演示了一些可用的参数。特别是下面这个例子，转储了firstdb数据库中的所有表：

```
% mysqldump firstdb > /db_backups/firstdb_2002_11-12.sql
```

在使用中，参数-v会给出更多信息。当程序有问题时，可以用于调试。

```
% mysqldump -v firstdb customer > /db_backups/customer_2002_11-12.sql
# Connecting to localhost...
# Retrieving table structure for table customer...
# Sending SELECT query...
# Retrieving rows...
# Disconnecting from localhost...
%
```

下面的例子用于只备份那些id>5的记录：

```
% mysqldump --where='id>5' firstdb customer > /db_backups/customer_2002_11-12.sql
```

产生的输出结果如下：

```
#
# Dumping data for table 'customer'
# WHERE:  id>5
#

INSERT INTO customer VALUES (7,'Winnie','Dlamini',NULL);
INSERT INTO customer VALUES (6,'Neil','Beneke',NULL);
INSERT INTO customer VALUES (10,'Breyton','Tshabalala','B');
```

-e（扩展）选项允许更快的插入操作：

```
% mysqldump -e firstdb customer > /db_backups/customer_2002_11-12.sql
```

这里使用了多行INSERT语句，正像从文本文件中看到的那样：

```
#
# Dumping data for table 'customer'
#

INSERT INTO customer VALUES
(1,'Yvonne','Clegg','X'),
(2,'Johnny','Chaka-Chaka','B'),
(3,'Winston','Powers','M'),
(4,'Patricia','Mankunku','C'),
(5,'Francois','Papo','P'),
(7,'Winnie','Dlamini',NULL),
(6,'Neil','Beneke',NULL),(10,'Breyton','Tshabalala','B');
```

因为只有一个INSERT语句，索引缓冲器只刷新一次，这比每次插入后都刷新的速度快。

用SELECT INTO做备份

另一种备份的方法是使用SELECT INTO。这与mysqldump相似，也创建一个可用于重建已备份表的文件。它与LOAD DATA INTO语句的用法相反。结果文件只能被建立在MySQL服务器上，而不是任何其他主机。其语句结构如下：

```
SELECT ... INTO OUTFILE 'path_and_filename'
```

任何有效的SELECT都可用于创建文件。

为了建立customer表的备份，应该首先在UNIX环境中做如下事情：

```
mysql> SELECT * FROM customer INTO OUTFILE '/db_backups/customer.dat';
Query OK, 8 rows affected (0.00 sec)
```

而在Windows下则为：

```
mysql> SELECT * FROM customer INTO OUTFILE 'c:\\db_backups\\bdb.dat';
Query OK, 8 rows affected (0.33 sec)
```

再次强调，做这件事时，必须小心。下面的错误是常见的：

```
mysql> SELECT * FROM customer INTO OUTFILE '/db_backups/customer.dat';
ERROR 1086: File '/db_backups/customer.dat' already exists
```

不能重写已存在的文件（这是一种安全措施，因为一个恶劣的安装系统可能允许关键文件被重写）。

在Windows下，还会有另一个常见错误：

```
mysql> SELECT * FROM customer INTO OUTFILE 'c:\db_backups\customer.dat';
ERROR 1: Can't create/write to file 'c:db_backupscustomer.dat' (Errcode: 2)
```

这里的错误信息很清楚：MySQL不能往这个目录中写入，因为忘了换码符\。在Windows中，\是换码符，但也是Windows路径的一部份，因此在上下文中要换码。在UNIX中，也有类似的常见错误：

```
mysql> SELECT * FROM customer INTO OUTFILE '\db_backups\customer.dat';
Query OK, 8 rows affected (0.18 sec)
```

在该例子中，MySQL没有警告出现错误。一些Windows用户容易犯不正确地使用斜线的错误，结果得不到他们想要得到的备份。在备份被创建后，永远不要忘记进行检查。

用任何文本编辑器（如VI或Notepad）看一下，将看到下面的信息：

```
1       Yvonne  Clegg   X
2       Johnny  Chaka-Chaka     B
3       Winston Powers  M
4       Patricia        Mankunku        C
5       Francois        Papo    P
7       Winnie  Dlamini \N
6       Neil    Beneke  \N
10      Breyton Tshabalala      B
```

跳格把域分开，换行则把记录分开，这与使用LOAD DATA INTO的默认设置相同。你可以通过在语句最后加入参数来改变这些默认设置。SELECT INTO（和LOAD DATA INTO ）的全部参数设置如下：

```
[FIELDS
        [TERMINATED BY '\t']
        [[OPTIONALLY] ENCLOSED BY '']
        [ESCAPED BY '\\' ]
    ]
    [LINES TERMINATED BY '\n']
```

这里是使用SELECT INTO时，利用非默认参数的一些例子，其结果文本归档如下：

```
mysql> SELECT * FROM customer INTO OUTFILE '/db_backups/customer2.dat'
 FIELDS TERMINATED BY 'zz';
Query OK, 8 rows affected (0.00 sec)
```

数据看上去像下面的情况：

```
1zzYvonnezzCleggzzX
2zzJohnnyzzChaka-ChakazzB
3zzWinstonzzPowerszzM
4zzPatriciazzMankunkuzzC
5zzFrancoiszzPapozzP
7zzWinniezzDlaminizz\N
6zzNeilzzBenekezz\N
10zzBreytonzzTshabalalazzB
```

在每个域之间，默认的制表符被字符zz所代替。

警告： 字符zz用在这里只是为了进行说明。用普通字符做分界标识是危险的。如果文本含有zzz这样的词，所有域将不会再整齐排列，因为MySQL认为前两个字是分界标识。所以请用传统字符，如跳格、换行或边（|）作为你的分界标识。

下面的语句建立一个很长的行：

```
mysql> SELECT * FROM customer INTO OUTFILE '/db_backups/customer3.dat'
 FIELDS TERMINATED BY '|' LINES TERMINATED BY '[end]';
Query OK, 8 rows affected (0.00 sec)
```

数据显示如下：

```
1|Yvonne|Clegg|X[end]2|Johnny|Chaka-Chaka|B[end]➥
3|Winston|Powers|M[end]4|Patricia|Mankunku C[end]➥
5|Francois|Papo|P[end]7|Winnie|Dlamini|\N[end]➥
6|Neil|Beneke|\N[end]10|Breyton|Tshabalala|B[end]
```

长行的中断被字符[end]代替。

在下面的例子，ENCLOSED关键字用指定的字符包围了所有的域：

```
mysql> SELECT * FROM customer INTO OUTFILE '/db_backups/customer4.dat'
 FIELDS TERMINATED BY '|' ENCLOSED BY '"' LINES TERMINATED BY '\n';
Query OK, 8 rows affected (0.00 sec)
```

数据显示如下：

```
"1"|"Yvonne"|"Clegg"|"X"
"2"|"Johnny"|"Chaka-Chaka"|"B"
"3"|"Winston"|"Powers"|"M"
"4"|"Patricia"|"Mankunku"|"C"
"5"|"Francois"|"Papo"|"P"
"7"|"Winnie"|"Dlamini"|\N
"6"|"Neil"|"Beneke"|\N
"10"|"Breyton"|"Tshabalala"|"B"
```

OPTIONALLY关键字导致了只有字符域被包括（正像加入记录时一样，用单引号包括字符域而不是数字域），例如：

```
mysql> SELECT * FROM customer INTO OUTFILE '/db_backups/customer5.dat'
 FIELDS TERMINATED BY '|' OPTIONALLY ENCLOSED BY '"' LINES TERMINATED
 BY '\n';
Query OK, 8 rows affected (0.00 sec)
```

数据的第一个域（类型为INT）没有被引号括起：

```
1|"Yvonne"|"Clegg"|"X"
2|"Johnny"|"Chaka-Chaka"|"B"
3|"Winston"|"Powers"|"M"
4|"Patricia"|"Mankunku"|"C"
5|"Francois"|"Papo"|"P"
7|"Winnie"|"Dlamini"|\N
6|"Neil"|"Beneke"|\N
10|"Breyton"|"Tshabalala"|"B"
```

你也能用SELECT语句中的一个条件备份数据的子集：

```
mysql> SELECT * FROM customer WHERE id<10 INTO OUTFILE
 '/db_backups/customer6.dat' FIELDS TERMINATED BY '|' LINES TERMINATED
 BY '\n';
Query OK, 7 rows affected (0.01 sec)
```

在这个文件中，只出现了七个可用记录：

```
1|Yvonne|Clegg|X
2|Johnny|Chaka-Chaka|B
3|Winston|Powers|M
4|Patricia|Mankunku|C
5|Francois|Papo|P
7|Winnie|Dlamini|\N
6|Neil|Beneke|\N
```

用LOAD DATA恢复表

为了恢复由SELECT INTO建立的表，要使用LOAD DATA语句。也可以用LOAD DATA添加由其他方法建立的数据，如某个应用程序或电子数据表。这是最快的添加数据的方法，特别是对于大量数据。语法如下：

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'filename'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [FIELDS
        [TERMINATED BY '\t']
        [[OPTIONALLY] ENCLOSED BY '']
        [ESCAPED BY '\\' ]
    ]
    [LINES TERMINATED BY '\n']
    [IGNORE number LINES]
    [(col_name,...)]
```

让我们从客户表中删除数据并用LOAD DATA恢复它：

```
mysql> TRUNCATE customer;
Query OK, 0 rows affected (0.02 sec)
```

在UNIX下恢复数据用：

```
mysql> LOAD DATA INFILE '/db_backups/customer.dat' INTO TABLE customer;
Query OK, 8 rows affected (0.01 sec)
Records: 8  Deleted: 0  Skipped: 0  Warnings: 0
```

在Windows下恢复数据用：

```
mysql> LOAD DATA INFILE 'c:\\db_backups\\customer.dat' INTO TABLE customer;
Query OK, 8 rows affected (0.01 sec)
Records: 8  Deleted: 0  Skipped: 0  Warnings: 0
```

正如你所见，数据被成功恢复了：

```
mysql> SELECT * FROM customer;
+----+------------+-------------+---------+
| id | first_name | surname     | initial |
+----+------------+-------------+---------+
|  1 | Yvonne     | Clegg       | X       |
|  2 | Johnny     | Chaka-Chaka | B       |
|  3 | Winston    | Powers      | M       |
|  4 | Patricia   | Mankunku    | C       |
|  5 | Francois   | Papo        | P       |
|  7 | Winnie     | Dlamini     | NULL    |
|  6 | Neil       | Beneke      | NULL    |
| 10 | Breyton    | Tshabalala  | B       |
+----+------------+-------------+---------+
8 rows in set (0.00 sec)
```

如果出错怎么办

出错的原因有下面几种：

- 如果不能成功使用LOAD DATA，也许是因为没有权限读取服务器上的文件。使用LOAD DATA的用户需要FILE特权（见第14章“数据库安全”一节），且文件需要在数据库中存在或完全可读。
- 一个普遍的错误是分界标识和包含符不匹配。在数据文件中（或在SELECT INTO语句中被指定时），它们必须保持一致。如果不这样，其结果是：看上去运行正常，但表却总是空的或全是NULL值。要获取更多信息，看下一章。
- 如果MySQL以--local-infile=0参数形式启动，用LOCAL关键字会不起作用（见第13章“配置和优化MySQL”一节）。
- 路径名和文件名错误（记住，在Windows路径名中使用换码符）。

使用带参数的LOAD DATA

让我们用一些其他的参数来恢复一些其他的备份：

```
mysql> LOAD DATA INFILE '/db_backups/customer2.dat' INTO TABLE customer;
Query OK, 8 rows affected (0.01 sec)
Records: 8  Deleted: 0  Skipped: 0  Warnings: 32
```

虽然这看上去管用，但数据没有被正确恢复：

```
mysql> SELECT * FROM customer;
+----+------------+---------+---------+
| id | first_name | surname | initial |
+----+------------+---------+---------+
|  1 | NULL       | NULL    | NULL    |
|  2 | NULL       | NULL    | NULL    |
```

```
|  3 | NULL       | NULL     | NULL     |
|  4 | NULL       | NULL     | NULL     |
|  5 | NULL       | NULL     | NULL     |
|  7 | NULL       | NULL     | NULL     |
|  6 | NULL       | NULL     | NULL     |
| 10 | NULL       | NULL     | NULL     |
+----+------------+----------+----------+
8 rows in set (0.00 sec)
```

问题出在分界标识没有正确匹配。记住：customer2.dat是用FIELDS TERMINATED BY 'zz'参数建立的。因此，要用同样的方法进行恢复：

```
mysql> TRUNCATE customer;
Query OK, 0 rows affected (0.00 sec)
mysql> LOAD DATA INFILE '/db_backups/customer2.dat' INTO TABLE customer
 FIELDS TERMINATED BY 'zz';
Query OK, 8 rows affected (0.01 sec)
Records: 8  Deleted: 0  Skipped: 0  Warnings: 0
mysql> SELECT * FROM customer;
+----+------------+-------------+---------+
| id | first_name | surname     | initial |
+----+------------+-------------+---------+
|  1 | Yvonne     | Clegg       | X       |
|  2 | Johnny     | Chaka-Chaka | B       |
|  3 | Winston    | Powers      | M       |
|  4 | Patricia   | Mankunku    | C       |
|  5 | Francois   | Papo        | P       |
|  7 | Winnie     | Dlamini     | NULL    |
|  6 | Neil       | Beneke      | NULL    |
| 10 | Breyton    | Tshabalala  | B       |
+----+------------+-------------+---------+
8 rows in set (0.00 sec)
```

同样适用于用LINES TERMINATED BY语句建立的customer3.dat：

```
mysql> TRUNCATE customer;
Query OK, 0 rows affected (0.00 sec)
mysql> LOAD DATA INFILE '/db_backups/customer3.dat' INTO TABLE customer
 FIELDS TERMINATED BY '|' LINES TERMINATED BY '[end]';
Query OK, 8 rows affected (0.00 sec)
Records: 8  Deleted: 0  Skipped: 0  Warnings: 0
mysql> SELECT * FROM customer;
+----+------------+-------------+---------+
| id | first_name | surname     | initial |
+----+------------+-------------+---------+
|  1 | Yvonne     | Clegg       | X       |
```

```
|  2 | Johnny     | Chaka-Chaka | B       |
|  3 | Winston    | Powers      | M       |
|  4 | Patricia   | Mankunku    | C       |
|  5 | Francois   | Papo        | P       |
|  7 | Winnie     | Dlamini     | NULL    |
|  6 | Neil       | Beneke      | NULL    |
| 10 | Breyton    | Tshabalala  | B       |
+----+------------+-------------+---------+
8 rows in set (0.00 sec)
```

如果customer4.dat用过ENCLOSED BY语句，该语句也要添加上：

```
mysql> TRUNCATE customer;
Query OK, 0 rows affected (0.00 sec)
mysql> LOAD DATA INFILE '/db_backups/customer4.dat' INTO TABLE customer
 FIELDS TERMINATED BY '|' ENCLOSED BY '"' LINES TERMINATED BY '\n';
Query OK, 8 rows affected (0.01 sec)
Records: 8  Deleted: 0  Skipped: 0  Warnings: 0
mysql> SELECT * FROM customer;
+----+------------+-------------+---------+
| id | first_name | surname     | initial |
+----+------------+-------------+---------+
|  1 | Yvonne     | Clegg       | X       |
|  2 | Johnny     | Chaka-Chaka | B       |
|  3 | Winston    | Powers      | M       |
|  4 | Patricia   | Mankunku    | C       |
|  5 | Francois   | Papo        | P       |
|  7 | Winnie     | Dlamini     | NULL    |
|  6 | Neil       | Beneke      | NULL    |
| 10 | Breyton    | Tshabalala  | B       |
+----+------------+-------------+---------+
8 rows in set (0.00 sec)
```

当然，同样也适用于建立customer5.dat的OPTIONALLY ENCLOSED语句：

```
mysql> TRUNCATE customer;
Query OK, 0 rows affected (0.00 sec)
mysql> LOAD DATA INFILE '/db_backups/customer5.dat' INTO TABLE customer
 FIELDS TERMINATED BY '|' OPTIONALLY ENCLOSED BY '"' LINES TERMINATED BY '\n';
Query OK, 8 rows affected (0.01 sec)
Records: 8  Deleted: 0  Skipped: 0  Warnings: 0
mysql> SELECT * FROM customer;
+----+------------+-------------+---------+
| id | first_name | surname     | initial |
+----+------------+-------------+---------+
|  1 | Yvonne     | Clegg       | X       |
|  2 | Johnny     | Chaka-Chaka | B       |
```

```
|  3 | Winston    | Powers      | M      |
|  4 | Patricia   | Mankunku    | C      |
|  5 | Francois   | Papo        | P      |
|  7 | Winnie     | Dlamini     | NULL   |
|  6 | Neil       | Beneke      | NULL   |
| 10 | Breyton    | Tshabalala  | B      |
+----+------------+-------------+--------+
8 rows in set (0.00 sec)
```

也能更新部份被备份的文件customer6.dat:

```
mysql> TRUNCATE customer;
Query OK, 0 rows affected (0.00 sec)
mysql> LOAD DATA INFILE '/db_backups/customer6.dat' INTO TABLE customer
 FIELDS TERMINATED BY '|' LINES TERMINATED BY '\n';
Query OK, 7 rows affected (0.01 sec)
Records: 7  Deleted: 0  Skipped: 0  Warnings: 0
mysql> SELECT * FROM customer;
+----+------------+-------------+---------+
| id | first_name | surname     | initial |
+----+------------+-------------+---------+
|  1 | Yvonne     | Clegg       | X       |
|  2 | Johnny     | Chaka-Chaka | B       |
|  3 | Winston    | Powers      | M       |
|  4 | Patricia   | Mankunku    | C       |
|  5 | Francois   | Papo        | P       |
|  7 | Winnie     | Dlamini     | NULL    |
|  6 | Neil       | Beneke      | NULL    |
+----+------------+-------------+---------+
7 rows in set (0.00 sec)
```

如果你认识到犯了错误并想恢复整个表格，该如何办？如果立即对包含整个备份的文件用LOAD DATA，会遇到下面的问题：

```
mysql> LOAD DATA INFILE '/db_backups/customer.dat' INTO
   TABLE customer;        ERROR 1062: Duplicate entry '1'
   for key 1                                  mysql> SELECT * FROM customer;
+----+------------+-------------+---------+
| id | first_name | surname     | initial |
+----+------------+-------------+---------+
|  1 | Yvonne     | Clegg       | X       |
|  2 | Johnny     | Chaka-Chaka | B       |
|  3 | Winston    | Powers      | M       |
|  4 | Patricia   | Mankunku    | C       |
|  5 | Francois   | Papo        | P       |
|  7 | Winnie     | Dlamini     | NULL    |
|  6 | Neil       | Beneke      | NULL    |
```

```
+----+------------+------------+---------+
7 rows in set (0.00 sec)
```

你得到了一个复制键的错误，文件在这里停止工作了。当然，可以简单地预先清除表，正像你已做过的那样。但如果试图恢复已包含记录的表，你可能不想移走数据再重新开始。此时，两个关键字REPLACE和IGNORE变得有用了。后者忽略任何行，以惟一的索引或主要的键复制已存在的行。因此，在这种情况下，IGNORE是有用的，当得知数据没有改变时，它可以让你不丢失数据而使其全部恢复：

```
mysql> LOAD DATA INFILE '/db_backups/customer.dat' IGNORE INTO TABLE customer;
Query OK, 1 row affected (0.00 sec)
Records: 8  Deleted: 0  Skipped: 7  Warnings: 0
```

能看到在8行数据中，有7个被跳过，只有缺少的记录被插入。对大型文件，这样做可以节省大量的时间，避免暂时无数据可用的麻烦。所有记录再次显示如下：

```
mysql> SELECT * FROM customer;
+----+------------+-------------+---------+
| id | first_name | surname     | initial |
+----+------------+-------------+---------+
|  1 | Yvonne     | Clegg       | X       |
|  2 | Johnny     | Chaka-Chaka | B       |
|  3 | Winston    | Powers      | M       |
|  4 | Patricia   | Mankunku    | C       |
|  5 | Francois   | Papo        | P       |
|  7 | Winnie     | Dlamini     | NULL    |
|  6 | Neil       | Beneke      | NULL    |
| 10 | Breyton    | Tshabalala  | B       |
+----+------------+-------------+---------+
8 rows in set (0.00 sec)
```

当想恢复磁盘上已经存在的记录且记录被更改过时，REPLACE关键字就排上用场了。为了演示这一点，假设你将犯一个大错：想更新一个记录时，却更新了所有记录，将它们的名字全换成了Fortune：

```
mysql> UPDATE customer SET surname='Fortune';
Query OK, 8 rows affected (0.00 sec)
Rows matched: 8  Changed: 8  Warnings: 0
```

在检查表的时候，你发现了错误！

```
mysql> SELECT * FROM customer;
+----+------------+---------+---------+
| id | first_name | surname | initial |
+----+------------+---------+---------+
|  1 | Yvonne     | Fortune | X       |
|  2 | Johnny     | Fortune | B       |
|  3 | Winston    | Fortune | M       |
```

```
|  4 | Patricia   | Fortune | C       |
|  5 | Francois   | Fortune | P       |
|  7 | Winnie     | Fortune | NULL    |
|  6 | Neil       | Fortune | NULL    |
| 10 | Breyton    | Fortune | B       |
+----+------------+---------+---------+
8 rows in set (0.00 sec)
```

现在，用REPLACE关键字恢复：

```
mysql> LOAD DATA INFILE '/db_backups/customer.dat' REPLACE INTO TABLE customer;
Query OK, 16 rows affected (0.00 sec)
Records: 8  Deleted: 8  Skipped: 0  Warnings: 0
mysql> SELECT * FROM customer;
+----+------------+-------------+---------+
| id | first_name | surname     | initial |
+----+------------+-------------+---------+
|  1 | Yvonne     | Clegg       | X       |
|  2 | Johnny     | Chaka-Chaka | B       |
|  3 | Winston    | Powers      | M       |
|  4 | Patricia   | Mankunku    | C       |
|  5 | Francois   | Papo        | P       |
|  7 | Winnie     | Dlamini     | NULL    |
|  6 | Neil       | Beneke      | NULL    |
| 10 | Breyton    | Tshabalala  | B       |
+----+------------+-------------+---------+
8 rows in set (0.00 sec)
```

LOAD DATA LOCAL参数允许将MySQL客户机上存在的文件内容上载到数据库服务器上。

LOW PRIORITY参数可以使新记录的添加工作在等到没有其他客户从表中读取数据时再进行（正像使用普通INSERT的作用）。

如果想让表可以读，CONCURRENT关键字是一种有用的方法。它允许使用其他途径读取MyISAM表（但会使LOAD DATA速度慢下来）。

使用LOAD DATA LOCAL的安全问题

从客户机恢复也许是方便的，但这会带来安全方面的风险。有些人可能会用LOAD DATA LOCAL读取那些与他们连接且可以访问的用户的任何文件。他们在加载完数据后，可以通过创建和读取一个表的方法实现这一目的。如果他们正像连接着网页服务器那样连接着用户，并有权进行查询，事情就会变得危险。默认状况下，MySQL允许使用LOAD DATA LOCAL。为了阻止这一点，禁用all LOAD DATA LOCAL，并可以用--local-infile=0参数启动MySQL服务器。也可以不用--enable-local-infile参数来汇编MySQL。

用mysqlimport代替LOAD DATA

LOAD DATA是从MySQL内部运行的，可以使用命令行的等价命令mysqlimport来代替它。语法结构如下：

```
% mysqlimport [options] databasename filename1 [filename2 ...]
```

许多参数都与LOAD DATA相当。输入数据的表由文件名决定。为了做到这一点，mysqlimport放弃了文件的扩展名，因此customer.dat被输入客户表。

用mysqlimport恢复数据如下：

```
mysql> SELECT * FROM customer;
+----+------------+-------------+---------+
| id | first_name | surname     | initial |
+----+------------+-------------+---------+
|  1 | Yvonne     | Clegg       | X       |
|  2 | Johnny     | Chaka-Chaka | B       |
|  3 | Winston    | Powers      | M       |
|  4 | Patricia   | Mankunku    | C       |
|  5 | Francois   | Papo        | P       |
|  7 | Winnie     | Dlamini     | NULL    |
|  6 | Neil       | Beneke      | NULL    |
| 10 | Breyton    | Tshabalala  | B       |
+----+------------+-------------+---------+
8 rows in set (0.00 sec)
mysql> TRUNCATE customer;
Query OK, 0 rows affected (0.01 sec)
mysql> exit
Bye
% mysqlimport firstdb /db_backups/customer.dat
firstdb.customer: Records: 8  Deleted: 0  Skipped: 0  Warnings: 0
[root@test data]# mysql firstdb;
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 4.0.1-alpha-max-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT * FROM customer;
+----+------------+-------------+---------+
| id | first_name | surname     | initial |
+----+------------+-------------+---------+
|  1 | Yvonne     | Clegg       | X       |
|  2 | Johnny     | Chaka-Chaka | B       |
|  3 | Winston    | Powers      | M       |
|  4 | Patricia   | Mankunku    | C       |
|  5 | Francois   | Papo        | P       |
|  7 | Winnie     | Dlamini     | NULL    |
```

```
|  6 | Neil       | Beneke      | NULL     |
| 10 | Breyton    | Tshabalala  | B        |
+----+------------+-------------+----------+
8 rows in set (0.00 sec)
```

数据被恢复了。

表11.2描述的是mysqlimport可用的参数。

表11.2 mysqlimport参数

参数	描述
-c, --columns=...	取逗号分割的域名列表为自变量。这些用于创建LOAD DATA INFILE命令
--character-sets-dir=name	告诉MySQL字符集所在的目录
-C, --compress	如果支持压缩，则压缩客户机和服务器之间传输的数据
-#, --debug[=option_string]	为了方便调试，显示跟踪程序的使用情况
-d, --delete	输入文本文件前，清空表
--fields-terminated-by=...	与LOAD DATA INFILE参数相同
--fields-enclosed-by=...	与LOAD DATA INFILE参数相同
--fields-optionally-enclosed-by=...	与LOAD DATA INFILE参数相同
--fields-escaped-by=...	与LOAD DATA INFILE参数相同
--lines-terminated-by=...	与LOAD DATA INFILE参数相同
-f, --force	即使在转储中MySQL有错，也继续运行
--help	显示帮助信息并退出
-h host_name, --host= host_name	输入数据到被发现的指定主机的MySQL服务器上。默认主机为localhost
-i, --ignore	忽略加入后导致复制键的错误的记录。通常它们导致错误并使进程停在原处
-l, --lock-tables	在处理任何文本文件之前，锁定所有准备写入的表，这可以保持表在服务器上的同步
-L, --local	从客户机读取输入文件。如果连接localhost（默认），文本文件被假设在服务器上
-ppassphrase, --password[=passphrase]	当连接服务器时，指明要使用的密码。通常，如果不能明确密码，将被提醒。这是一个更安全的参数
-P portnumber, --port=portnumber	当连接主机时，指明TCP/IP的端口号。这不适用于所连主机为localhost的情况。参见-S参数
-r, --replace	用导致复制键的添加记录替代相同键的原记录。通常这会导致错误并使进程在这一点停止

（续表）

参数	描述
-s, --silent	只有出错时，才显示输出
-S /path/to/socket, --socket=/path/to/socket	当连接localhost（默认）时，指明套接字文件
-u username, --user=username	当连接服务器时，指明要使用的用户名。默认值是UNIX的登录名
-v, --verbose	运行时显示更多的信息
-V, --version	显示版本信息并退出

使用mysqlhotcopy备份

为了使备份变得容易，mysqlhotcopy利用Perl脚本。它仍然为beta版（查最新的文献，看你读到这里时是否情况还如此），因此它可能不会在所有情况下都正常工作。它快速易用，通过锁定和刷新表把文件拷贝到指定的目录中（见表11.3所示）。它只能把文件拷贝到服务器上的其他地方。其语法如下：

```
% mysqlhotcopy databasename backup_directory_path
```

表11.3描述的是mysqlhotcopy可用的参数。

表11.3 mysqlhotcopy参数

参数	描述
-?, --help	显示屏幕帮助信息，并退出
-u, --user=#	连接服务器所需的用户名
-p, --password=#	连接服务器所需的密码
-P, --port=#	连接本地服务器所用的端口
-S, --socket=#	连接本地服务器所用的套接字
--allowold	如文件已存在，mysqlhotcopy通常会被异常中断。该参数添加_old到文件名中，并继续运行
--keepold	--allowold产生的重命名文件，通常在运行结束后被删除。该参数则会保留它们
--noindices	该参数在备份中不包含索引文件，这可以加快处理速度。在恢复文件之后，索引能用myisamchk -rq重建
--method=#	允许指明是否用cp或scp来拷贝文件
-q, --quiet	只有错误信息被输出
--debug	允许调试

（续表）

参数	描述
-n, --dryrun	输出信息但不运行
--regexp=#	拷贝所有名字与规则表达匹配的数据库
--suffix=#	给出被拷贝数据库名称的后缀
--checkpoint=#	在指定数据库表中插入检查点项
--flushlog	所有表被锁定时，刷新日志
--tmpdir=#	允许指定临时目录

mysqlhotcopy从客户和参数文件的mysqlhotcopy组中获得它的参数。

恢复用mysqlhotcopy完成的备份的方法，是直接在数据目录中替换这些文件，就好像自己做了直接拷贝一样。

为了运行mysqlhotcopy，有一些先决条件需要满足：

- 需要能在数据库服务器上运行的Perl语言。
- 为了运行，mysqlhotcopy依赖于下面的Perl的类别：Getopt::Long、Data::Dumper、File::Basename、File::Path、DBI和Sys::Hostname。
- 需要试图对其备份的目录的写入权限。
- 需要正备份的数据库的选择特权。
- 为了刷新表，需要重新调用特权。

使用二进制的更新日志文件，恢复数据库到最近的位置

二进制更新日志是恢复数据库到其损坏前最近地点的理想方法（看第10章“基本管理”）。二进制更新日志文件会记录所有数据库发生的变化。当MySQL以--log-bin参数启动时，二进制更新日志就可以使用了。可以用--log-bin = filename指定一个名字；否则默认名将是服务器名加上-bin。新的日志文件会在下列情况每次发生时建立：服务器重新启动、日志刷新、服务器刷新或达到最大容量（在max_bin_log_size中设置）。

在用mysqldump做了备份后，用--log-bin参数启动MySQL。当恢复的时候，恢复mysqldump文件，然后用二进制日志还原数据库到它最近的状态。

例如，让我们假设最后一次备份来自customer.dat文件，恢复其10个记录如下：

```
mysql> SELECT * FROM customer;
+----+------------+-------------+---------+
| id | first_name | surname     | initial |
+----+------------+-------------+---------+
|  1 | Yvonne     | Clegg       | X       |
|  2 | Johnny     | Chaka-Chaka | B       |
|  3 | Winston    | Powers      | M       |
```

```
|  4 | Patricia   | Mankunku    | C       |
|  5 | Francois   | Papo        | P       |
|  7 | Winnie     | Dlamini     | NULL    |
|  6 | Neil       | Beneke      | NULL    |
| 10 | Breyton    | Tshabalala  | B       |
+----+------------+-------------+---------+
8 rows in set (0.00 sec)
```

一旦你处在这个阶段（刚做了备份），若还没有二进制日志功能，那就在启动服务器时带着这一功能：

```
C:\MySQL\bin> mysqladmin shutdown
020601 23:59:01  mysqld ended
```

如果这一功能不存在，就把下面的参数放入你的my.cnf或my.ini文件，使二进制日志功能有效：

```
log-bin
```

现在重启服务器：

```
C:\MySQL\bin> mysqld-max
020602 18:58:21  InnoDB: Started
C:\MySQL\bin> mysql firstdb;
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 4.0.1-alpha-max-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> INSERT INTO customer VALUES(11,'Robin','McKenzie',NULL);
Query OK, 1 row affected (0.00 sec)
```

现在让我们通过停止服务器和删除客户数据及索引文件，来模拟一次意外中断：

```
mysql> exit
Bye
C:\MySQL\bin> del c:\MySQL\data\firstdb\customer.*
```

这取决于你的设置，可能在服务器关闭或换到根目录之前，没有权限来移走文件。

如果删除了文件，且保持正常连接，那就可以试着查询客户表，仍然会得到结果，因为结果可能还存在缓存中。但当关闭服务器且重启时，将不会发现任何客户数据：

```
C:\MySQL\bin> mysqladmin shutdown
020601 23:59:01  mysqld ended
C:\MySQL\bin> mysqld-max
020602 18:58:21  InnoDB: Started
C:\MySQL\bin>  mysql firstdb;
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 4.0.1-alpha-max-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

```
mysql> SELECT * FROM customer;
ERROR 1146: Table 'firstdb.customer' doesn't exist
mysql> exit
Bye
```

现在恢复早先的备份：

```
C:\MySQL\bin> copy c:\db_backups\customer.* c:\MySQL\data\firstdb
```

做一次查询，可以看到已经丢失的最近的记录，它们是在备份后被加入的：

```
C:\MySQL\bin> mysql firstdb;
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 4.0.1-alpha-max-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> SELECT * FROM customer;
+----+------------+-------------+---------+
| id | first_name | surname     | initial |
+----+------------+-------------+---------+
|  1 | Yvonne     | Clegg       | X       |
|  2 | Johnny     | Chaka-Chaka | B       |
|  3 | Winston    | Powers      | M       |
|  4 | Patricia   | Mankunku    | C       |
|  5 | Francois   | Papo        | P       |
|  7 | Winnie     | Dlamini     | NULL    |
|  6 | Neil       | Beneke      | NULL    |
| 10 | Breyton    | Tshabalala  | B       |
+----+------------+-------------+---------+
8 rows in set (0.00 sec)
```

为了恢复它，需要使用二进制更新日志。首先，让我们看二进制更新日志文件中的内容。它不是文本文件，因此不能使用普通的文本编辑器，但MySQL带有一个应用程序mysqbinlog。对某一个二进制更新日志文件运行这一程序将会输出日志的内容。语法如下：

```
mysqlbinlog path_to_binary_update_log
```

让我们看日志文件里的内容：

```
C:\MySQL\bin>mysqlbinlog ..\data\speed_demon-bin.001
# at 4
#020602 18:58:21 server id  1   Start: binlog v 2, server v 4.0.1-alpha-max-log
created 020602 18:58:21
# at 79
#020602 19:01:11 server id  1   Query   thread_id=2 exec_time=0  error_code=0
use firstdb;
SET TIMESTAMP=1023037271;
INSERT INTO customer VALUES(11,'Robin','McKenzie');
# at 167
#020602 19:01:48 server id  1   Stop
```

如果正在运行二进制更新日志文件，你会有很多日志文件。选择第二个最近的，它应该抓住了最近的插入语句。

当然，这种输出结果在屏幕上不是很好看。可以按下面的方法把它送到实际的数据库中：

```
C:\MySQL\bin>mysqlbinlog ..\data\speed_demon-bin.001 | mysql firstdb
```

现在，能看到表和已恢复的记录了：

```
C:\MySQL\bin> mysql firstdb;
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 4.0.1-alpha-max-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> SELECT * FROM customer;
+----+------------+-------------+---------+
| id | first_name | surname     | initial |
+----+------------+-------------+---------+
|  1 | Yvonne     | Clegg       | X       |
|  2 | Johnny     | Chaka-Chaka | B       |
|  3 | Winston    | Powers      | M       |
|  4 | Patricia   | Mankunku    | C       |
|  5 | Francois   | Papo        | P       |
|  7 | Winnie     | Dlamini     | NULL    |
|  6 | Neil       | Beneke      | NULL    |
| 10 | Breyton    | Tshabalala  | B       |
| 11 | Robin      | McKenzie    | NULL    |
+----+------------+-------------+---------+
9 rows in set (0.00 sec)
```

记录已被成功恢复。

表11.4描述了mysqlbinlog可用的参数。

表11.4 mysqlbinlog参数

参数	描述
-?, --help	显示帮助信息和退出方法
-d, --database=dbname	只列出指定数据库的条目
-s, --short-form	只显示查询信息，没有多余的信息
-o, --offset=N	从开始跳过N个条目
-h, --host=server	从指定服务器中得到二进制文件
-P, --port=port	用指定端口连接到远程服务器
-u, --user=username	连接服务器的用户名
-p, --password=password	连接服务器的密码
-r, --result-file=file	在指定文件中放入输出结果
-j, --position=N	在位置N处开始读二进制日志
-t, --table=name	用COM_TABLE_DUMB得到原始表的转存文件
-V, --version	显示版本信息并退出

备份并恢复InnoDB表

目前，当服务器用标准分布运行时，不可能做InnoDB表的标准联机备份。这种状况不久会被改变，因此留意MySQL的文件。

有一种允许联机备份InnoDB表的付费工具，叫InnoDB HotBackup，具体细节，请访问www.innodb.com/hotbackup.html。

通常为了备份，需要关闭服务器或切断客户机的访问。有两种主要的备份方法。对于关键数据，两者都需要。一个方法是使用mysqldump（与对MyISAM表一样），备份期间没有写入权限，利用能恢复表的SQL语句创建文本文件。第二个是备份二进制文件。为了这样做，要在没有任何错误的情况下关闭服务器，然后拷贝数据文件、InnoDB日志文件、配置文件（my.cnf或my.ini）和定义文件到安全的地方：

```
% mysqladmin shutdown
% ls -l
total 76145
drwx------   2 mysql    mysql        2048 Jun  1 21:01 firstdb
-rw-rw----   1 mysql    mysql       25088 May  4 20:08 ib_arch_log_0000000000
-rw-rw----   1 mysql    mysql     5242880 Jun  1 21:04 ib_logfile0
-rw-rw----   1 mysql    mysql     5242880 May  4 20:08 ib_logfile1
-rw-rw----   1 mysql    mysql    67108864 Jun  1 21:04 ibdata1
drwxrwx---   2 mysql    mysql        1024 May  4 20:07 mysql
drwxrwx---   2 mysql    mysql        1024 Dec 23 17:44 test
-rw-rw----   1 mysql    mysql          98 May 19 15:03 test-bin.001
-rw-rw----   1 mysql    mysql       30310 Jun  1 21:04 test-bin.002
-rw-rw----   1 mysql    mysql          30 May 19 15:09 test-bin.index
-rw-r--r--   1 mysql    mysql        7292 Jun  1 21:04 test.dummymysql.co.za.err
```

应该以ib开头从数据目录中拷贝所有文件，因为有InnoDB的日志和数据。例如：

```
% cd /usr/local/mysql/data/
% cp ib*/db_backups/
```

现在，复制配置文件（记住，如果不止有一个配制文件，对它们都要进行复制）：

```
% cp /etc/my.cnf /db_backups/
```

然后拷贝定义文件，在这种情况中，innotest在firstdb目录里（所有定义文件，也包括MySQL的数据和索引文件，存在于一个与数据库同名的目录中）：

```
% cp firstdb/innotest.frm /db_backups/
```

因为恶意用户能毁坏数据，所以现在按顺序重启服务器：

```
% mysqld-max
% Starting mysqld daemon with databases from /usr/local/mysql/data
% mysql firstdb
```

```
mysql> TRUNCATE innotest;
Query OK, 11 rows affected (0.00 sec)
```

所有数据被删除，是恢复备份的时候了。需要再一次关闭服务器，以避免干扰：

```
% mysqladmin shutdown
020601 21:20:34  mysqld ended
% cp /db_backups/ib* /usr/local/mysql/data/
cp: overwrite '/usr/local/mysql/data/ib_arch_log_0000000000'? y
cp: overwrite '/usr/local/mysql/data/ib_logfile0'? y
cp: overwrite '/usr/local/mysql/data/ib_logfile1'? y
cp: overwrite '/usr/local/mysql/data/ibdata1'? y
```

此时不必恢复配置和定义文件，因为它们没有受到损坏。一旦硬件出错，就必须恢复这类文件：

```
% mysqld-max
% Starting mysqld daemon with databases from /usr/local/mysql/data
% mysql firstdb
mysql> SELECT * FROM innotest;
+------+------+
|   f1 | f2   |
+------+------+
|    1 | NULL |
|    2 | NULL |
|    3 | NULL |
|    4 | NULL |
|    5 | NULL |
|    6 | NULL |
|    7 | NULL |
|    8 | NULL |
|    9 | NULL |
|   10 | NULL |
+------+------+
10 rows in set (0.12 sec)
```

数据被成功恢复了。

在服务器被中断的情况下，只需简单地重启服务器就可以恢复InnoDB。如果通常的日志记录和存档功能被开启（推荐），InnoDN表将自动从MySQL日志中恢复（MySQL日志是普通日志，不是InnoDB日志）。任何中断时出现的未授权处理将被压制。输出结果如下：

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 24785115
InnoDB: Doing recovery: scanned up to log sequence number 0 24850631
InnoDB: Doing recovery: scanned up to log sequence number 0 24916167
```

```
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 982
InnoDB: Rolling back of trx no 98 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

提示：InnoDB文件没有MyISAM文件可移植性好。它们只能用在与产生它们的机器具有相同浮点数的机器的平台上。例如，可以在Intel x86机器之间转移文件，而不必管你所用的操作系统是什么。

复制是备份的一种方法

复制是另一种保持备份的方法（见第12章“数据库复制”）。它可在硬件出现错误的情况下，保护被复制的数据库，但对用户犯糊涂和恶意行为无效。如果用户删除了一些记录，而这一过程仍将被复制到其他服务器上，则复制无法作为一种可靠的备份形式。如果做了复制，就能减少对硬件失败的担心，但仍然要用其他方式进行备份。

小结

备份是任何MySQL管理员工具箱中的关键部分。下面这些方法可以使用：

- 用BACKUP语句建立MyISAM表的定义和数据拷贝。用RESTORE语句恢复数据。
- 直接拷贝文件。需要做锁定工作，通过将数据文件放回到数据目录的方法恢复数据。
- mysqldump建立一个包含用于再生表的SQL语句的文本文件。利用该文件作为MySQL后台的输入而恢复数据。
- 利用SELECT INTO语句创建可以用LOAD DATA命令或mysqlimport程序恢复的文本文件。
- 使用mysqlhotcopy应用程序。这是将数据文件拷贝到另一目录的Perl语句。然后，通过将数据文件放回到数据目录的方法恢复数据。
- 复制。它备份数据到另一台机器上，但如果SQL语句导致了数据损失，这种损失也会被复制。

激活的二进制更新日志文件能记录一切数据库表的变化。mysqlbinlog程序能被用于查看日志内容或更新已备份的数据库。InnoDB表不能像MyISAM表那样被存储在文件中，因此需要格外小心。它们也有其自身的日志机制。

第12章　数据库复制

MySQL有一个非常实用的特色叫复制，使其可以把服务器（叫主服务器）上的一个或多个数据库自动映射到一个或多个其他服务器（叫从属服务器）上。复制是一种有效的备份策略和性能调谐技术。本章将解释复制过程是如何工作的，并展示如何建立和设置复制。

本章要点：

- 建立复制
- 设置从属和主文件
- 了解从属和主SQL语句

了解复制工作

复制的工作过程如下：从属服务器首先对主服务器上的数据进行准确的拷贝，然后，激活主服务器上的二进制日志功能。接着，从属服务器将定期连接主服务器并检查二进制日志，看在上次连接后有何变化发生。从属服务器自己将自动重复上述过程。从属服务器上的master.info文件允许它了解其在主服务器二进制日志文件中的位置。主服务器的二进制日志与从属服务器的master.info文件之间的关系非常重要；如果二者失去同步，两个服务器上的数据就不会再相同。复制可以是一种有用的备份形式（防止磁盘错误和人工错误），并且可以加快运行速度。复制是一种运行多重数据库的实用方法，特别是在SELECT语句远比INSERT或UPDATE语句多的情况下（对SELECT语句从属服务器可以完全被优化，由主服务器去操作INSERT和UPDATE语句）。

但是，复制不能解决所有运行中的问题。更新仍然需要在从属服务器上重复进行，并且即使复制能被更优化，但当MyISAM表的更新次数太多和被锁定的次数太频繁，那么把这些表转换成InnoDB也许是更好的解决问题的办法。另外，在从属服务器的更新数据被复制期间有一个延迟，其时间长短取决于网络性能和数据服务器自身的性能。因此，对应用程序而言，不能简单地假设它能使用主或从属服务器作为数据库服务器。主服务器上的记录不一定会立即出现在从属服务器上，这也许会导致应用程序出现问题。

复制通常以一种分级的方式被执行（如图12.1和图12.2所示），但也能建立循环结构（如图12.3和图12.4所示）。可是，要确信客户代码之间没有冲突；另外，由于数据不规范，复制很容易失败，如图12.3和图12.4所示，这些都是不常用的结构。

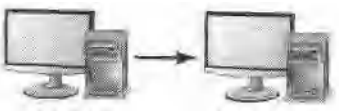

图12.1　一个主服务器，一个从属服务器

图12.2 一个主服务器，多个从属服务器

图12.3 主服务器/从属服务器的循环关系

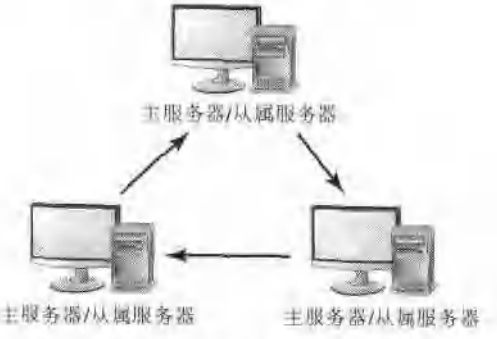

图12.4 主服务器/从属服务器的“菊花链”

没有必要保持不间断的连接。如果连接因任何原因中断，从属服务器将会试图重新连接，并在连接恢复后，马上从停止处开始进行更新。

建立复制

对于建立主服务器和从属服务器之间的关系，有许多具体的境况。本章中的例子包括了一些情况。下面是启动复制的基本步骤。

在主服务器上执行如下步骤：

1. 用REPLICATION SLAVE权限建立复制用户：

```
GRANT REPLICATION_SLAVE ON *.* TO replication_user IDENTIFIED BY
 'replication_password';
```

2. 做一份表和数据的拷贝。如果数据库已经被用过一段时间且二进制日志也已存在（见第10章“基本管理”中有关二进制日志的具体细节介绍），则要立刻记录备份后的差异（更多信息，见本章后边“复制命令”一节）。在从属服务器上进行LOAD DATA FROM MASTER操作，能代替这一步骤。目前，LOAD DATA FROM MASTER只能对MyISAM表起作用，而且最好用于小型数据表或主服务器上的数据可以在操作期间被锁定的情况。版本4.1应该能解决这些不足。

3. 将下面的代码加入配置文件（my.cnf或my.ini）。log-bin指明主服务器将使用二进制更新日志，而server-id是区分每个主和从属服务器的惟一代码。习惯上，主服务器通常被设置为1，而从属服务器被设置为2或更大值：

```
[mysqld]
log-bin
server-id=1
```

在从属服务器上执行下面的步骤：

1. 将下面的内容加入配置文件（my.cnf或my.ini）。master_hostname是主服务器的主机名，master_user和master_password值分别是用户名和密码，被建立在主服务器上用于复制（使用复制从属特权）。master_TCP/IP_port是主服务器监听到的端口号（它只有在端口号为非标准时才需要），号码惟一，从数字2到2^32-1：

```
[mysqld]
master-host=master_hostname
master-user=replication_user
master-password=replication_password
master-port=master_TCP/IP_port
server-id=unique_number
```

2. 将从主服务器上取得的数据拷贝到从属服务器上（如果没有运行LOAD DATA FROM MASTER）。
3. 启动从属服务器。
4. 如果还没有得到数据，用LOAD DATA FROM MASTER访问它。

现在，随着双方服务器的运行，将会开始看到复制进程的出现。

复制参数

表12.1描述了可用于主服务器的不同复制参数，而表12.2描述了可用于从属服务器的不同复制参数。

表12.1 主服务器配置文件参数

参数	描述
log-bin=filename	激活二进制日志。为使复制可以发生，该参数必须出现在主服务器上。文件名是可用参数。为了清除日志，运行RESET MASTER，且别忘了在所有从属服务器上运行RESET SLAVE。默认的二进制日志文件名为hostname.xxx，xxx是从001开始的数字，日志每更新一次，该数字就加1
log-bin-index=filename	指定二进制日志索引文件名（它按顺序列出二进制日志文件，因此从属服务器总会知道激活的是哪一个）。默认值为hostname.index
sql-bin-update-same	如果设置SQL_LOG_BIN为1或0，SQL_LOG_UPDATE将自动被设为相同的数值，反之亦然。SQL_LOG_UPDATE将不再被需要，所以该参数可能不会被使用

（续表）

参数	描述
binlog-do-db=database_name	只对database_name数据库进行二进制日志更新。其他数据库被忽略。也可以在从属服务器上约定数据库
binlog-ignore-db=database_name	更新除database_name以外的二进制日志。你也可以在从属服务器上设置需要忽略的数据库

表12.2 从属服务器配置文件参数

参数	描述
master-host=host	指定需要连接的主服务器主机名或IP地址。这是开启复制功能必须被设置的。一旦复制开始，master.info数据将决定这个主服务器，为了改变它，需要使用CHANGE MASTER语句
master-user=username	指定从属服务器连接的主服务器上的用户名。用户要有在主服务器上的REPLICATION SLAVE权限。默认时，要进行测试。一旦复制开始，master.info数据将决定这个主服务器，为了改变它，需要使用CHANGE MASTER语句
master-password=password	指定连接到主服务器上的密码。默认时为一空字符串。一旦复制开始，master.info数据将决定这个主服务器，为了改变它，需要使用CHANGE MASTER语句
master-port=portnumber	指定主服务器要监听到的端口（默认为MYSQL_PORT的值，通常为3306）。一旦复制开始，master.info数据将决定这个主服务器，为了改变它，需要使用CHANGE MASTER语句
master-connect-retry=seconds	如果主服务器和从属服务器间的连接中断，MySQL将在试图重新连接时，等待一些时间（默认为60秒）
master-ssl	指明复制发生时，使用安全套接层（SSL）
master-ssl-key=key_name	如果SSL被设置使用（master-ssl参数），该参数指明主服务器的SSL关键文件名
master-ssl-cert=certificate_name	如果SSL被设置使用（master-ssl参数），该参数指明主服务器的SSL证书名
master-info-file=filename	指明主服务器的信息文件（默认为数据目录中的master.info），该文件用于在复制过程中找到从属服务器在二进制日志中的位置
report-host	指明从属服务器通知给主服务器其将要使用的主机名或IP地址（用于SHOW SLAVE HOSTS语句）。没有默认值。其他决定主机的方法不可靠，因此该参数是需要的
report-port	指明端口，以便连接那些向主服务器汇报的从属服务器。只有当从属服务器在非默认端口或连接以一种非标准方式发生时，才需要这样做

（续表）

参数	描述
replicate-do-table=db_name.table_name	确认从属服务器时只从指定数据库中复制指定表的名字。可以多次使用该参数去复制多个表格
replicate-ignore-table=db_name.table_name	告诉服务器不要复制更新指定表的语句（即使其他表也被相同的语句更新）。可以指定该参数的多个例子
replicate-wild-do-table=db_name.table_name	告诉从属服务器仅复制与指定表格匹配的语句（类似于replicate_do_table参数），但这里的匹配要考虑到通配符。例如，表名是db%.tb%，则匹配适用于任何以db和tb字母开头的表
replicate-wild-ignore-table=db_name.table_name	告诉从属服务器不复制更新指定表格的语句，即使其他表也被相同的语句更新（类似于replicate-ignore-table参数），但要考虑通配符。例如，表名是db%.tb%，则复制不包括任何以db和tb字母开头的表。可以指定该参数的多个例子
replicate-ignore-db=database_name	告诉从属服务器，当前数据库是database_name时，不复制任何语句。可以多次使用该参数，指明要忽略的多个数据库
replicate-do-db=database_name	告诉从属服务器，只有当数据库为database_name时，才复制语句。可以多次使用该参数去复制多个数据库
log-slave-updates	告诉从属服务器将复制更新记入二进制日志。没有默认设置。如果准备把从属服务器用做其他从属服务器的主服务器，需要这个参数
replicate-rewrite-db=master_database->slave_database	如果从属服务器上的数据库与主服务器上相同数据库的名字不同，就需要用这个参数来映射这种关系
slave-skip-errors= [err_code1, err_code2,... | all]	当复制遇到错误时，将会停止（由于错误通常意味着数据不协调，所以手动步骤是需要的）。该参数告诉MySQL，当错误是列出错误中的一种时，继续复制。错误代码以数字形式给出（相同的数字在错误日志中给出）并由逗号来分隔。可以使用所有的参数对应所有可能的错误。不能滥用该参数，因为错误的运用会导致数据与主服务器失去同步。除了重新拷贝主服务器数据外，没有现实的方法可以恢复同步
skip-slave-start	该参数被设置时，复制不会在服务器启动的时候开始。可以用SLAVE START命令手动启动复制
slave_compressed_protocol=#	如果设为1且服务器支持压缩，则MySQL将使用压缩方式传送从属和主服务器之间的数据
slave_net_timeout=#	在读取失败后，确定从主服务器上获得更多数据的等待时间

复制命令

应该熟悉从属和主服务器的复制命令。下面是从属服务器的命令：

- SLAVE START和SLAVE STOP分别用于启动和停止复制过程。

- SHOW SLAVE STATUS返回有关从属服务器的信息，包括从属服务器是否与主服务器相连（Slave_IO_Running），复制是否在运行（Slave_SQL_Running），被使用的二进制日志内容（Master_Log_File和Relay_Master_Log_File）和当前二进制日志的位置（Read_Master_Log_Pos和Exec_master_log_pos）。
- CHANGE MASTER TO语句是保持复制同步或在适当地方启动复制的重要语句。MASTER_LOG_FILE查询主服务器的二进制日志，从属服务器必须从这里开始复制。其中，MASTER_LOG_POS是指在文件中的位置（本章稍后将会看到相关的例子）。当主服务器失败时，也使用该语句，你需要改变与从属服务器相连的主服务器。CHANGE MASTER TO参数的全部设置如下：

```
CHANGE MASTER TO MASTER_HOST = 'master_hostname',
   MASTER_USER='replication_username',
   MASTER_PASSWORD=''replication_user_password',
   MASTER_PORT='master_port',
   MASTER_LOG_FILE='master_binary_logfile',
   MASTER_LOG_POS='master_binary_log_position'
```

- RESET SLAVE语句使从属服务器忘掉其在主服务器日志中的位置。
- LOAD DATA FROM MASTER从主服务器上取一份数据拷贝，并把它放在从属服务器上。目前，该语句对大的数据列无效，或在主服务器不能长时间使用的情况下也无效，因为该语句拷贝数据时要获得全局锁。它也更新MASTER_LOG_FILE和MASTER_LOG_POS的值。目前，它只对MyISAM表起作用。该语句可能成为将来从属服务器进行准备的标准方法，所以一定要阅读最新的文档。
- SET GLOBAL SQL_SLAVE_SKIP_COUNTER=n语句使从属服务器从主服务器二进制日志上跳过n条语句。

下面是主服务器的复制命令：

- SET SQL_LOG_BIN=n语句既不激活二进制更新日志（如设置为0），也不重新激活它（如果设置为1）。需要SUPER特权去运行该语句。
- RESET MASTER删除所有二进制日志并从001开始重新计数。
- SHOW MASTER STATU显示当前二进制日志、二进制日志中的位置和是否有任何数据库因进行二进制日志而被忽略。
- PURGE MASTER LOGS TO binary_log_filename删除所有指定二进制日志前的日志。确信在删除这些日志前，从属服务器不再需要二进制日志。看本章稍后“从主服务器上删除旧日志和启动”一节中的例子。
- SHOW MASTER LOGS显示可用二进制日志文件的清单。在清除日志前，会经常用到它。
- SHOW SLAVE HOSTS返回向主服务器登记的从属服务器清单（注意，默认时，从属服务器不会注册自己，但会要求设置report-host配置参数）。
- SHOW BINLOG EVENTS [IN 'logname'] [FROM pos] [LIMIT [offset,] rows]语句是用于从二进制日志中读取语句的。

复制的复杂性

下面是一些至关重要的事情，在安装和设置复制时，需要记住它们：

- FLUSH语句不被复制。如果在主服务器上直接更新权限表，然后用FLUSH去激活这些变化的话，你会受到影响。直到运行FLUSH语句，变化才会生效。
- 确认主和从属服务器有相同的字符设置。
- 当传入一个任意表达式为自变量时，The RAND()函数不能正确工作。需要安全使用像UNIX_TIMESTAMP()这样的函数。
- 对更新数据和使用用户变量这样的查询将不会被安全复制（这条将被改变，请查看最新的文档）。
- 复制在不同的MySQL版本中通常会正常工作，甚至是在3.23.x和4.0.x版本之间。但会有例外（如4.0.0、4.0.1或4.0.2之间，工作就有问题），因此要查看最新的文档以核实。另外，如果可能，始终使用最新的版本。

复制数据库

为了这个例子，你要创建一个带有表的新数据库，并将其复制到另一个服务器上。你需要有两台运行MySQL的服务器（最好是同样的版本），且两个服务器能互相看到，以便测试这个例子。

首先，在主服务器上建立一个叫replication_db的数据库，表名为replication_db，现在给这个表添加数据，如下所示：

```
mysql> CREATE DATABASE replication_db;
Query OK, 1 row affected (0.01 sec)
mysql> USE replication_db;
Database changed
mysql> CREATE TABLE replication_table(f1 INT,f2 VARCHAR(20));
Query OK, 0 rows affected (0.03 sec)
mysql> INSERT INTO replication_table (f1,f2) VALUES(1,'first');
Query OK, 1 row affected (0.03 sec)
```

现在，授予从属服务器进行复制的权限。从属服务器用户名为replication_user，密码为replication_pwd：

```
mysql> GRANT REPLICATION SLAVE ON *.* TO replication_user IDENTIFIED BY
 'replication_pwd';
```

在从属服务器上，关闭服务器，在配置文件（my.cfg和my.ini）中加入下面的代码。用从属服务器的IP代替master-host的设置。server_id可以是任何数字，只要不与主服务器的server_id相同：

```
master-host     = 192.168.4.100
master-user     = replication_user
master_password = replication_pwd
```

```
server-id       = 3
replicate-do-db = replication_db
```

在从属服务器上建立replication_db数据库，从主服务器上拷贝replication_table表的数据到从属服务器（作为MyISAM表，数据将在replication_db目录中）。如果不能确定如何做，参见第11章“数据库备份”。当拷贝文件到从属服务器时，确信权限是正确的（UNIX的是chown mysql.mysql *, chmod 700 *）。另外要注意，如果主服务器已经使用了二进制日志，需要用RESET MASTER重设二进制日志，这样从属服务器就可以从第一个二进制日志的开头开始更新了。现在启动从属服务器并连接。一旦连接上，查看从属服务器的状态，看是否复制已经正确地开始了：

```
mysql> SHOW SLAVE STATUS;
+---------------+-----------------+-------------+---------------+-----
--------------+---------------------+--------------------------+------
---------+----------------------+-----------------+-------------------
-+-------------------+-------------------+-------------+-------------+---
----------+----------------------+-----------------+
| Master_Host   | Master_User       | Master_Port | Connect_retry |
Master_Log_File    | Read_Master_Log_Pos | Relay_Log_File           |
Relay_Log_Pos | Relay_Master_Log_File | Slave_IO_Running |
Slave_SQL_Running | Replicate_do_db | Replicate_ignore_db | Last_errno
| Last_error | Skip_counter | Exec_master_log_pos | Relay_log_space |
+---------------+-----------------+-------------+---------------+-----
--------------+---------------------+--------------------------+------
---------+----------------------+-----------------+-------------------
-+-------------------+-------------------+-------------+-------------+---
----------+----------------------+-----------------+
| 192.168.4.100 | replication_user | 3306        | 60            |
g-bin.001 | 79                   | s-bin.002 |
124           | g-bin.001    | Yes              | Yes
| replication_db  |                      | 0           |             | 0
| 79                | 132             |
+---------------+-----------------+-------------+---------------+-----
--------------+---------------------+--------------------------+------
---------+----------------------+-----------------+-------------------
-+-------------------+-------------------+-------------+-------------+---
----------+----------------------+-----------------+
1 row in set (0.00 sec)
mysql> INSERT INTO replication_table (f1,f2) VALUES(2,'second');
Query OK, 1 row affected (0.06 sec)
mysql> SELECT * FROM replication_table;
+------+--------+
| f1   | f2     |
+------+--------+
```

```
|    1 | first  |
|    2 | second |
+------+--------+
2 rows in set (0.00 sec)

mysql> SHOW SLAVE STATUS;
+---------------+-----------------+-------------+---------------+-----
--------------+-------------------+-----------------------+------
---------+----------------------+-----------------+-----------------
-+-----------------+-------------------+-----------+------------+---
----------+--------------------+-----------------+
| Master_Host   | Master_User       | Master_Port | Connect_retry |
Master_Log_File     | Read_Master_Log_Pos | Relay_Log_File            |
Relay_Log_Pos | Relay_Master_Log_File | Slave_IO_Running |
Slave_SQL_Running | Replicate_do_db | Replicate_ignore_db | Last_errno
| Last_error | Skip_counter | Exec_master_log_pos | Relay_log_space |
+---------------+-----------------+-------------+---------------+-----
---------------+-------------------+-----------------------+------
---------+----------------------+-----------------+-----------------
-+-----------------+-------------------+-----------+------------+---
----------+--------------------+-----------------+
| 192.168.4.100 | replication_user | 3306        | 60            |
g-bin.001 | 180                  | s-bin.002 |
 225          | g-bin.001     | Yes              | Yes
| replication_db  |                     | 0          |             | 0
| 180                 | 233             |
+---------------+-----------------+-------------+---------------+-----
---------------+-------------------+-----------------------+------
---------+----------------------+-----------------+-----------------
-+-----------------+-------------------+-----------+------------+---
----------+--------------------+-----------------+
1 row in set (0.00 sec)
```

回到主服务器，可以运行DELETE和UPDATE语句，这些将被映射到从属服务器上。例如：

```
mysql> DELETE FROM replication_table WHERE f1=1;
Query OK, 1 row affected (0.34 sec)
mysql> UPDATE replication_table SET f1=1;
Query OK, 1 row affected (0.05 sec)
```

检查从属服务器，将看到下面的内容：

```
mysql> SELECT * FROM replication_table;
+------+--------+
| f1   | f2     |
+------+--------+
```

```
|    1 | second |
+------+--------+
1 row in set (0.01 sec)
```

从属服务器不必为了保持同步而总与主服务器保持连接，只要二进制日志正确就可以，正像下一个例子所展示的。首先，关闭从属服务器：

```
% /usr/local/mysql/bin/mysqladmin -uroot -pg00r002b shutdown
020821 17:25:37  mysqld ended
```

然后添加另一个记录到主服务器：

```
mysql> INSERT INTO replication_table (f1,f2) VALUES(3,'third');
Query OK, 1 row affected (0.03 sec)
```

回到从属服务器，重启服务器，连接replication_db数据库，将看到新的记录已经被加入：

```
% bin/mysqld_safe &
[1] 1989
% /usr/local/mysql/bin/mysql -uroot -pg00r002b mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 4.0.2-alpha-max-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> SELECT * FROM replication_table;
+------+--------+
| f1   | f2     |
+------+--------+
|    3 | third  |
|    1 | second |
+------+--------+
2 rows in set (0.02 sec)
```

主服务器也可能停止，从属服务器将不断反复连接（每隔master-connect-retry秒，默认值为60），直到主服务器再次工作为止。

当改变二进制日志时要特别小心，因为这是从属服务器必须做的事情。下一个例子将显示数据在哪里可能丢失。首先，关闭从属服务器：

```
% /usr/local/mysql/bin/mysqladmin -uroot -pg00r002b shutdown
020821 17:25:37  mysqld ended
```

像过去一样，加入另一个记录到主服务器中，但这次，在加入完毕后要运行RESET MASTER（这样可以清除所有旧的二进制日志，然后从二进制日志1再次启动）：

```
mysql> INSERT INTO replication_table (f1,f2) VALUES(4,'fourth');
Query OK, 1 row affected (0.01 sec)
mysql> RESET MASTER;
Query OK, 0 rows affected (0.03 sec)
```

现在，当你重启从属服务器时，它将不会改变。

```
% bin/mysqld_safe &
[1] 1989
% /usr/local/mysql/bin/mysql -uroot -pg00r002b mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 4.0.2-alpha-max-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> SELECT * FROM replication_table;
+------+--------+
| f1   | f2     |
+------+--------+
|    3 | third  |
|    1 | second |
+------+--------+
2 rows in set (0.00 sec)
```

你能通过查看从属服务器的状态知道为何这样做：

```
mysql> SHOW SLAVE STATUS;
+---------------+-----------------+-------------+---------------+-----
--------------+--------------------+--------------------+------
--------+----------------------+-----------------+------------------
-+-------------------+-----------------------+------------+------------+---
----------+--------------------+-----------------+
| Master_Host   | Master_User       | Master_Port | Connect_retry |
Master_Log_File     | Read_Master_Log_Pos | Relay_Log_File           |
Relay_Log_Pos | Relay_Master_Log_File | Slave_IO_Running |
Slave_SQL_Running | Replicate_do_db | Replicate_ignore_db | Last_errno
| Last_error | Skip_counter | Exec_master_log_pos | Relay_log_space |
+---------------+-----------------+-------------+---------------+-----
--------------+--------------------+--------------------------+------
--------+----------------------+-----------------+------------------
-+-------------------+-----------------------+------------+------------+---
----------+--------------------+-----------------+
| 192.168.4.100 | replication_user | 3306          60             |
g-bin.001 | 443                  | s-bin.004 | 4
| g-bin.001 | Yes                | Yes               |
replication_db  |                      | 0          |            | 0
| 443                 | 500             |
+---------------+-----------------+-------------+---------------+-----
-- ------------+--------------------+--------------------------+------
--------+----------------------+-----------------+------------------
-+-------------------+-----------------------+------------+------------+---
----------+--------------------+-----------------+
1 row in set (0.00 sec)
```

主服务器日志被设定在位置443，这是主服务器二进制日志的实际内容：

```
mysql> SHOW MASTER STATUS;
+-----------+----------+--------------+------------------+
| File      | Position | Binlog_do_db | Binlog_ignore_db |
+-----------+----------+--------------+------------------+
| g-bin.001 | 79       |              |                  |
+-----------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

能通过重设从属服务器恢复它们之间的同步，如下：

```
mysql> RESET SLAVE;
Query OK, 0 rows affected (0.01 sec)
```

从属服务器的状态现在已经改变，再看二进制日志1的开始部份，即位置79：

```
mysql> SHOW SLAVE STATUS;
+---------------+-----------------+-------------+---------------+-----
--------------+--------------------+-----------------------+------
--------+----------------------+-----------------+-----------------
-+-----------------+--------------------+-----------+------------+---
----------+--------------------+----------------+
| Master_Host   | Master_User       | Master_Port | Connect_retry |
Master_Log_File    | Read_Master_Log_Pos | Relay_Log_File           |
Relay_Log_Pos | Relay_Master_Log_File | Slave_IO_Running |
Slave_SQL_Running | Replicate_do_db | Replicate_ignore_db | Last_errno
| Last_error | Skip_counter | Exec_master_log_pos | Relay_log_space |
+---------------+-----------------+-------------+---------------+-----
--------------+--------------------+-----------------------+------
--------+----------------------+-----------------+-----------------
-+-----------------+--------------------+-----------+------------+---
----------+--------------------+----------------+
| 192.168.4.100 | replication_user | 3306        | 60            |
g-bin.001 | 79                  | s-bin.002 |
124           | g-bin.001     | Yes              | Yes
| replication_db |                      | 0           |            | 0
| 79                  | 132             |
+---------------+-----------------+-------------+---------------+-----
--------------+--------------------+-----------------------+------
--------+----------------------+-----------------+-----------------
-+-----------------+--------------------+-----------+------------+---
----------+--------------------+----------------+
1 row in set (0.00 sec)
```

现在回到主服务器，再加入记录，看一下主服务器的状态，二进制日志的位置已经移到了位置180：

```
mysql> INSERT INTO replication_table (f1,f2) VALUES(4,'fourth');
Query OK, 1 row affected (0.00 sec)
mysql> SHOW MASTER STATUS;
+-----------+----------+--------------+------------------+
| File      | Position | Binlog_do_db | Binlog_ignore_db |
+-----------+----------+--------------+------------------+
| g-bin.001 | 180      |              |                  |
+-----------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

从属服务器再次使用INSERT:

```
mysql> SELECT * FROM replication_table;
+------+--------+
| f1   | f2     |
+------+--------+
|    3 | third  |
|    1 | second |
|    4 | fourth |
+------+--------+
3 rows in set (0.00 sec)
```

你会发现记录在主服务器上被加入了两次:

```
mysql> SELECT * FROM replication_table;
+------+--------+
| f1   | f2     |
+------+--------+
|    3 | third  |
|    1 | second |
|    4 | fourth |
|    4 | fourth |
+------+--------+
4 rows in set (0.00 sec)
```

这个例子说明:复制工作有效时,并不能保证两个服务器上的数据相同。通过好的设计(比如为表加上优先级的键),你能避免这个问题;但当使用二进制日志工作时,要仔细监视从属和主服务器的状态。

用主服务器上激活的二进制日志进行复制

这里的例子将示范如何处理主服务器已经激活了二进制日志,且已经运行了一段时间,而你想设置一个复制的情况。首先,关闭从属服务器,避免与前一个例子发生任何冲突。

```
% /usr/local/mysql/bin/mysqladmin -uroot -pg00r002b shutdown
020821 23:40:49  mysqld ended.
```

你将使用与前一个例子一样的表。在主服务器上，为了在插入一些记录之前从头开始，需删除这个表并重设二进制日志，如下：

```
mysql> DELETE FROM replication_table;
Query OK, 4 rows affected (0.09 sec)
mysql> RESET MASTER;
Query OK, 0 rows affected (0.02 sec)
mysql> INSERT INTO replication_table (f1,f2) VALUES(1,'first');
Query OK, 1 row affected (0.01 sec)
mysql> INSERT INTO replication_table (f1,f2) VALUES(2,'second');
Query OK, 1 row affected (0.01 sec)
```

现在拷贝这些数据到从属服务器上，并且检查主服务器上二进制日志的差异。确信在拷贝数据之后和能检查状态之前，新的数据写入了主服务器：

```
mysql> SHOW MASTER STATUS;
+-----------+----------+--------------+------------------+
| File      | Position | Binlog_do_db | Binlog_ignore_db |
+-----------+----------+--------------+------------------+
| g-bin.001 | 280      |              |                  |
+-----------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

现在，在从属服务器上执行与前一个例子相同的参数，即拷贝数据、设置配置参数（用随后的变化）、删除可能存在的master.info文件（由前一个例子在数据目录中创建的，如C:\mysql\data或/usr/local/mysql/data），然后启动从属服务器。惟一的不同是配置文件应该含有这个参数：

```
skip-slave-start
```

直到你已设置从属服务器在正确的点开始工作，才能启动它进行复制，现在为主服务器添加更多的记录：

```
mysql> INSERT INTO replication_table (f1,f2) VALUES(3,'third');
Query OK, 1 row affected (0.01 sec)
mysql> INSERT INTO replication_table (f1,f2) VALUES(4,'fourth');
Query OK, 1 row affected (0.01 sec)
```

现在，在从属服务器上，需要告诉服务器以正确的二进制日志文件和偏差启动。为了做到这一点，设置MASTER_LOG_FILE为g-bin.001（或者任何运行SHOW MASTER STATUS所显示的内容），且设置MASTER_LOG_POS为280（或者任何适用于此情况的值）。一旦完成上述设置，启动从属服务器复制功能，并测试结果。

```
mysql> CHANGE MASTER TO MASTER_LOG_FILE='g-bin.001',
 MASTER_LOG_POS=280;
Query OK, 0 rows affected (0.00 sec)
mysql> SLAVE START;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT * FROM replication_table;
+------+--------+
| f1   | f2     |
+------+--------+
|    1 | first  |
|    2 | second |
|    3 | third  |
|    4 | fourth |
+------+--------+
4 rows in set (0.01 sec)
```

从主服务器删除旧二进制日志，然后启动

当复制时，删除二进制日志是危险的，因为从属服务器可能尚未完成你准备删除的日志的备份工作。

为了这个例子，需要删除数据和重设主服务器，以便从清楚的记录开始，然后加入些数据：

```
mysql> DELETE FROM replication_table;
mysql> RESET MASTER;
mysql> INSERT INTO replication_table (f1,f2) VALUES(1,'first');
mysql> INSERT INTO replication_table (f1,f2) VALUES(2,'second');
mysql> INSERT INTO replication_table (f1,f2) VALUES(3,'third');
```

拷贝数据到从属服务器（如果已经在从属服务器上运行了前一个例子，确信删除了master.info文件，然后用skip-slave-start参数启动从属服务器）。

现在清除主服务器上的日志，并模拟服务器已经运行了一段时间：

```
mysql> FLUSH LOGS;
mysql> FLUSH LOGS;
```

现在，观察主服务器状态时，将看到服务器已经有了第三个二进制日志：

```
mysql> SHOW MASTER STATUS;
+-----------+----------+--------------+------------------+
| File      | Position | Binlog_do_db | Binlog_ignore_db |
+-----------+----------+--------------+------------------+
| g-bin.003 | 4        |              |                  |
+-----------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

现在启动从属服务器，并从正确的点启动它的复制功能：

```
mysql> CHANGE MASTER TO MASTER_LOG_FILE='g-bin.003',MASTER_LOG_POS=4;
Query OK, 0 rows affected (0.01 sec)
mysql> SLAVE START;
Query OK, 0 rows affected (0.00 sec)
```

从属服务器现在从主服务器上的正确日志启动了。在主服务器上仍然有其他两个占据着空间的二进制日志，你需要开始维护日志文件以便确保它们不会失去控制。可以删除一两个日志，但从属服务器可能会用到它们，这样做是不安全的。

为了检查这点，需要检查每一个从属服务器的状态。按现在的情况，只有一个：

```
mysql> SHOW SLAVE STATUS;
+---------------+-----------------+-------------+---------------+-----
--------------+--------------------+-----------------------+------
---------+----------------------+-----------------+-----------------
-+-----------------+--------------------+------------+------------+---
----------+--------------------+-----------------+
| Master_Host   | Master_User      | Master_Port | Connect_retry |
Master_Log_File    | Read_Master_Log_Pos | Relay_Log_File          |
Relay_Log_Pos | Relay_Master_Log_File | Slave_IO_Running |
Slave_SQL_Running | Replicate_do_db | Replicate_ignore_db | Last_errno
| Last_error | Skip_counter | Exec_master_log_pos | Relay_log_space |
+---------------+-----------------+-------------+---------------+-----
--------------+--------------------+-----------------------+------
---------+----------------------+-----------------+-----------------
-+-----------------+--------------------+------------+------------+---
----------+--------------------+-----------------+
| 192.168.4.100 | replication_user | 3306        | 60            |
g-bin.003 | 4                  | s-bin.003 |
830           | g-bin.003    | Yes             | Yes
| replication_db |                     | 0          |            | 0
| 4                  | 1885           |
```

你能看见从属服务器正使用g-bin.003并获得了更新（位置4）。如果所有的从属服务器都是更新过的，你可以用PURGE MASTER LOGS语句安全的从主服务器上清除二进制日志1和2了，如下所示：

```
mysql> PURGE MASTER LOGS TO "g-bin.003";
Query OK, 0 rows affected (0.00 sec)
```

如果在数据目录里有一个清单（或者任何已指定的二进制日志所在的地方），你将看到两个先前的日志已经被清除。如果活动状态的从属服务器正试图读取删除的日志，语句将会失败，并给出如下错误信息：

```
mysql> PURGE MASTER LOGS TO "g-bin.003";
ERROR:
A purgeable log is in use, will not purge
```

如果从属服务器没有被连接，在删除还没用过的二进制日志时，从属服务器将不能继续复制。在某些阶段，这个过程能被自动化，但眼下，你必须人工检查每一个从属服务器，查看它的位置。如果没有检查，让我们看一下会发生情况。首先，停止从属服务器：

```
mysql> SLAVE STOP;
Query OK, 0 rows affected (0.00 sec)
```

现在，在主服务器上，再次刷新日志，加入新的记录，然后清除二进制日志：

```
mysql> FLUSH LOGS;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO replication_table (f1,f2) VALUES(4,'fourth');
Query OK, 1 row affected (0.00 sec)
mysql> FLUSH LOGS;
Query OK, 0 rows affected (0.00 sec)
mysql> SHOW MASTER STATUS;
+-----------+----------+--------------+------------------+
| File      | Position | Binlog_do_db | Binlog_ignore_db |
+-----------+----------+--------------+------------------+
| g-bin.005 | 4        |              |                  |
+-----------+----------+--------------+------------------+
1 row in set (0.00 sec)
mysql> PURGE MASTER LOGS TO "g-bin.005";
Query OK, 0 rows affected (0.02 sec)
```

现在，如果重启从属服务器，它将不进行复制，因为它正在找一个不存在的二进制日志：

```
mysql> SLAVE START;
Query OK, 0 rows affected (0.00 sec)
```

当前的问题是主服务器上最近的INSERT语句没有出现在日志中，因为所有旧的日志都被清除了。如果你做了二进制日志的备份，就能很容易地恢复它，但如果没有，可以手工再次运行该语句，然后让从属服务器查看最近的日志，如下所示：

```
mysql> INSERT INTO replication_table (f1,f2) VALUES(4,'fourth');
Query OK, 1 row affected (0.00 sec)
mysql> CHANGE MASTER TO MASTER_LOG_FILE='g-bin.005',MASTER_LOG_POS=4;
Query OK, 0 rows affected (0.01 sec)
```

现在，如果你添加另一个记录到主服务器：

```
mysql> INSERT INTO replication_table (f1,f2) VALUES(5,'fifth');
Query OK, 1 row affected (0.00 sec)
```

它将顺利地被复制到从属服务器上：

```
mysql> SELECT * FROM replication_table;
+------+--------+
| f1   | f2     |
+------+--------+
|    1 | first  |
|    2 | second |
```

```
|    3 | third  |
|    4 | fourth |
|    5 | fifth  |
+------+--------+
5 rows in set (0.00 sec)
```

这个例子显示了复制不是严格的数据拷贝。更合适的说法是，它是把语句从一个服务器复制到另一个。这是准确的数据拷贝，但如果master.info文件或二进制日志被损坏，MySQL将不能依次按顺序执行命令，这将导致数据失去同步。

避免太多的更新操作

这个例子显示如果不从主服务器二进制日志的正确点启动从属服务器，可能会发生的情况。

启动主服务器，并加入一些记录，然后在拷贝完成后立刻记录二进制日志的细节：

```
mysql> DELETE FROM replication_table;
mysql> RESET MASTER;
mysql> INSERT INTO replication_table (f1,f2) VALUES(1,'first');
mysql> INSERT INTO replication_table (f1,f2) VALUES(2,'second');
mysql> SHOW MASTER STATUS;
+-----------+----------+--------------+------------------+
| File      | Position | Binlog_do_db | Binlog_ignore_db |
+-----------+----------+--------------+------------------+
| g-bin.001 | 280      |              |                  |
+-----------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

将数据拷贝到从属服务器，然后启动从属服务器。从属服务器应该是简洁的（没有master.info文件，并且在replication_db数据库中没有数据），并且在它的配置文件中含有设置参数的普通内容，如下所示：

```
master-host     = 192.168.4.100
master-user     = replication_user
master_password = replication_pwd
server-id       = 3
replicate-do-db = replication_db
```

启动从属服务器并且看一下从属服务器状态，以便查看复制已经正确开始了：

```
mysql> SHOW SLAVE STATUS;
+---------------+-----------------+-------------+---------------+-----
--------------+---------------------+-------------------------+------
--------+--------------------+-----------------+-----------------
-+-----------------+--------------------+-----------+-------------+---
----------+--------------------+-----------------+
```

```
| Master_Host    | Master_User       | Master_Port | Connect_retry |
Master_Log_File     | Read_Master_Log_Pos | Relay_Log_File           |
Relay_Log_Pos | Relay_Master_Log_File | Slave_IO_Running |
Slave_SQL_Running | Replicate_do_db | Replicate_ignore_db | Last_errno
| Last_error | Skip_counter | Exec_master_log_pos | Relay_log_space |
+---------------+-----------------+-------------+---------------+-----
--------------+---------------------+--------------------------+------
---------+---------------------+-----------------+------------------
-+-------------------+---------------------+-----------+------------+---
----------+----------------------+-----------------+
| 192.168.4.100 | replication_user | 3306        | 60            |
g-bin.001 | 280                  | s-bin.003 |
325           | g-bin.001     | Yes              | Yes
| replication_db  |                      | 0          |            | 0
| 280                  | 329              |
+---------------+-----------------+-------------+---------------+-----
--------------+---------------------+--------------------------+------
---------+---------------------+-----------------+------------------
-+-------------------+---------------------+-----------+------------+---
----------+----------------------+-----------------+
1 row in set (0.00 sec)
```

每件事好像都工作正常。复制已经开始，从属服务器与主服务器处在相同的点上，即二进制日志的g-bin.001和位置280。但是当你检查从属服务器上的数据时，会感到不愉快并很惊讶：

```
mysql> SELECT * FROM replication_table;
+------+--------+
| f1   | f2     |
+------+--------+
|    1 | first  |
|    2 | second |
|    1 | first  |
|    2 | second |
+------+--------+
4 rows in set (0.00 sec)
```

问题是：从属服务器从第一个二进制日志的开头开始复制。这意味着：尽管数据已被拷贝，从属服务器还是重复了两次INSERT语句。有两种解决方法，可以在拷贝后，在主服务器上立即运行RESET MASTER，或者在开始复制前运行CHANGE MASTER TO语句，这样可以将其设置在正确的开始点上，正像在“利用主服务器上已被激活的二进制日志进行复制”一节中所提到的（这需要用skip-slave-start参数启服务器）。

避免关键错误

这个例子展示了当数据在从属服务器上更新而导致关键错误时发生的情况。当你用循环方式在主和从属服务器上进行复制，或直接更新到从属服务器时，通常会导致这样的错误。

因此，要在replication_table表中加入优先级锁。可以修改存在的表，如下所示：

```
mysql> ALTER TABLE replication_table MODIFY f1 INT NOT NULL,ADD PRIMARY KEY(f1);
Query OK, 0 rows affected (0.36 sec)
```

或者，再建一个表，如下所示：

```
mysql> CREATE TABLE replication_table(f1 INT NOT NULL,f2 VARCHAR(20),
 PRIMARY KEY(f1));
Query OK, 0 rows affected (0.03 sec)
```

加入一些记录到主服务器，然后重设主服务器，这样就不会重复前一个例子中的错误：

```
mysql> INSERT INTO replication_table (f1,f2) VALUES(1,'first');
mysql> INSERT INTO replication_table (f1,f2) VALUES(2,'second');
mysql> RESET MASTER;
```

拷贝数据到从属服务器，然后启动服务器开始复制。在主服务器上加入一条新记录：

```
mysql> INSERT INTO replication_table (f1,f2) VALUES(3,'third');
Query OK, 1 row affected (0.01 sec)
```

从属服务器上的数据现在应显示如下：

```
mysql> SELECT * FROM replication_table;
+----+--------+
| f1 | f2     |
+----+--------+
|  1 | first  |
|  2 | second |
|  3 | third  |
+----+--------+
3 rows in set (0.00 sec)
```

目前为止，一切运行正常。如果你在从属服务器上加入一条记录，然后又在主服务器上加入同一条记录，问题就出现了。在这个例子中，故意将键设置成相同的，但当使用AUTO_INCREMENT域时，很可能发生错误。先在从属服务器上插入如下的记录，然后是主服务器：

```
mysql> INSERT INTO replication_table (f1,f2) VALUES(4,'fourth');
Query OK, 1 row affected (0.01 sec)
```

虽然当查看两个服务器的数据时，它们是相同的，但在从属服务器上实际上已出现了错误，因为它试图两次INSERT记录。除非你在配置文件中使用有风险的slave-skip-errors参

数，复制现在就会停止，从属服务器将报告错误信息，如下所示：

```
mysql> SHOW SLAVE STATUS;
+---------------+-----------------+-------------+---------------+-----
--------------+--------------------+-----------------------+------
---------+---------------------+-----------------+-----------------
-+-----------------+--------------------+-----------+---------------
-----------------------------------------------------------------------
------------------+-------- -------+--------------------+-------------
---+
| Master_Host    | Master_User       | Master_Port | Connect_retry |
Master_Log_File     | Read_Master_Log_Pos | Relay_Log_File           |
Relay_Log_Pos | Relay_Master_Log_File | Slave_IO_Running |
Slave_SQL_Running | Replicate_do_db | Replicate_ignore_db | Last_errno
| Last_error
| Skip_counter | Exec_master_log_pos | Relay_log_space |
+---------------+-----------------+-------------+---------------+-----
---------------+-----  ---------------+-----------------------+------
---------+---------------------+-----------------+-----------------
-+-----------------+--------------------+-----------+---------------
-----------------------------------------------------------------------
------------------+-------------+--------------------+-------------
---+
| 192.168.4.100 | replication_user | 3306        | 60            |
g-bin.001 | 279                  | s-bin.002 |
224           | g-bin.001     | Yes               | No
| replication_db  |                      | 1062        | error 'Duplicate
entry '4' for key 1' on query 'INSERT INTO replication_table(f1,f2)
values(4,'fourth')' | 0             | 179                  | 332
|
+---------------+-----------------+-------------+---------------+-----
---------------+--------------------+-----------------------+------
---------+---------------------+-----------------+-----------
-------+----------------+--------------------+-----------+----------
-----------------------------------------------------------------------
----------------------+-------------+--------------------+---------
---------+
1 row in set (0.00 sec)
```

错误报告是明确的，因此能调查错误的原因并采取行动。在这个例子中，错误是：在不恰当的时候，语句在从属服务器上被重复了。可以使从属服务器再次正常复制的方法是，告诉它跳过主控二进制日志的下一条命令，然后从那里继续运行。可以用SET SQL_SLAVE_SKIP_COUNTER命令实现这一步。一旦这样做有了效果，就能启动从属服务器（当复制停止时，只能告诉从属服务器跳过），它将像过去一样继续运行，如下所示：

```
mysql> SET GLOBAL SQL_SLAVE_SKIP_COUNTER=1;
Query OK, 0 rows affected (0.00 sec)
mysql> SLAVE START;
Query OK, 0 rows affected (0.00 sec)
mysql> SHOW SLAVE STATUS;
+---------------+-----------------+-------------+---------------+-----
--------------+------------------+----------------------+------
---------+----------------------+-----------------+-----------------
-+-----------------+--------------------+------------+------------+---
----------+--------------------+-----------------+
| Master_Host    | Master_User       | Master_Port | Connect_retry |
Master_Log_File     | Read_Master_Log_Pos | Relay_Log_File            |
Relay_Log_Pos | Relay_Master_Log_File | Slave_IO_Running |
Slave_SQL_Running | Replicate_do_db | Replicate_ignore_db | Last_errno
| Last_error | Skip_counter | Exec_master_log_pos | Relay_log_space |
+---------------+-----------------+-------------+---------------+-----
---------------+--------------------+----------------------+------
---------+----------------------+-----------------+-----------------
-+-----------------+--------------------+------------+------------+---
----------+--------------------+-----------------+
| 192.168.4.100 | replication_user | 3306        | 60            |
g-bin.001 | 378                  | s-bin.002 |
423           | g-bin.001     | Yes               | Yes
| replication_db  |                      | 0           |               | 0
| 378                 | 431              |
+---------------+-----------------+-------------+---------------+-----
---------------+--------------------+----------------------+------
---------+----------------------+-----------------+-----------------
-+-----------------+--------------------+------------+------------+---
----------+--------------------+-----------------+
1 row in set (0.00 sec)
```

在主服务器上插入一条记录：

```
mysql> INSERT INTO replication_table(f1,f2) values(5,'fifth');
Query OK, 1 row affected (0.00 sec)
```

它在从属服务器上将再次被复制：

```
mysql> SELECT * FROM replication_table;
+----+--------+
| f1 | f2     |
+----+--------+
|  1 | first  |
|  2 | second |
|  3 | third  |
|  4 | fourth |
```

```
| 5 | fifth |
+---+-------+
5 rows in set (0.00 sec)
```

小结

复制是一种很有用的功能，它允许你准确地把数据从一个服务器拷贝到另一个服务器上。主数据库写入二进制日志，任意多的从属服务器与主服务器相连，然后读取二进制日志，并在它们自己的服务器上复制这些语句。

复制也是一种有用的备份方式，同时能提高性能。

主服务器二进制日志和从属服务器的**master.info**文件之间的关系，是保持同步的关键。如果在从属服务器使用这些日志之前它们被清除了，则复制将会失败。

第13章 配置并优化MySQL

只要你能够掌握mysqld变量复杂的使用方法及如何影响MySQL的性能，那么就可以声称已经通晓MySQL了。本章通过解释如何设置变量和在提高性能时所要寻找的方法，来告诉读者如何配置和优化MySQL。

本章要点：

- mysqld参数和变量
- 详细解释如何优化table_cache、key_buffer_size、delayed_queue、back_log和sort_buffer变量
- 处理更多的连接
- 改变变量而无需重新启动服务器
- 配置InnoDB表
- 改进硬件设施
- 运行基准测试程序
- 在ANSI模式下运行MySQL
- 使用不同的语言和字符集

优化mysqld变量

为了查看当前mysqld变量的值，可以使用命令行方式的mysqladmin命令：

```
% mysqladmin -uroot -pg00r002b variables
```

或在连接到MySQL时使用：

```
mysql> SHOW VARIABLES
+-----------------------+-----------------------------------+
| Variable_name         | Value                             |
+-----------------------+-----------------------------------+
| back_log              | 50                                |
| basedir               | /usr/local/mysql-max-4.0.1-       |
|                       |  alpha-pc-linux-gnu-i686          |
| bdb_cache_size        | 8388600                           |
| bdb_log_buffer_size   | 32768                             |
| bdb_home              | /usr/local/mysql/data/            |
| bdb_max_lock          | 10000                             |
| bdb_logdir            |                                   |
| bdb_shared_data       | OFF                               |
| bdb_tmpdir            | /tmp/                             |
| bdb_version           | Sleepycat Software: Berkeley DB   |
```

```
|                          |  3.2.9a: (December 23, 2001)     |
| binlog_cache_size        | 32768                            |
| character_set            | latin1                           |
| character_sets           | latin1 big5 czech euc_kr gb2312  |
|                          |  gbk latin1_de sjis tis620 ujis  |
|                          |  dec8 dos german1 hp8 koi8_ru    |
|                          |  latin2 swe7 usa7 cp1251 danish  |
|                          |  hebrew win1251 estonia          |
|                          |  hungarian koi8_ukr win1251ukr   |
|                          |  greek win1250 croat cp1257      |
|                          |  latin5                          |
| concurrent_insert        | ON                               |
| connect_timeout          | 5                                |
| datadir                  | /usr/local/mysql/data/           |
| delay_key_write          | ON                               |
| delayed_insert_limit     | 100                              |
| delayed_insert_          |                                  |
|  timeout                 | 300                              |
| delayed_queue_size       | 1000                             |
| flush                    | OFF                              |
| flush_time               | 0                                |
| ft_min_word_len          | 4                                |
| ft_max_word_len          | 254                              |
| ft_max_word_len_for_     |                                  |
|  sort                    | 20                               |
| ft_boolean_syntax        | + -><()~*:""&|                   |
| have_bdb                 | YES                              |
| have_innodb              | YES                              |
| have_isam                | YES                              |
| have_raid                | NO                               |
| have_symlink             | YES                              |
| have_openssl             | NO                               |
| init_file                |                                  |
| innodb_additional_       |                                  |
|  mem_pool_size           | 1048576                          |
| innodb_buffer_pool       |                                  |
|  size                    | 8388608                          |
| innodb_data_file_path    | ibdata1:64M                      |
| innodb_data_home_dir     |                                  |
| innodb_file_io_          |                                  |
|  threads                 | 9                                |
| innodb_force_recovery    | 0                                |
| innodb_thread_           |                                  |
|  concurrency             | 8                                |
| innodb_flush_log_at_     |                                  |
```

trx_commit	OFF
innodb_fast_shutdown	OFF
innodb_flush_method	
innodb_lock_wait_ timeout	1073741824
innodb_log_arch_dir	
innodb_log_archive	OFF
innodb_log_buffer_ size	1048576
innodb_log_file_size	5242880
innodb_log_files_in_ group	2
innodb_log_group_ home_dir	./
innodb_mirrored_log_ groups	1
interactive_timeout	28800
join_buffer_size	131072
key_buffer_size	16773120
language	/usr/local/mysql-max-4.0.1-alpha -pc-linux-gnu-i686/share/mysql/ english/
large_files_support	ON
locked_in_memory	OFF
log	ON
log_update	OFF
log_bin	ON
log_slave_updates	OFF
log_long_queries	ON
long_query_time	20
low_priority_updates	OFF
lower_case_table_ names	0
max_allowed_packet	1047552
max_binlog_cache_size	4294963200
max_binlog_size	1073741824
max_connections	100
max_connect_errors	10
max_delayed_threads	20
max_heap_table_size	16777216
max_join_size	4294967295
max_sort_length	1024
max_user_connections	0
max_tmp_tables	32
max_write_lock_count	4294967295

myisam_bulk_insert_	
tree_size	8388608
myisam_max_extra_	
sort_file_size	256
myisam_max_sort_	
file_size	2047
myisam_recover_	
options	OFF
myisam_sort_buffer_	
size	8388608
net_buffer_length	7168
net_read_timeout	30
net_retry_count	10
net_write_timeout	60
open_files_limit	0
pid_file	/usr/local/mysql/data/host.pid
port	3306
protocol_version	10
record_buffer	131072
record_rnd_buffer	131072
rpl_recovery_rank	0
query_buffer_size	0
query_cache_limit	1048576
query_cache_size	0
query_cache_startup_	
type	1
safe_show_database	OFF
server_id	1
slave_net_timeout	3600
skip_external_locking	ON
skip_networking	OFF
skip_show_database	OFF
slow_launch_time	2
socket	/tmp/mysql.sock
sort_buffer	524280
sql_mode	0
table_cache	64
table_type	MYISAM
thread_cache_size	0
thread_stack	65536
transaction_isolation	READ-COMMITTED
timezone	SAST
tmp_table_size	33554432
tmpdir	/tmp/
version	4.0.1-alpha-max-log
wait_timeout	28800

当调整是服务器自身提供的信息时，也很重要。通过命令行方式的命令来查看：

```
% mysqladmin extended-status
```

或在连接到MySQL时使用：

```
mysql> SHOW STATUS
+--------------------------+----------+
| Aborted_clients          | 142      |
| Aborted_connects         | 5        |
| Bytes_received           | 9005619  |
| Bytes_sent               | 15444786 |
| Connections              | 794      |
| Created_tmp_disk_tables  | 1        |
| Created_tmp_tables       | 716      |
| Created_tmp_files        | 0        |
| Delayed_insert_threads   | 0        |
| Delayed_writes           | 0        |
| Delayed_errors           | 0        |
| Flush_commands           | 1        |
| Handler_delete           | 27       |
| Handler_read_first       | 1534     |
| Handler_read_key         | 608840   |
| Handler_read_next        | 652228   |
| Handler_read_prev        | 164      |
| Handler_read_rnd         | 14143    |
| Handler_read_rnd_next    | 1133372  |
| Handler_update           | 90       |
| Handler_write            | 131624   |
| Key_blocks_used          | 6682     |
| Key_read_requests        | 2745899  |
| Key_reads                | 6026     |
| Key_write_requests       | 63925    |
| Key_writes               | 63790    |
| Max_used_connections     | 20       |
| Not_flushed_key_blocks   | 0        |
| Not_flushed_delayed_rows | 0        |
| Open_tables              | 64       |
| Open_files               | 128      |
| Open_streams             | 0        |
| Opened_tables            | 517      |
| Questions                | 118245   |
| Select_full_join         | 0        |
| Select_full_range_join   | 0        |
| Select_range             | 2300     |
| Select_range_check       | 0        |
```

```
| Select_scan                | 642     |
| Slave_running              | OFF     |
| Slave_open_temp_tables     | 0       |
| Slow_launch_threads        | 0       |
| Slow_queries               | 8       |
| Sort_merge_passes          | 0       |
| Sort_range                 | 3582    |
| Sort_rows                  | 16287
| Sort_scan                  | 806     |
| Table_locks_immediate      | 82957   |
| Table_locks_waited         | 2       |
| Threads_cached             | 0       |
| Threads_created            | 793     |
| Threads_connected          | 1       |
| Threads_running            | 1       |
| Uptime                     | 1662790 |
+----------------------------+---------+
```

变量列表及其状态信息会随着变量每次被重新释放后变得越来越长。读者使用的版本的信息可能比上表所列的还要多，应该阅读最新的文档，确切地看一下这些多出来的信息是什么。对当前正在使用的信息的完整说明将在本章稍后的表13.2中解释。

绝大多数MySQL版本带有4种样本配置文件：

my-huge.cnf 用于内存大于1GB的系统，此内存主要由MySQL使用。

my-large.cnf 用于内存至少为512MB的系统，此内存主要由MySQL使用。

my-medium.cnf 用于内存至少为32MB的系统，此内存完全由MySQL使用，或机器上至少有128MB的内存，用于多种目的（如双Web/数据库服务器）的系统。

my-small.cnf 用于内存少于64MB的系统， MySQL不能占用太多资源的情况。

这些文件通常可以在：RPM安装模式下的/usr/share/doc/packages/MySQL/目录中、UNIX二进制安装模式的/usr/local/mysql-max-4.x.x-platform-operating-system-extra/support-files/目录中或Windows的C:\mysql\目录中找到。

警告：在Windows系统中，.cnf扩展名的文件会与FrontPage和NetMeeting发生冲突。

在最初使用时，建议用这些配置中的一种来替换my.cnf文件（或my.ini文件），要选择最接近读者要求的配置方法。

给系统选择正确的配置将有助于向最优化迈进一大步，但为了达到最佳使用效果，要求对系统和特殊的使用方法进行很好的调整。

优化table_cache变量

变量table_cache是最有用的需要进行调整的变量之一。每次MySQL访问表的时候，如果缓存中有空间，那么表就被放置在缓存中。访问内存中的表要比表在磁盘上快。通过检查open_tables在高峰时间的值（用命令SHOW STATUS或mysqladmin variables查看到的状态扩展值之一），可以判断出是否要增加table_cache的值。如果发现open_tables的值与

table_cache的值相同，并且opened_tables的值（另一个状态扩展值）正在增加，就应该在有足够内存的情况下增加table_cache的值。

说明： 打开的表的数量可以比数据库中的表的数量多。MySQL是多线程的，许多条查询语句可以同时运行，每一条查询语句都可以打开一个表。

参考以下三种在高峰时间的真实情景。

情景1 这个情景来自一个正在工作但并不是很忙的服务器：

```
table_cache - 512
open_tables - 103
opened_tables - 1723
uptime - 4021421 (measured in seconds)
```

在这个例子中，table_cache的值看上去设得并不是太高。服务器已经启动了很长一段时间（如果服务器刚刚启动，你无法知道是否很快达到了table_cache的值，或者opened_tables的值是否很快开始增加）。考虑到这是一个高峰时间，被打开的表的数量相当得少，并且打开的表的数量恰好在应该的数值以下。

情景2 这个情景来自一台开发服务器。

```
table_cache - 64
open_tables - 64
opened_tables - 431
uptime - 1662790 (measured in seconds)
```

此处考虑到服务器已经启动一段时间了，尽管open_tables达到了最大值，但open_tables的值相当低。很可能增加table_cache的值也不会有什么作用。

情景3 这个情景来自一台工作状态不佳的服务器：

```
table_cache - 64
open_tables - 64
opened_tables - 22423
uptime - 19538
```

在这个实例中，table_cache的值被设置得太低。尽管uptime的值少于6个小时，但open_tables达到了最大值，并且open_tables的值很高。如果系统还有剩余的内存可以获得，就应该增加table_cache的值。

警告： 不要盲目地把table_cache的值设置得太高。如果你不需要这么高的值，就保持table_cache的值在一个合理的范围。如果这个值设置得太高，可能会用完文件描述符，导致性能不稳定或连接被拒绝。

优化key_buffer_size

key_buffer_size的值会影响索引缓存的大小，在特殊索引读入的时候，索引缓存的大小又会影响索引处理的速度。值越高，在内存中可以保留的MySQL索引就越多，这样就比从磁盘进行访问快得多。一条来自经验的建议是把key_buffer_size的值设置在服务器可用内存

值的1/4到1/2之间（如果你的服务器专门用于MySQL）。通过比较key_read_requests和key_reads的状态值，就可以获得如何调整key_buffer_size值的好方法。key_reads与key_read_requests的比值应该尽可能的低，1:100是可以接受的最高限值（1:1000比较好，1:10就太糟糕了）。key_reads的值表示键要被从磁盘上读取的次数，这就是应该尽可能避免将键的缓存设置过高的原因。

下面的情景展示了两种可能性。

情景1 正常状态：

```
key_buffer_size - 402649088 (384M)
key_read_requests - 597579931
key_reads - 56188
```

情景2 应该警惕的状态：

```
key_buffer_size - 16777216 (16M)
key_read_requests - 597579931
key_reads - 53832731
```

情景1反映的是正常的状态，比值超过了1:10000。情景2中的状况应值得关注，此处比值达到了令人担忧的1:11。作为一种解决方案，应该增加key_buffer_size的值达到内存允许的范围。如果没有足够的内存达到这个要求，就要升级你的硬件设备了。

查看key_writes与key_write_requests的比值很有用。当每次主要是插入和更新一条记录时，这个比值通常接近于1；但如果每次经常插入或更新大批量数据，这个值应该低一些。使用语句INSERT DELAYED也可以降低这个比值。

如何处理连接太多的情况

一个常见（并且有时候很容易确定）的错误在系统过于繁忙的时候会发生，就是错误Too many connections。当threads_connected的值经常超过max_connections的时，就是应该做出改变的时候。如果查询语句能够被顺利地处理，增加max_connections的值就是最简单的解决方案。

大多数应用程序会使用持久连接，而不是普通的连接（例如在PHP中，使用函数pconnect()，而不是connect()函数）。即使查询已经运行完毕，持久不变的连接也会保持打开状态。在繁忙的服务器上，这意味着下一条查询语句不会占用任何资源去再次做连接。维持大量持久的但不经常使用的连接，要比快速连续地进行连接、断开连接和重新连接的操作节省资源。

说明： 在CGI模式下不能使用持久连接，而且在Apache web服务器上会受到KeepAlive设置的影响。

这个情景是一台在超负荷状态下使用持久连接的Web服务器：

```
max_connections - 250
max_used_connections - 210
threads_connected - 202
threads_running - 1
```

在这个情景中，看上去好像是MySQL在消耗资源，其实很简单，根据服务器实例的数目有202个threads_connected持久连接，它们几乎不占用任何资源。实际上，只有一个线程在运行，所以数据库的负担很可能不会太重。threads_connected的值不断接近max_connections的值是不会有任何问题的，但为了避免超过连接数量的限制应该增加max_connections的值。通过查看max_used_connections来了解连接数量与最大值之间的差距，如果接近或等于max_connections的值，就到了为连接的增长留有余量的时候了。

从个人的观点看，我认为持久连接更好。虽然有一些报道说，因为MySQL连接的开销比其他数据库（如Oracle，在大多数情况下你不得不使用持久连接）要高得多，所以不会有什么差别，甚至会使性能下降。最好的建议是去测试一下你自己系统的性能。

警告：测试的时候，要严格测试负载状态下的性能。网上有一些文档，是有关持久连接和非持久连接的各种错误比较。

在上一个情景的系统中，threads_running值的不断增加经常是数据库不能承受负荷的暗示。检查过程列表可以帮助识别出查询语句导致的死锁。在数据库服务器崩溃之前，找到随后要发生的是数据库服务器输出结果的哪一部分。threads_connected的值继续增加直到服务器无法再处理并且瘫痪为止。命令Processlist的输出结果可以帮助识别出有问题的查询语句：

```
% mysqladmin processlist;
Id     User   Host       Db      Command Time State  Info
6464   mysql  websrv2...news     Sleep   590
6482   mysql  websrv2...news     Sleep   158
6486   mysql  websrv2...news     Sleep   842
7549   mysql  websrv2...news     Sleep   185
8126   mysql  websrv2...news     Sleep   349
9938   mysql  websrv2...news     Sleep   320
1696   mysql  websrv2...news     Sleep   100
4143   mysql  websrv2...news     Sleep   98
5071   mysql  websrv2...news     Sleep   843
5135   mysql  websrv2...news     Sleep   155
92707  mysql  zubat...   news    Sleep   530
93014  mysql  zubat...   news    Query   13  Locked select s_id from arts
                                            where a_id =
                                            'E232625'
93060  mysql  zubat...   news    Sleep   190
93096  mysql  zubat...   news    Query   171 Copying
                                           to tmp
                                           table   select distinct
                                                    arts.a_id,
                                                    arts.headline1,
                                                    nartsect.se_id,
                                                    arts.mdate,
                                                    arts.set_
93153  mysql  zubat...   news    Sleep   207
```

93161	mysql	zubat...	news	Query	30	Locked	SELECT DISTINCT arts.a_id FROM arts,keywordmap WHERE arts.s_id in (1) AND arts.
93165	mysql	zubat...	news	Query	36	Locked	select arts.name, arts.headline1, arts.se_id, n_blurb.blurb, slook.name as sectna
93204	mysql	zubat...	news	Query	31	Locked	select arts.name, arts.headline1, arts.se_id, n_blurb.blurb, slook.name as sectna
93205	mysql	zubat...	news	Query	156	Copying to tmp table	select distinct arts.a_id, arts.headline1, nartsect.sc_id, arts.mdate, arts.set_
93210	mysql	zubat...	news	Query	50	Locked	select arts.a_id, arts.headline1, nartfpg.se_id, nartfpg.se_id, arts.mdate, nartfpg.p
93217	mysql	zubat...	news	Query	38	Locked	select arts.name, arts.headline1, arts.se_id, n_blurb.blurb, slook.name as sectna
93222	mysql	zubat...	news	Query	8	Locked	select arts.name, arts.headline1, arts.se_id, n_blurb.blurb, slook2.name as sectn
93226	mysql	zubat...	news	Query	39	Locked	select arts.name, arts.headline1,

```
                                                                arts.se_id,
                                                                n_blurb.blurb,
                                                                slook.name as
                                                                sectna
93237  mysql   zubat...  news  Query    33  Locked   select arts.a_id,
                                                                arts.headline1,
                                                                nartfpg.se_id,
                                                                nartfpg.se_id,
                                                                arts.mdate,
                                                                nartfpg.p
93244  mysql   zubat...  news  Query    99  Copying
                                            to tmp
                                            table    select distinct
                                                                arts.a_id,
                                                                arts.headline1,
                                                                nartsect.se_id,
                                                                arts.mdate,
                                                                arts.set_
93247  mysql   zubat...  news  Query    64  Locked   select s_id from
                                                                arts where
                                                                a_id='32C436'
93252  mysql   zubat...  news  Query   120  Copying
                                            to tmp
                                            table    select distinct
                                                                arts.a_id,
                                                                arts.headline1,
                                                                nartsect.se_id,
                                                                arts.mdate,
                                                                arts.set_
93254  mysql   zubat...  news  Sleep    47
93256  mysql   zubat...  news  Sleep   171
93257  mysql   zubat...  news  Query   176  Copying
                                            to tmp
                                            table    select distinct
                                                                arts.a_id,
                                                                arts.headline1,
                                                                nartsect.se_id,
                                                                arts.mdate,
                                                                arts.set_
93261  mysql   zubat...  news  Sleep   349
93262  mysql   zubat...  news  Sleep   1
93263  mysql   zubat...  news  Query   153  Copying
                                            to tmp
                                            table    select distinct
                                                                arts.a_id,
```

```
                                                                arts.headline1,
                                                                nartsect.se_id,
                                                                arts.mdate,
                                                                arts.set_
93267  mysql   zubat...  news  Query    27   Locked  select arts.name,
                                                                arts.headline1,
                                                                arts.se_id,
                                                                n_blurb.blurb,
                                                                slook.name as
                                                                sectna
93276  mysql   zubat...  news  Query    29   Locked  select arts.name,
                                                                arts.headline1,
                                                                arts.se_id,
                                                                n_blurb.blurb,
                                                                slook.name as
                                                                sectna
93278  mysql   zubat...  news  Query    183  Copying
                                                    to tmp
                                                    table       select distinct
                                                                arts.a_id,
                                                                arts.headline1,
                                                                nartsect.se_id,
                                                                arts.mdate,
                                                                arts.set
93280  mysql   zubat...  news  Sleep    36
93285  mysql   zubat...  news  Sleep    10
93284  mysql   zubat...  news  Query    49   Locked  select arts.name,
                                                                arts.headline1,
                                                                arts.se_id,
                                                                n_blurb.blurb,
                                                                slook.name as
                                                                sectna
```

列表中显示出了两台Web服务器，服务器websrv2运行正常（它所有线程的任务都完成了，并且连接都处于睡眠状态），但服务器zubat却存在成堆的查询问题。

事实上有很多查询语句，这里只是完整列表中的很小一部分。在这个例子中你要查看的查询语句的开始部分如下所示：

```
select distinct arts.a_id, arts.headline1, nartsect.se_id, arts.mdate, arts.set ...
```

注意所有这些查询语句的状态都是Copying to tmp table，其他的是Locked。这个例子中的问题是，开发者对查询语句做了修改以至于不必再使用索引了。在第4章“索引和查询优化”中更详细地讨论过这个话题。

经常检查processlist的输出结果可以在引起像服务器瘫痪这样灾难之前，帮助识别出那些运行缓慢的查询语句。

slow_queries的值是另一个应该查看的结果。如果这个值总在缓慢地增长，很可能表明有问题存在。调整好的系统应尽可能少地有运行缓慢的查询语句。一些复杂的连接语句不可避免地会使速度很慢，但更有可能的是那些慢的查询语句严重地影响了性能最优化。

优化delayed_queue_size和back_log变量

正如在第4章中所讨论的，语句INSERT DELAYED释放客户程序，但如果队列中还有其他语句就不会立刻处理查询语句。MySQL不是这样，它等待空隙处理插入语句。在这里变量delayed_queue_size发挥了作用。如果变量被设置成默认值1000，这意味着1000个超时的语句将排队等待之后，客户程序将不再被释放，也只能等待。有如此多的排队等待的查询语句通常不是什么好事，但如果你的系统几乎在同一时刻产生了大量的插入语句，并且发现即便使用了语句INSERT DELAYED，客户程序仍旧需要等待，你就应该增加delayed_queue_size的值。

另一个用来帮助管理短暂突发行为的变量是back_log。如果一个系统在很短的时间间隔内收到大量的连接请求，MySQL将统计那些作为待处理事项一部分的还没有被处理的连接的数量。只要一达到back_log的限定值，任何更多的请求将被排进队列而不是被拒绝。如果系统突然间得到大量的连接请求，并且由于这个原因导致一些请求被拒绝，就应该增加back_log值。如果系统只是忙，并且请求数量是稳定的，那么只增加back_log值本身是不会有多少作用的。这样虽然能给服务器喘息的时间去处理突发事件，但对于负担过重的系统不会有帮助。

优化sort_buffer变量

就变量sort_buffer能够加快myisamchk的运算速度（在第10章“基本管理”中）的内容已经讨论过了，对于精确调整常规运算的速度这也是一个很有用的变量。如果你通常做许多排序运算（例如，经常在很大的表中使用ORDER BY语句），改用sort_buffer是很有帮助的。配置文件my-huge.cnf（用于至少有1GB内存的系统）默认sort_buffer对myisamchk使用256M内存，对mysqld使用2M内存。尽管希望mysqld图表能够处理大量排序，但如果有许多同时发生的连接在执行从句ORDER BY，又因为每一个连接都被分配了一个变量sort_buffer，所以会出现内存问题。

配置InnoDB表

为了让表InnoDB运行稳定，正确配置变量的要求比表MyISAM严格得多。最重要的是变量innodb_data_file_path，它用来指定表（数据和索引）可以获得的空间。它指定一个或多个数据文件，还给文件分配大小。应该让最后一个数据文件自动扩展（只有最后一个数据文件能这样做）。所以，不是在所有的空间被占用的时候只简单地用掉空间，自动扩展的数据文件将不断增长以容纳额外的数据。例如：

```
innodb_data_file_path=/disk1/ibdata1:900M;/disk2/ibdata2:50M:autoextend
```

这里的两个数据文件放置在不同的磁盘上（分别称做disk1和disk2）。数据先被放在ibdata1中，直到达到900MB的限定值，此后数据将被放入ibdata2中。一旦达到50MB的限制，

ibdata2将自动以8MB的程序块进行扩展。

如果磁盘的物理空间满了，就需要在另外一块磁盘上增加另一个数据文件，这要求进行一些手工配置。为了这样做，先查看最后一个数据文件占用物理空间的大小，然后四舍五入到最接近的兆字节数。明确设定这个数据文件的大小，并添加新的数据文件说明。例如，如果预先指定disk2中ibdata2的最大空间是109MB，就要使用类似下面这样的定义：

```
innodb_data_file_path=/disk1/ibdata1:900M;/disk2/ibdata2:109M;/disk3/
ibdata3:500M:autoextend
```

为了让改变生效，需要重新启动服务器。

介绍mysqld参数

表13.1描述了mysqld的参数。

表13.1 mysqld参数

参数	描述
--ansi	MySQL不仅有许多扩展名，还与标准ANSI（标准SQL，由美国国家标准协会定义）有许多不同之处。如果设置了这个参数，MySQL将在ANSI模式下（这个原因导致的改变将在本章稍后一节“在ANSI模式下运行MySQL”中讨论）运行
-b, --basedir=path	路径指向根目录或MySQL安装目录。其他路径通常相对于这个路径选取
--big-tables	当内存不足时，通过节省文件上的所有临时设置而允许产生大的输出结果
--bind-address=IP	用Internet协议（IP）地址或主机名捆绑MySQL
--character-sets-dir=path	字符集所在的目录
--chroot=path	出于安全目的，可以用这个参数在chroot环境中启动MySQL。这会引起MySQL隐藏起完整的目录结构，在目录的一个子集中运行。然而，这种做法限制了语句LOAD DATA INFILE和SELECT ... INTO OUTFILE的使用
--core-file	这个参数在mysqld意外终止的时候，会引起核心文件被写入。一些系统可能会要求指定--core-file-size参数。如果参数--user被使用，有些系统就不会写核心文件
-h, --datadir=path	数据所在目录的路径
--debug[...]=	如果MySQL被配置使用--with-debug参数，则这个参数会被用来产生一个记录mysqld活动的跟踪文件
--default-character-set=charset	用来设置默认的字符集（默认值是latin1）
--default-table-type=type	用来设置默认的表类型（如果在CREATE语句中没有指定表的类型，则会产生这种类型的表）。默认状态下，表类型是*MyISAM*

（续表）

参数	描述
--delay-key-write-for-all-tables	使用这个参数，MySQL不会刷新任何用于MyISAM表的写语句间的键的缓存
--des-key-file=filename	从这个文件读取默认的键。这个参数由函数DES_ENCRYPT()和DES_DECRYPT()使用
--enable-external-locking	这个参数能让系统锁定。此参数在锁定的后台程序不能完全工作的系统中不应该被使用（此参数应用于Linux系统，尽管新的版本中已经不存在这样的问题了）
--enable-named-pipe	在Windows NT/2000/XP中，这个参数能够支持命名管道
-T, --exit-info	在调试过程中使用不同标记的位掩码。除非知道自己在做什么，否则不建议使用这个参数
--flush	这个参数确保MySQL在执行完每一条SQL语句后，刷新磁盘上所有的变更。通常由操作系统进行处理，除非遇到问题，否则不需要使用这个参数
-?, --help	显示所有帮助列表和退出程序
--init-file=file	当启动的时候，告诉MySQL去执行包含这个文件的SQL语句
-L, --language=...	这个参数用来设置用于客户出错信息的语言。可以是语言种类，也可以是语言文件所在的完整路径
-l, --log[=file]	在指定的文件中对连接和查询操作进行日志记录
--log-isam[=file]	这个参数在指定的文件中，把所有对MyISAM或ISAM文件进行的变更记入日志（只有在调试这些表的类型时才使用）
--log-slow-queries[=file]	把执行时间长于变量long_query_time值（以秒计算）的查询操作记入慢查询日志文件
--log-update[=file]	把所有更新操作记入指定的更新日志文件，而不是使用参数--log-bin
--log-bin[=file]	把所有更新操作记入指定的二进制更新日志文件
--log-long-format	记录更多的信息。如果慢查询日志文件被使用（--log-slow-queries），任何不使用索引的查询操作也被记入日志
--low-priority-updates	如果这个参数被使用，所有的插入、更新和删除操作将比选择语句拥有更低的优先权。如果你不想此参数应用到所有的查询语句，可以用SET OPTION SQL_LOW_PRIORITY_UPDATES=1对指定的线程应用，或LOW_PRIORITY ...应用于指定的INSERT、UPDATE或DELETE查询操作
--memlock	将mysqld锁定在内存中。这个参数只有在系统支持mlockall()函数（如Solaris系统中）时才能使用。通常只在操作系统出现问题和mysqld转储到磁盘上时，使用这个参数。通过查看变量locked_in_memory的值，你可以看到参数-memlock是否被使用

（续表）

参数	描述
--myisam-recover [=option[,option...]]]	可用的参数是DEFAULT、BACKUP、FORCE、QUICK或""。如果这个参数被设置为除""之外的任何值，它在MySQL启动的时候将检查表，看是否这些表被做上崩溃或非正常关闭的标记。如果是这样，此参数将检查表并试图修复损坏的表。如果BACKUP参数被使用，在修复期间发生任何改变，MySQL都将生成一个.MYD数据文件的备份文件（扩展名为.BAK）。参数FORCE用来进行强制修复，即使数据会被丢失。如果没有删除的数据块，参数QUICK不对行进行检查。所有的错误都将在错误日志中记录，所以可以查看发生的事情。同时设置BACKUP和FORCE参数，能让MySQL从许多问题中自动恢复（万一出错有备份文件）。参数DEFAULT相当于没有任何参数
--pid-file=path	指定pid（过程id）文件的路径
-P, --port=...	MySQL用来监听TCP/IP连接的端口号
-o, --old-protocol	指定MySQL使用旧的3.20协议用来与一些旧的相同的客户程序兼容
--one-thread	指定MySQL只能使用一个线程。你只能将这个线程用于调试
-O, --set-variable var=option	设置一个可以进行优化的变量。变量的完整列表（表13.2）跟在这张表后面
--safe-mode	跳过一些优化阶段。这个参数包含--skip-delay-key-write参数
--safe-show-database	只在最早的MySQL 4版本中使用。一旦设置，没有数据库使用权的用户在执行完SHOW DATABASES语句后，将无法看到数据库列表（SHOW DATABASES语句的权限删除了这种需求）
--safe-user-create	这个参数通过禁止用户生成新用户增加了安全性（使用GRANT语句），除非用户对mysql用户表或这个表中的列拥有INSERT权限
--skip-concurrent-insert	无效的并发插入操作（选择和插入操作可以在优化的表中同时被执行）。除非做调试，否则不需要用这个参数
--skip-delay-key-write	让MySQL忽略所有表中的delay_key_write参数
--skip-grant-tables	这个参数启动不带权限表（给每个人完全访问权）的MySQL。除非不得已（例如，作为root用户忘记密码的时候），不要使用这个参数。一旦参数设置完毕，再次使用权限表运行mysqladmin flush-privileges或mysqladmin reload去启动MySQL
--skip-host-cache	主机名通常被放入高速缓存，但你可以强制MySQL为每一个连接查询DNS服务器。这样做将降低连接速度
--skip-external-locking	禁用系统锁定。这对myisamchk有重要影响，见第10章中的“用myisamchk分析表”
--skip-name-resolve	MySQL不解析主机名，这样权限表中的所有Host值必须是一个具体的IP（或localhost）。主机名没有被解析，如果你有许多主机或速度较慢的DNS，这个参数可以提高连接的速度

（续表）

参数	描述
--skip-networking	这个参数会让MySQL只允许本地连接，它不会监听TCP/IP连接。如果可能，这是一个好的安全措施
--skip-new	MySQL使用ISAM作为默认的表类型，并且不使用版本3.23中的一些新参数。此参数还包含--skip-delay-key-write参数。除非此参数的功能发生改变，否则不应该再需要了
--skip-symlink	这个参数确保了任何人不能删除或更改数据目录中指向symlinked的文件及文件名。如果你没有使用symlinks作为安全措施来保证任何人不能丢弃或更改在mysqld数据目录外面的文件，就应该使用这个参数
--skip-safemalloc	这个参数由于避免了检查超出限度的内存分配和内存释放（当MySQL使用参数--with-debug=full时，才做这些检查），所以加快了服务器性能
--skip-show-database	一旦设置，SHOW DATABASES语句不会返回结果，除非客户程序拥有PROCESS权限（MySQL 4早期版本中引入的SHOW DATABASES权限删掉了这个要求）
--skip-stack-trace	不使用堆栈跟踪（如果在调试器下运行mysqld，这个参数很有用）。为获得核心文件，一些系统要求有这个参数
--skip-thread-priority	禁止使用具有更快响应时间优先级的线程
--socket=path	指向套接字文件的路径，此文件用于本地连接（不使用默认值，通常路径为/tmp/mysql.sock）
--sql-mode=option[,option[,option...]]	可以使用这些参数设置ANSI标准和MySQL之间的各种差异。它们是REAL_AS_FLOAT、PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、SERIALIZE、ONLY_FULL_GROUP_BY或""（用来重新复位）。所有参数的使用方法与-ansi参数相同。本章稍后的“在ANSI模式下运行MySQL”中将讨论每一和差异
--temp-pool	这个参数只有在操作系统生成大量不同文件名的新文件且发生内存泄漏时使用（某些版本的Linux发生过这种情况）。MySQL将使用很少一组文件名用于临时文件
--transaction-isolation= { READ-UNCOMMITTED | READ-COMMITTED | REPEATABLE-READ | SERIALIZABLE }	设置默认的事务隔离级别。参见第4章中的讨论
-t, --tmpdir=path	此目录用来存储临时文件和表。如果临时的空间太小以致不能容纳临时文件，这个参数对于改变存储路径非常有用
-u, --user= [user_name | userid]	提供一个用来运行MySQL的用户名。当作为root用户启动mysqld时，这个参数必须被使用

（续表）

参数	描述
-V, --version	显示版本信息和退出方法
-W, --warnings	警告信息将在错误文件中显示

了解mysqld变量

当用命令行方式运行mysqladmin variables时，或从mysql提示符下运行SHOW VARIABLES，一个很长的变量列表将会显示出来。大多数时候，它们与配置文件中设置的某个参数有关。表13.2对所显示的变量进行了解释。根据系统的设置和版本，你可能不会有全部这些参数，或者很可能因为MySQL不断向表中添加变量，你会有更多参数。

表13.2 mysqld变量

变量	描述
ansi_mode	MySQL不仅有很多扩展名，而且在与标准ANSI性能方面有很多不同之处。如果设置了这个参数，MySQL将在ANSI模式下运行（这个参数所引起的变化将在本章稍后一节“在ANSI模式下运行MySQL”中讨论）
back_log	排队等待的连接的数量请求MySQL在开始拒绝连接操作之前就已经在等待。这个参数与监听进来的TCP/IP连接所在队列的长度是一样的，参数的值也要受到操作系统的限制。较低的back_log值和操作系统限制将会得到应用。获得更多的信息，请参见操作系统的文档（例如，man在Unix上进行监听）
basedir	指向基准目录或MySQL安装目录的路径
bdb_cache_size	分配给缓存数据的缓冲区的大小和用于BDB表的索引的大小。如果你的系统不使用BDB表，则可以使用参数--skip-bdb来避免浪费内存空间
bdb_lock_detect	对Berkeley锁的检测，它可以是DEFAULT、OLDEST、RANDOM或YOUNGEST中的一个
bdb_log_buffer_size	用于BDB日志的缓冲区的大小。如果你的系统不使用BDB表，则可以使用参数--skip-bdb来避免浪费内存空间
bdb_home	BDB表的基准目录，此目录必须与--datadir相同
bdb_max_lock	BDB表可以使用的锁的最大数量。如果事务可能太长或查询时要求检查许多行，则可以增加这个值。类似的错误bdb: Lock table is out of available locks or Got error 12 from ...表明需要增加此变量值。默认的值是10000
bdb-no-recover	一旦设置，MySQL在恢复模式下不会启动BDB。通常只有在BDB日志中有崩溃记录从而阻止成功启动时，才设置这个变量
bdb-no-sync	一旦设置，MySQL将不会同步刷新日志
bdb_logdir	包含BDB日志的目录

（续表）

变量	描述
bdb_shared_data	一旦设置，MySQL在多进程模式下启动BDB，意味着不会使用DB_PRIVATE
bdb_tmpdir	存储BDB临时文件的目录
binlog_cache_size	用被写入事务的二进制日志的高速缓存的大小。如果事务很大，并且占用比默认的32KB缓存更多的空间，就应该增加这个变量的值
character_set	在没有指定其他字符集的时候（通常是latin1），设置所要使用的字符集
character_sets	系统支持的字符集的完整列表。如果你正在编译MySQL并且知道永远不会用到这些字符集，就可以编译不支持额外的字符集
concurrent_inserts	如果变量是激活状态（默认状态），就可以在查询的同时对MyISAM表进行插入操作，从而提升性能（只要表中没有以前删除记录后留下的空隙，通过经常优化这些表你就可以保证这一点）。参数--safe或--skip-new对此无效
connect_timeout	在Bad handshake（握手失败）超时之前，MySQL等待数据包的时间（以秒计算）。这会帮助阻止拒绝服务攻击（拒绝服务攻击是为了阻止合法用户进行连接，故意尝试许多次恶意连接）
datadir	数据被存储的目录
delay_key_write	如果变量是激活状态（默认状态），MySQL在每次索引更新带有DELAY_KEY_WRITE参数的表的时候，将不会刷新表使用的键的缓存。然而，当表被关闭的时候，缓存被刷新。这会增加键写入的速度，但它也会增加系统崩溃的几率。所以，应该经常检查表。你可以通过使用参数--delay-key-write-for-all-tables来指定参数DELAY_KEY_WRITE对所有的表都是默认参数。参数--safe或--skip-new对此无效
delayed_insert_limit	插入delayed_insert_limit指定的行以后，MySQL进行检查，如果还有SELECT语句未被执行，就在继续保留INSERT DELAYED语句之前，执行这些语句。默认的行数值是100
delayed_insert_timeout	这个变量决定INSERT DELAYED线程在结束之前，应该等待INSERT语句多长时间
delayed_queue_size	分配给INSERT DELAYED队列的行数。如果达到这个限定值，执行INSERT DELAYED语句的客户程序就将一直等待，到有空间为止
flush	一旦设置，MySQL将刷新所有每条SQL语句执行完毕后对磁盘的变更操作。通常操作系统会进行处理。除非有问题，否则你不需要使用这个变量，所以默认值为OFF状态
flush_time	自动刷新（此时表被关闭，并且在磁盘上进行同步更新）操作之间的时间（以秒计算）。通常变量值设为0，除非运行的系统只有很少的资源或是Windows 95/98/Me

（续表）

变量	描述
ft_min_word_len	FULLTEXT索引中包含的字的最小长度。改变这个变量的值以后，任何FULLTEXT索引将需要重建。此变量的默认值是4
ft_max_word_len	FULLTEXT索引中包含的字的最大长度。改变这个变量的值以后，任何FULLTEXT索引将需要重建。此变量的默认值是254
ft_max_word_len_sort	当索引被重建的时候，使用快速索引重建方法将这个长度的字，或比这个长度更短的字插入FULLTEXT索引。使用较慢的方法将比这个长度长的字插入索引。如果你的索引中的字的长度不是正常值，那么你很可能想改变这个默认值（20）。如果这个值设置得太高，会因为临时文件变得大造成进程变慢，并且键将在同一个分类区。如果值设得太高，会有很多字会被用慢的方法插入
ft_boolean_syntax	MATCH ... AGAINST(... IN BOOLEAN MODE)语句(+ -><()~*:""&|)支持的运算符的列表
have_innodb	如果MySQL支持InnoDB表，则变量设为YES，或者如果使用参数--skip-innodb，则变量设为DISABLED
have_bdb	如果MySQL支持BDB表，则变量设为YES，或者如果使用参数--skip-bdb，则变量设为DISABLED
have_raid	如果MySQL支持RAID（廉价磁盘冗余阵列）参数，则变量设为YES
have_openssl	如果MySQL支持客户端与服务器之间的SSL（安全套接层）加密技术，则变量设为YES
init_file	当服务器启动的时候，包含了要执行的SQL语句的文件名（默认情况下，不指定任何文件名）
innodb_data_home_dir	用于InnoDB数据文件的主目录，并作为路径的通用部分。如果不提，这个变量将与datadir相同。通过将这个变量设置为空字符串，可以在innodb_data_file_path中使用绝对文件路径
innodb_data_file_path	个人数据文件路径及文件大小。路径的innodb_data_home_dir部分被添加到这个变量中以给出完全路径。文件的大小用兆字节（M）或十亿字节（G）指定。在支持大文件的操作系统中，可以使用比4GB更大的文件，并且总和应该至少为10MB
innodb_mirrored_log_groups	指定为数据库保留的日志组的相同副本的数量。目前这个值为1
innodb_log_group_home_dir	指定到InnoDB日志文件的目录路径
innodb_log_files_in_group	指定日志组中日志文件的个数。3是建议的值（日志是循环写入的）
innodb_log_file_size	指定日志组中每个日志文件的大小（用兆字节为单位）。建议的值是从1MB到innodb_buffer_pool_size的1/innodb_log_files_in_group。因为要求减少刷新活动，所以设置高的值是为了节省磁盘的输入/输出（I/O）操作，但在系统崩溃后降低了恢复速度。在32位的计算机上，日志文件大小的总和不应该多于4GB

（续表）

变量	描述
innodb_log_buffer_size	指定用来写日志文件的缓冲区的大小。建议的值是8MB。缓冲区越大，建议的磁盘I/O值越小，因为直到事务被提交，否则事务不需要写入磁盘
innodb_flush_log_at_trx_commit	一旦设置，只要事务被提交，日志文件就被刷新到磁盘上（并且是永久性的，能在系统崩溃后继续存在）。如果事务很重要，除了ON以外，这个变量通常不设置为任何值。如果性能非常关键，并且考虑成本的安全性希望降低磁盘的I/O，可以将变量值设置为OFF
innodb_log_arch_dir	指定日志文件被存档的目录。通常这个变量应该与innodb_log_group_home_dir相同，因为日志归档目前已经不使用了
innodb_log_archive	一旦设置，InnoDB日志文件将被归档。目前，MySQL使用自己的日志文件进行恢复，所以变量应该被设置为OFF
innodb_buffer_pool_size	指定用于缓存表的索引和数据的内存缓冲区的大小（用字节为单位）。值越大，性能越好，因为要求的磁盘的I/O越少。在专用数据库服务器上建议的值是内存的80%（最大值），因为较大的值会引起操作系统分页
innodb_additional_mem_pool_size	指定用于存储内部数据结构信息的内存池的大小（用字节为单位）。2MB最常用，但如果你有很多表，要确保有足够的可分配的内存，否则MySQL将使用操作系统的内存（如果这种情况发生，并且增加了变量值，你就可以在错误日志中看到警告信息）
innodb_file_io_threads	InnoDB中文件I/O线程的数量。建议的值是4，但猜想Windows可能受益于设置较高的值
innodb_lock_wait_timeout	InnoDB事务在等待回滚之前等待锁定的时间（以秒为单位）。InnoDB在自己的锁定表中，自动检测死锁，但如果死锁来自外部（如LOCK TABLES语句），则可能引发死锁，在这种情况下，使用此变量的值
innodb_flush_method	一种刷新的方法。默认值是fdatasync，并且可选择的值是O_DSYNC。通常，值fdatasync会更快一些，虽然在一些Linux和UNIX的版本中，它被证明速度较慢
interactive_timeout	服务器在关闭连接之前，等待任何一种交互式连接（连接的时候是使用参数CLIENT_INTERACTIVE）活动的时间（以秒为单位）。参数wait-timeout应用于普通的连接。此变量的默认值是28 800
join_buffer_size	用于完全联接（当不使用索引的时候使用联接操作）的缓冲区的大小（以字节为单位）。缓冲区被分配给每一个完全联接使用。增加这个变量的值将使完全联接操作更快，尽管加快联接速度最好的方法是添加适当的索引
key_buffer_size	用于存放索引数据的缓冲区的大小（以字节为单位）。这个变量将在“优化键的缓冲区大小”一节中详细讨论
language	错误信息使用的语言文件的位置

（续表）

变量	描述
large_file_support	如果被MySQL编译时为支持大文件，则此变量值设为ON。默认值也是ON
locked_in_memory	如果MySQL被锁定在内存中（换句话说，如果mysqld启动的时候使用了--memlock参数），则此变量值设为ON。默认值是OFF。只有在操作系统出现问题且mysqld转储到磁盘时，此变量值才设置为ON
log	启用记录所有查询操作的日志时，变量值设置为ON
log_update	启用记录更新操作的日志时（这种情况应该使用二进制日志文件），变量值设置为ON
log_bin	启用记录二进制更新操作的日志时，变量值设置为ON
log_slave_updates	如果要对来自从属磁盘的更新操作进行日志记录，则此变量的值设为ON
long_query_time	用于规定慢查询操作的时间（以秒为单位）。查询花费的时间比这个值长时，将导致slow queries计数器的值被增加，并且在慢查询日志被启用的情况下，将会被记录到慢查询日志文件中
lower_case_table_names	如果表名是用小写字母存储的且大小写不敏感，则此变量的值设为1。默认值为0
max_allowed_packet	一个数据包被允许的最大字节数。消息的缓冲区的初始值大小由net_buffer_length指定，但它可以增长到这个大小。如果使用large BLOB或TEXT列，则将这个变量的值设置为列的最大值
max_binlog_cache_size	多语句事务可以使用的最大内存空间（以字节为单位），不会抛出错误信息：多语言事务需要的存储空间应大于max_binlog_cache_size
max_binlog_size	只要当前的二进制日志文件超过这个变量值，日志就会发生滚动，并且生成一个新的日志记录
max_connections	允许的最大连接的数量。参见本章前面讨论过的“如何处理连接太多的情况”一节
max_connect_errors	在主机被从远程连接封锁以前，主机可以尝试连接的最大次数。强加这个限制的目的是为了降低遭到拒绝服务攻击的可能性。通过运行FLUSH HOSTS，主机不会被封锁
max_delayed_threads	可以处理INSERT DELAYED语句的线程的最大数量。一旦达到这个最大值，远程的INSERT语句将执行普通插入操作，并且不使用DELAYED属性
max_heap_table_size	HEAP表能够允许的最大字节数
max_join_size	MySQL确定的联接操作，如果返回的行数多于这个限制值，则会返回一个错误信息。这样做是为了阻止用户偶然地（或恶意地）运行大量的查询语句，从而返回数百万的行数并占用太多的资源

（续表）

变量	描述
max_sort_length	排序BLOB或TEXT字段的时候，最多可以使用的字节数（以字节为单位）。例如，如果这个变量被设置到1024，那么只有前边的1024个字符将在排序的时候被使用
max_user_connections	确定单一用户已经激活的连接的最大数量。默认值0的意思是对连接数量没有限制（除了max_connections以外）
max_tmp_tables	客户在同一时刻能够保持打开状态的临时表的最大数量（表正在写入的时候，不使用这个变量；请查看最近的文档）
max_write_lock_count	如果出现许多写操作的锁，那么MySQL将允许运行读操作的锁
myisam_bulk_insert_tree_size	当MySQL进行大量的插入操作时（例如，LOAD DATA INFILE...），它使用像树一样的缓存来加快进程的速度。这个变量的值是每一个线程能使用的最大缓存空间（以字节为单位）。默认值是8MB，设置为0表示禁用这个功能。只有在向非空的表中增加记录的时候，缓存才被使用
myisam_recover_options	可用的参数有DEFAULT、BACKUP、FORCE、QUICK或OFF。除了OFF，如果这个变量被设置成其他参数，当MySQL启动的时候，它将查看表是否因为崩溃或非正常关闭而被打上标记。如果有，MySQL将检查表并试图修复毁坏的表。如果使用了BACKUP参数，MySQL将在修复期间发生任何变更的时候，生成.MYD数据文件的备份文件（扩展名为.BAK）。参数FORCE强制进行修复，即使造成数据丢失。如果数据中没有被删除的块，那么参数QUICK不对行进行检查。所有错误都将在错误日志文件中进行说明，所以你可以去查看有什么错误发生。一起设置BACKUP和FORCE参数允许MySQL可以从任何问题中自动恢复（带有备份功能以防发生错误）
myisam_sort_buffer_size	排序或修复索引时，分配给缓冲区的空间（以字节为单位）
myisam_max_extra_sort_file_size	MySQL生成一个索引的时候，从用来生成带有快速索引的临时表的空间中减掉键的缓存的空间。如果差值大于这个变量值（用兆字节指定），MySQL将使用键缓存的方法
myisam_max_sort_file_size	MySQL在创建成修复索引时，生成的临时文件的最大值（以兆字节为单位）。如果这个值被超出，则MySQL使用较慢的键缓存方法来创建或修复索引
net_buffer_length	查询之间使用的通信缓冲区的大小（以字节为单位）。低耗内存的系统为了节省内存，将这个值设置为由客户发送的SQL语句的期望长度。如果语句超过这个长度，则这个变量的值被自动增大到max_allowed_packet的大小
net_read_timeout	在终止读操作之前，MySQL等待来自连接的数据的时间（以秒为单位）。如果没有数据，则使用net_write_timeout，并且slave_net_timeout被用于主/从连接

（续表）

变量	描述
net_retry_count	在终止之前，读操作在通信端口反复重试的次数
net_write_timeout	在终止之前，等待数据块被写入连接的时间秒数
open_files_limit	MySQL使用这个值来预留文件描述符。如果出现错误信息“Too many open files（打开的文件太多）”，可以增加这个值。通常这个值被设为0。在出现错误信息的情况下，MySQL将使用更大的值max_connections*5或max_connections + table_cache*2
pid_file	到pid（进程标识符）文件的路径
port	MySQL使用的用来监听TCP/IP连接的端口号
protocol_version	MySQL使用的协议的版本
record_buffer	MySQL为每一个执行连续扫描操作的线程分配的缓冲区的大小（以字节为单位）。如果你做了很多连续扫描，你会希望增加这个值（连续扫描是指按顺序，一个接一个地读取记录）
record_rnd_buffer	当用不连续的方式对行进行读取（例如，排序以后）时，分配给缓冲区的大小（以字节为单位）。这种方式读取行可以避免进行磁盘扫描。如果不设置这个值，这个变量将与record_buffer相同
query_buffer_size	分配给查询缓冲区的初始大小（以字节为单位）。这个空间应该对大多数查询操作是足够的，否则要增加值
query_cache_limit	对查询缓存空间的限制值（以字节为单位）。大于这个值的结果将不被缓存。此变量的默认值是1MB
query_cache_size	分配给查询缓存（存储来自以前查询的结果）的大小（以字节为单位）。值为0表示禁用缓存
query_cache_startup_type	此变量值可以是0、1或2。值为0（关闭）意味着MySQL不缓存或检索结果。值为1（打开）意味着MySQL缓存所有的结果，除非这些结果来自SQL_NO_CACHE。值为2（要求）意味着只缓存SQL_CACHE的查询结果
safe_show_database	只在MySQL 4最早的版本中使用。一旦设置，对数据库没有任何权限用户，将看不到执行SHOW DATABASES语句后列出来的数据库（变量删除了SHOW DATABASES的这个权限）
server_id	服务器的ID。对识别服务器的复制用途非常重要
skip_external_locking	如果变量设置为ON，则禁用系统锁定功能。对myisamchk有重要的作用（参见第10章用myisamchk分析表的相关信息）
skip_networking	将变量值设为ON，则MySQL只允许本地连接
skip_show_database	一旦设置，SHOW DATABASES语句不返回结果，除非客户程序有PROCESS权限（MySQL 4早期版本中介绍的SHOW DATABASES权限删除了这种要求）

（续表）

变量	描述
slave_net_timeout	在读取操作终止之前，MySQL等待来自主/从连接的数据的时间（以秒为单位）
slow_launch_time	在线程启动之前，增加的slow_launch_threads计数器的时间（以秒为单位）
socket	由服务器使用的UNIX套接字的路径
sort_buffer	分配给在排序时使用的缓冲区的大小（以字节为单位）。参见本章前面“优化sort_buffer变量”一节中的讨论
table_cache	为所有线程打开的表的数量。参见本章前面“优化table_cache”一节中的讨论
table_type	默认的表的类型（通常为MyISAM）
thread_cache_size	可以复用的保存在缓存中的线程的数量。如果有，新的线程从缓存中取得，当断开连接的时候如果有空间，客户的线程被放置在缓存中。如果有很多新的线程，为了提高性能可以增加这个变量值。有好的执行线程的系统，通常不会收益很多。通过比较Connections和Threads_created状态的变量，可以看到这个变量的作用
thread_concurrency	在Solaris系统中，MySQL使用这个变量来确定是否要调用函数thr_setconcurrency()，这个函数用来帮助线程系统了解在同一时刻运行的线程的数量
thread_stack	每个线程使用的堆栈的大小（以字节为单位）。基准测试程序crash-me的行为依赖于这个变量值
timezone	服务器的时间区
tmp_table_size	内存中用于临时表的最大空间。随着变量值的增加，它将自动成为磁盘上的MyISAM表。如果所执行的许多查询语句的结果将形成一张大的临时表（例如，复杂的GROUP BY从句）和消耗很多内存，则应该增加变量值
tmpdir	用来存储临时文件和表的目录
version	服务器的版本号
wait_timeout	在关闭之前，MySQL等待连接活动的时间（以秒为单位）。interactive_timeout应用于交互式的连接

了解所有的状态变量

状态变量在每次服务器重新启动的时候，重新被复位。这个变量允许管理员对服务器的行为进行监控并识别可能存在的瓶颈、问题和所做的性能改变。表13.3是一个内容全面的列表。

表13.3 MySQL状态值

变量值	说明
Aborted_clients	指出由于某种原因客户程序不能正常关闭连接而导致失败的连接的数量。如果客户不在退出之前调用mysql_close()函数，wait_timeout或interactive_timeout的限制已经被超出，或者是客户程序在传输的过程中被关闭，则这种情况会发生
Aborted_connects	指出试图连接到MySQL的失败的次数。这种情况在客户尝试用错误的密码进行连接时，没有权限进行连接时，为获得连接的数据包所花费的时间超过了connect_timeout限制的秒数，或数据包中没有包含正确的信息时，都会发生
Bytes_received	从所有客户处已经接收到的字节数
Bytes_sent	已经发送给所有客户的字节数
Com_[statement]	用于每一种语句的这些变量中的一种。变量值表示这条语句被执行过的次数
Connections	试图连接到MySQL服务器的次数
Created_tmp_disk_tables	执行语句时，磁盘上生成的隐含临时表的数量
Created_tmp_tables	执行语句时，内存中生成的隐含临时表的数量
Created_tmp_files	由mysqld生成的临时文件的数量
Delayed_insert_threads	当前正在使用的延迟插入句柄的线程数量
Delayed_writes	由INSERT DELAYED语句写入的记录的个数
Delayed_errors	当发生错误时，由INSERT DELAYED语句写入的记录的个数。绝大多数普通的错误是复制键
Flush_commands	被执行的FLUSH语句的个数
Handler_commit	内部COMMIT命令的个数
Handler_delete	从一个表中删除行的次数
Handler_read_first	一条索引中的第一个条目被读取的次数，通常是指完全索引扫描（例如，假定indexed_col被索引，语句SELECT indexed_col from tablename导致了一个完全索引扫描）
Handler_read_key	当读取一行数据时，使用索引的请求的个数。如果查询时使用了索引，就希望这个值快速增加
Handler_read_next	按照索引顺序读取下一行数据的请求的个数。如果使用了完全索引进行扫描，或者在一个不变的范围内查询一个索引，则这个值就会增加
Handler_read_prev	按照索引的顺序读取前面一行数据的请求的个数。这个变量值由SELECT fieldlist ORDER BY fields DESC类型的语句使用

（续表）

变量值	说明
Handler_read_rnd	在固定的位置读取一行数据的请求的个数。要求结果被保存起来的查询操作将增加这个计数器的值
Handler_read_rnd_next	读取数据文件中下一行数据的请求的个数。一般，这个值不能太高，因为这意味着查询操作不会使用索引，并且必须从数据文件中读取
Handler_rollback	内部ROLLBACK命令的数量
Handler_update	在表中更新一条记录的请求的数量
Handler_write	在表中插入一条记录的请求的数量
Key_blocks_used	用在键的缓存中的数据块的数量
Key_read_requests	引起从键的缓存读取键的数据块的请求的数量。Key_reads与Key_read_requests的比率不应该高于1:100（也就是，1:10很糟糕）
Key_reads	引起从磁盘读取键的数据块的物理读取操作的数量。Key_reads与Key_read_requests的比率不应该高于1:100（同样，1:10很糟糕）
Key_write_requests	引起键的数据块被写入缓存的请求的数量
Key_writes	向磁盘写入键的数据块的物理写操作的次数
Max_used_connections	在任意时刻，正在使用的连接的最大数量。参见本章前面讨论的“处理连接太多的情况”一节
Not_flushed_key_blocks	在键的缓存中，已经发生了改变但还没有被刷新到磁盘上的键的数据块的数量
Not_flushed_delayed_rows	当前在INSERT DELAY队列中，等待被写入的记录的个数
Open_tables	目前打开的表的数量。参见本章前面有关表的缓存的讨论“优化table_cach”
Open_files	当前打开的文件的数量
Open_streams	当前打开的流数据的数量。这些流数据主要用于日志记录
Opened_tables	已经被打开的表的数量。参见本章前面有关表的缓存的讨论“优化table_cache”
Qcache_queries_in_cache	缓存中查询的个数
Qcache_inserts	添加到缓存中的查询的个数
Qcache_hits	查询缓存被访问的次数
Qcache_not_cached	没有被缓存（由于太大，或因为QUERY_CACHE_TYPE）的查询的数量
Qcache_free_memory	仍然可用于查询缓存的内存的数量

（续表）

变量值	说明
Qcache_total_blocks	在查询缓存中数据块的总数
Qcache_free_blocks	在查询缓存中空闲内存块的数量
Questions	初始的查询操作的总数
Rpl_status	安全复制的状态（这个变量只在MySQL 4之后的版本中使用）
Select_full_join	已经被执行的没有使用索引的联接的数量。不能将这个变量值设得太高
Select_full_range_join	在引用的表中使用范围搜索的被执行的联接的数量
Select_range	在第一个表中使用范围的被执行的联接的数量（通常这个值大一些比较好）
Select_range_check	在每一行数据后对索引进行检查的不带索引且已被执行的联接的数量
Select_scan	对第一个表实施完全扫描的已被执行的联接的数量
Slave_open_temp_tables	被从属服务器保留的当前打开的临时表的数量
Slave_running	值为ON或OFF。如果服务器是连接到主服务器的从属服务器，则值为ON
Slow_launch_threads	创建时间长于slow_launch_time的线程的数量
Slow_queries	时间长于long_query_time的查询的数量
Sort_merge_passes	在排序期间，已执行的合并的数量。如果这个变量值变得太大，就应该增加sort_buffer
Sort_range	在范围内执行的排序的数量
Sort_rows	已排序的行的数量
Sort_scan	通过扫描表执行的排序的数量
ssl_[variables]	由SSL使用的各种变量的内容。这个变量不在早期MySQL 4的版本中执行
Table_locks_immediate	立即获得的表的锁的次数
Table_locks_waited	不能立即获得的表的锁的次数。通常，变量值较高是性能出现问题的征兆。通过改善查询和索引操作、使用另一个表类型、拆分表或使用复制，来进行优化
Threads_cached	线程缓存中当前线程的数量
Threads_connected	当前打开的连接的数量
Threads_created	生成的用来处理连接的线程的数量
Threads_running	激活的（非睡眠状态）线程的数量
Uptime	服务器已经运行的时间（以秒为单位）

在服务器运行的同时，改变变量值

为了改变变量值经常要重新启动服务器，直到有了MySQL版本4.0.3。现在，可以使用更方便的SET语句来改变变量值而无需关闭服务器。

SET语句有两种使用方法。默认状态是所做的变化只影响SESSION，意思是当你下一次连接的时候（对于所有连接），变量将仍然使用配置文件指定的设置。如果指定GLOBAL关键字，所有新的连接将使用新的值。然而，当服务器重新启动的时候，仍然使用配置文件中指定的值，所以你也需要对配置文件中的值进行改变。为了用GLOBAL参数设置变量，需要有SUPER权限。

语法如下所示：

```
SET [GLOBAL | SESSION] sql_variable=expression, [[GLOBAL | SESSION]
  sql_variable=expression...]
```

例如，

```
mysql> SET SESSION max_sort_length=2048;
```

与

```
mysql> SET max_sort_length=2048
```

相同。

这还有一个可选的使用@@的语法，用来与其他数据库管理系统（DBMS）进行兼容，如下所示：

```
SET @@{global | local}.sql_variable=expression, [@@{global |
  local}.sql_variable=expression]
```

为了用这个语法中重复前面的例子，使用下面的

```
mysql> SET @@local.max_sort_length=2048;
```

SESSION和LOCAL是同义字。

实验完新的变量，如果你决定回去使用旧的值，不需要相信你的记忆或在配置文件中查找。可以使用DEFAULT关键字恢复GLOBAL值为配置文件中的值，或用SESSION值代替GLOBAL值。例如

```
mysql> SET SESSION max_sort_length=DEFAULT;
```

和

```
mysql> SET GLOBAL max_sort_length=DEFAULT;
```

表13.4显示的变量是用非标准的方法进行设置的。

表13.4 非标准变量

语法	描述
AUTOCOMMIT= 0 | 1	当值为1时，除非将语句包装在BEGIN和COMMIT之中，否则将自动COMMIT语句。当操作者设置了AUTOCOMMIT时，MySQL还自动COMMIT所有开放的事务
BIG_TABLES = 0 | 1	当值为1时，所有临时表被存储在磁盘上，而不是内存中。这将使临时表变得较慢，但是它可以防止内存用光的问题。默认值为0
INSERT_ID = #	设置AUTO_INCREMENT值（这样下一个使用AUTO_INCREMENT字段的INSERT语句将使用这个值）
LAST_INSERT_ID = #	设置从下一个LAST_INSERT_ID()功能中返回的值
LOW_PRIORITY_UPDATES = 0 | 1	当值为1时，所有的更新语句（INSERT、UPDATE、DELETE、LOCK TABLE WRITE）进行等待，以便在它们正在访问的表上完成所有进行中的读操作（SELECT、LOCK TABLE READ）
MAX_JOIN_SIZE = value | DEFAULT	通过设置行的最大空间，操作者可以防止MySQL执行一些查询，这些查询可能不能正确地使用索引，或者在服务器运行大批量任务或处于峰值时间时可能使服务器变慢。除了DEFAULT复位SQL_BIG_SELECTS，可以设置为任意值。如果设置了SQL_BIG_SELECTS，那么MAX_JOIN_SIZE会被忽略。如果查询已经进行了缓存，那么MySQL将忽略这些限制，并返回结果
QUERY_CACHE_TYPE = OFF | ON | DEMAND	为线程设置查询缓存
QUERY_CACHE_TYPE = 0 | 1 | 2	为线程设置查询缓存
SQL_AUTO_IS_NULL = 0 | 1	如果设置（默认值为1），那么AUTO_INCREMENT最后插入的行可以通过WHERE auto_increment_column IS NULL找到。MS Access和其他ODBC程序使用这个变量
SQL_BIG_SELECTS = 0 | 1	如果设置（默认值为1），那么MySQL准许大型的查询。如果不设（0），那么MySQL将不准许查询，但是它将必须检查比max_join_size更多的行。这对于避免那些可能导致服务器关闭的意外的或恶意的查询是有用的
SQL_BUFFER_RESULT = 0 | 1	如果设置（1），那么MySQL将查询结果放入临时的表中（在一些情况下，较早释放表的锁可以提高性能）
SQL_LOG_OFF = 0 | 1	如果设置（1），那么MySQL将不会为客户端进行日志记录（这不是更新日志）。需要SUPER许可权
SQL_LOG_UPDATE = 0 | 1	如果设置（0），那么MySQL将不会为客户端使用更新日志。需要SUPER许可权

（续表）

语法	描述
SQL_QUOTE_SHOW_CREATE = 0 | 1	如果设置（默认值为1），那么MySQL将引用表和列的名字
SQL_SAFE_UPDATES = 0 | 1	如果设置（1），那么MySQL将不会执行UPDATE或DELETE语句，它们不使用索引或LIMIT子句，这可以帮助防止不愉快的事情发生
SQL_SELECT_LIMIT = value | DEFAULT	设置最大的可能由SELECT语句返回的记录数（默认无限制）。LIMIT优先于此设置
TIMESTAMP = timestamp_value | DEFAULT	设置客户端的时间。在使用更新日志完成行的恢复时，操作者可以使用这个变量获得原始时间戳。timestamp_value是UNIX epoch时间戳

改进硬件以加速服务器的运行

对于性能很差的数据库服务器，硬件是一种最简单的改进手段（如果有足够的资金的话）。作为首选，首先配备你能有的所有内存，然后使用所能得到的最快的磁盘，最后使用最快的CPU（中央处理单元）。最好是对系统进行基准测试，使用可用的各种工具对操作系统进行测试，查看是否CPU、内存、磁盘速度或者它们的混合成为系统的瓶颈。这将为操作者提供最好的使用建议并协助操作者进行升级。运行基准测试软件包（本章稍后的“使用基准测试”部分）将帮助显示不同类型任务的性能。

内存

由于内存准许你调整myqld变量，因此它是最重要的组件。大量的内存意味着操作者可以建立大型的键和表的缓存。尽可能大的内存准许MySQL使用较快的内存而不是尽可能多的磁盘，并且内存更快意味着MySQL可以更快地访问存储在那里的数据。如果操作者不主动调试mysqld变量使用外部内存，那么大量的内存并不会很有用。

磁盘

最后，MySQL必须从磁盘获取数据，并且这也是快速磁盘访问起作用的地方。由于磁盘的巡道时间决定了物理磁盘能够移动以获得所需数据的快速程度，因此操作者应该选择具有最快巡道时间的磁盘。同样，SCSI（小型计算机系统接口）磁盘通常比IDE（智能或集成磁盘设备）磁盘更快，因此操作者可能会选择SCSI磁盘。

操作者可以进行的重要改善是在多个磁盘上进行条带化（操作系统将数据分开，均匀地分布在多个磁盘上），以及symlinking（建立从数据目录到其他磁盘的链接）。InnoDB表提供一种算法，它可以很容易地在多个磁盘之间拆分数据，但是MyISAM表不能这样做（由单一文件组成），因此条带化或其他的RAID形式可能会很有用。第16章“多台驱动器”提供了更多有关这方面的内容。

CPU

处理器越快，计算完成得也就越快，向客户端返回结果也就越快。除了处理器的速度，总线速度和缓存的大小也很重要。对可用处理器的分析超出了本书的范围，并且可能在本书出版前就已经过时了，但是要保证尽可能仔细地对处理器进行研究分析，查看在不同的基准测试中它是如何运行的。

使用基准测试程序

MySQL产品带有一个称为run-all-tests的基准测试包。操作者可以使用它测试多种DBMS，查看它们的运行情况。为了使用这个测试包，操作者需要具备Perl程序、Perl DBI模块和为准备测试的DBMS准备的DBD模块。表13.5对run-all-tests的参数进行了描述。

表13.5 run-all-tests的参数

参数	描述
--comments	向基准测试输出信息中添加注释
--cmp=server[,server...]	从指定的服务器根据限定运行测试。以相同的--cmp参数运行所有服务器，测试结果在两个不同的SQL服务器之间是可比的
--create-options=#	向所有的create语句指定额外的参数。例如，为了建立所有像BDB表的表，需要使用参数--create-options=TYPE=BDB
--database	指定建立测试表的数据库。默认是test数据库
--debug	显示调试信息。正常情况下，操作者只在对测试进行调试时使用它
--dir	指出测试结果应该存储的地方。默认是默认的输出位置
--fast	准许使用非标准的ANSI SQL命令加快测试
--fast-insert	可能的情况下使用快速插入，这包括多个值列表，如INSERT INTO tablename VALUES (values), (values)或INSERT INTO tablename VALUES (values)，而不是INSERT INTO tablename(fields) VALUES (values)
--field-count	指定在测试表中有多少个字段。通常只在对测试进行调试时使用
--force	即使遇到问题仍继续测试。在建立新的表之前，删除这些表。通常只在对测试进行调试时使用
--groups	指出在测试中将有多少不同的组。通常只在对测试进行调试时使用
--lock-tables	准许使用表锁闭以获得更高的速度
--log	将结果存在--dir目录下
--loop-count (Default)	指出每次测试循环要被执行多少次。通常只在对测试进行调试时使用
--help	显示参数表

（续表）

参数	描述
--host='host name'	指定数据库服务器所在的主机。默认是localhost
--machine="machine or os_name"	添加到基准测试输出文件名的机器名或操作系统名。默认是操作系统名+版本
--odbc	使用DBI ODBC驱动程序连接到数据库
--password='password'	指定测试连接使用的用户密码
--socket='socket'	指定连接使用的套接字（如果支持套接字的话）
--regions	制定应该如何对AND级别进行测试。通常只在对测试进行调试时使用
--old-headers	从旧的RUN文件中获得旧的基准测试头
--server='server name'	指定在哪里对哪个DBMS执行测试。这些可能包括Access、Adabas、AdabasD、Empress、Oracle、Informix、DB2、mSQL、MS-SQL、MySQL、Pg、Solid和Sybase。默认是MySQL
--silent	在测试开始时不输出服务器的信息
--skip-delete	指定所建立的测试表不被删除。通常只在对测试进行调试时使用
--skip-test=test1[,test2,...]	在运行基准测试时，不包括制定的测试
--small-test	利用更小的限定来加速测试
--small-tables	使用更少的行进行测试。如果数据库出于某些原因不能控制大的表，那么应该使用它（它们可能有小的部分）
--suffix	向基准测试输出文件名中的数据库名字添加后缀。用于希望运行多个测试又不覆盖这些测试结果的情况中。在使用--fast参数时，后缀自动生成为_fast
--random	为测试执行的顺序生成临时的初始值，这可以用来模拟真实的环境
--threads=#	为多用户基准测试指定所使用的线程数。默认值是5
--tcpip	使用TCP/IP连接到服务器。准许在TCP/IP堆栈可能被填满时，在一行中测试程序生成很多新的连接
--time-limit	指定在测试结束前、结果模拟出来前，测试循环限定的时间，以秒为单位。默认是600秒
--use-old-results	使用--dir目录中旧的结果，而不是实际执行测试
--user='user_name'	指定连接所用的用户
--verbose	显示更多的信息。通常只在对测试进行调试时使用
--optimization='some comments'	添加有关已在测试之前完成的优化的注释
--hw='some comments'	为测试添加有关所用硬件的注释

为了运行run-all-tests，需要从基准目录转到sql-bench目录。下面是基准测试程序的输出例子：

```
% cd sql-bench
% perl run-all-tests --small-test --password='g00r002b'
Benchmark DBD suite: 2.14
Date of test:        2002-07-21 21:35:42
Running tests on:    Linux 2.2.5-15 i686
Arguments:           --small-test
Comments:
Limits from:
Server version:      MySQL 4.0.1 alpha max log
Optimization:        None
Hardware:

ATIS: Total time: 19 wallclock secs ( 5.23 usr  0.96 sys +  0.00 cusr
  0.00 csys =  0.00 CPU)
alter-table: Total time:  2 wallclock secs ( 0.12 usr  0.03 sys +  0.00
 cusr  0.00 csys =  0.00 CPU)
big-tables: Total time:  1 wallclock secs ( 0.43 usr  0.10 sys +  0.00
 cusr  0.00 csys =  0.00 CPU)
connect: Total time:  8 wallclock secs ( 2.90 usr  0.66 sys +  0.00
 cusr  0.00 csys =  0.00 CPU)
create: Total time:  0 wallclock secs ( 0.15 usr  0.01 sys +  0.00
 cusr  0.00 csys =  0.00 CPU)
insert: Total time: 31 wallclock secs ( 8.47 usr  1.43 sys +  0.00
 cusr  0.00 csys =  0.00 CPU)
select: Total time: 55 wallclock secs (17.76 usr  1.71 sys +  0.00
 cusr  0.00 csys =  0.00 CPU)
transactions: Test skipped because the database doesn't support
 transactions
wisconsin: Total time: 42 wallclock secs ( 9.55 usr  1.84 sys +  0.00
 cusr  0.00 csys =  0.00 CPU)

All 9 test executed successfully

Totals per operation:
Operation                    seconds     usr     sys     cpu   tests
alter_table_add                 1.00    0.07    0.00    0.00      92
alter_table_drop                0.00    0.03    0.00    0.00      46
connect                         0.00    0.22    0.02    0.00     100
connect+select_1_row            1.00    0.27    0.04    0.00     100
connect+select_simple           1.00    0.27    0.04    0.00     100
count                           1.00    0.13    0.00    0.00     100
count_distinct                  1.00    0.13    0.02    0.00     100
count_distinct_2                1.00    0.16    0.02    0.00     100
count_distinct_big              1.00    0.12    0.04    0.00      30
```

count_distinct_group	1.00	0.17	0.00	0.00	100
count_distinct_group_on_key	1.00	0.13	0.01	0.00	100
count_distinct_group_on_key_parts	1.00	0.16	0.01	0.00	100
count_distinct_key_prefix	1.00	0.12	0.01	0.00	100
count_group_on_key_parts	1.00	0.09	0.00	0.00	100
count_on_key	20.00	6.11	0.60	0.00	5100
create+drop	0.00	0.01	0.00	0.00	10
create_MANY_tables	0.00	0.01	0.00	0.00	10
create_index	1.00	0.00	0.00	0.00	8
create_key+drop	0.00	0.12	0.01	0.00	100
create_table	1.00	0.03	0.00	0.00	31
delete_all_many_keys	1.00	0.08	0.00	0.00	1
delete_big	0.00	0.01	0.00	0.00	1
delete_big_many_keys	1.00	0.07	0.00	0.00	128
delete_key	0.00	0.06	0.01	0.00	100
delete_range	1.00	0.01	0.00	0.00	12
drop_index	0.00	0.00	0.00	0.00	8
drop_table	0.00	0.01	0.00	0.00	28
drop_table_when_MANY_tables	0.00	0.00	0.00	0.00	10
insert	57.00	13.90	2.51	0.00	44768
insert_duplicates	1.00	0.29	0.04	0.00	1000
insert_key	0.00	0.04	0.01	0.00	100
insert_many_fields	0.00	0.12	0.00	0.00	200
insert_select_1_key	0.00	0.00	0.00	0.00	1
insert_select_2_keys	0.00	0.00	0.00	0.00	1
min_max	1.00	0.06	0.01	0.00	60
min_max_on_key	17.00	8.12	0.64	0.00	7300
multiple_value_insert	0.00	0.03	0.00	0.00	1000
order_by_big	1.00	0.30	0.10	0.00	10
order_by_big_key	1.00	0.29	0.14	0.00	10
order_by_big_key2	1.00	0.28	0.12	0.00	10
order_by_big_key_desc	0.00	0.35	0.08	0.00	10
order_by_big_key_diff	0.00	0.35	0.05	0.00	10
order_by_big_key_prefix	1.00	0.31	0.09	0.00	10
order_by_key2_diff	0.00	0.01	0.00	0.00	10
order_by_key_prefix	0.00	0.02	0.00	0.00	10
order_by_range	0.00	0.03	0.00	0.00	10
outer_join	1.00	0.01	0.00	0.00	10
outer_join_found	1.00	0.01	0.01	0.00	10
outer_join_not_found	1.00	0.03	0.01	0.00	10
outer_join_on_key	0.00	0.01	0.00	0.00	10
select_1_row	1.00	0.27	0.06	0.00	1000
select_1_row_cache	1.00	0.18	0.07	0.00	1000
select_2_rows	1.00	0.43	0.05	0.00	1000
select_big	0.00	0.31	0.10	0.00	17

select_big_str	1.00	0.55	0.22	0.00	100
select_cache	4.00	0.95	0.18	0.00	1000
select_cache2	4.00	1.28	0.11	0.00	1000
select_column+column	1.00	0.35	0.06	0.00	1000
select_diff_key	0.00	0.02	0.00	0.00	10
select_distinct	1.00	0.30	0.06	0.00	80
select_group	4.00	0.61	0.09	0.00	391
select_group_when_MANY_tables	0.00	0.00	0.00	0.00	10
select_join	1.00	0.06	0.03	0.00	10
select_key	0.00	0.02	0.01	0.00	20
select_key2	0.00	0.02	0.00	0.00	20
select_key2_return_key	1.00	0.12	0.00	0.00	20
select_key2_return_prim	0.00	0.00	0.00	0.00	20
select_key_prefix	0.00	0.05	0.00	0.00	20
select_key_prefix_join	2.00	0.62	0.16	0.00	10
select_key_return_key	0.00	0.02	0.00	0.00	20
select_many_fields	1.00	0.31	0.10	0.00	200
select_range	2.00	0.23	0.05	0.00	41
select_range_key2	1.00	0.43	0.02	0.00	505
select_range_prefix	1.00	0.42	0.05	0.00	505
select_simple	0.00	0.21	0.04	0.00	1000
select_simple_cache	0.00	0.14	0.05	0.00	1000
select_simple_join	0.00	0.13	0.05	0.00	50
update_big	1.00	0.01	0.00	0.00	10
update_of_key	1.00	0.20	0.03	0.00	500
update_of_key_big	0.00	0.02	0.01	0.00	13
update_of_primary_key_many_keys	0.00	0.12	0.02	0.00	256
update_with_key	4.00	1.03	0.12	0.00	3000
update_with_key_prefix	1.00	0.58	0.01	0.00	1000
wisc_benchmark	2.00	0.70	0.16	0.00	34
TOTALS	156.00	43.84	6.55	0.00	76237

基准测试程序包对于不同平台的比较是很有用的。MySQL带有一套结果，但是这些结果比较陈旧，不是特别的有用。为了使这些测试结果对实际环境有意义，建议自己重复进行测试。

在大量应用自己的应用程序之前，对它们进行基准测试也很重要（在可能最重的负载情况下）。能够帮助操作者在服务器上强加大量负载的应用程序是super-smack，在MySQL站点上是可下载的。

MySQL带有的另一个有用的脚本是crash-me，它会对具体的安装进行功能性校验，并对重压下的服务器进行可靠性测试（见表13.6）。在对安装的测试失败时，会从结果中获得其名称。它也很轻便，并且可以比较测试多个数据库平台。根据其操作的结果，它不应该在实时的环境中运行。它不仅会使数据库服务器崩溃，同时它还会占用大量的内存。也就是说它会影响服务器上运行的其他程序。要知道虽然MySQL使用它增强了其实力，并且在对比时弱化了MySQL的弱点。但是像MySQL当前没有使用的triggers和procedures，从查看crash-me

输出内容的角度可能看起来与一些非标准的MySQL特性（例如为OR使用||，而不是串连接）一样重要。

表13.6 CRASH-ME参数

参数	描述
--help, --Information	显示参数的帮助信息
--batch-mode	执行测试，不提示输入并且在遇到错误时退出
--comment='some comment'	向crash-me限定文件中加入指定的注释
--check-server	如果服务器一直在运行，那么每次它检查时便与服务器形成一个新的连接。如果之前的查询返回错误的数据，那么这个参数便很有用
--database='database'	在数据库中建立测试表。默认是test
--dir='directory_name'	将输出内容存放在这个目录中
--debug	如果有问题，那么显示大量输出内容以辅助调试（debug）
--fix-limit-file	重新格式化crash-me限定文件。这并不重新运行crash-me
--force	立即开始测试，不产生告警，不等待输入。可以利用这个参数进行自动测试
--log-all-queries	显示所有可执行的查询。主要用于调试crash-me
--log-queries-to-file='filename'	将所有的查询日志记录到指定的文件
--host='hostname'	在指定的主机上运行测试。默认是localhost
--password='password'	指定当前用户的密码
--restart	在每次测试中保存状态信息，在出现系统崩溃时，使得以相同的参数重新启动并从停止处继续成为可能
--server='server name'	指定运行测试的服务器。包括Access、Adabas、AdabasD、Empress、Oracle、Informix、DB2、Mimer、mSQL、MS-SQL、MySQL、Pg、Solid或Sybase。默认是MySQL。其他的服务器将不会被通告它们的名字
--user='user_name'	指定连接时使用的用户名
--start-cmd='command to restart server'	在它要死机时，将使用指定的命令重新启动数据库服务器（参数的有效性已经说明了一切！）
--sleep='time in seconds'	指定重新启动服务器之前所要等待的时间，此时间以秒为单位。默认是10秒钟

下面是crash-me的例子：

```
% perl crash-me --password='g00r002b'
Running crash-me 1.57 on 'MySQL 4.0.1 alpha max log'
```

```
I hope you didn't have anything important running on this server....
Reading old values from cache: /usr/local/mysql-max-4.0.1-alpha-pc-
linux-gnu-i686/sql-bench/limits/mysql.cfg

NOTE: You should be familiar with 'crash-me --help' before continuing!

This test should not crash MySQL if it was distributed together with
 the running MySQL version.
If this is the case you can probably continue without having to worry
 about destroying something.

Some of the tests you are about to execute may require a lot of
memory.  Your tests WILL adversely affect system performance. It's
not uncommon that either this crash-me test program, or the actual
database back-end, will DIE with an out-of-memory error. So might
any other program on your system if it requests more memory at the
wrong time.

Note also that while crash-me tries to find limits for the database server
it will make a lot of queries that can't be categorized as 'normal'.
It's not unlikely that crash-me finds some limit bug in your server so
if you run this test you have to be prepared that your server may die
during it!

We, the creators of this utility, are not responsible in any way if
your database server unexpectedly crashes while this program tries to
find the limitations of your server. By accepting the following
question with 'yes', you agree to the above!

You have been warned!

Start test (yes/no) ?
Tables without primary key: yes
SELECT without FROM: yes
Select constants: yes
Select table_name.*: yes
Allows ' and " as string markers: yes
Double '' as ' in strings: yes
Multiple line strings: yes
" as identifier quote (ANSI SQL): error
` as identifier quote: yes
[] as identifier quote: no
Column alias: yes
Table alias: yes
Functions: yes
Group functions: yes
Group functions with distinct: yes
Group by: yes
Group by position: yes
```

```
Group by alias: yes
Group on unused column: yes
Order by: yes
Order by position: yes
Order by function: yes
Order by on unused column: yes
Order by DESC is remembered: no
Compute: no
INSERT with Value lists: yes
INSERT with set syntax: yes
allows end ';': yes
LIMIT number of rows: with LIMIT
SELECT with LIMIT #,#: yes
Alter table add column: yes
Alter table add many columns: yes
Alter table change column: yes
Alter table modify column: yes
Alter table alter column default: yes
Alter table drop column: yes
Alter table rename table: yes
rename table: yes
truncate: yes
Alter table add constraint: yes
Alter table drop constraint: no
Alter table add unique: yes
Alter table drop unique: with drop key
Alter table add primary key: with constraint
Alter table add foreign key: yes
Alter table drop foreign key: with drop foreign key
Alter table drop primary key: drop primary key
Case insensitive compare: yes
Ignore end space in compare: yes
Group on column with null values: yes
Having: yes
Having with group function: yes
Order by alias: yes
Having on alias: yes
binary numbers (0b1001): no
hex numbers (0x41): yes
binary strings (b'0110'): no
hex strings (x'lace'): no
Value of logical operation (1=1): 1
Simultaneous connections (installation default): 101
query size: 1048574

Supported sql types
```

```
Type character(1 arg): yes
Type char(1 arg): yes
Type char varying(1 arg): yes
Type character varying(1 arg): yes
Type boolean: no
Type varchar(1 arg): yes
Type integer: yes
Type int: yes
Type smallint: yes
Type numeric(2 arg): yes
Type decimal(2 arg): yes
Type dec(2 arg): yes
Type bit: yes
Type bit(1 arg): yes
Type bit varying(1 arg): no
Type float: yes
Type float(1 arg): yes
Type real: yes
Type double precision: yes
Type date: yes
Type time: yes
Type timestamp: yes
Type interval year: no
Type interval year to month: no
Type interval month: no
Type interval day: no
Type interval day to hour: no
Type interval day to minute: no
Type interval day to second: no
Type interval hour: no
Type interval hour to minute: no
Type interval hour to second: no
Type interval minute: no
Type interval minute to second: no
Type interval second: no
Type national character varying(1 arg): yes
Type national character(1 arg): yes
Type nchar(1 arg): yes
Type national char varying(1 arg): yes
Type nchar varying(1 arg): yes
Type national character varying(1 arg): yes
Type timestamp with time zone: no

Supported odbc types
Type binary(1 arg): yes
Type varbinary(1 arg): yes
```

```
Type tinyint: yes
Type bigint: yes
Type datetime: yes

Supported extra types
Type blob: yes
Type byte: no
Type long varbinary: yes
Type image: no
Type text: yes
Type text(1 arg): no
Type mediumtext: yes
Type long varchar(1 arg): no
Type varchar2(1 arg): no
Type mediumint: yes
Type middleint: yes
Type int unsigned: yes
Type int1: yes
Type int2: yes
Type int3: yes
Type int4: yes
Type int8: yes
Type uint: no
Type money: no
Type smallmoney: no
Type float4: yes
Type float8: yes
Type smallfloat: no
Type float(2 arg): yes
Type double: yes
Type enum(1 arg): yes
Type set(1 arg): yes
Type int(1 arg) zerofill: yes
Type serial: no
Type char(1 arg) binary: yes
Type int not null auto_increment: yes
Type abstime: no
Type year: yes
Type datetime: yes
Type smalldatetime: no
Type timespan: no
Type reltime: no
Type int not null identity: no
Type box: no
Type bool: yes
Type circle: no
```

```
Type polygon: no
Type point: no
Type line: no
Type lseg: no
Type path: no
Type interval: no
Type serial: no
Type inet: no
Type cidr: no
Type macaddr: no
Type varchar2(1 arg): no
Type nvarchar2(1 arg): no
Type number(2 arg): no
Type number(1 arg): no
Type number: no
Type long: no
Type raw(1 arg): no
Type long raw: no
Type rowid: no
Type mlslabel: no
Type clob: no
Type nclob: no
Type bfile: no
Remembers end space in char(): no
Remembers end space in varchar(): no
Supports 0000-00-00 dates: yes
Supports 0001-01-01 dates: yes
Supports 9999-12-31 dates: yes
Supports 'infinity dates: error
Type for row id: auto_increment
Automatic row id: _rowid

Supported sql functions

Supported odbc functions

Supported extra functions

Supported where functions

Supported sql group functions
Group function AVG: yes
Group function COUNT (*): yes
Group function COUNT column name: yes
Group function COUNT(DISTINCT expr): yes
Group function MAX on numbers: yes
Group function MAX on strings: yes
Group function MIN on numbers: yes
```

```
Group function MIN on strings: yes
Group function SUM: yes
Group function ANY: no
Group function EVERY: no
Group function SOME: no

Supported extra group functions
Group function BIT_AND: yes
Group function BIT_OR: yes
Group function COUNT(DISTINCT expr,expr,...): yes
Group function STD: yes
Group function STDDEV: yes
Group function VARIANCE: no

mixing of integer and float in expression: yes
No need to cast from integer to float: yes
Is 1+NULL = NULL: yes
Is concat('a',NULL) = NULL: yes
LIKE on numbers: yes
column LIKE column: yes
update of column= -column: yes
String functions on date columns: yes
char are space filled: no
DELETE FROM table1,table2...: no
Update with sub select: no
Calculate 1--1: yes
ANSI SQL simple joins: yes
max text or blob size: 1048543 (cache)
constant string size in where: 1048539 (cache)
constant string size in SELECT: 1048565 (cache)
return string size from function: 1047552 (cache)
simple expressions: 1837 (cache)
big expressions: 10 (cache)
stacked expressions: 1837 (cache)
tables in join: 63 (cache)
primary key in create table: yes
unique in create table: yes
unique null in create: yes
default value for column: yes
default value function for column: no
temporary tables: yes
create table from select: yes
index in create table: yes
null in index: yes
null in unique index: yes
null combination in unique index: yes
```

```
null in unique index: yes
index on column part (extension): yes
different namespace for index: yes
case independent table names: no
drop table if exists: yes
create table if not exists: yes
inner join: yes
left outer join: yes
natural left outer join: yes
left outer join using: yes
left outer join odbc style: yes
right outer join: yes
full outer join: no
cross join (same as from a,b): yes
natural join: yes
union: no
union all: no
intersect: no
intersect all: no
except: no
except all: no
except: no
except all: no
minus: no
natural join (incompatible lists): yes
union (incompatible lists): no
union all (incompatible lists): no
intersect (incompatible lists): no
intersect all (incompatible lists): no
except (incompatible lists): no
except all (incompatible lists): no
except (incompatible lists): no
except all (incompatible lists): no
minus (incompatible lists): no
subqueries: no
insert INTO ... SELECT ...: yes
atomic updates: no
views: no
foreign key syntax: yes
foreign keys: no
Create SCHEMA: no
Column constraints: no
Table constraints: no
Named constraints: no
NULL constraint (SyBase style): yes
```

```
Triggers (ANSI SQL): no
PSM procedures (ANSI SQL): no
PSM modules (ANSI SQL): no
PSM functions (ANSI SQL): no
Domains (ANSI SQL): no
many tables to drop table: yes
drop table with cascade/restrict: yes
-- as comment (ANSI): yes
// as comment: no
# as comment: yes
/* */ as comment: yes
insert empty string: yes
Having with alias: yes
table name length: 64 (cache)
column name length: 64 (cache)
select alias name length: +512 (cache)
table alias name length: +512 (cache)
index name length: 64 (cache)
max char() size: 255 (cache)
max varchar() size: 255 (cache)
max text or blob size: 1048543 (cache)
Columns in table: 3398 (cache)
unique indexes: 32 (cache)
index parts: 16 (cache)
max index part length: 255 (cache)
index varchar part length: 255 (cache)
indexes: 32
index length: 500 (cache)
max table row length (without blobs): 65534 (cache)
table row length with nulls (without blobs): 65502 (cache)
number of columns in order by: +64 (cache)
number of columns in group by: +64 (cache)
crash-me safe: yes
reconnected 0 times
```

在ANSI模式下运行MySQL

在ANSI模式下运行MySQL可使它以一种比通常更标准的方式运行，并且使它在今后能够更容易地移至其他的数据库中。如果以--ansi参数启动MySQL，那么MySQL操作起来会有下面的一些不同：

- 符号||并不意味着OR，而是串连接。即参数PIPES_AS_CONCAT mysqld sql-mode。
- 在功能的名字前使用空格才不再导致错误的出现。结果所有功能的名字都成为了保留字。即参数IGNORE_SPACE mysqld sql-mode。

- REAL是FLOAT的同义词，不是DOUBLE的同义词。即参数REAL_AS_FLOAT mysqld sql-mode。
- 默认的事务隔离级别被设为SERIALIZABLE。即参数SERIALIZE mysqld sql-mode。
- 双引号（“）字符将是引用字符的标识符，而不是串引用字符。即参数ANSI_QUOTES mysqld sql-mode。

在MySQL中使用不同的语言

世界上许多使用MySQL的人并不以英语作为他们的母语，在这种情况下，操作者输入MySQL的数据可以是他所希望的任何语言。MySQL AB是目前负责MySQL的公司，公司设于瑞典，并且大部分主要的开发人员都在斯堪的纳维亚。因此MySQL产品支持其他的语言也就不足为怪了。下面是一些当前支持的语言，很可能还会有更多的语言加进来：Czech（捷克语）、Danish（丹麦语）、Dutch（荷兰语）、Englishi（英语，默认语言）、Estonian（爱沙尼亚语）、French（法语）、German（德语）、Greek（希腊语）、Hungarian（匈牙利语）、Italian（意大利语）、Korean（韩语）、Norwegian（挪威语）、Norwegian-ny、Polish（波兰语）、Portuguese（葡萄牙语）、Romanian（罗马尼亚语）、Russian（俄语）、Slovak（斯洛伐克）、Spanish（西班牙语）和Swedish（瑞典语）。

用其他的语言显示出错信息

可以很容易地在启动时使用--language或-L参数，令MySQL开始使用这些语言中的一种来显示出错信息。为了在配置文件中做到这些，需要像下面这样简单地增加一行：

```
language=french
```

操作者还可以自己编辑出错信息（可能操作者希望数据库具有自己的风格），或者向其他语言贡献自己的设置，并且向MySQL组织做出一些反馈。为了对这些出错信息进行更改，需要简单地在适当的语言目录中编辑errmsg.txt文件（通常是MySQL基准目录中的/share/language_name），运行cmp_error工具，并且重新启动服务器。举例来说：

```
% cp errmsg.txt errmsg.bak
% vi errmsg.txt
```

这里我对出错信息进行了编辑：

```
"No Database Selected",
```

改为：

```
"Haven't you forgotten something - No Database Selected",
```

然后存储它：

```
"errmsg.txt" 229 lines, 12060 characters written
% comp_err errmsg.txt errmsg.sys
Found 226 messages in language file errmsg.sys
```

随后重新启动服务器，新的错误信息将出现：

```
% mysqladmin shutdown
% /etc/rc.d/init.d/mysql start
% mysql -uroot -pg00r002b
mysql> SELECT * FROM a;
ERROR 1046: Haven't you forgotten something - No Database Selected
```

如果你更新到一个新版本的MySQL，那么你将不得不重复这些修改。

使用一个不同的字符集

MySQL默认使用Latin1字符集（ISO-8859-1）。字符集决定什么字符可以使用，以及为查询进行分类。操作者可以通过改变--default-character-set参数的值在启动服务器时改变字符集。当前可用的字符集包括下列的这些：

latin1	dos	estonia
big5	german1	hungarian
czech	hp8	koi8_ukr
euc_kr	koi8_ru	win1251ukr
gb2312	latin2	greek
gbk	swe7	win1250
latin1_de	usa7	croat
sjis	cp1251	cp1257
tis620	danish	latin5
ujis	hebrew	
dec8	win1251	

你可以通过查看字符集变量的值来查看哪些字符集在你的版本中可用。

当你改变了字符集后，你需要重建索引来确保它们根据新字符集的规则进行分类。

MySQL默认时利用--with-extra-charsets=complex进行编译，这使得其他的字符集在必要时可用。如果你在自己编译MySQL，并且你知道永远不需要其他的字符集，那么你可以使用--with-extra-charsets=none参数。

添加自己的字符集

操作者还可以添加自己的字符集，如果是一个简单的字符集，并且不需要多字符支持，或排序程序进行排序，那么添加它是很容易的。如果需要这些额外的内容，那么将变得更复杂。为了添加一个字符集，需要执行下面的步骤：

1. 向sql/share/charsets/Index文件中添加新的字符集，并且赋予它惟一的ID。不同的版本，其路径可能不同，但是它始终是Index文件。这里，你可以称这个新的字符集为martian，ID为31：

```
# sql/share/charsets/Index
#
```

```
# This file lists all of the available character sets.

big5            1
czech           2
dec8            3
dos             4
german1         5
hp8             6
koi8_ru         7
latin1          8
latin2          9
swe7           10
usa7           11
ujis           12
sjis           13
cp1251         14
danish         15
hebrew         16
# The win1251 character set is deprecated.  Please use cp1251 instead.
win1251        17
tis620         18
euc_kr         19
estonia        20
hungarian      21
koi8_ukr       22
win1251ukr     23
gb2312         24
greek          25
win1250        26
croat          27
gbk            28
cp1257         29
latin5         30
martian        31
```

2. 创建.conf文件，把它放在这个目录中，例如sql/share/charsets/martian.conf。可以利用现有的.conf文件作为模板开始。

   在.conf文件中，以#开始的行为注解，字之间通过空格分隔，并且每个字必须是十六进制的格式。有4个序列。它们按顺序是ctype（包括257个码元）、to_lower和to_upper（每个包含256个码元）和sort_order（也包含256个码元）。下面是一个.conf文件的例子（这是标准的latin1.conf）：

```
# Configuration file for the latin1 character set

# ctype array (must have 257 elements)
  00
```

```
  20  20  20  20  20  20  20  20  20  28  28  28  28  28  20  20
  20  20  20  20  20  20  20  20  20  20  20  20  20  20  20  20
  48  10  10  10  10  10  10  10  10  10  10  10  10  10  10  10
  84  84  84  84  84  84  84  84  84  84  10  10  10  10  10  10
  10  81  81  81  81  81  81  01  01  01  01  01  01  01  01  01
  01  01  01  01  01  01  01  01  01  01  01  10  10  10  10  10
  10  82  82  82  82  82  82  02  02  02  02  02  02  02  02  02
  02  02  02  02  02  02  02  02  02  02  02  10  10  10  10  20
  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
  48  10  10  10  10  10  10  10  10  10  10  10  10  10  10  10
  10  10  10  10  10  10  10  10  10  10  10  10  10  10  10  10
  01  01  01  01  01  01  01  01  01  01  01  01  01  01  01  01
  01  01  01  01  01  01  01  10  01  01  01  01  01  01  01  02
  02  02  02  02  02  02  02  02  02  02  02  02  02  02  02  02
  02  02  02  02  02  02  02  10  02  02  02  02  02  02  02  02

# to_lower array (must have 256 elements)
  00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
  10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
  20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
  30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
  40  61  62  63  64  65  66  67  68  69  6A  6B  6C  6D  6E  6F
  70  71  72  73  74  75  76  77  78  79  7A  5B  5C  5D  5E  5F
  60  61  62  63  64  65  66  67  68  69  6A  6B  6C  6D  6E  6F
  70  71  72  73  74  75  76  77  78  79  7A  7B  7C  7D  7E  7F
  80  81  82  83  84  85  86  87  88  89  8A  8B  8C  8D  8E  8F
  90  91  92  93  94  95  96  97  98  99  9A  9B  9C  9D  9E  9F
  A0  A1  A2  A3  A4  A5  A6  A7  A8  A9  AA  AB  AC  AD  AE  AF
  B0  B1  B2  B3  B4  B5  B6  B7  B8  B9  BA  BB  BC  BD  BE  BF
  E0  E1  E2  E3  E4  E5  E6  E7  E8  E9  EA  EB  EC  ED  EE  EF
  F0  F1  F2  F3  F4  F5  F6  D7  F8  F9  FA  FB  FC  FD  FE  DF
  E0  E1  E2  E3  E4  E5  E6  E7  E8  E9  EA  EB  EC  ED  EE  EF
  F0  F1  F2  F3  F4  F5  F6  F7  F8  F9  FA  FB  FC  FD  FE  FF

# to_upper array (must have 256 elements)
  00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
  10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
  20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
  30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
  40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
  50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
  60  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
  50  51  52  53  54  55  56  57  58  59  5A  7B  7C  7D  7E  7F
  80  81  82  83  84  85  86  87  88  89  8A  8B  8C  8D  8E  8F
  90  91  92  93  94  95  96  97  98  99  9A  9B  9C  9D  9E  9F
```

```
  A0  A1  A2  A3  A4  A5  A6  A7  A8  A9  AA  AB  AC  AD  AE  AF
  B0  B1  B2  B3  B4  B5  B6  B7  B8  B9  BA  BB  BC  BD  BE  BF
  C0  C1  C2  C3  C4  C5  C6  C7  C8  C9  CA  CB  CC  CD  CE  CF
  D0  D1  D2  D3  D4  D5  D6  D7  D8  D9  DA  DB  DC  DD  DE  DF
  C0  C1  C2  C3  C4  C5  C6  C7  C8  C9  CA  CB  CC  CD  CE  CF
  D0  D1  D2  D3  D4  D5  D6  F7  D8  D9  DA  DB  DC  DD  DE  FF

# sort_order array (must have 256 elements)
  00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
  10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
  20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
  30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
  40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
  50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
  60  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
  50  51  52  53  54  55  56  57  58  59  5A  7B  7C  7D  7E  7F
  80  81  82  83  84  85  86  87  88  89  8A  8B  8C  8D  8E  8F
  90  91  92  93  94  95  96  97  98  99  9A  9B  9C  9D  9E  9F
  A0  A1  A2  A3  A4  A5  A6  A7  A8  A9  AA  AB  AC  AD  AE  AF
  B0  B1  B2  B3  B4  B5  B6  B7  B8  B9  BA  BB  BC  BD  BE  BF
  41  41  41  41  5C  5B  5C  43  45  45  45  45  49  49  49  49
  44  4E  4F  4F  4F  4F  5D  D7  D8  55  55  55  59  59  DE  DF
  41  41  41  41  5C  5B  5C  43  45  45  45  45  49  49  49  49
  44  4E  4F  4F  4F  4F  5D  F7  D8  55  55  55  59  59  DE  FF
```

ctype序列包含位值，一个字符一个码元。to_lower和to_upper序列仅仅控制着对应于字符集的每个成员的大小写字符。例如to_lower['A']包含a，而to_upper['z']包含Z。sort_order序列指出字符要进行排序（通常对应于to_upper，这时排序与大小写有关）。除ctype是通过字符值+1（一种老方法）进行索引之外，所有的序列都通过字符值进行索引。

3. 向configure.in文件中的CHARSETS_AVAILABLE和COMPILED_CHARSETS列表添加新的字符集（martian.conf）。
4. 重新配置并重新编译MySQL，并且测试新的字符集。

如果你有足够的勇气来增加一个新的复杂的字符集，那么这个处理过程需要再多几个步骤。对于所需要的内容可以在MySQL文档中看到（以及当前复杂的字符集文档：czech、gbk、sjis和tis160）。

小结

为了理解如何从数据库服务器中获得最多的内容，理解在精调服务器时所用的参数是很重要的。为了查看现有的服务器如何被安装，可以使用SHOW VARIABLES语句，以及来查看它如何进行SHOW STATUS处理。这两条语句的输出信息可以显示很多隐藏的问题，包括没有优化的查询，对可用内存的不当使用，或的确是时候进行升级了。

MySQL提供了四个配置文件，它们可以帮助服务器获得更好的性能。可以从my-huge.cnf、my-large.cnf、my-medium.cnf或my-small.cnf中为服务器选择与其条件最相近的文件。

最简单的、最重要的两个调节变量是table_cache（MySQL能够保持打开状态的表的数量）和key_buffer_size（MySQL能够在内存中保持的索引数量，可最小化磁盘的访问）。

InnoDB数据库具有自己的操作特点，其工作方式与MyISAM表有着根本的不同，这里每个表都与明确的文件相关。由于磁盘空间事先被分配，因此InnoDB配置需要仔细的计划。

硬件也是服务器改善性能的一种简单方法，内存、CPU和磁盘具有首要的重要性。

MySQL带有一套基准测试程序，它可以比较不同平台的性能，包括其他的数据库。

MySQL在斯堪的纳维亚进行开发，对除英语之外其他语言具有很好的支持。可以很容易地用其他语言显示错误消息或添加字符集。

第14章 数据库安全

安全不是额外的可选参数。在所有的事情都建立之后，再去弥补出现的漏洞要比从一开始就确保数据正确难得多。作为一名数据库管理员，应当信任用户，不过他们也会犯错误，如果他们偶尔误删甚至他们可能不知道曾经放在那儿的表，你就要被别人责怪了。本章的大部分内容是关于如何管理用户和确保他们只做需要做的事。

本章包括：

- 连接时的安全
- 改变和分配密码
- 管理用户和许可权
- MySQL的许可权表
- 授权和撤销
- 危险的权限
- 应用程序和系统安全
- 加载本地数据的安全性问题

连接时的安全

在进行连接时，用下列方式连接才算是安全：

```
% mysql -uusername -ppassword
```

为了举例方便，本章中都使用可见的密码。这样做虽然比较方便，不过作为一个有安全意识的用户，不应该用这种方式连接，其原因如下：

- 任何人从你背后都能看到明文形式的密码。
- 从历史记录中可以发现密码（例如，在UNIX系统中，一个人进入了别的用户的终端系统后，可以通过浏览最近使用过的命令找到密码），所以，当看到提示要输入密码时，应当：

```
% mysql -uroot -p
Enter password:
```

如果要把密码存入一个文件中，要确保这个文件是足够安全的。例如，如果密码保存在服务器的用户本地目录下的my.cnf文件中，那么这个文件就不会被其他的用户看到，不过系统的root用户当然可以看到这个文件。应当明白，系统的root用户并不是必须和MySQL的root用户相同。同样，应用程序经常会使用一个配置文件来保存数据库的密码，所以也要确保它的安全。

警告：决不要把一个含有Web应用或者其他类似功能设置的数据库的密码保存在Web树中。

最后如果希望有安全措施，就不要使用环境变量MYSQL_PWD来保存密码。同样的原因，不要在命令行中详细列出密码。环境变量也是不安全的。

管理用户和许可权

MySQL是一个设计优秀、灵活性好和易于管理的许可权系统。许可权就是允许或者不允许特定的用户或者主机连接数据库服务器以及在数据库、表甚至表中的特定列中执行特定的操作。

例如，我们假设：

- 一个新闻网站包括一个数据库服务器、一个Web服务器，还有一个工作人员可以更新消息的内部局域网。来自服务器的连接，只应该有执行数据库中的SELECT命令的许可权，而来自内部局域网的连接则允许在数据库上使用UPDATE和INSERT命令。
- 一个金融交易系统拥有一个记载日志记录的数据库和一个含有结算客户收支差额的数据库。UPDATE命令可以在结算客户收支差额的数据库里面使用，却不能在日志数据库里面使用。
- 一个预约系统里普通的用户只能在指定的表里插入记录，而管理员则可以升级这个表。

MySQL数据库

一旦装上了MySQL，就可以自动生成MySQL数据库。对这个数据库表的彻底了解对于有效地管理系统至关重要（见表14.1）。MySQL数据库中的六个表会影响对系统的访问：

```
mysql> USE mysql;
Database changed
mysql> SHOW TABLES;
+-----------------+
| Tables_in_mysql |
+-----------------+
| columns_priv    |
| db              |
| func            |
| host            |
| tables_priv     |
| user            |
+-----------------+
6 rows in set (0.00 sec)
```

表14.1 MySQL表

表	描述
user	列出可以访问服务器的用户、相关主机和密码和具有的全局许可权。最好不要允许任何全局的许可权，而只允许他们访问特定的一个或几个表
db	列出用户可以访问的数据库。数据库中的所有表都被授予了许可权
host	和db表一起允许的基于特定的主机的更严格的访问控制
tables_priv	列出可以访问的特定的表，表中的所有列都被授予了许可权
columns_priv	列出可以访问的特定的列
func	尚未被使用

表中包含的字段

我们看一下在MySQL数据库中的表。你的版本可能会有一些细小的差别。

```
mysql> SHOW COLUMNS FROM user;
+------------------------+---------------+------+-----+---------+-------+
| Field                  | Type          | Null | Key | Default | Extra |
+------------------------+---------------+------+-----+---------+-------+
| Host                   | varchar(60)
                           binary        |      | PRI |         |       |
| User                   | varchar(16)
                           binary        |      | PRI |         |       |
| Password               | varchar(16)
                           binary        |      |     |         |       |
| Select_priv            | enum('N','Y')|      |     | N       |       |
| Insert_priv            | enum('N','Y')|      |     | N       |       |
| Update_priv            | enum('N','Y')|      |     | N       |       |
| Delete_priv            | enum('N','Y')|      |     | N       |       |
| Create_priv            | enum('N','Y')|      |     | N       |       |
| Drop_priv              | enum('N','Y')|      |     | N       |       |
| Reload_priv            | enum('N','Y')|      |     | N       |       |
| Shutdown_priv          | enum('N','Y')|      |     | N       |       |
| Process_priv           | enum('N','Y')|      |     | N       |       |
| File_priv              | enum('N','Y')|      |     | N       |       |
| Grant_priv             | enum('N','Y')|      |     | N       |       |
| References_priv        | enum('N','Y')|      |     | N       |       |
| Index_priv             | enum('N','Y')|      |     | N       |       |
| Alter_priv             | enum('N','Y')|      |     | N       |       |
| Show_db_priv           | enum('N','Y')|      |     | N       |       |
| Super_priv             | enum('N','Y')|      |     | N       |       |
| Create_tmp_table_priv  | enum('N','Y')|      |     | N       |       |
| Lock_tables_priv       | enum('N','Y')|      |     | N       |       |
| Execute_priv           | enum('N','Y')|      |     | N       |       |
```

```
| Repl_slave_priv       | enum('N','Y')|      |     | N       |       |
| Repl_client_priv      | enum('N','Y')|      |     | N       |       |
| ssl_type              | enum('','ANY'
                         ,'X509'
                         ,'SPECIFIED') |      |     |         |       |
| ssl_cipher            | blob         |      |     |         |       |
| x509_issuer           | blob         |      |     |         |       |
| x509_subject          | blob         |      |     |         |       |
| max_questions         | int(11)
                           unsigned     |      |     | 0       |       |
| max_updates           | int(11)
                           unsigned     |      |     | 0       |       |
| max_connections       | int(11)
                           unsigned     |      |     | 0       |       |
+-----------------------+--------------+------+-----+---------+-------+
31 rows in set (0.00 sec)
mysql> SHOW COLUMNS FROM db;
+-----------------+-----------------+------+-----+---------+-------+
| Field           | Type            | Null | Key | Default | Extra |
+-----------------+-----------------+------+-----+---------+-------+
| Host            | char(60) binary |      | PRI |         |       |
| Db              | char(64) binary |      | PRI |         |       |
| User            | char(16) binary |      | PRI |         |       |
| Select_priv     | enum('N','Y')   |      |     | N       |       |
| Insert_priv     | enum('N','Y')   |      |     | N       |       |
| Update_priv     | enum('N','Y')   |      |     | N       |       |
| Delete_priv     | enum('N','Y')   |      |     | N       |       |
| Create_priv     | enum('N','Y')   |      |     | N       |       |
| Drop_priv       | enum('N','Y')   |      |     | N       |       |
| Grant_priv      | enum('N','Y')   |      |     | N       |       |
| References_priv | enum('N','Y')   |      |     | N       |       |
| Index_priv      | enum('N','Y')   |      |     | N       |       |
| Alter_priv      | enum('N','Y')   |      |     | N       |       |
+-----------------+-----------------+------+-----+---------+-------+
13 rows in set (0.01 sec)
mysql> SHOW COLUMNS FROM host;
+-----------------+-----------------+------+-----+---------+-------+
| Field           | Type            | Null | Key | Default | Extra |
+-----------------+-----------------+------+-----+---------+-------+
| Host            | char(60) binary |      | PRI |         |       |
| Db              | char(64) binary |      | PRI |         |       |
| Select_priv     | enum('N','Y')   |      |     | N       |       |
| Insert_priv     | enum('N','Y')   |      |     | N       |       |
| Update_priv     | enum('N','Y')   |      |     | N       |       |
| Delete_priv     | enum('N','Y')   |      |     | N       |       |
```

```
| Create_priv      | enum('N','Y')    |      |     | N       |       |
| Drop_priv        | enum('N','Y')    |      |     | N       |       |
| Grant_priv       | enum('N','Y')    |      |     | N       |       |
| References_priv  | enum('N','Y')    |      |     | N       |       |
| Index_priv       | enum('N','Y')    |      |     | N       |       |
| Alter_priv       | enum('N','Y')    |      |     | N       |       |
+------------------+------------------+------+-----+---------+-------+
12 rows in set (0.01 sec)
mysql> SHOW COLUMNS FROM tables_priv;
+-------------+---------------------+------+-----+---------+-------+
| Field       | Type                | Null | Key | Default | Extra |
+-------------+---------------------+------+-----+---------+-------+
| Host        | char(60) binary     |      | PRI |         |       |
| Db          | char(64) binary     |      | PRI |         |       |
| User        | char(16) binary     |      | PRI |         |       |
| Table_name  | char(60) binary     |      | PRI |         |       |
| Grantor     | char(77)            |      | MUL |         |       |
| Timestamp   | timestamp(14)       | YES  |     | NULL    |       |
| Table_priv  |set('Select','Insert',
              'Update', 'Delete',
              'Create', 'Drop',
              'Grant', 'References',
              'Index', 'Alter')   |      |     |         |       |
| Column_priv |set('Select','Insert',
              'Update','References')|     |     |         |       |
+-------------+---------------------+------+-----+---------+-------+
8 rows in set (0.01 sec)
mysql> SHOW COLUMNS FROM columns_priv; ;
+-------------+---------------------+------+-----+---------+-------+
| Field       | Type                | Null | Key | Default | Extra |
+-------------+---------------------+------+-----+---------+-------+
| Host        | char(60) binary     |      | PRI |         |       |
| Db          | char(64) binary     |      | PRI |         |       |
| User        | char(16) binary     |      | PRI |         |       |
| Table_name  | char(64) binary     |      | PRI |         |       |
| Column_name | char(64) binary     |      | PRI |         |       |
| Timestamp   | timestamp(14)       | YES  |     | NULL    |       |
| Column_priv | set('Select','Insert',
              'Update','References') |     |     |         |       |
+-------------+---------------------+------+-----+---------+-------+
7 rows in set (0.01 sec)
mysql> SHOW COLUMNS FROM func;
+-------+-----------------------------+------+-----+---------+-------+
| Field | Type                        | Null | Key | Default | Extra |
+-------+-----------------------------+------+-----+---------+-------+
```

```
| name  | char(64) binary                | | PRI |          |       |
| ret   | tinyint(1)                     | |     | 0        |       |
| dl    | char(128)                      | |     |          |       |
| type  | enum('function','aggregate') | |     | function |       |
+-------+--------------------------------+------+-----+----------+-------+
4 rows in set (0.01 sec)
```

表14.2描述了各种权限。

表14.2 每列的含义

列	说明
Host	用户用于连接的主机
User	用于连接的用户名（即-u参数）
Password	用户连接时使用的密码（即-p参数）
Db	用户试图执行操作的数据库
Select_priv	从一个表（SELECT语句）返回数据的权限。即使用户没有SELECT权限，SELECT的结果也可以被计算，不需要为返回结果而再次访问表
Insert_priv	在表中添加新的记录的权限（INSERT语句）
Update_priv	在表中修改数据的权限（UPDATE语句）
Delete_priv	从表中删除数据的权限（DELETE语句）
Create_priv	生成数据库和表的权限
Drop_priv	永久删除数据库和表的权限
Reload_priv	重新加载数据库的权限（FLUSH语句或者reload、refresh，或者从mysqladmin发出的刷新命令）
Shutdown_priv	关闭服务器的权限
Process_priv	查看当前的MySQL进程或者终止MySQL进程的权限（用于SHOW PROCESSLIST或KILL SQL语句）
File_priv	在服务器上读写文件的权限（用于LOAD DATA INFILE或SELECT INTO OUTFILE语句）。任何MySQL用户可以读取的文件都是可读的
Grant_priv	一个用户可以给其他用户授予特权的权限
References_priv	目前还没有被MySQL使用
Index_priv	生成、修改和撤销索引的权限
Alter_priv	改变当前表的结构的权限（ALTER语句）
Show_db_priv	浏览所有数据库的权限
Super_priv	即使已经达到最大的连接数目，连接、执行CHANGE MASTER、KILL线程，mysqladmin调试，PURGE MASTER、LOGS和SET GLOBAL命令的权限

（续表）

列	说明
Create_tmp_table_priv	生成临时表的权限（CREATE TEMPORARY TABLE）
Lock_tables_priv	锁定用户已具有SELECT权限的表的权限
Execute_priv	运行被存储的程序（为MySQL 5预定的）的权限
Repl_client_priv	询问从属服务器和主服务器有关复制的权限
Repl_slave_priv	复制的权限（见第12章“数据库复制”）
ssl_type	只有在使用安全套接层的情况下，授予连接的权限
ssl_cipher	只有在提供特别的加密程序的情况下，授予连接的权限
x509_issuer	只有在认证是由特定的使用者发出时才能授予连接的权限
x509_subject	只有在证书是由特定发行者发放的情况下，授予连接的权限
max_questions	用户每小时可以执行查询操作的最大次数
max_updates	用户每小时可以执行更新操作的最大次数
max_connections	用户每小时可以执行连接操作的最大次数

MySQL如何检查允许访问的权限

当一个用户试图连接时，MySQL首先检查用户表，以确定所列出的特定用户、主机和密码的组合。如果没有，则用户被拒绝访问。如果用户试图直接连接数据库，即便通过了其他检查，db表还是要被检查的。如果用户没有连接数据库的权限，访问将被拒绝。

当已完成连接的用户试图执行管理操作（比如说，mysqladmin shutdown），MySQL会检查用户表中与操作有关的列。如果所要求的操作被授予了权限，操作就能继续进行；如果没有，则操作会失败。

如果已完成连接的用户试图执行与数据库有关的操作（如select、update，等等），MySQL将从用户表中检查相关的字段，如果所要求的操作（select、update等）被授予了权限，操作就会被允许。如果没有，MySQL就会进入下一个步骤。

下一步是检查db表。MySQL查找用户正在其上执行操作的的数据库。如果这个数据库不存在，则许可权被禁止，操作失败。如果数据库存在，主机和用户匹配，则与操作有关的字段会被检查。如果所要求的操作被授予了权限，操作会成功。如果没有被授予权限，MySQL接着进入下一步。如果数据库和用户的组合存在，且主机的字段为空，MySQL就会检查主机表，看看主机是否能执行所要求的操作。如果在主机表中找到了主机和数据库，则在主机和db表上都相关的字段决定操作能否成功。如果两个表都授予了许可权，操作就会成功；如果没有，MySQL进入下一步。

MySQL检查tables_priv表，考虑要执行操作的表的情况。如果主机、用户、db与表的组合不存在，操作就会失败。如果存在，就会检查相关的字段。如果权限没有被授予，MySQL进入下一步。如果权限被授予，操作则会成功。

最后，MySQL检查columns_priv表，考虑操作中所用的列的情况。如果所要求的操作被授予了权限，则操作成功，否则，操作失败。

MySQL许可权表的优先权顺序参见图14.1。

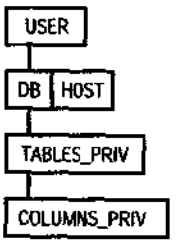

图14.1 MySQL许可权表的优先权

怎样产生许可权表

产生的许可权表由默认值填充：

```
mysql> SELECT * FROM user;
+-----------------------+----------------+-------------------+----
---------+-------------+-------------+-------------+-------------+-----
------+-------------+--------------+-------------+-------------+-------
-----+-------------------+------------+-----------+--------------+------
------+-----------------------+-------------------+--------------+------
-----------+-----------------+-----------+-----------+-------------+--
-----------+---------------+-------------+------------------+
| Host                  | User             | Password          |
Select_priv | Insert_priv | Update_priv | Delete_priv | Create_priv |
Drop_priv | Reload_priv | Shutdown_priv | Process_priv | File_priv |
Grant_priv | References_priv | Index_priv | Alter_priv | Show_db_priv |
Super_priv | Create_tmp_table_priv | Lock_tables_priv | Execute_priv |
Repl_slave_priv | Repl_client_priv | ssl_type | ssl_cipher |
x509_issuer | x509_subject | max_questions | max_updates |
max_connections |
+-----------------------+----------------+-------------------+----
---------+-------------+-------------+-------------+-------------+-----
------+-------------+--------------+-------------+-------------+-------
-----+-------------------+------------+-----------+--------------+------
------+-----------------------+-------------------+--------------+------
-----------+-----------------+-----------+-----------+-------------+--
-----------+---------------+-------------+------------------+
| localhost             | root             |                   | Y
| Y           | Y           | Y           | Y           | Y           | Y
| Y             | Y             | Y           | Y           | Y
| Y           | Y           | Y           | Y           | Y
| Y               | Y             | Y             | Y
|             |               |             |             |             0 |
```

```
0 |             0 |
| test.testhost.co.za       | root             |                  | Y
| Y           | Y           | Y           | Y           | Y           | Y
| Y           | Y           | Y           | Y           | Y
| Y           | Y           | Y           | Y           | Y
| Y           | Y           | Y           | Y
|             |             |             |             |             0 |
0 |             0 |
| localhost                 |                  |                  | N
| N           | N           | N           | N           | N           | N
| N           | N           | N           | N           | N
| N           | N           | N           | N           | N
| N           | N           | N           | N
|             |             |             |             |             0 |
0 |             0 |
| test.testhost.co.za       |                  |                  | N
| N           | N           | N           | N           | N           | N
| N           | N           | N           | N           | N
| N           | N           | N           | N           | N
| N           | N           | N           | N
|             |             |             |             |             0 |
0 |             0 |
+-------------------------+-------------------+-----------------+----
---------+-------------+-------------+-------------+-------------+-----
------+-------------+--------------+-------------+-----------+-------
-----+-------------------+-----------+-----------+--------------+------
------+----------------------+-----------------+--------------+------
----------+-----------------+----------+-----------+-------------+--
------------+--------------+------------+-----------------+
4 rows in set (0.05 sec)
mysql> SELECT * FROM db;
+------+---------+------+-------------+-------------+-------------+----
---------+-------------+-----------+-----------+-----------------+----
--------+------------+
| Host | Db      | User | Select_priv | Insert_priv | Update_priv |
Delete_priv | Create_priv | Drop_priv | Grant_priv | References_priv |
Index_priv | Alter_priv |
+------+---------+------+-------------+-------------+-------------+----
---------+-------------+-----------+-----------+-----------------+----
--------+------------+
| %    | test    |      | Y           | Y           | Y           | Y
| Y           | Y           | N           | Y           | Y           |
Y           |
| %    | test\_% |      | Y           | Y           | Y           | Y
```

```
| Y          | Y          | N          | Y              | Y          |
Y          |
+------+---------+------+------------+------------+-------------+----
---------+-------------+----------+-----------+-----------------+----
--------+-----------+
2 rows in set (0.01 sec)
mysql> SELECT * FROM host;
Empty set (0.00 sec)
mysql> SELECT * FROM tables_priv;
Empty set (0.00 sec)
mysql> SELECT * FROM columns_priv;
Empty set (0.00 sec)
```

注意，默认设置并不安全。任何人都能作为root用户从本地主机进行连接，并且有完整权限。任何匿名用户（不提供用户名）可以从本地主机连接到默认测试的数据库，还可以连接到任何名称以test开头的数据库。

说明：在UNIX中，如果没有用户名，MySQL就使用UNIX的登陆用户名。这意味着某些人无需特殊的用户名，就能以root身份进入MySQL，而且他们还有完整的权限。

在一个新安装的系统里，首先要执行的任务之一就是设置新的权限，并且为root用户设置至少一个新密码。

直接处理许可权表

有两种设置权限的方法：通过使用MySQL的grant和revoke语句，或者直接改变表中的值。最容易也是最有效的方式是用grant和revoke语句，但理解表是如何影响权限的也是很重要的。现在看一下，通过使用基本的insert、update和delete等sql语句改变表中的值来改变权限。稍后，在“使用grant和revoke处理许可权表”一节中，你将看到其他方法。为了给root用户添加密码，写出下列语句：

```
mysql> UPDATE user SET password=PASSWORD('g00r002b') WHERE user='root';
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

注意，password()函数的使用。在直接更新表的时候，必须使用这个函数。通过给密码加密，就不能通过简单地读取表的内容而得知密码。例如：

```
mysql> SELECT host,user,password FROM user;
+--------------------+------+-------------------+
| host               | user | password          |
+--------------------+------+-------------------+
| localhost          | root | 43b591f759a842a9 |
| test.testhost.co.za | root | 43b591f759a842a9 |
| localhost          |      |                   |
| test.testhost.co.za |      |                   |
+--------------------+------+-------------------+
4 rows in set (0.00 sec)
```

警告：直接改变许可权的时候一定要小心。疏忽where从句，意味着所有的密码都改变了，并且当前所有的用户都不能进行连接了。

当直接改变mysql表时，改变的权限不会立刻生效。MySQL需要重新读授权表。可以通过使用flush privileges、mysqkadmin flush-priviledges，或者mysqladmin reload命令，强制MySQL这样做。

```
mysql> INSERT INTO user (Host,User,Password) VALUES ('localhost',
 'administrator', PASSWORD('admin_pwd'));
Query OK, 1 row affected (0.09 sec)
```

在许可权被刷新以前，这个数据并不生效。可以作为管理员用户不用密码就能连接。

```
% mysql -uadministrator;
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6 to server version: 4.0.1-alpha-max-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

在数据库被重新加载以前，作为管理员去连接是可以接受的。因为，没有发现特定的用户名，连接的状态与不需要密码的匿名用户一样。可以通过查看用户表中的第三条和第四条记了解这个情况。刷新以后，管理员用户再也不能不用密码就进行连接了。

```
mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
mysql> exit
Bye
 % mysql -uadministrator;
ERROR 1045: Access denied for user: 'administrator@localhost' (Using password: NO)
```

除了管理，做任何事情都使用root用户决不是什么好事。为了用户要执行的任务，日常的连接应该发展有特别许可权的用户。对于这个销售系统，要建立两个用户——管理员和普通用户。管理员拥有足够的权限去做任何事情，而普通用户会受到一定的限制。要增加管理员，只需简单地在用户表中添加一条记录，并给予一整套完整的权限。但这将意味着，这个销售系统的管理员将有访问系统中任何一个数据库的权限。通常，将权限范围限制在用户级别，然后在较低的级别激活权限。可以通过给用户添加一条记录，或者给数据库添加一条记录，完成权限的设置。不使用特殊字段的insert语句（为了打印方便），以db表为例，确认这个字段与你的版本中的字段相匹配，以防它们发生了变化。

```
mysql> INSERT INTO user (host,user,password)
VALUES('localhost','administrator',PASSWORD('l3tm31n'));
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO db
VALUES('localhost','firstdb','administrator','Y','Y','Y','Y','N','N',
'N','N','N','N');
Query OK, 1 row affected (0.01 sec)
```

表中包含的内容如下：

```
mysql> SELECT * FROM user;
+-----------------------+-----------------+-----------------+----
---------+-------------+-------------+-------------+-------------+-----
------+-------------+--------------+-------------+-----------+-------
-----+-----------------+-----------+-----------+-------------+------
------+-----------------------+-----------------+-------------+------
----------+-------------------+----------+-----------+-------------+--
-----------+---------------+-------------+-----------------+
| Host                  | User            | Password        |
Select_priv | Insert_priv | Update_priv | Delete_priv | Create_priv |
Drop_priv | Reload_priv | Shutdown_priv | Process_priv | File_priv |
Grant_priv | References_priv | Index_priv | Alter_priv | Show_db_priv |
Super_priv | Create_tmp_table_priv | Lock_tables_priv | Execute_priv |
Repl_slave_priv | Repl_client_priv | ssl_type | ssl_cipher |
x509_issuer | x509_subject | max_questions | max_updates |
max_connections |
+-----------------------+-----------------+-----------------+----
---------+-------------+-------------+-------------+-------------+-----
------+-------------+--------------+-------------+-----------+-------
-----+-----------------+-----------+-----------+-------------+------
------+-----------------------+-----------------+-------------+------
----------+-------------------+----------+-----------+-------------+--
-----------+---------------+-------------+-----------------+
| localhost             | root            |                 | Y
| Y           | Y           | Y           | Y           | Y           | Y
| Y           | Y           | Y           | Y           | Y           |
| Y           | Y           | Y           | Y           | Y           |
| Y           | Y           | Y           | Y           |
|             |             |             |             |             | 0 |
0 |             0 |
| test.testhost.co.za   | root            |                 | Y
| Y           | Y           | Y           | Y           | Y           | Y
| Y           | Y           | Y           | Y           | Y           |
| Y           | Y           | Y           | Y           | Y           |
| Y           | Y           | Y           | Y           |
|             |             |             |             |             | 0 |
0 |             0 |
| localhost             |                 |                 | N
| N           | N           | N           | N           | N           | N
| N           | N           | N           | N           | N           |
| N           | N           | N           | N           | N           |
| N           | N           | N           | N           |
|             |             |             |             |             | 0 |
0 |             0 |
```

```
| test.testhost.co.za      |                  |                  | N
| N          | N          | N          | N          | N          | N
| N            | N           | N          | N          | N
| N          | N          | N          | N          | N
| N              | N           | N             | N
|          |           |            |              |             0 |
0 |              0 |
| localhost               | administrator    | 26981a09472b4835 | N
| N          | N          | N          | N          | N          | N
| N            | N           | N          | N          | N
| N          | N          | N          | N          | N
| N              | N           | N             | N
|          |           |            |              |             0 |
0 |              0 |
+-------------------------+----------------+------------------+----
---------+-------------+------------+------------+-------------+-----
------+-------------+-------------+------------+-----------+--------
------+-----------------+------------+-----------+-------------+------
------+-------------------------+----------------+-------------+------
----------+-----------------+---------+-----------+-------------+--
----------+--------------+------------+-----------------+
5 rows in set (0.05 sec)

mysql> SELECT * FROM db;;
+-----------+---------+---------------+-------------+-------------+-----
---------+-------------+-------------+-----------+-----------+--------
---------+------------+-------------+
| Host      | Db      | User          | Select_priv | Insert_priv |
Update_priv | Delete_priv | Create_priv | Drop_priv | Grant_priv |
References_priv | Index_priv | Alter_priv |
+-----------+---------+---------------+-------------+-------------+-----
---------+-------------+-------------+-----------+-----------+--------
---------+------------+-------------+
| %         | test    |               | Y           | Y           | Y
| Y           | Y           | Y         | N           | Y
| Y           | Y           |
| %         | test\_% |               | Y           | Y           | Y
| Y           | Y           | Y         | N           | Y
| Y           | Y           |
| localhost | firstdb | administrator | Y           | Y           | Y
| Y           | N           | N         | N           | N
| N           | N           |
+-----------+---------+---------------+-------------+-------------+-----
---------+-------------+-------------+-----------+-----------+--------
---------+------------+-------------+
3 rows in set (0.01 sec)
```

管理员可以通过使用密码连接到数据库，但只能在firstdb数据库中处理数据。记住在这些许可权生效以前刷新表。

```
% mysqladmin reload -u root -p
Enter password:
% mysql mysql;
ERROR 1045: Access denied for user: 'root@localhost' (Using password: NO)
```

如果你不是作为root用户登录的，将得到匿名用户没有此权限的错误信息提示。

```
% mysql mysql;
ERROR 1044: Access denied for user: '@localhost' to database 'mysql'
```

使用grant和revoke处理许可权表

与直接更新表和必须刷新数据库相比，管理许可权更简单的方式是使用grant和revoke语句。grant的基本语法如下：

```
GRANT privilege ON table_or_database_name TO user_name@hostname
 IDENTIFIED BY 'password'
```

要为销售系统添加普通的用户，可以按照以下步骤执行：

```
mysql> GRANT SELECT ON sales.* TO regular_user@localhost IDENTIFIED BY '13tm37n_2';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM user;
+-----------------------+-----------------+-------------------+----
---------+-------------+-------------+-------------+-------------+-----
------+-------------+---------------+--------------+-----------+-------
-----+-----------------+------------+------------+---------------+------
------+-----------------------+-----------------+-------------+------
----------+-----------------+----------+------------+-------------+--
-----------+---------------+-------------+-----------------+
| Host                  | User            | Password          |
Select_priv | Insert_priv | Update_priv | Delete_priv | Create_priv |
Drop_priv | Reload_priv | Shutdown_priv | Process_priv | File_priv |
Grant_priv | References_priv | Index_priv | Alter_priv | Show_db_priv |
Super_priv | Create_tmp_table_priv | Lock_tables_priv | Execute_priv |
Repl_slave_priv | Repl_client_priv | ssl_type | ssl_cipher |
x509_issuer | x509_subject | max_questions | max_updates |
max_connections |
+-----------------------+-----------------+-------------------+----
---------+-------------+-------------+-------------+-------------+-----
------+-------------+---------------+--------------+-----------+-------
-----+-----------------+------------+------------+---------------+------
------+-----------------------+-----------------+-------------+------
----------+-----------------+----------+------------+-------------+--
-----------+---------------+-------------+-----------------+
```

```
| localhost                | root             |                   | Y
| Y          | Y          | Y          | Y          | Y          | Y
| Y            | Y            | Y          | Y          | Y
| Y          | Y          | Y            | Y          | Y
| Y              | Y            | Y            | Y
|            |            |              |              | 0 |
0 |              0 |
| test.testhost.co.za      | root             |                   | Y
| Y          | Y          | Y          | Y          | Y          | Y
| Y            | Y            | Y          | Y          | Y
| Y          | Y          | Y            | Y          | Y
| Y              | Y            | Y            | Y
|            |            |              |              | 0 |
0 |              0
| localhost                |                  |                   | N
| N          | N          | N          | N          | N          | N
| N            | N            | N          | N          | N
| N          | N          | N            | N          | N
| N              | N            | N            | N
|            |            |              |              | 0 |
0 |              0 |
| test.testhost.co.za      |                  |                   | N
| N          | N          | N          | N          | N          | N
| N            | N            | N          | N          | N
| N          | N          | N            | N          | N
| N              | N            | N            | N
|            |            |              |              | 0 |
0 |              0 |
| localhost                | administrator    | 26981a09472b4835 | N
| N          | N          | N          | N          | N          | N
| N            | N            | N          | N          | N
| N          | N          | N            | N          | N
| N              | N            | N            | N
|            |            |              |              | 0 |
0 |              0 |
| localhost                | regular_user     | 1bfcf83b2eb5e59  | N
| N          | N          | N          | N          | N          | N
| N            | N            | N          | N          | N
| N          | N          | N            | N          | N
| N              | N            | N            | N
|            |            |              |              | 0 |
0 |              0 |
+--------------------------+------------------+-------------------+----
--------+------------+------------+------------+------------+-----
------+------------+------------+------------+-----------+-------
```

```
-----+---------------+-----------+-----------+-------------+------
------+---------------------+-----------------+-------------+------
----------+-----------------+----------+-----------+-------------+--
-----------+---------------+-------------+-----------------+
6 rows in set (0.05 sec)
mysql> SELECT * FROM db;
+-----------+---------+--------------+-------------+-------------+----
---------+-------------+-------------+-----------+-------------+--------
---------+-----------+-----------+
| Host      | Db      | User         | Select_priv | Insert_priv |
Update_priv | Delete_priv | Create_priv | Drop_priv | Grant_priv |
References_priv | Index_priv | Alter_priv |
+-----------+---------+--------------+-------------+-------------+----
---------+-------------+-------------+-----------+-------------+--------
---------+-----------+-----------+
| %         | test    |              | Y           | Y           | Y
| Y           | Y           | Y         | N           | Y
| Y           | Y           |
| %         | test\_% |              | Y           | Y           | Y
| Y           | Y           | Y         | N           | Y
| Y           | Y           |
| localhost | firstdb | administrator | Y           | Y           | Y
| Y           | N           | N         | N           | N
| N           | N           |
| localhost | sales   | regular_user | Y           | N           | N
| N           | N           | N         | N           | N
| N           | N           |
+-----------+---------+--------------+-------------+-------------+----
---------+-------------+-------------+-----------+-------------+--------
---------+-----------+-----------+
4 rows in set (0.01 sec)
```

当使用grant命令时，口令自动被加密，因此不需要用password()函数进行加密。事实上，如果这样做了，就是给口令加密了两次！通过用新口令重新使用相同的许可权，可以变更口令。

可以用与授予许可权相同的方式，撤销许可权。

```
mysql> REVOKE SELECT ON sales.* FROM regular_user@localhost;
Query OK, 0 rows affected (0.01 sec)
mysql> SELECT * FROM user;
+-----------------------+-----------------+------------------+----
---------+-------------+-------------+-------------+-------------+-----
------+-------------+--------------+-------------+-----------+-------
-----+---------------+-----------+-----------+-------------+------
------+---------------------+-----------------+-------------+------
----------+-----------------+----------+-----------+-------------+--
```

```
-----------+--------------+-------------+-----------------+
| Host                      | User             | Password         |
Select_priv | Insert_priv | Update_priv | Delete_priv | Create_priv |
Drop_priv | Reload_priv | Shutdown_priv | Process_priv | File_priv |
Grant_priv | References_priv | Index_priv | Alter_priv | Show_db_priv |
Super_priv | Create_tmp_table_priv | Lock_tables_priv | Execute_priv |
Repl_slave_priv | Repl_client_priv | ssl_type | ssl_cipher |
x509_issuer | x509_subject | max_questions | max_updates |
max_connections |
+-------------------------+----------------+------------------+----
---------+------------+-------------+-------------+-------------+-----
------+------------+---------------+-------------+-----------+-------
-----+----------------+------------+---------+-------------+------
------+--------------------+-----------------+-------------+------
----------+-----------------+----------+-----------+-------------+--
-----------+---------------+-------------+-----------------+
| localhost                 | root             |                  | Y
| Y           | Y           | Y           | Y           | Y           | Y
| Y             | Y             | Y           | Y           | Y
| Y           | Y           | Y           | Y           | Y
| Y               | Y             | Y             | Y
|             |             |             |             |             0 |
0 |             0 |
| test.testhost.co.za       | root             |                  | Y
| Y           | Y           | Y           | Y           | Y           | Y
| Y             | Y             | Y           | Y           | Y
| Y           | Y           | Y           | Y           | Y
| Y               | Y             | Y             | Y
|             |             |             |             |             0 |
0 |             0 |
| localhost                 |                  |                  | N
| N           | N           | N           | N           | N           | N
| N             | N             | N           | N           | N
| N           | N           | N           | N           | N
| N               | N             | N             | N
|             |             |             |             |             0 |
0 |             0 |
| test.testhost.co.za       |                  |                  | N
| N           | N           | N           | N           | N           | N
| N             | N             | N           | N           | N
| N           | N           | N           | N           | N
| N               | N             | N             | N
|             |             |             |             |             0 |
0 |             0 |
| localhost                 | administrator    | 26981a09472b4835 | N
```

```
| N          | N          | N          | N          | N          | N
| N            | N          | N          | N          | N
| N          | N          | N          | N          | N
| N              | N          | N          | N
|          |          |          |          |          0 |
0 |          0 |
| localhost              | regular_user    | 1bfcf83b2eb5e59   | N
| N          | N          | N          | N          | N          | N
| N            | N          | N          | N          | N
| N          | N          | N          | N          | N
| N              | N          | N          | N
|          |          |          |          |          0 |
0 |          0 |
+------------------------+----------------+----------------+----
--------+-------------+------------+------------+------------+-----
------+-------------+-------------+-------------+-----------+-------
-----+-----------------+------------+------------+-------------+------
------+---------------------+-----------------+-------------+------
----------+-----------------+---------+------------+-------------+--
-----------+--------------+------------+----------------+
6 rows in set (0.05 sec)
mysql> SELECT * FROM db;
+----------+--------+--------------+------------+------------+----
--------+------------+------------+-----------+-----------+--------
---------+-----------+------------+
| Host      | Db      | User          | Select_priv | Insert_priv |
Update_priv | Delete_priv | Create_priv | Drop_priv | Grant_priv |
References_priv | Index_priv | Alter_priv |
+----------+--------+--------------+------------+------------+----
--------+------------+------------+-----------+-----------+--------
---------+-----------+------------+
| %         | test    |              | Y          | Y          | Y
| Y          | Y          | Y         | N          | Y
| Y          | Y          |
| %         | test\_% |              | Y          | Y          | Y
| Y          | Y          | Y         | N          | Y
| Y          | Y          |
| localhost | firstdb | administrator | Y          | Y          | Y
| Y          | N          | N         | N          | N
| N          | N          |
+----------+--------+--------------+------------+------------+----
--------+------------+------------+-----------+-----------+--------
---------+-----------+------------+

3 rows in set (0.00 sec)
```

注意，用户的所有记录都会从db数据库中删除，但是用户还存在于用户表中。除了直接删除之外，没有别的方式可以删除这些记录。在MySQL的早期版本里，没有许可权（称做USAGE权限）的用户仍然可以连接到服务器并访问某些信息，浏览现有的数据库。例如：

```
mysql> exit
Bye
% mysql -uregular_user -pl3tm37n_2
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 21 to server version: 4.0.1-alpha-max-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| firstdb  |
| mysql    |
| test     |
+----------+
3 rows in set (0.00 sec)
```

要删除用户的所有记录，可以直接从用户表中删除（以root用户连接时）：

```
mysql> exit
Bye
% mysql mysql -uroot -pg00r002b
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 22 to server version: 4.0.1-alpha-max-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> DELETE FROM user WHERE user='regular_user';
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM user;
+-------------------------+-----------------+------------------+----
---------+-------------+-------------+-------------+-------------+-----
------+-------------+---------------+--------------+-----------+-------
-----+-----------------+------------+------------+--------------+------
------+----------------------+-------------------+--------------+------
----------+-------------------+----------+-----------+-------------+--
----------+---------------+-------------+-----------------+
| Host                    | User            | Password         |
Select_priv | Insert_priv | Update_priv | Delete_priv | Create_priv |
Drop_priv | Reload_priv | Shutdown_priv | Process_priv | File_priv |
Grant_priv | References_priv | Index_priv | Alter_priv | Show_db_priv |
Super_priv | Create_tmp_table_priv | Lock_tables_priv | Execute_priv |
Repl_slave_priv | Repl_client_priv | ssl_type | ssl_cipher |
```

```
x509_issuer | x509_subject | max_questions | max_updates |
max_connections |
+-------------------------+----------------+----------------+----
---------+------------+------------+------------+-----------+-----
-----+------------+---------------+------------+-----------+------
-----+------------+---------------+------------+-----------+------
------+---------------------+----------------+-------------+------
----------+----------------+----------+-----------+-------------+--
----------+---------------+------------+----------------+
| localhost                 | root             |                 | Y
| Y           | Y           | Y           | Y           | Y         | Y
| Y             | Y           | Y           | Y           | Y
| Y           | Y           | Y           | Y           | Y
| Y               | Y           | Y               | Y
|           |             |             |             |             0 |
0 |             0 |
| test.testhost.co.za       | root             |                 | Y
| Y           | Y           | Y           | Y           | Y         | Y
| Y             | Y           | Y           | Y           | Y
| Y           | Y           | Y           | Y           | Y
| Y               | Y           | Y               | Y
|           |             |             |             |             0 |
0 |             0 |
| localhost                 |                  |                 | N
| N           | N           | N           | N           | N         | N
| N             | N           | N           | N           | N
| N           | N           | N           | N           | N
| N               | N           | N               | N
|           |             |             |             |             0 |
0 |             0 |
| test.testhost.co.za       |                  |                 | N
| N           | N           | N           | N           | N         | N
| N             | N           | N           | N           | N
| N           | N           | N           | N           | N
| N               | N           | N               | N
|           |             |             |             |             0 |
0 |             0 |
| localhost                 | administrator    | 26981a09472b4835 | N
| N           | N           | N           | N           | N         | N
| N             | N           | N           | N           | N
| N           | N           | N           | N           | N
| N               | N           | N               | N
|           |             |             |             |             0 |
0 |             0 |
+-------------------------+----------------+-----------------+----
```

```
---------+-----------+-----------+-----------+-----------+-----
------+-----------+-----------+-----------+----------+-------
-----+-----------+-----------+-----------+-----------+-------
------+-----------+-----------+-----------+-----------+-------
----------+-----------+----------+-----------+-----------+--
----------+-----------+----------+----------------+
5 rows in set (0.05 sec)
```

表14.3描述了所有可用的特权。

表14.3 特权

特权	描述
ALL	授予所有基本的许可权
ALL PRIVILEGES	与ALL一样
ALTER	改变表的结构（ALTER语句）的许可权，但不包括索引
CREATE	创建数据库或表的许可权，不包括索引
CREATE TEMPORARY TABLES	生成临时表（CREATE TEMPORARY TABLE语句）的许可权
DELETE	从表（DELETE语句）中清除记录的许可权
DROP	撤销数据库或者表的许可权，不包括索引
EXECUTE	运行存储的程序的许可权（为MySQL 5预定的）
FILE	在服务器上读写文件的许可权（LOAD DATA INFILE或者SELECT INTO OUTFILE语句）。所有MySQL用户可以读的文件都是可读的
GRANT	将用户自己拥有的许可权授予其他用户的许可权
INDEX	生成、修改和删除索引的许可权
INSERT	在表中添加新记录的许可权（INSERT语句）
LOCK TABLES	对有SELECT权限的用户锁定一个表的许可权
PROCESS	查看当前MySQL进程或者终止MySQL进程的许可权（SHOW PROCESSLIST或KILL SQL语句）
REFERENCES	目前MySQL还没有使用
RELOAD	重新载入数据库的许可权（FLUSH语句，或者从mysqladmin执行的reload、refresh或flush语句）
REPLICATION CLIENT	询问从属服务器和主服务器有关复制情况的许可权
REPLICATION SLAVE	从服务器进行复制的（从属服务器需要这个权限进行复制）许可权
SHOW DATABASES	浏览所有数据库的权限
SELECT	从表中返回数据的许可权（SELECT语句）

（续表）

特权	描述
SHUTDOWN	关闭服务器的许可权
SUPER	即使达到了最大的连接数量，也能连接并且执行CHANGE MASTER、KILL线程、mysqladmin调试、PURGE MASTER LOGS和SET GLOBAL命令的权限
UPDATE	在表中修改命令的许可权（UPDATE语句）
USAGE	连接到服务器并执行所有可以执行的语句的许可权（在MySQL 4的早期版本中，这个权限包括SHOW DATABASES语句）

前面的例子在销售数据库中对所有的表都授予了许可权。可以简单地通过变更数据库和表的名称来处理这个问题（见表14.4）。

表14.4 数据库和表的名称

名称	描述
.	在所有数据库中的所有表
*	在当前数据库中的所有表
databasename.*	在数据库databasename中所有的表
databasename.tablename	在数据库databasename中的表tablename

例如：

```
mysql> GRANT SELECT ON *.* TO regular_user@localhost IDENTIFIED BY '13tm37n_2';
Query OK, 0 rows affected (0.00 sec)
```

因为所有的数据库都被授予了许可权，所以就不需要有条目在数据库的表中，只有带有select_priv字段的用户表设置为Y：

```
mysql> SELECT * FROM user;
+-----------------------+--------------+-------------------+-------------
+-------------+-------------+-------------+-------------+-----------+--
----------+--------------+--------------+-----------+-----------+---
-------------+------------+------------+----------+------------+------
-------+--------------+
| Host                  | User         | Password          | Select_priv
| Insert_priv | Update_priv | Delete_priv | Create_priv | Drop_priv |
Reload_priv | Shutdown_priv | Process_priv | File_priv | Grant_priv |
References_priv | Index_priv | Alter_priv | ssl_type | ssl_cipher |
x509_issuer | x509_subject |
+-----------------------+--------------+-------------------+-------------
+-------------+-------------+-------------+-------------+-----------+--
```

```
----------+-------------+-------------+----------+------------+---
----------+-------------+-------------+----------+------------+------
------+--------------+
| localhost           | root          | 43b591f759a842a9 | Y
| Y           | Y           | Y           | Y           | Y           | Y
| Y             | Y           | Y           | Y           | Y
| Y           | Y           | NONE      |             |             |
|
| test.testhost.co.za | root          | 43b591f759a842a9 | Y
| Y           | Y           | Y           | Y           | Y           | Y
| Y             | Y           | Y           | Y           | Y
| Y           | Y           | NONE      |             |             |
|
| localhost           |               |                  | N
| N           | N           | N           | N           | N           | N
| N             | N           | N           | N           | N
| N           | N           | NONE      |             |             |
|
| test.testhost.co.za |               |                  | N
| N           | N           | N           | N           | N           | N
| N             | N           | N           | N           | N
| N           | N           | NONE      |             |             |
|
| localhost           | administrator | 74126e0c6742d7e9 | N
| N           | N           | N           | N           | N           | N
| N             | N           | N           | N           | N
| N           | N           | NONE      |             |             |
|
| localhost           | regular_user  | 1bfcf83b2eb5e591 | Y
| N           | N           | N           | N           | N           | N
| N             | N           | N           | N           | N
| N           | N           | NONE      |             |             |
|
+---------------------+-------------+-----------------+------------
+------------+-------------+-------------+-------------+-----------+--
-----------+--------------+--------------+-----------+-----------+---
-------------+-----------+-----------+----------+-----------+------
-------+--------------+
6 rows in set (0.00 sec)
```

用SET设置用户密码

另一个改变密码的方式，是用**set password**语句。任何非匿名用户可以用这种方式更改自己的密码（这是要认真分配用户的另一个原因。确实有用户变更密码，拒绝他人的访问，因为他们没有意识到他们共享这个用户名！）。

可以为已连接的用户设置密码：

```
mysql> SET PASSWORD=PASSWORD('g00r002b2');
Query OK, 0 rows affected (0.00 sec)
```

通过指定用户，一个在mysql数据库中可以访问用户表的用户能够为别的用户设置密码。

```
mysql> SET PASSWORD FOR root=PASSWORD('g00r002b');
Query OK, 0 rows affected (0.00 sec)
```

记住，要用password()函数加密口令。如果没有这样做，密码以明文方式存在用户表中，但是在与用户表中的密码进行比较之前，连接时给出的密码会自动被加密，因此无法连接（如果试图这样做，参考“如果不能连接时怎么做”一节中的内容）：

```
mysql> SET PASSWORD FOR root='g00r002b';
Query OK, 0 rows affected (0.00 sec)
```

现在，你又不能以root身份重新连接：

```
% mysql -uroot -pg00r002b2
ERROR 1045: Access denied for user: 'root@localhost' (Using password: YES)
```

使用mysqladmin设置用户密码

使用mysqladmin的时候，与grant语句一样，不能使用password()函数：

```
% mysqladmin -uroot -pg00r002b password g00r002b
```

通配符许可权

如果需要连接1001个主机，你不需要给这么多主机授予访问权。MySQL在主机表中接受通配符。例如，下面的例子就允许用户与以marsorbust.co.za结尾的主机进行连接。

```
mysql> GRANT SELECT ON sales.* TO regular_user@"%.marsorbust.co.za"
 IDENTIFIED BY '13tm37n';
Query OK, 0 rows affected (0.03 sec)
mysql> SELECT * FROM user WHERE host LIKE '%mars%';
+-------------------------+-----------------+-----------------+----
---------+-------------+-------------+-------------+-------------+-----
------+-------------+--------------+-------------+-----------+-------
-----+-----------------+-----------+-----------+--------------+------
------+---------------------+-----------------+--------------+------
----------+-----------------+----------+-----------+-------------+--
----------+---------------+-------------+-----------------+
| Host                    | User            | Password        |
Select_priv | Insert_priv | Update_priv | Delete_priv | Create_priv |
Drop_priv | Reload_priv | Shutdown_priv | Process_priv | File_priv |
Grant_priv | References_priv | Index_priv | Alter_priv | Show_db_priv |
Super_priv | Create_tmp_table_priv | Lock_tables_priv | Execute_priv |
Repl_slave_priv | Repl_client_priv | ssl_type | ssl_cipher |
```

```
x509_issuer | x509_subject | max_questions | max_updates |
max_connections |
+-----------------------+----------------+-----------------+----
---------+-------------+-------------+-------------+-------------+-----
------+-------------+---------------+-------------+-----------+-------
-----+-----------------+------------+-------------+-------------+------
------+-----------------------+-----------------+-------------+-------
----------+-----------------+----------+-------------+-------------+--
----------+--------------+-------------+-----------------+
| %.marsorbust.co.za          | regular_user     | 1bfcf83b2eb5e591 | N
| N           | N           | N           | N           | N          | N
| N             | N           | N          | N          | N
| N          | N          | N           | N          | N
| N                | N           | N              | N
|          |            |             |              |             0 |
0 |               0 |
+-----------------------+----------------+-----------------+----
---------+-------------+-------------+-------------+-------------+-----
------+-------------+---------------+-------------+-----------+-------
-----+-----------------+------------+-------------+-------------+------
------+-----------------------+-----------------+-------------+-------
----------+-----------------+----------+-------------+-------------+--
----------+--------------+-------------+-----------------+
1 row in set (0.05 sec)
```

grant语句中引号中的部分允许使用通配符，或任何特殊字符。当然，也可以直接在MySQL表中插入通配符。

如果无法连接或者没有许可权时怎么办

这并非没有可能。也许你遗忘了重要的信息，在不该用的地方使用了DELETE命令，或者毁坏了含有许可权表的文件，即使作为root用户，现在也根本无法进行连接。不用害怕，有解决方案。

首先，彻底停止MySQL。作为UNIX的根用户，如果你在/init.d目录外运行MySQL，可以运行下列的程序：

```
% /etc/rc.d/init.d/mysql stop
Killing mysqld with pid 5091
Wait for mysqld to exit\c
.\c
.\c
.\c
.\c
020612 01:14:41  mysqld ended

done
```

如果不这样，作为根用户，你需要终止特定的MySQL相关的进程。

```
% ps -ax |grep mysql
 5195 pts/0    S      0:00 sh /usr/local/mysql/bin/mysqld_safe --datadir=/usr/lo
 5230 pts/0    S      0:00 /usr/local/mysql-max-4.0.1-alpha-pc-linux-gnu-i686/bi
 5232 pts/0    S      0:00 /usr/local/mysql-max-4.0.1-alpha-pc-linux-gnu-i686/bi
 5233 pts/0    S      0:00 /usr/local/mysql-max-4.0.1-alpha-pc-linux-gnu-i686/bi
 5234 pts/0    S      0:00 /usr/local/mysql-max-4.0.1-alpha-pc-linux-gnu-i686/bi
 5235 pts/0    S      0:00 /usr/local/mysql-max-4.0.1-alpha-pc-linux-gnu-i686/bi
 5236 pts/0    S      0:00 /usr/local/mysql-max-4.0.1-alpha-pc-linux-gnu-i686/bi
 5237 pts/0    S      0:00 /usr/local/mysql-max-4.0.1-alpha-pc-linux-gnu-i686/bi
 5238 pts/0    S      0:00 /usr/local/mysql-max-4.0.1-alpha-pc-linux-gnu-i686/bi
 5239 pts/0    S      0:00 /usr/local/mysql-max-4.0.1-alpha-pc-linux-gnu-i686/bi
 5240 pts/0    S      0:00 /usr/local/mysql-max-4.0.1-alpha-pc-linux-gnu-i686/bi
 5241 pts/0    S      0:00 /usr/local/mysql-max-4.0.1-alpha-pc-linux-gnu-i686/bi
% kill 5195 5230 5232 5233 5234 5235 5236 5237 5238 5239 5240 5241
mysqld ended
```

有可能这样做没有效果，那么不得不使用kill－9（后面是进程的id）去真正终止进程。在Windows中，可以简单地使用任务管理器关闭MySQL。

然后，在没有授权表的情况下（这里忽略了许可权限制）重新启动MySQL：

```
% mysqld_safe --skip-grant-tables
```

现在，应该能够添加一个根密码，无论是直接处理表，用GRANT语句，还是用mysqladmin：

```
% mysqladmin -u root password 'g00r002b'
```

不要忘记停止服务器，并在不使用--skip-grant-tables的情况下重新启动服务器，来激活根密码。

如果用户表崩溃了怎么办

有时会发生用户表发生崩溃的情况，因此就不能用mysqladmin变更密码。我曾在一次系统瘫痪事故后发生了这种情况，并且在文件目录被损坏后，这种情况也有可能发生。如果你想使用下面这个例子，可以通过重新命名来模仿用户表的损失情况，然后刷新表（原件可能被缓存在其他地方）：

```
% mv user.MYD user_bak.olddata
% mysql -uroot -pg00r002b;
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6 to server version: 4.0.1-alpha-max-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>FLUSH TABLES;
mysql> SELECT * FROM user;
ERROR 1016: Can't open file: 'user.MYD'. (errno: 145)
```

如果是这种情况，还可以在没有授权表的情况下启动MySQL，接着试着丢弃表：

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

CREATE TABLE user (
  Host varchar(60) binary NOT NULL default '',
  User varchar(16) binary NOT NULL default '',
  Password varchar(16) binary NOT NULL default '',
  Select_priv enum('N','Y') NOT NULL default 'N',
  Insert_priv enum('N','Y') NOT NULL default 'N',
  Update_priv enum('N','Y') NOT NULL default 'N',
  Delete_priv enum('N','Y') NOT NULL default 'N',
  Create_priv enum('N','Y') NOT NULL default 'N',
  Drop_priv enum('N','Y') NOT NULL default 'N',
  Reload_priv enum('N','Y') NOT NULL default 'N',
  Shutdown_priv enum('N','Y') NOT NULL default 'N',
  Process_priv enum('N','Y') NOT NULL default 'N',
  File_priv enum('N','Y') NOT NULL default 'N',
  Grant_priv enum('N','Y') NOT NULL default 'N',
  References_priv enum('N','Y') NOT NULL default 'N',
  Index_priv enum('N','Y') NOT NULL default 'N',
  Alter_priv enum('N','Y') NOT NULL default 'N',
  Show_db_priv enum('N','Y') NOT NULL default 'N',
  Super_priv enum('N','Y') NOT NULL default 'N',
  Create_tmp_table_priv enum('N','Y') NOT NULL default 'N',
  Lock_tables_priv enum('N','Y') NOT NULL default 'N',
  Execute_priv enum('N','Y') NOT NULL default 'N',
  Repl_slave_priv enum('N','Y') NOT NULL default 'N',
  Repl_client_priv enum('N','Y') NOT NULL default 'N',
  ssl_type enum('','ANY','X509','SPECIFIED') NOT NULL default '',
  ssl_cipher blob NOT NULL,
  x509_issuer blob NOT NULL,
  x509_subject blob NOT NULL,
  max_questions int(11) unsigned NOT NULL default '0',
  max_updates int(11) unsigned NOT NULL default '0',
  max_connections int(11) unsigned NOT NULL default '0',
  PRIMARY KEY  (Host,User)
) TYPE=MyISAM COMMENT='Users and global privileges';
Query OK, 0 rows affected (0.00 sec)
```

这样做也许还不能给你许可权，然而：

```
mysql> GRANT SELECT ON sales.* TO regular_user@localhost IDENTIFIED BY '13tm37n';
ERROR 1047: Unknown command
mysql> exit
Bye
```

```
[root@test mysql]# mysqladmin -uroot password 'g00r002b'
mysqladmin: unable to change password; error: 'You must have privileges
 to update tables in the mysql database to be able to change passwords
 for others'
```

你需要再一次在表中插入一些值。此处添加默认值。如果有什么不同，就是要确认这些值与列所生成的用户表的列值相匹配。

```
[root@test mysql]# mysql mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5 to server version: 4.0.1-alpha-max-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>INSERT INTO user VALUES ('localhost', 'root', '', 'Y', 'Y', 'Y',
 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
 'Y', 'Y', 'Y', 'Y', '', '', '', '', 0, 0, 0);
Query OK, 1 row affected (0.01 sec)
mysql>INSERT INTO user VALUES ('%', 'root', '', 'Y', 'Y', 'Y',
 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
 'Y', 'Y', 'Y', 'Y', '', '', '', '', 0, 0, 0);
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO user VALUES ('localhost', '', '', 'N', 'N', 'N',
 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
 'N', 'N', 'N', 'N', '', '', '', '', 0, 0, 0);
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO user VALUES ('localhost', '', '', 'N', 'N', 'N',
 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
 'N', 'N', 'N', 'N', '', '', '', '', 0, 0, 0);
Query OK, 1 row affected (0.00 sec)
```

你需要为了激活许可权重新载入（或者刷新权限表），接着再一次开始使用命令：

```
mysql> exit
Bye
[root@test mysql]# mysqladmin reload
[root@test mysql]# mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 4.0.1-alpha-max-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> GRANT SELECT ON sales.* TO regular_user@localhost IDENTIFIED BY '13tm37n';
Query OK, 0 rows affected (0.00 sec)
```

其他的授权参数

在默认状态下，MySQL并不允许用户将自己的许可权传递给其他人。我不建议你允许你的用户执行这类的操作。你可能有足够的原因不先给他们授予许可权，而且你不希望其他用户的权限超过这个范围。但是，如果你必须——也许在有若干值得信赖的用户的情况下，

这也是个办法。With grant option命令将允许用户将自己的许可权授权其他用户。

下面通过使用两个数据库：销售和客户数据库，来展示了这种情况。管理员创建了在销售数据库中有执行select查询许可权的regular_user2，接着授权GRANT参数给第一个regular_user，他有在客户数据库执行SELECT语句的许可权，反过来，他又给regular_user2授予了同样的权力。

```
mysql> GRANT SELECT ON sales.* TO regular_user2@localhost IDENTIFIED BY '13tm37n';
Query OK, 0 rows affected (0.01 sec)
mysql> GRANT SELECT ON customer.* TO regular_user@localhost IDENTIFIED
 BY '13tm37n' WITH GRANT OPTION;
Query OK, 0 rows affected (0.00 sec)
mysql> exit
Bye
% mysql -u regular_user2 -p13tm37n
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 4.0.1-alpha-max-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> GRANT SELECT ON customer.* TO regular_user@localhost IDENTIFIED
 BY '13tm37n' WITH GRANT OPTION;
ERROR 1044: Access denied for user: 'regular_user2@localhost' to database 'customer'
```

regular_user2不能给其他用户授予任何权限。

```
mysql> exit
Bye
[root@test /root]# mysql -u regular_user -p13tm37n
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4 to server version: 4.0.1-alpha-max-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> GRANT SELECT ON customer.* TO regular_user@localhost IDENTIFIED
 BY '13tm37n' WITH GRANT OPTION;
Query OK, 0 rows affected (0.00 sec)
```

然而，regular_user可以授予regular_user2许可权。

还有其他的有效参数能够帮助避免出现一个用户私自占有连接的情况。这些参数限制查询、更新，或者每小时连接的次数。有下面三种选择：

```
MAX_QUERIES_PER_HOUR n
MAX_UPDATES_PER_HOUR n
MAX_CONNECTIONS_PER_HOUR n
```

如果没有这三个参数，那么对用户活动的惟一限制就是max_user_connections变量。但这是全局变量，你不能对来自某个活动连接的用户进行限制。

对用户的限制在何处有效

限制对数据库深入细致的查询（例如，搜索涉及许多大型表的联接）可能有效。有了这个限制，应用程序能够以不同用户的身份进行连接，所以应用程序仍是一个可以做这种限制的好地方。但是，每小时一次的限制在某些情况下可能会有用，例如，可能会受到拒绝服务攻击的情况，或者同一个用户有若干连接，会导致数据库服务器性能下降的情况。

使用哪一种类型的用户限制

连接通常对数据库的影响最小，不过还是有这种可能性，即max_user_connections会被同一个用户占用。对MAX_CONNECTIONS_PER_HOUR进行设置是最谨慎的选择。

有时你并不担心连接的数量，但是担心同一个用户在同一时刻，执行了多个查询，或者执行外部的查询语句。可以设置MAX_QUERIES_PER_HOUR语句以避免用户执行不必要的查询和浪费资源，或者在很短的时间内执行太多的使负荷过重的查询语句。

UPDATE比SELECT对性能的影响更大，并且在锁定表的时候（例如，默认MyISAM表类型）也有很显著的影响。出于系统性能的原因，或者对不需要进行太多更新操作的用户，可以使用MAX_UPDATES_PER_HOUR进行限制。

提示： 这似乎是妄想。用户可能会有意地或无意地滥用数据库。如果认为某些许可权对他们没有用，就不要授权给他们。一旦授予许可权，取消它们可比授权的时候难得多。曾经碰到一个大型系统，它的安全方面只由一个用户和一个密码组成。一旦安全受到威胁，再从容地加上限制通常就不大可能了。

下面是一个限制用户的例子，你可以将regular_user2限制在每小时进行两次连接：

```
mysql> GRANT SELECT ON sales.* TO regular_user2@localhost IDENTIFIED BY '13tm37n'
    WITH MAX_CONNECTIONS_PER_HOUR 2;
Query OK, 0 rows affected (0.00 sec)
```

如果regular_user2超过了连接的限制次数，他们就会得到一个出错信息，如下所示：

```
ERROR 1226: User 'regular_user2' has exceeded the 'max_questions'
resource (current value: 2)
```

同样的，如果MAX_QUERIES_PER_HOUR也被分配值了且被超过了，错误信息就会如下所示：

```
ERROR 1226: User 'regular_user2' has exceeded the 'max_questions'
 resource (current value: 4)
```

提示： 你的限制当然要合情合理。如果一个用户应该一小时只执行一次查询，你就要认识到，他可能会因为输入错误而不得不执行第二次查询操作。

安全管理用户的策略

需求越复杂，策略也就越复杂。简单的网站有两个用户就足够了：一个可以更新数据的管理员，一个可以在特定表中执行SELECT语句的网站应用程序的用户。通用原则是：只给用户授予他们需要的权限，不要太多。如果他们以后需要更多权限，给他们增加权限是一件很容易的事情，但是如果要把这些权限撤销就是另外一回事了。

MySQL用户大体上有三种类型：个人用户（例如，Anique或Channette）、应用程序（例如，工资系统或新闻网站）和任务（例如，更新新闻或更新工资）。这些用户的任务在不同程度上会彼此交叉。例如，Anique会更新工资和新闻，即同时使用应用程序并执行两项任务。DBA需要决定是否给Anique和Channette其各自的密码，给新闻系统和工资系统设定密码，或者生成一个可以更新新闻和工资的用户。

如果你选择个人用户，并给Anique她自己的密码，那她只要记住一个注册码就可以进入数据库了。但是她需要被授予更新工资和新闻数据库的权限。如果她，或她使用的应用程序犯了一个错误，那么就会有损坏数据的可能性，造成她无法工作。例如，工资和新闻数据库都有一个days_data表。新闻数据库的版本不断更新，直到被存档；工资数据在被处理后，手工删除。那么就有可能她想删除的是工资表，却删除了新闻表。

如果你选择应用程序，要解决这样一些问题。现在，好像一个用户不得不记住两个密码。而且，你无法跟踪是哪个用户对哪个数据库进行了变更操作。但有解决的办法，因为安全性是必不可少的，很可能个人用户不得不记录日志到应用程序中（潜在地允许你跟踪那些对数据库做出修改的个人），然后应用程序记录日志到数据库。对两个应用程序，用户可以有相同的用户名和密码，但它们决不能在作为工资应用程序连接的时候，破坏新闻数据（只要你不给工资用户能更新新闻数据库的许可权就可以了）。

应用程序经常会有许多带有不同级别用户的任务。也许每个人都可以浏览自己的工资信息，但只有管理员能够更新它。给应用程序更新数据的许可权，就是潜在地给普通用户更新数据的权力。还要考虑开发过程：一个可信赖的高级开发人员构建了工资管理应用程序部分，一组级别较低的开发者构建了工资浏览部分的应用程序。授予相同的密码给应用程序就是允许低级别的开发者更新他们不需要更新或者不应该更新的数据。在这种情况下，你可以授予一个基于任务和应用程序组合在一起的用户名（工资管理员、工资浏览、新闻管理员、新闻浏览）。

应当牢记下列几个原则：

- 不要给用户根密码。它们总是与别的用户名和密码连在一起。
- 总是授予你所能给的最小许可权（但是要合情合理。你会发现有人因为被授予逐条查询的许可权而欣喜若狂。例如，允许了用户读取姓氏那一列，就迫使他们请求更多的许可权去读取紧随其后的名字）。在用户表中分配全局许可权时，应当总是否定，然后在db表中授权可以访问的特定的数据库。
- 对于关键的数据，一定要能够跟踪个人所做的变更过程。一般而言，人们都会把应用程序与数据库相结合，对个人级别访问权管理的负担，通常落在应用程序上。

避免授予危险的权限

尽管你总是授予所要求的最小权限，但是还会有一些权限特别危险，这些危险的威胁可能会超过其有益的方面。记住，决不要在全局级别上授权。

下列这些权限可能会有安全风险：

在mysql数据库上的权限 恶意用户权在浏览了加密的密码之后，就能获得访问权。

ALTER 恶意用户可以改变权限表，比如说重新命名，这会导致权限无效。

DROP 如果一个用户可以抛弃MySQL数据库，那么许可权限制将不会有什么作用。

FILE 有FILE权限的用户可以访问任何人可读的文件。他们还可以生成具有MySQL用户权限的文件。

GRANT 这个权限允许用户将他们的权限授权给那些可靠性不如他们的人。

PROCESS 正在运行的查询可以用明文的形式被浏览，这些查询包括任何改变或设置密码的操作。

SHUTDOWN 一个DBA不太可能傻到轻易授予他人这个权限。拥有SHUTDOWN权限的用户可以不打招呼就关闭服务器，造成任何人都无法访问。

SSL连接

客户和服务器之间的连接在默认状态下是不加密的。在大多数的网络体系结构中，这不应该有风险，因为数据库客户和服务器之间的连接是不公开的。但也有这样的情况，即数据需要移动到公用线路中，那么没有加密的连接就可能在数据移动时被看到。

MySQL可以设置为支持SSL连接，尽管这样做会对性能造成影响。要做到这一点，可以按照下列步骤进行：

1. 安装openssl库，此库可以在www.openssl.org/中找到。

2. 用--with-vio --with-openssl参数配置MySQL。

如果需要检测现在安装的MySQL是否支持SSL（或者安装是否正常工作），可以检查变量have_openssl的值是否为YES：

```
mysql> SHOW VARIABLES LIKE '%ssl';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| have_openssl  | YES   |
+---------------+-------+
1 row in set (0.00 sec)
```

一旦支持SSL，你就可以使用各种不同的参数（见表14.5）。

表14.5 SSL授权的参数

参数	描述
REQUIRE SSL	客户必须用SSL加密才能连接
REQUIRE X509	客户必须有合法的证书才能连接
REQUIRE ISSUER cert_issuer	客户必须有cert_issuer发布的有效证书才能够连接
REQUIRE SUBJECT cert_subject	客户必须有cert_subject主题的有效证书才能够连接
REQUIRE CIPHER cipher	客户必须使用特定的密码

REQUIRE SSL是SSL参数中限制最少的。任何形式的SSL加密都是可以接受的。如果你不想用明文形式发送，这个命令会有用，但对连接进行简单的加密就足够了。

```
mysql> GRANT ALL PRIVILEGES ON securedb.* TO root@localhost IDENTIFIED
 BY "g00r002b" REQUIRE SSL;
Query OK, 0 rows affected (0.01 sec)
```

REQUIRE X509也一样，不过相对而言限制更严格一些，因为必须有有效的证书：

```
mysql> GRANT ALL PRIVILEGES ON securedb.* TO root@localhost IDENTIFIED
 BY "g00r002b" REQUIRE X509;
Query OK, 0 rows affected (0.01 sec)
```

REQUIRE ISSUER和REQUIRE SUBJECT更安全，因为证书必须由指定的证书发布者颁发，或者包含有指定主题的证书：

```
mysql> GRANT ALL PRIVILEGES ON securedb.* TO root@localhost IDENTIFIED
 BY "g00r002b" REQUIRE ISSUER "C=ZA, ST=Western Cape, L=Cape Town,
 O=Mars Inc CN=Lilian Nomvete/Email=lilian@marsorbust.co.za";
Query OK, 0 rows affected (0.01 sec)
mysql> GRANT ALL PRIVILEGES ON securedb.* TO root@localhost IDENTIFIED
 BY "g00r002b" REQUIRE SUBJECT "C=ZA, ST=Western Cape, L=Cape Town,
 O=Mars Inc CN=Benedict Mhlala/Email=benedict@marsorbust.co.za";
Query OK, 0 rows affected (0.01 sec)
```

因为可以指定一个特殊的密码，REQUIRE CIPHER允许你不使用弱的SSL加密算法：

```
mysql> GRANT ALL PRIVILEGES ON securedb.* TO root@localhost IDENTIFIED
 BY "g00r002b" REQUIRE CIPHER "EDH-RSA-DES-CBC3-SHA";
Query OK, 0 rows affected (0.01 sec)
```

你也能同时指定任何一个或者以前所有的参数（用AND作为参数）：

```
mysql> GRANT ALL PRIVILEGES ON securedb.* TO root@localhost IDENTIFIED
 BY "g00r002b" REQUIRE ISSUER "C=ZA, ST=Western Cape, L=Cape Town,
 O=Mars Inc CN=Lilian Nomvete/Email=lilian@marsorbust.co.za" AND
 SUBJECT "C=ZA, ST=Western Cape, L=Cape Town, O=Mars Inc CN=Benedict
 Mhlala/Email=benedict@marsorbust.co.za" AND CIPHER "EDH-RSA-DES-CBC3-SHA";
Query OK, 0 rows affected (0.01 sec)
```

应用程序的安全性

大部分的安全漏洞是由于糟糕的应用程序所引起的。以下是需要避免的缺陷：

- 不要信任用户的数据。对用户输入的数据要进行校验。
- 在网站中，插入引号的方式是常见错误的原因。例如，一个应用程序接收了用户名和密码参数，然后运行如下这种查询语句：SELECT * FROM passwords WHERE username='$username' AND password='$password'。设计糟糕的应用程序会允许$password包含如下字符：aaa';DELETE FROM passwords;
  MySQL分解查询语句，认为在3个a后的单引号是查询语句的结束，MySQL会接着进行下一条查询语句。大部分计算机语言都有避免在一串字符中出现引号的简单功能，

比如C语言中的mysql_real_escape_string()和PHP中的addslashes()。

- 检查数据的大小。有单一数字的复杂运算可以运行得很好，但如果用户输入250个数字的参数，就有可能导致应用程序的崩溃。
- 删除任何传给MySQL的字符串中的特殊字符。
- 在数据和字符串旁边使用注释。

系统安全

在默认情况下，MySQL作为自己的用户在UNIX中运行。用户和组都是mysql。决不要让任何人以mysql用户的身份访问系统，这个用户只是为数据库本身设计的。MySQL也会生成一个单独的目录放置数据文件。只有MySQL用户才能访问这个目录。因为某种原因选择了默认设置，记住坚持这些原则：

- 隔离数据目录。
- 保证数据库目录的安全（没有人能够在MySQL目录中读取，更不要说写数据了）。
- 作为自己的用户运行MySQL。

LOAD DATA LOCAL的安全性问题

从客户机器中恢复数据是很方便的事情，不过它有一定的安全风险。有些人可以使用LOAD DATA LOCAL读取他们连接并访问的任何文件。他们可以通过生成一个表，再在载入数据后从表中读取数据。如果作为与Web服务器一样的用户正在进行连接，并且能访问和运行查询，就会很危险。在默认状态下，MySQL允许使用LOAD DATA LOCAL。要防止这种情况并禁止运行all LOAD DATA LOCAL，启动MySQL时使用--local-infile=0参数。没有--enable-local-infile参数也能编译MySQL。

小结

MySQL的用户管理机制是强大的、灵活的，但经常被滥用。MySQL数据库中包含不同的许可权表，它允许对用户、主机和执行的操作进行控制访问。可以使用SQL语句直接更新表（在这种情况下，刷新表会激活所做的变更），或者使用更方便的GRANT和REVOKE的语句。

在新安装的系统中，第一个任务就是设立一个根密码。到目前为止，任何人都能作为根用户进行连接，并且有完全的访问权。可以使用mysqladmin、SET语句或者GRANT语句设置密码。

MySQL允许用SSL连接增加安全性。但这不是默认安装的，因为这是附加的性能。

为了保护数据，应当了解以下这些基本原则：

- 不要给用户根密码，他们会用别的用户名进行连接。
- 不要让任何人访问用户表，即便是读一下。仅仅浏览一下加密的密码，就能让用户

有完整的访问权限。

- 只授予能授予的最少的许可权。授予最少的许可权意味着用户表中所有的列值都为N。
- 对于至关重要的数据，一定要可以跟踪个人所做的变更过程。一般来说，人们通过应用程序与数据库进行交互。管理在个人级别上的访问的责任通常会落在应用程序上。
- 确保从任何服务器，如果没有密码就不能作为根用户进行连接。
- 密码不应当以明文的形式保存，也不应当是字典词语。
- 经常检查用户的权限，确保没有人给他人授予了不适当的权限。

第15章 安装MySQL

可能你是一位初学者，需要安装MySQL进行一些实践。也可能系统管理员已经为你安装好了MySQL，你很好奇。不论什么原因，如果要深入地使用MySQL，那么某种程度上就需要自己安装MySQL。如果对于安装还有些犹豫，那么本章将帮助你轻松地完成安装，并使你了解所有这些烦恼都是些什么。

本章要点：

- 决定使用二进制版本产品还是源代码产品
- 在Windows上安装
- 在UNIX上安装
- 安装源代码产品和二进制版本产品
- 最优编译MySQL
- 在同一台机器上安装多个服务器
- 使用mysqld_multi来管理多台服务器
- 从MySQL版本3.23升级到版本4

决定安装源代码产品还是二进制产品

MySQL是一个开放源代码的开发产品，也就是说任何希望免费得到源代码的人都可以得到。像Linux和FreeBSD这样的操作系统也是开放源代码产品，但是Windows是专有软件，即源代码归Microsoft所有和控制。由于MySQL源代码是可用的，因此操作者可以通过两种方法来安装MySQL：

- 安装二进制版本，即操作者使用经MySQL开发商（或另一方）编译过的版本进行安装。
- 安装源代码版本，即操作者自己编译MySQL源代码并进行安装。

安装二进制版本的MySQL通常是最容易、最快的方法，但是这依赖于很多的因素，以及操作者对软件进行编译的接受程度。Windows用户几乎不需要对软件进行编译，但是像FreeBSD这样的用户可能会发现他们经常这样做。存在很多的原因导致可能要安装源代码产品：

- 正在安装的系统没有二进制版本软件。在写作本书时，二进制版本软件已经可以运行在Linux、FreeBSD、Windows、Solaris、MacOS X、HP-UX、AIX、SCO、SGI Irix、Dec OSF和BSDi上，尽管它们支持的并不都是最新版本的MySQL。
- 操作者认为可以使用不同的编译器或不同的编译参数对MySQL进行更好的优化。
- 操作者所希望的内容在二进制版本的软件中并没有，如额外的字符集、bug修复软件或不同的配置。

表15.1和表15.2提供了在二进制版本和源代码版本默认安装情况下的典型目录配置。

表15.1 二进制版本安装的目录

目录	描述
bin	二进制可执行文件的所在位置，包括所有的重要的mysqld，以及所有像mysqladmin、mysqlcheck、mysqlcheck和mysqldump这样的工具
data	实际的数据库，以及日志文件
include	C头文件
lib	编译库
scripts	包含mysql_install_db脚本
share/mysql	包含具体语言的错误信息的文件所在的目录
sql-bench	基准测试结果和工具

表15.2 源代码软件安装的目录

目录	描述
bin	客户端程序和工具的二进制可执行程序的所在位置，如mysqladmin、mysqlcheck和mysqldump
include	C头文件
info	Info格式的文档文件
lib	编译库
libexec	在源代码的默认安装中mysqld服务器所在位置，不是bin目录
share/mysql	包含具体语言的错误信息的文件所在的目录
sql-bench	基准测试结果和工具
var	实际的数据库以及日志文件

在Windows上安装MySQL

在Windows设备上安装MySQL需要下列配置：

- Windows家族的操作系统，换句话说就是现在的Windows 95/98/Me或Windows NT/2000/XP。
- MySQL的可执行程序或MySQL源代码软件的拷贝（如果操作者希望自己编译MySQL的话）。
- 软件解压程序。
- 为安装MySQL准备的足够的空间。
- 支持TCP/IP（传输控制协议/网间协议）。如果设备可以连接到Internet，那么系统已经支持TCP/IP协议了。如果不支持，那么需要安装TCP/IP（它是一个协议）。
- 如果操作者希望通过ODBC（开放式数据库互联）连接到MySQL，那么还需要MyODBC驱动程序，例如连接Microsoft Access到MySQL。

在Windows上安装二进制版本的软件

在Windows NT/2000/XP上安装时需要保证以具有管理员权限的用户登录。从较早安装的MySQL上进行升级需要停止服务器的运行。如果运行时是一个服务，那么可以像下面这样停止它：

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

或者也可以使用mysqladmin：

```
C:\mysql\bin> mysqladmin -u root -pg00r002b shutdown
```

如果操作者希望改变正在使用的可执行程序（如从mysqld-max-nt到mysqld-opt），在Windows NT/2000/XP上必须删除相应的服务：

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

执行下面的步骤：

1. 将压缩的软件解压到临时目录中。
2. 运行可执行文件setup.exe开始安装。虽然许多Windows用户喜欢安装到像C:\Program Files\MySQL\这样的目录中，但是MySQL默认安装在目录C:\mysql中。如果操作者改变了安装路径，那么必须在配置文件中指定新的位置（通常是my.ini）：

```
basedir=D:/installation-path/
datadir=D:/data-path/
```

3. MySQL带有很多可执行文件。根据需要选择其中之一运行（见表15.3）。

表15.3 可执行文件

文件	描述
mysqld	支持调试、自动内存分配检查、事务表（InnoDB和BDB）和符号链接
mysqld-opt	优化的二进制程序，不支持事务表（InnoDB或BDB）
mysqld-nt	支持命名管道的优化二进制程序（在NT/2000/XP中使用）。它可以在95/98/Me上运行，但是不生成命名管道，因为这些操作系统不支持它
mysqld-max	支持事务表和符号链接的优化二进制程序（InnoDB和BDB）
mysqld-max-nt	支持事务表（InnoDB和BDB）和符号链接的优化的二进制程序，这些事务表在NT/2000/XP上运行时是命名管道

在Windows NT/2000/XP上作为服务安装MySQL

如果操作者真地要在Windows上运行MySQL，那么当然希望它作为一个服务运行了。这可以使进程在Windows启动时自动启动（操作者一直希望数据库服务器这样做），并且在Windows关闭时自动地关闭。服务由Windows自己运行，并且不受用户登录或退出的影响。

在NT/2000/XP上，可以按照下面的方法将MySQL作为一个服务来安装：

```
C:\> c:\mysql\bin\mysqld-max-nt --install
```

如果操作者不希望自动启动MySQL，但还希望它作为服务来运行，那么可以运行相同的带有manual参数的命令：

```
C:\mysql\bin> mysqld-max-nt --install-manual
```

操作者随后可以按照下面的方法启动该服务：

```
C:\> net start mysql
The MySql service is starting.
The MySql service was started successfully.
```

并且通常可以利用命令mysqladmin shutdown或下面的方法停止它：

```
C:\> net stop mysql
The MySql service is stopping.....
The MySql service was stopped successfully.
```

如果要删除该服务，那么可以运行下面的命令：

```
C:\> c:\mysql\bin\mysqld-max-nt -remove
```

操作者还可以使用服务控制面板（Services Control Panel），并且点击启动（Start）或关闭（Stop）。

由于Windows NT并不等待MySQL而关机，因此我们可以注意到在MySQL的自动关闭方面，Windows NT存在问题（即关闭并不彻底，这增加了毁坏的机率）。

为了在NT上克服这个问题，需要进入注册表（Registry）并修改\winnt\system32\regedt32.exe，将注册表树中目录HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control下的WaitToKillServiceTimeout的值设置为新的毫秒值。

对于启动MySQL的更详细的内容可以见第10章“基本管理”。可以参考本章前面的表15.1对新建立目录的内容进行概要的了解。

在UNIX上安装MySQL

操作者将需要下面的环境以便在一台UNIX机器上安装MySQL：

- UNIX家族的操作系统（Linux、FreeBSD等等）
- MySQL的二进制版本或源代码版本的拷贝（如果操作者希望自己对MySQL进行编译）
- gunzip（gzip或zcat）和tar可以解开并解压缩发布的文件（推荐使用GNU版本）
- 在系统中为MySQL留出足够的空间
- 如果操作者计划对源代码软件进行编译，那么可以使用C++编译器（如gcc）

在UNIX上安装二进制版本（tar）的软件

要想在UNIX上安装MySQL的二进制版本软件，需要执行下面的步骤：

1. 变为根用户。操作者需要以根用户的身份来执行下面的命令：

```
% su -
Password:
```

2. 添加MySQL用户和MySQL组，MySQL将会以它来运行。永远都不要以根的身份运行MySQL！如果需要，操作者可以为用户和组赋予其他的名字：

```
% groupadd mysql
% useradd -g mysql mysql
```

说明： 这些命令在不同版本的UNIX上可能略有不同（例如adduser和addgroup）。

3. 进入所希望的存储MySQL的目录。默认情况下，二进制版本软件将安装在/usr/local下，但是操作者可以将它放在所希望的任何地方。如果改变了位置，那么操作者需要对设置和一些MySQL的工具进行一些调整，将它们指向新的位置：

```
% cd /usr/local
```

4. 现在解压文件：

```
% gunzip -c /home/ mysql-max-4.x.x-platform-os-extra.tar.gz | tar -xf -
```

操作者将看到的名字与正在使用的软件版本有关。现在的软件版本是mysql-max-4.0.2-alpha-pc-linux-gnu-i686。应该确保拥有与系统相配的正确的版本。

警告： Sun版本的tar已经出现了问题，因此可以使用GNU版本的tar。

根据操作者所安装的软件版本，在操作完成后，新的目录就会生成，如下所示：

```
% ls -l my*
total 1
drwxr-xr-x  13 mysql   users        1024
 Jul  1 14:15 mysql-max-4.x.x-platform-os-extra
```

对于每天的应用，这个名字有些笨拙。操作者更希望建立符号链接（symlink）mysql指向新的目录，这样MySQL的目录可以是/usr/local/mysql/：

```
% ln -s mysql-max-4.x.x-platform-os-extra mysql
% ls -l my*
lrwxrwxrwx   1 root     root           40
 Jul 27 23:07 mysql -> mysql-max-4.x.x-platform-os-extra
```

新安装的目录包括下面的内容：

```
% cd mysql
% ls -l
total 4862
```

```
-rw-r--r--   1 mysql    users       19106 Jul  1 14:06 COPYING
-rw-r--r--   1 mysql    users       28003 Jul  1 14:06 COPYING.LIB
-rw-r--r--   1 mysql    users      122323 Jul  1 13:16 ChangeLog
-rw-r--r--   1 mysql    users        6808 Jul  1 14:06 INSTALL-BINARY
-rw-r--r--   1 mysql    users        1937 Jul  1 13:16 README
drwxr-xr-x   2 mysql    users        1024 Jul  1 14:15 bin
-rwxr-xr-x   1 mysql    users         773 Jul  1 14:15 configure
drwxr-x---   4 mysql    users        1024 Jul  1 14:15 data
drwxr-xr-x   2 mysql    users        1024 Jul  1 14:15 include
drwxr-xr-x   2 mysql    users        1024 Jul  1 14:15 lib
drwxr-xr-x   2 mysql    users        1024 Jul  1 14:15 man
-rw-r--r--   1 mysql    users     2508431 Jul  1 14:06 manual.html
-rw-r--r--   1 mysql    users     2159032 Jul  1 14:06 manual.txt
-rw-r--r--   1 mysql    users       91601 Jul  1 14:06 manual_toc.html
drwxr-xr-x   6 mysql    users        1024 Jul  1 14:15 mysql-test
drwxr-xr-x   2 mysql    users        1024 Jul  1 14:15 scripts
drwxr-xr-x   3 mysql    users        1024 Jul  1 14:15 share
drwxr-xr-x   7 mysql    users        1024 Jul  1 14:15 sql-bench
drwxr-xr-x   2 mysql    users        1024 Jul  1 14:15 support-files
drwxr-xr-x   2 mysql    users        1024 Jul  1 14:15 tests
```

MySQL现在已经安装好了。若要安装许可权表（参见第14章“数据库安全”），需要运行脚本mysql_install_db：

```
% scripts/mysql_install_db
Preparing db table
Preparing host table
Preparing user table
Preparing func table
Preparing tables_priv table
Preparing columns_priv table
Installing all prepared tables
020701 23:19:07  ./bin/mysqld: Shutdown Complete
```

现在如果新建立的MySQL用户还没有拥有MySQL和数据目录的控制权，那么可以改变它们的所有权以确保新建立的MySQL用户对它们的控制：

```
% chown -R root /usr/local/mysql
% chgrp -R mysql /usr/local/mysql
% chown -R mysql /usr/local/mysql/data
```

操作完成后，MySQL将可以利用mysqld_safe来运行了：

```
% /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

提示： 需要记住一完成安装，就为mysqladmin分配一个管理密码。否则MySQL将对系统中的所有人开放。同样操作者应该立即着手使配置文件适合自己的设置。

参见本章前面的表15.1，可以对新建立的目录中的内容有一个大概的了解。

在UNIX上安装二进制版本的软件（rpm）

Red Hat Linux也允许操作者利用RPM（Red Hat包管理器）文件安装MySQL。表15.4显示了全部可用的RPM（版本号将反映操作者正在安装的软件版本）。

表15.4 Red Hat包管理器文件

文件	描述
MySQL-4.x.x-platform-os-extra.rpm	MySQL服务器软件，如果操作者没有与服务器连接，那么需要此软件
MySQL-client-4.x.x-platformos-extra.rpm	MySQL客户端软件，需要它来连接到MySQL服务器
MySQL-bench-4.x.x-platform-os-extra.rpm	各种MySQL测试和基准测试软件。需要Perl和msql-mysql的rpm文件
MySQL-devel-4.x.x-platformos-extra.rpm	各种库文件，包括为了编译其他MySQL客户端所需要的文件
MySQL-shared-4.x.x-platform-os-extra.rpm	MySQL客户端共享的一些库
MySQL-embedded-4.x.x-platformos-extra.rpm	MySQL内嵌服务器
MySQL-4.x.x-platform-osextra.src.rpm	旧rpm文件的源代码。如果操作者在进行二进制版本软件的安装，那么不需要这个软件
MySQL-Max-4.x.x-platformos-extra	MySQL Max rpm（支持InnoDB等数据库表）

要安装这些rpm文件，操作者仅运行列出的所要安装的rpm工具就可以了。客户端和服务器是操作者要安装的最小的部分：

```
% rpm -i MySQL-4.x.x-platform-os-extra.rpm MySQL-client-4.x.x-platform-os-extra.rpm
```

通过rpm安装的结果与从普通二进制版本软件安装的结果略有不同。数据放在/var/lib/mysql目录中，并且通过在/etc/rc.d/目录中建立条目，即建立脚本/etc/rc.d/init.d/mysql，MySQL被设置成在机器启动后自动开始运行。以这种方式在以前安装的MySQL上进行安装时要小心，这是因为操作者将不得不重新进行以前所做的改动。

安装完后，MySQL将可以利用mysqld_safe来运行了：

```
% /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

不要忘记分配一个管理员密码，并查看配置文件。

在UNIX上利用源代码文件安装

除非对从产品的源代码编译MySQL很有经验，否则操作者是不会希望做这类事的。但是，一些操作者可能还是会这样做，因此这一节将对这一过程进行详细的介绍。

提示： 由于接下来的信息随着新的版本代替旧的版本，可能很快就过期了，因此查阅最新的MySQL文档是很重要的。

利用源代码文件安装，操作者需要下列环境：

- gunzip（gzip或zcat）。GNU版本的软件被大家认为是可以工作的。
- tar（GNU版本的tar可以工作，但是Solaris版本的tar在过去产生了一些问题）。
- make（推荐使用GNU版本的make，Solaris和FreeBSD版本的make已经被大家知道出了一些问题了）。
- gcc或pgcc（或其他ANSI C++编译器）。虽然不同版本的软件通常使用不同的编译器进行编译，但是当前推荐使用版本2.95.2的软件。我们建议大家查阅最近的MySQL文档，看一看它们为你的操作系统编译了什么软件，并且是否存在一些其他编译器所报告的错误。

要安装源代码版本的软件，需要执行下面的步骤：

1. 像二进制文件安装时那样，需要成为根用户，并且添加MySQL用户和组：

```
% su -
Password:
% groupadd mysql
% useradd -g mysql mysql
```

2. 进入文件所要安装的目录（例如/usr/local/src或习惯指定的任何地方）。
3. 解开文件：

```
% gunzip -c /tmp/ mysql-4.x.x-platform-os-extra.tar.gz | tar -xf -
```

4. 一旦解压完成，新的目录就建立好了。进入这个新目录，我们即将从这里配置并安装MySQL：

```
% cd mysql-4.x.x-extra
```

5. 运行脚本文件configure，它是软件所携带的一个很有用的小脚本。它允许操作者为安装设置各种参数。这里有大量的参数可供使用。一些更有用的参数在后面的内容中会进行描述，其他的在后面的“最优配置MySQL”中涉及。
6. 默认情况下，利用源代码编译的MySQL将安装到/usr/local中，数据和日志文件存在/usr/local/var中。要想改变这些位置，可以使用prefix参数，例如：

```
% ./configure --prefix=/usr/local/mysql
```

这将全部安装的预置位置改变为/usr/local/mysql。同样，操作者可以改变数据的目录到/usr/local/mysql/data中，余下的安装与下面的相同：

```
% ./configure --prefix=/usr/local \
        --localstatedir=/usr/local/mysql/data
```

如果更喜欢，那么操作者可以像下面这样做：

- 如果操作者不希望编译服务器，而只是连接到已有服务器的客户端程序，那么可以使用--without-server参数：

```
% ./configure --without-server
```

- 要使用libmysqld.a，即嵌入式MySQL库，操作者需要采用--with-embedded-server参数：

```
% ./configure --with-embedded-server
```

- 要将默认的目录（通常是/tmp）改编为新的套接字文件目录，需要像下面这样使用configure（必须是全路径）：

```
% ./configure --with-unix-socket-path=/usr/local/sockets/mysql.sock
```

- 下面的操作可以得到所有可用的参数集：

```
% ./configure --help
```

7. 操作完成后（根据操作者的设置，这可能要花费一些时间），操作者需要使用make命令建立二进制软件：

```
% make
```

8. 接下来，操作者将需要安装这个二进制软件：

```
% make install
```

9. 现在，如果要安装二进制软件，那么将需要建立许可权表，并且改变这些文件的所有人。下面的这些例子假设操作者以/usr/local/mysql作为prefix参数：

```
% cd /usr/local/mysql
% scripts/mysql_install_db
Preparing db table
Preparing host table
Preparing user table
Preparing func table
Preparing tables_priv table
Preparing columns_priv table
Installing all prepared tables
010726 19:40:05  ./bin/mysqld: Shutdown Complete
% chown -R root /usr/local/mysql
% chgrp -R mysql /usr/local/mysql
% chown -R mysql /usr/local/mysql/data
```

10. 然后，MySQL将可以利用mysqld_safe运行了：

```
% /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

不要忘记分配一个根密码并查看配置文件。

最优编译MySQL

标准的MySQL版本相当接近于最优的编译，但是如果企图每次都能获得最大的性能，那么操作者需要进行一些改进。追得其反并使系统慢下来是很容易的，因此在使用下面的技巧时要非常小心：

- 操作者可以定义由编译器使用的标志或编译器的名字，例如：

```
% CFLAGS=-03
% CXX=gcc
% CXXFLAGS=-03
% CC=gcc
% export CC CFLAGS CXX CXXFLAGS
```

- 连接是静态的，不是动态的（换句话说，使用了--static参数）。这使用了更多的磁盘空间，但是运行起来更快了（根据MySQL的测试，在Linux上性能提高了13%）。
- MySQL的二进制版本软件大部分利用gcc进行编译，这是因为pgcc（Pentium gcc）已经在非Intel处理器上出现了问题。如果操作者的处理器是Intel Pentium家族的产品，那么利用pgcc进行编译可能会有一些优点（根据MySQL的测试，具有1%的增长，据报告最高可达10%的改善）。相似的，在Sun的服务器上，SunPro C++编译器过去比gcc快大约5%。
- 优化至可能的最高级别（使用gcc的-O3）。
- 不使用debug进行编译（--without-debug参数）。这会比使用--with-debug=full参数快20%到35%，比使用--with-debug快15%）。
- MySQL软件支持所有的字符集。如果操作者只是在使用默认的ISO-8859-1（Latin1）字符集，那么使用with-extra-charsets=none参数。对计划使用的字符集使用--with-charset=xxx参数。
- 如果在x86机器上运行Linux，那么不使用帧指针（frame pointers，-fomit-frame-pointer或-fomit-frame-pointer -ffixed-ebp）会使性能改善1%到4%。

在相同的机器上安装多个MySQL服务器

操作者很少会需要在相同的机器上运行多个MySQL服务器。这不会带来任何性能的增长，并且几乎没有操作者会允许多版本的服务器访问相同的数据。最可能的情况是，操作者在没有删除以前安装的MySQL时，对新版本的MySQL进行测试。

如果希望在同一台机器上运行多个版本的MySQL，那么操作需要确保它们没有企图使用相同的套接字（Socket）文件，或在同一个TCP/IP端口上进行侦听。它们还将具有自己的pid文件。在大多数系统中，默认的配置是端口3306和/tmp/mysql.sock。管理它的一个方便的途径是利用mysqld_multi工具，在本章稍后进行讨论。假设这是一个与其他服务器不同的配置，操作者可以在配置文件中改变默认的端口和TCP/IP设置。例如：

```
socket=/tmp/mysql2.sock
port=3307
```

客户端可以使用--socket参数连接到运行不同套接字的服务器：

```
% mysql --socket=/tmp/mysql2.sock -uroot -pg00r002b
```

还可以通过制定配置文件，为客户端指定要连接的服务器。例如：

```
% mysql --defaults-file=/usr/local/mysql2/etc/my.cnf -uroot -pg00r002b
```

如果操作者在编译MySQL，那么对第二台服务器配置新的端口号、套接字路径和安装目录。例如：

```
% ./configure  --with-tcp-port=3307 \
               --with-unix-socket-path=/tmp/mysql2.sock \
               --prefix=/usr/local/mysql2
```

警告： 永远不要令多台服务器控制相同的数据，这会很容易导致系统失效。它们也不应该需要写到相同的日志文件。

MySQL软件带有名为mysqld_multi的便利的工具，它对于管理多台MySQL服务器是很有用的工具（运行在不同的套接字和端口上）。要使用mysqld_multi，操作者需要设置配置文件中mysqld_multi的部分，以及对应于每台MySQL服务器的部分。例如：

```
[mysqld_multi]
mysqld     = /usr/local/bin/mysqld_safe
mysqladmin = /usr/local/bin/mysqladmin
user       = root
password   = g00r002b
[mysqld1]
socket     = /tmp/mysql.sock
port       = 3306
pid-file   = /usr/local/mysql/var/hostname.pid
datadir    = /usr/local/mysql/var
language   = /usr/local/share/mysql/english
user       = hartmann
[mysqld2]
socket     = /tmp/mysql.sock2
port       = 3307
pid-file   = /usr/local/mysql/var2/hostname.pid
datadir    = /usr/local/mysql/var2
language   = /usr/local/share/mysql/french
user       = yves
[mysqld3]
socket     = /tmp/mysql.sock3
port       = 3308
pid-file   = /usr/local/mysql/var3/hostname.pid
datadir    = /usr/local/mysql/var3
language   = /usr/local/share/mysql/german
user       = cleo
[mysqld4]
socket     = /tmp/mysql.sock4
port       = 3309
pid-file   = /usr/local/mysql/var4/hostname.pid
datadir    = /usr/local/mysql/var4
language   = /usr/local/share/mysql/english
user       = caledon
```

mysqld_multi的语法如下：

```
mysqld_multi [option/s] {start|stop|report} [group_no,group_no2...]
```

按照上面这个例子假设一个配置文件的设置，下面的例子显示了关于服务器状态的mysqld_multi的报告，并随后被用来关闭它：

```
% mysqld_multi --user=root --password=g00r002b report 1
Reporting MySQL servers
MySQL server from group: mysqld1 is running
% mysqld_multi --user=root --password=g00r002b stop 1
% 020729 04:20:50  mysqld ended
% mysqld_multi --user=root --password=g00r002b report 1
Reporting MySQL servers
MySQL server from group: mysqld1 is not running
```

表15.5对mysqld_multi参数进行了描述。

表15.5 mysqld_multi参数

参数	描述
--config-file=...	为组设置可选的配置文件（它将不会影响[mysqld_multi]组）
--example	提供一个配置文件样本
--help	显示帮助并退出
--log=...	指定日志文件，取文件的全路径和名字。如果这个文件已经存在，那么日志将附加到文件的末尾
--mysqladmin=...	mysqladmin二进制文件的全路径和名字，用来关闭服务器
--mysqld=...	mysqld二进制文件或者时常使用的mysqld_safe的全路径名和名字。参数被传递到mysqld。操作者将需要对mysqld_safe进行一些改变，或确保它在环境变量PATH中
--no-log	输出到标准输出，而不是日志文件。默认是写到日志文件中
--password=...	用户mysqladmin的密码
--tcp-ip	使mysqld_multi通过TCP/IP连接到MySQL服务器，而不是UNIX套接字。默认情况是通过UNIX上的套接字连接的
--user=...	mysqladmin的用户。确保该用户拥有正确的权利以完成所要做的事情（shutdown_priv）
--version	显示版本号并退出

避免常见的安装问题

事情出现了错误会使人非常的灰心，而且你那时还没有想法子开始。特别是对于初学者，在碰到深奥的错误信息时很难知道该做些什么。下面的部分将关注安装过程中发生的一

些常见问题。

启动mysqld时的问题

当操作者试图安装许可权表并失败时，启动mysqld的问题就会出现。就像在下面的列表中看到的那样，问题会有很多的原因。检查错误日志是查找问题原因的最好方法。

- 可能操作者的配置文件存在问题（my.cnf或my.ini）。仔细查看语法规则，或使用软件所带的标准的配置文件来查看是否可以启动MySQL。
- 另一个常见的错误是：

```
Can't start server: Bind on unix socket....
```

或者：

```
Can't start server: Bind on TCP/IP port: Address already in use
```

这个错误发生在试图再次安装MySQL的拷贝，它与现有的安装好的软件使用相同的端口或套接字。操作者需要确信在开始之前指定了一个不同的端口或套接字。

- 许可权问题也是很常见的问题。操作者需要确信按照安装一节中列出的步骤进行操作，这样至少MySQL目录是正确的。如果使用套接字，那么操作者还需要确信自己具有向套接字文件进行些操作的权利（通常是对/tmp）。
- 另一个常见的问题是库。举例来说，如果操作者运行的是Linux，并且已经安装了共享库，那么操作者需要确信这些共享库的位置在文件/etc/ld.so.conf中列出。例如，如果具有：

```
/usr/local/lib/mysql/libmysqlclient.so
```

那么需要确信/etc/ld.so.conf包含：

```
/usr/local/lib/mysql
```

然后运行ldconfig。

- 当在存有当前使用的BDB表的地方启动MySQL时，操作者可能会发现下面这样的问题：

```
020814 19:18:02  bdb:  warning: ./bdb/news.db: No such file or directory
020824 19:18:02  Can't init databases
```

这意味着BDB在恢复现有日志文件上存在问题。操作者既可以利用参数--bdb-no-recover启动MySQL，或者也可以删除这些日志文件。

编译问题

如果操作者在编译时遇到问题，并且需要再做一次，那么他需要确信configure从一个良好的无故障的环境中运行。否则，它会使用存放在config.cache文件中的先前操作的信息，在进行配置时，操作者每次都需要删除这个文件。同样，旧的一些对象文件可能一直存在，因此为了确保有一个良好的无故障的重编译，操作者应该删除这些文件。执行下面的操作：

```
% rm config.cache
% make clean
```

如果具有distclean，操作者还可以运行它。

操作者的编译器可能已经过期了。当前MySQL的建议是使用gcc 2.95.2或egcs 1.0.3a，但是它有可能已变了，因此需要查看最新个文档。

不兼容的make版本可能导致其他一些问题。当前，MySQL推荐GNU make，版本3.75或更高。

如果在编译sql_yacc.cc时遇到问题，那么可能会出现磁盘空间溢出。在一些情况下，这个文件的编译会耗尽太多的资源（甚至看起来还有很多可用的资源时）。错误可能是下面的一种：

```
Internal compiler error: program cclplus got fatal signal 11
Out of virtual memory
Virtual memory exhausted
```

使用参数--low-memory运行配置通常可以解决问题：

```
% ./configure --with-low-memory
```

如果相关的库有了问题，如g++、libg++或libstdc++（可能不可用），那么可以试着用gcc作为C++编译器，如下所示：

```
% CXX="gcc -O3" ./configure
```

除了更加优化外，静态链接还可以解决一些不明确的问题。

Windows问题

如果操作者双击setup.exe，进程开始执行但却一直不停，那么可能与MySQL有些冲突。可以尝试下面这些方法：

- 关闭所有的Windows应用程序，包括服务和系统框中的程序。
- 换一种方式，试着在安全模式下进行安装（在系统启动时按F8键，然后从菜单中选择）。
- 最坏的情况下，操作者可能不得不重新安装Windows，并且在安装其他软件之前先安装MySQL。这个问题很少发生在专门用做MySQL数据库服务器的生产机上。相反，常出现在运行各类其他应用程序的多种用途的计算机上。

从MySQL 3.x升级到MySQL 4

MySQL从版本3.23.xx到4有了很大的变化，在升级时有很多不同的地方需要当心：

- 脚本mysqld_safe代替了safe_mysqld。
- 在用户表中有大量的新权限（在mysql数据库中）。MySQL已经提供了一个脚本在维持现有权限的同时，添加这些新的权限。mysql_fix_privilege_tables从旧的FIL权限中取得REPLICATION SLAVE和REPLICATION CLIENT权限，从旧的PROCESS中取

得SUPER和EXECUTE权限。如果不运行这个脚本，所有的用户都将具有SHOW DATABASES、CREATE TEMPORARY TABLES和LOCK TABLES的权限。

- length和max_length（在结构MYSQL_FIELD中）的属性现在是unsigned long而不是unsigned int。
- 旧的--safe-show-database参数不推荐使用（它不再具备任何的功能，由user表中SHOW DATABASES的权限代替）。
- 一些变量进行了重命名：

```
myisam_bulk_insert_tree_size改为bulk_insert_buffer_size
query_cache_startup_type改为query_cache_type
record_buffer改为read_buffer_size
record_rnd_buffer改为read_rnd_buffer_size
sort_buffer改为sort_buffer_size
warnings改为log-warnings
```

- 一些mysqld启动参数有了新的名字：

```
--skip-locking改为--skip-external-locking
--enable-locking改为--external-locking
```

- 启动参数sam_max_extra_sort_file_size和myisam_max_extra_sort_file_size的大小现在以字节为单位，不是兆字节。
- 一些启动参数已经不推荐使用了（它们将运行到目前截止）：

```
record_buffer
sort_buffer
warnings
```

- 外部锁现在默认关闭了。
- 下面的SQL变量有了新的名字（旧的名字还可使用，但是不赞成使用）。

```
SQL_BIG_TABLES to BIG_TABLES
SQL_LOW_PRIORITY_UPDATES to LOW_PRIORITY_UPDATES
SQL_MAX_JOIN_SIZE to MAX_JOIN_SIZE
SQL_QUERY_CACHE_TYPE to QUERY_CACHE_TYPE
```

- SIGNED是一个保留字。
- DOUBLE和FLOAT类型的列不再忽略UNSIGNED的标记。
- 现在BIGINT列存储整数要比存储字符串更有效了。
- 因为STRCMP()函数在进行比较时，它使用当前的字符集设置，所以默认情况下是区分大小写字母的。
- TRUNCATE TABLE比DELETE FROM tablename快，这是因为它不返回删除的行数目。
- 如果其中一个变量是二进制串，那么LOCATE()和INSTR()函数是区分大小写的。
- 现在当传送一个字符串时，HEX()函数将每个字符转换成为两个十六进制数字。
- SHOW INDEX语句有两个额外的列：Null和Index_type。

- 当存在激活的锁时（从LOCK TABLES表中或从事务中），操作者将不能再运行TRUNCATE TABLE或DROP DATABASE语句，而会返回错误信息。
- ORDER BY column_name DESC子句将在所有的情况下先分类NULL值，而以前它并不这样做。
- 所有位运算符（<<、>>、|、&和~）的结果现在不分正负，如同当两个整数中的任一个无正负时，它们之间相减的结果（后者可以通过在MySQL启动时使用参数--sql-mode=NO_UNSIGNED_SUBTRACTION而禁止）。
- 如果操作者希望使用语句MATCH ... AGAINST (... IN BOOLEAN MODE)，那么他将不得不利用ALTER TABLE table_name TYPE=MyISAM重建表。如果表已经是MyISAM，这也适用。
- 在使用INSERT INTO ... SELECT类型的语句时，操作者需要指定一个IGNORE子句，否则MySQL将停止，并可能回退。
- RAND(seed)函数现在返回一个与以前不同的随机数字序列（为了进一步区分RAND(seed)和RAND(seed+1)）。
- SHOW OPEN TABLES的格式已经改变了。
- 旧的mysql_drop_db()、mysql_create_db()和mysql_connect() C API函数不再被支持。操作者可以使用CFLAGS=-DUSE_OLD_FUNCTIONS参数来编译MySQL，而不是更新客户端使用版本4.0的API。
- 如果操作者在使用Perl DBD::mysql模块，那么他将需要使用比1.2218更新的版本，这是由于旧版本使用旧的drop_db()函数。
- 操作者需要使用SET GLOBAL SQL_SLAVE_SKIP_COUNTER=#，而不是SET SQL_SLAVE_SKIP_COUNTER=#。
- 多线程客户端应该使用函数mysql_thread_init()和mysql_thread_end()。

小结

安装MySQL并不难。如果选择二进制版本的软件（在大多数情况下推荐这样做），那么在Windows中可以进行点击操作，在UNIX中可以执行一些简单的命令。选择源代码版本的软件是有根据的，如可能MySQL并没有操作者现有平台支持的可用软件，或者需要通过更加优化的编译，使MySQL发挥最大的作用。无论是什么原因，谨慎地选择编译参数，操作者就可以进行更快而且稳定的MySQL安装。

第16章 多台驱动器

数据库通常会越变越大，并且大量的数据在一个驱动器上，意味着驱动器和驱动控制器会成为严重的瓶颈。使用多个磁盘和多个控制器（使用RAID）是克服瓶颈问题的方法之一，并且大多数RAID种类具有额外的冗余功能。

符号链接是一种不使用RAID，跨多个磁盘拆分数据库或表的方法。通过创建从一台驱动器到另一台驱动器的链接，一台驱动器被过度使用的瓶颈问题得到缓解。

本章要点：

- 使用RAID
- 使用符号链接

了解RAID

RAID是指冗余廉价磁盘阵列（redundant array of inexpensive disks）。尽管术语独立性（independent）比较准确，但错误的消息来源在Internet上被复制的速度之快，令人很吃惊。这个术语出自研究员David Patterson、Garth Gibson和Randy Katz在1987年写的一篇论文。它提高了磁盘性能和容错能力。与RAID相对的说法是单一大型昂贵磁盘（single large expensive disk，SLED）。

RAID有许多不同的类型，以下各节将分别进行描述。所有的RAID类型都建立在三个基本概念之上：镜像（mirroring，是指在另一台驱动器上重复写操作）；条带化（striping，是指将数据分散在多台驱动器中）；奇偶校验（parity，是指为了防止错误和从错误中恢复数据，而对数据位进行检查）。

RAID 0

RAID 0（有时候被称为条带化（striping），因为它不是一种真正的RAID类型）是把数据拆分成数据块，然后分散到多个驱动器中。每个驱动器上块的大小是相同的，但可以根据不同情况，在开始的时候设置大小不同的块。由于其中一个主要瓶颈移动到了驱动器的磁头，这样可以提高磁盘的性能。因为在不同的驱动器上，所以RAID 0提高了能够同时访问数据的机会。这意味着，读取操作可以同时发生，而不必等待直到读完一个数据，再去读第二个数据集。通常，驱动器越多，性能越高。如果每个驱动器都有自己的控制器，性能会更好，尽管这个要求并不迫切。只要确认控制器在负责多个驱动器时仍能处理负荷就行了。RAID 0没有容错能力，实际上，它会提高发生错误的可能性，因为当一个以上的驱动器发生故障的时候，将导致数据集无法获得。

RAID 0不应该在数据的可用性非常关键的情况下使用。实现RAID 0至少需要两个驱动器。由于RAID 0不是真正的RAID类型，所以有时幽默地把它称为AID，因为这个缩写词中不包含冗余的形式（RAID中的字母R）。

图16.1展示了如何在3个驱动器上实现RAID 0，第一个块数据被写入驱动器A，第二块数据在驱动器B上，第三块数据在驱动器C上。然后，下一块数据又被写入第一个驱动器，以此类推。虽然第一和第三块数据存储在不同的设备上，但它们可以同时被读取。

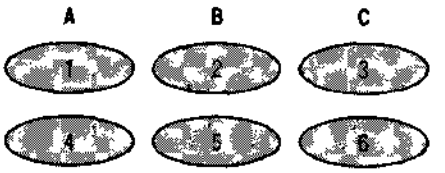

图16.1 RAID 0

RAID 1

RAID 1（也称镜像（mirroring））是把写入一个驱动器的数据再重复写入另一个驱动器。这样做提高了容错能力，因为万一一个驱动器出现了故障，还有一份最新的备份数据。驱动器故障是硬件故障类型中最常见的，RAID 1对防止出现这种故障起到了作用。由于写操作是同时发生的，所以写的性能很差，但读的操作可以稍微快一些，因为有多个驱动器可以访问。实现RAID 1至少需要两台驱动器。

图16.2显示了用两台驱动器实现RAID 1，每台驱动器包含同样的备份数据。

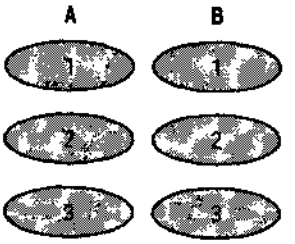

图16.2 RAID 1

RAID 2和RAID 3

RAID 2是很少用到的版本，它使用汉明错误修正代码（这是使用7位字中的3位用于错误校验和修正），并且主要用途是针对无法进行错误防护的驱动器（SCSI驱动器做错误防护）。

RAID 3与RAID 0一样，除了它也为纠正错误留出一个专用驱动器外，它提供了容错的一些级别。数据以字节为基准被条带化到多个驱动器上。另一个驱动器存储奇偶校验数据。奇偶校验在写操作过程中确定，在读操作过程中被检查。如果一块驱动器出现了故障，那么奇偶校验信息可以提供恢复操作。操作者需要至少3块驱动器来实现RAID 3。它通常需要硬件的支持才有用，小型数据的写和读操作速度快，但是大块数据通常需要数据从所有的驱动器读出，这意味着性能可能会像单个驱动器一样慢。

图16.3显示RAID 3在两个驱动器上（A和B）进行数据条带化，第三个驱动器用来存储奇偶校验。

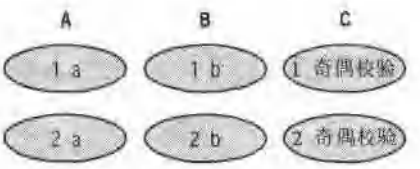

图16.3 RAID 3

RAID 4

RAID 4与RAID 3类似，除了条带化在块一级进行外，没有字节一级。对小于一个块的数据进行读操作将比较快（一般在增加新的驱动器后会更快）。像RAID 3一样，它至少需要3块驱动器，并且如果一个驱动器出现故障，校验数据可以提供恢复操作。奇偶校验驱动器可能会成为瓶颈，选择RAID 5可以克服这个限制。

图16.4显示RAID 4在3个驱动器上进行数据条带化（A、B和C），利用第4个驱动器（D）存储奇偶校验。

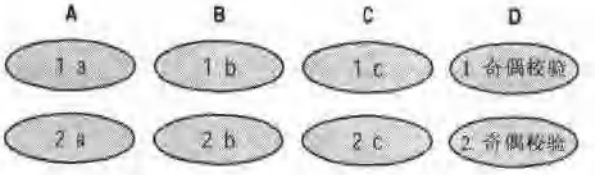

图16.4 RAID 4

RAID 5

RAID 5也进行条带化，也条带化错误纠正数据，改善性能和容错。除了奇偶校验数据存储在所有驱动器上之外，它与RAID 4类似。写操作比RAID 4更快（没有单个驱动器的瓶颈），但是读操作较慢，这是因为奇偶校验信息在所有驱动器上占据了空间，并且必须被跳过。RAID 5常常被推荐用于数据库服务器，这是由于建立起冗余性并改善了性能。

实现RAID 5需要至少3个驱动器。

图16.5显示RAID 5在3个驱动器上（A，B和C）对数据进行条带化，它们都包含奇偶校验数据。

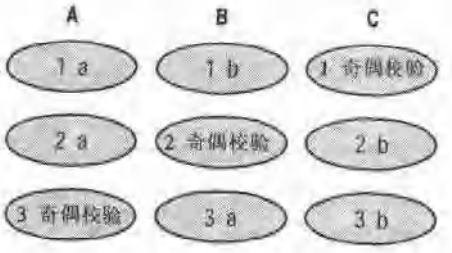

图16.5 RAID 5

RAID 10

RAID 10是RAID 1和RAID 0的混合（镜像和条带化）。它提供所有条带化的性能优点以及镜像的容错。这是两种情况下最好的办法，可就是成本很高。它需要至少4个驱动器才能实现。

图16.6显示了RAID 10在4个驱动器上（A、B、C和D）实现的情况。A和B镜像数据（数据块一到四在两个磁盘上都有），并且驱动器C和D利用数据在其上的条带化提供性能的提高。虽然成本较高（由于驱动器的数量较多），但由于RAID 10具有最高级的性能增强和容错，因此常常被建议用于数据库服务器上。

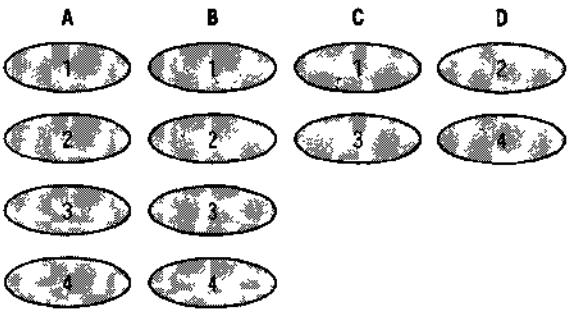

图16.6 RAID 10

RAID 0+1

RAID 0+1常常与RAID 10混淆。RAID 10是条带化的驱动器阵列，这些驱动器的段被镜像，RAID 0+1是一个镜像的驱动器阵列，这些驱动器的段被条带化。一般RAID 0+1在性能要求高于可靠性要求时选用，RAID 10在可靠性的要求高于性能的要求时选用。RAID 0+1也很贵，至少需要4个驱动器实现。图16.7显示了RAID 0+1在4个驱动器上实现的情况（A、B、C和D）。

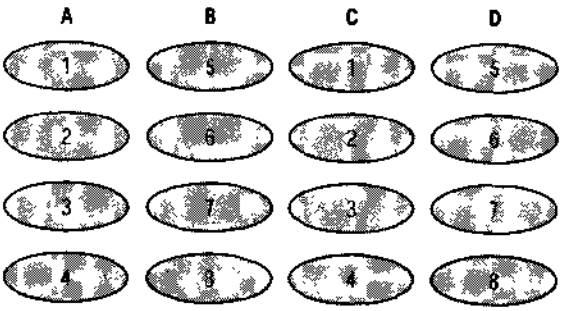

图16.7 RAID 0+1

其他类型的RAID

前面的RAID类型并不是所有可用的类型。还有其他的一些实际很少使用的类型。

操作者可以使用硬件或软件来实现RAID（hardware RAID和software RAID），或两

者的混合。MySQL的--with-raid-option是一个有限制的软件RAID形式（当前使用RAID 0）。其主要目的是克服文件空间的限制。以后的版本将扩展其用途。复制，虽然不像RAID讲得那样严格，但是它是MySQL软件与RAID类似（RAID 1）的另一个特性。一旦硬件RAID建立并执行，硬件设备将多个驱动器在软件系统前呈现为一个驱动器，并且处理所有的冗余和条带化的操作，使MySQL像通常一样操作，因此硬件RAID通常更容易。软件RAID使用软件（例如vinum，它是运行在FreeBSD上实现RAID 0、RAID 1和RAID 5的工具），其结果是利用中央处理单元（central processing unit，CPU）来实现。如果操作者的CPU有剩余的能力，那么软件RAID可能是一种好的选择。

使用符号链接

操作者可以利用符号链接（symlinks）很容易地改善数据库的性能，并减少磁盘延迟。方法是不在一块磁盘上存储所有的数据和索引，而是建立有一块磁盘到另一块磁盘的符号链接（symlink），这实际上是在存储数据。现在操作者可以只建立MyISAM符号链接（symlink）表和数据库。较早版本的MySQL具有基本符号链接（symlink）表的问题（当某些操作在它们上执行时，它们将返回到它们原来的位置），但是版本4已经处理了这些问题中的很大一部分。

数据库的符号链接

为了建立MyISAM数据库的符号链接，需要下面这些步骤：

1. 在新的位置建立数据库目录。
2. 确信许可权和所有权是正确的（700和mysql.mysql）。
3. 在指向新位置的数据目录上建立符号链接（symlink）。

我们可以测试一下新数据库s_db的建立，它将是要进行符号链接（symlink）的对象。首先在UNIX系统上，假设操作者已经有了一个目录disk2，它是准备放置数据库的第二块磁盘，按照下列步骤建立准备存储数据的目录（/disk2/mysql/data/s_db），并且改变许可权和所有权关系：

```
% cd /disk2
% mkdir mysql
% mkdir mysql/data
% mkdir mysql/data/s_db
% chown mysql /disk2/mysql/data/s_db/
% chgrp mysql /disk2/mysql/data/s_db/
% chmod 700 /disk2/mysql/data/s_db/
```

现在，回到数据目录中，建立符号链接（symlink）：

```
% cd /usr/local/mysql/data
% ln -s /disk2/mysql/data/s_db s_db
```

一旦完成，数据库将被建立起来（记住MyISAM数据库只是数据目录中的子目录）。操作者可以通过连接到MySQL对此进行验证：

```
% /usr/local/mysql/bin/mysql -uroot -pg00r002b
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6202 to server version: 4.0.2-alpha-max-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> SHOW DATABASES LIKE 's_db';
+-----------------+
| Database (s_db) |
+-----------------+
| s_db            |
+-----------------+
1 row in set (0.00 sec)
```

现在可以看到数据实际上被放在第二块磁盘上，按照下面的步骤，建立并填充一个小的表：

```
mysql> USE s_db
Database changed
mysql> CREATE TABLE s1( f1 INT);
Query OK, 0 rows affected (0.23 sec)
mysql> INSERT INTO s1 VALUES(1);
Query OK, 1 row affected (0.03 sec)
```

对第二块磁盘上建立的新的数据进行检查：

```
mysql> exit
Bye
% ls -l /disk2/mysql/data/s_db/
total 14
-rw-rw----  1 mysql  mysql     5 Jul  8 02:26 s1.MYD
-rw-rw----  1 mysql  mysql  1024 Jul  8 02:26 s1.MYI
-rw-rw----  1 mysql  mysql  8550 Jul  8 02:25 s1.frm
```

在Windows系统上为数据库建立一个符号链接（symlink）有许多不同。许可权相对简单，建立符号链接不是通过ln -s，操作者只是建立了一个具有扩展名.sym的文本文件。首先，在包含下列文本的数据目录中建立名为s2_db.sym的文件：

```
D:\s_db
```

由于这是在数据目录中，因此它将与其他的MyISAM数据库（这里有firstdb、mysql和test）出现在相同的层上。并且当操作者连接到MySQL时，操作者将会看到它就像一个现有数据库一样：

```
C:\mysql\bin>dir ..\data\
 ...
Directory of C:\mysql\data
09/03/2002  08:58p      <DIR>          .
09/03/2002  08:58p      <DIR>          ..
09/03/2002  08:31p      <DIR>          firstdb
```

```
09/03/2002  08:22p      <DIR>          mysql
09/03/2002  08:23p               4,342 mysql.err
09/03/2002  08:57p                   7 s2_db.sym
09/03/2002  08:22p      <DIR>          test
               2 File(s)          4,349 bytes
...
C:\mysql\bin>mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8 to server version: 4.0.3-beta-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| firstdb  |
| mysql    |
| s2_db    |
| test     |
+----------+
4 rows in set (0.00 sec)
mysql> USE s2_db
Database changed
mysql> CREATE TABLE s1(f1 INT);
Query OK, 0 rows affected (0.23 sec)
mysql> INSERT INTO s1 VALUES(1);
Query OK, 1 row affected (0.03 sec)
C:\mysql\bin>dir d:\s2_db
...
Directory of d:\s2_db
09/03/2002  09:38p      <DIR>          .
09/03/2002  09:38p      <DIR>          ..
09/03/2002  09:38p               8,550 s1.frm
09/03/2002  09:38p                   5 s1.MYD
09/03/2002  09:38p               1,024 s1.MYI
               3 File(s)          9,579 bytes
...
```

说明： 如果以前的代码不能运行了，那么操作者可以运行较早版本的MySQL（这时必须向配置文件my.ini中增加一行--use-symbolic-links）。还可能操作者的MySQL版本没有使用-DUSE_SYMDIR编译，这时符号链接（symlink）根本无法工作。通常，mysql-max和mysql-max-nt服务器是利用这个参数编译的。然而，操作者应该查看一下最新的文档。

表的符号链接

我们不建议在单个表上使用符号表，因为它几乎没有什么功能可以与符号链接表（symlinked）一起正常工作（请查看最新的文档，很可能这一点就要发生变化）。

不能与符号链接表一起工作的功能有：

- BACKUP TABLE和RESTORE TABLE（符号链接将会被丢失）。
- Mysqldump在切断电源的时候，不存储符号链接的信息。
- ALTER TABLE（它忽略INDEX/DATA DIRECTORY="path" CREATE TABLE的参数）。

在创建表的时候为了做符号链接，应该使用INDEX或DATA DIRECTORY PATH参数。参数DATA DIRECTORY为.MYD文件创建一个符号链接，INDEX DIRECTORY放置一个.MYI文件。下面这个例子把新表的数据文件放在你先前创建的新目录下的firstdb数据库中。

```
mysql> USE firstdb;
Database changed
mysql> CREATE TABLE s_table (a int) DATA DIRECTORY =
 '/disk2/mysql/data/s_db';
Query OK, 0 rows affected (0.20 sec)
mysql> INSERT INTO s_table VALUES(1);
Query OK, 1 row affected (0.01 sec)
```

可以看到.frm文件（包含结构）是在普通的数据目录下，并且.MYD数据文件是在新的位置。

```
% cd /usr/local/mysql/data/firstdb/
% ls -l /disk2/mysql/data/s_db/
total 16
-rw-rw----  1 mysql  mysql     5 Jul  8 02:26 s1.MYD
-rw-rw----  1 mysql  mysql  1024 Jul  8 02:26 s1.MYI
-rw-rw----  1 mysql  mysql  8550 Jul  8 02:25 s1.frm
-rw-rw----  1 mysql  mysql     5 Jul  8 05:35 s_table.MYD
% ls -l s*
lrwxrwx--x  1 mysql  mysql    34 Jul  8 05:34 s_table.MYD ->
 /disk2/mysql/data/s_db/s_table.MYD
-rw-rw----  1 mysql  mysql  1024 Jul  8 05:35 s_table.MYI
-rw-rw----  1 mysql  mysql  8548 Jul  8 05:34 s_table.frm
```

MySQL已经为表创建了符号链接。你也可以自己创建这个符号链接（用在创建数据库符号链接时，同样方式处理权限）。

像这个例子显示的那样，可以在不同的地方生成数据和索引文件。假设存在/disk3/mysql/data/indexes目录：

```
mysql> CREATE TABLE s2_table (a int) DATA DIRECTORY =
 '/disk2/mysql/data/s_db' INDEX DIRECTORY =
 '/disk3/mysql/data/indexes';
Query OK, 0 rows affected (0.04 sec)
```

查看这些处于其新位置的文件：

```
% ls -l /disk3/mysql/data/indexes/
total 2
```

```
-rw-rw----  1 mysql  mysql  1024 Jul  8 06:01 s2_table.MYI
% ls -l /disk2/mysql/data/s_db/
...
-rw-rw----  1 mysql  mysql  0 Jul  8 06:01 s2_table.MYD
...
% ls -l /usr/local/mysql/data/firstdb/
...
lrwxrwx--x  1 mysql  mysql    35 Jul  8 06:01 s2_table.MYD ->
 /disk2/mysql/data/s_db/s2_table.MYD
lrwxrwx--x  1 mysql  mysql    38 Jul  8 06:01 s2_table.MYI ->
 /disk3/mysql/data/indexes/s2_table.MYI
-rw-rw----  1 mysql  mysql  8548 Jul  8 06:01 s2_table.frm
...
```

说明： INDEX DIRECTORY和DATA DIRECTORY参数在运行Windows上的MySQL时不工作（尽管已查看过最新的文档）。

小结

RAID是利用多驱动器存储数据的一种方法。RAID 0（条带化）将数据块分布在多个磁盘上。它虽然提高了性能，但是并没有冗余能力。RAID 1（镜像）降低了写操作的速度，提高了读操作的速度，但是它主要用于避免驱动器的故障。RAID 2、3、4和RAID 5都使用条带化，并使用各种奇偶校验形式实现冗余。RAID 10和RAID 0+1混合镜像和条带化（虽然它们执行起来有些不同——RAID 10更重视可靠性，RAID 0+1更重视速度）。

符号链接使操作者能够将MyISAM数据库或表放在与常用数据目录不同的地方，通常会是一个不同的驱动器。

附　　录

附录A　MySQL语法参考

该附录包含MySQL 4.0版本中使用的SQL指令和语法。对于更新的版本，请查看随产品带来的参考资料，或访问MySQL网站（www.mysql.com）。

附录中通用的规定如下：

- 方括弧（[]）表示某些可选项。例如：

```
SELECT expression [FROM table_name [WHERE where_clause]]
```

表明表达式是必须选的（例如SELECT 42/10），而WHERE从句是可选的，但只有FROM table_name选项存在时，才会选（可以有SELECT * FROM t1，但不能有SELECT * WHERE f1>10 x，因为table_name从句缺失了）。

- 垂直线分隔开可供选择的选项。例如：

```
CREATE [UNIQUE | FULLTEXT] INDEX
```

表示UNIQUE和FULLTEXT是单独的选项。

- 花括弧（{}）表示某个选项必须被选中。例如：

```
CREATE TABLE ... [TYPE = {BDB | HEAP | ISAM | InnoDB | MERGE |
 MRG_MYISAM | MYISAM }]
```

如果选项TYPE从句被指定，BDB、HEAP、ISAM、InnoDB、MERGE、MRG_MYISAM或MYISAM中的某个选项也必须被指定。

- 三个点（...）表明选项能被重复。例如：

```
SELECT expression,...
```

表明表达式能被重复（用逗号分开），如SELECT f1, f2, f3。

ALTER

ALTER语法如下：

```
ALTER [IGNORE] TABLE table_name alter_specification [, alter_specification ...]
```

alter_specification语法可以为下列中的任何一个：

```
ADD [COLUMN] create_definition [FIRST | AFTER field_name ]
ADD [COLUMN] (create_definition, create_definition,...)
ADD INDEX [index_name] (index_field_name,...)
ADD PRIMARY KEY (index_field_name,...)
ADD UNIQUE [index_name] (index_field_name,...)
ADD FULLTEXT [index_name] (index_field_name,...)
ADD [CONSTRAINT symbol] FOREIGN KEY index_name
 (index_field_name,...)[reference_definition]
ALTER [COLUMN] field_name {SET DEFAULT literal | DROP DEFAULT}
CHANGE [COLUMN] old_field_name create_definition [FIRST | AFTER field_name]
MODIFY [COLUMN] create_definition [FIRST | AFTER field_name]
DROP [COLUMN] field_name
DROP PRIMARY KEY
DROP INDEX index_name
DISABLE KEYS
ENABLE KEYSALTER TABLE
RENAME [TO] new_table_name
ORDER BY field_name
   table_options
```

ALTER TABLE允许改变已经存在的表的结构。你能够添加（ADD）列，变更（CHANGE）列名和定义，在不改名的情况下修改（MODIFY）（非ANSI Oracle扩展名）列的定义，中止（DROP）列或索引，重命名（RENAME）表，排序（ORDER）数据和使用（DISABLE）或禁用（ENABLE）索引。

非ANSI MySQL扩展名是ALTER TABLE能在一条指令中包含多个命令（CHANGE、ADD，等等）。

为了使用ALTER TABLE，你需要有针对表格的ALTER、INSERT和CREATE特权。

IGNORE（非ANSI扩展名）使MySQL删除那些会导致完全相同的主键或惟一键的记录。通常，MySQL将终止运行，ALTER将失败。

FIRST和ADD...AFTER允许你在定义中指定加入字段的位置。

ANALYZE TABLE

```
ANALYZE TABLE table_name [,table_name...]
```

对于MyISAM和BDB表，该命令分析和储存指定表的键分布。在运行期间，它用只读锁锁定表。

BACKUP TABLE

```
BACKUP TABLE table_name [,table_name...] TO 'path_name'
```

对于MyISAM表，该指令拷贝数据和数据定义到备份目录中。

BEGIN

```
BEGIN
```

BEGIN指令开始一次事务或语句集。事务将保持开放，直到遇见下一条COMMIT或ROLLBACK指令。

CHECK TABLE

```
CHECK TABLE tbl_name[,tbl_name...] [option [option...]]
```

选项可以是下面中的某一个：

```
CHANGED
EXTENDED
FAST
MEDIUM
QUICK
```

该指令检查MyISAM或InnoDB表中的错误，并且更新MyISAM表的索引统计记录。QUICK选项不会为了检查链接而扫描行。FAST选项只检查没有正确关闭的表。CHANGED选项与FAST相同，但它还会对上次检查之后有变化的表进行检查。MEDIUM选项校验被删除的链接是否正确，EXTENDED选项对每行中的每个键进行一次完全的索引检查。

COMMIT

```
COMMIT
```

COMMIT指令结束事务或指令集，并刷新结果到磁盘。

CREATE

CREATE语法可以是下面中的一条：

```
CREATE DATABASE [IF NOT EXISTS] database_name
CREATE [UNIQUE|FULLTEXT] INDEX index_name ON table_name (field_name[(length)],... )
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] table_name [(create_definition,...)]
 [table_options] [select_statement]
```

create_definition语法可以是下面中的任何一条：

```
field_name type [NOT NULL | NULL] [DEFAULT default_value]
   [AUTO_INCREMENT] [PRIMARY KEY] [reference_definition]
PRIMARY KEY (index_field_name,...)
```

```
KEY [index_name] (index_field_name,...)
INDEX [index_name] (index_field_name,...)
UNIQUE [INDEX] [index_name] (index_field_name,...)
FULLTEXT [INDEX] [index_name] (index_field_name,...)
[CONSTRAINT symbol] FOREIGN KEY [index_name] (index_field_name,...)
 [reference_definition]
CHECK (expr)
```

type语法可以是下面中的任何一条：

```
TINYINT[(length)] [UNSIGNED] [ZEROFILL]
SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
INT[(length)] [UNSIGNED] [ZEROFILL]
INTEGER[(length)] [UNSIGNED] [ZEROFILL]
BIGINT[(length)] [UNSIGNED] [ZEROFILL]
REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]
NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]
CHAR(length) [BINARY]
VARCHAR(length) [BINARY]
DATE
TIME
TIMESTAMP
DATETIME
TINYBLOB
BLOB
MEDIUMBLOB
LONGBLOB
TINYTEXT
TEXT
MEDIUMTEXT
LONGTEXT
ENUM(value1,value2,value3,...)
SET(value1,value2,value3,...)
```

index_field_name可以是下面中的一条：

```
field_name [(length)]
```

reference_definition可以是下面中的一条：

```
REFERENCES table_name [(index_field_name,...)] [MATCH FULL
  | MATCH PARTIAL] [ON DELETE reference_option] [ON UPDATE reference_option]
```

reference_option语法可能是下面中的一条：

```
RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
```

table_options语法可能是下面中的任何一条：

```
TYPE = {BDB | HEAP | ISAM | InnoDB | MERGE | MRG_MYISAM | MYISAM }
AUTO_INCREMENT = #
AVG_ROW_LENGTH = #
CHECKSUM = {0 | 1}
COMMENT = "string"
MAX_ROWS = #
MIN_ROWS = #
PACK_KEYS = {0 | 1 | DEFAULT}
PASSWORD = "string"
DELAY_KEY_WRITE = {0 | 1}
ROW_FORMAT= { default | dynamic | fixed | compressed }
RAID_TYPE= {1 | STRIPED | RAID0 } RAID_CHUNKS=#  RAID_CHUNKSIZE=#
UNION = (table_name,[table_name...])
INSERT_METHOD= {NO | FIRST | LAST }
DATA DIRECTORY="absolute_path_to_directory"
INDEX DIRECTORY="absolute_path_to_directory"
```

select_statement语法如下：

```
[IGNORE | REPLACE] SELECT ...  (select statement)
```

CREATE指令创建数据库、表或索引。

除非使用IF NOT EXISTS从句，否则如果数据库或表已经存在，则MySQL返回错误。

只要连接激活，TEMPORARY表就存在。这里需要有CREATE TEMPORARY TABLES权限。

字段定义的默认值是NULL。数字字段的默认值是0（除非有AUTO_INCREMENT参数），而字符串字段的默认值为空字符串（除了ENUM字段，其默认值为第一个选项）。日期和时间字段的默认值为0。

默认情况下，AUTO_INCREMENT字段从1开始计数，每次新添一个记录，就会增加1。

KEY和INDEX在上下文中是同义字。

PRIMARY KEY指定索引不能相同，字段（或字段组合）必须被指定为NOT NULL。

UNIQUE指定索引不能相同。

RAID_TYPE选项帮助不支持大型文件的操作系统克服文件大小的限制。STRIPED选项是当前惟一被使用的。对MyISAM表，它在数据库内部目录中创建子目录，每个目录中包含数据文件的一部份。第一个1024 * RAID_CHUNKSIZE字节放入第一部分，下一个1024 * RAID_CHUNKSIZE字节放入下一部分，以此类推。

DATA DIRECTORY="directory"和INDEX DIRECTORY="directory"选项指定存储数据或索引文件的绝对路径。

PACK_KEYS=1选项为MyISAM表压缩索引中的数值字段（字符串也如此，这是默认的做法）。该选项只有在索引中有很多相同的数字时，才会有用。

使用AVG_ROW_LENGTH是为了让MySQL知道表中行的平均长度。这只有在表很大且记录大小经常变化的情况下才有用。

如果想对所有行保留校验和，将MyISAM表中的CHECKSUM设置为1。这样做对于修复崩溃的表比较容易，但会减慢表的运行速度。

COMMENT允许写出多达60个字节的表的注释。

MAX_ROWS和MIN_ROWS分别指明准备存入表中的最大和最小行。

PASSWORD将用密码加密数据定义文件（.frm）。

DELAY_KEY_WRITE使MySQL在更新索引之前处于等待状态，直到MyISAM表被关闭，这样做可以加快运行大量UPDATE和INSERT操作的速度。

ROW_FORMAT指明MyISAM表是否将为FIXED或DYNAMIC状态。

DELETE

DELETE语法可以是下面中的任何一个：

```
DELETE [LOW_PRIORITY | QUICK] FROM table_name [WHERE
   where_clause] [ORDER BY ...] [LIMIT rows]
DELETE [LOW_PRIORITY | QUICK] table_name[.*]
   [,table_name[.*] ...] FROM table-references [WHERE where_clause]
DELETE [LOW_PRIORITY | QUICK] FROM table[.*], [table[.*]
   ...] USING table-references [WHERE where_clause]
```

DELETE指令删除表中与where_clause有关联的记录（如果没有从句，就是所有记录）。

LOW PRIORITY关键字使DELETE处于等待状态，直到没有其他客户读取表才继续执行。

QUICK关键字使MySQL在DELETE期间不合并索引，这使得运行速度有时会快一点。

LIMIT决定被删除记录的最大数值。

ORDER BY从句使得MySQL按一定顺序删除记录（在LIMIT从句中有用）。

DESC

DESC是DESCRIBE的同义词。

DESCRIBE

```
DESCRIBE table_name {field_name | wildcard}
```

DESCRIBE返回指定表和字段的定义（与SHOW COLUMNS FROM table_name相同）。

通配符可以是字段名的一部份，也可以是代表许多字符的百分号（%），或是代表一个字符的下划线（_）。

DO

DO语法如下：

```
DO expression, [expression, ...]
```

DO与SELECT有相同的作用，但它不返回结果（加快它的运行速度）。

DROP

DROP语法如下：

```
DROP DATABASE [IF EXISTS] database_name
DROP TABLE [IF EXISTS] table_name [, table_name,...] [RESTRICT | CASCADE]
DROP INDEX index_name ON table_name
```

DROP DATABASE删除数据库及其所有的表。

DROP TABLE删除指定的表。

DROP INDEX删除所有指定的索引。

除非使用IF EXISTS从句，否则MySQL在数据库不存在时将返回错误。

DROP TABLE自动提交激活的事务。

RESTRICT和CASCADE目前不能被执行。

EXPLAIN

```
EXPLAIN table_name
EXPLAIN select_query
```

select_query与在SELECT描述中指定的相同。

使用带有文件名参数的EXPLAIN，与使用DESCRIBE table_name相同。使用带有查询语句的参数EXPLAIN可以提供有关查询执行情况的反馈，这有利于优化查询和很好地利用关联的索引。

FLUSH

```
FLUSH flush_option [,flush_option] ...
```

flush_option可以为下面中的任何一种：

```
DES_KEY_FILE
HOSTS
LOGS
```

```
QUERY CACHE
PRIVILEGES
STATUS
TABLES
[TABLE | TABLES] table_name [,table_name...]
TABLES WITH READ LOCK
USER_RESOURCES
```

刷新DES_KEY_FILE将重新加载DES键。使用选项HOSTS，主机的缓存会被清空（例如，改变IP地址之后）。刷新LOGS会关闭和重新打开日志文件，并增加二进制日志。刷新QUERY CACHE会整理查询缓存的碎片。刷新PRIVILEGES将从mysql数据库中重新加载权限表。刷新STATUS将重设状态变量。刷新TABLES与刷新QUERY CACHE相同，但它也关闭所有打开的表。可以指定要被刷新的表。为了进行复制，可以在表上设置READ LOCK，这对于锁住一组表是有用的。刷新USER_RESOURCES将清除用户资源（被用于受限的查询操作、连接和每小时一次的更新）。

GRANT

```
GRANT privilege_type [(field_list)] [, privilege_type [(field_list)]
 ...] ON {table_name | * | *.* | database_name.*} TO user_name
 [IDENTIFIED BY [PASSWORD] 'password'] [, user_name [IDENTIFIED BY
 'password'] ...] [REQUIRE NONE | [{SSL| X509}] [CIPHER cipher [AND]]
 [ISSUER issuer [AND]] [SUBJECT subject]] [WITH [GRANT OPTICN |
 MAX_QUERIES_PER_HOUR # | MAX_UPDATES_PER_HOUR # |
 MAX_CONNECTIONS_PER_HOUR #]]
```

GRANT授予用户一种特别的权限或特权。表A.1描述了可用的特权。

表A.1 特权

特权	描述
ALL	授予所有基本权限
ALL PRIVILEGES	与ALL相同
ALTER	改变表结构的权限（ALTER语句），不包括索引
CREATE	创建数据库和表的权限，不包括索引
CREATE TEMPORARY TABLES	创建临时表的权限
DELETE	删除表中记录的权限（DELETE语句）
DROP	中断数据库和表的权限，不包括索引
EXECUTE	运行被保存的程序的权限（为MySQL 5预定的）
FILE	在服务器上读写文件的权限（对LOAD DATA INFILE或SELECT INTO OUTFILE语句）。任何MySQL用户能读的文件都是可读的

（续表）

特权	描述
INDEX	创建、修改或中断索引的权限
INSERT	往表中添加新记录的权限（INSERT语句）
LOCK TABLES	锁定表，使用户有SELECT权限
PROCESS	查看当前MySQL进程或取消MySQL进程的权限（对SHOW PROCESSLIST或KILL SQL语句）
REFERENCES	目前MySQL不再使用，是为兼容ANSL SQL提供的（它用于外部键）
RELOAD	重新加载数据库的权限（FLUSH语句或mysqladmin的重新加载、更新或刷新）
REPLICATION CLIENT	询问复制从属服务器和主服务器的权限
SHOW DATABASES	查看所有数据库的权限
SELECT	从表中返回数据的权限（SELECT语句）
SHUTDOWN	关闭服务器的权限
SUPER	即使达到了最大连接次数和执行了CHANGE MASTER、KILL线程、mysqladmin调试、PURGE MASTER LOGS和SET GLOBAL命令，也可以进行连接的权限
UPDATE	修改表中数据的权限（UPDATE语句）
USAGE	连接服务器和执行所有可用语句的权限（对于早期的MySQL 4版本，包括SHOW DATABASES）

INSERT

INSERT语法可以是下面中的任何一种：

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE] [INTO] table_name
   [(field_name,...)] VALUES ((expression | DEFAULT),...),(...),...
INSERT [LOW_PRIORITY | DELAYED] [IGNORE] [INTO] table_name
   [(field_name,...)] SELECT ...
INSERT [LOW_PRIORITY | DELAYED] [IGNORE] [INTO] table_name
   SET field_name=(expression | DEFAULT), ...
INSERT [LOW_PRIORITY] [IGNORE] [INTO] table_name [(field list)] SELECT ...
```

INSERT在表中插入新的行。没有初始字段列表时，假设字段与它们定义时的顺序相同。每个字段都必须被赋值。任何没有明确设置的列，都被设为默认值。

LOW PRIORITY关键字使INSERT处于等待状态，直到没有客户读取表时，才开始操作。使用DELAYED关键字，MySQL会释放客户程序，而不终止INSERT语句。

IGNORE使MySQL忽略引起主键或惟一键重复的INSERT操作，而不会终止INSERT语句。

INSERT...SELECT允许在一个或多个表中存在的行里插入一个表。

JOIN

MySQL接受下面任何的join语法：

```
table_name, table_name
table_name [CROSS] JOIN table_name
table_name INNER JOIN table_name condition
table_name STRAIGHT_JOIN table_name
table_name LEFT [OUTER] JOIN table_name condition
table_name LEFT [OUTER] JOIN table_name
table_name NATURAL [LEFT [OUTER]] JOIN table_name
table_name LEFT OUTER JOIN table_name ON conditional_expr
table_name RIGHT [OUTER] JOIN table_name condition
table_name RIGHT [OUTER] JOIN table_name
table_name NATURAL [RIGHT [OUTER]] JOIN table_name
```

表可以是table_name，使用别名（用AS），或者指明或忽略索引（用USE/IGNORE索引）。

condition语法如下：

```
ON conditional_expr | USING (field_names)
```

conditional_expr与WHERE从句中的内容相同。

KILL

```
KILL thread_id
```

删除指定的线程。可以使用SHOW PROCESSLIST去识别线程的ID。SUPER特权被用来删除不被当前连接所拥有的进程。

LOAD DATA INFILE

LOAD DATA INFILE语法如下：

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE
   'file.txt' [REPLACE | IGNORE] INTO TABLE table_name
   [FIELDS [TERMINATED BY '\t'] [[OPTIONALLY] ENCLOSED BY
   ''] [ESCAPED BY '\\' ]] [LINES TERMINATED BY '\n']
   [IGNORE number LINES] [(field_name,...)]
```

LOAD DATA从文本文件中读取数据并把它加入表中。比起使用INSERT语句，这是一种更快的添加大量数据的方法。

LOCAL关键字说明文件在客户机上；否则，文件被认为在数据库服务器上。如果服务器启动时使用了--local-infile=0选项，或者客户机不支持，LOCAL将不会起作用。

服务器上的文件必须都是可读的或在数据库目录中，为了在服务器上对文件使用LOAD DATA，需要FILE权限。

在服务器上，如果没有指出路径，则文件被认为是在当前数据库的数据库目录中。如果是相对路径，则被认为是相对的数据目录。也可以使用绝对路径。

LOW PRIORITY关键字使INSERT处于等待状态，直到没有客户读取表时，才开始操作。

CONCURRENT关键字允许其他线程在LOAD DATA运行时访问MyISAM表（这会减慢LOAD DATA的速度）。

REPLACE关键字会使MySQL删除和替代与新记录有相同主键或惟一键的旧记录。IGNORE使MySQL继续执行下一条记录。

如果指定了FIELDS从句，则至少要求使用TERMINATED BY、[OPTIONALLY] ENCLOSED BY和ESCAPED BY中的一个。如果FIELDS从句没被指定，默认为FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'。这些从句指定字段末端的字符（默认为tab键）、包含字段的字符（默认为没有）和esc符（默认是反斜线）。当使用Windows路径时，注意正确的退出路径。

若没有LINES从句，默认值被认为是LINES TERMINATED BY '\n'。它指定了记录末端的字符（默认为换行）。

IGNORE number LINES选项忽略文件前面的行（当文件含有标题时，这很有用）。

LOAD DATA INFILE是SELECT...INTO INFILE的补充。

LOCK TABLES

```
LOCK TABLES table_name [AS alias] {READ | [READ LOCAL] | [LOW_PRIORITY]
 WRITE} [,table_name {READ | [LOW_PRIORITY] WRITE} ...]
```

LOCK TABLES给指定的表上锁。锁可以是READ（其他连接不能写入，只能读取）、READ LOCAL（与READ相同，但只要不冲突其他连接也可以写入）或WRITE（这阻止其他连接的读或写）。如果WRITE锁是LOW PRIORITY（优先级低），READ锁先被设置。通常，WRITE锁有较高的优先级。

OPTIMIZE

```
OPTIMIZE TABLE table_name [,table_name]...
```

对MyISAM表，该语句对索引进行分类、更新统计和整理数据文件的碎片。

对于BDB表，该语句与ANALYZE TABLE相同。

该语句在操作期间锁定表（这会花些时间）。

RENAME

RENAME语法如下：

```
RENAME TABLE table_name TO new_table_name[, table_name2 TO new_table_name2,...]
```

RENAME允许给表（或表的清单）一个新名字。也能通过指定database_name.table_name，把表移到一个新的数据库中，只要该数据库在同一磁盘上。

在旧的表上需要ALTER和DROP权限，在新表上需要CREATE和INSERT权限。

REPAIR TABLE

```
REPAIR TABLE table_name [,table_name...] [EXTENDED] [QUICK] [USE_FRM]
```

用来修复一个崩溃的MyISAM表。如果用QUICK选项，只有索引树被修复。如果用EXTENDED，索引被逐行重新创建。用USE_FRM，索引将根据数据定义文件被修复（适于索引丢失或完全崩溃的时候）。

REPLACE

REPLACE语法可以为下面中的某一个：

```
REPLACE [LOW_PRIORITY | DELAYED] [INTO] table_name
   [(field_name,...)] VALUES (expression,...),(...),...
REPLACE [LOW_PRIORITY | DELAYED] [INTO] table_name [(field_name,...)] SELECT ...
REPLACE [LOW_PRIORITY | DELAYED] [INTO] table_name SET
   field_name=expression, field_name=expression, ...
```

REPLACE很像INSERT，但当MySQL遇到带有主键或惟一键的记录已经存在时，它将删除和替代记录。

RESET

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

reset_option可以为下面中的任何一个：

```
MASTER
QUERY CACHE
SLAVE
```

RESET MASTER删除所有二进制日志和清空二进制文件的索引。RESET SLAVE重新设置从属服务器的位置，这是为了与主控服务器之间进行复制。RESET QUERY CACHE清空查询缓存。

RESTORE TABLE

```
RESTORE TABLE table_name [,table_name...] FROM 'path'
```

恢复用BACKUP TABLE备份的表。它不会覆盖已有的表。

REVOKE

```
REVOKE privilege_type [(field_list)] [,privilege_type [(field_list)]
 ...] ON {table_name | * | *.* | database_name.*} FROM user_name
 [, user_name ...]
```

取消授予指定用户的特权。privilege_type可以是GRANT清单中列出的任何特权。

ROLLBACK

```
ROLLBACK
```

ROLLBACK语句终止一个事务或指令集，撤销事务中的所有指令。

SELECT

SELECT语法如下：

```
SELECT [STRAIGHT_JOIN] [SQL_SMALL_RESULT] [SQL_BIG_RESULT]
   [SQL_BUFFER_RESULT] [SQL_CACHE | SQL_NO_CACHE]
   [SQL_CALC_FOUND_ROWS] [HIGH_PRIORITY] [DISTINCT |
   DISTINCTROW | ALL] expression, ... [INTO {OUTFILE |
   DUMPFILE} 'file_name' export_options]
[FROM table_names
[WHERE where_clause] [GROUP BY {unsigned_integer |
   field_name | formula} [ASC | DESC], ... [HAVING
   where_definition] [ORDER BY {unsigned_integer |
   field_name | formula} [ASC | DESC], ...] [LIMIT
   [offset,] rows] [PROCEDURE procedure_name] [FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT指令从表中返回数据。expression通常是一个字段列表（如果要求的话，要带上一个函数），但也可以是与表的字段无关的计算和函数。例如：

```
SELECT VERSION();
```

或如下：

```
SELECT 42/10;
```

字段可以被指定为field_name、table_name.field_name或database_name.table_name.field_name。如果名字不明确的话，就需要更长的名字。

表达式可以用关键字AS给出别名。例如：

```
SELECT 22/7 AS about_pi
```

表达式可以用在语句的任何地方（但不能用于WHERE从句，这通常是先决定的）。

table_names从句用于查询操作中使用的表的由逗号分隔的清单。也可以给表名起别名。例如：

```
SELECT watts FROM wind_water_solar_power AS n;
```

如果不满意MySQL的选择，可以用表名后的USE INDEX和IGNORE INDEX从句，控制MySQL索引的用法（通过使用EXPLAIN）。语法如下：

```
table_name [[AS] alias] [USE INDEX £®indexlist)] [IGNORE INDEX £®indexlist)]
```

ORDER BY从句对返回结果按升序（默认或用ASC关键字）或降序（DESC关键字）排序。在表达式中，不必使用这些项目。例如：

```
SELECT team_name FROM results ORDER BY points DESC
```

为了被返回，WHERE从句由行关联所需的条件组成（可以包含函数）：

```
SELECT team_name FROM results WHERE points > 10
```

GROUP BY将输出行合成一组，当使用集合函数时，这个语句很有用。GROUP BY两个可用的非ANSI MySQL扩展是ASC或DESC，而且还可以用GROUP BY从句中没提到的表达式中的字段。例如：

```
SELECT team_name, team_address, SUM(points) FROM teams GROUP BY team_name DESC
```

HAVING从句也是一个条件，但它被最后应用，因此它能应用于分组项。例如：

```
SELECT team_name, SUM(points) FROM teams GROUP BY team_name HAVING SUM(points) > 20
```

不要用它代替WHERE从句，因为它将会降低查询速度。

DISTINCT和它的同义字DISTINCTROW指出返回的行应该是惟一的。ALL（默认）返回所有的行，无论是否惟一。

HIGH_PRIORITY（非ANSI MySQL扩展）给予SELECT比任何更新操作更高的权限。

SQL_BIG_RESULT和SQL_SMALL_RESULT（非ANSI MySQL扩展）支持MySQL的优化器，它让优化器在开始工作前知道返回结果是大还是小。两者都与GROUP BY和DISTINCT从句一起使用，且通常导致MySQL为更快的速度而使用临时表。

SQL_BUFFER_RESULT（ANSI MySQL扩展）使MySQL把结果放入临时表。

LIMIT采用一或二个自变量去限制返回行的数目。如果有一个自变量，它是返回行的最大数目；如果有两个自变量，第一个是偏离量，第二个是返回行的最大数目。如果第二个自变量是－1，MySQL将从指定的偏移量中返回所有的行，直到结尾。例如，为返回行2到结尾，使用如下：

```
SELECT f1 FROM t1 LIMIT 1,-1
```

SQL_CALC_FOUND_ROWS从句使MySQL计算返回行的数目，条件是LIMIT从句不存在。该数字能用SELECT FOUND_ROWS()函数返回。

SQL_CACHE使MySQL把结果存入询问缓存器中，SQL_NO_CACHE从句则相反。这两者都是非ANSI MySQL扩展。

STRAIGHT_JOIN（非ANSI MySQL扩展）使优化器按表在FROM从句中的顺序与表连接，这样，如果表是按非优化方式连接的，可以加快询问的速度（用EXPLAIN去检查这些）。

SELECT...INTO OUTFILE 'file_name'把结果写入服务器上的新文件（对每个人都可读）。做这一步，需要FILE权限。它是LOAD DATA INFILE的补充，并用相同的选项。

使用INTO DUMPFILE使MySQL在文件中写入一行，不含任何列或行终止符，及换码符。

对InnoDB和BDB表，FOR UPDATE从句把锁写入行中。

SET

```
SET [GLOBAL | SESSION] variable_name=expression, [[GLOBAL | SESSION |
 LOCAL ] variable_name=expression...]
```

SET可以设置变量值。SESSION（或LOCAL，同义词）是默认值，它为当前连接的持续时间设置值。GLOBAL需要SUPER特权，它为所有的新连接设置变量，直到服务器重新启动。为了使该选项保持激活状态，在服务器启动后，还需要在配置文件中设置它。可以用SHOW VARIABLES发现变量名的完全清单。表A.2描述了用非标准方式设置的变量。

表A.2　用非标准方式设置的变量

语法	描述
AUTOCOMMIT= 0 | 1	当设置为1时，MySQL自动执行COMMIT指令，除非在BEGIN和COMMIT语句中屏蔽了该变量。当设置为AUTOCOMMIT时，MySQL也对所有开放的事务执行COMMIT
BIG_TABLES = 0 | 1	当设置为1时，所有临时表都存在磁盘里，而不是内存里。这使得临时表变慢，但可以防止内存用尽。默认是0
INSERT_ID = #	设置AUTO_INCREMENT值（因此下一个使用AUTO_INCREMENT字段的INSERT语句将会使用这个值）
LAST_INSERT_ID = #	设置从下一个LAST_INSERT_ID()函数返回的值
LOW_PRIORITY_UPDATES = 0 | 1	当设为1时，所有更新语句（INSERT、UPDATE、DELETE、LOCK TABLE WRITE）等到没有要求执行的读取命令（SELECT、LOCK TABLE READ）后，才处理表

（续表）

语法	描述
MAX_JOIN_SIZE = value | DEFAULT	通过设置行的最大值，可以阻止MySQL运行那些不能恰当利用索引的询问，或在负载大或高峰期时，有潜在减慢服务器速度可能的询问。除DEFAULT重设为SQL_BIG_SELECTS外，设置可以是任何值。如果SQL_ BIG_SELECTS被设置，MAX_JOIN_SIZE会被忽略。如果询问已经被存入缓存存储器， MySQL将忽略这一限制并返回结果
QUERY_CACHE_TYPE = OFF | ON | DEMAND	设置询问缓存存储器设置为线程
QUERY_CACHE_TYPE = 0 | 1 | 2	设置询问缓存存储器设置为线程
SQL_AUTO_IS_NULL = 0 | 1	如果设置（1为默认值），那么最后一个为AUTO_INCREMENT插入的行可以用WHERE auto_increment_column IS NULL发现。这被用于Microsoft Access和其他通过ODBC连接的程序
SQL_BIG_SELECTS = 0 | 1	如果设置（1为默认值），则MySQL允许大的询问。如果不设置（0），则MySQL将不允许检查在多于max_join_size的行中的询问。这有利于避免运行导致服务器关闭的意外或恶意询问
SQL_BUFFER_RESULT = 0 | 1	如果设置（1），MySQL把询问结果放入临时表中（在一些情况下，提前释放表锁可以加快执行速度）
SQL_LOG_OFF = 0 | 1	如果设置（1），MySQL将不对客户机做日志（这不是更新日志）。需要SUPER权限
SQL_LOG_UPDATE = 0 | 1	如果不设置（0），MySQL将不对客户机使用更新日志。这需要SUPER权限
SQL_QUOTE_SHOW_CREATE = 0 | 1	如果设置（1为默认），MySQL将引用表和列的名字
SQL_SAFE_UPDATES = 0 | 1	如果设置（1），MySQL将不执行那些不使用索引或LIMIT从句的UPDATE或DELETE语句
SQL_SELECT_LIMIT = value | DEFAULT	设置记录的最大值（默认为无限），该值可以用SELECT指令返回。LIMIT对该变量有优先权
TIMESTAMP = timestamp_value | DEFAULT	为客户机设置时间。当用更新日志恢复行时，该变量用于获得初始时间标记。timestamp_value是UNIX初相时间标记

老的SET OPTION语法现已不推荐使用，因此不要再使用它。

SET TRANSACTION

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL { READ UNCOMMITTED
 | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }
```

设置事务的隔绝级别。默认时，它只针对下一个事务，除非SESSION或GLOBAL关键字被使用 （这两者分别是对当前连接的所有事务设置级别或对所有新连接的事物设置级别）。

SHOW

SHOW语法可以为下列任何一种：

```
SHOW DATABASES [LIKE expression]
SHOW [OPEN] TABLES [FROM database_name] [LIKE expression]
SHOW [FULL] COLUMNS FROM table_name [FROM database_name] [LIKE expression]
SHOW INDEX FROM table_name [FROM database_name]
SHOW TABLE STATUS [FROM database_name] [LIKE expression]
SHOW STATUS [LIKE expression]
SHOW VARIABLES [LIKE expression]
SHOW LOGS
SHOW [FULL] PROCESSLIST
SHOW GRANTS FOR user
SHOW CREATE TABLE table_name
SHOW MASTER STATUS
SHOW MASTER LOGS
SHOW SLAVE STATUS
```

SHOW列出数据库、表或列，或提供服务器的状态信息。

匹配符可以是数据库、表或字段名的一部份，它可以是百分号（%）， 代表许多字符，或是下划线（_），代表一个字符。

TRUNCATE

```
TRUNCATE TABLE table_name
```

TRUNCATE语句从表中删除所有记录。它比同等的DELETE语句快，因为它终止和生成表。它不是事务安全的（因此如果有任何激活的事务或锁，会返回错误）。

UNION

```
SELECT ... UNION [ALL] SELECT ... [UNION SELECT ...]
```

Union把许多结果合并到一个中。

如果没有ALL关键字，行是惟一的。

UNLOCK TABLES

```
UNLOCK TABLES
```

释放所有被当前连接持有的锁。

UPDATE

```
UPDATE [LOW_PRIORITY] [IGNORE] table_name SET field_name1=expression1 [, field_
  name2=expression2, ...] [WHERE where_clause] [LIMIT #]
```

UPDATE更新数据库已存在行的内容。

SET从句指明哪些字段要更新和新值的内容。

where_clause给出行为了更新而必须遵守的条件。

IGNORE从句使MySQL忽略将导致双重主键或惟一键的UPDATE，而不是异常中断UPDATE。

LOW PRIORITY关键字使UPDATE等待，直到没有其他客户读取表才开始执行。

表达式可以取字段的当前值；例如，给所有雇员的拥金加五，其使用方法如下：

```
UPDATE employee SET commission=commission+5;
```

LIMIT决定要更新记录的最大值。

USE

```
USE database_name
```

改变当前激活数据库到指定数据库。

附录B MySQL函数和运算符索引

MySQL有很多有用的运算符和函数。运算符是MySQL基本的要素之一，并且大家对它们不应该陌生。虽然有很多可能从来不用的、不明显的函数，但还是值得对这个列表进行学习，这是因为操作者可能会发现一些内容将要使用，而一些内容以后会用。

逻辑运算符

逻辑运算符或逻辑算子（布尔运算符）查看一些事是对还是错。如果表达式是错的，那么它们返回值0，如果是对的，那么返回值1。空值（Null）的含义根据不同的运算符有所不同。通常它们返回一个NULL结果。

AND, &&

```
value1 AND value1
value1 && value2
```

如果两个值都是对的，那么返回true（1）。

例如：

```
mysql> SELECT 1 AND 0;
+----------+
| 1 AND 0 |
+----------+
|        0 |
+----------+
mysql> SELECT 1=1 && 2=2;
+--------------+
| 1=1 && 2=2 |
+--------------+
|            1 |
+--------------+
```

OR, ||

```
value1 OR value2
value1 || value 2
```

如果value1或value2是对的（true），那么返回true（1）。

例如：

```
mysql> SELECT 1 OR 1;
+---------+
```

```
| 1 OR 1 |
+--------+
|      1 |
+--------+
mysql> SELECT 1=2 || 2=3;
+------------+
| 1=2 || 2=3 |
+------------+
|          0 |
+------------+
```

NOT, !

```
NOT value1
! value1
```

返回与value1相反的值，如果value1是错的（false），则为true；如果value1是对的（true），则为false。

例如：

```
mysql> SELECT !1;
+----+
| !1 |
+----+
|  0 |
+----+
mysql> SELECT NOT(1=2);
+----------+
| NOT(1=2) |
+----------+
|        1 |
+----------+
```

数学运算符

数学运算符进行基本的数学运算。如果任何值都是空的，那么全部运算的结果也通常是空。由于计算的缘故，字符串被转换成数字。一些字符串被转换成为相同的数字（如字符串“1”和“33”），但是其他的被转换为0（如字符串“one”和“abc”）。

+

```
value1 + value2
```

将两个值加在一起。

例如：

```
mysql> SELECT 1+3;
+-----+
| 1+3 |
+-----+
|   4 |
+-----+
mysql> SELECT 15+"9";
+--------+
| 15+"9" |
+--------+
|     24 |
+--------+
```

-

```
value1 - value2
```

从value1中减去value2。

例如：

```
mysql> SELECT 1-9;
+-----+
| 1-9 |
+-----+
|  -8 |
+-----+
```

*

```
value1 * value2
```

将两个值相乘。

例如：

```
mysql> SELECT 12 * 10;
+---------+
| 12 * 10 |
+---------+
|     120 |
+---------+
```

/

```
value1 / value2
```

value1被value2除。

例如：

```
mysql> SELECT 4/2;
+------+
```

```
| 4/2  |
+------+
| 2.00 |
+------+
mysql> SELECT 10005.00000/10004.00000;
+--------------------------+
| 10005.00000/10004.00000 |
+--------------------------+
|                1.0001000 |
+--------------------------+
```

%

返回模（在value1被value2除后的余数）。

例如：

```
mysql> SELECT 3%2;
+------+
| 3%2  |
+------+
|    1 |
+------+
```

比较运算符

比较运算符比较数值，并根据结果返回对或错（true或false，1或0）。如果存在null值，那么大多数情况下运算符将返回NULL作为结果。不同的类型之间可以进行比较（字符串、数字、日期等等），虽然类型是不同的，但是操作者需要仔细。MySQL将这些类型转换为对等的内容。

如果操作者正在比较字符串，那么它们不分字符大小写进行比较，除非它们是BINARY。例如，A就是a，但是BINARY A与BINARY a的意思是不同的。由于大写字母在前，所以BINARY A小于BINARY a。相类似的情况还有，由于字符串的比较是从左到右的，所以字串10小于字符串2。首先检查是否1小于2，结果是1小于2，因此检查在此处就结束了（这与az小于b是相同的）。

=

```
value1 = value2
```

如果value1和value2是相同的，则为true。如果任一个是null，那么返回结果将是NULL。

例如：

```
mysql> SELECT 1=2;
+-----+
| 1=2 |
+-----+
```

```
|   0 |
+-----+
mysql> SELECT 'A' = 'a';
+-----------+
| 'A' = 'a' |
+-----------+
|         1 |
+-----------+
mysql> SELECT BINARY 'a' = 'A';
+-----------------+
| BINARY 'a' = 'A' |
+-----------------+
|                0 |
+-----------------+
mysql> SELECT NULL=NULL;
+-----------+
| NULL=NULL |
+-----------+
|      NULL |
+-----------+
```

!=, <>

```
value1 <> value2
value1 != value2
```

如果value1不等于value2，则正确（true）。

例如：

```
mysql> SELECT 'a' != 'A';
+-------------+
| 'a' != 'A' |
+-------------+
|           0 |
+-------------+
mysql> SELECT BINARY 'a' <> 'A';
+-------------------+
| BINARY 'a' <> 'A' |
+-------------------+
|                 1 |
+-------------------+
```

>

```
value1 > value2
```

如果value1大于value2，则正确（true）。

例如：

```
mysql> SELECT 1>2;
+-----+
| 1>2 |
+-----+
|   0 |
+-----+
mysql> SELECT 'b'>'a';
+---------+
| 'b'>'a' |
+---------+
|       1 |
+---------+
```

<

```
value1 < value2
```

如果value1小于value2，则正确（true）。

例如：

```
mysql> SELECT 'b' < 'd';
+-----------+
| 'b' < 'd' |
+-----------+
|         1 |
+-----------+
mysql> SELECT '4' < '34';
+------------+
| '4' < '34' |
+------------+
|          0 |
+------------+
```

>=

```
value1 >= value2
```

如果value1大于或等于value2，则正确（true）。

例如：

```
mysql> SELECT 4 >= 4;
+--------+
| 4 >= 4 |
+--------+
|      1 |
+--------+
```

<=

```
value1<= value2
```

如果value1小于或等于value2，则正确（true）。

例如：

```
mysql> SELECT 4 <= 3;
+--------+
| 4 <= 3 |
+--------+
|      0 |
+--------+
```

<=>

```
value1 <=> value2
```

如果value1等于value2，包括null，则正确。这使得操作者可以假设NULL是实际的值，并且在使用NULL与非NULL进行比较时得到正确（True）或错误（False）的结果（而不是NULL）。相反，MySQL拒绝为“Is 4 equal to NULL?（4和NULL相等吗）”给出结果。反而它会恰当地做出解释，表达式4 = NULL求解出不确定的值（NULL）。

例如：

```
mysql> SELECT NULL<=>NULL;
+--------------+
| NULL<=>NULL |
+--------------+
|            1 |
+--------------+
mysql> SELECT 4 <=> NULL;
+-------------+
| 4 <=> NULL |
+-------------+
|           0 |
+-------------+
```

IS NULL

```
value1 IS NULL
```

如果value1是null，则正确（true，而不是false）。

例如：

```
mysql> SELECT NULL IS NULL;
+--------------+
| NULL IS NULL |
+--------------+
```

```
|           1 |
+-------------+
mysql> SELECT 0 IS NULL;
+-----------+
| 0 IS NULL |
+-----------+
|         0 |
+-----------+
```

BETWEEN

```
value1 BETWEEN value2 AND value3
```

如果value1包含在value2和value3之间的范围内，则正确（true）。

例如：

```
mysql> SELECT 1 BETWEEN 0 AND 2;
+------------------+
| 1 BETWEEN 0 AND 2 |
+------------------+
|                1 |
+------------------+
mysql> SELECT 'a' BETWEEN 'A' and 'B';
+------------------------+
| 'a' BETWEEN 'A' and 'B' |
+------------------------+
|                      1 |
+------------------------+
mysql> SELECT BINARY 'a' BETWEEN 'A' AND 'C';
+--------------------------------+
| BINARY 'a' BETWEEN 'A' AND 'C' |
+--------------------------------+
|                              0 |
+--------------------------------+
```

LIKE

```
value1 LIKE value2
```

在SQL格式匹配上，如果value1与value2相匹配，则正确（true）。百分号（%）代表任意数量的字符，下划线（_）代表一个字符。

例如：

```
mysql> SELECT 'abc' LIKE 'ab_';
+-----------------+
| 'abc' LIKE 'ab_' |
+-----------------+
|               1 |
```

```
+------------------+
mysql> SELECT 'abc' LIKE '%c';
+------------------+
| 'abc' LIKE '%c' |
+------------------+
|                1 |
+------------------+
```

IN

```
value1 IN (value2 [value3,...])
```

如果value1与由逗号分隔的列表中的某个值相等，则正确（true）。

例如：

```
mysql> SELECT 'a' IN('b','c','aa');
+----------------------+
| 'a' IN('b','c','aa') |
+----------------------+
|                    0 |
+----------------------+
mysql> SELECT 'a' IN('A','B');
+------------------+
| 'a' IN('A','B') |
+------------------+
|                1 |
+------------------+
```

REGEXP, RLIKE

```
value1 REGEXP value2
value1 RLIKE value2
```

如果value1通过一个常规的表达式与value2相匹配，则正确。表B.1列出了常规表达式字符。

表B.1 常规表达式字符

字符	描述
*	与0匹配或与它前面子表达式的多个实例相匹配
+	与其前面子表达式的一个或多个实例相匹配
?	与0匹配或与它前面子表达式的一个实例相匹配
.	匹配任意一个字符
[xyz]	匹配x、y或z中的任意一个字符（括弧中的字符）
[A-Z]	匹配任意大写字母
[a-z]	匹配任意小写字母
[0-9]	匹配任意数字

（续表）

字符	描述
^	从头开始匹配
$	从末尾开始匹配
|	在常规表达式中分隔子表达式
{n,m}	子表达式必须出现n次，但不多于m次
{n}	子表达式必须正好出现n次
{n,}	子表达式必须至少出现n次
()	将字符分组入子表达式

例如：

```
mysql> SELECT 'pqwxyz' REGEXP 'xyz';
+-----------------------+
| 'pqwxyz' REGEXP 'xyz' |
+-----------------------+
|                     1 |
+-----------------------+
mysql> SELECT 'xyz' REGEXP '^x';
+--------------------+
| 'xyz' REGEXP '^x' |
+--------------------+
|                  1 |
+--------------------+
mysql> SELECT 'abcdef' REGEXP 'g+';
+----------------------+
| abcdef' REGEXP 'g+' |
+----------------------+
|                    0 |
+----------------------+
mysql> SELECT 'abcdef' REGEXP 'g*';
+----------------------+
| 'abcdef' REGEXP 'g*' |
+----------------------+
|                    1 |
+----------------------+
mysql> SELECT 'ian' REGEXP 'iai*n';
+----------------------+
| 'ian' REGEXP 'iai*n' |
+----------------------+
|                    1 |
+----------------------+
mysql> SELECT 'aaaa' REGEXP 'a{3,}';
+----------------------+
```

```
| 'aaaa' REGEXP 'a{3,}' |
+-----------------------+
|                     1 |
+-----------------------+
mysql> SELECT 'aaaa' REGEXP '^aaa$';
+-----------------------+
| 'aaaa' REGEXP '^aaa$' |
+-----------------------+
|                     0 |
+-----------------------+
mysql> SELECT 'abcabcabc' REGEXP 'abc{3}';
+-----------------------------+
| 'abcabcabc' REGEXP 'abc{3}' |
+-----------------------------+
|                           0 |
+-----------------------------+
mysql> SELECT 'abcabcabc' REGEXP '(abc){3}';
+-------------------------------+
| 'abcabcabc' REGEXP '(abc){3}' |
+-------------------------------+
|                             1 |
+-------------------------------+
mysql> SELECT 'abcbbcccc' REGEXP '[abc]{3}';
+-------------------------------+
| 'abcbbcccc' REGEXP '[abc]{3}' |
+-------------------------------+
|                             1 |
+-------------------------------+
mysql> SELECT 'abcbbcccc' REGEXP '(a|b|c){3}';
+---------------------------------+
| 'abcbbcccc' REGEXP '(a|b|c){3}' |
+---------------------------------+
|                               1 |
+---------------------------------+
```

位运算符

位运算符不经常使用。它们使操作者可以根据位值来工作，并且在查询中执行位计算。

&

```
value1 & value2
```

执行逐位的与操作（AND）。将值转换为二进制，并且比较这些位。只有在双方相应的位是1时，结果位才是1。

例如：

```
mysql> SELECT 2&1;
+-----+
| 2&1 |
+-----+
|   0 |
+-----+
mysql> SELECT 3&1;
+-----+
| 3&1 |
+-----+
|   1 |
+-----+
```

|

```
value1 | value2
```

执行逐位的或操作（OR）。将值转换为二进制，并且比较这些位。如果相应位的任一方是1，则结果位也为1。

例如：

```
mysql> SELECT 2|1;
+-----+
| 2|1 |
+-----+
|   3 |
+-----+
```

<<

```
value1 << value2
```

将value1转换为二进制，并且将value1的这些位按照value2的数量向左移。

例如：

```
mysql> SELECT 2<<1;
+------+
| 2<<1 |
+------+
|    4 |
+------+
```

>>

```
value1 >> value2
```

将value1转换为二进制，并且将value1的这些位按照value2的数量向右移。

例如：

```
mysql> SELECT 2>>1;
+------+
| 2>>1 |
+------+
|    1 |
+------+
```

日期和时间函数

在处理时间值时，操作者可以使用日期和时间函数，如在将当前时间以某种格式返回时或查看到某一天将会有多少天时。大多数情况下，date类型的值以YYYY-MM-DD形式存储（例如2002-12-25），并且time类型的值以hh:mm:ss的格式存储（例如11:23:43）。还有datetime类型，以YYYY-MM-DD hh:mm:ss格式存储。采用日期（date）和时间（time）的大多数函数将认可datetime的格式，并忽略它们不需要的部分。同样如果操作者少输入一些数值（在要求一个hh:mm:ss的数值时，操作者只输入了mm:ss部分），MySQL将假设剩下的部分为0，并进行正确的计算。操作者可以在日期和时间串中使用任意分界符，为了保持一致不要使用冒号（:）和破折号（-）。

某些函数使用特定的日期类型（例如DATE_ADD()，它需要一个典型的间隔来进行计算）。

下面是日期和时间类型：

- SECOND
- MINUTE
- HOUR
- DAY
- MONTH
- YEAR
- MINUTE_SECOND: "mm:ss"（例如，“41:23”）
- HOUR_MINUTE: "hh:mm"（例如，“12:23”）
- DAY_HOUR: "DD hh"（例如，“11 09”）
- YEAR_MONTH: "YYYY-MM"（例如，“2002-12”）
- HOUR_SECOND: "hh:mm:ss"（例如，“11:24:36”）
- DAY_MINUTE: "DD hh:mm"（例如，“09 11:31”）
- DAY_SECOND: "DD hh:mm:ss"（例如，“09 11:31:21”）

为进行日期的计算，操作者还可以使用常用的运算符（+、-等等），而不是日期函数。MySQL还可以在两个单位之间做出正确的转换。例如，当操作者向12月增加一个月时，MySQL将增加年份，并对月份做出正确的计算。

ADDDATE

```
ADDDATE(date,INTERVAL expression type)
```

DATE_ADD()的同义词。

CURDATE

```
CURDATE()
```

CURRENT_DATE()函数的同义词。

CURRENT_DATE

```
CURRENT_DATE()
```

按上下文关系向当前系统返回字符串YYYY-MM-DD或数字的YYYYMMDD。
例如：

```
mysql> SELECT CURRENT_DATE();
+----------------+
| CURRENT_DATE() |
+----------------+
| 2002-09-10     |
+----------------+
mysql> SELECT CURRENT_DATE()+1;
+------------------+
| CURRENT_DATE()+1 |
+------------------+
|         20020911 |
+------------------+
```

CURRENT_TIME

```
CURRENT_TIME()
```

按上下文关系向当前系统返回字符串hh:mm:ss或数字的hhmmss。
例如：

```
mysql> SELECT CURRENT_TIME();
+----------------+
| CURRENT_TIME() |
+----------------+
| 23:53:15       |
+----------------+
mysql> SELECT CURRENT_TIME() + 1;
+--------------------+
| CURRENT_TIME() + 1 |
+--------------------+
```

```
|              235434 |
+---------------------+
```

CURRENT_TIMESTAMP

```
CURRENT_TIMESTAMP()
```

NOW()函数的同义函数。

CURTIME

```
CURTIME()
```

CURRENT_TIME()函数的同义词。

DATE_ADD

```
DATE_ADD(date,INTERVAL expression type)
```

向指定的日期添加一个确定的时间段。操作者可以对表达式使用负值，意思是减掉这个值。其类型必须是本节开始时列出的类型（“日期和时间函数”）中的一种，表达式必须与类型相匹配。

例如：

```
mysql> SELECT DATE_ADD('2002-12-25',INTERVAL 1 MONTH);
+-----------------------------------------+
| DATE_ADD('2002-12-25',INTERVAL 1 MONTH) |
+-----------------------------------------+
| 2003-01-25                              |
+-----------------------------------------+
mysql> SELECT DATE_ADD('2002-12-25 13:00:00',INTERVAL -14 HOUR);
+-------------------------------------------------+
| DATE_ADD('2002-12-25 13:00:00',INTERVAL -14 HOUR) |
+-------------------------------------------------+
| 2002-12-24 23:00:00                             |
+-------------------------------------------------+
```

DATE_FORMAT

```
DATE_FORMAT(date,format_string)
```

基于格式字符串格式化指定日期，这可能会包含表B.2中列出的指定值。

表B.2 日期格式指定值

指定值	描述
%a	日期名字（Sun～Sat）的缩写
%b	月份名字（Jan～Dec）的缩写
%c	1～12的月份数字
%D	具有英文后缀的月份日期的数字表示（1st、2nd、等等）
%d	具有两个数字的月份日期的数字表示，00～31
%e	具有一个或两个数字的月份日期的数字表示，0～31
%H	00到23的小时的数字表示
%h	01～12的小时的数字表示
%I	00～59的分的数字表示
%I	01～12的小时的数字表示
%j	001～366的一年中日期的数字表示
%k	0～23的具有一个或两个数字的小时
%l	1～12的具有一个数字的小时
%M	January～December的月份名
%m	01～12的月份的数字表示
%p	A.M.或P.M.
%r	12小时时间，hh:mm:ss A.M.或P.M.
%S	00～59的秒的数字表示
%s	00～59的秒的数字表示
%T	24小时时间，hh:mm:ss
%U	00～53的周的数字表示，周日是一周的第一天
%u	00～53的周的数字表示，周一是一周的第一天
%V	01～53的周的数字表示，周日是一周的第一天
%v	01～53的周的数字表示，周一是一周的第一天
%W	Sunday～Saturday的一周中的每一天的名字
%w	一周的每一天的数字表示，周日是0，周六是6
%X	周日作为一周的第一天时，年的四位数字的表示形式
%x	周一作为一周的第一天时，年的四位数字的表示形式
%Y	年的四位数字的表示形式
%y	年的两位数字的表示形式
%%	转义的百分号

例如：

```
mysql> SELECT DATE_FORMAT('1999-03-02','%c %M');
+-----------------------------------+
| DATE_FORMAT('1999-03-02','%c %M') |
+-----------------------------------+
| 3 March                           |
+-----------------------------------+
```

DATE_SUB

```
DATE_SUB(date,INTERVAL expression type)
```

从指定的日期减去一个确定的时间段。操作者可以对表达式（expression）使用负值，这时它将进行加法运算。其类型必须是本节（“日期和时间”）开始时列出的类型之一，并且表达式必须与类型匹配。

例如：

```
mysql> SELECT DATE_SUB('2002-12-25 13:00:00',INTERVAL '14:13' MINUTE_SECOND);
+-----------------------------------------------------------------+
| DATE_SUB('2002-12-25 13:00:00',INTERVAL '14:13' MINUTE_SECOND) |
+-----------------------------------------------------------------+
| 2002-12-25 12:45:47                                             |
+-----------------------------------------------------------------+
```

DAYNAME

```
DAYNAME(date)
```

返回指定日期的日期名。

例如：

```
mysql> SELECT DAYNAME('2000-12-25');
+-----------------------+
| DAYNAME('2000-12-25') |
+-----------------------+
| Monday                |
+-----------------------+
```

DAYOFMONTH

```
DAYOFMONTH(date)
```

返回指定日期的月份日期，从1到31的数字。

例如：

```
mysql> SELECT DAYOFMONTH('2000-01-01');
+--------------------------+
| DAYOFMONTH('2000-01-01') |
+--------------------------+
|                        1 |
+--------------------------+
```

DAYOFWEEK

```
DAYOFWEEK(date)
```

返回指定日期的周内日期的数字，1代表周日，7代表周六，这是开放式数据库连接性（Open Database Connectivity，ODBC）的标准。

例如：

```
mysql> SELECT DAYOFWEEK('2000-01-01');
+--------------------------+
| DAYOFWEEK('2000-01-01') |
+--------------------------+
|                        7 |
+--------------------------+
```

利用WEEKDAY()来返回天的索引，0~6，周一到周日的数字标志。

DAYOFYEAR

```
DAYOFYEAR(date)
```

对所提供的日期给出一年中天的数字，从1到366。
例如：

```
mysql> SELECT DAYOFYEAR('2000-12-25');
+--------------------------+
| DAYOFYEAR('2000-12-25') |
+--------------------------+
|                      360 |
+--------------------------+
```

EXTRACT

```
EXTRACT(date_type FROM date)
```

用指定的日期类型给出日期的某个部分。参见date函数开始之前的日期列表。
例如：

```
mysql> SELECT EXTRACT(YEAR FROM '2002-02-03');
+---------------------------------+
| EXTRACT(YEAR FROM '2002-02-03') |
+---------------------------------+
|                            2002 |
+---------------------------------+
mysql> SELECT EXTRACT(MINUTE_SECOND FROM '2002-02-03 12:32:45');
+-------------------------------------------------+
| EXTRACT(MINUTE_SECOND FROM '2002-02-03 12:32:45') |
+-------------------------------------------------+
|                                            3245 |
+-------------------------------------------------+
```

FROM_DAYS

```
FROM_DAYS(number)
```

将从0年1月1日起计算的指定天数转换成日期，并返回结果。不包括在转换成罗马教皇Gregory历法变革中丢弃的天数。

例如：

```
mysql> SELECT FROM_DAYS(731574);
+-------------------+
| FROM_DAYS(731574) |
+-------------------+
| 2002-12-25        |
+-------------------+
```

FROM_UNIXTIME

```
FROM_UNIXTIME(unix_timestamp [, format_string])
```

将指定的时间戳转换成日期，并返回结果。如果提供了日期格式，那么返回数据将按此格式转换。格式串可以是DATE_FORMAT()函数中提供的任意一个。

例如：

```
mysql> SELECT FROM_UNIXTIME(100);
+---------------------+
| FROM_UNIXTIME(100)  |
+---------------------+
| 1970-01-01 00:01:40 |
+---------------------+
mysql> SELECT FROM_UNIXTIME(1031621727,'%c %M');
+-----------------------------------+
| FROM_UNIXTIME(1031621727,'%c %M') |
+-----------------------------------+
| 9 September                       |
+-----------------------------------+
```

HOUR

```
HOUR(time)
```

返回指定时间的小时数字，从0到23。

例如：

```
mysql> SELECT HOUR('06:59:03');
+------------------+
| HOUR('06:59:03') |
+------------------+
|                6 |
+------------------+
```

MINUTE

```
MINUTE(time)
```

返回指定时间的分钟数，从0到59。

例如：

```
mysql> SELECT MINUTE('00:01:03');
+--------------------+
| MINUTE('00:01:03') |
+--------------------+
|                  1 |
+--------------------+
```

MONTH

```
MONTH(date)
```

返回指定的日期的月份，从1到12。

例如：

```
mysql> SELECT MONTH('2000-12-25');
+---------------------+
| MONTH('2000-12-25') |
+---------------------+
|                  12 |
+---------------------+
```

MONTHNAME

```
MONTHNAME(date)
```

返回指定日期的月份的名称。例如：

```
mysql> SELECT MONTHNAME('2000-12-25');
+-------------------------+
| MONTHNAME('2000-12-25') |
+-------------------------+
| December                |
+-------------------------+
```

NOW

```
NOW()
```

返回当前的时间戳（以格式YYYY-MM-DD hh:mm:ss表示的日期和时间），根据环境，返回的数据可以是字符串，也可以是数字。此函数在一个查询操作的多个调用中将返回相同的结果。

例如：

```
mysql> SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2002-09-10 00:58:06 |
+---------------------+
```

与CURRENT_TIMESTAMP()和SYSDATE()函数的功能相同。

PERIOD_ADD

```
PERIOD_ADD(period,months)
```

把月份加入时间段（以YYMM或YYYYMM的形式指定的），并返回YYYYMM形式的结果。

例如：

```
mysql> SELECT PERIOD_ADD(200205,3);
+----------------------+
| PERIOD_ADD(200205,3) |
+----------------------+
|               200208 |
+----------------------+
mysql> SELECT PERIOD_ADD(200205,-42);
+-----------------------+
| PERIOD_ADD(200205,-42) |
+-----------------------+
|                199811 |
+-----------------------+
```

PERIOD_DIFF

```
PERIOD_DIFF(period1,period2)
```

返回在period和period2（用格式YYMM或YYYYMM指定的）之间的月份的数字值。

例如：

```
mysql> SELECT PERIOD_DIFF(200212,200001);
+----------------------------+
| PERIOD_DIFF(200212,200001) |
+----------------------------+
|                         35 |
+----------------------------+
mysql> SELECT PERIOD_DIFF(199903,199904);
+----------------------------+
| PERIOD_DIFF(199903,199904) |
+----------------------------+
```

```
|                          -1 |
+-----------------------------+
```

QUARTER

```
QUARTER(date)
```

返回指定日期的季节，从1到4。

例如：

```
mysql> SELECT QUARTER('2002-06-30');
+-----------------------+
| QUARTER('2002-06-30') |
+-----------------------+
|                     2 |
+-----------------------+
```

SEC_TO_TIME

```
SEC_TO_TIME(seconds)
```

将秒数转化为时间，根据环境需要，以字符串（hh:mm:ss）或数值（hhmmss）的形式返回。

例如：

```
mysql> SELECT SEC_TO_TIME(1000);
+--------------------+
| SEC_TO_TIME(1000)  |
+--------------------+
| 00:16:40           |
+--------------------+
mysql> SELECT SEC_TO_TIME(-10000);
+---------------------+
| SEC_TO_TIME(-10000) |
+---------------------+
| -02:46:40           |
+---------------------+
```

SECOND

```
SECOND(time)
```

返回指定时间的秒数，从0到58。

例如：

```
mysql> SELECT SECOND('00:01:03');
+--------------------+
| SECOND('00:01:03') |
+--------------------+
```

```
|                    3 |
+----------------------+
```

SUBDATE

```
SUBDATE(date,INTERVAL expression type)
```

DATE_SUB()函数的同义词。

SYSDATE

```
SYSDATE()
```

NOW()函数的同义词。

TIME_FORMAT

```
TIME_FORMAT(time,format)
```

与函数DATE_FORMAT()相同，但只能使用处理时间格式的子集（或返回NULL值）。

TIME_TO_SEC

```
TIME_TO_SEC(time)
```

将时间转化为秒数，并返回结果。

例如：

```
mysql> SELECT TIME_TO_SEC('00:01:03');
+--------------------------+
| TIME_TO_SEC('00:01:03') |
+--------------------------+
|                       63 |
+--------------------------+
```

TO_DAYS

```
TO_DAYS(date)
```

从0年的1月开始，返回到指定日期的天数。此函数不把罗马教皇Gregory历法变革中损失的天数考虑在内。

例如：

```
mysql> SELECT TO_DAYS('2000-01-01');
+-----------------------+
| TO_DAYS('2000-01-01') |
+-----------------------+
|                730485 |
+-----------------------+
```

UNIX_TIMESTAMP

```
UNIX_TIMESTAMP([date])
```

返回表示当前系统时间（如果函数被调用时不带参数）或指定日期的UNIX时间戳（从1970年1月1日午夜开始计算秒数）的无符号的整数。

例如：

```
mysql> SELECT UNIX_TIMESTAMP();
+------------------+
| UNIX_TIMESTAMP() |
+------------------+
|       1031621727 |
+------------------+
mysql> SELECT UNIX_TIMESTAMP('1970-01-01 00:01:40');
+---------------------------------------+
| UNIX_TIMESTAMP('1970-01-01 00:01:40') |
+---------------------------------------+
|                                   100 |
+---------------------------------------+
```

WEEK

```
WEEK(date [,week_start])
```

返回指定日期的在给定年份的周。除非可选的参数week_start被设置为1，即假定周是从星期一开始的，否则认为周是从星期日开始的。当然也可以明确将参数设置为0，表示周是从星期日开始的。对给定年份中的第一个星期日（或星期一）之前的日期，此函数将返回0。

例如：

```
mysql> SELECT WEEK('2002-06-31');
+--------------------+
| WEEK('2002-06-31') |
+--------------------+
|                 26 |
+--------------------+
mysql> SELECT WEEK('2002-06-31',1);
+----------------------+
| WEEK('2002-06-31',1) |
+----------------------+
|                   27 |
+----------------------+
mysql> SELECT WEEK('1998-12-31',1);
+----------------------+
| WEEK('1998-12-31',1) |
+----------------------+
|                   53 |
```

```
+----------------------+
mysql> SELECT WEEK('1998-01-01');
+--------------------+
| WEEK('1998-01-01') |
+--------------------+
|                  0 |
+--------------------+
```

如果日期在给定年份的第一个星期日（或星期一）之前，使用函数YEARWEEK()将返回到从前一年开始的那一周。

WEEKDAY

```
WEEKDAY(date)
```

返回指定日期所在那周的天数，数字0到6分别表示星期一到星期日。
例如：

```
mysql> SELECT WEEKDAY('2000-01-01');
+-----------------------+
| WEEKDAY('2000-01-01') |
+-----------------------+
|                     5 |
+-----------------------+
```

根据ODBC标准（1～7表示星期日到星期六），使用函数DAYOFWEEK()返回天数的索引值。

YEAR

```
YEAR(date)
```

返回指定日期的年份，从1000到9999。
例如：

```
 mysql> SELECT YEAR('2002-06-30');
+--------------------+
| YEAR('2002-06-30') |
+--------------------+
|               2002 |
+--------------------+
```

YEARWEEK

```
YEARWEEK(date [,week_start])
```

返回指定日期的年份和周的组合值。除非可选的参数week_start被设置为1，即假定周是从星期一开始的，否则假定周是从星期日开始。当然也可以明确将参数设置为0，表示周是从星期日开始的。年份可以是从前一年到本年或下一年中第一个星期日（或星期一）之前的日子。

例如：

```
mysql> SELECT YEARWEEK('2002-12-25');
+-----------------------+
| YEARWEEK('2002-12-25') |
+-----------------------+
|                200251 |
+-----------------------+
mysql> SELECT YEARWEEK('1998-12-31',1);
+--------------------------+
| YEARWEEK('1998-12-31',1) |
+--------------------------+
|                   199853 |
+--------------------------+
mysql> SELECT YEARWEEK('1998-01-01');
+-----------------------+
| YEARWEEK('1998-01-01') |
+-----------------------+
|                199752 |
+-----------------------+
```

使用WEEK()函数返回给定年份中的周。

字符串函数

字符串函数主要接受字符串参数，并以字符串形式返回结果。与大多数程序语言不同，MySQL字符串中第一个字符的位置是1，而不是0。

ASCII

```
ASCII(string)
```

返回字符串中第一个字符（最左边）的ASCII值，如果字符串为空，则值为0，并且如果字符串无效，返回的值是NULL。

例如：

```
mysql> SELECT ASCII('a');
+------------+
| ASCII('a') |
+------------+
|         97 |
+------------+
mysql> SELECT ASCII('aa');
+-------------+
| ASCII('az') |
+-------------+
|          97 |
+-------------+
```

如果字符是多字节的，则使用函数ORD()返回ASCII值。

BIN

```
BIN(number)
```

返回指定的BIGINT数字的二进制值（字符串表示法），如果数字不能被转换（此函数从左边开始将尽可能进行转换），返回数值0，并且如果字符串无效，返回值为NULL。

例如：

```
mysql> SELECT BIN(15);
+---------+
| BIN(15) |
+---------+
| 1111    |
+---------+
mysql> SELECT BIN('8');
+----------+
| BIN('8') |
+----------+
| 1000     |
+----------+
mysql> SELECT BIN('2w');
+-----------+
| BIN('2w') |
+-----------+
| 10        |
+-----------+
mysql> SELECT BIN('w2');
+-----------+
| BIN('w2') |
+-----------+
| 0         |
+-----------+
```

这个函数等同于CONV(number,10,2)。

BIT_LENGTH

```
BIT_LENGTH(string)
```

返回字符串的长度（以位为单位）。

例如：

```
mysql> SELECT BIT_LENGTH('MySQL');
+--------------------+
| BIT_LENGTH('MySQL') |
+--------------------+
```

```
| 40                    |
+-----------------------+
```

CHAR

```
CHAR(number1[, number2[, ...]])
```

如果每个数字都是整数，则这个函数返回的字符是从数字转换过来的ASCII代码，跳过无效的值。十进制的数四舍五入到最近的整数。

例如：

```
mysql> SELECT CHAR(97,101,105,111,117);
+--------------------------+
| CHAR(97,101,105,111,117) |
+--------------------------+
| aeiou                    |
+--------------------------+
mysql> SELECT CHAR(97.6,101,105,111,117);
+----------------------------+
| CHAR(0.97,101,105,111,117) |
+----------------------------+
| beiou                      |
+----------------------------+
```

CHAR_LENGTH

与LENGTH()函数相同，只是多字节字符只被计算一次。

CHARACTER_LENGTH

与函数LENGTH()相同，只是多字节字符只被计算一次。

CONCAT

```
CONCAT(string1[,string2[,...]])
```

连接字符串参数，并返回合成的字符串，或者如果任何一个参数无效，则返回NULL。不是字符串的参数被转换成字符串。

例如：

```
mysql> SELECT CONCAT('a','b');
+-----------------+
| CONCAT('a','b') |
+-----------------+
| ab              |
+-----------------+
mysql> SELECT CONCAT('a',12);
+----------------+
| CONCAT('a',12) |
```

```
+----------------+
| a12            |
+----------------+
mysql> SELECT CONCAT(.3,'NULL');
+-------------------+
| CONCAT(.3,'NULL') |
+-------------------+
| 0.3NULL           |
+-------------------+
mysql> SELECT CONCAT(.3,NULL);
+-----------------+
| CONCAT(.3,NULL) |
+-----------------+
| NULL            |
+-----------------+
```

CONCAT_WS

```
CONCAT_WS(separator, string1[, string2[, ...]])
```

除了第一个参数作为分隔符，放在要连接的字符串之间，其他功能与CONCAT相似。此函数将跳过任何无效的字符串（除了分隔符，若分隔符无效，则结果为NULL）。分隔符没必要必须是字符串。

例如：

```
mysql> SELECT CONCAT_WS('-','a','b');
+------------------------+
| CONCAT_WS('-','a','b') |
+------------------------+
| a-b                    |
+------------------------+
mysql> SELECT CONCAT_WS(1,.3,.4);
+--------------------+
| CONCAT_WS(1,.3,.4) |
+--------------------+
| 0.310.4            |
+--------------------+
mysql> SELECT CONCAT_WS(NULL,'a','b');
+-------------------------+
| CONCAT_WS(NULL,'a','b') |
+-------------------------+
| NULL                    |
+-------------------------+
mysql> SELECT CONCAT_WS('-','a',NULL,'c');
+-----------------------------+
| CONCAT_WS('-','a',NULL,'c') |
+-----------------------------+
```

```
| a-c                          |
+------------------------------+
```

CONV

```
CONV(number,from_base,to_base)
```

将数字从一种基数转换为另一种基数，并返回转换后的代表字符串的数字。如果转换没有成功，则结果为0（此函数将从左边开始尽可能地转换）；如果数字无效，则结果为NULL。数字可认为是整数，但可以作为字符串进行传递。数字应该是无符号的，除非基数是负数。基数可以是2和36之间的任意一个数（带有to_base可能就是负数）。

例如：

```
mysql> SELECT CONV(10,2,10);
+---------------+
| CONV(10,2,10) |
+---------------+
| 2             |
+---------------+
mysql> SELECT CONV('a',16,2);
+-----------------+
| CONV('a',16,2)  |
+-----------------+
| 1010            |
+-----------------+
mysql> SELECT CONV('3f',16,10);
+-------------------+
| CONV('3f',16,10)  |
+-------------------+
| 63                |
+-------------------+

 mysql> SELECT CONV('z3',16,10);
+-------------------+
| CONV('z3',16,10)  |
+-------------------+
| 0                 |
+-------------------+
1 row in set (0.00 sec)
mysql> SELECT CONV('3z',16,10);
+-------------------+
| CONV('3z',16,10)  |
+-------------------+
| 3                 |
+-------------------+
```

ELT

```
ELT(number, string1 [,string2, ...])
```

使用number作为索引来决定返回哪一个字符串。1返回第一个字符串，2返回第二个字符串，以次类推。如果没有匹配的字符串，则返回结果为NULL。

例如：

```
mysql> SELECT ELT(2,'one','two');
+--------------------+
| ELT(2,'one','two') |
+--------------------+
| two                |
+--------------------+
mysql> SELECT ELT(0,'one','two');
+--------------------+
| ELT(0,'one','two') |
+--------------------+
| NULL               |
+--------------------+
```

函数FIELD()是ELT()的补充。

EXPORT_SET

```
EXPORT_SET(number,on,off[,separator[,number_of_bits]])
```

检查number是否为二进制数。如果返回on，则对每一比特位设置进行设置，若返回off，则不进行设置。默认的分隔符是逗号，但你可以指定为其他符号。使用64位，但也可以改变number_of_bits。

例如：

```
mysql> SELECT EXPORT_SET(2,1,0,' ',4);
+-------------------------+
| EXPORT_SET(2,1,0,' ',4) |
+-------------------------+
| 0 1 0 0                 |
+-------------------------+
mysql> SELECT EXPORT_SET(7,'ok','never',' : ',6);
+-------------------------------------+
| EXPORT_SET(7,'ok','never',' : ',6)  |
+-------------------------------------+
| ok : ok : ok : never : never : never |
+-------------------------------------+
```

FIELD

```
FIELD(string, string1 [, string2 , ...])
```

返回string在下表中的索引值。如果string1匹配，则索引值为1。如果string2匹配，则索引值为2，以此类推。如果没有找到匹配的字符串，则返回结果为0。

例如：

```
mysql> SELECT FIELD('b','a','b','c');
+------------------------+
| FIELD('b','a','b','c') |
+------------------------+
|                      2 |
+------------------------+
mysql> SELECT FIELD('a','aa','b','c');
+-------------------------+
| FIELD('a','aa','b','c') |
+-------------------------+
|                       0 |
+-------------------------+
```

FIND_IN_SET

```
FIND_IN_SET(string,string list)
```

与FIELD()相似，因此它返回匹配此字符串的索引。但是此函数搜寻由逗号分隔的或SET类型的字符串。如果字符串匹配逗号前的第一个子串（或者元素集中的某一项），它将返回数值1，如果匹配第二个子串，它将返回2，以此类推。如果没有符合的项，则返回0。需要注意的是，它要与整个逗号分隔的子串去比较，而不是字符串中的任何一部分。

例如：

```
mysql> SELECT FIND_IN_SET('b','a,b,c');
+--------------------------+
| FIND_IN_SET('b','a,b,c') |
+--------------------------+
|                        2 |
+--------------------------+
mysql> SELECT FIND_IN_SET('a','aa,bb,cc');
+-----------------------------+
| FIND_IN_SET('a','aa,bb,cc') |
+-----------------------------+
|                           0 |
+-----------------------------+
1 row in set (0.00 sec)
```

HEX

```
HEX(string or number)
```

返回指定BIGINT数的十六进制值（字符串表示法），如果此数字无法转换，则返回0（此函数将从左到右进行转换，一直到无法转换为止），如果此数字是空的，则返回NULL。

如果参数是数字，它被转换为十六进制值（相当于CONV(number,10,16)函数）。如果参数是字符串，则字符串中的每一个字符被转换为ASCII表中等值的数值（例如，a = 97，b = 98，等等），并且这些数字中的每一个反过来又被转换为等值的十六进制数。

例如：

```
mysql> SELECT HEX(13);
+---------+
| HEX(13) |
+---------+
| D       |
+---------+

mysql> SELECT ORD('a');
+----------+
| ORD('a') |
+----------+
|       97 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT ORD('b');
+----------+
| ORD('b') |
+----------+
|       98 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT HEX(97);
+---------+
| HEX(97) |
+---------+
| 61      |
+---------+
1 row in set (0.00 sec)

mysql> SELECT HEX(98);
+---------+
| HEX(98) |
+---------+
| 62      |
```

```
+---------+
1 row in set (0.00 sec)

mysql> SELECT HEX('ab');
+-----------+
| HEX('ab') |
+-----------+
| 6162      |
+-----------+
1 row in set (0.00 sec)
```

INSERT

```
INSERT(string,position,length,newstring)
```

从位置position开始连续length个字符，用newstring替换字符串中的一部分。newstring的长度和指定的长度可以不同，在这种情况下，原先的字符串将改变长度。

这个函数是多字节安全的。

例如：

```
mysql> SELECT INSERT('MySQL',1,0,'What is ');
+--------------------------------+
| INSERT('MySQL',1,0,'What is ') |
+--------------------------------+
| What is MySQL                  |
+--------------------------------+
mysql> SELECT INSERT('MySQL',1,1,'P');
+------------------------+
| INSERT('MySQL',1,1,'P') |
+------------------------+
| PySQL                  |
+------------------------+
mysql> SELECT INSERT('MySQL',1,1,'Py');
+-------------------------+
| INSERT('MySQL ,1,1,'Py') |
+-------------------------+
| PyySQL                  |
+-------------------------+
```

INSTR

```
INSTR(string,substring)
```

搜索大小写不相关的substring第一次出现在string中的位置（除非任意一个字符串是二进制的），并返回所在位置，或者如果substring没有找到，则返回0。第一个字母的位置是1。

例如：

```
mysql> SELECT INSTR('MySQL','My');
+--------------------+
| INSTR('MySQL','My') |
+--------------------+
|                  1 |
+--------------------+
mysql> SELECT INSTR('Cecilia','i');
+----------------------+
| INSTR('Cecilia','i') |
+----------------------+
|                    4 |
+----------------------+
```

LCASE

```
LCASE(string)
```

函数LOWER()的同义词。

LEFT

```
LEFT(string,length)
```

返回字符串最左边起长度为length的字符。这个函数是多字节安全的。

例如：

```
mysql> SELECT LEFT('abc',2);
+---------------+
| LEFT('abc',2) |
+---------------+
| ab            |
+---------------+
```

LENGTH

```
LENGTH(string)
```

返回字符串中字符的个数。如果可以，将参数转换为一个字符串。

例如：

```
mysql> SELECT LENGTH('MySQL');
+-----------------+
| LENGTH('MySQL') |
+-----------------+
|               5 |
+-----------------+
mysql> SELECT LENGTH(99);
+------------+
| LENGTH(99) |
```

```
+-------------+
|           2 |
+-------------+
```

CHAR_LENGTH()、CHARACTER_LENGTH()和OCTET_LENGTH()是同义词，只是多字节字符在CHAR_LENGTH()和CHARACTER_LENGTH()函数中只被计算一次。

LOAD_FILE

```
LOAD_FILE(file_name)
```

读取文件并以字符串形式返回文件内容。这个文件必须在服务器上，必须指定文件的完整路径名，并且必须有FILE权限。文件必须是所有人可读的，而且要小于max_allowed_packet。如果因为前述原因之一导致文件不存在或不可读，则函数返回NULL。

例如，如果文件/home/iang/test.txt包含文本123456，用函数LOAD_FILE()应该返回下列内容：

```
mysql> SELECT LOAD_FILE('/home/iang/test.txt');
+---------------------------------+
| LOAD_FILE('/home/iang/test.txt') |
+---------------------------------+
| 123456                          |
+---------------------------------+
```

LOAD_FILE经常与BLOB一起被加载入数据库。例如：

```
mysql> INSERT INTO table_with_blob(id,image)
 VALUES(1,LOAD_FILE('/images/pic.jpg'));
```

LOCATE

```
LOCATE(substring, string [,position])
```

搜索大小写不相关的substring第一次出现在string中的位置（除非任意一个字符串是二进制的），并返回所在位置。如果substring没有找到，则返回0。如果可选的参数position被使用，则从这个位置开始搜索。第一个字符的位置是1。

例如：

```
mysql> SELECT LOCATE('My','MySQL');
+----------------------+
| LOCATE('My','MySQL') |
+----------------------+
|                    1 |
+----------------------+
mysql> SELECT LOCATE('C','Cecilia',2);
+------------------------+
| LOCATE('C','Cecilia',2) |
+------------------------+
|                      3 |
+------------------------+
```

这个函数与INSTR()相同，但参数的顺序是颠倒的。

LOWER

```
LOWER(string)
```

返回所有字符转换为小写字母的字符串（根据当前字符集的映射）。此函数是多字节安全的。

例如：

```
mysql> SELECT LOWER('AbC');
+--------------+
| LOWER('AbC') |
+--------------+
| abc          |
+--------------+
```

函数LCASE()是同义词。

LPAD

```
LPAD(string,length,padding_string)
```

用padding_string来从左填补字符串，直到字符的长度达到length。如果字符串的长度比length长，它将被缩减为length个字符。

例如：

```
mysql> SELECT LPAD('short',7,'-');
+---------------------+
| LPAD('short',7,'-') |
+---------------------+
| --short             |
+---------------------+
mysql> SELECT LPAD('too_long',7,' ');
+------------------------+
| LPAD('too_long',7,' ') |
+------------------------+
| too_lon                |
+------------------------+
mysql> SELECT LPAD('a',4,'12');
+------------------+
| LPAD('a',4,'12') |
+------------------+
| 121a             |
+------------------+
```

LTRIM

```
LTRIM(string)
```

从字符串中删除前置的空格，并返回结果。

例如：

```
mysql> SELECT LTRIM('   Yes');
+-----------------+
| LTRIM('   Yes') |
+-----------------+
| Yes             |
+-----------------+
```

MAKE_SET

```
MAKE_SET(number, string1 [, string2, ...])
```

返回一个字符串集（其中的元素是用逗号分隔的字符串），它与转换为二进制的number匹配。如果二进制值设为0，则返回第一个字符串；如果二进制值设为1，则返回第二个字符串，以此类推。如果比特位参数设为3，则返回前两个字符串，因为3的二进制表示法是11。

例如：

```
mysql> SELECT MAKE_SET(3,'a','b','c');
+-------------------------+
| MAKE_SET(3,'a','b','c') |
+-------------------------+
| a,b                     |
+-------------------------+
mysql> SELECT MAKE_SET(5,'a','b','c');
+-------------------------+
| MAKE_SET(5,'a','b','c') |
+-------------------------+
| a,c                     |
+-------------------------+
```

OCT

```
OCT(number)
```

返回指定的BIGINT数字的八进制值（字符串表示法）。如果数字不能被转换（此函数尽可能从左边开始转换），返回值为0；如果数字无效，则返回NULL。

例如：

```
mysql> SELECT OCT(09);
+---------+
| OCT(09) |
+---------+
```

```
| 11       |
+----------+
mysql> SELECT OCT('a1');
+-----------+
| OCT('a1') |
+-----------+
| 0         |
+-----------+
mysql> SELECT OCT('13b');
+------------+
| OCT('13b') |
+------------+
| 15         |
+------------+
```

这个函数等价于CONV(number,10,8)。

OCTET_LENGTH

函数LENGTH()的同义词。

ORD

```
ORD(string)
```

返回字符串的第一个（最左边）字符的ASCII值。如果字符串为空，则返回0；如果字符串无效，则返回NULL。这个函数与ASCII函数相同，除非字符是多字节的。这种情况下，值是以256为基数计算出来的，也就是，每个字节相当于比下一个字节多出256倍。例如，有两个字节的字符的计算公式如下：（第一个字节的ASCII码×256）+（第二个字节的ASCII码）。

例如：

```
mysql> SELECT ORD("a");
+-----------+
| ORD("a") |
+-----------+
|        97 |
+-----------+
mysql> SELECT ORD("az");
+------------+
| ORD("az") |
+------------+
|         97 |
+------------+
```

可以使用函数BIN()、OCT()和HEX()将十进制的数分别转换为二进制、八进制和十六进制的数。

POSITION

```
POSITION(substring IN string)
```

搜索大小写相关的substring第一次出现在string中的位置（除非任意一个参数是二进制的字符串），并返回所在位置（从1开始）。如果substring没有找到，则返回0。这个函数是多字节安全的。

例如：

```
mysql> SELECT POSITION('i' IN 'Cecilia');
+----------------------------+
| POSITION('i' IN 'Cecilia') |
+----------------------------+
|                          4 |
+----------------------------+
```

QUOTE

```
QUOTE(string)
```

转义单引号（'）、双引号（"）、ASCII NULL和Ctrl+Z字符，用单引号将字符串括起来，这样可以在SQL语句中安全地使用。如果参数是NULL，则不加单引号。

例如：

```
mysql> SELECT QUOTE("What's Up?");
+---------------------+
| QUOTE("What's Up?") |
+---------------------+
| 'What\'s Up?'       |
+---------------------+
```

REPEAT

```
REPEAT(string,count)
```

重复字符串的参数count次，并返回结果。如果count不是正整数，则返回空字符串，或者如果任何一个参数是无效，则返回NULL。

例如：

```
mysql> SELECT REPEAT('a',4);
+---------------+
| REPEAT('a',4) |
+---------------+
| aaaa          |
+---------------+
mysql> SELECT REPEAT('a',-1);
+----------------+
| REPEAT('a',-1) |
```

```
+----------------+
|                |
+----------------+
mysql> SELECT REPEAT('a',NULL);
+------------------+
| REPEAT('a',NULL) |
+------------------+
| NULL             |
+------------------+
```

REPLACE

```
REPLACE(string,from_string,to_string)
```

用to_string替换string中所有出现from_string的地方，并返回结果。这个函数是多字节安全的。

例如：

```
mysql> SELECT REPLACE('ftp://test.host.co.za','ftp','http');
+-----------------------------------------------+
| REPLACE('ftp://test.host.co.za','ftp','http') |
+-----------------------------------------------+
| http://test.host.co.za                        |
+-----------------------------------------------+
```

REVERSE

```
REVERSE(string)
```

颠倒string中字符的顺序，并返回结果。这个函数是多字节安全的。

例如：

```
mysql> SELECT REVERSE('abc');
+----------------+
| REVERSE('abc') |
+----------------+
| cba            |
+----------------+
```

RIGHT

```
RIGHT(string,length)
```

返回string中从最右边开始长度为length的字符。这个函数是多字节安全的。

例如：

```
mysql> SELECT RIGHT('abc',2);
+----------------+
| RIGHT('abc',2) |
```

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

RPAD

```
RPAD(string,length,padding_string)
```

用padding_string填充字符串中右边的空间，直到达到length个字符长度。如果字符串比length值长，则字符串被缩减为length个字符。

例如：

```
mysql> SELECT RPAD('short',7,'-');
+--------------------+
| RPAD('short',7,'-') |
+--------------------+
| short--            |
+--------------------+
mysql> SELECT RPAD('too_long',7,' ');
+-----------------------+
| RPAD('too_long',7,' ') |
+-----------------------+
| too_lon               |
+-----------------------+
mysql> SELECT RPAD('a',4,'12');
+------------------+
| RPAD('a',4,'12') |
+------------------+
| a121             |
+------------------+
```

RTRIM

```
RTRIM(string)
```

从字符串中删除尾部的空格，并返回结果。

例如：

```
mysql> SELECT CONCAT('a',RTRIM('b        '),'c');
+-----------------------------------+
| CONCAT('a',RTRIM('b        '),'c') |
+-----------------------------------+
| abc                               |
+-----------------------------------+
```

SOUNDEX

```
SOUNDEX(string)
```

返回一个探测型字符串，它是一个语音字符串，被设计用来作为一种更方便的索引方法来克服拼写错误。字符串的发音若相同，则返回相同的探测字符串。探测字符串通常4个字符长，但这个函数返回一个任意长度的字符串。在SOUNDEX()前面使用函数SUBstring()，可以返回标准的探测函数。非字母数字的字符被忽略，非英语的国际字母的字符被作为元音对待。

例如：

```
mysql> SELECT SOUNDEX('MySQL');
+------------------+
| SOUNDEX('MySQL') |
+------------------+
| M240             |
+------------------+
mysql> SELECT SOUNDEX('MySequl');
+--------------------+
| SOUNDEX('MySequl') |
+--------------------+
| M240               |
+--------------------+
```

SPACE

```
SPACE(number)
```

返回的字符串由number个空格组成。

例如：

```
mysql> SELECT "A",SPACE(10),"B";
+---+------------+---+
| A | SPACE(10)  | B |
+---+------------+---+
| A |            | B |
+---+------------+---+
```

SUBSTRING

```
SUBSTRING(string, position [,length])
SUBSTRING(string FROM position [FOR length])
```

从位置position开始（从1开始计算）返回字符串参数的子串，并可以指定length作为可选的参数。

例如：

```
mysql> SELECT SUBSTRING('MySQL',2);
+----------------------+
| SUBstring('MySQL',2) |
+----------------------+
| ySQL                 |
```

```
+----------------------+
mysql> SELECT SUBSTRING('MySQL' FROM 3);
+--------------------------+
| SUBstring('MySQL' FROM 3) |
+--------------------------+
| SQL                      |
+--------------------------+
1 row in set (0.16 sec)
mysql> SELECT SUBSTRING('MySQL',1,2);
+------------------------+
| SUBstring('MySQL',1,2) |
+------------------------+
| My                     |
+------------------------+
1 row in set (0.22 sec)
```

此函数是多字节安全的。函数MID(string, position, length)是SUBSTRING(string, position, length)的同义词。

SUBSTRING_INDEX

```
SUBSTRING_INDEX(string,delimiter,count)
```

返回遇到count个分隔符之前（count为正数）的字符串的子串，或分隔符之后（count为负数）的字符串的子串。

这个函数是多字节安全的。

例如：

```
mysql> SELECT SUBSTRING_INDEX('a||b||c||d','||',3);
+-------------------------------------+
| SUBstring_INDEX('a||b||c||d','||',3) |
+-------------------------------------+
| a||b||c                             |
+-------------------------------------+
mysql> SELECT SUBSTRING_INDEX('I am what I am','a',2);
+---------------------------------------+
| SUBstring_INDEX('I am what I am','a',2) |
+---------------------------------------+
| I am wh                               |
+---------------------------------------+
mysql> SELECT SUBSTRING_INDEX('I am what I am','a',-2);
+----------------------------------------+
| SUBstring_INDEX('I am what I am','a',-2) |
+----------------------------------------+
| t I am                                 |
+----------------------------------------+
```

TRIM

```
TRIM([[BOTH | LEADING | TRAILING] [trim_string] FROM] string)
```

如果没有指定任何一个可选的参数，TRIM()删掉字符串中前导的和后置的空格。指定参数LEADING或TRAILING以便只删除其中的一种或指定默认值BOTH。还可以通过指定trim_string来删除除了空格以外的其他字符。这个函数是多字节安全的。

例如：

```
mysql> SELECT TRIM('   What a waste of space        ') AS t;
+-----------------------+
| t                     |
+-----------------------+
| What a waste of space |
+-----------------------+
mysql> SELECT TRIM(LEADING '0' FROM '0001');
+-------------------------------+
| TRIM(LEADING '0' FROM '0001') |
+-------------------------------+
| 1                             |
+-------------------------------+
mysql> SELECT TRIM(LEADING FROM '         1');
+---------------------------------+
| TRIM(LEADING FROM '         1') |
+---------------------------------+
| 1                               |
+---------------------------------+
mysql> SELECT TRIM(BOTH 'abc' FROM 'abcabcaabbccabcabc');
+-------------------------------------------+
| TRIM(BOTH 'abc' FROM 'abcabcaabbccabcabc') |
+-------------------------------------------+
| aabbcc                                    |
+-------------------------------------------+
```

UCASE

```
UCASE(string)
```

UPPER()函数的同义字。

UPPER

```
UPPER(string)
```

返回所有字符已转换成大写字母的字符串（根据当前字符集的映射）。这个函数是多字节安全的。

例如：

```
mysql> SELECT UPPER('aBc');
+--------------+
| UPPER('aBc') |
+--------------+
| ABC          |
+--------------+
```

函数UCASE()是同义词。

数字函数Numeric Functions

数字函数用来处理数字，主要接收数字参数并返回数字结果。在出现错误的情况下，返回NULL值。要注意，不要超出数字的范围——大多数MySQL函数在BIGINT范围内工作（无符号为2^{63}，有符号为2^{64}）。如果超出这个范围，MySQL通常返回NULL值。

ABS

```
ABS(number)
```

返回数字的绝对值（正值）。此函数在BIGINT范围内可以安全使用。
例如：

```
mysql> SELECT ABS(24-26);
+-------------+
| ABS(24-26) |
+-------------+
|           2 |
+-------------+
```

ACOS

```
ACOS(number)
```

返回数字的反余弦值。这个数字必须在－1和1之间，否则函数返回NULL值。
例如：

```
mysql> SELECT ACOS(0.9);
+------------+
| ACOS(0.9) |
+------------+
|  0.451027 |
+------------+
```

ASIN

```
ASIN(number)
```

返回数字的反正弦值。这个数字必须在－1和1之间，否则函数返回NULL值。

例如：

```
mysql> SELECT ASIN(-0.4);
+-------------+
| ASIN(-0.4) |
+-------------+
|  -0.411517 |
+-------------+
```

ATAN

```
ATAN(number1 [, number2])
```

返回数字的反正切值，或两个数字（点number1、 number2）的反正切值。
例如：

```
mysql> SELECT ATAN(4);
+-----------+
| ATAN(4)   |
+-----------+
| 1.325818 |
+-----------+
mysql> SELECT ATAN(-4,-3);
+--------------+
| ATAN(-4,-3) |
+--------------+
|   -2.214297 |
+--------------+
```

ATAN2

```
ATAN2(number1,number2)
```

ATAN(number1, number2)的同义词。

CEILING

```
CEILING(number)
```

四舍五入数字到最接近的整数，并作为BIGINT返回。
例如：

```
mysql> SELECT CEILING(2.98);
+---------------+
| CEILING(2.98) |
+---------------+
|             3 |
+---------------+
mysql> SELECT CEILING(-2.98);
```

```
+-----------------+
| CEILING(-2.98) |
+-----------------+
|              -2 |
+-----------------+
```

使用FLOOR()向下四舍五入，使用ROUND()向上或向下四舍五入。

COS

```
COS(number_radians)
```

返回number_radians的余弦值。

例如：

```
mysql> SELECT COS(51);
+-----------+
| COS(51)   |
+-----------+
| 0.742154 |
+-----------+
```

COT

```
COT(number_radians)
```

返回number_radians的余切值。

例如：

```
mysql> SELECT COT(0.45);
+-------------+
| COT(0.45)   |
+-------------+
| 2.07015736 |
+-------------+
```

DEGREES

```
DEGREES(number)
```

将数字由弧度转换为度数，并返回结果。

例如：

```
mysql> SELECT DEGREES(2);
+------------------+
| DEGREES(2)       |
+------------------+
| 114.59155902616 |
+------------------+
```

```
mysql> SELECT DEGREES(PI()/2);
+-----------------+
| DEGREES(PI()/2) |
+-----------------+
|              90 |
+-----------------+
```

EXP

```
EXP(number)
```

返回数值e的（自然对数的底数）指定幂。

例如：

```
mysql> SELECT EXP(1);
+----------+
| EXP(1)   |
+----------+
| 2.718282 |
+----------+
mysql> SELECT EXP(2.3);
+----------+
| EXP(2.3) |
+----------+
| 9.974182 |
+----------+
mysql> SELECT EXP(0.3);
+----------+
| EXP(0.3) |
+----------+
| 1.349859 |
+----------+
```

FLOOR

```
FLOOR(number)
```

四舍五入数字到最接近的整数，并作为BIGINT返回。

例如：

```
mysql> SELECT FLOOR(2.98);
+-------------+
| FLOOR(2.98) |
+-------------+
|           2 |
+-------------+
mysql> SELECT FLOOR(-2.98);
+--------------+
```

```
| FLOOR(-2.98) |
+--------------+
|           -3 |
+--------------+
```

使用CEILING()向上四舍五入，使用ROUND()向上或向下四舍五入。

FORMAT

```
FORMAT(number,decimals)
```

用逗号把每三位数字分隔开，并将数字四舍五入到指定的位数。

例如：

```
mysql> SELECT FORMAT(88777634.232,2);
+------------------------+
| FORMAT(88777634.232,2) |
+------------------------+
| 88,777,634.23          |
+------------------------+
```

GREATEST

```
GREATEST(argument1, argument2 [, ...])
```

返回所给参数中的最大值。将根据返回值的情况或参数类型用不同的方式比较参数，参数可以是整数、实数或字符串（大小写相关和默认的）。

例如：

```
mysql> SELECT GREATEST(-3,-4,5);
+-------------------+
| GREATEST(-3,-4,5) |
+-------------------+
|                 5 |
+-------------------+
mysql> SELECT GREATEST('Pa','Ma','Ca');
+--------------------------+
| GREATEST('Pa', Ma','Ca') |
+--------------------------+
| Pa                       |
+--------------------------+
```

LEAST

```
LEAST(argument1, argument2 [, ...])
```

返回所给参数中的最小值。将根据返回值的情况或参数类型用不同的方式比较参数，参数可以是整数、实数或字符串（大小写相关和默认的）。

例如：

```
mysql> SELECT LEAST(-3,-4,5);
+----------------+
| LEAST(-3,-4,5) |
+----------------+
|             -4 |
+----------------+
mysql> SELECT LEAST('Pa','Ma','Ca');
+-----------------------+
| LEAST('Pa','Ma','Ca') |
+-----------------------+
| Ca                    |
+-----------------------+
```

LN

```
LN(number)
```

函数LOG(number)的同义词。

LOG

```
LOG(number1 [, number2])
```

如果没有参数，则返回number1的自然对数。还可以通过提供第二个参数，使用任意一个底数，这种情况下，函数返回LOG(number2) / LOG(number1)。

例如：

```
mysql> SELECT LOG(2);
+----------+
| LOG(2)   |
+----------+
| 0.693147 |
+----------+
mysql> SELECT LOG(2,3);
+----------+
| LOG(2,3) |
+----------+
| 1.584963 |
+----------+
```

LOG10

```
LOG10(number1)
```

返回以10为底数的number1的对数。这个函数相当于LOG(number1)/LOG(10)。

例如：

```
mysql> SELECT LOG10(100);
+------------+
| LOG10(100) |
+------------+
|   2.000000 |
+------------+
```

LOG2

```
LOG2(number1)
```

返回以2为底数的number1的对数。这个函数相当于LOG(number1)/LOG(2)。

例如：

```
mysql> SELECT LOG2(4);
+----------+
| LOG2(4)  |
+----------+
| 2.000000 |
+----------+
```

MOD

```
MOD(number1,number2)
```

返回number1和number2的模数（number1除以number2后的余数）。这个函数与操作符%相同。在范围内可以安全使用这个函数。

例如：

```
mysql> SELECT MOD(15,4);
+-----------+
| MOD(15,4) |
+-----------+
|         3 |
+-----------+
mysql> SELECT MOD(3,-2);
+-----------+
| MOD(3,-2) |
+-----------+
|         1 |
+-----------+
```

PI

```
PI()
```

返回π值（或至少是最接近的表示方法）。MySQL使用完整的双精度，但默认情况下只返回5个字符。

例如：

```
mysql> SELECT PI();
+----------+
| PI()     |
+----------+
| 3.141593 |
+----------+
mysql> SELECT PI() + 0.0000000000000000;
+---------------------------+
| PI() + 0.0000000000000000 |
+---------------------------+
|         3.1415926535897931 |
+---------------------------+
```

POW

```
POW(number1,number2)
```

这个函数是POWER(number1, number2)的同义词。

POWER

```
POWER(number1,number2)
```

number1的number2次方，并返回值。

例如：

```
mysql> SELECT POWER(2,3);
+------------+
| POWER(2,3) |
+------------+
|   8.000000 |
+------------+
```

RADIANS

```
RADIANS(number1)
```

将数字由度数转换为弧度，并返回结果。

例如：

```
mysql> SELECT RADIANS(180);
+-----------------+
| RADIANS(180)    |
+-----------------+
| 3.1415926535898 |
+-----------------+
```

RAND

```
RAND([number])
```

返回一个0到1之间的随机数（浮点数）。参数是随机数的开端。通常使用时间戳作为开端。此函数可以被用来返回一个随机顺序的结果集。

例如：

```
mysql> SELECT RAND();
+------------------+
| RAND()           |
+------------------+
| 0.70100469486881 |
+------------------+
mysql> SELECT RAND(20021010081523);
+----------------------+
| RAND(20021010081523) |
+----------------------+
|     0.80558716573924 |
+----------------------+
mysql> SELECT * FROM t1 ORDER BY RAND() LIMIT 1;
+----+
| f1 |
+----+
| 20 |
+----+
```

ROUND

```
ROUND(number1 [, number2])
```

返回参数number1，四舍五入到最近的整数。可以使用第二个参数来指定小数进行四舍五入（默认值是0，或没有小数）。四舍五入的数字若恰好为中间值，则根据下面的C库处理。

例如：

```
mysql> SELECT ROUND(2.49);
+-------------+
| ROUND(2.49) |
+-------------+
|           2 |
+-------------+
mysql> SELECT ROUND(2.51);
+-------------+
| ROUND(2.51) |
+-------------+
|           3 |
```

```
+--------------+
mysql> SELECT ROUND(-2.49,1);
+----------------+
| ROUND(-2.49,1) |
+----------------+
|           -2.5 |
+----------------+
```

SIGN

```
SIGN(number)
```

根据参数是否为负数、零或不是数字、正数来返回－1、0或1。

例如：

```
mysql> SELECT SIGN(-7);
+----------+
| SIGN(-7) |
+----------+
|       -1 |
+----------+
mysql> SELECT SIGN('a');
+------------+
| SIGN('a') |
+------------+
|         0 |
+------------+
```

SIN

```
SIN(number_radians)
```

返回number_radians的正弦值。

例如：

```
mysql> SELECT SIN(45);
+----------+
| SIN(45)  |
+----------+
| 0.850904 |
+----------+
```

SQRT

```
SQRT(number)
```

返回参数的平方根。

例如：

```
mysql> SELECT SQRT(81);
+----------+
| SQRT(81) |
+----------+
| 9.000000 |
+----------+
```

TAN

```
TAN(number_radians)
```

返回number_radians的正切值。

例如：

```
mysql> SELECT TAN(66);
+----------+
| TAN(66)  |
+----------+
| 0.026561 |
+----------+
```

TRUNCATE

```
TRUNCATE(number,decimals)
```

缩减（或增加）数字到指定的小数的位数。

例如：

```
mysql> SELECT TRUNCATE(2.234,2);
+--------------------+
| TRUNCATE(2.234,2)  |
+--------------------+
|               2.23 |
+--------------------+
mysql> SELECT TRUNCATE(2.4,5);
+-----------------+
| TRUNCATE(2.4,5) |
+-----------------+
|         2.40000 |
+-----------------+
mysql> SELECT TRUNCATE(2.998,0);
+--------------------+
| TRUNCATE(2.998,0)  |
+--------------------+
|                  2 |
+--------------------+
mysql> SELECT TRUNCATE(-12.43,1);
+--------------------+
```

```
| TRUNCATE(-12.43,1) |
+--------------------+
|              -12.4 |
+--------------------+
```

求和函数

合计函数作用于一群数据（也就是说，它们可以在GROUP BY从句中使用）。如果没有GROUP BY从句，它们将认为全部的结果集是一群数据，并返回一个结果。下面这个例子中，就假设存在一个简单的表，如下所示：

```
mysql> SELECT * FROM table1;
+--------+
| field1 |
+--------+
|      4 |
|     12 |
|     12 |
|     20 |
+--------+
4 rows in set (0.00 sec)
```

AVG

```
AVG(expression)
```

返回组中表达式的平均值。如果表达式不是数值，将返回0。

例如：

```
mysql> SELECT AVG(field1) FROM table1;
+-------------+
| AVG(field1) |
+-------------+
|     12.0000 |
+-------------+
```

BIT_AND

```
BIT_AND(expression)
```

从组中（执行64位的精度）按位与表达式中的所有位进行AND操作，返回结果。

```
mysql> SELECT BIT_AND(field1) FROM table1;
+-----------------+
| BIT_AND(field1) |
+-----------------+
|               4 |
+-----------------+
```

BIT_OR

```
BIT_OR(expession)
```

从组中（执行64位的精度）按位与表达式中的所有位进行OR操作，返回结果。

例如：

```
mysql> SELECT BIT_OR(field1) FROM table1;
+----------------+
| BIT_OR(field1) |
+----------------+
|             28 |
+----------------+
```

COUNT

```
COUNT( [DISTINCT] exprcssion1, [expression2])
```

返回组中有效数字的个数。

如果表达式是一个字段，返回这个字段中不包含无效值的行数。COUNT(*)返回所有行数。参数DISTINCT返回单一的非无效值的个数（如果使用的表达式多于 -个，则为组合值）。

```
mysql> SELECT COUNT(*) FROM table1;
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
```

MAX

```
MAX(expression)
```

返回组中表达式的最大值。表达式可以是数字或字符串。

例如：

```
mysql> SELECT MAX(field1) FROM table1;
+-------------+
| MAX(field1) |
+-------------+
|          20 |
+-------------+
```

MIN

```
MIN(expression)
```

返回组中表达式的最小值。表达式可以是数字或字符串。

例如：

```
mysql> SELECT MIN(field1) FROM table1;
+-------------+
| MIN(field1) |
+-------------+
|           4 |
+-------------+
```

STD

```
STD(expression)
```

从组中返回表达式值的标准偏差。

例如：

```
mysql> SELECT STD(field1) FROM table1;
+-------------+
| STD(field1) |
+-------------+
|      5.6569 |
+-------------+
```

STDDEV

```
STDDEV(expression)
```

函数STD()的同义词。

SUM

```
SUM(expression)
```

返回组中表达式的最小值，如果不存在行，则返回NULL。表达式可以是数字或字符串。

例如：

```
mysql> SELECT SUM(field1) FROM table1;
+-------------+
| MIN(field1) |
+-------------+
|          48 |
+-------------+
```

其他函数

下面这些函数包括加密函数、比较函数、流控制函数和其他各种函数。

AES_DECRYPT

```
AES_DECRYPT(encrypted_string,key_string)
```

对AES_ENCRYPT()函数的结果进行解密。

AES_ENCRYPT

```
AES_ENCRYPT(string,key_string)
```

使用高级加密标准算法（Rijndae）根据key_string对字符串加密。这个函数使用默认长度为128位的密钥。AES_DECRYPT()对结果进行解密。

BENCHMARK

```
BENCHMARK(count,expression)
```

运行表达式count次。主要用来测试MySQL运行表达式的速度有多快。经常会返回0，显示在函数下面的时间是输出结果中有用的部分。

例如：

```
mysql> SELECT BENCHMARK(10000,SHA('how long'));
+-----------------------------------+
| BENCHMARK(10000,SHA('how long')) |
+-----------------------------------+
|                                 0 |
+-----------------------------------+
1 row in set (0.95 sec)
```

CASE

```
CASE value WHEN [compare_value1] THEN result1 [WHEN [compare_value2]
 THEN result2 ...] [ELSE result3] END
CASE WHEN [condition1] THEN result1 [WHEN [condition2]
 THEN result2 ...] [ELSE result3] END
```

CASE语句有两种使用方法。第一种根据值返回结果。它把这个值与各种不同的compare_values进行比较，并返回与这个值（THEN后面）相关的结果，如果没有找到合适的值，则返回ELSE后面的结果。如果没有结果返回，则返回NULL。

第二种方法比较各种条件，并在发现条件为真的时候，返回相关的结果。如果没有找到，则返回ELSE后面的结果。如果没有结果返回，则返回NULL。

例如：

```
mysql> SELECT CASE 'a' WHEN 'a' THEN 'a it is' END;
+-------------------------------------+
| CASE 'a' WHEN 'a  THEN 'a it is' END |
+-------------------------------------+
| a it is                              |
+-------------------------------------+
```

```
mysql> SELECT CASE 'b' WHEN 'a' THEN 'a it is' WHEN 'b' THEN 'b it is' END;
+-----------------------------------------------------------+
| CASE 'b' WHEN 'a' THEN 'a it is' WHEN 'b' THEN 'b it is' END |
+-----------------------------------------------------------+
| b it is                                                   |
+-----------------------------------------------------------+
mysql> SELECT CASE 9 WHEN 1 THEN 'is 1' WHEN 2 THEN 'is 2' ELSE 'not found' END;
+-------------------------------------------------------------------+
| CASE 9 WHEN 1 THEN 'is 1' WHEN 2 THEN 'is 2' ELSE 'not found' END |
+-------------------------------------------------------------------+
| not found                                                         |
+-------------------------------------------------------------------+
mysql> SELECT CASE 9 WHEN 1 THEN 'is 1' WHEN 2 THEN 'is 2' END;
+-------------------------------------------------+
| CASE 9 WHEN 1 THEN 'is 1' WHEN 2 THEN 'is 2'  END |
+-------------------------------------------------+
| NULL                                            |
+-------------------------------------------------+
mysql> SELECT CASE WHEN 1>2 THEN '1>2' WHEN 2=2 THEN 'is 2' END;
+-------------------------------------------------+
| CASE WHEN 1>2 THEN '1>2' WHEN 2=2 THEN 'is 2'  END |
+-------------------------------------------------+
| is 2                                            |
+-------------------------------------------------+
mysql> SELECT CASE WHEN 1>2 THEN '1>2' WHEN 2<2 THEN '2<2' ELSE 'none' END;
+-------------------------------------------------------------+
| CASE WHEN 1>2 THEN '1>2' WHEN 2<2 THEN '2<2' ELSE 'none' END |
+-------------------------------------------------------------+
| none                                                        |
+-------------------------------------------------------------+
mysql> SELECT CASE WHEN BINARY 'a' = 'A' THEN 'bin' WHEN 'a'='A' THEN 'text' END;
+-----------------------------------------------------------------+
| CASE WHEN BINARY 'a' = 'A' THEN 'bin' WHEN 'a'='A' THEN 'text' END |
+-----------------------------------------------------------------+
| text                                                            |
+-----------------------------------------------------------------+
mysql> SELECT CASE WHEN BINARY 1=1 THEN '1' WHEN 2=2 THEN '2' END;
+-------------------------------------------------+
| CASE WHEN BINARY 1=1 THEN '1' WHEN 2=2 THEN '2' END |
+-------------------------------------------------+
| 1                                               |
+-------------------------------------------------+
```

返回值的类型（INTEGER、DOUBLE或STRING）与第一次返回的值（第一个THEN后面的表达式）的类型相同。

CAST

```
CAST(expression AS type)
```

将表达式转换为指定的类型，并返回结果。此类型可以是下面这些类型中的一种：BINARY、DATE、DATETIME、SIGNED、SIGNED INTEGER、TIME、UNSIGNED和UNSIGNED INTEGER。

MySQL通常会自动进行类型转换。例如，如果你增加了两个数字字符串，结果将是一个数值。或者如果部分计算结果是无符号的，则整个结果将为无符号的。可以使用CAST()来改变这种行为。

例如：

```
mysql> SELECT "4" + "3";
+-----------+
| "4" + "3" |
+-----------+
|         7 |
+-----------+
mysql> SELECT CAST(("4"+"3") AS TIME);
+-------------------------+
| CAST(("4"+"3") AS TIME) |
+-------------------------+
| 7                       |
+-------------------------+
mysql> SELECT CAST(50-60 AS UNSIGNED INTEGER);
+---------------------------------+
| CAST(50-60 AS UNSIGNED INTEGER) |
+---------------------------------+
|            18446744073709551606 |
+---------------------------------+
mysql> SELECT CAST(50-60 AS SIGNED INTEGER);
+-------------------------------+
| CAST(50-60 AS SIGNED INTEGER) |
+-------------------------------+
|                           -10 |
+-------------------------------+
```

CONVERT()是使用ODBC语法的同义词。

CONNECTION_ID

```
CONNECTION_ID()
```

返回连接的惟一一个线程号。

例如：

```
mysql> SELECT CONNECTION_ID();
+-----------------+
| CONNECTION_ID() |
+-----------------+
|               7 |
+-----------------+
```

CONVERT

```
CONVERT(expression,type)
```

这个函数是CAST(expression AS type)的同义词，它是ANSI SQL99语法。

DATABASE

```
DATABASE()
```

返回当前数据库的名字，如果没有数据库，则返回空字符串。

例如：

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| test       |
+------------+
```

DECODE

```
DECODE(encoded_string,password_string)
```

对使用密码的被加密的字符串进行解码，并返回结果。被解密的字符串通常先由ENCODE()函数产生。

例如：

```
mysql> SELECT DECODE('g','1');
+-----------------+
| DECODE('g','1') |
+-----------------+
| a               |
+-----------------+
mysql> SELECT DECODE('wer','1sz');
+--------------------+
| DECODE('wer','1sz') |
+--------------------+
| 8                  |
+--------------------+
```

DES_DECRYPT

```
DES_DECRYPT(encrypted_string [, key_string])
```

对用DES_ENCRYPT()加密的字符串解密。

DES_ENCRYPT

```
DES_ENCRYPT(string [, (key_number | key_string) ] )
```

使用数据加密标准（DES）算法对字符串加密，并返回一个二进制字符串。如果可选的密钥参数被忽略，则使用des-key文件中的第一个密钥。如果密钥参数是数字（从0到9），则使用des-key文件中相对应的密钥。如果密钥参数是字符串，那么这个字符串就是密钥。

如果des-key文件中的密钥值发生了改变，MySQL在运行FLUSH_DES_KEY_FILE语句的时候，可以读取新的值，这要求有reload权限。

这个函数只在MySQL支持安全套接层（SSL）的时候可以工作。

ENCODE

```
ENCODE(string,password_string)
```

返回一个加密的二进制字符串。可以使用DECODE()和同样的password_string返回原始字符串。加密和解密的字符串具有同样的长度。

例如：

```
mysql> SELECT ENCODE('a','1');
+-----------------+
| ENCODE('a','1') |
+-----------------+
| g               |
+-----------------+
mysql> SELECT ENCODE('ah','2');
+------------------+
| ENCODE('ah','2') |
+------------------+
| U                |
+------------------+
```

ENCRYPT

```
ENCRYPT(string [, salt])
```

使用UNIX crypt()系统调用来加密字符串，并返回结果。可选的salt参数是加密过程使用的字符串。它的行为取决于下面的系统调用。

例如：

```
mysql> SELECT ENCRYPT('keepmecut');
+----------------------+
```

```
| ENCRYPT('keepmeout') |
+----------------------+
| V9tOly.dRY55k        |
+----------------------+
mysql> SELECT ENCRYPT('keepmeout','ab');
+--------------------------+
| ENCRYPT('keepmeout','ab') |
+--------------------------+
| abpr3o3DrHzJo            |
+--------------------------+
```

FOUND_ROWS

```
FOUND_ROWS()
```

返回满足前面SELECT SQL_CALC_FOUND_ROWS查询（如果没有用LIMIT从句限制，将被返回）要求的行数。

例如：

```
mysql> SELECT SQL_CALC_FOUND_ROWS user FROM user LIMIT 1;
+------+
| user |
+------+
|      |
+------+
1 row in set (0.00 sec)

mysql> SELECT FOUND_ROWS();
+---------------+
| FOUND_ROWS() |
+---------------+
|             4 |
+---------------+
```

GET_LOCK

```
GET_LOCK(string,timeout)
```

在timeout秒之内，试图获得一个叫做string的锁。如果成功，它将返回1。如果超时，则返回0。如果出现其他错误，则返回NULL。使用RELEASE_LOCK()可以释放锁，GET_LOCK()获得新锁，或者线程结束。可以使用IS_FREE_LOCK()查看锁是否为自由的。

这个函数主要用来作为应用程序的额外锁定机制。

例如：

```
mysql> SELECT GET_LOCK('one',1);
+--------------------+
| GET_LOCK('one',1) |
```

```
+------------------+
|                1 |
+------------------+
```

IF

```
IF(expression1,expression2,expression3)
```

如果expression1为真，返回expression2，否则返回expression3。这个函数可以根据情况，返回一个数字或字符串。expression1被作为整数进行求值，所以真实的比较可能不会产生出你希望的结果。

例如：

```
mysql> SELECT IF('a'='a',1,2);
+-----------------+
| IF('a'='a',1,2) |
+-----------------+
|               1 |
+-----------------+
mysql> SELECT IF(9<4,1,2);
+-------------+
| IF(9<4,1,2) |
+-------------+
|           2 |
+-------------+
mysql> SELECT IF(NULL,'a','b');
+-------------------+
| IF(NULL,'a','b') |
+-------------------+
| b                 |
+-------------------+
mysql> SELECT IF(16-6-10,'a',NULL);
+----------------------+
| IF(16-6-10,'a',NULL) |
+----------------------+
| NULL                 |
+----------------------+
```

下一个例子返回false，因为实数0.49被作为整数0进行求值：

```
mysql> SELECT IF(0.49,'true','false');
+-------------------------+
| IF(0.49,'true','false') |
+-------------------------+
| false                   |
+-------------------------+
```

IFNULL

```
IFNULL(expression1,expression2)
```

如果不是无效，返回expression1，否则返回expression2。根据情况，结果可以是数字或字符串。

例如：

```
mysql> SELECT IFNULL(1,2);
+-------------+
| IFNULL(1,2) |
+-------------+
|           1 |
+-------------+
mysql> SELECT IFNULL(NULL,'nothing here');
+----------------------------+
| IFNULL(NULL,'nothing here') |
+----------------------------+
| nothing here               |
+----------------------------+
mysql> SELECT IFNULL(RELEASE_LOCK('nonexistant'),'The lock never existed');
+-----------------------------------------------------------------+
| IFNULL(RELEASE_LOCK('nonexistant'),'The lock never existed') |
+-----------------------------------------------------------------+
|                                          The lock never existed |
+-----------------------------------------------------------------+
```

INET_ATON

```
INET_ATON(dotted_quad_string)
```

从点分的字符串返回4字节或8字节的网络地址。

例如：

```
mysql> SELECT INET_ATON('196.26.90.168');
+----------------------------+
| INET_ATON('196.26.90.168') |
+----------------------------+
|                 3290061480 |
+----------------------------+
```

INET_NTOA

```
INET_NTOA(network_address)
```

从4字节或8字节的网络地址返回点分的地址，并返回表示点分地址的字符串。

例如：

```
mysql> SELECT INET_NTOA(3290061480);
+-----------------------+
| INET_NTOA(3290061480) |
+-----------------------+
| 196.26.90.168         |
+-----------------------+
```

IS_FREE_LOCK

```
IS_FREE_LOCK(string)
```

此函数被用来查看锁命名的字符串（由GET_LOCK()产生）是否为自由的。

如果锁是自由的，它将返回1。如果锁被持有，则返回0。如果出现其他错误，则返回NULL。

例如：

```
mysql> SELECT GET_LOCK('one',1);
+-------------------+
| GET_LOCK('one',1) |
+-------------------+
|                 1 |
+-------------------+
mysql> SELECT IS_FREE_LOCK('one');
+---------------------+
| IS_FREE_LOCK('one') |
+---------------------+
|                   0 |
+---------------------+
mysql> SELECT GET_LOCK('two',1);
+-------------------+
| GET_LOCK('two',1) |
+-------------------+
|                 1 |
+-------------------+
mysql> SELECT IS_FREE_LOCK('one');
+---------------------+
| IS_FREE_LOCK('one') |
+---------------------+
|                   1 |
+---------------------+
```

LAST_INSERT_ID

```
LAST_INSERT_ID([expression])
```

返回来自这个连接的插入字段的最后一个值。如果没有插入值，则返回0。

例如：

```
mysql> SELECT LAST_INSERT_ID();
+------------------+
| last_insert_id() |
+------------------+
|                0 |
+------------------+
```

MASTER_POS_WAIT

```
MASTER_POS_WAIT(log_name, log_position)
```

被用来进行复制同步。在从属服务器上运行，这个函数一直等待，直到从属服务器完成所有的在主服务器日志文件中指定位置的更新操作。

例如：

```
mysql> SELECT MASTER_POS_WAIT('g-bin.001',273);
+-----------------------------------------+
| MASTER_POS_WAIT('g-bin.001',273)        |
+-----------------------------------------+
|                                    NULL |
+-----------------------------------------+
```

MD5

```
MD5(string)
```

使用消息摘要算法计算出字符串的128位的校验和，并返回32位的十六进制数字。

例如：

```
mysql> SELECT MD5('how many more');
+----------------------------------+
| MD5('how many more')             |
+----------------------------------+
| 75dea0eddd9ffb8db451448a9931e764 |
+----------------------------------+
```

使用函数SHA()作为更安全的加密替代方法。

NULLIF

```
NULLIF(expression1,expression2)
```

除非expression1等于expression2，否则返回expression1。若等于，则返回NULL。如果expression1等于expression2，这个函数对expression1求值两次。

例如：

```
mysql> SELECT NULLIF('a','b');
+-----------------+
```

```
| NULLIF('a','b') |
+-----------------+
| a               |
+-----------------+
mysql> SELECT NULLIF(1,'1');
+---------------+
| NULLIF(1,'1') |
+---------------+
|          NULL |
+---------------+
```

PASSWORD

```
PASSWORD(string)
```

将字符串转换为一个加密的密码，并返回结果。这个函数被用于在mysql数据库的用户表中加密口令。这个加密是不可逆的，并且它与普通的UNIX口令加密方法不同。

例如：

```
mysql> SELECT PASSWORD('a');
+-------------------+
| PASSWORD('a')     |
+-------------------+
| 60671c896665c3fa  |
+-------------------+
mysql> SELECT PASSWORD(PASSWORD('a'));
+--------------------------+
| PASSWORD(PASSWORD('a')) |
+--------------------------+
| 772a81723a030f10         |
+--------------------------+
```

ENCRYPT()将字符串转换为UNIX方式的密码。

RELEASE_LOCK

```
RELEASE_LOCK(string)
```

释放较早时候由GET_LOCK()获得的锁的字符串。如果锁被释放，则返回1。如果锁没有被释放，则返回0，因为这个连接没有创建这个锁。如果锁不存在，则返回NULL。

例如：

```
mysql> SELECT GET_LOCK('one',1);
+-------------------+
| GET_LOCK('one',1) |
+-------------------+
|                 1 |
+-------------------+
mysql> SELECT RELEASE_LOCK('one');
```

```
+----------------------+
| RELEASE_LOCK('one') |
+----------------------+
|                    1 |
+----------------------+
mysql> SELECT RELEASE_LOCK('one');
+----------------------+
| RELEASE_LOCK('one') |
+----------------------+
|                 NULL |
+----------------------+
```

SESSION_USER

```
SESSION_USER()
```

返回与当前线程连接的MySQL用户和主机。

例如：

```
mysql> SELECT SESSION_USER();
+----------------+
| SESSION_USER() |
+----------------+
| root@localhost |
+----------------+
```

SYSTEM_USER()和USER()是同义词。

SHA

```
SHA(string)
```

使用安全哈希算法计算字符串的160位校验和，并返回40位的十六进制数字。它是比MD5()函数更安全的加密算法。

例如：

```
mysql> SELECT SHA('how many more');
+------------------------------------------+
| SHA('how many more')                     |
+------------------------------------------+
| 38ccbb8146b0673fa91abba3239829af6f3e5a6b |
+------------------------------------------+
```

SHA1

```
SHA1(string)
```

SHA()的同义词。

SYSTEM_USER

```
SYSTEM_USER()
```

SESSION_USER()的同义词。

USER

```
USER()
```

SESSION_USER()的同义词。

VERSION

```
VERSION()
```

以字符串形式返回MySQL服务器的版本，如果允许日志，则使用-log附加参数。例如：

```
mysql> SELECT VERSION();
+-----------------+
| VERSION()       |
+-----------------+
| 4.0.3-beta-log  |
+-----------------+
```

附录C PHP API

PHP是与MySQL一起使用的最流行的语言中的一种，特别是在Web环境中。本附录列出了所有对MySQL起作用的PHP功能——包括一些还没有在PHP的发布版本中包括的内容。

PHP配置参数

PHP配置文件称为php.ini，其中有许多专门为MySQL使用的参数。这些参数包括：

mysql.allow_persistent boolean 如果准许到MySQL的持续连接，则此参数值为On。默认值为On。通常并不将其关闭。

mysql.max_persistent integer 每个进程可持续连接的最大数量。默认值为-1（没有限制）。

mysql.max_links integer 每个进程的所有类型的MySQL连接的最大数量。默认值为-1（没有限制）。

mysql.default_port string 连接到MySQL的默认TCP端口号。如果没有默认值设置，那么PHP将使用环境变量MYSQL_TCP_PORT。UNIX还可以按顺序使用/etc/services中的mysql-tcp条目或编译时间（compile-time）MYSQL_PORT常数。默认值是NULL。

mysql.default_socket string 用来连接MySQL的默认UNIX套接字名。默认值是NULL。

mysql.default_host string 用来连接MySQL的默认主机名。安全模式（Safe-mode）将使这个参数失效 。默认值是NULL。

mysql.default_user string 用来连接MySQL的默认用户名。在安全模式下不适用，并且此参数在安全模式下失效。默认值为NULL。

mysql.default_password string 用来连接MySQL的默认密码。在安全模式下不适用，并且此参数在安全模式下失效。默认值为NULL。不推荐在这里设置密码。

mysql.connect_timeout integer 连接超时时间，以秒为单位。

PHP MySQL函数

PHP函数与C API函数密切相关。这里列出的函数是PHP本身携带的函数。同样，PHP中的许多库都为与MySQL接口增加了抽象层，它们中最重要的有ADODB、PEAR、Metabase和旧的PHPLib。

mysql_affected_rows

```
int mysql_affected_rows([resource mysql_connection])
```

被已经对数据进行修改的上一条语句（INSERT、UPDATE、DELETE、LOAD DATA、REPLACE）影响的行的数量，如果查询失败，那么返回值为－1。需要记住，对于在原始表中的每个被影响的行（一是DELETE，一是INSERT）REPLACE INTO将影响两行。如果没有指定数据库连接，那么将使用最近建立的连接。

如果操作者正在使用事务，在调用COMMIT之前调用mysql_affected_rows()。

返回由SELECT语句返回的行的数量，可以使用mysql_num_rows()。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// update an unknown number of fields in the database table
mysql_query("UPDATE $table SET field1=2 WHERE field1=3");

// store the number of rows updated
$num_rows_updated =  mysql_affected_rows();
```

mysql_change_user

```
boolean mysql_change_user(string username, string password
 [, string database [, resource mysql_connection]])
```

将当前的MySQL用户（已经登录的用户）改变为另一个用户（指定用户的用户名和密码）。操作者还可以同时改变数据库或指定一个新的连接，否则将使用当前的连接和数据库。如果成功则返回true，否则返回false，这时继续使用当前的用户和详细资料。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

$change_succeeded = mysql_change_user($new_user, $new_password, $database,
   $connect);
```

mysql_client_encoding

```
int mysql_client_encoding ([resource mysql_connection])
```

如果没有指定字符集，那么返回指定连接或者最近建立的开放式连接的默认字符集（如latin1）。例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

$charset = mysql_client_encoding($connect);
print "The current character set is $charset";
```

mysql_close

```
boolean mysql_close([resource mysql_connection])
```

关闭指定的连接或者最近建立的开放式连接。并不关闭持续连接。例如：

```
// open a connection to the database
$connect = mysql_connect($hostname, $username, $password);
// ... do some processing
mysql_close($connect);
```

mysql_connect

```
mysql_connection mysql_connect([string hostname [, string username
 [, string password [, boolean new_connection [, int client_flags]]]]])
```

建立到MySQL服务器的连接（如果需要，由主机名、用户名和密码指定），并且返回连接标识符以便由其他函数使用。除非new_connection参数设为true，否则如果之后又有一个同样的代码调用出现，那么将返回相同的连接标识符。

主机名同样可以占用一个端口（主机名后是一个冒号，然后是端口）。

最后的参数可能是下列标记中的一个或多个，它决定了MySQL在连接时的工作基础：

mysql_client_compress 使用压缩协议。

mysql_client_ignore_space 在函数名后允许使用空格。

mysql_client_interactive 在关闭一个不活动的连接之前，等待interactive_timeout的值，而不是wait_timeout mysqld变量。

mysql_client_ssl 使用SSL协议。

例如：

```
// set connection settings (should usually be done outside the script)
$hostname = "localhost:3306";
$username = "guru2b";
$password = "g00r002b";

// open a connection to the database
$connect = mysql_connect($hostname, $username, $password, MYSQL_CLIENT_COMPRESS);
```

mysql_create_db

```
boolean mysql_create_db (string database [, resource mysql_connection])
```

在服务器上利用指定的连接建立一个新的数据库，如果没有指定连接，则使用最近建立的开放式连接。如果成功则返回true，否则返回false。

新建立的数据库并不是活动的数据库。操作者需要使用函数mysql_select_db()激活它。

这个函数代替了函数mysql_createdb()，但它仍然有效。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

if (mysql_create_db("new_db", $connect)) {
  print "Database new_db successfully created";
}
else {
  print "Database new_db was not created";
}
```

mysql_data_seek

```
boolean mysql_data_seek (resource query_result, int row)
```

将与查询结果相关联的内部行指针（0是第一行）移动到新的位置。被检索的下一行（例如，从mysql_fetch_row()或mysql_fetch_array()开始）将是被指定的行。

如果移动成功则返回true，否则返回false（通常由于查询结果没有任何或有很多相关联的行）。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);

// set and run the query
$sql = "SELECT field1,field2 FROM table1";
$result = mysql_query($sql, $connect);

// number of rows returned
$x = mysql_num_rows($result);

// if the tenth row exists, go straight to it
if ($x >= 10) {
  mysql_data_seek($result, 9);
}

// return the data from the 10th row
$row = mysql_fetch_array($result);
print "Field1: " . $row["field1"] . "<br>\n";
print "Field2: " . $row["field2"];
```

mysql_db_name

```
string mysql_db_name (resource query_result, int row[, mixed unused])
```

返回数据库的名字。查询结果将会从较早的mysql_list_dbs()函数的调用给出。行指定查询结果集（从0开始）中返回的项。

mysql_num_rows()函数返回mysql_list_dbs()的数据库数量。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// return the list of databases on the connection
$result = mysql_list_dbs($connect);

// loop through the results, returning the database names one by one
for ($i=0; $i < mysql_num_rows($result); $i++) {
     print mysql_db_name($result, $i) . "<br>\n";
}
```

mysql_db_query

```
query_result mysql_db_query ( string database, string query
 [, resource mysql_connection])
```

如果查询成功则返回查询的结果，否则返回false。查询由指定的连接送往指定的数据库（或者如果没有指定连接，则使用最近建立的开放式连接）。

此函数已经不再建议使用，替代它的是mysql_select_db()和mysql_query()。

mysql_drop_db

```
boolean mysql_drop_db(string database [, resource mysql_connection])
```

撤销指定连接的指定数据库，或者如果没有指定连接则使用最近建立的开放式连接。如果成功则返回true。如果数据库不能撤销，则返回false。

此函数和旧的mysql_dropdb()函数都不建议使用。操作者应该使用函数撤销mysql_query()数据库。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// drop the old_db database
if (mysql_drop_db("old_db", $connect)) {
  print "Database old_db is gone";
else {
  print "Database old_db could not be dropped";
}
```

mysql_errno

```
int mysql_errno([resource connection])
```

返回最近执行的MySQL函数的错误号，如果没有错误，则返回0。使用指定的连接（或者如果没有指定的连接，则使用最近建立的开放式连接）。

在成功执行了任意MySQL相关的函数后，此函数将返回0，这不包括mysql_error()和mysql_errno()，它们的值不变。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// attempt to use a database that you've just dropped
mysql_select_db("old_db", $connect);

// Displays the error code - 1049
if (mysql_errno()) {
  print "MySQL has thrown the following error: ".mysql_errno();
}
```

mysql_error

```
string mysql_error([resource mysql_connection])
```

当最近执行的MySQL函数的出错时，返回出错信息的文本内容，如果没有错误，则返回空字符串（''）。它使用指定的连接，如果没有指定连接，则使用最近建立的开放式连接。

在任意MySQL相关的函数成功执行后，此函数将返回空串，这不包括mysql_error()和mysql_errno()，它们的值不变。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// attempt to use a database that you've just dropped
mysql_select_db("old_db", $connect);

// Displays the error text - Unknown database 'old_db'
if (mysql_errno()) {
  print "MySQL has thrown the following error: ".mysql_error();
}
```

mysql_escape_string

```
string mysql_escape_string (string stringname)
```

返回一个字符串，其中包含所有可以导致查询退出的字符（在它们的前面放有反斜线）。被转义的这些字符包括空（\x00）、换行（\n）、回车（\r）、反斜线（\）、单引号（'）、双引号（"）和Ctrl+Z（\x1A）。

百分号和下划线不被转义。

这使得操作者可以放心地使用查询。无论何时用户输入被用于查询中，此函数应该被用来使得查询更加安全可靠。操作者还可以使用相比之下略差的addslashes()函数。

例如：

```
// orginal unsafe string
$field_value = "Isn't it true that the case may be";
```

```
// escape the special characters
$field_value = mysql_escape_string($field_value);

// now it's safe and displays: Isn\'t it true that the case may be
print "$field_value";
```

mysql_fetch_array

```
array mysql_fetch_array (resource query_result [, int array_type])
```

在像mysql_query()这样的函数返回的查询结果中，根据其中的一行给出字符串队列，如果没有可用的行，则返回false。根据内部行指针的位置返回行信息，内部行指针随后加一（在启动一个查询后，行指针立即从0开始）。

第二个参数指定如何返回数据。如果队列类型被设成MYSQL_ASSOC，那么数据将以结合队列（associative array）的形式给出（与使用mysql_fetch_assoc()函数是相同的）。如果队列的类型被设置成MYSQL_NUM，那么数据将以数字队列的形式给出（与使用mysql_fetch_row()函数是相同的）。如果没有指定第三个参数，MYSQL_BOTH则为默认值，并且它使操作者能够以结合或数字队列的形式访问数据。

结合队列作为一个键，只取队列的名字（丢弃任何表名字的前缀）。如果有重复的字段名，那么操作者需要使用匿名，否则后者将取代前者。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);

// set and run the query
$sql = "SELECT field1,field2 FROM table1";
$result = mysql_query($sql, $connect);

// return the data in both associative and numeric arrays (default)
// loop through the rows, printing the data
while ($row = mysql_fetch_array($result)) {
  print "Field1: ".$row["field1"]."<br>\n";
  print "Field2: ".$row["field2"]."<br>\n";
}
```

mysql_fetch_assoc

```
array mysql_fetch_assoc (resource query_result)
```

在像mysql_query()这样的函数返回的查询结果中，根据其中的一行给出字符串队列，如果查询失败或没有可用的行，则返回false。根据内部行指针的位置返回行信息，内部行指针随后加一（在启动一个查询后，行指针立即从0开始）。

数据以结合队列的形式返回，结合队列作为一个键，只取队列的名字（丢弃任何表名字的前缀）。如果有重复的字段名，那么操作者需要使用匿名，否则后者将取代前者。此函

数与带有MYSQL_ASSOC参数的函数mysql_fetch_array()是相同的。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);

// set and run the query
$sql = "SELECT field1,field2 FROM table1";
$result = mysql_query($sql, $connect);

// return the data in associative array format and loop through
// the result, printing the row values
while ($row = mysql_fetch_assoc($result)) {
  print "Field1: ".$row['field1']."<br>\n";
  print "Field2: ".$row['field2']."<br>\n";
}
```

mysql_fetch_field

```
object mysql_fetch_field(resource query_result [, int offset ])
```

在像mysql_query()这样的函数返回的查询结果中，根据其中的一行给出包含某个字段信息的对象。如果没有指定偏移量，那么将返回下一个未搜索的字段（因此操作者可以多次调用这个参数，获得所有字段的信息）。否则将根据偏移量决定返回的结果（对于第一个字段，偏移量是0）。

对象的属性如下：

name 字段名；

table 字段所属的表的名字；

max_length 字段的最大长度；

not_null 如果字段不包含“空（null）”，则返回1；

primary_key 如果字段是主键，则返回1；

unique_key 如果字段是惟一键，则返回1；

multiple_key 如果字段并非惟一键，则返回1；

numeric 如果字段是数字的，则返回1；

blob 如果字段是BLOB，则返回1；

type 字段的类型；

unsigned 如果字段无符号，则返回1；

zerofill 如果字段填满0，则返回1。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);
```

```
// return a list of all fields in database1.table1
$result = mysql_list_fields("database1", "table1");

// loop through the fields and display the field name, type
// and maximum length
while ($row = mysql_fetch_field($result)) {
  $max_length = $row->max_length;
  $name = $row->name;
  $type = $row->type;
  print "Name:$name <br>\n";
  print "Type:$type <br>\n";
  print "Maximum Length:$max_length <br><br>\n\n";
}
```

mysql_fetch_lengths

```
array mysql_fetch_lengths(resource query_result)
```

在查询结果中取得的最后一行中，返回包含每个字段长度的队列（结果的长度，不是最大长度）。如果查询未成功，则返回**false**。使用函数**mysql_field_len()**将返回字段的最大长度。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);

// set and run the query
$sql = "SELECT field1,field2 FROM table1";
$result = mysql_query($sql, $connect);

// return the data in associative array format and loop through
// the result, retrieving the length of the fields, and printing
// the row values and lengths
while ($row = mysql_fetch_assoc($result)) {
  $lengths = mysql_fetch_lengths($result);
  print "Field1: ".$row["field1"]."Length: ".$lengths[0]."<br>\n";
  print "Field2: ".$row["field2"]."Length: ".$lengths[1]."<br>\n";
}
```

mysql_fetch_object

```
object mysql_fetch_object(resource query_result)
```

返回一个对象，其属性根据像**mysql_query()**这样的函数返回的查询结果中的一行得出。根据内部行指针的位置返回行信息，内部行指针随后加一（在启动一个查询后，行指针立即从0开始）。

依据查询中字段的名字（或匿名）得出每个对象的属性。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);

// set and run the query
$sql = "SELECT field1,field2 FROM table1";
$result = mysql_query($sql, $connect);

// loop through the rows returning each as an object
// and display the fields
while ($row = mysql_fetch_object($result)) {
  print "Field1: ".$row->field1."<br>\n";
  print "Field2: ".$row->field2."<br>\n";
}
```

mysql_fetch_row

```
array mysql_fetch_row(resource query_result)
```

在像mysql_query()这样的函数返回的查询结果中，根据其中的一行给出字符串队列，如果查询失败或没有可用的行，则返回false。根据内部行指针的位置返回行信息，内部行指针随后加一（在启动一个查询后，行指针立即从0开始）。

数据以数字队列的形式返回（与使用带有MYSQL_NUM参数的函数mysql_fetch_array()是相同的）。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);

// set and run the query
$sql = "SELECT field1,field2 FROM table1";
$result = mysql_query($sql, $connect);

// loop through the rows returning each as a numeric array
// and display the fields
while ($row = mysql_fetch_row($result)) {
  print "Field1: ".$row[0]."<br>\n";
  print "Field2: ".$row[1]."<br>\n";
}
```

mysql_field_flags

```
string mysql_field_flags(resource query_result, int offset)
```

在像mysql_query()这样的函数返回的查询结果中，根据其中的一行给出包含指定字段标记的字符串。偏移量决定了要检查的字段（对第一个字段，偏移量是0）。

标记包括not_null、primary_key、unique_key、multiple_key、blob、unsigned、zerofill、binary、enum、auto_increment和timestamp。

旧的函数mysql_fieldflags()完成相同的功能，但不推荐使用。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);

// set and run the query
$sql = "SELECT field1,field2 FROM table1";
$result = mysql_query($sql, $connect);

// Display the properties for fields 1 and 2
print "Field1 flags: ".mysql_field_flags($result, 0)."<br>\n";
print "Field1 flags: ".mysql_field_flags($result, 1)."<br>\n";
```

mysql_field_len

```
int mysql_field_len (resource query_result, int offset)
```

在像mysql_query()这样的函数返回的查询结果中，根据其中的一行给出指定字段的最大长度（由数据库结构决定）。偏移量决定被查询的字段（从0开始）。

旧的函数mysql_fieldlen()功能相同，但不推荐使用。

使用函数来决定返回字段的指定长度。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);

// set and run the query
$sql = "SELECT field1,field2 FROM table1";
$result = mysql_query($sql, $connect);

// Display the properties for fields 1 and 2
print 'Field1 maximum length: '. mysql_field_len($result, 0). "<br>\n";
print 'Field1 maximum length: '. mysql_field_len($result, 1). "<br>\n";
```

mysql_field_name

```
string mysql_field_name(resource query_result, int offset)
```

在像mysql_query()这样的函数返回的查询结果中，根据其中的一行给出指定字段的名字。偏移量决定被查询的字段（从0开始）。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);

// set and run the query
$sql = "SELECT * FROM table1";
$result = mysql_query($sql, $connect);

// loop through the fields and display the name
for($i=0; $i < mysql_num_fields($result); $i++) {
  print "Field name: ".mysql_field_name($result, $i). "<br>\n";
}
```

mysql_field_seek

```
boolean mysql_field_seek(resource query_result, int offset)
```

根据偏移量（第一个字段的偏移量为0），将内部指针移动到查询结果中新的一个字段。下一次对于函数mysql_fetch_field()的调用将从此偏移量开始。由于操作者可以直接通过函数mysql_fetch_field()移动指针，因此这个函数并不是很有用。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);

// set and run the query
$sql = "SELECT * FROM table1";
$result = mysql_query($sql, $connect);

// jump to the 2nd field
mysql_field_seek($result, 1);

$field = mysql_fetch_field($result);
print "The name of the 2nd field is: " . $field->name;
```

mysql_field_table

```
string mysql_field_table(resource query_result, int offset)
```

根据引用字段的偏移量（从0开始），返回查询结果中的表的名字，如果出现错误，则返回false。函数mysql_fieldtable()虽然不推荐使用，但功能是相同的。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);

// set and run the query
$sql = "SELECT field1,field2 FROM table1,table2 WHERE field1=field2";
$result = mysql_query($sql, $connect);

// Get the name of the table for field1 (offset 0)
echo "field 1 belongs to the table: ".mysql_field_table($result, 0);
```

mysql_field_type

```
string mysql_field_type(resource query_result, int offset)
```

根据偏移量（从0开始），返回查询结果中的字段类型，如果出现错误，则返回false。举例来说，字段类型包括int、real、string和blob。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);

// set and run the query
$sql = "SELECT field1,field2 FROM table1,table2 WHERE field1=field2";
$result = mysql_query($sql, $connect);

for($i=0;$i<mysql_num_fields($result);$i++) {
  echo "Field $i is of type: ".mysql_field_type($result, $i) . '<br>\n';
}
```

mysql_free_result

```
boolean mysql_free_result(resource query_result)
```

释放所有query_result使用的内存，使其可被再利用。如果成功，则返回true，否则为false。即使不调用此函数，在脚本的结尾处内存也会自动释放。函数mysql_freeresult()虽然不推荐使用，但功能是相同的。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);

// set and run the query
$sql = "SELECT field1,field2 FROM table1";
$result = mysql_query($sql, $connect);

// loop through the rows returning each as a numeric array
// and display the fields
while ($row = mysql_fetch_row($result)) {
  print "Field1: ".$row[0]."<br>\n";
  print "Field2: ".$row[1]."<br>\n";
}

// free the resources associated with the query
mysql_free_result($result);
```

mysql_get_client_info

```
string mysql_get_client_info()
```

返回包含MySQL客户端库版本（例如，4.0.2）的一个字符串。

例如：

```
// displays - Client library version is: 4.0.2 (for example)
print "Client library version is: ".mysql_get_client_info();
```

mysql_get_host_info

```
string mysql_get_host_info([resource mysql_connection])
```

返回包含连接信息（例如，“通过UNIX套接字建立连接的本地主机Localhost”）的一个字符串。信息来自于指定的连接（如果没有指定连接，那么使用最近建立的开放式连接）。

例如：

```
// displays - Type of connection: Localhost via UNIX socket
// (for example)
print "Type of connection: ".mysql_get_host_info();
```

mysql_get_proto_info

```
int mysql_get_proto_info([resource mysql_connection])
```

返回包含了由连接使用的协议版本的整数（如10）。信息来自于指定的连接（如果没有指定连接，那么使用最近建立的开放式连接）。

例如：

```
// displays - Protocol version: 10 (for example)
print "Protocol version: ".mysql_get_proto_info();
```

mysql_get_server_info

```
string mysql_get_server_info([resource mysql_connection])
```

返回包含MySQL服务器版本的字符串（例如4.0.3）。信息来自于指定的连接（如果没有指定连接，那么使用最近建立的开放式连接）。

例如：

```
// displays - Server version: 4.0.3-beta-log (for example)
print "Server version: ".mysql_get_server_info();
```

mysql_info

```
string mysql_info ( [resource mysql_connection])
```

返回包含最近查询的详细信息的字符串。详细信息包括记录，匹配的行、修改和告警。信息来自于指定的连接（如果没有指定连接，那么使用最近建立的开放式连接）。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);

// set and run the query
$sql = "UPDATE table1 set field1 = 2 WHERE field2=3";
$result = mysql_query($sql, $connect);

// displays:
// Query info: String format: Rows matched: 19 Changed: 19 Warnings: 0
//(for example)
print "Query info: ".mysql_info();
```

mysql_insert_id

```
int mysql_insert_id([resource mysql_connection])
```

返回一个整数，它包含连接的最新AUTO_INCREMENT值，如果失败，则返回false。信息来自于指定的连接（如果没有指定连接，那么使用最近建立的开放式连接）。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);
```

```
// set and run the query
$sql = "INSERT INTO table1(field1, field2) VALUES(3,4)";
$result = mysql_query($sql, $connect);

// Displays: AUTO_INCREMENT value: 10 (for example)
print "AUTO_INCREMENT value: ".mysql_insert_id();
```

mysql_list_dbs

```
query_result mysql_list_dbs([resource mysql_connection])
```

返回指向连接中可用数据库列表的资源。信息来自于指定的连接（如果没有指定连接，那么使用最近建立的开放式连接）。结果可以由mysql_db_name()或mysql_result()的函数处理。

函数mysql_listdbs()虽然不推荐使用，但是功能是相同的。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// return the list of databases on the connection
$result = mysql_list_dbs($connect);

// loop through the results, returning the database names one by one
for ($i=0; $i < mysql_num_rows($result); $i++) {
     print mysql_db_name($result, $i) . "<br>\n";
}
```

mysql_list_fields

```
query_result mysql_list_fields(string database, string table
 [, resource mysql_connection])
```

返回指向给定数据库和表的字段列表的资源。信息来自于指定的连接（如果没有指定连接，那么使用最近建立的开放式连接）。

函数mysql_listfields()虽然不推荐使用，但是功能是相同的。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// return a list of all fields in database1.table1
$result = mysql_list_fields("database1", "table1");

// loop through the fields and display the field name, type
// and maximum length
while ($row = mysql_fetch_field($result)) {
  $max_length = $row->max_length;
  $name = $row->name;
  $type = $row->type;
```

```
  print "Name:$name <br>\n";
  print "Type:$type <br>\n";
  print "Maximum Length:$max_length <br><br>\n\n";
}
```

mysql_list_processes

```
query_result mysql_list_processes ([resource mysql_connection])
```

返回包含当前MySQL进程列表的资源，如果失败则返回false。操作者随后可以使用像mysql_fetch_assoc()这样的函数，得出包含下面这些元素的队列：Id、Host、db、Command和Time。信息来自于指定的连接（如果没有指定连接，那么使用最近建立的开放式连接）。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// return all the processes
$result = mysql_list_processes($connect);

// loop through the rows displaying the various process elements
while ($row = mysql_fetch_assoc($result)){
  print $row["Id"];
  print $row["Host"];
  print $row["db"];
  print $row["Command"];
  print $row["Time"];
  print "<br>\n"
}
```

mysql_list_tables

```
query_result mysql_list_tables(string database, resource mysql_connection])
```

返回指向给定数据库中的表的列表资源，如果失败则返回false。信息来自于指定的连接（如果没有指定连接，那么使用最近建立的开放式连接）。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// returns the list of tables
$result = mysql_list_tables("database1");

// loops through the rows of tables, and displays the names
for($i=0; i < mysql_num_rows($result); $i++) {
  print "Table name: ".mysql_tablename($result, $i)."<br>\n";
}
```

mysql_num_fields

```
int mysql_num_fields(resource query_result)
```

返回包含查询结果中字段数量的整数，如果出现错误，则返回NULL。

函数mysql_numfields()虽然不推荐使用，但是功能是相同的。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// return a list of all fields in database1.table1
$result = mysql_list_fields("database1", "table1");

// Displays: Num fields in database1: 6 (for example)
print "Num fields in database1: ".mysql_num_fields($result);
```

mysql_num_rows

```
int mysql_num_rows(resource query_result)
```

返回包含查询结果中行数量的整数，如果出现错误则返回NULL。如果是使用函数mysql_affected_rows()得出的查询结果，则此函数不能运行。操作者应该使用mysql_affected_rows()得出查询修改过的数据行的数量（例如，在执行了INSERT或UPDATE后）。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// return the list of databases on the connection
$result = mysql_list_dbs($connect);

// loop through the results, returning the database names one by one
for ($i=0; $i < mysql_num_rows($result); $i++) {
    print mysql_db_name($result, $i) . "<br>\n";
}
```

mysql_pconnect

```
mysql_connection mysql_pconnect([string hostname
 [, string username [, string password [, int client_flags]]]])
```

建立与MySQL服务器（如果需要，可以通过主机名、用户名和密码指定）的永久连接（可以重复使用的连接），并返回可由其他函数使用的连接标识符。如果已经存在这样一条连接，则将重复使用此连接。最终的参数可以是下列一个或多个标记，它决定了连接时MySQL的操作内容。

mysql_client_compress 使用压缩协议

mysql_client_ignore_space 允许函数名后存在空格

mysql_client_interactive 在关闭不活动的连接之前，等待mysqld变量interactive_timeout的值，而不是wait_timeout的值

mysql_client_ssl 使用SSL协议

在经过wait_timeout（mysqld的一个变量）秒后，或生成连接的进程关闭后，MySQL关闭永久连接。举例来说，Web服务器进程可以对多个脚本重复使用相同的连接，但是一旦Web服务器连接关闭，则连接也将关闭。

例如：

```
// set connection settings (should usually be done outside the script)
$hostname = "localhost:3306";
$username = "guru2b";
$password = "g00r002b';

// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);
```

mysql_ping

```
boolean mysql_ping ([resource mysql_connection])
```

如果MySQL服务器启动，则返回true，否则返回false。ping试图使用指定的连接（如果没有指定连结，则使用最近建立的开放式连接）。如果ping失败，那么脚本将试图利用相同的参数重新进行连接。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// time passes...

if (mysql_ping()) {
  print "Still connected";
}
else {
  print "Connection lost";
}
```

mysql_query

```
query_result mysql_query(string query[, resource mysql_connection
 [, int result_mode]])
```

返回查询结果（如果这个查询可以生成结果，如SELECT或DESCRIBE），如果查询没有生成结果，但却成功完成（如DELETE或UPDATE），那么返回true。如果查询失败，则返回false。查询被送往一个指定的使用指定连接的数据库（如果没有指定连接，那么使用最近建立的开放式连接）。

可选的result_mode参数可以是MYSQL_USE_RESULT中的一个，它将使结果不被缓存，如同使用参数mysql_unbuffered_query()或MYSQL_STORE_RESULT（默认情况下）。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);

// set and run the query
$sql = "SELECT field1,field2 FROM table1";
$result = mysql_query($sql, $connect);

// return the data in associative array format and loop through
// the result, printing the row values
while ($row = mysql_fetch_assoc($result)) {
  print "Field1: ".$row["field1"]."<br>\n";
  print "Field2: ".$row["field2"]."<br>\n";
}
```

mysql_real_escape_string

```
string mysql_real_escape_string (string stringname [, resource mysql_connection])
```

返回一个字符串，其中包含所有可以导致查询转义的字符（在它们的前面放有反斜线）。被转义的这些字符包括空（\x00）、换行（\n）、回车（\r）、反斜线（\）、单引号（'）、双引号（"）和Ctrl+Z（\x1A）。百分号和下划线不被转义。

这使得操作者可以放心地使用查询。与mysql_escape_string()不同，此函数将考虑当前字符集。

例如：

```
// orginal unsafe string
$field_value = "Isn't it true that the case may be";

// escape the special characters
$field_value = mysql_real_escape_string($field_value);

// now it's safe and displays: Isn\'t it true that the case may be
print "$field_value";
```

mysql_result

```
mixed mysql_result(resource query_result, int row [, mixed field_specifier])
```

返回查询结果中单一字段的内容。如果在查询中提供field_specifier，那么它可以是一个偏移量（从0开始）或字段名，带有或没有表的分类符（换句话说，tablename.fieldname或只是fieldname）。如果没有提供字段分类符，那么将返回第一个字段。

此函数要比返回整个行的函数慢得多，如mysql_fetch_row()和mysql_fetch_array()，因此可以使用其中之一代替此函数。除此之外，不要把它的功能与返回整个行的函数的功能相混淆。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);

// return the average of field1
$sql = "SELECT AVG(field2) FROM table1";
$result = mysql_query($sql, $connect);

// display the average value of this field
print "Field2 average: ".mysql_result($result, 0);
```

mysql_select_db

```
boolean mysql_select_db(string database [, resource mysql_connection])
```

改变当前的数据库到指定的数据库。使用指定的连接（如果没有指定连接，则使用最近建立的开放式连接）。如果没有建立连接，它将试图调用**mysql_connect()**，在不带任何参数的情况下建立连接。如果成功，则返回**true**，否则返回**false**。

虽然不推荐使用**mysql_selectdb()**函数，但其功能是相同的。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);
```

mysql_stat

```
string mysql_stat ([resource mysql_connection])
```

返回包括服务器状态的字符串，其中包含已运行的时间（**uptime**）、线程（**thread**）、问题、缓慢的查询、打开的数据库、刷新表、开放式表和平均每秒进行的查询。利用指定的连接（如果没有指定连接，则使用最近建立的开放式连接）。

例如：

```
// displays (for example):
// Uptime: 109  Threads: 2  Questions: 199  Slow queries: 1  Opens: 4
// Flush tables: 1  Open tables: 2  Queries per second avg: 1.826
print "Server Status: ".mysql_stat();
```

mysql_tablename

```
string mysql_tablename(resource query_result, int row)
```

返回由函数**mysql_list_tables()**基于行（从0开始）得出的查询结果中的表名字。如果出现错误，则返回**false**。操作者可能得到**mysql_num_rows()**查询结果中行的数目。此函数实际

上是mysql_result()的匿名函数，但是由于它的名字针对表的名字而不同，在其他场合使用时会发生混淆，因此对于同等功能的其他函数而言，它并不是一个很好的程序。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// returns the list of tables
$result = mysql_list_tables("database1");

// loops through the rows of tables, and displays the names
for($i=0; $i < mysql_num_rows($result); $i++) {
  print "Table name: ".mysql_tablename($result, $i)."<br>\n";
}
```

mysql_thread_id

```
int mysql_thread_id ([resource mysql_connection])
```

返回包含当前线程ID的整数。

例如：

```
// displays - Thread id: 2394 (for example)
print "Thread id: ".mysql_thread_id();
```

mysql_unbuffered_query

```
query_result mysql_unbuffered_query(string query
 [, resource mysql_connection [, int result_mode]])
```

返回未缓存的查询结果（如果查询可以生成结果，如SELECT或DESCRIBE），如果查询没有生成结果，但是成功执行了，那么将返回true。如果查询失败，则返回false。查询使用指定的连接（如果没有指定连接，则使用最近建立的开放式连接）。

它和mysql_query()之间的区别在于，由于结果不进行缓存，因此它使用较少的内存。并且一旦第一行已经找到，操作者就可以利用查询结果开始工作了。缺点是操作者不能使用mysql_num_rows()。它主要用于大型的、缓慢的查询中。

可选的result_mode参数可以是MYSQL_USE_RESULT中之一（默认情况下）或MYSQL_STORE_RESULT，它使得结果不被缓存，就像mysql_query()一样。

例如：

```
// open a persistent connection to the database
$connect = mysql_pconnect($hostname, $username, $password);

// select the database
mysql_select_db("database1", $connect);

// set and run the query
$sql = "SELECT field1,field2 FROM table1";
$result = mysql_unbuffered_query($sql, $connect);
```

```
// return the data in both associative and numeric arrays (default)
// loop through the rows, printing the data
while ($row = mysql_fetch_array($result)) {
  print "Field1: ".$row["field1"]."<br>\n";
  print "Field2: ".$row["field2"]."<br>\n";
}
```

附录D Perl DBI

在Perl语言中，与数据库（不仅仅是MySQL）连接的推荐方式是用DBI模块。这是一个以相同方式访问不同数据库的通用接口。除了DBI模块，还需要DBD模块。每一个DBD模块针对一个指定的数据库。这里需要与MySQL相关的DBD模块。

说明： 为了安装支持MySQL的Perl DBI，需要DBI、DBD-mysql、Data-Dumper和File-Spec模块。可以从www.mysql.com/CPAN下载最新的MySQL版本。完全安装指南随软件一起带来。

在整个附录中，变量名的约定如下：

$dbh 数据库句柄对象，由connect()或connect_cached()程序返回。

$sth 语句句柄对象，由其他的prepare()程序返回。

$drh 驱动程序程序句柄对象（在应用程序中很少使用）。

$h 数据库、语句或驱动程序程序的句柄。

$rc 布尔值返回的代码（成功返回为true，出现错误为false）。

$rv 排序的返回值，通常是整数。

@ary 数值型数组，通常是从查询中返回的一行数据。

$rwos 被处理的行数，如果不知道为－1。

$fh 文件句柄。

undef NULL或未定义的值。

\%attributes 属性值的混编引用，被方法用于不同目的。

为了使用DBI，需要在脚本开始处加载DBI模块，如下所示：

```
use DBI;
```

然后，需要返回数据库句柄，通常用connect() DBI类的程序。然后，数据库句柄访问可以运行查询的方法，并返回结果，通常返回语句句柄。

DBI类的方法

DBI类的方法把类作为一个整体而应用。最重要的是connect()方法，它在成功时将返回一个数据库句柄。

available_drivers

```
@ary = DBI->available_drivers[($quiet)];
```

返回可获得的驱动程序程序的列表（DBD模块）。如果有驱动程序程序重名，会给出警告信息。设置参数$quiet值为true，能消除警告声。

connect

```
$dbh = DBI->connect($datasource, $username, $password [, \%attributes]);
```

使用指定的数据源、用户名和密码，创建数据库的连接，并返回数据库句柄。数据源由DBI驱动程序（在这种情况是dbi:mysql）、数据库名、可选主机名（如果未指定，为localhost）、可选端口名（如果未指定，为3306）和一些修改器组成，每个参数都用分号隔开。

```
mysql_read_default_file=filename
```

指定文件被作为选项文件（MySQL配置文件，通常是服务器上的my.ini或my.cnf）使用。

```
mysql_read_default_group=groupname
```

当阅读一个选项文件时，要使用的默认组是[client]。这将改变组为[groupname]。

下面的选项使客户机与服务器之间的通信信息被压缩：

```
mysql_compression=1
```

下面的选项指定了连接服务器的UNIX套接字：

```
mysql_socket=/path/to/socket
```

如果没有指明，可选的用户名和密码将采用DBI_USER和环境变量DBI_PASS的值。

如果连接失败了，它将返回undef并设置$DBI:err和$DBI:errstr。

可以使用\%attributes参数设置不同的属性，例如AutoCommit（推荐）、RaiseError和PrintError。

例如：

```
my $hostname = 'localhost';
my $database = 'firstdb';
my $username = 'guru2b';
my $password = 'g00r002b';

#Connect to the database
my $dbh = DBI->connect("dbi:mysql:$database:$hostname", $username,?
 $password, (AutoCommit => 0, RaiseError => 1, PrintError => 0))
or die $DBI::errstr;
```

connect_cached

```
$dbh = DBI->connect_cached($data_source, $username, $password[ , \%attributes])
```

除了把数据库句柄的详细信息也存入混编数组外，该变量与connect()一样。如果仍然有效的话，相同的数据库句柄被用于进一步调用相同的connect_cached()。CachedKids属性（通过$dbh->{Driver}->{CachedKids}访问）中包含缓存的数据。

这仍然是相当新的方法，很可能会发生变化。它与Apache::DBI的持久连接不一样。

data_sources

```
@ary = DBI->data_sources($driver [, \%attributes]);
```

返回被命名的驱动程序的所有数据库数组（在目前情况下是mysql）。

trace

```
trace($trace_level [, $trace_filename])
```

该程序使用或禁用跟踪。当作为DBI类方法被调用时，它影响对所有数据库和语句句柄的跟踪。当作为数据库或语句句柄方法被调用时，它影响对所有句柄（和这个句柄将来的子句柄）的跟踪。

跟踪级别从0到9。0为禁止跟踪，1最适于一般情况，2是最通用的方法，其他方法增加越来越多的驱动程序程序和DBI细节信息。

默认情况下，输出转到STDERR，或被添加到指定的跟踪文件。

例如：

```
DBI->trace(2);                              # trace everything
$dth->trace(2, "/tmp/dbi_trace.out");       # trace the database handle
                                            # to /tmp/dbi_trace.out
$sth->trace(2);                             # trace the statement handle
```

可以通过设置环境变量DBI_TRACE为0～9中的任何值来激活跟踪，在这种情况中，跟踪级别被设为2，并输出到该文件中。

对所有句柄通用的DBI方法

下面的方法适用于数据库句柄、语句句柄和驱动程序句柄。它们主要被用于错误处理。

err

```
$rv  = $h->err;
```

从上一个方法中返回本地错误代码（通常为整数）。

errstr

```
$error_string = $dbh->errstr;
```

从前一个失败的调用中返回错误的字符串。

例如：

```
my $hostname = 'localhost';
my $database = 'firstdb';
my $username = 'guru2b';
my $password = 'g00r002b';
```

```
#Connect to the database
my $dbh = DBI->connect("dbi:mysql:$database:$hostname", $username,?
 $password) or die $DBI::errstr;
```

func

```
$h->func(@func_arguments, $func_name);
```

被用来调用其他非标准驱动程序方法。这采用变量数组，把方法的名字作为自变量。

这不会触发通常的错误探测机制（例如RaiseError或PrintError）或清除前面的错误（例如$DBI::err或$DBI::errstr）。

set_err

```
$rv = $h->set_err($err, $errstr [, $state, $method [, $rv]]);
```

主要是由DBI驱动程序和子类使用的新方法。它为句柄设置err、errstr和state值（通过RaiseError等允许进行错误处理）。

$method为错误字符串设置一个更有用的名字，$rv设置返回值（通常为undef）。

例如：

```
sub doodle {
 # ...try to 'doodle'
 or return $sth->set_err(1234, "Nope. Sorry. Out of luck. It all?
went wrong", undef, "doodle");
}
```

state

```
$rv  = $h->state;
```

以SQLSTATE格式返回错误代码。当驱动程序不支持SQLSTATE时，返回一个通用的S1000码。

trace

```
trace($trace_level [, $trace_filename])
```

参看前面的trace方法。

trace_msg

```
$h->trace_msg($message_text [, $minimum_level]);
```

如果跟踪被激活，将信息文本输出到跟踪文件。如果设置了最小级别（默认为1），如果跟踪级别至少是设置的级别，则只输出信息。

DBI实用函数

DBI数据包也提供DBI实用函数。

hash

```
$hash_value = DBI::hash($buffer [, $type]);
```

返回32位整数，这是一个由在缓冲区上运行的$type指定的哈希算法的结果。0类型（默认）执行Perl 5.1哈希算法，结果为负值。如果是类型1，则使用Fowler/Noll/Vo算法。

looks_like_number

```
@bool = DBI::looks_like_number(@array);
```

返回布尔数组，值为真表示原始数组的每个元素看上去都像数字，值为假则表示不像数字，对于空或未定义的元素，用undef表示。

neat

```
$neat_string = DBI::neat($value [, $maxlen]);
```

设计数值的格式和整理数值，并且引用字符串以便于显示，而不是为跳过数据库服务器。如果超过了最大长度，字符串的长度将被缩短到$maxlen-4，并把省略号（...）添加到字符串尾部。如果没有指定$maxlen（默认值为400），将使用$DBI::neat_maxlen。

neat_list

```
$neat_string = DBI::neat_list(\@listref [, $maxlen [, $field_sep]]);
```

为列表上的每个元素调用neat()函数，并返回每个部分由$field_sep分隔开的字符串，默认使用逗号（,）分隔。

数据库句柄方法

这些方法可用于数据库句柄，因此在使用它们之前要打开连接。这些方法中的一些会返回语句句柄，然后可以用后面列出的语句句柄方法做进一步处理。

begin_work

```
$rc  = $dbh->begin_work  or die $dbh->errstr;
```

开始一个事务，并关闭AutoCommit，直到事务以commit()或rollback()结束。

column_info

```
$sth = $dbh->column_info($catalog,$schema,$table,$column);
```

返回激活的语句句柄，用以得到列的信息的实验方法。

commit

```
$rc  = $dbh->commit;
```

提交当前的事务。为了使其有效，参数AutoCommit需要被关闭。

disconnect

```
$rc = $dbh->disconnect;
```

使用指定的数据库句柄断开与数据库的连接。如果成功，返回真值，否则返回假值。

方法并不说明是退回还是提交当前打开的事务，因此在调用disconnect之前，要在应用程序里确保提交或回退事务。

do

```
$rv = $dbh->do($statement [,\%attributes [,@bind_values]]);
```

准备并执行SQL语句，返回受到影响的行数。如果没有行被影响，返回0E0（被当做真值），如果出错，则返回undef。

通常被用于不返回结果（如INSERT和UPDATE）和不使用占位符的查询操作。该方法比prepare()和execute()速度快。

它返回一个整数。

例如：

```
my $hostname = 'localhost';
my $database = 'firstdb';
my $username = 'guru2b';
my $password = 'g00r002b';

#Connect to the database
my $dbh = DBI->connect("dbi:mysql:$database:$hostname", $username,➥
 $password) or die $DBI::errstr;

$sql = "INSERT INTO customer(id,surname) VALUES(11,'Sandman')";
$dbh->do($sql);
```

foreign_key_info

```
$sth = $dbh->foreign_key_info($pk_catalog, $pk_schema, $pk_table➥
 [, $fk_catalog, $fk_schema, $fk_table]);
```

一个实验方法，为得到外部键信息而返回一个语句句柄。变量$pk_catalog、$pk_schema和$pk_table指定主键的表。变量$fk_catalog、$fk_schema和$fk_table指定外部键的表。

返回结果取决于提供的表。如果仅提供了外部键表（通过传递作为主键变量的undef），结果将包含表中所有的外部键和相关的主键。如果仅提供主键表，结果将包含表中主键和所有相关的外部键。如果两种表都提供了，结果包括来自外部键表的外部键，它引用自主键表的主键。

get_info

```
$value = $dbh->get_info( $info_type );
```

返回实现信息的实验方法。

ping

```
$rc = $dbh->ping;
```

检查数据库是否仍在运行，连接是否被激活。

prepare

```
$sth = $dbh->prepare($statement [, \%attributes])
```

返回语句句柄的引用，并准备执行的SQL语句（用execute方法）。通常准备运行返回结果的语句，如SELECT和DESCRIBE。

prepare_cached

```
$sth = $dbh->prepare_cached($statement [, \%attributes,[ $allow_active]]);
```

与prepare相同，但语句句柄被存在一个混编数组中，以便进一步调用返回同样句柄的相同变量。$allow_active变量有三种设置。默认值0，产生警告信息并在返回结果之前在语句句柄上调用finish()，而值1调用finish()，但禁止警告信息。如果值设置为2，DBI在返回语句前将不调用finish()。

primary_key

```
@key_column_names = $dbh->primary_key($catalog, $schema, $table);
```

primary_key_info方法的实验接口，该方法返回为指定的表由主键按顺序组成的字段名数组。

primary_key_info

```
$sth = $dbh->primary_key_info($catalog, $schema, $table);
```

为得到主键列的信息的实验方法。

quote

```
$quoted_string = $dbh->quote($string [,$data_type])
```

返回一个带有特殊换码符的字符串（如单引号或双引号），并加入外部引号。如果指定数据类型，Perl将用它来决定被要求的引用行为。

quote_identifier

```
$sql = $dbh->quote_identifier( $name1[ , $name2, $name3, \%attributes ]);
```

在查询中使用的在标识符（例如字段名）中转义任何特殊字符。

rollback

```
$rc  = $dbh->rollback;
```

重新开始当前的事务。为了生效，需要关闭参数AutoCommit。

selectall_arrayref

```
$ary_ref  = $dbh->selectall_arrayref($statement [, \%attributes [,@bind_values]]);
```

为了容易使用，该方法把prepare()、execute()和fetchall_arrayref()合并成一个方法。它返回一个数组的引用，该数组包含的每行数据是从查询操作中返回的数组的引用。语句还可以是一个已经准备好的语句句柄，在这种情况，方法并不执行prepare()。

可以在%attributes中设置其他变量传递给selectall_arrayref()。

selectall_hashref

```
$hash_ref = $dbh->selectall_hashref($statement, $key_field➥
 [, \%attributes [,@bind_values]]);
```

为了容易使用，该方法把prepare()、execute()和fetchall_hashref()合并成一个方法。它返回一个哈希算法的引用，其中包含从查询中返回的行的一个条目。每个字段的键由$key_field指定，数值是哈希算法的引用。语句也可以是一个已经准备好的语句句柄，在这种情况，方法跳过prepare()。

selectcol_arrayref

```
$ary_ref = $dbh->selectcol_arrayref($statement [, \%attributes [,@bind_values]]);
```

该方法通过从查询返回的所有行中取出一列或多列，合并prepare()和execute()。它返回一个数组的引用，此数组包含了每一行中列的值。语句也可以是一个已经准备好的语句句柄，在这种情况下，方法跳过prepare()。

默认状况下，它返回每行的第一列，但也能用Columns属性返回更多列，属性是一个包含列数的数组的引用。

例如：

```
# perform a query and return the two columns
my $array_ref = $dbh->selectcol_arrayref("SELECT first_name, surname➥
 FROM customer", { Columns=>[1,2] });
# create the hash from key-value pairs so $hash{$first_name} => surname
my %hash = @$array_ref;
```

selectrow_array

```
@row_ary  = $dbh->selectrow_array($statement [, \%attributes [,@bind_values]]);
```

为了容易使用，该方法把prepare()、execute()和fetchrow_array()合并成一个。它返回从

查询中返回的数据的第一行。语句也可以是一个已经准备好的语句句柄，在这种情况，方法不执行prepare()。

selectrow_arrayref

```
$ary_ref = $dbh->selectrow_arrayref($statement [, \%attributes [,@bind_values]]);
```

为了容易使用，该方法把prepare()、execute()和fetchrow_arrayref()合并成一个。语句也可以是一个已经准备好的语句句柄，在这种情况，方法不执行prepare()。

selectrow_hashref

```
$hash_ref = $dbh->selectrow_hashref($statement [, \%attributes [,@bind_values]]);
```

为了容易使用，该方法把prepare()、execute()和fetchrow_ hashref()合并成一个。它返回从查询中返回的数据的第一行。语句也可以是一个已经准备好的语句句柄，在这种情况，方法不执行prepare()。

table_info

```
$sth = $dbh->table_info($catalog, $schema, $table, $type [, \%attributes]);
```

一个实验方法，返回激活的语句句柄，以便从数据库中获得有关表和视图的信息。

tables

```
@names = $dbh->tables($catalog, $schema, $table, $type);
```

table_info()方法的实验接口，返回表的名字数组。

type_info

```
@type_info = $dbh->type_info($data_type);
```

一个实验方法，返回包含数据类型变量信息的哈希引用数组。

type_info_all

```
$type_info_all = $dbh->type_info_all;
```

一个实验方法，返回数组引用信息，包含数据库和驱动程序支持的数据类型信息。

语句句柄方法

这些方法对语句句柄起作用，通过调用数据库句柄方法而获得，如prepare()。

bind_col

```
$rc = $sth->bind_col($column_number, \$column_variable);
```

在结果中，捆绑从SELECT语句到变量的结果中的字段（从1开始）。参看bind_columns，以获取更多信息。

bind_columns

```
$rc = $sth->bind_columns(@list_of_refs_to_vars_to_bind);
```

从SELECT语句中，对每个字段调用bind_col()。引用的数目必须与字段个数匹配。例如：

```
# set RaiseError to 1 to avoid having to check each method call
$dbh->{RaiseError} = 1;
$sth = $dbh->prepare(q{SELECT first_name,surname FROM customer});
$sth->execute;
my ($first_name, $surname);
# Bind Perl variables to columns:
$rv = $sth->bind_columns(\$first_name, \$surname);
while ($sth->fetch) {
    print "$first_name $surname\n";
}
```

bind_param

```
$rv = $sth->bind_param($bind_num, $bind_value [\%attributes | $bind_type]);
```

被用于把一个数值与占位符捆绑在一起，由问号（?）指出。占位符在准备多次运行类似查询的地方使用，这些地方每次只能改变一个参数。

例如：

```
$sth = $dbh->prepare("SELECT fname, sname FROM tname WHERE sname LIKE ?");
$sth->bind_param(1, "Vusi%");  # placeholders begin from 1
$sth->execute;
```

不能使用占位符去替代表名或字段名，也不能替代单精度标量值。例如，下面占位符的用法是错误的：

```
SELECT fname, ? FROM tname WHERE sname LIKE 'Vusi%'
SELECT fname, sname FROM ? WHERE sname LIKE 'Vusi%'
SELECT fname, sname FROM tname WHERE sname IN (?)
```

可以使用可选的捆绑类型参数去指定占位符的类型。例如：

```
$sth->bind_param(1, $bind_value, {TYPE => SQL_INTEGER});
```

或等价的快捷方式（它要求引入带有use DBIqw(:sql_types);的DBI）

```
$sth->bind_param(1, $bind_value, SQL_INTEGER);
```

还可以使用参数\%attributes，如下所示：

```
$sth->bind_param(1, $bind_value, {TYPE => SQL_INTEGER});
```

返回的是一个整数。

例如：

```
my $hostname = 'localhost';
my $database = 'firstdb';
my $username = 'guru2b';
my $password = 'g00r002b';

# Connect to the database
my $dbh = DBI->connect("dbi:mysql:$database:$hostname", $username,➥
 $password) or die $DBI::errstr;

# Create the query, with a ? to indicate the placeholder
my $query = 'SELECT first_name,surname FROM customer WHERE id=?';

# Prepare the query
my $sth = $dbh->prepare($query);

# Create an array of id's to use to replace the placeholder
my @ids = (1,4,5,6);

# Loop through the array and execute the query
for(@ids) {
  $sth->bind_param(1, $_, SQL_INTEGER);
  $sth->execute();

  my( $first_name, $surname);
  $sth->bind_columns(undef, \$first_name, \$surname);

  # Loop through the rows returned and display the results
  while( $sth->fetch()) {
    print '$first_name $surname\n";
  }
}
$sth->finish();
```

bind_param_array

```
$rc = $sth->bind_param_array($p_num, $array_ref_or_value
   [, \%attributes | $bind_type])
```

被用于把一个数组捆绑到一个占位符上，该占位符被设置在准备用execute_array()方法执行的语句中。

例如：

```
# set RaiseError to 1 to avoid having to check each method call
$dbh->{RaiseError} = 1;
$sth = $dbh->prepare("INSERT INTO customer(first_name, surname) VALUES(?, ?)");
# Each array must have the same number of elements
$sth->bind_param_array(1, [ 'Lyndon', 'Nkosi', 'Buhle' ]);
$sth->bind_param_array(2, [ 'Khumalo', 'Battersby', 'Lauria' ]);
my %tuple_status;
$sth->execute_array(\%tuple_status);
```

bind_param_inout

```
$rv = $sth->bind_param_inout($p_num, \$bind_value,
 $max_len [, \%attributes | $bind_type])     or ...
```

与bind_param()相同，但可以更新数值（为了存储过程）。MySQL目前不支持这个功能。

dump_results

```
$rows = $sth->dump_results($max_len, $lsep, $fsep, $fh);
```

在为每行调用DBI::neat_list之后，从语句句柄中输出所有的行到$fh（默认为STDOUT）。$lsep是行隔离符（默认为\n），$fsep是字段隔离符，默认为逗号（,），$max_len默认为35。

execute

```
$rv = $sth->execute([@bind_values]);
```

执行准备好的语句，返回受影响的行数（对于不返回数据的查询操作，例如INSERT和UPDATE）。如果没有行受到影响，返回0E0（作为真值对待），如果出错则返回undef，使用其中一种宏指令方法去处理数据。

返回的结果为整数。

例如：

```
my $hostname = 'localhost'
my $database = 'firstdb';
my $username = 'guru2b';
my $password = 'g00r002b ;

# Connect to the database
my $dbh = DBI->connect("dbi:mysql:$database:$hostname", $username, $password);

# Create the query, with a ? to indicate the placeholder
my $query = 'SELECT first_name,surname FROM customer WHERE id=2';

# Prepare and execute the query
my $sth = $dbh->prepare($query);
$sth->execute();

my( $first_name, $surname);
$sth->bind_columns(undef, \$first_name, \$surname);

# Loop through the rows returned and display the results
while( $sth->fetch()) {
  print "$first_name $surname\n";
}

$sth->finish();
```

execute_array

```
$rv = $sth->execute_array(\%attributes[, @bind_values]);
```

对每一个用bind_param_array()设置的参数或在@bind_values中的参数，执行准备好的语句，并返回受影响的行数。

fetch

fetchrow_arrayref()的别名。

fetchall_arrayref

```
$table = $sth->fetchall_arrayref [[($slice[, $max_rows])]];
```

从每行都包含引用的作为数组的引用的查询中返回所有的行。

如果没有返回的行，它返回一个空数组引用。如果有错误发生，它返回取得的数据，直到出现任何错误。

可选的$slice可以是一个数组引用或混编引用。如果它是数组引用，方法用fetchall_arrayref去取得作为数组引用的每一行。如果指定索引，它返回字段（从0开始）。如果没有参数，或如果$slice未定义，则方法运行时就像传递了一个空数组引用。

如果$slice是混编引用，方法用fetchall_hashref去获取混编引用的每一行。返回的字段取决于混编键。混编值应该总为1。

举一些例子将使这个方法更清楚。前两个例子将返回由数组引用组成的数组的引用。首先，为了仅返回每行中的第二个字段，使用方法如下：

```
$tbl_ary_ref = $sth->fetchall_arrayref([1]);
```

为了返回每行倒数第三个和最后一个字段，使用方法如下：

```
$tbl_ary_ref = $sth->fetchall_arrayref([-3,-1]);
```

下两个例子返回混编引用数组的引用。首先，为了取得混编引用的所有行的所有字段，使用方法如下：

```
$tbl_ary_ref = $sth->fetchall_arrayref({});
```

用名字为FNAME和sname的键，去获得混编引用的每行中名为fname和sname的字段，使用方法如下：

```
$tbl_ary_ref = $sth->fetchall_arrayref({ FNAME=>1, sname=>1 });
```

如果可选的$max_rows被定义为正整数（它可以为0），返回的行数被限制为这个整数。为了返回更多的行，要再次调用fetchall_arrayref。如果没有足够的内存一次返回所有的行，但仍想有运行fetchall_arrayref的性能优势，需要使用上述方法。

fetchall_hashref

```
$hash_ref = $dbh->fetchall_hashref($key_field);
```

返回混编引用，每行中最多含有一个条目。如果查询没有返回行，则方法返回一个空混编的引用。如果有错误，它返回取得的数据，直到出现任何错误。

$key_field参数指定字段名，该字段名的值被用于保存返回的混编的键，或者该参数可以是一个对应于字段的数字（注意，从1而不是0开始）。如果键与字段不匹配（名字或数值），方法返回错误信息。

当每行键的字段值是惟一的时，通常只能使用这个方法；否则，第二个及以后的行的值将覆盖早先有同样键的值。

例如：

```
$dbh->{FetchHashKeyName} = 'NAME_lc';
$sth = $dbh->prepare("SELECT id, fname, sname FROM tname");
$hash_ref = $sth->fetchall_hashref('id');
print "The surname for id 8: $hash_ref->{8}->{sname}";.
```

fetchrow_array

```
@row = $sth->fetchrow_array;
```

利用前面的预设语句句柄，从下一行数据中返回字段值数组。然后，每行的元素能用$row[0]、$row[1]等形式访问。这会移动行指针，以便下次调用方法时返回接下来的行。

fetchrow_arrayref

```
$row_ref = $sth->fetchrow_arrayref
```

利用前面的预设语句句柄，从下一行数据中返回字段值数组引用。然后，行的元素能以$row_ref->[0]、$row_ref->[1]等形式访问。

这会移动行指针，以便下次调用方法时，返回接下来的行。

fetchrow_hashref

```
$hash_ref = $sth->fetchrow_hashref[($name)];
```

利用前面的预设语句句柄，返回一个以字段名作为键和以字段内容为数值的混编表引用。然后，行的元素能以$hash_ref->{fieldname1}、$hash_ref->{fieldname2}等形式访问。

这会移动行指针，以便下次调用方法时，返回接下来的行。

可选的name参数指定属性被分配的名字。为了方便，NAME_uc或NAME_lc（大写字母或小写字母）是被推荐的，但默认为NAME。

fetchrow_arrayref和fetchrow_array方法的速度明显更快。

finish

```
$rc = $sth->finish;
```

释放与语句句柄有关的系统资源，表明不再有数据被返回。

成功返回真值，失败为假。

rows

```
$rv = $sth->rows;
```

返回最后被SQL语句改变的行数（例如，UPDATE或INSERT语句之后），如果返回的行数不清楚，则返回－1。

对所有句柄通用的DBI属性

这些属性提供信息并能应用于所有句柄。最常见的是那些被用于错误句柄的属性，RaiseError和PrintError。例如，$h->{'RaiseError'}会导致句柄错误，从而引发异常。

Active

```
Active (boolean, read-only)
```

如果句柄对象是激活的，为真（对数据库句柄，连接到数据库；对语句句柄，取得更多的数据）。

ActiveKids (integer, read-only)

```
ActiveKids (integer, read-only)
```

当前激活的数据库句柄数目（驱动程序句柄），或当前激活的语句句柄数目（数据库句柄）。激活的意思是：对数据库句柄，连接到数据库；语句句柄，取得更多的数据。

CachedKids (hash ref)

```
CachedKids (hash ref)
```

对数据库句柄，这包含由prepare_cached()方法创建的语句句柄的混编引用。对驱动程序句柄，这包含由connect_cached()方法创建的数据库句柄混编。

ChopBlanks

```
ChopBlanks (boolean, inherited)
```

指明是否各种取值方法要从CHAR字段中删掉前面和后面的空白。如果要这样做，设为真值，否则为假（默认）。

CompatMode (boolean, inherited)

```
CompatMode (boolean, inherited)
```

模拟层，以确保驱动程序的兼容性行为。

FetchHashKeyName

```
FetchHashKeyName (string, inherited)
```

指明为得到用于混编键的字段名fetchrow_hashref()方法应使用的属性名。默认为NAME，但为了方便，应该设为NAME_lc或NAME_uc。

HandleError

```
HandleError (code ref, inherited)
```

允许创建自己的方法来处理错误。可设置它为子程序的引用，当检测到错误时（即RaiseError和PrintError将被调用时），该程序将被调用。如果子程序返回假值，正常情况下，RaiseError和PrintError将被选中。被调用的子程序使用三个参数：错误信息字符串（与RaiseError和PrintError使用的一样）、DBI句柄和失败方法返回的第一个值。

InactiveDestroy

```
InactiveDestroy (boolean)
```

为UNIX应用程序设计，该应用程序用于派生子过程。假值（默认）意味着当句柄传递超出范围时，句柄自动被破坏。真值意味着句柄没有被破坏。

Kids

```
Kids (integer, read-only)
```

包含当前数据库句柄的数目（驱动程序句柄）或当前语句句柄的数目（数据库句柄）。

LongReadLen

```
LongReadLen (unsigned integer, inherited)
```

控制长字段（BLOB和TEXT）的最大长度。它应该比最长的字段设置得稍微长一些。设置为0意味着长数据不能被自动获取（换句话说，当处理长字段时，fetch()将返回undef——不是字段的实际值）。把它设置得太大是浪费内存。

LongTruncOK

```
LongTruncOk (boolean, inherited)
```

假值（默认）表示试图获取比LongReadLen更长的长值，将导致装取过程失败。真值将返回一个被删减了的数值。

PrintError

```
PrintError (boolean, inherited)
```

当设置为真时（默认），方法中的错误将产生警告。当RaiseError设置为真时，通常设置此方法为假。

private_*

```
private_*
```

通过指定用private_开头的名字，把额外的信息作为私人属性存储在DBI句柄中。应该用private_your_module_name_description方式进行命名，而且只用一个属性。

由于Perl tie机制的工作方式，无法安全使用||=操作符直接初始化属性。

为了初始化，使用下面这样的两个步骤的方法：

```
my $descriptive_name = $dbh->{private_your_module_name_descriptive_name};
$descriptive_name ||= $dbh->{➥
 private_your_module_name_descriptive_name } = { ... };
```

不要使用下面的方法：

```
my $descriptive_name = $dbh->{➥
 private_your_module_name_descriptive_name } ||= { ... };
```

Profile

```
Profile (inherited)
```

允许方法调用时间统计信息。参看DBI::Profile文档以获取更多细节。

RaiseError

```
RaiseError (boolean, inherited)
```

默认值为假；当设置为真时，将导致引发异常的错误，而不是仅返回错误代码。通常，当RaiseError为真时，PrintError被设置为假。

ShowErrorStatement

```
ShowErrorStatement (boolean, inherited)
```

当为真时，从RaiseError和PrintError添加语句的文本到错误信息中。它应用于语句句柄错误和prepare()、do()及选择数据库句柄的方法。

Taint

```
Taint (boolean, inherited)
```

如果设为真，且Perl正运行在感染（taint）模式（-T）下，数据库中所有的数据都作为被感染处理，就像大多数DBI方法的变量。它默认值为假。在后期，更多的数据可能被当

做感染处理，因此，使用Taint要当心！

TraceLevel

```
TraceLevel (integer, inherited)
```

用于设置DBI跟踪级别，是除trace()方法以外的另一种选择。跟踪级别为0到9。0是禁止跟踪，1最适合于一般查看，2用得最普遍，其他方法则加入了更多的驱动程序和DBI细节。

Warn

```
Warn (boolean, inherited)
```

默认为真，启用警告。可以用$SIG{__WARN__}捕获警告信息。

数据库句柄属性

数据库句柄属性只适用于数据库句柄。

AutoCommit

```
AutoCommit (boolean)
```

如果设置为真，那么SQL语句被自动提交。如果为假，它们默认为部份事务，且需要被提交或重新处理。

Driver

```
Driver (handle)
```

包含上一级（父级）驱动程序的句柄。
例如：

```
$dbh->{Driver}->{Name}
```

Name

```
Name (string)
```

数据库名。

RowCacheSize

```
RowCacheSize (integer)
```

如果行的缓存没有被执行，应用程序的大小是本地行的缓存的大小或是undef。设置它为负数，指明被用于缓存的内存的大小。设置为0，缓存的大小自动决定。1是禁止缓存，更大一些的正数是缓存的大小，以行为单位。

Statement

```
Statement (string, read-only)
```

最近的SQL语句被传递给prepare()。

语句句柄的属性

这些属性应用于从预设的查询中已经被返回的语句句柄。它们大多数是只读的，并专用于语句句柄。

CursorName

```
CursorName (string, read-only)
```

与语句句柄相关的指针名，如果没有得到名字，则返回undef。

NAME

```
NAME (array-ref, read-only)
```

字段名数组的引用。名字可以是大写、小写或大小写混合。为了方便访问系统，可以用NAME_lc或NAME_uc代替。

例如，为了显示第二列，使用方法如下：

```
$sth = $dbh->prepare("select * from customer");
$sth->execute;
@row = $sth->fetchrow_array;
print "Column 2: $sth->{NAME}->[1]";
```

NAME_hash

```
NAME_hash (hash-ref, read-only)
```

把字段名信息作为混编引用返回。名字可以是大写、小写或大小写混合。混编键是字段名，数值是字段索引（对第一个字段，从0开始）。为了方便访问系统，可以用NAME_hash_lc或NAME_hash_uc代替。

例如：

```
$sth = $dbh->prepare("select first_name, surname from customer");
$sth->execute;
@row = $sth->fetchrow_array;
print "First name: $row[$sth->{NAME_hash}{first_name}]\n";
print "Surname: $row[$sth->{NAME_hash}{surname}]\n";
```

NAME_lc

```
NAME_lc (array-ref, read-only)
```

与NAME相同，但只返回小写的名字。

NAME_lc_hash

```
NAME_lc_hash (hash-ref, read-only)
```

与NAME_hash相同，但只返回小写的名字。

NAME_uc

```
NAME_uc (array-ref, read-only)
```

与NAME相同，但只返回小写的名字。

NAME_uc_hash

```
NAME_uc_hash (hash-ref, read-only)
```

与NAME_hash相同，但只返回大写的名字。

NULLABLE

```
NULLABLE (array-ref, read-only)
```

字段是否能够含有空值的数组引用。0表示否，1表示是，2表示不知道。
例如：

```
print "Field 1 can contain a NULL" if $sth->{NULLABLE}->[0];
```

NUM_OF_FIELDS

```
NUM_OF_FIELDS (integer, read-only)
```

预设语句将返回的字段个数。对于不返回字段的语句（INSERT、UPDATE等），值是0。

NUM_OF_PARAMS

```
NUM_OF_PARAMS (integer, read-only)
```

预设语句中占位符的数目。

ParamValues

```
ParamValues (hash ref, read-only)
```

混编引用，包含捆绑在占位符上的值。如没有的话，为undef。

PRECISION

```
PRECISION (array-ref, read-only)
```

对每个字段，是其整数数组引用，指的是字段最大长度（非数值字段）或有效数字的最大数目（不显示大小——它把正负号、小数点，E字符等排除在外）。

RowsInCache

```
RowsInCache (integer, read-only)
```

缓存中未获取的行的数目，如果驱动程序不支持本地行的缓存，值为undef。

SCALE

```
SCALE (array-ref, read-only)
```

返回对每列的整数值的数组引用，NULL（undef）值说明列的大小不能应用。

Statement

```
Statement (string, read-only)
```

传递给prepare()方法的最后一个SQL查询。

TYPE

```
TYPE (array-ref, read-only)
```

每个字段的整数值（代表数据类型）的数组引用。

动态属性

这是一些使用时间短的属性，只在被设置后短时间有效。它们用于刚被返回的句柄。

err

```
$DBI::err
```

与$handle->err相同。

errstr

```
$DBI::errstr
```

与$handle->errstr相同。

lasth

```
$DBI::lasth
```

返回与最近的DBI方法调用一同使用的DBI句柄，如果方法调用的是DESTROY，则返回句柄的父一级（如果存在的话）。

rows

```
$DBI::rows
```

与$handle->rows相同。

state

```
$DBI::state
```

与$handle->state相同。

Perl DBI的例子

清单D.1连接到服务器，用占位符准备查询语句，把一些数值捆绑到占位符，然后循环执行每行和每个查询，以显示结果。

程序清单D.1 example.pl

```
#!/usr/bin/perl -w

use strict;  # you don't have to use strict, but you should!
use DBI;     # the main module

# set variables with the connection details
my $hostname = 'localhost';
my $database = 'firstdb ;
my $username = 'guru2b';
my $password = 'g00r002b';

#Connect to the database
my $dbh = DBI->connect("dbi:mysql:$database:$hostname", $username, $password);

# Define and prepare the query, with the ? specifying a bind variable.
my $sql = q{SELECT first_name,surname FROM customer WHERE id=?};
my $sth = $dbh->prepare($sql);

# Create an array of id's to use to replace the placeholder
my @ids = (1,4,5,6);

# Loop through the array and execute the query
for(@ids) {
  $sth->bind_param(1, $_, SQL_INTEGER);
  $sth->execute();

  my( $first_name, $surname);
  $sth->bind_columns(undef, \$first_name, \$surname);

  # Loop through the rows returned and display the results
  while( $sth->fetch()) {
    print "$first_name $surname\n";
  }
}
$sth->finish();

$dbh->disconnect;
```

附录E Python数据库API

Python为其数据库连通性使用与数据库无关的DB-API。特别是对于MySQL，它使用MySQLdb模块。最终的API版本是Python Database API规范2.0。它仍然是相当新的API，其他API中的许多可用特性还没有实现，然而其简单性使得它易学易用。

在http://sourceforge.net/projects/mysql-python/上可以找到MySQLdb和整个安装指南。

属性

对于整个模块，属性可能都是可用的，或者它们可能针对一个指针。本节将根据属性是如何可用的讲述这些可用的属性。

模块属性

这些属性对于整个模块都可用。

apilevel

字符串常数，包含被支持的DB-API的版本（举例来说，如果在使用版本2.0，那么值为2.0）。

conv

将MySQL类型映射到Python对象。默认值为MySQLdb.converters.conversions。

paramstyle

字符串常数，包含接口使用的参数标记（占位符）格式化的的类型。它可能是format，例如：

```
...WHERE fieldname=%s
```

它也可能是pyformat，例如：

```
...WHERE fieldname=%(name)s
```

threadsafety

整数常数，包含线程安全性的级别。它可能是0（非线程共享）、1（线程只能够共享模块）、2（线程可以共享模块和连结）或者3（线程可以共享模块，连接和指针）。默认值是1。

指针属性

这些属性针对指针对象，来自cursor()程序。

arraysize

指定由fetchmany()程序返回的行的数目，影响程序fetchall()的性能。默认值为1，或每次一行。

description

此属性只读，并且描述结果集中列的序列，每个序列有7个子项。它们是name、type_code、display_size、internal_size、precision、scale和null_ok。

name和type_code子项是强制的，并且余下的子项在值无意义时，它们会被设成None。

如果查询没有返回任何行或还没有被调用，那么它被设成None。

rowcount

此属性只读，它指出上次查询影响到的或返回的行的数目。如果行的数目未知，或查询还没有被调用，那么返回－1。

程序

程序对于整个模块、连接或指针都是可用的。本节将依据模块、连接或指针程序讲述各种不同的程序。

模块程序

模块程序对于整个模块都是可用的。最重要的是connect()程序：

```
dbh = MySQLdb.connect(parameters)
```

利用指定的用户名和密码连接指定的主机名和数据库，返回一个连接对象（或数据库句柄）。这些参数包括：

host 默认值为本地主机（localhost）。

user 默认值为当前用户。

passwd 没有默认值（空密码）。

db 没有默认值。

conv 将文字映射到Python类型的字典。默认为MySQLdb.converters.conversions。

cursorclass cursor()使用的种类，默认值为MySQLdb.cursors.Cursor。

compress 启动协议压缩。

named_pipe 在Windows中，与一个命名管道相连接。

init_command 一旦连接建立，就为数据库服务器指定一条语句来运行。

read_default_file 使用的MySQL配置文件。

read_default_group 读取的默认组。

unix_socket 在UNIX中，连接使用指定的套接字。它默认使用TCP。

port 默认是3306。

例如：

```
dbh = MySQLdb.connect(user='guru2b', passwd='g00r002b',➥
 host='test.host.co.za', db='firstdb')
```

连接方法

这些方法用于一个连接对象，这些对象由MySQLdb.connect()方法返回。操作者差不多会始终使用cursor()和close()程序。commit()和rollback()程序只用于事务。

begin

```
dbh.begin()
```

开始一个事务，如果AUTOCOMMIT已经开启则关闭它，直到事务以调用commit()或rollback()结束。

close

```
dbh.close()
```

关闭连接并释放相关的资源。

commit

```
dbh.commit()
```

提交所有开放的事务。

cursor

```
dbh.cursor([cursorClass])
```

返回一个新的指针对象（它提供访问和操作数据的方法）。如果操作者愿意（默认情况下，它是在连接中指定的是cursorclass，其默认值为Cursor类别），那么操作者可以指定一个不同的类别。

rollback

```
dbh.rollback()
```

回退所有的开放式事务。关闭未明确调用此方法的连接将对所有开放式事务隐含调用rollback。

指针方法

这些方法用于访问和操作数据。它们对指针对象起作用，这些指针对象来自于cursor()方法。

close

```
cursor.close()
```

立即释放与指针相关的资源。

execute

```
cursor.execute(query[,parameters])
```

准备并执行数据库查询。此方法还准许操作者通过指定不同的参数使用占位符来优化重复的相同类型的查询。占位符常常用问号（?）标注，但是MySQLdb当前不支持它。由于MySQLdb将所有的值都看做是字符串，无论其字段类型实际是什么，因此操作者需要使用%s来指定一个占位符（如果paramstyle属性设置成format）。

例如：

```
cursor.execute('INSERT INTO customer(first_name,surname) VALUES➥
 (%s, %s)',('Mike', 'Harksen'))
```

如果操作者设置了MySQLdb.paramstyle = 'pyformat'，那么还可以使用Python映射作为第二个自变量。

操作者可以使用tuples列表作为第二个参数，但并不推荐这样用。可以用executemany来代替。

executemany

```
cursor.executemany(query,seq_of_parameters)
```

准备数据库查询并运行多个带有占位符的请求，对相似查询的重复进行优化。

例如：

```
cursor.executemany('INSERT INTO customer(first_name,surname) VALUES➥
 (%s, %s)',(('Mike', 'Harksen'),( 'Mndeni', 'Vidal'),( 'John', 'Vilakazi')))
```

如果操作者设置了MySQLdb.paramstyle = 'pyformat'，那么还可以使用Python映射作为第二个自变量。

fetchall

```
cursor.fetchall()
```

取出查询结果中的所有行（从当前行指针中），以序列的顺序（tuples列表）返回信息。这个指针的arraysize属性可以影响程序的性能。

如果出现错误，它将出现异常。

例如：

```
cursor.execute("SELECT first_name, surname FROM customer")
for row in cursor.fetchall():
    print "Firstname: ", row[0]
    print "Surname: ", row[1]
```

fetchmany

```
cursor.fetchmany([size=cursor.arraysize]);
```

从查询结果中取出许多行，以序列的形式返回结果（tuples列表）。操作者利用可选的大小参数指定行的数量，或者如果没有指定空间参数，那么将使用指针的arraysize。此方法将不会返回比可用行更多的行。

由于性能的原因，或者为了在两个fetchmany调用之间保持相同的大小参数，最好使用arraysize属性。

如果出现错误，那么它将出现异常。

fetchone

```
cursor.fetchone()
```

从查询结果集中返回下一行。

如果出现错误，那么它将出现异常。

例如：

```
cursor.execute("SELECT first_name, surname FROM customer")
row = cursor.fetchone():
    print "Firstname: ", row[0]
    print "Surname: ", row[1]
```

insert_id

```
cursor.insert_id()
```

这是一个非DB API（non-DB-API）标准方法，它返回前面AUTO_INCREMENT字段的值。

nextset, setinputsizes, and setouputsizes

```
cursor.nextset()
```

这些标准方法当前不用于MySQL。

Python的一个小范例

程序清单E.1向你显示了连接、运行查询和结果处理的基本内容。

程序清单E.1 example.py

```
#!/usr/bin/env python

import MySQLdb
dbh = None
try:
    dbh = MySQLdb.connect(user='guru2b', passwd='g00r002b',➥
      host='test.host.co.za', db='firstdb')
```

```
except:
    print "Could not connect to MySQL server."
    exit(0)

try:
    cursor = dbh.cursor()
    cursor.execute("UPDATE customer SET surname='Arendse' WHERE➥
      surname='Burger'")
    print "Rows updated: ", cursor.rowcount
    cursor.close()
except:
    print "Could not update the table."

try:
    cursor = dbh.cursor()
    cursor.execute("SELECT first_name,surname FROM customer")
    for row in cursor.fetchall():
        print "Name: ", row[0], row[1]
    dbh.close()
 except:
    print "Failed to perform query"
    dbh.close()
```

附录F Java API

Java使用先进的JDBC API进行数据库的访问。有两个主要的MySQL驱动程序："官方"的MySQL连接器/J（即MM.MySQL），它可以从MySQL节点上下载。另一个是驱动程序Caucho MySQL JDBC。

基本的过程是例示一个连接对象、一个语句对象，然后得出结果集（result set）对象。

此附录将简要讲述Java中用于数据库的主要方法，并为插入和选择数据提供一个简单的范例。

通用程序

本节将讲述各类用于连接或访问文件中的配置数据的各种方法。

getBundle

```
bundle.getBundle(filename)
```

从称为Config.properties的参数文件中加载数据。虽然不是专为JDBC使用，但是在向配置文件中存储连接数据时，需要使用它。

例如，Config.properties包含下列内容：

```
Driver = com.mysql.jdbc.Driver
Conn = jdbc:mysql://test.host.com/firstdb?user=guru2b&password=g00r002b
```

主程序随后包含下列内容：

```
ResourceBundle rb = ResourceBundle.getBundle("Conn");
String conn = rb.getString("Conn");
...
Class.forName(rb.getString("Driver"));
Connection = DriverManager.getConnection(conn);
```

getConnection

```
DriverManager.getConnection(connection_details)
```

利用指定的细节请求一个连接，返回一个连接对象。根据操作者使用的驱动程序，指定连接的细节信息的方式将略有不同。对于Caucho驱动程序，格式如下：

```
jdbc:mysql-caucho://host_name[:port]/database_name
```

例如：

```
Connection connection = DriverManager.getConnection("jdbc:mysql-caucho://➥
  test.host.co.za/firstdb", "guru2b", "g00r002b");
```

Connector/J驱动程序使用略微不同的格式，如下所示：

```
jdbc:mysql://[host_name][:port]/database_name[?property1=value1][&property=value2]
```

虽然操作者通常只会使用密码和用户名，但是这些参数可以是表F.1中列出的任何参数。

表F.1 连接参数

名字	描述
autoReconnect	在连接关闭后，自动再次尝试进行连接。默认值为false
characterEncoding	当字符集是Unicode时，指定要使用的Unicode编码
initialTimeout	两次重新连接尝试之间的时间。默认值为2
maxReconnects	尝试重新连接的最大数目。默认值为3
maxRows	查询能够返回的最大行数目，0表示所有的行
password	用户的密码（无默认值）
useUnicode	指定Unicode为连接使用的字符集。默认值为false
user	连接使用的用户（无默认值）

例如：

```
DriverManager.getConnection("jdbc:mysql://➥
  test.host.co.za/firstdb?user=guru2b&password=g00r002b");
```

getString

```
bundle.getString(string)
```

参见getBundle及其范例，它可以从配置文件中读取数据。

连接方法

连接方法需要一个有效的连接，这个连接可以从getConnection()程序中获得。

clearWarnings

```
connection.clearWarnings()
```

清除连接的所有告警信息，返回空白。

close

```
connection.close()
```

关闭数据库连接，并且释放所有连接资源，返回空白。

commit

```
connection.commit()
```

提交开放式事务。

createStatement

```
connection.createStatement([int resultSetType, int resultSetConcurrency])
```

返回一个Statement对象，此对象是将查询传递到数据库的一种算法，并且通过它的连接对象接收返回结果。利用可选的自变量，生成的ResultSet对象将具有指定的类型。

getAutoCommit

```
connection.getAutoCommit()
```

如果对连接设置了AutoCommit模式，则返回true，否则false。

getMetaData

```
connection.getMetaData()
```

返回数据库元数据对象，它包含了有关已经建立的连接的数据库的元数据。

getTransactionIsolation

```
connection.getTransactionIsolation()
```

返回一个整数，它包含连接的事务隔离级别。事务隔离级别可以是下列级别中的一种：TRANSACTION_READ_UNCOMMITTED、TRANSACTION_READ_COMMITTED、TRANSACTION_REPEATABLE_READ、TRANSACTION_SERIALIZABLE或TRANSACTION_NONE。

getTypeMap

```
connection.getTypeMap
```

返回与连接关联的一个Map对象。

isClosed

```
connection.isClosed()
```

如果连接已经关闭，则返回true。如果连接仍然处于开启的状态，则返回false。

isReadOnly

```
connection.isReadOnly()
```

如果连接式只读的，则返回true。否则返回false。

nativeSQL

```
connection.nativeSQL(String sql)
```

利用提供的转换到系统所带SQL的字符串，返回一个字符串。

prepareStatement

```
connection.prepareStatement(String sql)
```

准备发送到数据库的语句，这意味着你可以使用占位符（或参数）。使用setInt()和setString()程序来设置参数值。

rollback

```
connection.rollback()
```

撤销当前事务的所有修改。

setAutoCommit

```
connection.setAutoCommit(boolean mode)
```

设置连接的AutoCommit模式（如果设置则为true，如果没有设置，则为false）。

setReadOnly

```
connection.setReadOnly(boolean mode)
```

传递此方法将设置连接为只读模式。

setTransactionIsolation

```
connection.setTransactionIsolation(int level)
```

等级可以是下面的一种：connection.TRANSACTION_READ_UNCOMMITTED、connection.TRANSACTION_READ_COMMITTED、connection.TRANSACTION_REPEATABLE_READ或connection.TRANSACTION_SERIALIZABLE。

setTypeMap

```
connection.setTypeMap(Map map)
```

设置连接的Map对象类型。

Statement和Prepared Statement方法

这些方法必须通过一个有效的Statement或PreparedStatement对象调用。

大多数这些方法适用于语句和预备语句。那些具有preparedstatement的方法作为一个对象只能与预备语句一起调用，而具有statement的程序作为一个对象可以被随便一个调用。

addBatch

```
statement.addBatch(String sql)
preparedstatement.addBatch()
```

向当前语句的列表中加入SQL语句，它随后可以利用executeBatch()方法执行。

clearBatch

```
statement.clearBatch()
```

在由方法addBatch()添加的批处理（batch）文件中，清除语句的列表。

clearWarnings

```
statement.clearWarnings()
```

清除所有与语句相关的告警信息。

close

```
statement.close()
```

释放所有与语句相关的资源。

execute

```
statement.execute(String sql [,int autoGeneratedKeys | int[] columnIndexes➥
  | String[] columnNames])
preparestatement.execute()
```

执行一条SQL语句。如果查询得出一组结果，则此语句返回true（如同一条SELECT语句），如果没有得出任何结果，则返回false（如同一条INSERT或UPDATE语句）。这些参数说明自动生成的键应该可用于检索——所有这些参数，或所有整数参数部分，或字符串序列。

executeBatch

```
statement.executeBatch()
```

执行批处理文件（由addBatch添加的）中的所有语句，返回一个更新计数的整数序列。如果有语句执行错误，那么此语句返回false。

executeQuery

```
statement.executeQuery(String sql)
preparedstatement.executeQuery()
```

执行查询，返回数据（如SELECT或SHOW）并返回单一一组结果。

executeUpdate

```
statement.executeUpdate(String sql)
preparedstatement.executeUpdate()
```

执行查询来修改数据（如UPDATE，INSERT或ALTER）并返回被修改的行的数目。

getConnection

```
statement.getConnection()
```

返回生成这条语句的连接对象。

getFetchSize

```
statement.getFetchSize()
```

返回一个整数值，作为这条语句的ResultSet对象的默认大小。

getMaxFieldSize

```
statement.getMaxFieldSize()
```

返回一个整数值，它是这条语句的ResultSet对象的字符和二进制列值的最大字节数。

getMaxRows

```
statement.getMaxRows()
```

返回一个整数值，它是语句的ResultSet对象尽可能包含的最大行数目。

getMoreResults

```
statement.getMoreResults([int current])
```

转到语句的下一个结果，如果有另一个有效的ResultSet，则返回true；如果没有，则返回false。如果没有参数，所有当前的ResultSet对象都将关闭，否则它们会根据current值（它可以是CLOSE_CURRENT_RESULT，KEEP_CURRENT_RESULT或CLOSE_ALL_RESULTS）进行处理。

getQueryTimeout

```
statement.getQueryTimeout()
```

返回驱动程序在等待查询执行时的最长时间。

getResultSet

```
statement.getResultSet()
```

返回当前语句的一个结果集。

getResultSetType

```
statement.getResultSetType()
```

返回当前语句ResultSet对象的类型。

getUpdateCount

```
statement.getUpdateCount()
```

返回更新计数的当前结果。如果结果是ResultSet对象或没有内容，则返回－1。

setXXX

```
preparedstatement.setXXX(int parameter, xxx value)
```

在前面的准备语句中设置参数，参数从1开始。参数值符合相应的类型（见表F.2）。

表F.2 SQL类型和等价的设置方法

SQL类型	JAVA方法
BIGINT	setLong()
BINARY	setBytes()
BIT	setBoolean()
BLOB	SetBlob()
CHAR	setString()
DATE	setDate()
DECIMAL	setBigDecimal()
DOUBLE	setDouble()
FLOAT	setDouble()
INTEGER	setInt()
LONGVARBINARY	setBytes()
LONGVARCHAR	setString()
NUMERIC	setBigDecimal()
OTHER	setObject()
REAL	setFloat()
SMALLINT	setShort()
TIME	setTime()
TIMESTAMP	setTimestamp()
TINYINT	setByte()
VARBINARY	setBytes()
VARCHAR	setString()

例如：

```
preparedstatement = connection.prepareStatement("UPDATE customer➥
  SET surname = ? WHERE id=?");
preparedstatement.setString(1,"Burger");
preparedstatement.setInt(2,9);
preparedstatement.executeUpdate();
```

setCursorName

```
statement.setCursorName(String cursorname)
```

设置SQL指针名，由稍后的execute()方法使用。

setEscapeProcessing

```
statement.setEscapeProcessing(boolean mode)
```

设置退出（escape）处理进程（如果模式值为true），或者关闭处理进程（如果模式值为false）。默认值为true。退出处理进程对于PreparedStatement对象没有影响。

setFetchSize

```
statement.setFetchSize(int size)
```

当语句需要更多的行时，指示驱动程序应该从数据库中返回多少行。

setMaxFieldSize

```
statement.setMaxFieldSize(int limit)
```

设置一个二进制或字符ResultSet列的最大字节数。

setMaxRows

```
statement.setMaxRows(int limit)
```

设置Result对象可以包含的最大行的数目。

setQueryTimeout

```
statement.setQueryTimeout(int seconds)
```

设置驱动程序等待查询执行时的最长时间，并且以秒为单位。

ResultSet Methods

这些程序需要一个有效的来自getResultSet()方法的ResultSet对象。

absolute

```
resultset.absolute(int row)
```

移动指针到结果集（result set）中的指定行（行从1开始）。操作者可以使用一个负数从结果集的末尾移动到某一行。如果指针在这一行上，则返回true，否则返回false。

afterLast

```
resultset.afterlast()
```

将指针移动到结果集的末尾。

beforeFirst

```
resultset.beforeFirst()
```

将指针移动到结果集的开始。

cancelRowUpdates

```
resultset.cancelRowUpdates()
```

取消对结果集中当前行的修改。

close

```
resultset.close()
```

关闭结果集，并释放相关的资源。

deleteRow

```
resultset.deleteRow()
```

从数据库（和结果集）中删除当前的ResultSet行。

findColumn

```
resultset.findColumn(String field_name)
```

在结果集中映射字段名（fieldname）到列，并且返回整数形式的列索引。

first

```
resultset.first()
```

移动指针到结果集中的第一行。如果首行正确，则返回true，否则返回false。

getXXX

```
resultset.getXXX(String fieldname | int fieldindex)
```

得出指定类型的某个字段的内容。操作者可以根据字段的名字或位置进行区分。表F.3给出了SQL类型和等价的Java方法。

表F.3 SQL类型和等价的Get方法

SQL类型	JAVA方法
BIGINT	getLong()
BINARY	getBytes()
BIT	getBoolean()
BLOB	getBlob()
CHAR	getString()

（续表）

SQL TYPE	JAVA METHOD
DATE	getDate()
DECIMAL	getBigDecimal()
DOUBLE	getDouble()
FLOAT	getDouble()
INTEGER	getInt()
LONGVARBINARY	getBytes()
LONGVARCHAR	getString()
NUMERIC	getBigDecimal()
OTHER	getObject()
REAL	getFloat()
SMALLINT	getShort()
TIME	getTime()
TIMESTAMP	getTimestamp()
TINYINT	getByte()
VARBINARY	getBytes()
VARCHAR	getString()

getCursorName

```
resultset.getCursorName()
```

得出结果集使用的指针的名字。

getFetchSize

```
resultset.getFetchSize()
```

返回结果集对象的存取空间大小。

getMetaData

```
resultset.getMetaData()
```

返回ResultSetMetaData对象，并带有结果集中列的号码、类型和属性。

getRow

```
resultset.getRow()
```

返回一个整数，并包含当前的行号。

getStatement

```
resultset.getStatement()
```

返回生成结果集的语句对象。

getType

```
resultset.getType()
```

返回结果集对象的类型。

getWarnings

```
resultset.getWarnings()
```

返回由结果集中ResultSet方法调用而触发的第一个Warning。

insertRow

```
resultset.insertRow()
```

向数据库（和结果集中）中插入被插入行的内容。

isAfterLast

```
resultet.isAfterLast()
```

如果指针位于结果集的最后一行之后，则返回true，否则返回false。

isBeforeFirst

```
resultset.isBeforeFirst()
```

如果指针位于结果集的第一行之前，则返回true，否则返回false。

isFirst

```
resultset.isFirst()
```

如果指针位于结果集中的第一行，则返回true，否则返回false。

isLast

```
resultset.isLast()
```

如果指针位于结果集的最后一行，则返回true，否则返回false。

last

```
resultset.last()
```

移动指针到结果集的最后一行。如果最后一行有效，则返回true，否则返回false。

moveToCurrentRow

```
resultset.moveToCurrentRow()
```

移动指针到记录的指针位置，通常是当前的行。如果指针不在插入的行上，则没有影响。参见moveToInsertRow()方法。

moveToInsertRow

```
resultset.moveToInsertRow()
```

移动指针到插入行（新行可以利用更新方法被放置的缓冲区）。它记录了当前指针的位置，利用moveToCurrentRow()方法可以将指针返回到这里。

next

```
resultset.next()
```

移动指针到结果集的下一行，如果正确则返回true，否则返回false（已经到达结尾处时）。

例如：

```
connection = DriverManager.getConnection(url, "guru2b", "g00r002b");
statement = connection.createStatement();
resultset = statement.executeQuery("SELECT first_name,surname FROM customer");
while(resultset.next()) {
  String first_name = resultset.getString("first_name");
  String surname = resultset.getString("surname");
  System.out.print("Name: " + first_name + " " + surname);
}
```

previous

```
resultset.previous()
```

移动指针到结果集中的前一行，如果正确则返回true，否则返回false（已经到达第一行）。

例如：

```
while(resultset.previous()) {
   ...
}
```

refreshRow

```
resultset.refreshRow()
```

利用数据库中最新的值对结果集中的行进行刷新。

relative

```
resultset.relative(int rows)
```

根据所在的rows数，将指针向前移动（如果rows是正数）或向后移动（如果rows是负数）。

rowDeleted

```
resultset.rowDeleted()
```

如果结果集中的某一行被查出已被删除，则返回true，否则返回false。

rowInserted

```
resultset.rowInserted()
```

如果结果集中检测到插入了某一行，则返回true，否则返回false。

rowUpdated

```
resultset.rowUpdated()
```

如果结果集中检测到修改了某一行，那么返回true，否则返回false。

setFetchSize

```
resultset.setFetchSize(int rows)
```

当结果集中需要更多的行时，指示驱动程序应该有多少行从数据库中返回。

updateXXX

利用指定类型值更新字段。表F.4显示了SQL类型和等价的Java方法。

表F.4 SQL类型和等价的Update方法

SQL类型	JAVA方法
BIGINT	updateLong()
BINARY	updateBytes()
BIT	updateBoolean()
BLOB	updateBlob()
CHAR	updateString()
DATE	updateDate()
DECIMAL	updateBigDecimal()
DOUBLE	updateDouble()
FLOAT	updateDouble()
INTEGER	updateInt()
LONGVARBINARY	updateBytes()
LONGVARCHAR	updateString()
NUMERIC	updateBigDecimal()
NULL	updateNull()
OTHER	updateObject()
REAL	updateFloat()
SMALLINT	updateShort()
TIME	updateTime()

（续表）

SQL类型	JAVA方法
TIMESTAMP	updateTimestamp()
TINYINT	updateByte()
VARBINARY	updateBytes()
VARCHAR	updateString()

updateRow

```
resultset.updateRow()
```

根据结果集中当前行的内容对数据库进行更新。

wasNull

```
resultSet.wasNull()
```

如果上一个字段是SQL NULL，则返回true，否则返回false。

ResultSetMetaData方法

这些方法需要来自getMetaData()方法的有效的ResultSetMetaData对象。它们从结果中获得信息。从第一列开始计数并进行偏移量计算。

getColumnCount

```
resultsetmetadata.getColumnCount()
```

返回结果集中列的数量。

getColumnDisplaySize

```
resultsetmetadata.getColumnDisplaySize(int column)
```

返回指定列的最大字符宽度。

getColumnName

```
resultsetmetadata.getColumnName(int column)
```

返回指定列的字段名。

getColumnType

```
resultsetmetadata.getColumnType(int column)
```

返回指定列的SQL类型。

getColumnTypeName

```
resultsetmetadata.getColumnTypeName(int column)
```

返回指定列的数据库具体类型名。

getPrecision

```
resultsetmetadata.getPrecision(int column)
```

返回指定列中的小数位数。

getScale

```
resultsetmetadata.getScale(int column)
```

返回指定列中小数点后的位数。

getTableName

```
resultsetmetadata.getTableName(int column)
```

返回拥有指定列的表的名字。

isAutoIncrement

```
resultsetmetadata.isAutoIncrement(int column)
```

如果指定列是一个自动增量的字段，则返回true，否则返回false。

isCaseSensitive

```
resultsetmetadata.isCaseSensitive(int column)
```

如果指定列区分大小写，则返回true，否则返回false。

isDefinitelyWritable

```
resultsetmetadata.isDefinitelyWritable(int column)
```

如果在指定列上的写操作成功执行了，则返回true，否则返回false。

isNullable

```
resultsetmetadata.isNullable(int column)
```

返回指定列的可空状态，这个列可以是columnNoNulls、columnNullable或columnNullableUnknown。

isReadOnly

```
resultsetmetadata.isReadOnly(int column)
```

如果指定列是只读的，则返回true，否则返回false。

isSearchable

```
resultsetmetadata.isSearchable(int column)
```

如果指定列可以用在WHERE子句中，则返回true，否则返回false。

isSigned

```
resultsetmetadata.isSigned(int column)
```

如果指定的列是一个具有正负值区别的列，则返回true，否则返回false。

isWritable

```
resultsetmetadata.isWritable(int column)
```

如果指定的列可以进行写操作，则返回true，否则为false。

SQLException方法

在SQLException对象已经建立起来时，操作者可以使用这些方法。

getErrorCode

```
sqlexception.getErrorCode()
```

返回来自供应商的错误代码。

getMessage

```
sqlexception.getMessage()
```

继承自Throwable，此方法为Throwable返回消息字符串。

getNextException

```
sqlexception.getNextException()
```

返回下一个SQLException，如果没有，则为空（null）。

getSQLState

```
sqlexception.getSQLState()
```

返回SQLState标识符。

printStackTrace

```
sqlexception.printStackTrace(PrintStream s)
```

继承自Throwable的分类，此方法将堆栈记录输出到标准的错误流中。

setNextException

```
setNextException(sqlexception e)
```

添加一个SQLException。

Warning方法

当SQLWarning对象已经建立起来时，操作者使用这些方法。

getNextWarning

```
sqlwarning.getNextWarning()
```

返回下一个SQLWarning，如果没有则为空（null）。

setNextWarning

```
sqlwarning.setNextWarning(SQLWarning w)
```

添加一个SQLWarning。

一个简单的Java例子

程序清单F.1在命令行中使用了三个参数，向customer表中插入它们。按照下面的形式运行命令：

```
% /usr/java/j2sdk1.4.1/bin/java InsertSelect 10 Leon Wyk
```

程序清单F.1 InsertSelect.java

```
import java.sql.*;
import java.util.*;

// The insertRecord class does the bare minimum for inserting a record
// and also performs a skeleton level of exception handling
public class InsertRecord {

  public static void main(String argv[]) {
    Connection dbh = null;
    ResultSet resultset;

    try {
      Class.forName("com.mysql.jdbc.Driver");
    }
    catch (Exception e) {
      System.err.println("Unable to load driver.");
      e.printStackTrace();
    }
```

```
    try {
      Statement sth,sth2;
      // connect to the database using the Connector/J driver
      dbh = DriverManager.getConnection("jdbc:mysql://➥
        localhost/firstdb?user=mysql");

      // instantiate the statement object, and run the query (an update query)
      // the three argv[] fields come from the command line
      sth = dbh.createStatement();
      sth.executeUpdate("INSERT INTO customer(id,first_name,surname)➥
        VALUES(" + argv[0] + ", '" + argv[1] + "', '"+argv[2]+ "')");
      sth.close();

      // instantiate the statement object, and run the SELECT query
      sth2 = dbh.createStatement();
      resultset = sth2.executeQuery("SELECT first_name,surname FROM customer");

      // loop through the result set, displaying the results
      while(resultset.next()) {
        String first_name = resultset.getString("first_name");
        String surname = resultset.getString("surname");
        System.out.println("Name: " + first_name + " " + surname);
      }
      sth2.close();
    }
    catch( SQLException e ) {
      e.printStackTrace();
    }
    finally {
      if(dbh != null) {
        try {
          dbh.close();
        }
      catch(Exception e) {}
      }
    }
  }
}
```

附录G C API

MySQL即带有C API，并且被包含在mysqlclient库中。MySQL的许多客户程序是用C语言写的，大多数来自其他语言的API使用C API（例如，查看一下C语言和PHP函数之间的相似之处）。还可以使用针对C++的C API进行程序开发，对于面向对象的方法，可以使用MySQL++，从MySQL的网站www.mysql.com上能够获得这些开发语言。

C API数据类型

在与数据库服务器进行交互的时候，C语言的数据类型用来描述将要处理的数据。这些数据类型都不相同，例如，一些结构复杂，另一些则是简单的布尔类型。你需要逐渐熟悉这些类型从而掌握C API，了解什么函数返回这些数据类型或把它们作为参数使用。

my_ulonglong

范围从－1到1.84e19的数值类型，用来从函数（例如，mysql_affected_rows()、mysql_num_rows()和mysql_insert_id()）返回值。

MYSQL

数据库句柄（数据库服务器的连接）。用函数mysql_init()初始化此变量，并被大多数API函数所使用。

MYSQL_FIELD

从mysql_fetch_field()函数返回的字段数据，包括字段名、类型和大小。实际的字段值被存储在MYSQL_ROW结构中。可以在结构中找到下面这些元素：

char * name 作为无效终止字符串的字段名。

char * table 包含字段或空字符串的表，如果没有字符串（如计算字段）。

char * def 作为无效终止字符串的字段的默认值。这个类型只有函数mysql_list_fields()被用来返回MYSQL_RES时才有效。

enum enum_field_types type 字段类型。值与MySQL字段类型相对应，如表G.1所示。

表G.1 MYSQL字段类型

值	MYSQL类型
FIELD_TYPE_TINY	TINYINT
FIELD_TYPE_SHORT	SMALLINT
FIELD_TYPE_LONG	INTEGER

（续表）

值	MYSQL类型
FIELD_TYPE_INT24	MEDIUMINT
FIELD_TYPE_LONGLONG	BIGINT
FIELD_TYPE_DECIMAL	DECIMAL or NUMERIC
FIELD_TYPE_FLOAT	FLOAT
FIELD_TYPE_DOUBLE	DOUBLE or REAL
FIELD_TYPE_TIMESTAMP	TIMESTAMP
FIELD_TYPE_DATE	DATE
FIELD_TYPE_TIME	TIME
FIELD_TYPE_DATETIME	DATETIME
FIELD_TYPE_YEAR	YEAR
FIELD_TYPE_STRING	CHAR or VARCHAR
FIELD_TYPE_BLOB	BLOB or TEXT
FIELD_TYPE_SET	SET
FIELD_TYPE_ENUM	ENUM
FIELD_TYPE_NULL	NULL-type
FIELD_TYPE_CHAR	不推荐，应用FIELD_TYPE_TINY替代

unsigned int length 字段的最大宽度，由表的定义来决定。

unsigned int max_length 字段在结果集中最大的宽度。通常这个类型会与length发生混淆，但max_length的值小一些。如果函数mysql_use_result()返回MYSQL_RES，则类型值中包含0。

unsigned int flags 表G.2中的任何一个标记都可以被设置。它们提供有关字段的额外信息。

表G.2 标记

标记	说明
NOT_NULL_FLAG	定义为NOT NULL型
PRI_KEY_FLAG	主键的一部分
UNIQUE_KEY_FLAG	惟一键的一部分
MULTIPLE_KEY_FLAG	非惟一键的一部分
UNSIGNED_FLAG	定义为UNSIGNED型
ZEROFILL_FLAG	定义为ZEROFILL型
BINARY_FLAG	定义为BINARY型
AUTO_INCREMENT_FLAG	定义为AUTO_INCREMENT型字段
ENUM_FLAG	定义为ENUM（不推荐，应使用FIELD_TYPE_ENUM）
SET_FLAG	定义为SET（不推荐，应使用FIELD_TYPE_SET）
BLOB_FLAG	定义为BLOB或TEXT（不推荐，应使用FIELD_TYPE_BLOB）
TIMESTAMP_FLAG	定义为TIMESTAMP（不推荐，应使用FIELD_TYPE_TIMESTAMP）

下面是使用其中一个选项的例子：

```
if (field->flags & MULTIPLE_KEY_FLAG) {
 /* do something with the fact that the field is a multiple key */
}
```

表G.3列出的宏指令可以使对一些标记值的测试更容易。

表G.3 宏指令

MACRO	DESCRIPTION
IS_NOT_NULL(flags)	如果字段被定义为NOT NULL，则值为真
IS_PRI_KEY(flags)	如果字段是主键或主键的一部分，则值为真
IS_BLOB(flags)	如果字段是BLOB或TEXT，则值为真（不推荐，应使用FIELD_TYPE_BLOB代替）
IS_NUM(flags)	如果字段是数值，则值为真

unsigned int decimals 由一个数字使用的小数位的位数。

MYSQL_FIELD_OFFSET

表示一个字段列表中字段指针的位置。第一个字段的位置从0开始。由函数mysql_field_seek()使用。

MYSQL_RES

包含返回数据的查询语句（例如，SELECT、DESCRIBE、SHOW或EXPLAIN）结果的结构。

MYSQL_ROW

从函数mysql_fetch_row()获得的一行数据。所有的数据表被表示为一个字符串数组，如果每一个数据是二进值的，则字符串中可能会包含无效字节。

C API函数

C API函数用来打开和关闭服务器连接，执行查询操作，分析查询结果，调试和执行管理任务。你需要非常熟悉这些函数，并且知道它们是如何与C API数据类型交互的，从而掌握C API。

mysql_affected_rows

```
my_ulonglong mysql_affected_rows(MYSQL *mysql)
```

返回最后一个查询语句所影响的行数（例如，使用DELETE语句删除的行数，或从SELECT语句返回的行数，在这种情况下，此函数与mysql_num_rows函数相同）。出现错误时，函数返回－1。

使用UPDATE语句，即使符合更新条件但没有执行，也不计算受影响行的个数，除非在与mysql_real_connect()连接的时候，设置了CLIENT_FOUND_ROWS标记。

例如：

```
/* Update the customer table, and return the number of records affected */
mysql_query(&mysql, "UPDATE customer SET first_name='Jackie' WHERE➥
 surname='Wood')";
affected_rows = mysql_affected_rows(&mysql);
```

mysql_change_user

```
my_bool mysql_change_user(MYSQL *mysql, char
   *username, char *password, char *database)
```

改变MySQL当前的用户（已登录的）为另一个用户（指定用户名和口令）。还可以在同一时刻改变数据库，或打开一个新的连接，否则，当前的连接和数据库将被使用。如果操作成功，返回真值；操作失败，返回假值，在这种情况下，现有用户及其详细信息将得到维护。

例如：

```
if (! mysql_change_user(&mysql, 'guru2b', 'g00r002b', 'firstdb')) {
  printf("Unable to change user and or database!");
}
```

mysql_character_set_name

```
char *mysql_character_set_name(MYSQL *mysql)
```

返回默认字符集（通常为ISO-8859-1或Latin1）的名称。

例如：

```
printf("The default character set is: %s \n", mysql_character_set_name(&mysql));
```

mysql_close

```
void mysql_close(MYSQL *mysql)
```

关闭连接并释放资源。

例如：

```
mysql_close(&mysql);
```

mysql_connect

```
MYSQL *mysql_connect(MYSQL *mysql, const char *host, const char *user,
                     const char *passwd)
```

用于与MySQL进行连接。不建议使用此函数，应使用mysql_real_connect()。

mysql_create_db

```
int mysql_create_db(MYSQL *mysql, const char *db)
```

用来创建数据库。不建议使用此函数，应使用mysql_query()。

mysql_data_seek

```
void mysql_data_seek(MYSQL_RES *res, unsigned int offset)
```

移动内部的、与mysql_store_result()返回的结果有关的行的指针（第一行从0开始）到一个新的位置。指针的偏移量是目的行，从0开始。

例如：

```
mysql_data_seek(results, mysql_num_rows(results)-1);
```

mysql_debug

```
mysql_debug(char *debug)
```

为了使用，MySQL客户程序必须已经被调试编译过。使用Fred Fish的调试库。

例如：

```
/* Traces application activity in the file debug.out */
mysql_debug("d:t:O,debug.out");
```

mysql_drop_db

```
int mysql_drop_db(MYSQL *mysql, const char *db)
```

用来撤销数据库。不建议使用此函数，应使用mysql_query()。

mysql_dump_debug_info

```
int mysql_dump_debug_info(MYSQL *mysql)
```

将连接调试信息写入日志。为了能够做到这点，连接需要SUPER权限。如果成功，返回0；否则返回一个非0值。

例如：

```
result = mysql_dump_debug_info(&mysql);
```

mysql_eof

```
my_bool mysql_eof(MYSQL_RES *result)
```

检查是否最后一行数据被读取过。不建议使用此函数，应使用mysql_err()或mysql_errno()。

mysql_errno

```
unsigned int mysql_errno(MYSQL *mysql)
```

返回最近一次API函数的错误代码，如果没有错误发生，则返回0。可以使用mysql_error()函数得到当前错误的文本。

例如：

```
error = mysql_errno(&mysql);
```

mysql_error

```
char *mysql_error(MYSQL *mysql)
```

返回最近一次API函数的错误消息（使用当前服务器的语言），如果没有错误，则返回一个空字符串。如果连接中没有错误，函数返回0。

例如：

```
printf("Error: '%s'\n", mysql_error(&mysql));
```

mysql_escape_string

```
unsigned int mysql_escape_string(char *to, const char *from, unsigned int
length)
```

返回包含所有可以中断查询操作的转义字符（这些字符前有反斜线符号）的字符串。用mysql_real_escape_string()作为替代函数，因为它使用当前的字符集。

mysql_fetch_field

```
MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result)
```

返回当前字段的字段数据。可以反复调用这个函数，返回结果中的字段后面的数据。当没有字段返回的时候，此函数返回空值。

例如：

```
while((field = mysql_fetch_field(results))) {
    /* .. process results by accessing field->name, field->length etc */
}
```

mysql_fetch_field_direct

```
MYSQL_FIELD * mysql_fetch_field_direct(MYSQL_RES * result,➥
 unsigned int field_number)
```

返回指定字段（从0开始）的字段数据。

例如：

```
/* Return the second field */
field = mysql_fetch_field_direct(results, 1);
```

mysql_fetch_fields

```
MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES * result)
```

从结果的每个字段中返回字段数组。

例如：

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *fields;

/* 返回 the number of fields in the result */
num_fields = mysql_num_fields(result);

/* 返回 an array of field data */
fields = mysql_fetch_fields(result);

/* ... Access field data as fields[0].name, fields[1].table and so on */
```

mysql_fetch_lengths

```
unsigned long *mysql_fetch_lengths(MYSQL_RES *result)
```

从当前的行中（用mysql_fetch_row()调用）返回字段长度的数组，如果出现错误，返回无效值。

只有这个函数正确地返回二进制字段（例如，BLOB）的长度。

例如：

```
unsigned long *lengths;

/* Return the next row of data */
row = mysql_fetch_row(results);

/* Return the array of lengths */
length_array = mysql_fetch_lengths(results);

/* ... Access lengths as length_array[0], length_array[1] and so on */
```

mysql_fetch_row

```
MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)
```

返回结果中下一行数据，如果没有行了或出现错误，则返回无效值。

例如：

```
MYSQL_ROW row;

row = mysql_fetch_row(results);
/* Access the row data as row[0], row[1] and so on
```

mysql_field_count

```
unsigned int mysql_field_count(MYSQL *mysql)
```

返回最后被执行的查询操作中的字段个数。它允许你在出现错误时，决定值NULL是从mysql_use_result()返回，还是从mysql_store_result()返回，因为它不能返回结果（一个非SELECT型的查询语句）。为查看成功返回的结果集中的字段个数，使用mysql_num_fields()。

例如：

```
results = mysql_store_result(&mysql);
/* test if no result set found */
if (results == NULL) {
  /* if no result, test whether  the field count was zero or not.
  if (mysql_field_count(&mysql) > 0) {
    /* the query was a SELECT-type, so the null store_result is an error */
  }
  else {
    /* no error, as the query was an INSERT-type, which 返回 no fields */
  }
}
```

mysql_field_seek

```
MYSQL_FIELD_OFFSET mysql_field_seek(MYSQL_RES *result, MYSQL_FIELD_OFFSET offset)
```

移动内部字段指针（从0开始）到指定的字段。下一次使用指定的字段调用mysql_fetch_field()。这个函数返回前一个字段位置的指针。

mysql_field_tell

```
MYSQL_FIELD_OFFSET mysql_field_tell(MYSQL_RES *result)
```

返回当前字段指针的位置。

例如：

```
/* Record the current position */
current_pos = mysql_field_tell(results);
```

mysql_free_result

```
void mysql_free_result(MYSQL_RES *result)
```

释放分配给结果集的资源。

mysql_get_client_info

```
char *mysql_get_client_info(void)
```

返回包含客户程序的MySQL库的版本的字符串。

例如：

```
/* Displays - Client library version is: 4.0.2 (for example) */
printf("Client library version is: %s\n", mysql_get_client_info());
```

mysql_get_host_info

```
char *mysql_get_host_info(MYSQL *mysql)
```

返回包含连接信息的字符串。

例如：

```
/* Displays - Type of connection: Localhost via UNIX socket (for example) */
printf("Type of connection: %s", mysql_get_host_info(&mysql));
```

mysql_get_proto_info

```
unsigned int mysql_get_proto_info(MYSQL *mysql)
```

返回包含连接所使用的协议版本的整数值（例如，10）。

例如：

```
/* displays - Protocol version: 10 (for example) */
printf("Protocol version: %d\n", mysql_get_proto_info(&mysql));
```

mysql_get_server_info

```
char *mysql_get_server_info(MYSQL *mysql)
```

返回包含MySQL服务器版本的字符串（例如，4.0.3）。

例如：

```
/* displays - Server version: 4.0.3-beta-log (for example) */
printf("Server version: %s\n", mysql_get_server_info(&mysql));
```

mysql_info

```
char *mysql_info(MYSQL *mysql)
```

返回包含有关最近一次查询操作的详细信息的字符串。详细信息中包含记录、匹配的行、变更和警告信息。

例如：

```
/* Query info String format: Rows matched: 19 Changed: 19 Warnings: 0
(for example) */
printf("Query info: %s\n", mysql_info(&mysql));
```

mysql_init

```
MYSQL *mysql_init(MYSQL *mysql)
```

返回已初始化的MySQL的句柄，为mysql_real_connect()做好准备。

参数可以是空指针，在这种情况下，将建立数据类型结构，或建立指向现有MYSQL结构的指针。如果创建了结构，则函数mysql_close()释放来自结构的资源。如果你传递现有结构，就需要在连接被关闭之后，自己释放资源。

mysql_insert_id

```
my_ulonglong mysql_insert_id(MYSQL *mysql)
```

返回包含最近被插入的AUTO_INCREMENT值的值，如果最近的查询操作的确插入一个自动增量值，则函数返回0。

例如：

```
last_auto_increment = mysql_insert_id(&mysql);
```

mysql_kill

```
int mysql_kill(MYSQL *mysql, unsigned long process_id)
```

MySQL终止由process_id指定的线程的请求。如果操作成功，返回0；操作失败返回非0值。

要求具有PROCESS权限。

例如：

```
kill = mysql_kill(&mysql, 1293);
```

mysql_list_dbs

```
MYSQL_RES *mysql_list_dbs(MYSQL *mysql, const char *wild)
```

返回的结果集中，包含匹配wild常规表达式（等价于SQL语句SHOW DATABASES LIKE 'wild'）的服务器上的数据库名称。如果出现错误，返回空值。如果传递一个空指针，此函数返回所有的数据库。

例如：

```
MYSQL_RES database_names;

/* 返回 a list of all databases with 'db' in the name */
database_names = mysql_list_dbs(&mysql, "%db%");

/* ... Don't forget to free the resources at a later stage
mysql_free_result(database_names);
```

mysql_list_fields

```
MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const char *wild)
```

返回的结果集中，包含匹配wild常规表达式（等价于SQL语句SHOW COLUMNS FROM tablename LIKE 'wild'）的指定表中的字段名称。如果出现错误，返回空值。如果传递一个空指针，此函数返回所有的字段。

例如：

```
MYSQL_RES field_names;

/* 返回 a list of all fields with 'name' in the name */
field_names = mysql_list_fields(&mysql, "customer", "%name%");

/* ... Don't forget to free the resources at a later stage
mysql_free_result(field_names);
```

mysql_list_processes

```
MYSQL_RES *mysql_list_processes(MYSQL *mysql)
```

返回的结果集中，包含对最近运行的数据库服务器线程的说明，如果出现错误，返回空值。与SHOW PROCESSLIST语句返回的信息相同（信息中包括进程ID、用户名、主机名、数据库、方式、时间、状态和信息）。可以与平常一样，为了访问结果而把此函数结果传递到mysql_fetch_row()。

例如：

```
MYSQL_RES *threadlist;
MYSQL_ROW row
threadlist = mysql_list_processes(&mysql);

row = mysql_fetch_row(threadlist);
/* Access the thread data as row[0], row[1] and so on

/* ... Don't forget to free the resources at a later stage
mysql_free_result(threadlist);
```

mysql_list_tables

```
MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)
```

返回的结果集中，包含匹配wild常规表达式（等价于SQL语句SHOW TABLES LIKE 'wild'）的当前数据库中表的名称。如果出现错误，返回空值。如果传递一个空指针，此函数返回所有的字段。

例如：

```
MYSQL_RES tablelist;

/* 返回 a list of all tables with 'customer' in the name */
tablelist = mysql_list_tables(&mysql, "%customer%");

/* ... Don't forget to free the resources at a later stage
mysql_free_result(tablelist);
```

mysql_num_fields

```
unsigned int mysql_num_fields(MYSQL_RES *result)
```

返回查询结果中字段的个数。使用mysql_field_count()函数来检查错误，并使用这个函

数检查成功返回的结果集中的字段个数。

例如：

```
num_fields = mysql_num_fields(results);
```

mysql_num_rows

```
int mysql_num_rows(MYSQL_RES *result)
```

返回查询结果中的行数（如果mysql_use_result()被用来得到结果集，只能得到日期结果）。

例如：

```
num_rows = mysql_num_rows(results);
```

mysql_options

```
int mysql_options(MYSQL *mysql, enum mysql_option option, void *value)
```

为要完成的连接设置额外的连接参数。此函数可以被调用多次，并在mysql_init()之后和mysql_real_connect()之前被调用。如果成功，函数返回0；如果一个非法参数被传递给这个函数，则返回非0值。函数的参数如下所示：

MYSQL_OPT_CONNECT_TIMEOUT 无符号的int *，指定连接超时的秒数。

MYSQL_OPT_COMPRESS 使用压缩的客户机/服务器协议。

MYSQL_OPT_LOCAL_INFILE 如果指针指向非零的无符号整数或不提供任何数值，则可以启用指向uint. LOAD DATA LOCAL的可选指针。

MYSQL_OPT_NAMED_PIPE 在Windows NT中，使用命名管道连接服务器。

MYSQL_INIT_COMMAND 一个char *，指定查询操作在连接被建立的时候开始运行（包括自动地重建连接）。

MYSQL_READ_DEFAULT_FILE 一个char *，参数来自命名文件，而不是通常的配置文件（my.cnf或my.ini）。

MYSQL_READ_DEFAULT_GROUP 一个char *，参数从配置文件（my.cnf或my.ini，或由MYSQL_READ_DEFAULT_FILE设置）内部的命名组中读取。

例如：

```
MYSQL mysql;
mysql_init(&mysql);

/* use compression, and flush the tables upon connecting */
mysql_options(&mysql, MYSQL_OPT_COMPRESS, 0 );
mysql_options(&mysql, MYSQL_INIT_COMMAND, "FLUSH TABLES" );

/* ... continue and connect */
if(!mysql_real_connect(&mysql, "localhost", "guru2b", "g00r002b",➥
 "firstdb", 0,NULL,0)) {
    printf("The following connection error occurred %s\n",
          mysql_error(&mysql));
}
```

mysql_ping

```
int mysql_ping(MYSQL *mysql)
```

如果MySQL服务器启动了，则返回0。如果没有启动，则返回非零值。如果ping操作失败了，程序将试着重新连接。

mysql_query

```
int mysql_query(MYSQL *mysql, const char *query)
```

执行指定的查询语句。如果成功，返回0；如果出现错误，返回非零值。为了用二进制数据（数据中可以包含空字符）执行查询操作，需要使用mysql_real_query()。也应该使用这个函数来撤销和创建数据库，并代替mysql_create_db()和mysql_drop_db()函数。

可以用mysql_store_result()或mysql_use_result()函数检索结果，如果这些函数可以使用。例如：

```
query_result = mysql_query(&mysql, "CREATE DATABASE seconddb');
mysql_real_connect
MYSQL *mysql_real_connect(MYSQL *mysql, const char *host, const char *user,➥
 const char *passwd, const char *db, uint port, const char *unix_socket,➥
 uint client_flag)
```

用指定的参数建立与MySQL服务器的连接，如下所示：

MYSQL *mysql 现有的MYSQL结构在调用时创建。

const char *host MySQL服务器的主机名或IP地址。可以是空字符串，与从同一台机器上的客户端连接到MySQL时相同。

const char *user 用户名。

const char *passwd 指定用户的密码。

const char *db 要连接的数据库（可以是空）。

uint port 指定用于TCP/IP连接的端口（如果被使用）。0是默认端口。

const char *unix_socket 为本地连接指定UNIX套接字的文件名或命名管道。这个参数可以为空以接受默认值。

还可以传递下列任何可选的标记：

CLIENT_FOUND_ROWS 用来说明：mysql_affected_rows()将返回匹配查询条件的行数，而不是实际被改变的行的数目。

CLIENT_IGNORE_SPACE 允许在函数名后面放置空格（这是为所有函数预设保留字的结果）。

CLIENT_INTERACTIVE 指定MySQL在interactive_timeout秒以后撤消连接，而不是在wait_timeout秒后（这是两个mysql变量）。在运行查询语句之前，客户程序等待用户交互式输入的较长一段时间里，通常使用这个函数。

CLIENT_NO_SCHEMA 主要用于ODBC，并且不允许在查询操作中使用database_name.tablename.fieldname语法。

CLIENT_COMPRESS 确保连接使用数据压缩技术。

CLIENT_ODBC 告诉MySQL，客户端是一个ODBC客户端，会引起一些操作上的改变。

CLIENT_SSL 确保连接使用安全套接层（SSL）加密技术，只要这个函数已经被编译入服务器。

例如：

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql, MYSQL_OPT_COMPRESS, 0 );
mysql_options(&mysql, MYSQL_INIT_COMMAND, "FLUSH TABLES" );
if(!mysql_real_connect(&mysql, "localhost", "guru2b", "g00r002b",➥
 "firstdb", 0, NULL,0)) {
    printf("The following connection error occurred %s\n",
          mysql_error(&mysql));
}
```

mysql_real_escape_string

```
unsigned long mysql_real_escape_string(MYSQL *mysql, char *new_string,?
 char *old_string, unsigned long old_string_length)
```

对字符串（长度为string_length的old_string）进行转义，这样可以在MySQL查询操作中使用，替代new_string（至少比原先那个字符串长度的两倍还要多1个字节，以防每一个字符都需要转义，还要把空字符串考虑在内）中的结果。转义的字符是NUL（ASCII 0）、\n、\r、\、'、"和Ctrl+Z。结果返回新字符串的长度。

例如：

```
/* the original query is 4 bytes (a,b,c and the null character) */
char *old_query = "abc\000";

/*the new length must be at least4*2+1 byted */
char new_query[9];
int new_length;
/* 返回 the new length */
new_query_length = mysql_real_escape_string(&mysql, new_query, old_query, 4);
```

mysql_real_query

```
int mysql_real_query(MYSQL *mysql, const char *query, unsigned long length)
```

执行查询操作（还可以使用二进制数据），还同时指定长度（不包括空字符）。如果适用，你可以用函数mysql_store_result()或mysql_use_result()检索结果。

例如：

```
query_result = mysql_real_query(&mysql, "CREATE DATABASE seconddb");
```

mysql_reload

```
int mysql_reload(MYSQL *mysql)
```

这是一个不建议使用的函数，它假设已连接的用户具有Reload许可权，可以重新加载授权的表。使用mysql_query()功能更恰当。

mysql_row_seek

```
MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET offset)
```

将内部的行指针移动到指定的行，返回原来的行指针。MYSQL_ROW_OFFSET应该是从mysql_row_tell()功能或mysql_row_seek()功能中得出的结构，不只是一个行号（因此操作者应该使用mysql_data_seek()功能）。

例如：

```
current_location = mysql_row_seek(result,row_offset);
```

mysql_row_tell

```
MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result)
```

返回行指针当前的位置。操作者可以与mysql_row_seek()一起使用，移动到指定的行。在mysql_store_result()后使用，而不是mysql_use_result()。

例如：

```
MYSQL_ROW_OFFSET current_position = mysql_row_tell(results);

/* A little later..., move back to this position */
moved_position = mysql_row_seek(result,current_position);
```

mysql_select_db

```
int mysql_select_db(MYSQL *mysql, const char *db)
```

改变当前的数据库为指定的数据库（假设用户具有更改的许可权）。如果成功，则返回0。如果出现错误，则返回非0的值。

例如：

```
mysql_select_db(&mysql, "seconddb");
```

mysql_shutdown

```
int mysql_shutdown(MYSQL *mysql)
```

要求MySQL服务器关闭。用户必须具备SHUTDOWN的特权。如果成功则返回0，如果出现错误则返回非0值。

例如：

```
mysql_shutdown(&mysql);
```

mysql_stat

```
char *mysql_stat(MYSQL *mysql)
```

返回包含服务器状态的字符串。它包括已启动的时间、线程、问题、缓慢的查询、打开的、刷新表、开放式表和平均每秒进行的查询。

例如：

```
/* displays (for example):
Uptime: 109  Threads: 2  Questions: 199  Slow queries: 1  Opens: 4
Flush tables: 1  Open tables: 2  Queries per second avg: 1.826b */
printf("Server status: %s\n", mysql_stat(&mysql));
```

mysql_store_result

```
MYSQL_RES *mysql_store_result(MYSQL *mysql)
```

对于所有返回数据的查询语句，你需要调用这个函数或mysql_use_result()。这个函数把查询结果存储在MYSQL_RES结构中。如果出现错误或没有返回数据（如INSERT的CREATE DATABASE语句之后），则返回空值。应该使用mysql_field_count()计算查询语句返回的期望的字段的个数。如果值非零（当查询操作没有像期望那样返回数据时），会发生错误。

然后，释放资源。

例如：

```
MYSQL_RES results;
mysql_query(&mysql, "SELECT first_name, surname FROM customers");
results = mysql_store_result(&mysql);
/* later ... */
mysql_free_result(results);
```

mysql_thread_id

```
unsigned long mysql_thread_id(MYSQL * mysql)
```

返回当前连接的线程ID，通常用于使用mysql_kill()关闭线程。

例如：

```
thread_id = mysql_thread_id(&mysql);
```

mysql_use_result

```
MYSQL_RES *mysql_use_result(MYSQL *mysql)
```

对于所有的可以返回数据的查询，操作者都需要调用这个功能或mysql_store_result()。此功能逐行读取数据，并不像mysql_store_result()那样一次读取所有的数据。因此它比较快，但是它直到所有的数据都返回来后，才允许其他的查询运行，使锁定与通常情况下相比更像是个麻烦。如果出现错误或查询没有返回数据（如在INSERT的CREATE DATABASE后），则返回一个空值。操作者应该使用mysql_field_count()来统计查询中字段的数目。如果不是

0（当查询没有返回任何数据时），则表明已经出现了错误。

例如：

```
MYSQL_RES results;
mysql_query(&mysql, "SELECT first_name,surname FROM customer");
results = mysql_use_result(&mysql);

/* can now use mysql_fetch_row() to access data one row at a time *

/* later ... */
mysql_free_result(results);
```

一个简短的C API例子

程序清单G.1显示了一个C API利用基本组件从应用数据库中返回数据的例子。操作者需要对它进行编译和运行。各个系统的编译虽然不同，但是看起来与下面这个例子差不多：

```
gcc -o example example.c -I/usr/local/mysql/include/mysql➥
 -L/usr/local/mysql/lib/mysql -lmysqlclient -lz
```

程序清单G.1 example.c

```
#include <stdio.h>
#include <mysql.h>
/* the two basic includes */

/* the main function */
int main(char **args) {
  MYSQL_RES *query_result;
  MYSQL_ROW row;
/* db_handle is the connection to the database, and will be used by
 many of the functions that follow */
  MYSQL *db_handle, mysql;
  int query_error;

  /* initialize and open the connection */
  mysql_init(&mysql);
  db_handle = mysql_real_connect(&mysql,"localhost", "guru2b",➥
   "g00r002b", "firstdb", 0, 0, 0);

  /* if the connection failed, then display the error and exit. */
  if (db_handle == NULL) {
    printf(mysql_error(&mysql));
    return 1;
  }
  query_error = mysql_query(db_handle,"SELECT first_name,surname FROM customer");

  /* if the query error is not 0 (no error), display the error and exit */
  if (query_error != 0) {
```

```
    printf(mysql_error(db_handle));
    return 1;
  }

  /* Return a query result */
  query_result = mysql_store_result(db_handle);

  /* Loop through and display each row in the query result */
  while (( row = mysql_fetch_row(query_result)) != NULL ) {
    printf("Name: %s %s\n",(row[0] ? row[0] : "NULL"),➥
    (row[1] ? row[1] : "NULL"));
  }
  /* free the resources associated with the query result */
  mysql_free_result(query_result);

  /* close the connection */
  mysql_close(db_handle);
}
```

附录H ODBC和.NET

MySQL可以通过开放式数据库连接（Open Database Connectivity，ODBC）与一些环境或语言相联接，而这些环境和语言可以不具备其自主开发的与MySQL互联的应用程序接口（API）和驱动程序。ODBC是用来连接关系型数据库并与之互操作的API，得到了广泛的应用。它并不依赖于数据库和语言。与MySQL一起，它一般会被用来连接类似Microsoft Access或Visual Basic的工具。为了实现上述功能，操作者需要安装MyODBC驱动程序。在MySQL的Web站点（www.mysql.com/downloads/api-myodbc.html）上可以找到这个驱动程序，并且还有全部的安装指南。在Windows上安装这个驱动程序只需要下载其可执行文件，然后运行这个文件。

本附录讲解了如何在UNIX和Windows上建立数据资源，如何从Microsoft Access向MySQL输出数据。可以为经验丰富的编程人员提供MyODBC功能的参考资料，并且为ODBC的连接、插入和利用VB .NET、C# .NET、DOA、ADO和RDO来选择记录等应用提供简单的脚本实例。本附录并不是ODBC和如何使用ODBC的完整说明。更多的信息，可以从几个方面获得：从Microsoft站点（www.microsoft.com/data/odbc/）上检索ODBC的文档，随开发环境携带的ODBC文档，或者MySQL站点上的信息（www.mysql.com/products/myodbc/）。

理解数据源

数据源可以是文件库的路径，服务器或对于MySQL来说，是一个MySQL数据库。连接信息与数据源（例如那些存储在Windows注册表中的数据源）相关联。为了连接到数据源，ODBC驱动程序管理器（ODBC Driver Manager）寻找与DSN相关的连接信息，并且使用这个连接信息进行连接。通过ODBC连接，当前具备DSN并不总是必要的。操作者可以直接指定驱动程序。除了稍后看到的DAO VB的例子，本附录中所有的例子连接时都没有使用预先设置的DSN。

在Windows中建立数据源

使一个Windows应用程序通过ODBC连接到MySQL，首先需要建立一个数据源：

1. 选择Start➤Settings➤Control Panel。
2. 根据Windows的版本，操作者可以选择ODBC、32位的ODBC或替代的管理工具或数据源（Administrative Tools and Data Sources，ODBC）。
3. 点击Add。
4. 从显示的列表中，选择为MyODBC安装的MySQL驱动程序，然后点击Finish。
5. 出现驱动程序的缺省配置屏幕。完成所需要的细节内容。Windows DSN可以是操作者希望的任何一个，主机、数据库、用户名、密码和端口细节信息都是通常连接到MySQL所需要的细节信息。操作者还可以指定连接到服务器时使用的SQL命令。

6. 操作者可以点击Options按钮选择各种选项，来满足每个应用程序的（包括那些并不是100%ODBC兼容的程序）要求。例如，当前运行Microsoft Access，操作者应当检查Return Matching Rows选项。操作者可能需要进行试验，使得各项工作得以平滑地进行。并且操作者还应当查看MySQL的Web站点，在那里会有很多常用应用程序的最新选项（或参数）。
7. 点击OK，数据源将被加入。
8. 根据ODBC的版本，操作者可能能够通过点击Test Data Source来测试其连接。如果测试失败了，那么需要对连接进行检查。

在UNIX上建立数据源

在UNIX上，操作者可以直接编辑文件ODBC.INI来建立数据源。
举例来说：

```
[ODBC Data Sources]
myodbc     = MySQL ODBC 3.51 Driver DSN

[myodbc]
Driver       = /usr/local/lib/libmyodbc3.so
Description  = MySQL ODBC 3.51 Driver DSN
SERVER       = localhost
PORT         =
USER         = root
Password     = g00r002b
Database     = firstdb
OPTION       = 3
SOCKET       =
```

在UNIX中为了对非ODBC兼容的应用程序设置不同的参数，操作者可以在表H.1的基础上设置OPTION的取值。

设置连接的参数

如果没有图形界面，操作者可以使用表H.1中的参数值进行连接。结合参数只需简单地将取值加在一起（于是3就成了1和2的结合）。

表H.1　连接参数

位	描述
1	无法控制收到的实际列宽度
2	无法控制收到的实际影响到的行的数量（将代之以找到的行的数量）
4	在C:\myodbc.log或/tmp/myodbc.log中加入调试（debug）记录
8	将结果和参数的包极限设置为无限制
16	不提示问题

（续表）

位	描述
32	切换动态指针支持（启动或关闭动态指针）
64	忽略结构的数据库名，如databasename.tablename.fieldname
128	强制使用ODBC管理器指针的实验选项
256	实验选项，关闭对扩展存取的应用
512	CHAR字段被填补到字段的全部长度之中
1024	SQLDescribeCol()函数返回列的全名
2048	使用压缩协议
4096	使服务器忽略函数名和括号之间的空格，并且使所有函数名成为关键字
8192	使用命名管道连接到运行NT/2000/XP的服务器
16384	LONGLONG字段被转换成INT字段
32768	实验选项，从SQLTables()函数中返回作为user的Table_qualifier和Table_owner
65536	读取MySQL配置文件中的client和odbc组的参数
131072	添加额外的安全性检查
262144	关闭事务
524288	在调试（debug）模式下，对文件c:\myodbc.sql或/tmp/myodbc.sql启动查询日志

从Microsoft Access向MySQL输出数据

有许多新手在开始使用数据库时，面对的是Microsoft Access，常见的ODBC应用是对MySQL进行升级，从Access输出数据。这行下面的这些步骤可以完成上述这些内容（第1步和第2步只需要在第一次这样做时执行）：

1. 安装MyODBC驱动程序。
2. 建立指向希望向其输出数据的目标MySQL服务器的数据源，如同在前面部分“在Windows上建立数据源”中讲述的。
3. 加载Microsoft Access。
4. 打开Microsoft Access数据库窗口，选择希望输出的目标表。
5. 点击File➤Export并选择Save as Type下拉列表中的ODBC Databases ()。
6. 如果操作者希望改变表的名字，选择新的名字然后点击OK。
7. 选择第2步中定义的数据源，并且点击OK。
8. 如果数据源连接的详细信息不正确，操作者如果需要改变它们，也可以在这里进行。这里要记住赋予MySQL许可权才能够准许访问。详细的内容可以参见第14章“数据库安全性”。

使用ODBC

接下来的一些例子演示了在不同的编程环境中通过ODBC向MySQL数据库插入记录和从MySQL数据库中选择记录。第一个例子显示了直接建立的连接，并且DAO例子显示了通过数据源建立的一个连接。

为了运行这些范例，操作者需要安装MyODBC和DSN（只对DAO的例子——参见前面的“在Windows中建立数据源”部分），并且还要有恰当的环境（.NET、Visual Basic等等）。

警告：确保格式相同，否则这些例子不能工作。

简单的.NET VB例子

程序清单H.1使用VB.NET和ODBC连接到MySQL服务器，并插入一条记录，选择并打印结果。操作者需要.NET环境指定的设置才能对程序代码进行编译。下面是一个例子（停顿是为了使编译参数可视）：

```
set path=%path%;C:\WINNT\Microsoft.NET\Framework\v1.0.3705➥
  C:\WINNT\Microsoft.NET\Framework\v1.0.3705\csc /t:exe➥
  /out:odbc_cnet.exe odbc_cnet.cs /r:"C:\Program?
  Files\Microsoft.NET\Odbc.Net\Microsoft.Data.Odbc.dll"➥
  pause
```

程序清单H.1　dbnet.vb

```
Imports Microsoft.Data.Odbc
Imports System

Module mysql_vbnet
  Sub Main()
    Try

      'Set the arguments for connecting to a MySQL firstdb database➥
        with MyODBC 3.51
      Dim MySQLConnectionArgs As String = " DRIVER={MySQL ODBC 3.51 Driver};➥
        SERVER=www.testhost.co.za;DATABASE=firstdb;UID=guru2b;➥
        PASSWORD=g00r002b;OPTION=0"

      'Open the ODBC connection and ODBC command
      Dim MySQLConnection As New OdbcConnection(MySQLConnectionArgs)
      MySQLConnection.Open()
      Dim MySQLCommand As New OdbcCommand()
      MySQLCommand.Connection = MySQLConnection

      'Insert a record into the customer table
      MySQLCommand.CommandText = "INSERT INTO customer➥
        (first_name, surname) VALUES('Frank', 'Weiss')"
      MySQLCommand.ExecuteNonQuery()
```

```
        'select a record from the customer table, return the results,
        'loop through the results displaying them
        MySQLCommand.CommandText = "SELECT id,first_name,surname FROM customer"
        Dim MySQLDataReader As OdbcDataReader
        MySQLDataReader = MySQLCommand.ExecuteReader
        While MySQLDataReader.Read
          Console.WriteLine (CStr(MySQLDataReader("id")) & ":" &➥
            CStr(MySQLDataReader("first_name")) & " " & CStr(MySQLDataReader("surname")))
        End While

      'If theres an ODBC Exception, catch it
      Catch MySQLOdbcException As OdbcException
        Console.WriteLine (MySQLOdbcException.ToString)

      'If there a program exception, catch it
      Catch AnyException As Exception
        Console.WriteLine (AnyException.ToString)
      End Try
    End Sub
  End Module
```

一个简单的.NET C#例子

程序清单H.2使用C# .NET和ODBC来连接到MySQL服务器，并插入一条记录，选择并打印结果。

操作者需要.NET环境指定的设置才能对程序代码进行编译。下面是一个例子（停顿是为了使编译参数可视）：

```
C:\WINNT\Microsoft.NET\Framework\v1.0.3705\csc /t:exe➥
  /out:odbc_cnet.exe odbc_cnet.cs /r:"C:\Program➥
  Files\Microsoft.NET\Odbc.Net\Microsoft.Data.Odbc.dll"
pause
```

程序清单H.2 dbnet.cs

```
using Console = System.Console;
using Microsoft.Data.Odbc;

namespace MyODBC {
  class MySQLCSharp {
    static void Main(string[] args) {
      try {
        // Set the arguments for connecting to a MySQL firstdb database➥
          with MyODBC 3.51

        string MySQLConnectionArgs = "DRIVER={MySQL ODBC 3.51 Driver};➥
          SERVER=www.testhost.co.za;DATABASE=firstdb;UID=guru2b;➥
          PASSWORD=g00r002b;OPTION=0";
```

```
            // Open the ODBC connection and ODBC command
            OdbcConnection MySQLConnection = new OdbcConnection(MySQLConnectionArgs);
            MySQLConnection.Open();

            // Insert a record into the customer table
            OdbcCommand MySQLCommand = new OdbcCommand("INSERT INTO➥
              customer (first_name, surname) VALUES('Frank', 'Weiss')",➥
              MySQLConnection);
            MySQLCommand.ExecuteNonQuery();

            // select a record from the customer table, return the results,
            // loop through the results displaying them
            MySQLCommand.CommandText = "SELECT id,
        first_name, surname FROM customer";
            OdbcDataReader MySQLDataReader;
            MySQLDataReader =  MySQLCommand.ExecuteReader();
            while (MySQLDataReader.Read()) {
              Console.WriteLine("" + MySQLDataReader.GetInt32(0) + ":" +➥
                MySQLDataReader.GetString(1) + " " + MySQLDataReader.GetString(2));
            }

            // Close all resources
            MySQLDataReader.Close();
            MySQLConnection.Close();
          }
          // If theres an ODBC Exception, catch it
          catch (OdbcException MySQLOdbcException) {
            throw MySQLOdbcException;
          }
        }
      }
    }
```

一个简单的ADO VB例子

程序清单H.3使用VB和ADO通过ODBC来连接到MySQL服务器，并插入一条记录（直接或间接添加到结果集中），选择并打印结果。

为了能够在Visual Basic中访问ADO 2.0对象，操作者需要设置对包含在MSADO15.DLL中的ADODB类型库的关联。它作为Microsoft ActriveX Data Objects 2.0库出现在References对话框中（在Project菜单中）。

代码需要出现在表格（form）中，程序Debug.Print才可以运行。换句话说，操作者可以将它变成MsgBox来使得代码可以执行。

程序清单H.3 dbado.vb

```
Private Sub MySQLADO()
  Dim MySQLConnection As ADODB.Connection
```

```
    Dim Results As ADODB.Recordset

    Dim SQLQuery As String

    'Open a connection using ADODB and set the connection string
    'for connecting to a MySQL firstdb database with MyODBC 3.51
    Set MySQLConnection = New ADODB.Connection
    MySQLConnection.ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};➥
      SERVER=www.testhost.co.za;DATABASE=customer;UID=guru2b;➥
      PWD=g00r002b;OPTION=0"  MySQLConnection.Open

    Set Results = New ADODB.Recordset
    Results.CursorLocation = adUseServer

    'There are two common ways of inserting - the first is the direct insert
    SQLQuery = "INSERT INTO customer (first_name, surname) VALUES➥
      ('Werner', 'Christerson')"
    MySQLConnection.Execute SQLQuery

    'The second way of inserting is to add to a result set using the
    'AddNew method. First return a result set
    Results.Open "SELECT * FROM customer", MySQLConnection,➥
      adOpenDynamic, adLockOptimistic
    Results.AddNew
    Results!first_name = "Lance"
    Results!surname = "Plaaitjies"
    Results.Update
    Results.Close

    'select a record from the customer table, return the results,
    'loop through the results displaying them
    Results.Open "SELECT id, first_name, surname FROM customer", MySQLConnection
    While Not Results.EOF
      Debug.Print Results!id & ":" & Results!first_name & " " & Results!surname
      Results.MoveNext
    Wend
    Results.Close

    MySQLConnection.Close
End Sub
```

简单的RDO VB例子

程序清单H.4使用VB和RDO通过ODBC连接到MySQL服务器、插入记录、选择并打印结果。Visual Basic始终支持RDO，但是操作者仍然可能希望使用新的ADO。

为了可以访问Visual Basic中的RDO 2.0对象，操作者需要设置对包含在MSRD020.DLL中的RDO类型库的关联。它作为Microsoft Remote Data Objects 2.0出现在References对话框中（在Project菜单中）。代码需要出现在表格（form）中，程序Debug.Print才可以运行。换句话说，操作者可以将它变成MsgBox来使得代码可以执行。

程序清单H.4 dbrdo.vb

```
Private Sub MySQLRDO()
  Dim Results As rdoResultset
  Dim MySQLConnection As New rdoConnection
  Dim SQLQuery As String

  'Open a connection using RDO and set the connection string
  'for connecting to a MySQL firstdb database with MyODBC 3.51

  MySQLConnection.Connect = "DRIVER={MySQL ODBC 3.51 Driver};➥
    SERVER=www.testhost.co.za;DATABASE=firstdb;UID=guru2b;➥
    PWD=g00r002b;OPTION=0"

  MySQLConnection.CursorDriver = rdUseOdbc
  MySQLConnection.EstablishConnection rdDriverNoPrompt

  'There are two common ways of inserting - the first is the direct insert
  SQLQuery = "INSERT INTO customer (first_name, surname) VALUES➥
    ('Lance', 'Plaaitjies')"
  MySQLConnection.Execute SQLQuery, rdExecDirect

  'The second way of inserting is to add to a result set using the
  'AddNew method. First return a result set
  SQLQuery = "SELECT * FROM customer"
  Set Results = MySQLConnection.OpenResultset(SQLQuery, rdOpenStatic,➥
    rdConcurRowVer, rdExecDirect)
  Results.AddNew
  Results!first_name = "Werner"
  Results!surname = "Christerson"
  Results.Update
  Results.Close

  'select a record from the customer table, return the results,
  'loop through the results displaying them
  SQLQuery = "select * from customer"
  Set Results = MySQLConnection.OpenResultset(SQLQuery, rdOpenStatic,➥
    rdConcurRowVer, rdExecDirect)
  While Not Results.EOF
    Debug.Print Results!id & ":" & Results!first_name & " " & Results!surname
    Results.MoveNext
  Wend
  'Free the result set, and the connection
  Results.Close
  MySQLConnection.Close
End Sub
```

简单的DAO VB案例

程序清单H.5使用VB和DAO通过ODBC连接到DSN、插入记录（直接和间接添加到结果集中）、选择和打印结果。Visual Basic始终支持DAO，但是操作者仍然可能希望使用新的ADO。

为了可以访问Visual Basic中的DAO对象，操作者需要设置对包含在DAO360.DLL中的DAO类型库的关联。它作为Microsoft DAO 3.6 Object Library出现在References对话框中（在Project菜单中）。

代码需要出现在表格（form）中，程序Debug.Print才可以运行。换句话说，操作者可以将它变成MsgBox来使得代码可以执行。

下面这个例子需要对DSN进行设置后才能运行。

程序清单H.5 dbdao.vb

```
Private Sub MySQLDAO()
  Dim Works As Workspace
  Dim MySQLConnection As Connection
  Dim Results As Recordset
  Dim SQLQuery As String

  'Open a workspace using DAO and set the connection string
  'for connecting to a DSN MySQL firstdb database with MyODBC 3.51
  Set Works = DBEngine.CreateWorkspace("MySQLWorkspace", "guru2b",➥
    "g00r002b", dbUseODBC)
  Set MySQLConnection = Works.OpenConnection("MySQLConn",➥
    rdDriverCompleteRequired, False, "ODBC;DSN=MyDAO")

  'There are two common ways of inserting - the first is the direct insert
  SQLQuery = "INSERT INTO customer (first_name, surname) VALUES➥
    ('Lance', 'Plaaitjies')"  MySQLConnection.Execute SQLQuery

  'The second way of inserting is to add to a result set using the
  'AddNew method. First return a result set
  Set Results = MySQLConnection.OpenRecordset("customer")
  Results.AddNew
  Results!first_name = "Werner"
  Results!surname = "Christerson"
  Results.Update
  Results.Close

  'Read customer table
  Set Results = MySQLConnection.OpenRecordset("customer", dbOpenDynamic)
  While Not Results.EOF
    Debug.Print Results!id & ":" & Results!first_name & " " & Results!surname
    Results.MoveNext
  Wend
  Results.Close
```

```
    MySQLConnection.Close
    Works.Close
End Sub
```

MyODBC函数

下面这一部分提供的函数可以作为经验丰富的程序员的参考。这个附录中的解释说明适用于MyODBC 3.5x。

SQLAllocConnect

为连接句柄分配内存。这个函数已经不建议使用，并且用SQLAllocHandle()来替代，SQLAllocHandle()由变元SQL_HANDLE_DBC调用。

SQLAllocEnv

从驱动器获得环境句柄。这个函数已经不建议使用，并且已经被SQLAllocHandle()所替代，SQLAllocHandle()由变元SQL_HANDLE_ENV调用。

SQLAllocHandle

```
SQLAllocHandle (handle_type, input_handle, output_handle_pointer);
```

分配一个句柄（或者是连接、描述符、环境句柄或者是语句句柄）。

handle_type可以是SQL_HANDLE_ENV（环境句柄）、SQL_HANDLE_DBC（连接句柄）或SQL_HANDLE_STMT（语句句柄）中的一个。

input_handle用来描述分配新句柄的语境。如果handle_type是SQL_HANDLE_ENV，则值为SQL_NULL_HANDLE；如果handle_type是SQL_HANDLE_DBC，则值为环境句柄；如果handle_type为SQL_HANDLE_STMT，则值为连接句柄。

output_handle_pointer是指向返回句柄所在缓存的指针。

SQLAllocStmt

为语句句柄分配内存。这个函数已经不建议使用，并且已经被SQLAllocHandle()所替代，SQLAllocHandle()由变元SQL_HANDLE_STMT调用。

SQLBindParameter

```
SQLBindParameter(statement_handle, parameter_number, parameter_type,➥
  value_type, sql_type, column_size, decimal_digits,➥
  parameter_value_pointer, buffer_length, string_length_pointer);
```

在SQL语句中用来绑定一个参数的标记。parameter_number从1开始。例如：

```
SQLUINTEGER id_ptr;
SQLINTEGER idl_ptr;
```

```
// Prepare SQL
SQLPrepare(sth, "INSERT INTO customer(id) VALUES(?)", SQL_NTS);

// Bind id to the parameter for the id column
SQLBindParameter(sth, 1, SQL_PARAM_INPUT, SQL_C_ULONG,
   SQL_LONG, 0, 0, &id_ptr, 0, &idl_ptr);

// ...
SQLExecute(sth);
```

SQLBulkOperations

```
SQLBulkOperations(statement_handle, operation);
```

执行大量的操作。

SQLCancel

```
SQLCancel(statement_handle)
```

取消指定的语句句柄上的操作。

SQLCloseCursor

```
SQLCloseCursor(statement_handle);
```

为指定的语句句柄关闭任何一个打开的临时表。

SQLColAttribute

```
SQLColAttribute (statement_handle, record_number,➥
  field_identifier, character_attribute_pointer, buffer_length,➥
  string_length_pointer, numeric_attribute_pointer);
```

描述来自结果集中的字段的属性。

变元record_number是记录的个数，从1开始。

变元field_identifier指出要被返回的字段。

变元character_attribute_pointer指向被返回的值（如果返回的值是字符串，否则不使用这个值）所在的缓存。

变元buffer_length可以包含下列值中的一种：

- 如果character_attribute_pointer指向一个字符串，则值为character_attribute_pointer（或SQL_NTS）的长度。
- 如果character_attribute_pointer指向一个二进制缓存，则值为SQL_LEN_BINARY_ATTR(length)的结果。
- 如果character_attribute_pointer指向一个固定长度的特殊数据类型，则值为SQL_IS_INTEGER、SQL_IS_UNINTEGER、SQL_SMALLINT或SQLUSMALLINT中的一种。
- 如果character_attribute_pointer指向另一个指针，则值为SQL_IS_POINTER。

变元string_length-pointer指向要被返回的（不包括空字节）、来自character_attribute_pointer的字节总数所在的缓存。

对于字符数据，如果buffer_length比要返回的字节数少，则数据被删减。对于其他情况，长度假设为32位。

变元numeric_attribute_pointer指向一个返回数值所在的整数缓存。如果返回的不是数值，则不使用这个变元。

SQLColAttributes

描述来自结果中的字段属性。此函数不建议被使用，并且已经被SQLColAttribute()替代。

SQLColumnPrivileges

```
SQLColumnPrivileges(statement_handle, catalog_name,
 catalog_name_length, schema_name, schema_name_length, table_name,
 table_name_length, column_name, column_name_length);
```

返回字段和权限列表。

SQLColumns

```
SQLColumns(statement_handle, catalog_name, catalog_name_length,
 schema_name, schema_name_length, table_name, table_name_length,
 column_name, column_name_length);
```

返回列名列表。

SQLConnect

```
SQLConnect(connection_handle, datasource_name, datasource_name_length,
user_name, user_name_length, password, password_length);
```

用指定的用户名和密码连接到数据源。

SQLDataSources

由驱动器管理器执行，此函数返回一个可获得的数据源的列表。

SQLDescribeCol

```
SQLDescribeCol(statement_handle, column_number, column_name,
 buffer_length, name_length_pointer, data_type_pointer,
 column_size_pointer, decimal_digits_pointer, nullable_pointer);
```

描述结果集中的一列。

变元column_number是结果集中的列数，从1开始计算。

变元column_name指向被返回的列名（从SQL_DESC_NAME中读取）所在的缓存。如果没有列名，则返回一个空字符串。

变元buffer_length是column_name缓存中字符的长度。

变元name_length_pointer指向column_name中返回字节数所在的缓存。如果被返回的长度比buffer_length大，则列名被删减。

变元data_type_pointer指向从SQL_DESC_CONCISE_TYPE获得的、被返回的SQL数据类型所在的缓存。如果没有可获得的数据类型，则返回SQL_UNKNOWN_TYPE。

变元column_size_pointer指向被返回的列的大小所在的缓存，如果没有获得，则返回0。

变元decimal_digits_pointer指向被返回的列的小数位数，如果没有获得，则返回0。

变元nullable_pointer指向被返回的非空值（SQL_NO_NULLS、SQL_NULLABLE或SQL_NULLABLE_UNKOWN）所在的缓存。

SQLDescribeParam

```
SQLDescribeParam(statement_handle, parameter_number, data_type_pointer,
 parameter_size_pointer, decimal_digits_pointer,
 nullable_pointer);
```

返回对参数的说明。

变元parameter_number用来指定变元个数（从1开始）。

变元data_type_pointer指向被返回的SQL数据类型所在的缓存。

变元parameter_size_pointer指向被返回的变元列容量所在的缓存。

变元decimal_digits_pointer指向被返回的列的小数位数所在的缓存，如果没有，返回0。

变元nullable_pointer指向被返回的非空值（SQL_NO_NULLS、SQL_NULLABLE或SQL_NULLABLE_UNKOWN）所在的缓存。

SQLDisconnect

```
SQLDisconnect(connection_handle);
```

关闭由连接句柄指定的连接。

SQLDriverConnect

```
SQLDriverConnect (connection_handle, window_handle, in_connection,
 in_connection_length, out_connection, out_connection_length,
 buffer_length,  prompt_flag);
```

用来与服务器相连接。不使用没有DSN的SQLConnect进行连接，而是使用特定的驱动器连接信息，或使用提示用户的连接信息。

如果没有对话框，也没有被正在使用的窗口句柄，则变元window_handle可以是上层窗口的句柄或一个空指针。

变元in_connection可以是一个完整连接、部分连接字符串或空字符串。

变元in_connection_length是字符串的长度（以字节为单位）。

变元out_connection指向被返回的连接字符串所在的缓存。

变元out_connection_length是out_connection缓存的长度。

变元buffer_length指向被返回的字符数所在的缓存。如果字符数比buffer_length长，则out_connection被删减。

变元prompt_flag指定是否驱动器必须提示更多的信息用于连接。变元可以是SQL_DRIVER_PROMPT、SQL_DRIVER_COMPLETE、SQL_DRIVER_COMPLETE_REQUIRED或SQL_DRIVER_NOPROMPT。

SQLDrivers

由驱动器管理器实现，这个函数返回所安装的驱动器的详细信息。

SQLEndTran

```
SQLEndTran(handle_type, handle, completion_type);
```

结束一个打开的事务，调用回退或提交的事务。

变元handle_type根据句柄类型（环境或连接句柄）包含SQL_HANDLE_ENV或SQL_HANDLE_DBC。

变元handle指定实际的句柄。

变元completion_type确定是否要以回退或提交的事务结束，它可以是SQL_COMMIT或SQL_ROLLBACK。

SQLError

这个函数用来返回出错的信息，但不推荐使用。可以使用SQLGetDiagRec或SQLGetDiagField来替代它。

SQLExecDirect

```
SQLExecDirect(statement_handle, sql, sql_length);
```

执行一条SQL语句。如果语句只能被执行一次，此函数的速度比SQLExecute快，因为不需要预先设置。

SQLExecute

```
SQLExecute(statement_handle);
```

执行预先准备好的语句（用SQLPrepare）。如果语句只被执行了一次，且没有预先设置，则使用SQLExecDirect。

SQLExtendedFetch

这个函数返回可以上下滚动的结果，但不推荐使用，用SQLFetchScroll来代替。

SQLFetch

```
SQLFetch(statement_handle);
```

返回下一行数据。

SQLFetchScroll

```
SQLFetchScroll(statement_handle, fetch_type, offset);
```

返回指定行的数据。

变元fetch_type可以是SQL_FETCH_NEXT（下一行，相当于函数SQLFetch ()）、SQL_FETCH_PRIOR（前一行）、SQL_FETCH_FIRST（第一行）、SQL_FETCH_LAST（最后一行）、SQL_FETCH_ABSOLUTE（从第一行开始的偏移量）、SQL_FETCH_RELATIVE或SQL_FETCH_BOOKMARK（从当前行开始的偏移量）。

根据变元fetch_type，变元offset指定读取的行，或从第一行开始，或从当前行开始。

SQLFreeConnect

释放连接句柄。不建议使用这个函数，用SQLFreeHandle代替。

SQLFreeEnv

释放环境句柄。不建议使用这个函数，用SQLFreeHandle代替。

SQLFreeHandle

```
SQLFreeHandle(handle_type,handle);
```

释放句柄（或者是连接、描述符、环境句柄或语句句柄）。

handle_type可以是SQL_HANDLE_ENV（环境句柄）、SQL_HANDLE_DBC（连接句柄）、SQL_HANDLE_STMT（语句句柄）或SQL_HANDLE_DESC（描述符句柄）。

变元handle指定被释放的句柄。

SQLFreeStmt

```
SQLFreeStmt(statement_handle, option);
```

停止语句的处理过程。

变元option可以是SQL_CLOSE（关闭临时表，与SQLCloseCursor相同，还能够重新打开临时表）、SQL_DROP（释放语句句柄并关闭临时表，尽管这种用法已经不再推荐，可以用SQLFreeHandle替代）、SQL_UNBIND（释放捆绑SQLBindCol的所有列缓存）和SQL_RESET_PARAMS（释放由SQLBindParameter设置的所有参数的缓存）。

SQLForeignKeys

```
SQLForeignKeys(statement_handle, primary_key_catalog_name,
primary_key_catalog_name_length, primary_key_schema_name,
primary_key_schema_name_length, primary_key_table_name,
priamry_key_table_name_length, foreign_key_catalog_name,
foreign_key_catalog_name_length, foreign_key_schema_name,
foreign_key_schema_name_length, foreign_key_table_name,
foregn_key_table_name_length);
```

返回指定的表中的外部键和与指定的表链接的其他表中的外部键。

SQLGetConnectAttr

```
SQLRETURN SQLGetConnectAttr(connection_handle, attribute,
 value_pointer, buffer_length, string_length_pointer);
```

返回连接属性的值。变元attribute可以是表H.2中的一个值。

表H.2 属性及相关的value_pointer内容

属性	VALUE_POINTER内容
SQL_ATTR_AUTOCOMMIT	指出是否使用自动提交或手动提交模式。变元值可以是SQL_AUTOCOMMIT_OFF（在这种情况下，事务必须用SQLEndTran来结束），或SQL_AUTOCOMMIT_ON（默认情况）
SQL_ATTR_CONNECTION_DEAD	变元值可以是SQL_CD_TRUE（连接已终止）或SQL_CD_FALSE（连接仍旧存在）
SQL_ATTR_CONNECTION_TIMEOUT	在超时之前等待SQL语句执行的秒数。设置值为0（默认状态），意思是没有超时
SQL_ATTR_CURRENT_CATALOG	目录的名称
SQL_ATTR_LOGIN_TIMEOUT	在超时之前等待连接的秒数。值为0表示没有超时
SQL_ATTR_ODBC_CURSORS	指出驱动器管理器如何使用临时表（cursor）库
SQL_ATTR_PACKET_SIZE	指出网络数据包的大小（以字节为单位）
SQL_ATTR_QUIET_MODE	应用程序的父窗口句柄，如果驱动器没有显示对话框，则值为空
SQL_ATTR_TRACE	值可以是SQL_OPT_TRACE_OFF（默认值，不执行跟踪操作），或是SQL_OPT_TRACE_ON（执行跟踪操作）
SQL_ATTR_TRACEFILE	跟踪文件的名称
SQL_ATTR_TRANSLATE_LIB	包含SQLDriverToDataSource和SQLDataSourceToDriver函数的库的名称
SQL_ATTR_TRANSLATE_OPTION	传递给翻译DLL的32位标志值
SQL_ATTR_TXN_ISOLATION	32位的掩码，用来设置事务的隔离级别。在用这个选项调用SQLSetConnectAttr之前，不需要用SQLEndTran结束事务

变元value_pointer是指向被返回的值的指针。

buffer_length变元可以包含下面这些值中的一种：

- 如果value_pointer指向一个字符串，则值为value_pointer（或SQL_NTS）的长度。
- 如果value_pointer指向一个二进制缓存，则值是SQL_LEN_BINARY_ATTR(length)的结果。

- 如果value_pointer指向一个固定长度的特殊数据类型，则值是SQL_IS_INTEGER、SQL_IS_UNINTEGER、SQL_SMALLINT或SQLUSMALLINT 。
- 如果value_pointer指向另一个指针，则值为SQL_IS_POINTER。

变元string_length_pointer指向被返回的字符个数（不包括空字节）所在的缓存。如果字符个数比buffer_length大，则value_pointer中的值被删减。

SQLGetConnectOption

返回连接的选项值。不建议使用此函数，用SQLGetConnectAttr替代。

SQLGetCursorName

```
SQLGetCursorName(statement_handle,cursor_name,cursor_name_length,
 name_length_pointer);
```

返回与语句句柄相关联的临时表的名称。

变元cursor_name指向被返回的临时表名所在的缓存。

变元name_length_pointer指向被返回的字符个数（不包括空字节）所在的缓存。如果字符个数比name_length_pointer大，则cursor_name的值被删减。

SQLGetDiagField

```
SQLGetDiagField(handle_type, handle, record_number,
 diagnostic_identifier, diagnostic_identifier_pointer, buffer_length,
 diagnostic_identifier_length_pointer);
```

返回出错信息、警告和状态诊断信息。

handle_type可以是SQL_HANDLE_ENV（环境）、SQL_HANDLE_DBC（连接）、SQL_HANDLE_STMT（语句）或SQL_HANDLE_DESC（描述符）。

handle_argument包括指定的handle_type类型句柄。

record_number 变元是从这里返回信息的记录（从1开始）。

diagnostic_identifier可以是表H.3中所示的一个值：

表H.3 变元SQLDiagField diagnostic_identifier

DIAGNOSTIC_IDENTIFIER	说明
SQL_DIAG_CLASS_ORIGIN	返回一个字符串，指出SQLSTATE类的来源（例如，ISO9075或ODBC 3.0）
SQL_DIAG_COLUMN_NUMBER	从结果集返回列的数目，或参数集中参数的个数，从0开始计算。如果不是这两种应用方式，变量值包括SQL_NO_COLUMN_NUMBER或SQL_COLUMN_NUMBER_UNKNOWN
SQL_DIAG_CONNECTION_NAME	返回一个字符串，包含与诊断有关的连接的名称
SQL_DIAG_CURSOR_ROW_COUNT	返回临时表中行的个数

（续表）

DIAGNOSTIC_IDENTIFIER	说明
SQL_DIAG_MESSAGE_TEXT	返回带有诊断（错误或警告）信息的字符串
SQL_DIAG_NATIVE	返回本地错误代码（一个整数）。如果没有错误代码，则返回0
SQL_DIAG_NUMBER	返回可以获得的句柄状态记录的个数（当前的驱动器返回1）
SQL_DIAG_RETURNCODE	返回函数返回的代码
SQL_DIAG_ROW_COUNT	返回受SQL操作影响的行数。这些操作是由SQLExecute、SQLExecDirect、SQLBulkOperations或SQLSetPos运行的用来修改数据（如INSERT或DELETE）的操作
SQL_DIAG_ROW_NUMBER	返回结果集中的行数，或参数集中参数的个数，从1开始计算。如果不是这两种应用方式，变量值包括SQL_NO_ROW_NUMBER或SQL_ROW_NUMBER_UNKNOWN
SQL_DIAG_SERVER_NAME	返回一个字符串，包含与诊断有关的服务器的名称
SQL_DIAG_SQLSTATE	返回一个有5个字符的字符串，包含SQLSTATE代码
SQL_DIAG_SUBCLASS_ORIGIN	返回一个字符串，表明SQLSTATE子类的来源（例如，ISO9075或ODBC 3.0）

diagnostic_id_pointer指向被返回的诊断数据所在的缓存。

变元buffer_length可以包含下面这些值中的一个：

- 如果diagnostic_identifier_pointer指向一个字符串，值为diagnostic_identifier_pointer（或SQL_NTS）的长度。
- 如果diagnostic_identifier_pointer指向一个二进制缓存，值为SQL_LEN_BINARY_ATTR(length)的结果。
- 如果diagnostic_identifier_pointer指向一个固定长度的特殊数据类型，则值为SQL_IS_INTEGER、SQL_IS_UNINTEGER、SQL_SMALLINT或SQLUSMALLINT中的一个。
- 如果diagnostic_identifier_pointer指向另一个指针，则值为SQL_IS_POINTER。

变元diagnostic_identifier_length_pointer指向被返回的字符个数（不包括空字节）所在的缓存。如果字符个数比buffer_length大，则diagnostic_identifier_pointer的值被删减。

SQLGetDiagField不对自己进行同样的诊断。它返回如下这些值：SQL_SUCCESS、SQL_SUCCESS_WITH_INFO（操作成功，但数据被删减）、SQL_INVALID_HANDLE、SQL_ERROR（如果变元无效，或类似情况）或SQL_NO_DATA。

SQLGetDiagRec

```
SQLGetDiagRec(handle_type, handle, recorn_number, sql_state,
 native_error_pointer, message_text, message_text_length,
 text_length_pointer);
```

返回附加的诊断信息。在对前一个函数的调用已经返回SQL_SUCCESS或SQL_SUCCESS_WITH_INFO时，通常调用此函数。

handle_type可以是SQL_HANDLE_ENV（环境）、SQL_HANDLE_DBC（连接）、SQL_HANDLE_STMT（语句）或SQL_HANDLE_DESC（说明）。

handle_argument包含指定的handle_type类型的句柄。

变元record_number是从这里返回信息的记录（从1开始）。

变元sql_state指向被返回的含5个字符的SQLSTATE代码所在的缓存。

变元native_error_pointer指向被返回的本地错误代码所在的缓存。

变元message_text指向被返回的诊断信息（错误或警告）所在的缓存。

变元message_text_length包含message_text缓存的长度。

变元text_length_pointer指向被返回的字符的个数所在的缓存。如果字符个数比message_text_length大，则message_text的值被删减。

SQLGetEnvAttr

```
SQLGetEnvAttr(environment_handle, attribute, value_pointer,
 buffer_length, string_length_pointer);
```

返回环境属性的值。

变元attribute可以是表H.4中所列出的值之一。

表H.4 变元SQLGetEnvAttr attribute

属性	VALUE_POINTER内容
SQL_ATTR_CONNECTION_POOLING	用于启用或禁用连接池的32位值
SQL_ATTR_CP_MATCH	用来确定如何从可获得的连接池选择一个连接的32位值
SQL_ATTR_ODBC_VERSION	一个32位整数，用来判断是否为ODBC 2.x或ODBC 3.x的操作。这个值可以是SQL_OV_ODBC3或SQL_OV_ODBC2

变元value_pointer指向被返回的属性值所在的缓存。

如果变元指向一个字符串，则buffer_length是value_pointer的长度，否则不使用这个变量。

string_length_pointer指向被返回的字符的个数所在的缓存。如果字符个数比buffer_length大，则value_pointer的值被删减。

SQLGetFunctions

```
SQLGetFunctions(connection_handle, function_id, supported_pointer);
```

返回驱动器所支持的函数。

变元function_id可以是单个函数的ID，还可以是SQL_API_ODBC3_ALL_FUNCTIONS或SQL_API_ALL_FUNCTIONS。前者由ODBC3使用，后者由ODBC2使用。

变元supported_pointer指向一个值，包括SQL_FALSE或SQL_TRUE（如果function_id是单一的函数，表明函数是否被支持）或这些值的数组（从0开始）。

SQLGetInfo

```
SQLGetInfo(connection_handle,info_type, info_value_pointer,
 buffer_length, string_length_pointer);
```

返回有关驱动器和服务器的信息。

变元info_type包括信息的类型（如SQL_DRIVER_HDESC或SQL_DRIVER_HSTMT）。

根据变元info_type，变元info_value_pointer指向返回的信息所在的缓存。

变元buffer_length包括info_value_pointer缓存的长度。如果info_value_pointer不指向字符串，则buffer_length被忽略。

string_length_pointer指向被返回的字节（不包括空字节）总数所在的缓存。如果总数比buffer_length大，则info_value_pointer的值被删减。

SQLGetStmtAttr

```
SQLGetStmtAttr(statement_handle, attribute, value_pointer,
 buffer_length, string_length_pointer);
```

返回语句属性的值。

变元attribute可以是表H.5中所列的选项之一。

表H.5 变元SQLGetStmt attribute

属性	VALUE_POINTER内容
SQL_ATTR_CURSOR_SCROLLABLE	值可以是SQL_NONSCROLLABLE（语句句柄不需要可上下滚动的临时表）。如果用这个句柄调用SQLFetchScroll，则fetch_type可以包含SQL_FETCH_NEXT（默认值）或SQL_SCROLLABLE（语句句柄需要可上下滚动的临时表）。如果用这个句柄调用SQLFetchScroll，则fetch_type可以包含任何一个有效的句柄。这个属性会影响对SQLExecDirect和SQLExecute的调用

（续表）

属性	VALUE_POINTER内容
SQL_ATTR_CURSOR_SENSITIVITY	值可以是SQL_UNSPECIFIED（默认值，但临时表的类型未指定）、SQL_INSENSITIVE（临时表显示出没有被其他临时表改变的结果集）或SQL_SENSITIVE（临时表显示出所有被其他临时表改变的结果集）。这个属性影响对SQLExecute和SQLExecDirect的调用
SQL_ATTR_CURSOR_TYPE	指定临时表的类型，并且包括SQL_CURSOR_FORWARD_ONLY、SQL_CURSOR_STATIC、SQL_CURSOR_KEYSET_DRIVEN（驱动器使用在SQL_ATTR_KEYSET_SIZE语句属性中指定的对行数的键）或SQL_CURSOR_DYNAMIC
SQL_ATTR_KEYSET_SIZE	SQL_CURSOR_KEYSET_DRIVEN类型临时表的键集中的行数
SQL_ATTR_MAX_LENGTH	驱动器从字符列或二进制列中返回的最大数据量。如果value_pointer比数据量少，SQLFetch和SQLGetData数据将被删减，并返回SQL_SUCCESS_WITH_INFO
SQL_ATTR_MAX_ROWS	驱动器能够返回的最大行数（行数为0或没有限制）
SQL_ATTR_NOSCAN	值可以是SQL_NOSCAN_OFF（默认情况下，驱动器扫描SQL语句中的转义字符），或是SQL_NOSCAN_ON（驱动器不扫描SQL语句中的转义字符）
SQL_ATTR_PARAM_BIND_TYPE	值可以是SQL_PARAM_BIND_BY_COLUMN（默认情况下，表示与列有关的捆绑），或结构的长度，或与行有关的捆绑的缓存
SQL_ATTR_PARAM_OPERATION_PTR	指向包含SQL_PARAM_PROCEED或SQL_PARAM_IGNORE的数组，它确定在执行过程中参数是否要被忽略。还可以设置为空指针，在这种情况下，不返回参数的状态值
SQL_ATTR_PARAM_STATUS_PTR	在对SQLExecute或SQLExecDirect的调用之后，指向一个带有状态信息的值的数组（每一行一个值），包括下列这些调用中的一种：SQL_PARAM_SUCCESS、SQL_PARAM_SUCCESS_WITH_INFO（成功返回警告信息）、SQL_PARAM_ERROR、SQL_PARAM_UNUSED（通常是因为设置了SQL_PARAM_IGNORE）或SQL_PARAM_DIAG_UNAVAILABLE。还可以设置为空指针，在这种情况下，不返回数据

（续表）

属性	VALUE_POINTER内容
SQL_ATTR_PARAMS_PROCESSED_PTR	指向被处理的参数集个数所在的缓存，包括返回的错误。可以是一个空指针，在这种情况下，不返回数字
SQL_ATTR_PARAMSET_SIZE	每一个参数的值的个数
SQL_ATTR_QUERY_TIMEOUT	在超时之前，等待要执行SQL语句的秒数。如果value_pointer是0，则没有超时
SQL_ATTR_ROW_ARRAY_SIZE	调用SQLFetch或SQLFetchScroll后返回的行数。默认值是1
SQL_ATTR_ROW_BIND_OFFSET_PTR	指向增加给指针的偏移量，用来改变列数据的捆绑
SQL_ATTR_ROW_BIND_TYPE	设置混合定位。可以是SQL_BIND_BY_COLUMN（指出列方式混合），或者结构的长度或结果将要在其中进行结合的缓冲区（行方式混合）
SQL_ATTR_ROW_NUMBER	当前行的行号，如果无法确定，则为0
SQL_ATTR_ROW_OPERATION_PTR	指向一组单元（每行一个），这些单元包含SQL_ROW_PROCEED或SQL_ROW_IGNORE，决定这一行是否包括在大块操作中（这不包括SQLBulkOperations）。这还可以设置成空（null）指针，这时不会返回数组
SQL_ATTR_ROW_STATUS_PTR	指向一组值（每行一个），这些值包含在对SQLFetch或SQLFetchScroll的调用后行的状态值。还可以被设置成空（null）指针，并且驱动程序并不返回数组
SQL_ATTR_ROWS_FETCHED_PTR	指向缓冲区，其中SQLFetch、SQLFetchScroll、SQLSetPos或SQLBulkOperations调用所返回或影响的行数量将被返回，包括错误
SQL_ATTR_SIMULATE_CURSOR	可以是SQL_SC_NON_UNIQUE（没有确定模拟定位的UPDATE或DELETE语句只影响一行）、SQL_SC_TRY_UNIQUE（驱动程序试图确保模拟定位的UPDATE或DELETE语句只影响一行）或SQL_SC_UNIQUE（模拟定位的UPDATE或DELETE语句一定只影响一行）

变元value_pointer指向包含attribute返回的值的缓冲区。

变元buffer_length可以包含下面这些值：

- 如果value_pointer指向一个字符串，则为value_pointer（或SQL_NTS）的长度。
- 如果value_pointer指向一个二进制缓冲区，则为SQL_LEN_BINARY_ATTR(length)的结果。
- 如果value_pointer指向固定长度的指定数据类型，则为SQL_IS_INTEGER、SQL_IS_UNINTEGER、SQL_SMALLINT或SQLUSMALLINT。

• 如果value_pointer指向另一个指针，则为SQL_IS_POINTER。

string_length_pointer指向的缓冲区包含返回的全部字节数（不包含空null字节）。如果长度比buffer_length大，则value_pointer的值被截短。

SQLGetStmtOption

返回语句参数值。不推荐使用这个参数，可以使用SQLGetStmtAttr。

SQLGetTypeInfo

```
SQLGetTypeInfo(statement_handle,data_type);
```

返回SQL结果集，带有指定数据类型的信息。

将data_type设置为SQL_ALL_TYPES，将得出由服务器返回的数据信息。

SQLNativeSql

```
SQLNativeSql(connection_handle, sql_string, sql_string_length,
 modified_sql_string, modified_sql_string_length,
 string_length_pointer);
```

返回由驱动程序修改的（未执行的）SQL字符串。

变元modified_sql_string指向的缓冲区包含返回的被修改的SQL语句。

变元modified_sql_string_length指向的缓冲区包含要返回的字节数（不包括空字节）

string_length_pointer指向的缓冲区包含返回的全部字节数（不包括空null字节）。如果长度比modified_sql_string_length大，那么modified_sql_string的取值被截短。

SQLNumParams

```
SQLNumParams(statement_handle, parameter_count_pointer);
```

返回语句中参数的数目。

变元parameter_count_pointer指向的缓冲区包含返回的参数的数目。

SQLNumResultCols

```
SQLNumResultCols(statement_handle, column_count_pointer);
```

返回结果集中的列的数目。

变元column_count_pointer argument指向返回的列的数目所在的缓冲区。

SQLParamData

与SQLPutData一起使用提供执行时使用的参数数据（对于长数据值很有用）。

SQLPrepare

```
SQLPrepare(statement_handle, sql_string, sql_string_length);
```

准备下一步要执行的SQL语句。

SQLPrimaryKeys

```
SQLPrimaryKeys(statement_handle, catalog_name, catalog_name_length,
 schema_name, schema_name_length, table_name, table_name_length);
```

返回指定表的主键列。

SQLPutData

```
SQLPutData(statement_handle, data_pointer, data_pointer_length);
```

用于执行时发送列或参数数据。

变元data_pointer指向包含参数或列数据的缓冲区（类型是由SQLBindParameter的变元value_type或SQLBindCol的变元target_type指定的）。

变元data_pointer_length指定发往SQLPutData（SQL_NTS、SQL_NULL_DATA或SQL_DEFAULT_PARAM）数据的长度。

SQLRowCount

```
SQLRowCount(statement_handle, row_count_pointer);
```

返回受到某个SQL语句影响的行数量，这个语句对数据进行了修改（INSERT或DELETE）。

变元row_count_pointer指向一个缓冲区，其中包括行的计数，如果不可用则为－1。

SQLSetConnectAttr

```
SQLSetConnectAttr(connection_handle, attribute, value_pointer, string_length);
```

设置一个连接属性。

可能的属性列表及其描述可以参见SQLGetConnectAttr。

变元value_pointer指向属性值，其类型依赖于attribute。

变元string_length可能包含下列值：

- 如果value_pointer指向一个字符串，则为value_pointer（或SQL_NTS）的长度。
- 如果value_pointer指向一个字符串，则为SQL_LEN_BINARY_ATTR(length)的结果。
- 如果value_pointer指向一个字符串，则为SQL_IS_INTEGER、SQL_IS_UNINTEGER、SQL_SMALLINT或SQLUSMALLINT。
- 如果value_pointer指向另一个指针，则为SQL_IS_POINTER。

SQLSetConnectOption

设置一个连接选项。不推荐使用这个函数，可以使用SQLSetConnectAttr。

SQLSetCursorName

```
SQLSetCursorName(statement_handle,cursor_name,cursor_name_length);
```

指定临时表的名字。

SQLSetEnvAttr

```
SQLSetEnvAttr(environment_handle, attribute, value_pointer, string_length_pointer);
```

设置一个环境属性。可能的属性列表可以参见SQLGetEnvAttr。

SQLSetPos

```
SQLSetPos(statement_handle, row_number, operation, lock_type);
```

将指针移动到已获取的数据块中的某个位置，并且还可以刷新行集的数据，或更新并删除其中的数据。

变元row_number选择结果集中受到操作影响的行（从1开始）。如果设为0，则对所有行进行操作。

变元operation指定要执行的操作，并且可以是SQL_POSITION、SQL_REFRESH、SQL_UPDATE或SQL_DELETE。表H.6对这些变元进行了描述。

表H.6 变元operation

操作	描述
SQL_POSITION	驱动程序定位第row_number行的指针
SQL_REFRESH	驱动程序定位第row_number行的指针，并且刷新这一行的数据。行的数据不会再被存取，这与对带有fetch_type的SQLFetchScroll的调用是不同的
SQL_UPDATE	驱动程序定位第row_number行的指针，并且利用行集缓冲区中的值对相关数据进行更新，这个缓冲区来自于SQLBindCol中的变元TargetValuePtr
SQL_DELETE	驱动程序定位第row_number行的指针，并且删除相关的数据

在操作开始执行后，变元lock_type指定行的锁定操作，并且可以是SQL_LOCK_NO_CHANGE、SQL_LOCK_EXCLUSIVE或SQL_LOCK_UNLOCK。

SQLSetScrollOptions

设置影响临时表操作的参数。不推荐使用这个函数，可以使用SQLSetStmtAttr。

SQLSetStmtAttr

```
SQLGetStmtAttr(statement_handle, attribute, value_pointer, string_length);
```

设置语句属性。

可能的attribute值的列表可以参见SQLGetStmtAttr。

变元value_pointer指向属性值，其类型依赖于attribute。

变元string_length可能包含下列取值：

- 如果value_pointer指向一个字符串，则为value_pointer（或SQL_NTS）的长度。
- 如果value_pointer指向一个二进制缓冲区，则为SQL_LEN_BINARY_ATTR(length)的结果。

- 如果value_pointer指向一个固定长度的特殊数据类型，则为SQL_IS_INTEGER、SQL_IS_UNINTEGER、SQL_SMALLINT或SQLUSMALLINT其中之一。
- 如果value_pointer指向另一个指针，则为SQL_IS_POINTER。

SQLSetStmtOption

设置语句参数。不推荐使用这个函数，可以使用SQLSetStmtAttr。

SQLSpecialColumns

```
SQLSpecialColumns(statement_handle, identifier_type, catalog_name,
 catalog_length, schema_name, schema_length, table_name,
 table_name_length, scope, nullable);
```

当记录中的一个值被一个事务改变时，返回列的信息，并且这个信息惟一标识指定表或自动更新的列中的行。

变元identifier_type包含返回列的类型。它必须是SQL_BEST_ROWID或SQL_ROWVER。SQL_BEST_ROWID返回惟一标识一个记录的列的最小集（返回的这些列可以是专门设计用来启动这个函数的）。当记录中的值被一个事务修改时，SQL_ROWVER返回那些被自动修改（或更新）的列。

变元scope是行ID的最小范围。它可以是下列函数之一：SQL_SCOPE_CURROW（行ID只有在该行上才有效）、SQL_SCOPE_TRANSACTION（行ID在当前事务期间有效）或SQL_SCOPE_SESSION（行ID对整个任务都有效）。

变元nullable指出是否包括那些可以包含空值（null）的列。它可以是SQL_NO_NULLS（不包括可以包含空值的列）或SQL_NULLABLE（包括那些可以包含空值的列）。

SQLStatistics

```
SQLStatistics(statement_handle, catalog_name, catalog_name_length,
 schema_name, schema_name_length, table_name, table_name_length,
 index_type, reserved);
```

返回表和相关索引的统计。

变元index_type可以是SQL_INDEX_UNIQUE或SQL_INDEX_ALL。

变元reserved可以是SQL_ENSURE（返回CARDINALITY和PAGES列）或SQL_QUICK（只有在它们可用时，才返回CARDINALITY和PAGES列）。

SQLTablePrivileges

```
SQLTablePrivileges(statement_handle, catalog_name, catalog_length,
Schema_name, schema_name_length, table_name, table_name_length);
```

返回表和相关特权的列表。

SQLTables

```
SQLTables(statement_handle, catalog_name, catalog_length,
Schema_name, schema_name_length, table_name, table_name_length,
 table_type, table_type_length);
```

返回表、目录或模式名和表的类型。

SQLTransact

结束一个事务。不推荐使用这个函数，可以使用SQLEndTran。

欢迎与我们联系

为了方便与我们联系，我们已开通了网站（www.medias.com.cn）。您可以在本网站上了解我们的新书介绍，并可通过读者留言簿直接与我们沟通，欢迎您向我们提出您的想法和建议。也可以通过电话与我们联系，电话号码（010）68252397。

[General Information]
书名=MySQL 4从入门到精通
作者=
页数=638
SS号=11138445
出版日期=
www.chinaz.com