完全手册丛书

MySQL数据库开发组审定

MySQL: The Complete Reference

MySQL 完全手册

[美] Vikram Vaswani 著

徐小青 路晓村 等译

薛荣华 审校

安装和配置MySQL数据库

管理、优化、备份和恢复MySQL数据库

用各种语言开发MySQL数据库驱动应用

McGraw Hill

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

(MySQL 数据库最权威的参考书)

MySQL: The Complete Reference

MySQL 完全手册

作为当前最流行的开放源代码数据库，MySQL 几乎席卷了整个产业界。MySQL 所具有的跨平台支持、独立存储引擎、高级权限与安全系统、查询高速缓存、全文索引与检索等特性，是 MySQL 得以取胜的法宝。本书由 MySQL 数据库开发组审定，其权威性不容置疑。本书涵盖了 MySQL 安装、使用、管理和开发各个方面的知识，是一本完整的参考手册。

本书主要内容

- 在 Linux/UNIX 或 Windows 平台上安装、配置和测试 MySQL 数据库
- 使用 MySQL 数据类型、语句、运算符和函数
- 用数据定义语言（DDL）管理数据库和表
- 用数据操纵语言（DML）添加、删除和查询记录
- 事务支持
- MySQL 的五层访问控制和权限系统
- 数据库优化、备份、恢复和复制的方法
- 用 C、C++、Perl、Java、PHP 以及其他语言创建应用程序

Vikram Vaswani 是 Melonfire 公司（www.melonfire.com）的创始人和 CEO。该公司是一家专门为各种客户从事 Web 设计、开发和内容创建的公司。Vaswani 还写过关于开放源代码技术（包括 MySQL、Perl、Python 和 PHP）的其他专著。

ISBN 7-121-00451-8

9 787121 004513 >

华信教育资源网

责任编辑：谭海平
许菊芳
封面设计：毛惠庚

本书贴有激光防伪标志，凡没有防伪标志者，属盗版图书

ISBN 7-121-00451-8　定价：45.00 元

完全手册丛书

MySQL 完全手册

MySQL: The Complete Reference

[美] Vikram Vaswani 著

徐小青 路晓村 等译

薛荣华 审校

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内容简介

MySQL是目前最流行的开放源代码数据库管理系统，全世界的装机量已超过400万台。本书详细介绍了如何使用可定制的MySQL数据库管理系统支持健壮的、可靠的、任务关键的应用程序。全书共分四部分21章，分别从MySQL的安装与配置，创建与使用数据库和表，数据的查询与运算，事务处理，MySQL的管理与配置，备份与恢复，性能优化，MySQL与C，Perl，PHP的开发接口等不同角度阐述了MySQL的特点。本书在内容上注重全面性，几乎涉及了与MySQL相关的所有主题，是一本完整的MySQL参考手册。

本书适合于准备用MySQL进行系统开发的设计人员和编程人员学习和参考，也适合用做软件专业本专科水平的教科书。

Vikram Vaswani: MySQL: The Complete Reference.
ISBN: 0-07-222477-0
Copyright © 2004 by The McGraw-Hill Companies, Inc.
Original language published by The McGraw-Hill Companies, Inc. All rights reserved. No part of this publication may be reproduced or distributed in any means, or stored in a database or retrieval system, without the prior written permission of the publisher.
Simplified chinese translation edition jointly published by McGraw-Hill Education (Asia) Co. and Publishing House of Electronics Industry. Copyright © 2004.

本书中文简体字翻译版由电子工业出版社和美国麦格劳-希尔教育出版（亚洲）公司合作出版。未经出版者预先书面许可，不得以任何方式复制或抄袭本书的任何部分。
本书封面贴有McGraw-Hill公司激光防伪标签，无标签者不得销售。

版权贸易合同登记号　图字：01-2002-4231

图书在版编目（CIP）数据

MySQL完全手册/（美）范斯瓦尼（Vaswani, V.）著；徐小青，路晓村等译.
-北京：电子工业出版社，2004.11
（完全手册丛书）
书名原文：MySQL: The Complete Reference
ISBN 7-121-00451-8

Ⅰ.M… Ⅱ.①范… ②徐… ③路… Ⅲ.关系数据库-数据库管理系统，MySQL-手册 Ⅳ.TP311.138-62

中国版本图书馆CIP数据核字（2004）第104806号

责任编辑：谭海平　许菊芳
印　　刷：北京兴华印刷厂
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　邮编：100036
经　　销：各地新华书店
开　　本：787 × 1092　1/16　　印张：26.5　字数：678千字
印　　次：2004年11月第1次印刷
定　　价：45.00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。质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

译 者 序

MySQL是目前最流行的开放源代码数据库管理系统，全世界的装机量已超过400万台。本书详细介绍了如何使用可定制的关系数据库管理系统支持健壮的、可靠的、任务关键的应用程序。在编排上从最基本的SQL概念讲起，介绍如何在Linux/UNIX或Windows平台上安装、配置和测试MySQL数据库，如何使用MySQL的数据类型、语句、运算符和函数，利用数据定义语言（DDL）管理数据库和表，利用数据操纵语言（DML）添加、删除和查询记录，利用事务支持，实现MySQL的五层访问控制和权限系统，如何处理数据库优化、备份、恢复和复制以及用C，C++，Perl，Java，PHP和其他语言创建应用程序。本书是为MySQL用户、开发者和数据库管理员准备的一本综合性参考书。它包含了关于MySQL（包括MySQL 4.x）的详尽信息，全面涵盖了与MySQL配置相关的从软件安装和配置到数据备份和复制的所有主题。

本书是以一个共同的目的面向大量读者的：一种在自己的系统上学习如何最好地配置和使用MySQL的期望。不管你是一个寻找免费的数据库系统并在上面练习SQL的学生，一个对创建基于MySQL的应用程序感兴趣的开发者，一个关心把数据移植到MySQL以降低成本的数据库管理员，本书都包含了研究和运行强大的开放源代码应用程序所需要的理论和实践示例。相信本书会成为读者案头一本实用的参考手册。

本书由徐小青（第一部分和第二部分）、路晓村（第三部分和第四部分）翻译，由薛荣华审校。参加本书译录校工作并给予大力协助的还有王景中、薛菲、闫慧娟、曹汉征、姚栋、许秀英、王泰东、李可、郭淼、矫克民、刘东顺、王建成、薛亮、沈兰英等同志。电子工业出版社的编辑们为此书的出版做了大量艰苦细致的工作，译者谨向他们表示衷心的感谢。译者水平有限，不妥之处欢迎读者批评指正。谢谢！

序

作者Vikram Vaswani是MySQL和其他许多开放源代码技术方面的专家。在写这本书的时候，Vaswani积极地与MySQL数据库开发组合作，以保证这本书在技术上的正确性，保证所涉及的主题中没有遗漏重要的知识点。

本书采取了一个全面的方法：涵盖了MySQL的主要特性以及诸如复制和各种API接口编程之类的高级主题。它还为理解关系模型提供了坚实的基础，全面涵盖了ACID事务和对关系数据库不够了解的人所要掌握的基本概念。

本书还涉及了熟悉Oracle或其他数据库系统的用户和管理员所需要的基本知识。体验过MySQL的人，对熟悉的主题会找到新的视角，从而更深入洞察MySQL的世界，提高他们现有的知识和经验。

在读者使用MySQL时，希望本书是一个颇有价值的资源。

Michael "Monty" Widenius
MySQL AB公司首席技术官
芬兰·赫尔辛基

关于作者

Vikram Vaswani 是 Melonfire 公司（http://www.melonfire.com/）的创始人和 CEO，该公司专门从事软件咨询、内容创新和内容企业联合组织化服务。Vaswani 是开放源代码运动的热情支持者，他通过 http://www.melonfire.com/community/columns/trog/ 上的每周专栏，经常把开放源代码技术（包括Perl，Python，PHP，MySQL以及Linux）方面的文章和指南贡献给整个MySQL社区。他最近的著作是“XML and PHP”（http://www.xmlphp.com/）。

Vaswani 在 IT 界有 8 年多的工作经验，其中有 6 年时间是作为用户、管理员和应用程序开发者与 MySQL 打交道。他在各种环境中部署过 MySQL，包括企业内部网、高流量 Internet 网站和要求严格的瘦客户机应用环境，这些都是创作本书时非常有用的经验！作为软件顾问，他仍然是一个有份量的 MySQL 倡导者。

当不在 Melonfire 的一个高度戒备的会议室里策划控制世界时，Vaswani 通过阅读、睡觉、看老电影、玩游戏、摆弄那些不断增多的电动小玩意收藏品来消遣自己，顺便会留意那些不友好的代理商。在 http://www.mysql-tcr.com/ 上可以知道关于他的更多信息，在这里还可以下载示例代码，与其他在线的 MySQL 爱好者交流。

致　谢

在紧张的交稿期限和非同寻常的环境下，经过长长的一年多的时间，终于写成了本书。幸运的是，在整个过程中，我得到了许多人无法估量的帮助，这本书的字里行间都凝聚着他们的汗水。

首先要感谢我的家人。在写这本书的时候，是他们为我提供了安静的工作地点，并且容忍了我随意的工作时间（甚至古怪的行为）。他们的宽容是显著的，非常感谢！

万分感谢 Harish Kamath，是他把相当多的 SQL 经验和知识带给了技术编辑的过程，他对本书第11章做出了贡献，并对我的闲聊电话和电子邮件总是非常有耐心。在本书的结构成型和确保尽可能内容全面上，他的想法和观点是很重要的。还要感谢 Claude Seidman 和 Efren Estevez，他们分别对第 2 章、第 15 章、第 16 章和第 17 章做出了贡献。

在此必须提到McGraw-Hill/Osborne的优秀团队——高级策划编辑Christopher Johnson，采购协调员 Athena Honore，项目编辑 Jenn Tust，Elizabeth Seymour 和 Monika Faltiss（在出书过程中，他自始至终指导了本书）。还要感谢技术编辑 Sasha Pachev，Jay Powers，Brian Kaney 和 Arjen Lentz，他们对技术上的编辑做出了贡献。还要特别致谢MySQL AB公司和它的富有献身精神的志愿者和开发者团队，是他们创建并不断增强了一个真正优秀的软件。

最后，在创作本书的整个过程中，我都比平常享受着更多的快乐，这应当归功于：Lawrence Block，Bryan Adams，the Stones，MAD杂志，Scott Adams，Gary Larson，MTV，Kylie Minogue，Buffy，Farah Malegam，MySQL邮件列表，Stephen King，John le Carre，Subway，Barry White，the Kamaths，Apple，Robert Crais，Robert B.Parker，Baz Luhrmann，Anna Kournikova，Swatch，Ling's Pavilion，Tonka，HBO，Ferrari，Mark Twain，Tim Burton，Friends 的全部列表，John Sandford，Tube，Dido，Google.com，The Matrix杂志，Alfred Hitchcock，Bruno D'Costa，Woody Allen，Palm，Michael Schumacher，Mambo's 和 Tito's，Kalindi Mehta，Humphrey Bogart，Library 酒吧，Amazon.com，U2，The Three Stooges，Oscar Wilde，Punch，Harry Potter，Scott Turow，Slackware Linux，Calvin 和 Hobbes，Blizzard Entertainment，Popeye 和 Olive，Dennis Lehane，Trattoria，Xerxes Antia，Dire Straits，Bruce Springsteen，Santana，David Mitchell，以及所有在国内和其他任何地方的我的朋友们。

前　言

自1996年开始，从一个简单的SQL工具到当前的“世界上最受欢迎的开放源代码数据库”地位，MySQL已经走过了一段很长的路。根据MySQL AB（MySQL的开发者）发布的信息，到2003年，MySQL的装机量在全世界已超过400万台。MySQL为Internet网站、搜索引擎、数据仓库、任务关键的软件应用和系统提供动力，它得到了像Sony，Xerox，HP和NASA这样的公司或组织的积极使用。简而言之，它无处不在，它的应用将会变得更广泛。

我们不难发现这样疾速增长的原因。相对于像Oracle和Microsoft SQL Server一样的更商业化、非开放源代码的系统来说，快速、健壮和友好的数据库引擎、高级的数据管理和恢复工具、不断改进的特性集合、遵守现有的SQL标准、友好的商业许可原则，都是促成MySQL成为可实施的选择因素。MySQL的较低总体拥有成本和更稳定、更安全的系统特性，使越来越多的企业把它们的系统移植到MySQL，并且收获着MySQL开放源代码带来的效益。

这就是本书的作用所在。如果你是一位承担着把RDBMS系统移植到MySQL的任务的业务专家，一个对创建基于MySQL的应用程序感兴趣的开发者，或者仅仅是一个对MySQL和它能为我们做什么感到好奇的爱好者，本书就是将你引入MySQL世界的惟一地图。

综述

本书是为MySQL用户、开发者和数据库管理员设计的一本综合性的参考书，它包含了关于MySQL RDBMS（包括MySQL 4.x）的详尽信息，全面覆盖了与MySQL配置相关的所有主题，包括软件安装和配置、数据备份和复制等。

本书还详细探讨了结构化查询语言（SQL）的MySQL特殊用法，深入研究了创建和管理数据库和表的命令，插入、更新和删除记录，创建满足特定条件的报表，使用连接和子查询，通过关键字执行数据完整性和优化查询，以求更快的速度和效率。

本书也没有忘记MySQL开发者，在书中包含特殊的几章，它专门适合基于MySQL的应用程序开发，为致力于MySQL体系结构，并且对它感兴趣的开发者提供了现实的指南和技巧。

读者

本书是以一个共同的目的面向大量读者的：一种在自己的系统上学习如何最好地配置和使用MySQL的期望。不管你是一个寻找免费的RDBMS并在上面练习SQL的学生，一个对创建基于MySQL的应用程序感兴趣的开发者，一个关心把数据移植到MySQL以降低成本的数据库管理员，本书都包含了研究和运行这个强有力的开放源代码应用程序所需要的理论和实践示例。

与其他很多书籍不同，本书没有假定读者预先具有SQL知识和数据库基础。相反，它通过举例、使用教程和现实世界的实例来解释基本概念，从而增进读者对MySQL RDBMS的熟悉

程度。在后面的章节中，将鼓励读者在自己的MySQL环境里试验各种实例。这样做不会破坏任何东西，反而会使读者从亲身的经历中收获大量的知识。

本书以教程和参考向导两种形式进行组织，所以可以用读者喜欢的任意次序阅读它。这里有一些建议：

- 如果你是MySQL初学者，按照次序阅读各章会比较容易，可以按结构化的方式学习基本知识。这是推荐的方法，MySQL的新用户应该采用它。
- 如果你已经使用过MySQL，可能更喜欢把本书作为一本桌面参考书，可随意把它翻到需要的位置来阅读特别的主题。
- 如果你是一位专家（比如数据库管理员或Web开发者），本书包含单独的关于MySQL管理和开发的部分，专门设计用来解决最经常遇到的问题，并且提供了所有与读者工作性质相关的问题和主题的信息。

组织

本书分为四个部分，下面列出了每部分包含的内容。

第一部分提供MySQL的简介，并且指导在UNIX和Windows上安装MySQL。

第1章探讨了MySQL的历史和发展，它的特性，并且解释了为什么它能提供如此引人注目的价值。

第2章粗略讲解了MySQL的优点，解释了各种MySQL子系统以及它们是如何相互配合的。

第3章研究如何在UNIX和Windows上获得、安装、配置和测试MySQL服务器版本。还从有助于读者选择恰当的服务器版本的角度出发，解释了可以使用的不同MySQL服务器版本之间的差异。

第二部分讲述MySQL服务器的实际应用，解释了怎样使用SQL命令集来创建数据库和表、添加和删除记录、执行查询和使用诸如事务和子查询之类的高级MySQL 4.x特性。

第4章讲述SQL的历史和发展，此外还包括一个简单的教程，使新用户熟悉用于创建和将信息输入数据库的基本SQL命令。

第5章研究MySQL所支持的各种数据类型，并且探讨了如何用它们来增强表的一致性和完整性。

第6章介绍MySQL的算术运算、逻辑运算、比较运算和位运算，并且演示了它们怎样用于执行计算、比较和转换。

第7章讲述MySQL的大量内置函数，向读者展示了如何用最少的时间和精力来允许MySQL开发者执行从数据处理到口令加密等一系列操作。

第8章深入讨论MySQL用来存储数据的数据库和表的结构，并且解释了用来创建、修改和删除数据库、表和索引的SQL命令。本章的核心是讨论MySQL强大的CREATE TABLE命令，它提供了表创建过程的适度控制，探讨了对MySQL相对比较新的外键和全文索引。

第9章继续前一章，解释了怎样把记录插入到现有的MySQL数据库中。本章的第一部分论述了INSERT，UPDATE和DELETE命令，举例说明了怎样使用它们来处理数据库记录，并且阐明了MySQL特有的关于标准命令句法的某些问题；第二部分深入研究了SELECT语句，

展示了如何使用它来创建数据库中记录的过滤子集；排序、分组和计算记录；使用会话变量；用多种不同的格式输入和输出数据等功能。

第10章讲述多表查询，展示了表之间的关系如何以不同的方式连接记录，以生成更有用的报表。本章论述了各种类型的连接，并且演示了它们在许多不同情况下的用法。

第11章讲述如何在查询里嵌套查询，以求更高级的查询运算，这是MySQL 4.1新引入的特性。

第12章研究MySQL 4.x的另一个非常强大的新特性：把一系列SQL语句分组成单个单元并且原子性地执行它们，或在出错的事件中撤销全部改动的能力。

第三部分论述在管理和维护一个MySQL RDBMS时要做的工作，包括像数据库备份和修复、安全性、访问控制、活动日志、服务器优化和故障排除这样的主题。

第13章分析了MySQL数据库管理员的作用，解释了像启动和关闭服务器、修改默认的服务器配置和检查MySQL日志文件这样的常规任务。

第14章讲述MySQL的安全和权限系统，探讨了用户账号和口令的管理（包括忘记了MySQL超级用户口令的解决办法）。

第15章提供了关于如何备份和恢复一个MySQL数据库，以及如何使用MySQL提供的功能从一个被破坏的数据库中恢复数据的信息和指导。

第16章提供了让MySQL服务器性能尽量达到最大的技巧和窍门，包括关于优化查询达到快速执行、修改高速缓冲存储器和缓冲器设置以达到迅速响应，以及使用特殊的运行时间参数以求更有效的资源利用等方面的信息。

第17章论述MySQL的另一个更有趣和强大的特性：自动复制跨越不同主机的数据库的能力。

第四部分主要针对开发者，它论述了MySQL API，一套建立自定义数据驱动的MySQL应用程序的功能全面而强大的工具。本部分包括实例和C，Perl，PHP API源代码。

第18章通过一系列API揭示了MySQL内部结构的情况和基本原理，对各种API的性能提供了简要的描述。其目的是帮助开发者选择适合他们的特殊要求的API。本章还对不同API之间相互比较的优点进行了论述。

第19章讨论如何使用MySQL发布版本包含的C库API编写MySQL客户应用程序。现实的例子包括一个交互式命令行SQL客户机和一个支持数据输入和简单报表生成功能的交互式开支追踪程序。

第20章讨论Perl DBI，特别介绍了MySQL驱动程序和可以通过Perl脚本连接MySQL数据库的DBI方法。在这里的现实应用软件是一个基于CGI的线索评论系统，通过一个标准网络浏览器可以访问它并且适合在内容丰富的网站上使用。

第21章主要围绕PHP进行论述，它是MySQL所包含的支持数据库Web应用程序的一种流行的脚本语言。此外还详细论述了相关的PHP函数。本章通过适用于任何网络浏览器的一个数据库驱动的书签应用程序，演示了MySQL与PHP组合的现实可用性。

本书使用约定

本书使用几个约定来突出特别的建议：

提示：使 MySQL 更好地为我们服务的简便方法

注释：了解 MySQL 和其他程序工作的方式。

警告：需特别注意的地方，从而可使读者少走弯路。

当提到相关的材料时，通常会给出包含希望阅读到的信息的章节编号。

在本书的代码清单中，用粗体字突出的句子是在提示符下输入的命令。例如在下面的清单中：

```
mysql> SELECT * FROM articles;
+-----+-----------------------------------+--------+
| aid | aname                             | fk_cid |
+-----+-----------------------------------+--------+
|  1  | Sample article on winter fashions |      2 |
+-----+-----------------------------------+--------+
1 row in set (0.00 sec)
```

粗体字行是用户要在 MySQL 命令提示符下输入的查询。可以使用这个例子作为一个向导来试验本书中的各种实例。

补充资料

这是补充资料的版式，补充资料被设置成有点远离正常文本，它包括与当前主题相关的信息，但是不一定必须符合信息的流向。

相关网站

学习 MySQL 的最好的方法是理论和实践相结合。要达到这个目标，可以在 MySQL 相关网站 http://www.mysql-tcr.com/ 上找到本书用于创建很多实例数据库的 SQL 代码，并且演示了本书第四部分中介绍的各种应用程序和脚本的源代码。当你访问这个网站时，可以顺便访问一下 MySQL 讨论论坛，联络其他 MySQL 用户和社区，以共享我们在 MySQL 使用和管理上的经验。

目　录

第一部分　安　装

第 1 章　数据库导论 2
1.1　概述 2
1.2　小型数据库 4
1.3　小结 8

第 2 章　MySQL 技术教程 9
2.1　MySQL 体系结构综述 9
2.2　MySQL 引擎 12
2.3　小结 19

第 3 章　MySQL 安装和配置 20
3.1　获得 MySQL 20
3.2　安装和配置 MySQL 22
3.3　测试 MySQL 35
3.4　安装之后的步骤 37
3.5　小结 38

第二部分　使　用

第 4 章　SQL 基础 40
4.1　SQL 简史 40
4.2　SQL 综述 40
4.3　(My)SQL 教程 41
4.4　小结 55

第 5 章　MySQL 数据类型 56
5.1　数据类型的要求和基本原理 56
5.2　MySQL 数据类型 57
5.3　小结 76

第 6 章　MySQL 运算符 77
6.1　算术运算符 77

6.2 比较运算符 81
6.3 逻辑运算符 93
6.4 位运算符 95
6.5 小结 97

第7章 MySQL函数 98
7.1 数学函数 98
7.2 聚合函数 103
7.3 字符串函数 106
7.4 日期和时间函数 112
7.5 加密函数 120
7.6 控制流函数 123
7.7 格式化函数 127
7.8 类型转化函数 129
7.9 系统信息函数 131
7.10 小结 135

第8章 使用数据库和表 136
8.1 创建数据库 136
8.2 选择需要的数据库 137
8.3 删除数据库 138
8.4 创建表 138
8.5 复制表 157
8.6 修改表 160
8.7 删除表 165
8.8 获得数据库、表、字段和索引的信息 165
8.9 小结 166

第9章 使用数据 167
9.1 插入、更新和删除记录 167
9.2 检索记录 177
9.3 复制、导入和导出记录 187
9.4 小结 194

第10章 连接 195
10.1 什么是连接 195
10.2 连接的类型 198
10.3 小结 210

第11章 子查询 211
11.1 什么是子查询 211

11.2 子查询的类型 215
11.3 小结 228

第 12 章 事务处理 229
12.1 什么是事务 229
12.2 事务的生命周期 233
12.3 控制事务行为 236
12.4 事务和性能 241
12.5 具有非事务表的伪事务 244
12.6 小结 249

第三部分 管 理

第 13 章 管理与配置 252
13.1 数据库管理与 MySQL 252
13.2 基本的服务器管理和配置任务 254
13.3 小结 264

第 14 章 安全、访问控制和权限 265
14.1 MySQL 授权表 265
14.2 授权、废除和浏览用户权限 273
14.3 改变用户口令 280
14.4 小结 282

第 15 章 维护、备份和恢复 283
15.1 维护 283
15.2 检查和修理表 286
15.3 备份和恢复 289
15.4 小结 292

第 16 章 性能优化 293
16.1 索引 293
16.2 查询高速缓存 295
16.3 查询分析 296
16.4 优化多表查询 298
16.5 使用临时表 300
16.6 优化表设计 301
16.7 调整服务器设置 302
16.8 基准技术 303
16.9 小结 306

第 17 章　MySQL 复制 307
17.1　复制基本原理 307
17.2　管理复制过程 310
17.3　小结 314

第四部分　开　　发

第 18 章　MySQL API 316
18.1　MySQL API 概述 316
18.2　MySQL API 组件 317
18.3　语言支持 317
18.4　选择 API 319
18.5　小结 321

第 19 章　MySQL 与 C 322
19.1　C 语言 322
19.2　MySQL 与 C 语言 323
19.3　真实范例 340
19.4　小结 348

第 20 章　MySQL 与 Perl 349
20.1　Perl 349
20.2　MySQL 与 Perl DBI 350
20.3　真实范例 366
20.4　小结 382

第 21 章　MySQL 与 PHP 383
21.1　历史和演变 383
21.2　MySQL 与 PHP 的组合 385
21.3　真实范例 401
21.4　小结 410

第一部分

安　　装

第 1 章　数据库导论
第 2 章　MySQL 技术教程
第 3 章　MySQL 安装和配置

第1章　数据库导论

不论我们喜欢还是讨厌数据库，在当今高速发展的世界中，没有数据库，我们将面临许多困难。从工艺生物学到电子商务的应用学科，近年来数据库已经占据了主流，它已不再是小型办公室里文件架后戴着眼镜的科研人员所研究的基础领域。在银行、地方图书馆和网络浏览器中，我们都能找到数据库，甚至我们邻近的杂货店也使用数据库。数据库无处不在，并且越来越聪明。

在过去的几年中，随着开放源代码运动的蓬勃发展，人们把越来越多的注意力集中到一个称为MySQL的少为人知的数据库系统上。这个数据库系统可以在Internet上免费使用，提供了很多最小性能损失方面的特性，并且做了大规模的可靠性和速度测试，能在各种应用和环境中成功使用。现今，拥有超过400万装机量的MySQL将迅速成为全世界数据库管理者和使用者的一个标准。

通过本章的介绍，我们将更进一步学习MySQL，了解它是什么、如何工作的以及为什么它的特性配置使得它与更出名、更具有资金援助的商业软件相比是一个具有竞争性的软件。

1.1　概述

在进入MySQL的讨论之前，让我们拿出少许时间来回答一个基本问题：什么是数据库？

从根本上讲，数据库是一个有组织的、根据某种标准分类的数据集合。传统的比喻是把文件装入抽屉（见图1.1），每个抽屉保存一些与特殊主题相关的文件，而且这些文件可能根据一套特殊的标准（字母顺序、颜色、数字代码等）进一步分类。抽屉中资料的组织结构和文件，使我们能够比较容易地迅速获得特殊的信息：只需把手放在某个拉手上，拉开适当的抽屉，选择与标准相匹配的单个或多个文件即可。

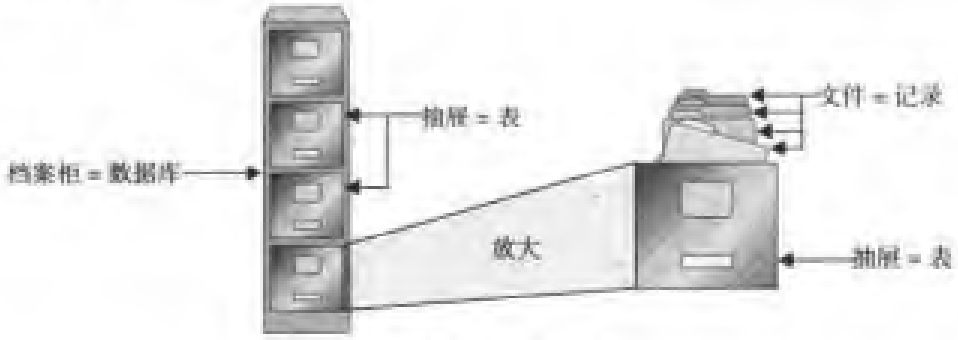

图1.1　数据库的形象化

电子形式的数据库有助于我们组织信息，并且为快速、有效地访问数据库中的特殊信息提供了必需的工具。在数据库中，称装有文件的抽屉为表，文件本身称为记录，取出信息的行为称为查询，结果数据称为结果集合。图1.2显示了我们怎样查询数据库（在这里是一个档案柜）来获取信息。在本书中将会遇到很多这样的术语，我们将会逐渐习惯它们，所以现在如果觉得它们有些陌生，不必在意。

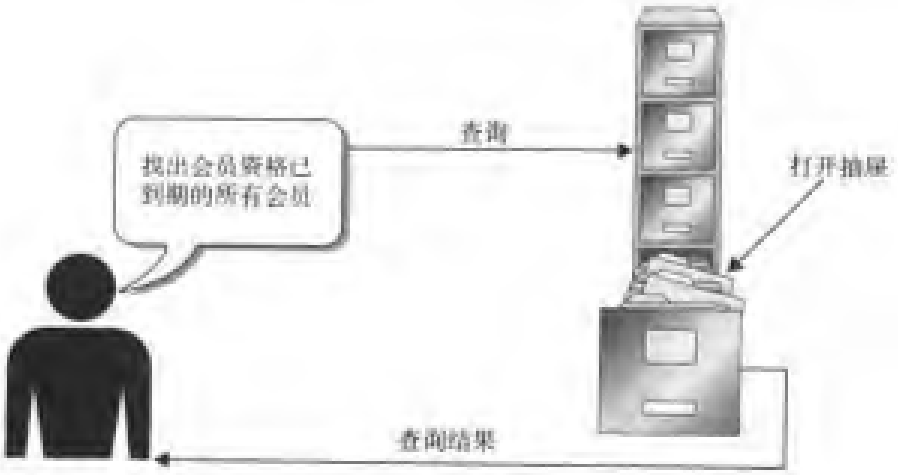

图 1.2　查询过程

虽然可以使用数据库来保存少量的信息，也可以保存数量稍多的信息，但是当我们需要处理非常大量的数据时，数据库的真正能力就体现出来了。例如，如果只有少量的数据需要处理时，我们用手工就可以很容易地操作和检索它们，然而当数据量增加时，执行一个手工检索就变得单调乏味而且代价昂贵了。考虑一个装有 20 个文件的档案柜和一个装有 2000 万个文件的档案柜，如果用人工检索的方法，在第一个柜中找出一个文件大约花费几分钟，而在第二个柜中找一个文件则几乎不可能（假定有效率限制）。

在这样的情形下，电子化的数据库管理系统（DBMS）可以充分简化我们的工作。这样的系统不仅占据的物理空间比传统的以纸张文件为基础的系统要少，而且还在组织数据、简化信息检索和修改方面为我们提供了辅助工具。内建式索引使迅速而有效地找到信息成为可能，而自动处理程序保证数据总以一致、无错的方式来存储和交叉引用。数据库还提供了可移植性和兼容性（一旦数据被组织和存储在数据库中，我们可以选择任意方式来提取和显示它们），并且为重要的信息提供了集中存储的方式。

关系数据库管理系统（RDBMS）通过在组成数据库的表中创建关系来更进一步地处理数据。这些关系可以使用不同的方式组合来自多个表的数据，允许用户从不同的角度查看同一数据，从而可以利用这个增强显示来做出更好的业务决定（使读取数据更有效率和成本效益更高）。通过在相关的信息条中生成连接，RDBMS 不但能够更有效地存储信息（通过除去冗余和重复），而且还可以在数据完全不同的部分产生尚未被发现的关系，并且允许有效地开发利用这些关系。

正如我们所能想像的，这种信息管理在商业和其他领域具有重要意义，这是 RDBMS 在当今高速发展的世界这么流行的诸多原因之一。大型软件公司，如 Microsoft 和 Oracle，意识到“知识就是力量”这句格言在这里和 Francis Bacon 先生最初发表它时一样正确，这些公司每年都花费上百万美元用于研究和开发商业数据库系统。

这些商业关系数据库管理系统是强大的、灵活的、富有特色的软件系统，专门用于处理数量巨大、事务繁重、任务重大的应用程序，它们每秒钟可以毫不费力地快速移动数百万条记录和执行数百条事务。

惟一的问题是，它们超出了大多数中小型交易的预算，这些中小型交易通常运行比较小而不太复杂的应用程序，不需要具有这么强大功能的数据库系统。

通常，用户需要的是一个快速的、可以信赖的、符合成本效益原则和效率要求的系统。这就是 MySQL 的作用所在。

作　业

对学习更多的关于关系数据库管理系统的知识感兴趣吗？考虑把下面这本书加到你的购物清单中吧："Introduction to Relational Databases and SQL Programming"，Catherine Creary和Christopher Allen 著（McGraw-Hill 出版，ISBN 0-07-222924-1，2003）。

1.2 小型数据库

MySQL 是一个高性能、多线程、多用户、建立在客户 - 服务器结构上的 RDBMS，专门为了速度和稳定性而设计。在过去的几年中，它已经成为线上和线下适合于数据库驱动的应用程序最受欢迎的 RDBMS 之一。现今，有超过 400 万的网站建立、使用并且配置了基于 MySQL 的应用程序，而且网站的数量每天都在增加（见 MySQL 官方网站 http://www.mysql.com/）。

MySQL RDBMS 由下面两部分组成：

- **服务器端工具**　包括 MySQL 数据库服务器以及其他管理多个 MySQL 数据库服务器、优化和修改 MySQL 表、创建故障记录的工具。MySQL 数据库服务器是核心系统，负责创建和管理数据库、执行查询和返回查询结果，并且对数据库的安全性负责。
- **客户端工具**　包括一个命令行方式的 MySQL 客户机、管理 MySQL 用户许可的工具以及导入导出 MySQL 数据库的实用程序。还包括查看和复制 MySQL 数据库和表、维护表和检索服务器状态信息的命令行工具。

在本书接下来的章节中，将详细论述这两个部分。

1.2.1 历史和发展

MySQL 始于 1979 年，最初是 Michael "Monty" Widenius 为瑞典的 TcX 公司创建的名为 UNIREG 的数据库系统。由于 UNIREG 没有结构化查询语言（SQL）接口，因此在 20 世纪 90 年代中期，它失去了 TcX 公司的宠爱。TcX 开始寻找其他的系统，mSQL 就是其中之一，它是由 David Hughes 创建的、具有竞争力的 DBMS。

可是，mSQL 没有为 TcX 尽力，于是 Widenius 决定创建一个新的、符合自己特殊需求的数据库系统。1996 年 5 月，这个系统完成并且发布给了一小部分人，成为我们今天所知的 MySQL 的最初版本。

几个月后，MySQL 3.11 作为 Solaris 的二进制版本首先公布于众。不久 MySQL 的 Linux 源代码和二进制代码版本公开发行。一个热情高涨的开发者社区和一个友好的基于通用公众许可（GPL）的特许政策对后来的版本开始关注（http://www.linuxjournal.com/ article.php?sid=3609 里有详细的信息）。今天，包括 Linux，MacOS 和 Windows 在内，MySQL 以源代码和二进制代码两种形式适用于多种平台。

几年之后，TcX 派生出一个私人公司 MySQL AB，现今它是 MySQL 服务器源代码和商标的惟一拥有者，并且负责 MySQL 数据库服务器的维护、销售和进一步开发。

Widenius仍然掌握着MySQL发展的方向。通过专职人员和全世界开发者社区的主动支持，与 David Axmark 和 Allan Larsson 一起，Widenius 卓有成效地改善和提高了 MySQL。

名字的含义是什么

名字MySQL来源于哪里呢？MySQL手册（http://www.mysql.com/doc/en/History.html）中的一段话指出，甚至MySQL的开发者都不知道它的来源："MySQL名字的出处不是很清楚。十多年来，MySQL基本的目录和大量的库和工具都有一个'my'前缀。不过，Monty的女儿也被取名为My。甚至对我们来说，这两者之一是哪一个把名字定为MySQL仍然是个谜。"

1.2.2 特征

MySQL 始终围绕三个基本原则而设计，它们是：性能、可靠性和容易使用。严格按照这些准则产生了一个价格便宜而富有特色、适应标准而容易扩展、速度快而效率高的RDBMS，使MySQL 成为开发者和管理者建立、维护和配置复杂应用程序的完美工具。

下面将讨论 MySQL 最具有竞争性的特征。

性能

在RDBMS 中，速度（执行一个查询和返回结果给查询者的时间）就是一切。甚至MySQL的最激烈的批评家都会承认 MySQL 非常迅速，有时主要指令的执行比它的竞争者都要快。在MySQL网站上的基准程序显示了MySQL几乎比当前可用的其他所有数据库（如Microsoft SQL Server 2000 和 IBM DB2）都要优越。请参阅下面的补充资料，可以得到更多关于 MySQL 是如何达到这个高标准性能的信息。

注释：可以在 http://www.eweek.com/article2/0,3959,293,00.asp 阅读基准程序测试的完整结果。

速度需求

MySQL性能显著的部分原因是它的允许多个并发数据库访问的完全多线程体系结构，这个多线程的体系结构是MySQL引擎的核心，允许多个客户同时读取同一个数据库，并且提供了大量的性能增益。MySQL代码也以模块化、多层次方式构建，为连接和索引这样复杂的任务提供了最小的冗余和特殊的优化。

MySQL设计者最初还省去了很多在竞争系统上导致性能降低的特性，包括事务、参照完整性和存储过程（这些特性增加了服务器的复杂性并且导致了性能碰撞）。但是用户会要求这些特性，这导致了一个创造性的折中：MySQL 3.23.34a 以后的版本包括对事务的支持，但允许用户选择使用它们（失去一些性能权衡）还是拒绝它们（继续高效率地运作）。这种选择甚至可以在表对表的基础上使用，为性能达到最大化提供了细微的优化。

最后，MySQL 4.0 还包括一个独一无二的新特性：查询高速缓冲存储器。它快速缓存常用查询的结果并且把这些存储的数据返回给调用者，而不必每次重新执行这个查询，这样就显著地改进了系统的性能（这与竞争的系统不同，比如 Oracle，因为那些系统仅仅高速缓存执行计划，而不存储结果。它们仍然需要执行查询，包括所有的连接，并且在每次运行时都要重新获得查询结果）。MySQL基准程序声称这个特性以200多个百分点改进了系统的性能，而且在客户端不要求特别的程序设计。

可靠性

大多数情况下，高级数据库的性能是以降低可靠性为代价的。然而 MySQL 不是这样，它提供了最大可靠性和正常运行时间，并且在大量要求严格的应用系统中已得到测试和认证。

MySQL巨大的用户基础有助于快速查找和解决存在的缺陷，并且能在各种环境中测试软件，这种方法已经造就了几乎完全没有缺陷的软件。此外，MySQL 的每一个新版本都必须经过MySQL的一系列内部测试（测试每个特性，而且还包括对以前固有的缺陷进行测试）和MySQL的 crash-me 工具测试，后者的主要目的是通过达到（或超过）它的极限来评估系统的能力。

容易使用

MySQL易于使用，初学者在几小时内就可以领会它的基本知识，并且能得到很好的支持：一本详细的手册、大量的免费在线指南、一个知识渊博的开发者社区以及大量的书籍（这本书就是其中之一）。当大多数与MySQL服务器交互的功能通过命令行界面发生时，许多基于浏览器和其他方式的图形工具还能够简化用于控制和管理MySQL数据库服务器的任务。最后，与具有数百个可以调整参数的专用系统不同，MySQL非常容易调整和优化，甚至对具有过分要求的应用程序也如此。对于商业环境，MySQL 更进一步得到了 MySQL AB 公司的支持，它提供了专业训练课程、顾问服务和技术支持。

多用户支持

MySQL是一个完全多用户系统，这意味着多个客户可以同时访问和使用一个（或更多）的MySQL数据库。在基于网络的应用程序（要求支持由多个远程客户同时建立的连接）的发展期间，这是具有特殊意义的。MySQL也包括一个强有力的、灵活的权限系统，它允许管理者使用基于用户和基于主机的认证方案的组合来限制对敏感数据的访问。

可扩展性

没有很多性能的下降，MySQL 就可以处理非常巨大和复杂的数据库。包含有几十万条记录、几十亿个字节的表是常见的，MySQL 网站本身就要求使用包含 5000 万条记录的数据库。在MySQL 网站引用的“eWEEK”杂志评测中，MySQL 有效地满足了 50 到 1000 个并发用户的访问。并且在超过 600 个用户的限度的情况下，MySQL 的性能并没有明显的下降。像 SAP，Yahoo!，NASA和Texas Instruments这样的商业集团；大容量网站，如Google（http://www.google.com/）和 Slashdot（http://www.slashdot.com/）；政府组织，如美国人口普查局和罗德岛州政府，都使用MySQL来加强他们的系统。从 http://www.mysql.com/press/user_stories/ 来看，他们都对系统非常满意。

注释： 可以在 http://www.eweek.com/article2/ 0,3959,293,00.asp 查看基准程序测试的完整结果。

可移植性

MySQL对UNIX和非UNIX操作系统都适用，包括Linux，Solaris，FreeBSD，OS/2，MacOS以及Windows 95/98/Me/2000/XP和NT，它可以在一系列体系结构上运行，包括Intel x86，Alpha，SPARC，PowerPC和IA64；它还支持从低档的386系列到高档的Pentium机器和IBM zSeries大型机等很多的硬件配置。

遵循现有标准

MySQL 的发展工作组试图使 MySQL 尽量适应标准。MySQL 4.0 支持 ANSI SQL-99 标准的大部分重要特性，而且支持在未来版本中希望增加的标准。另外，MySQL 使用自定义扩展、

函数和能够改善可移植性和给用户提供强大功能的数据类型，扩展了 ANSI 标准（见下列补充资料，从这里可以得到更多关于 MySQL 与其他适应 SQL 的数据库系统之间兼容性的信息）。

合适的选择

MySQL 是否适合于你，或把当前的 RDBMS 转换成 MySQL 是否值得？精明的决定是先看一下 http://www.mysql.com/information/crash-me.php 和 http://www.mysql.com/information/benchmarks.html 的特性比较网页，在上面对 MySQL 和它的竞争对手进行了短兵相接的比较。

国际化

MySQL 作为一个被全球很多国家数百万人使用的系统，如果没有包括对各种语言和字符集的支持，将是不正常的。MySQL 4.0 支持很多重要的字符集合（包括拉丁语、汉语和欧洲字符集）。从版本 4.1 开始，支持 Unicode 字符集。

广泛的应用程序支持

MySQL 应用程序编程接口（API）面向很多编程语言，因此用自己的语言写出数据库驱动的应用程序是可能的。当前，MySQL 与 C，C++，ODBC（开放数据库互连），Java，PHP，Perl，Python 和 Tcl（工具命令语言）有接口，本书第 18 章到第 21 章将讨论如何使用 MySQL API 开发用 C，Perl 以及 PHP 语言编写的应用程序。

热心的开发者社区

与大部分开放源代码工程一样，MySQL 得到了一个开发者社区的积极支持，开发者社区至少在一定程度上负责软件的普及。大容量、消息灵通的邮件列表和用户组有助于问题和难题的快速解决，而且忠于 MySQL 用户和开发者的全球网络提供了丰富的建议、缺陷修补和第三方的实用程序。

开放源代码

MySQL的开发者MySQL AB是开放源代码运动的坚定支持者，并且MySQL软件可在GPL（见下列补充资料）下畅通使用。用户可以免费下载和修改符合他们需要的应用程序的源代码，并且可以免费使用它来增强自己的应用程序。这个公开许可政策促进了 MySQL 的普及，成就了一个积极而又富有热情的 MySQL 开发者和使用者全球化社区。通过在全世界数百万装置上大规模地测试软件的可靠性和不断扩展软件引擎，以保持领先于最新的技术和最新的发展，让 MySQL 走在它的竞争者的前面，这个社区起了积极的作用。

关于 GPL

应该注意到，MySQL服务器及关联的驱动程序得到了GPL的认可，因而如果我们的应用程序也得到 GPL 的认可（或者是由 MySQL AB 通过的其他任何相容的开放源认可），就可以免费使用和在自己的软件应用中重新发布它们。在这种情形下，MySQL 软件免费提供给我们使用。

可是，如果我们的 MySQL 驱动应用程序没有得到 GPL 或一个等价的认可方案的特许，而我们又想重新发布它（无论在内部还是在外面），我们都需要从 MySQL AB 购买一个同样的商业许可。

MySQL AB 从销售这些许可和为 MySQL 数据库服务器提供支持、培训和咨询服务两方面取得收入。

1.2.3 应用

MySQL的官方网站引述MySQL是“世界上最受欢迎的开放源代码数据库”。这不是狂妄之语，数字可以证明它：根据MySQL网站上最近公布的统计，全世界有超过400万的系统使用MySQL，每天下载的MySQL数据库服务器副本超过25 000个。

注释：可以在http://www.mysql.com/访问MySQL的官方网站。

作为一个可信赖的、富于特色的数据库系统，MySQL已经在商业、教育、科学和工程领域得到应用（MySQL AB的客户名单足以证明这个事实，客户包括Motorola，Sony，NASA，HP，Xerox和Silicon Graphics等）。现今，MySQL支持多种应用，它们包括Internet网站、电子商务应用程序、搜索引擎、数据仓库、嵌入式应用程序、大容量存储以及要求严格的软件系统等。

今天，MySQL的主要应用程序出现在网络舞台上，这并不令人吃惊。随着网站以及基于Web的分布式应用程序变得越来越复杂，有效管理数据来改善处理效率、降低响应时间和提高用户的全面技能就变得越来越重要了。因此，我们迫切需要一个速度快、性能稳定和安全的数据库管理解决方案（可以非常省心地配置和使用它，并且为将来的发展奠定坚实的基础）。

很多原因让MySQL正合需要。经过证实的记录让它的可靠性和寿命得到保证，开放源代码的根本能够确保迅速调整缺陷和性能持续增强的周期（更不必提及较低的总体拥有成本）；对不同编程语言和技术的可移植性和支持，使它适合多种应用程序；它的低成本/高性能的价值表现吸引了很多人，包括家庭用户、中小型商业机构和政府组织等。

MySQL不支持的特性

MySQL是当前可以使用的特性最全面的开放源代码数据库系统之一，然而少数特性它没有提供！这里有一个简单列表：

- **存储过程** 预先定义的SQL语句序列，它们被编译和存储在数据库内以求更有效地执行。
- **触发器** 当一个特定事件（比如表内数据的一个变化）发生时自动执行的数据库操作。
- **视图** 源于现有表内容的虚表，它能够以与标准表完全相同的方式执行操作。

MySQL的发展路线表明以后的软件版本中将包含这些特性。

1.3 小结

本章对MySQL进行了简要介绍，先对数据库怎样工作进行了讲述，然后对MySQL关系数据库管理系统进行了综述。本章使读者了解了MySQL的历史和发展进程，证实了MySQL广受全世界开发者欢迎的基本特征，并且介绍了MySQL的一些最常见的应用程序。

现在读者对MySQL已有少许了解，接着可以开始下一层次的学习了。下一章将给出MySQL更多技术上的综述，介绍MySQL体系结构和MySQL数据库引擎的重要特性。

第2章　MySQL技术教程

通过很多能够操作存储在不同类型数据库系统中的数据的应用程序实例，本书大部分内容将讲述MySQL引擎的能力，举例说明MySQL的使用。在开始使用它之前，对MySQL如何工作有一个充分的技术上的理解是非常重要的，这样就可以更好地配置和优化应用程序，以便符合我们的需要。

为此，本章主要对MySQL的各组成部分和功能进行概述。理解这些特性对我们更进一步阅读本书、应用我们的知识开发基于MySQL的应用程序是非常有用的。

2.1　MySQL体系结构综述

MySQL建立在一个层次体系结构上，由主子系统和辅助部件组成，它们之间相互配合，读取、分析数据和执行查询，并且可以高速缓存和返回查询结果。

2.1.1　主子系统

MySQL体系结构由五个主子系统组成，它们协同工作，响应MySQL数据库服务器发出的请求。这五个主子系统是：

- 查询引擎
- 存储管理器
- 缓冲管理器
- 事务管理器
- 恢复管理器

这些特性的组织结构如图2.1所示。我们将简要解释每一个主子系统，以便更好地理解这五个部分是如何结合在一起工作的。

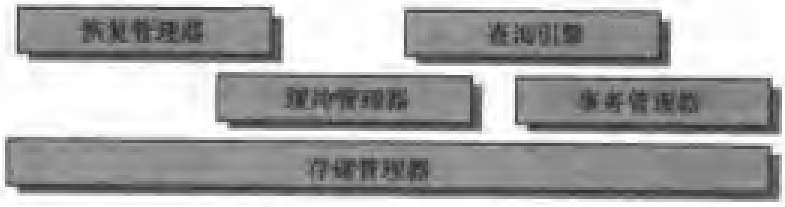

图2.1　MySQL子系统

查询引擎

这个子系统包含三个相互关联的部件：

- 语法分析器
- 查询优化器
- 执行部件

语法分析器把它接收到的SQL指令由调用程序分解成可以让MySQL引擎识别的一种形式。将要被使用的对象连同语法的正确性一起被验证。语法分析器同样对引用的目标对象进行检查，以保证调用程序的权限允许它使用这些对象。

然后查询优化器简化语法句子，使之更有效率，以备执行部件使用。这样就准备好了最有效的查询执行计划。查询优化器检查使用哪一个索引会尽可能快速、有效地取出数据。然后从找到的几种方式中选择一种，生成能够让执行部件识别的执行计划。

查询优化器使用基于概率的归纳法简化数据。如果已经知道了访问查询结果的理想方法，我们就可以越过它，这样就避免引擎使用另一个不够优化的计划。如果不希望引擎查找最优化的方法，而是希望它使用特殊的索引，那么MySQL为我们提供了给引擎设置某些“提示”的选项。

执行部件然后解释执行计划，并且根据它接收到的信息，向其他部件发出取出记录的请求。

存储管理器

存储管理器与操作系统（OS）相互配合，有效地向磁盘存入数据。因为存储功能属于一个单独的子系统，所以MySQL引擎在不同于操作系统的抽象级上操作。这意味着如果我们连接不同的操作系统，它们使用不同的存储机制，我们只要改写引擎的存储代码选项即可，而引擎的其他部分不用改变。在MySQL函数库的帮助下，和存储系统内部数据一样，存储管理器把在用户表、索引以及日志文件中的所有数据存入磁盘。

适应环境

MySQL利用查询引擎和其他部件的分离性提供了除固有的MyISAM处理程序以外的表处理程序，比如HEAP和MERGE表类型处理程序。它还为每个表处理程序提供了新的存储管理器和新的事务管理器。因为InnoDB和BDB表类型具有在MySQL类型中不存在的事务特性，所以它们通过各自的第三方创建者得到修改，以便插入到MySQL现有的查询引擎中。

在第8章中可以看到更多的关于MySQL支持的各种表类型的信息。

在MySQL中，还使用一种特殊的查询高速缓冲存储器。如果一个查询返回一组确定的记录，重复相同的查询应该返回相同的记录，除非基础数据以某种方式改变了。这种情形看上去很明显，但是其他主要关系数据库管理系统厂商很少能够提供很好地利用这一规则的特性。这些数据库产品通常能够有效存储最优化的访问计划，这些计划能详细描述数据取出的过程，这样的计划允许与以前生成的查询相似的查询，而跳过再一次分析索引的过程。

通过在内存中存储结果集合本身，结果集合高速缓冲存储器更进一步拓展了这个规则，因而完全回避了搜索数据库的需要。查询结果数据简单地存储在高速缓冲存储器中，当一个相似的查询发出时，这些数据又被返回，好像在回应第一次生成它的查询一样。

MySQL引擎使用一种非常有效的查询结果高速缓冲存储器机制，就是我们所熟知的查询高速缓冲存储器，它显著提高了要求取出的数据与以前的查询相同的查询的响应时间。

这个机制非常有效，以致一个主流的计算出版物宣称MySQL查询速度比Oracle和SQL Server的速度都要快。使用带有预先录制好的记录或数据驱动网页的MySQL的决策支持系统，可以提供远远超过那些没有查询高速缓冲存储器的系统所预期的响应速度。

缓冲管理器

这个子系统负责处理所有在查询引擎和存储管理器数据请求中引起的内存管理问题。MySQL优先使用内存来缓存能够返回的结果集合，而不是向存储管理器发出重复的请求。高速缓冲存储器由缓冲管理器维护。

当新的记录在等待目标表和索引使用时，也可以存储在缓冲管理器中。如果需要新的数据，要先从存储管理器中取出数据，放置在缓冲管理器中，然后发送到查询引擎。

事务管理器

事务管理器的功能是促进数据访问的并发性。这个子系统提供一个加锁工具，保证同时访问的多个用户协调地访问数据，不会以任何方式破坏或者损坏数据。事务控制通过锁定管理子系统发生作用，锁定管理子系统对在事务中使用的各种对象进行加锁和解锁操作。每个事务表处理程序执行各自的事务管理器来处理所有的加锁和并发性需求。

恢复管理器

恢复管理器的工作是防止数据丢失，保存数据副本以备后来提取。它还负责把数据库中修改数据和其他重要事件的指令记入日志文件。

到目前为止，只有 InnoDB 和 BDB 表处理程序提供恢复管理功能。MyISAM 处理程序没有事务恢复过程，只提供了使用某种恢复特性防止服务器中断的机制，这些特性克服了任何内在的不一致性，不一致性可能导致数据库系统崩溃的后果。不一致性通常跟索引相关，这些索引在没有被完全更新之前就用来表示尚未被完全写进数据库的表或记录的内容。

2.1.2　辅助部件

除了五个主子系统以外，MySQL 体系结构还包括下面两个辅助部件：

- 过程管理器
- 函数库

过程管理器

这个部件在系统中有两个功能。第一，通过与客户连接的网络管理模块管理用户连接；第二，通过多线程、线程加锁和执行线程安全操作模块，使竞争性的任务和过程同步。

函数库

这个部件包含被其他所有子系统使用的通用例程。它包括字符串处理程序、排序操作程序以及像内存管理和文件 I/O 之类的操作系统特定功能。

2.1.3　子系统 / 部件相互作用和控制流

查询引擎要求从缓冲管理器读取数据或者把数据写入缓冲管理器以满足用户的查询。它依赖事务管理器请求锁定数据才能保证并发性。要生成和删除表，查询引擎要直接访问存储管理器以生成或者删除文件系统中的文件，而不用访问缓冲管理器。

缓冲管理器高速缓存来自存储管理器中的数据，查询引擎就可以有效地取回数据。在执行任何修改操作之前，检查数据的锁定状态依赖于事务管理器。

只有在查询高速缓冲存储器和存储管理器分别对内存和文件系统中的数据加锁后，事务管理器才能正常工作。

恢复管理器利用存储管理器来存储指令、事件日志文件和文件系统中的备份文件。获得被写入日志文件的锁定状态依赖于事务管理器。在从崩溃状态恢复期间，恢复管理器还需要利用到缓冲管理器。

存储管理器必须依赖操作系统的文件系统才能永久性地存储数据和取出数据。获得锁定状态的信息依赖于事务管理器.

2.2 MySQL 引擎

MySQL 支持不常见的 5000 万条记录的数据仓库，同样也支持小型、嵌入式的应用程序。这种多功能性在某种意义上是可能的，因为 MySQL 引擎的设计具有最大限度的可缩放性、最大程度的资源效率性能和对各种平台和体系结构的易移植性。本节将详细讨论这个引擎的重要特性。

2.2.1 连通性

MySQL是在假定它的大部分应用程序都运行在TCP/IP协议的基础上而设计的。TCP/IP协议不仅非常健壮、安全，而且通用于我们很可能遇到的 UNIX，Windows，OS/2 以及其他几乎所有的重要操作系统上，这是非常好的情况。当客户机和服务器在同一个 UNIX 机器上时，MySQL 使用带有可以在 UNIX 领域内运行的 UNIX 套接字（socket）的 TCP/IP 协议，也就是说，它们通常在同一个 UNIX 系统的进程之间被使用（与运行于网络之间的网络套接字相对应）。

MySQL 允许命名管道连接，这样设计主要是为了支持在早期没有 TCP/IP 协议的网络中的网络连接，比如 LAN 管理器和 Windows NetBEUI（NetBEUI 使用一种基于 NetBIOS 机器名称的寻址方案，而不是一个可路由的 IP 地址）。

2.2.2 SQL

结构化查询语言（SQL）是一个开放标准，自 1986 年以来，它一直由美国国家标准协会（ANSI）进行维护。虽然这个标准在不同厂商的支持程度不同，但是可以很公正地说，SQL 是当今使用最广泛的语言之一。与其他系统的 SQL 实现一样，比如 SQL Server 的 T-SQL（Transact-SQL）和 Oracle 的 SQL，MySQL 也具有自己的 SQL 标准变体，它添加了强大的性能，超过了在这个标准之内可用的性能。虽然会在以后的章节中详细研究 MySQL 的 SQL 查询，但是在本节中将介绍它的用途，以提供一个理解 MySQL 数据提取和决策支持性能的框架。

需要说明的是，SQL 非常容易入门，使用下面一个简单的语句就可以检索数据：

```
mysql> SELECT year, make, model, price FROM vehiclesales LIMIT 10;
+------+-----------------+---------------------+-------+
| year | make            | model               | price |
+------+-----------------+---------------------+-------+
| 1982 | AMERICAN MOTORS | EAGLE 50-4WD-6 CYL. |   575 |
| 1983 | BUICK           | CENTURY LIMITED-V6  |  1000 |
| 1983 | BUICK           | REGAL LIMITED       |   525 |
```

```
| 1983 | BUICK          | REGAL LIMITED       | 1250 |
| 1983 | BUICK          | REGAL LIMITED       | 2250 |
| 1983 | BUICK          | REGAL LIMITED       |  700 |
| 1983 | BUICK          | SKYLARK LIMITED-V6  |  500 |
| 1983 | BUICK          | LESABRE CUSTOM-V8   |  550 |
| 1983 | BUICK          | ELECTRA LIMITED-V8  |  500 |
| 1984 | BUICK          | CENTURY CUSTOM-V6   |  850 |
+------+----------------+---------------------+------+
10 rows in set (0.00 sec)
```

可以看到，SQL的语法并不神秘，而且也不难理解——事实上，它看上去就像英语口语（对英语演讲者来说很重要）。如果想让上面的结果集合按price进行排序，我们只要在语句的结尾加上短语ORDER BY price，然后运行它就可以了。

与C++，Java和Perl相比，SQL乍看上去好像很容易学习，不容易混淆，这经常使SQL的新接触者认为学习曲线要比真实的情况短得多。事实上，掌握SQL花费的时间与学习其他语言需要的时间一样多（人们已经编写了完整的关于使用SQL进行数据抽取的书籍，甚至没有提到SQL提供的数据处理功能）。观察下面的查询，它也许是需要我们执行的比较典型的查询类型：

```
mysql> SELECT
a.year,
a.make,
a.model,
a.trim,
AVG(a.price) AS s_price,
AVG(b.price) AS a_price
FROM vehiclesales a INNER JOIN vehiclesales b
ON
a.year = (b.year + 1) and
a.make = b.make and
a.model = b.model and
a.trim = b.trim
GROUP BY
a.year,
a.make,
a.model,
a.trim,
b.year,
b.make,
b.model,
b.trim
HAVING AVG(a.price) > (AVG(b.price)* 1.25);
```

这个查询取回了具有平均价格（是前年同一类型交通工具平均价格的25%）的交通工具的名称。

2.2.3 数据完整性

通过使用主键和外键约束，MySQL支持引擎级的数据完整性。MySQL可以对列[1]下定义，

① 编者注：在本书中，并不区分列（column）和字段（field）的含义。

以便明确地表明NULL值不能出现在列中。为了防止出现空列，MySQL支持默认值的使用，当与NOT NULL属性组合时，就能保证把有效的数据输入一个列，否则这个列为空。

即将来临

MySQL将来的版本估计在所有的表类型中都有外键，当前参照完整性语句只在InnoDB表类型中可用。

2.2.4 事务

直到最近，MySQL才以处理事务的能力而出名。然而在版本3.23以后，MySQL就提供表处理程序，比如InnoDB表和BDB表，它们使用与其他商业RDBMS产品相同的方式管理事务。

一个安全事务数据库系统必须通过知名的ACID测试才有遵守标准的资格。一个适应ACID的数据库必须支持下面的特性：

- 原子性
- 一致性
- 分离性
- 持久性

让我们来详细了解这些特性。

原子性

事务被定义为一个或者一系列可以访问或改变数据库内容的动作。在SQL术语中，当一个或更多的SQL语句作为一个单元运行时，一个事务就发生了。在这样一个单元中，每一个SQL语句都与其他的SQL语句相互依赖。换句话说，如果一个语句没有完成，则整个的单元就会回滚，所有改变了的数据都要返回到事务开始以前的状态。把SQL语句分组作为一个单元（或者事务）的一部分将指示MySQL：整个单元将会被原子性地执行。

原子执行是一个要么全做要么全不做的命题。所有SQL语句必须被完成，才能使数据库维持一个数据完整性状态，否则就没有语句被执行且提交给磁盘。在MySQL中，事务的开始标记为一个BEGIN语句。直到发布一个COMMIT命令来指示MySQL完成这个动作，我们才认为完成了一个事务（或工作单元）。必要时，ROLLBACK命令会把所有的改变初始化到BEGIN命令执行之前的状态。

在银行业务中可以找到这样一个日常现实世界中的例子。通过在银行账户上取款和存款，银行就可以在一个事务之内增加和减少账户上资金的数目。这些更新通常涉及到多个表。只有保证整个事务发生，而不是仅仅事务的一部分发生，银行才能够维持数据的完整性。

事务管理对执行数据输入的客户-服务器系统，以及必须能够保证高度安全以防无法觉察的数据丢失的应用程序都是非常重要的，比如上面描述的银行业务。

一致性

当每一个事务使得系统处于一致状态时，不管事务是成功完成还是中途失败，一致性都会存在。

例如，设想银行使用一个事务，它支持把钱从一个银行账户转移到另一个。如果事务从一个银行账户取出需要的数目，而没有把相应的数目存入另一个银行账户，这时系统将不再处于一致性状态。在这种情况下，事务违反了一致性约束条件，系统将不再适应 ACID 标准。

在 MySQL 中，一致性主要由 MySQL 日志机制来处理，它记录了对数据库所做的所有改变，并且为事务恢复提供检查跟踪依据。如果系统在事务执行中途关机了，MySQL 恢复程序将使用这些日志来检查事务是否顺利完成，必要时要执行回滚。

除了日志程序以外，MySQL 还提供了锁定机制，保证组成事务的所有表、行和索引由初始化程序锁定足够长的时间，以便提交事务或者回滚事务。

分离性

分离性意味着每个事务在各自的空间内发生，与系统内可能发生的其他事务相互独立，而且还意味着只有组成事务的全部事件序列完全被执行后，事务的结果才可见。在这样的系统中，即使有多个事务同时发生，分离性原则也会保证只有在事务完全完成之后，单个事务的结果才是可见的。

当系统支持多个同步用户和连接时（像 MySQL 那样），分离性是非常重要的。当一个事务在执行过程中被其他竞争的、常常是相互矛盾的事务迅速侵犯时，不支持这个基本原则的系统会导致严重的数据破坏。

有趣的是，MySQL 提供了服务器端信号变量的使用，这些信号变量充当通信管理器，有助于程序管理它们自己的分离机制。在不希望引起事务管理器系统开销的情况下，或当一个恢复计划在日志恢复范围外就可能进行时，这些变量是很有用的。MySQL InnoDB 表在包含多个查询的事务中提供分离性，而 MyISAM 表允许我们通过 LOCK TABLES 命令来模拟分离性。

持久性

持久性意味着即使在系统崩溃时，也要保存已提交的事务所做的改变。当一个事务完成并且数据库的日志被更新时，持久性就开始起作用了。通过对在数据库中以任何方式改变了数据的所有活动保存一个日志的方法，大多数 RDBMS 产品能保证数据的一致性。这个数据库日志追踪每一个对表、查询、报告等所做的更新。如果打开过数据库日志文件，我们就会知道当它开始写数据时，使用它会使数据库的执行速度减慢（但是它不会影响查询速度）。

在 MySQL 中，根据应用程序的特点，通过选择适当的表处理程序，我们可以指定是否使用事务。InnoDB 表处理程序执行日志文件与 BDB 不同，而 MyISAM 不支持保证持久数据库的日志类型。默认情况下，InnoDB 表对崩溃前最后 1 秒的数据变化情况有 100% 的持久性。MyISAM 表提供部分持久性（最后一个 FLUSH TABLES 命令之前提交给系统的所有改变能保存到磁盘）。

从本质上说，通过修复一个以前的备份数据库，“重新执行”所有后来的事务直到失败点，SQL Server 和 Oracle 能够把数据库恢复到以前的一个状态。这些数据库产品不支持对日志文件的直接使用（也不显示日志文件的内部数据结构），因为这些文件是数据库引擎恢复机制的一部分。

MySQL 还以连续的顺序为所有的数据操作行为保持一个二进制日志。然而，与其他数据库中的日志不同，这个日志容易阅读，因此根据这个日志使用最近的备份来恢复丢失的数据是一件比较简单的事情。

2.2.5 可扩展性

在大部分RDBMS产品中，可以通过使用存储过程来扩展数据库的能力，其方法通常是通过提高包含控制流语句和条件逻辑的SQL来进一步扩展编程能力，像带有T-SQL的SQL Server和带有 PL/SQL 的 Oracle 那样。

至今，MySQL不包括对存储过程的支持，但是这个RDBMS的优点之一是它的可扩展性。在它的开放源引导程序中，MySQL把原始资料代码作为发布版本的一部分，允许开发者把经过编译输入主机的新功能和特性增加为核心产品的一部分。MySQL 还允许在启动时把个别 C 和 C++ 库装载入与该引擎相同的内存空间中。

这些方法允许用户像使用MySQL中已经嵌入的、可用的任何功能［比如SUM()和AVG()］一样来使用这些功能。因为这些功能与MySQL在相同的内存空间上运行，并且在MySQL所在的服务器上完成，所以可以使调用程序和服务器之间的网络通信量减到最少，因此性能越来越强大。

可以通过一个专门的用户自定义功能接口给 MySQL 增加功能。用户自定义功能最初是作为特殊C/C++库生成的，然后用CREATE FUNCTION和DROP FUNCTION语句动态地添加和删除。用户自定义功能随之带来了更多的负担，我们不得不在每次安装 MySQL 时安装所有的库。然而，这样会使得部署更快速，因为我们可以把这些功能加载到二进制发布版本中，而不必费力地再一次编译MySQL来合并新的功能。另外，让我们的库在MySQL新的版本发行时能够无变化地继续起作用，这通常是一个好办法。

还可以把添加的功能作为MySQL本身具有的（嵌入）功能。本身具有的功能被编译进入MySQL服务器引擎，并且成为MySQL本身安装的一个永久部分。这使得新的安装比使用库要容易得多，因为一旦定义了我们的发布版本，就可以确信以后所有的安装将包含这些已经添加的功能。否则的话，必须为 MySQL 的每一个新的版本重新编译和生成我们的发布版本。

获得创新

在MySQL中，通过在一个表中存储代码，然后重新获得这些代码的文本格式，再由调用程序执行它，这个过程可以模拟存储过程的封装和集中化。例如，一个Perl程序具有一个各行中只保存Perl代码的表，客户程序要取出需要的代码块并且在运行时评估这些代码。

这个技术允许我们在服务器上设置逻辑，它并不要求客户程序意识到任何代码的改变。因为客户机要与服务器多次往返通信，并且这个方法只用于解释语言，如果比较从其他RDBMS实现的存储过程中获得的所有性能来说，它不是一个真正的替换，但是很相近。

2.2.6 MySQL 的对称多处理

为了利用多处理器体系结构，MySQL 采用多线程设计，允许在多个处理器之间分派线程以达到更高的并行度。对需要理解 MySQL 如何最好地利用处理器能力的数据库管理员和可以使用自定义功能扩展 MySQL 的开发者来说，熟悉这一点很重要。所有的自定义功能必须是线程安全的。更确切地说，在与 MySQL 相同的进程中，它们必须不会妨碍其他线程的工作。

进程和线程的区别

为了在整体上更好地理解线程，我们必须注意线程和进程之间的关系。每个进程包含一个执行程序和一个资源集，比如文件映射和地址空间。与给定的任务相关的所有线程共享任务的资源。因此，一个线程本质上是一个程序计数器，一个堆栈和寄存器集合，所有其他数据结构都属于任务。一个进程就像一个带有单个线程的任务一样有效地执行操作。

线程是一个拥有自己的堆栈和执行一个给定代码段的小型进程。一个线程正常情况下与其他线程共享内存，而真正的进程总是要为每一个进程分配一个不同的内存区域。线程组是在内部执行同一个进程的所有线程的一个集合。它们共享相同的内存，因此可以访问相同的全局变量、相同的堆内存、相同的文件描述符集合等。当在单个处理器上运行时，一个线程组中的所有线程通过使用时间片执行操作，如果多个处理器可用时，则可并行执行操作。

使用一个线程组比使用一个进程组的优势是线程之间的语境转换要比进程之间的语境转换快得多。换句话说，系统从一个线程转换到另一个线程要比从一个进程转换到另一个进程快得多。两个线程之间的通信要比两个进程之间的通信快速而且易于实现，因为线程总是共享共同地址空间中的共享变量。

POSIX 线程库是一个用于 C 和 C++ 的基于标准的线程 API。它们最有效地用在多处理器系统上，这样程序流可以被安排在另一个处理器上运行，然后通过并行或分布式处理提高速度。线程要比分支或产生一个新的程序需要更少的开销，因为系统不会为程序初始化一个新的虚拟内存空间和环境。POSIX 最适于用在一个多处理器的系统上，而且在单个处理器系统上也取得了成效，这些单处理器系统能够在输入/输出（I/O）设备和其他可以中止进程执行的系统功能上利用等待时间。

根据不同的平台，MySQL 使用各种线程程序包。POSIX 线程用在大多数 UNIX 变体上，比如 FreeBSD 和 Solaris。LinuxThreads 用于 Linux。由于效率的原因，Windows 线程用在 Windows 平台上，但是处理它们的代码是为模拟 POSIX 线程而设计的。

MySQL 是一个线程应用程序，因此它能够让操作系统接管调整线程分派的任务来平衡多个处理器的工作量。MySQL 利用这些线程来完成下面的事情：

- 全局连接线程处理所有的连接请求，并且生成一个新的专用线程来为每一个连接处理认证工作和 SQL 查询处理过程。这种类型的线程在 TCP/IP 和命名管道的连接上起作用。
- 一个单独的线程处理内部信号和报警监听功能。
- 当被请求时，生成一个专用线程来刷新在指定区间内的所有表。
- 使用一个延时的 INSERT 命令的单表获得自己的线程。
- 在复制时，主机的同步由个别线程处理。

当然，利用多重处理的另一种方式是在同一个机器上运行多个 MySQL 的实例，因此需要为每一个实例产生一个单独的程序。这种方法对提供主机服务的公司和提供内部协作环境的主机服务公司尤其实用。通过在同一个计算机上运行多个 MySQL 实例，我们可以很容易地提供多个需要不同配置选项的用户数据库。

2.2.7　安全性

访问一个 MySQL 数据库的过程可以分解成两个任务：连接 MySQL 服务器和访问个别对象，比如表或列。MySQL 有一个内建的安全性来检查两个阶段上用户的可信度。

MySQL 通过用户表管理用户认证，用户表不仅核查用户是否使用适当的用户名和口令正确登录，而且还核查连接是否来自一个授权的 TCP/IP 地址。

一旦用户被连接，系统管理员就给予用户在对象和 MySQL 中实施的行为上的用户专有权力。例如，我们可能允许 fred@thiscompany.com 在一个库存表上只执行 SELECT 查询，而允许 anna@thatcompany.net 在同一个表上运行 INSERT，UPDATE 和 DELETE 语句。

安全性也是一个连接问题。由网络传输的口令和重要数据，容易受到市场上的许多网络信息包分析器的任意一个的拦截攻击。默认情况下，MySQL的较老版本对用户名和口令使用一种相当简单的加密方法，它可能会阻止非预谋的黑客，但不会阻止那些愿意花一点时间破解加密算法的人。从版本 4.1 开始，MySQL 采用一个比较安全的协议，即使一个黑客有发现和破解网络通信的能力，入侵 MySQL 也会比较困难。

在网络上传输的实际数据（比如查询结果）是不加密的，因此可以被黑客看到。要保护数据安全，可以使用 SSH（Secure Shell，安全外壳）协议，需要在客户机应用程序和使用的操作系统上安装它。如果使用的是MySQL 4.0或者更高版本，还可以使用SSL（Secure Socket Layer，安全套接字层）加密协议，它可以配置成在 MySQL 内起作用，使得在 Internet 上或者其他公共网络上的使用比较安全。

2.2.8 复制

复制是一种数据发布机制，它允许在远距离的地方放置表和数据库的副本，以便用户可以更加方便地访问它们。

产品数据库就是使用这种复制机制的典型。例如，一个国内或全球公司可能有一个通用的集中更新的产品数据库，但在每个本地办公室能使用它。其方法不是每次需要的时候强迫应用程序远距离地查询这个表，而是为每个人发布一个副本，这样会更有效，因为每个办公室只负担了一次传输的开销。

MySQL 3.23.15之前的版本没有复制功能，在这个版本发行之后到今天，MySQL 支持单向复制，即一个数据库作为主数据库，一个或者更多数据库作为从数据库。

复制机制依赖一个 MySQL 日志来追踪对数据库做的所有改变。主数据库把这个日志传送给从数据库，然后从数据库在自己的数据上应用这个日志。一个从数据库不能阻止用户在复制环境以外的情况下更新日志，因此必须谨慎对待，确保这种情况不会发生，否则同步的相对确信度就会丧失，以后的日志倒查甚至可能会失败。

2.2.9 应用程序接口

对应用程序开发者来说，MySQL 提供了一个用 C 语言编写的客户端库和一组 API（应用程序接口）。API 提供了一组可以理解的规则，通过这些规则，主机语言可以连接 MySQL 和发送命令。使用 API，可以保护客户端程序不受 MySQL 中能够影响连通性的任一改变的破坏。

许多 API 都是可用的，其中有一些几乎对所有的编程语言都可用。多数 API 使用除 Java 驱动程序以外的 C 客户端库，Java 驱动程序是以 100% 纯 Java 语言写成的。当前，MySQL API 对 Perl，PHP，C/C++，Java，Visual Basic，Python，Ruby 和.NET 都可用。

2.2.10 附加工具

与 MySQL 相配合的软件程序列表在不断增加。下面是较好的软件程序的简要列表：

- MySQL CC，来自 http://www.mysql.com/products/mysqlcc/index.html。对于 MySQL，它是一个极好的前端查询和数据库管理工具。现在有 Windows，UNIX 和 Linux 版本可用，并且 Mac OS X 的版本现在也可用。
- DBTools 专业管理器，来自 http://www.dbtools.com.br，是一个用于管理 MySQL 数据库、表和索引的图形客户，其输入来自其他 RDBMS 的数据，该管理器提供一个用于查询和报告设计的接口。
- 对基于Web的MySQL管理来说，可以尝试phpMyAdmin，来自http://www.phpmyadmin.org。

2.3 小结

要做出关于 MySQL 数据库服务器的安装、数据库和表的设计以及最后的优化的最好的决定，首先应对控制表、记录、索引和数据库行为的根本体系结构有一个清晰的理解，这是极为重要的。这些知识是基础，数据库管理员根据它做出合理的决定，应用程序设计者使用它让自己的代码达到最优化。

在本章中，我们尝试从数据库管理者 / 开发者的立场，为那些最后可能创建自己的库或者甚至添加 MySQL 源代码的人们展示了 MySQL 所有组件的基本知识。在本书的后面部分，将更加详细地讲解这些概念。

第3章　MySQL安装和配置

MySQL是通用公众许可（GPL）下的一个开放源代码应用程序，这是它的一个优点，它鼓励用户下载、修改和免费使用。这个公开许可政策对MySQL在开发者社区中被广泛接受和普遍流行起了重要作用。根据MySQL网站公布的统计资料，全世界有超过400万的站点使用MySQL，而且每天下载的MySQL数据库副本超过25 000个。

本章中，我们将通过在工作站上获得、安装、配置和测试MySQL的过程来增加这些统计数字。本章包括在UNIX和Microsoft Windows上进行二进制代码和源代码两个版本的安装说明，并且还解释了可用的MySQL服务器版本之间的差别，其目的是帮助我们选择合适的版本。

3.1　获得MySQL

本过程的第一步是访问MySQL的官方网站（http://www.mysql.com/downloads/mysql.html），得到最近的稳定版本的一个副本。这并不一定像它听上去那么简单（像冰淇淋一样，MySQL有很多种口味，我们需要选择最适合要求的那种）。

当要选择哪一个MySQL发布版本下载使用时，需要做出两个主要决定：

- 选择哪一个版本安装。
- 在二进制代码和源代码发布版本之间做出选择。

3.1.1　选择哪一个版本安装

目前，MySQL AB在它的网站上放置了下面两个可用的MySQL数据库服务器版本：

- **MySQL Standard**　这是MySQL数据库服务器的标准版本，它包括对常规的非事务表和新的安全事务表两种表的支持。该版本适合于一个要求稳定的、灵活的和健壮的数据库引擎的环境。
- **MySQL Max**　这个版本包含标准版本的特性集，并且还具有新的、更具有实验性的功能。它不是总能最好地适应生产环境，因为它通常包含一些不完全稳定的增强特性。

MySQL的这两个版本都得到了GPL的许可，并且在许可条款下可以免费下载和使用。

大多数情况下，我们应该使用MySQL Standard版本（本书所有的实例都使用这个版本，它稳定、富有特性，适合于大部分通用应用程序）。只有当MySQL Max版本包括新的、我们需要的或者渴望试验的新特性时（或者当我们是那种总是对生活充满情趣的人时），才应该选择它。

3.1.2　在二进制和源代码之间选择

MySQL AB在它的网站上放置的MySQL数据库服务器的二进制代码和源代码发布版本都可以下载使用。到目前为止，二进制发布版本可以用于Linux，Solaris，FreeBSD，Mac OS X，

Windows 95/98/2000/XP/NT/ME，HP-UX，IBM AIX，SCO OpenUNIX 和 SGI Irix，源代码发布版本可以用于 Windows 和 UNIX 两种平台。

MySQL 的商业版本

应该注意到，除了在本节讨论的两个版本以外，MySQL AB 还另外提供了两个版本的 MySQL 数据库服务器：MySQL Pro 和 MySQL Classic，它们都得到商业许可，因此不能免费下载使用。

这两个版本之间的差别是什么？MySQL Pro 包括对事务的支持（除了许可条款以外，它与 MySQL Standard 版本完全一样），而 MySQL Classic 不支持。

如果我们对购买这些 MySQL 数据库服务器的商业许可比较感兴趣，可以访问 http://order.mysql.com，从这里还可以找到更多的想要购买的东西。

在大多数情况下，使用一个预编译的二进制代码版本比使用一个源代码版本更可取（MySQL AB 花费了大量的时间和精力为各种平台建立了二进制代码版本，使用这些版本通常被认为是一个好的办法）。预编译的二进制版本还比源代码版本容易安装，它们由 MySQL 开发者优化了预编译的二进制版本，以保证它们在目标系统上的最佳性能。

但是基于很多可能的原因，对我们来说源代码版本会比二进制代码版本更好：

- 希望使用不同于 MySQL 工作组提供的默认值的编译时间选项来重新编译 MySQL。例如，我们可能喜欢对安装路径设置一个不同的值。
- 希望编译一个更小型、更轻松的 MySQL 版本，它不包括标准二进制代码版本的所有特性，从而节省开销。
- 希望能够支持默认情况下标准二进制代码版本所没有的、新的、具有实验性的特性。
- 希望对应用程序的源代码进行查看或修改。
- 如果有太多的闲暇时间，并且希望用深奥的、渊博的 MySQL 内部构件知识给隔壁的漂亮少女留下深刻印象。顺便说一句，根据调查，MySQL 用户中只有 6% 是女性。

对 MySQL 的初学者，建议使用二进制代码版本——源代码版本典型适合于富有经验的开发者，他们要么为了自己的需要调整 MySQL 的默认设置，要么对研究源代码如何工作比较感兴趣。他们总是有时间，喜欢用专门的技术来诊断或费神地编译和配置在安装过程中可能出现的问题。

确定了需要的 MySQL 版本后，把浏览器指向 http://www.mysql.com/downloads/mysql.html，从可以使用的选项上选择适当的文件。MySQL 软件还镜像反映在世界范围内的其他许多站点上，我们可以在站点 http://www.mysql.com/downloads/mirrors 上获得一个镜像列表。选择在地理上最靠近我们的站点，可以使下载速度更快。下载成功以后，把它存储在硬盘驱动器的一个目录中，然后跳到下一节。

名字的含义是什么

我们可能注意到，MySQL 版本使用一个特殊的命名模式来命名，例如 mysql-4.0.9-gamma.tar.gz，文件名中的数字表明了软件的版本号，额外的后缀表明软件的稳定级别（alpha，beta 或 gamma）。

请注意，在本章中，我们假定使用 MySQL 4.0.15，并且所有下载的文件存储在目录 /tmp（UNIX）或者 c:\temp（Windows）中，在以后的用法说明中，我们将提及这个目录。

3.2 安装和配置 MySQL

得到 MySQL 的一个副本之后，就可以着手在指定的平台上安装和配置它。本节包括了在 Windows 和 UNIX 两个平台上，使用二进制代码和源代码版本如何来完成这些事情的信息。

应该注意到，本节仅仅为安装和配置 MySQL 的过程提供了一个综述和总的向导，并不是有意要替换 MySQL 包含的安装文件。如果我们在安装或者配置这里描述的各种程序时遇到了困难，则可以访问 MySQL 网站或者搜索邮件列表，以便查找详细的故障修复信息和建议。

3.2.1 在 UNIX 上安装和配置 MySQL

MySQL以二进制形式几乎适用于所有的UNIX版本，并且可以被编译成用于那些没有二进制代码版本的 UNIX 变种。本节将论述在可以使用源代码和二进制代码两种版本的 Linux 系统上安装和配置 MySQL，虽然我们应该引用包含在 MySQL 版本中的文件来获得特殊平台注释，但这个过程对其他的 UNIX 变种是相似的。

从二进制 RPM 格式安装 MySQL

在一个 Linux 系统上安装 MySQL，我们推荐的方式是通过 RPM 安装。MySQL AB 在它的网站上许可下面的 RPM 文件可以下载使用：

- **MySQL** MySQL 数据库服务器，它管理数据库和表、控制用户访问和处理 SQL 查询。
- **MySQL-client** MySQL 客户程序，它使连接服务器和与服务器相互作用成为可能。
- **MySQL-devel** 当编译其他使用 MySQL 的程序时使用的库和头文件。
- **MySQL-shared** MySQL 客户共享库。
- **MySQL-bench** MySQL 数据库服务器基准和性能测试工具。

不同的压缩格式

MySQL 软件发布通常以 zip，tar（tape archive，磁带档案文件）或 RPM（RPM 封装管理器）格式压缩，在压缩格式下它们的大小范围从 7 MB 到 20 MB，解压缩后能达到 100 MB。根据选择的格式，我们需要适当的解压缩工具把文件从源档案文件中释放出来。对 zip 和 tar 文件，我们需要 GNU tar 和 GNU gunzip，从 http://www.gnu.org 可以得到。

对 RPM 文件，我们需要 rpm，从 http://www.rpm.org 可以得到。

另外，如果想从一个源代码版本来编译和安装 MySQL，我们需要一个像 UNIX 上的 gcc 或 Windows 上的 Visual C++ 那样的 C++ 编译器。gcc 编译器从 http://www.gnu.org/software/gcc 可以获得，而 C++ 编译器从 http://msdn.microsoft.com/visualc 可以得到。

在这里列举的 MySQL RPM 文件都是建立在一个 SuSE Linux 系统上的，但是通常它们也可以毫无困难地在其他 Linux 变种上继续使用。

安装 RPM 格式的 MySQL 非常简单，只要对想要安装的每个 RPM 运行一个单独命令（*rpm* 命令）即可。下面是要做的工作。

1. 首先保证以 root 用户登录：

```
[user@host]# su - root
```

2. 切换到包含 RPM 文件的目录：

```
[root@host]# cd /tmp
```

3. 通过执行下面的命令安装 MySQL 数据库服务器（记住用我们的 RPM 文件名替换用斜体字书写的文件名）：

```
[root@host]# rpm -i MySQL-4.0.9-0.i386.rpm
```

RPM 执行下面的任务，启动 MySQL，使它在我们的系统上运行：

- 把 MySQL 二进制代码复制到系统适当的位置上（通常二进制代码存储到 /usr/bin 和 /usr/sbin，而数据库和表存储到 /var/lib/mysql）。
- 为系统添加一个 mysql 用户和组来处理所有与 MySQL 相关的操作和管理任务。
- 修改 MySQL 二进制代码的所有权，以便它们由 mysql 用户和组所拥有。
- 创建和初始化 MySQL 的授权表。
- 为系统启动脚本增加适当条目，这样 MySQL 服务器可以在引导装入程序时自动启动。
- 启动服务器，就能立即使用 MySQL。

图 3.1 显示了在安装过程中可能见到的情况。

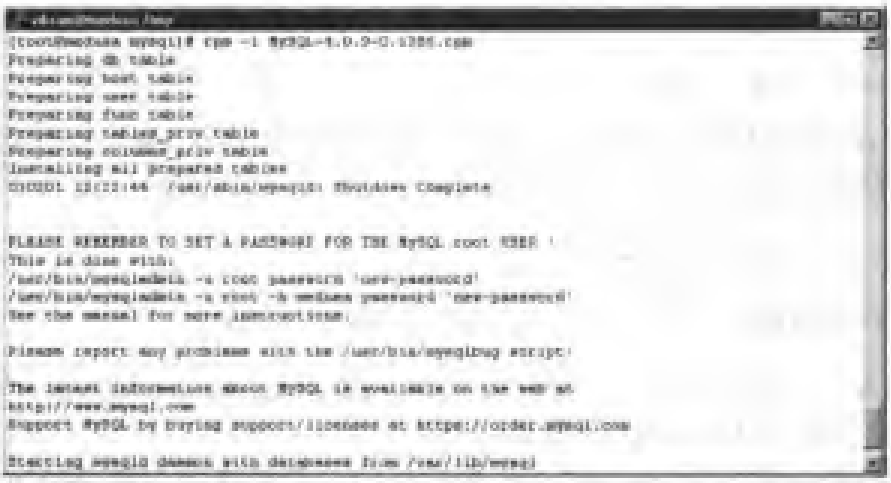

图 3.1　通过 RPM 安装 MySQL 服务器

4. 现在以类似的方法安装剩余的 RPM 文件：

```
[root@host]# rpm -i MySQL-client-4.0.9-0.i386.rpm
[root@host]# rpm -i MySQL-devel-4.0.9-0.i386.rpm
[root@host]# rpm -i MySQL-shared-4.0.9-0.i386.rpm
[root@host]# rpm -i MySQL-bench-4.0.9-0.i386.rpm
```

图 3.2 是当执行这些任务时应该看到的内容。

请注意，只有服务器的安装是必须的，但是我们建议也安装客户机，这样可以从系统控制台与服务器相互作用。只有想要测试 MySQL 时，才需要安装基准实用程序，而当我们编译其他使用 MySQL 的公用程序或者工具时（例如 PHP 脚本语言），库和头文件才派得上用场。

图 3.2 通过 RPM 安装 MySQL 的附属工具和实用程序

安装过程顺利完成以后，就可以跳到 3.3 节，检验 MySQL 各部分是否正常工作。

从二进制 tarball 格式安装 MySQL

为了防止我们正在使用一个不支持 RPM 的 Linux 版本，还可以从 MySQL 网站上安装一个使用二进制 tarball 格式的 MySQL。

从一个二进制版本进行安装，本质上意味着需要手动地执行安装步骤，而不是让 RPM 自动监控它。下面是要完成的步骤。

1. 保证以 root 用户登录：

```
[ user@host] # su - root
```

2. 把 tarball 的内容保存到系统的适当目录（假设位置是 /usr/local/）。记住要用 tarball 文件名替换用斜体字书写的文件名称：

```
[ root@host] # cd /usr/local
[ root@host] # tar -xzvf
mysql-standard-4.0.9-gamma-pc-linux-i686.tar.gz
```

MySQL 文件应该保存到一个以格式 mysql-*version-os-architecture* 命名的目录中，例如 mysql-standard-4.0.9-gamma-pc-linux-i686。

3. 注意到在前一步中创建的目录有一个过长而且繁琐的目录名，类似mysql-standard-4.0.9-gamma-pc-linux-i686。为了便于使用，可在同一位置创建一个名为 mysql 的软连接：

```
[ root@host] # ln -s mysql-standard-4.0.9-gamma-pc-linux-i686 mysql
```

4. 进入这个目录，查看文件是怎样被安排的。我们应该看到类似于图 3.3 那样的情形（可参阅下一页的补充资料，获得更多的关于每个目录所包含内容的信息）。
5. MySQL 数据库服务器以系统root用户或者以系统上的其他用户运行。从安全角度来看，以 root 用户运行 MySQL 数据库服务器是一件很糟糕的事情。基于这个原因，创建一个特定的 mysql 用户和组，对我们来说就变得很有必要了。

图 3.3　在 Linux 上解压缩一个 MySQL 二进制 tarball 文件获得的目录结构

进入树结构

如果我们有时间和兴趣，会发现研究MySQL的目录结构是很有意思的，这有助于我们更好地理解重要文件在哪个位置。

对一个二进制代码版本，典型的 MySQL 安装目录结构如下所示：

```
<mysql-install-root>
|-- bin            （客户机和服务器二进制文件）
|-- data           （数据库和错误日志）
|-- include        （头文件）
|-- lib            （编译库）
|-- man            （手册）
|-- mysql-test     （测试程序）
|-- share          （不同语言的错误信息）
|-- scripts        （启动、关闭和初始化脚本）
|-- sql-bench      （基准测试的查询和数据文件）
|-- support-files  （配置文件模板）
|-- tests          （测试用例）
```

对一个源代码版本，典型 MySQL 安装的目录结构如下所示：

```
<mysql-install-root>
|-- bin            （客户机二进制文件）
|-- libexec        （服务器二进制文件）
|-- var            （数据库和错误日志）
|-- lib            （编译库）
|-- include        （头文件）
|-- info           （信息页）
|-- man            （手册）
|-- mysql-test     （测试程序）
|-- share          （不同语言的错误信息）
|-- sql-bench      （基准测试的查询和数据文件）
```

MySQL 发布版本包含的文档有这个目录结构的更详细的论述。

我们可以使用 groupadd 和 useradd 命令来添加用户和组：

```
[ root@host] # groupadd mysql
[ root@host] # useradd -g mysql mysql
```

6. 运行这个程序包含的初始化脚本 mysql_install_db：

```
[ root@host ] # /usr/local/mysql/scripts/mysql_install_db
```

图 3.4 演示了当我们这样做时会看到的情形。

图 3.4　运行 MySQL 的初始化脚本的输出结果

从图中的输出可以看到，这个初始化脚本准备并且安装了各种 MySQL 基本表，而且为 MySQL 设置了默认访问许可。

7. 修改 MySQL 二进制代码的所有权，使它们由 root 用户拥有：

```
[ root@host ] # chown -R root /usr/local/mysql
```

8. 现在保证新创建的 mysql 用户可以对 MySQL 数据目录进行读 / 写访问：

```
[ root@host ] # chown -R mysql /usr/local/mysql/data
[ root@host ] # chgrp -R mysql /usr/local/mysql
```

9. 通过手工运行 mysqld 守护程序启动 MySQL 服务器：

```
[ root@host ] # /usr/local/mysql/bin/mysqld_safe -user=mysql &
```

MySQL 应该读取在 /usr/local/mysql/data 中创建的基本表，正常启动。

安装过程顺利完成以后，就可以跳到 3.3 节检验服务器是否正常工作了。

从源代码版本安装 MySQL

如果计划使用一个源代码版本安装 MySQL，就需要解开源代码的目录结构，并且要经历传统的 configure-make-make install 过程来配置 MySQL 并且运行它，这是一个相当费时而又复杂的过程。事实上，初学者实在不应该尝试它。如果我们下决心这样使用它，操作步骤如下。

1. 保证以 root 用户登录：

```
[ user@host ] # su - root
```

2. 切换到包含源代码tarball的目录，在目录里抽取文件（请注意，大约需要80 MB的空闲空间来存放源代码的目录结构）。

```
[ root@host] # cd /tmp
[ root@host] # tar -xzvf mysql-4.0.9-gamma.tar.gz
```

请记住用源代码 tarball 文件名替换用斜体字书写的文件名称。

3. 进入包含源代码的目录：

```
[ root@host] # cd mysql-4.0.9-gamma
```

并且注意 ls 命令的内容：

```
[ root@host] # ls -l
```

我们将看到如图 3.5 显示的情形。

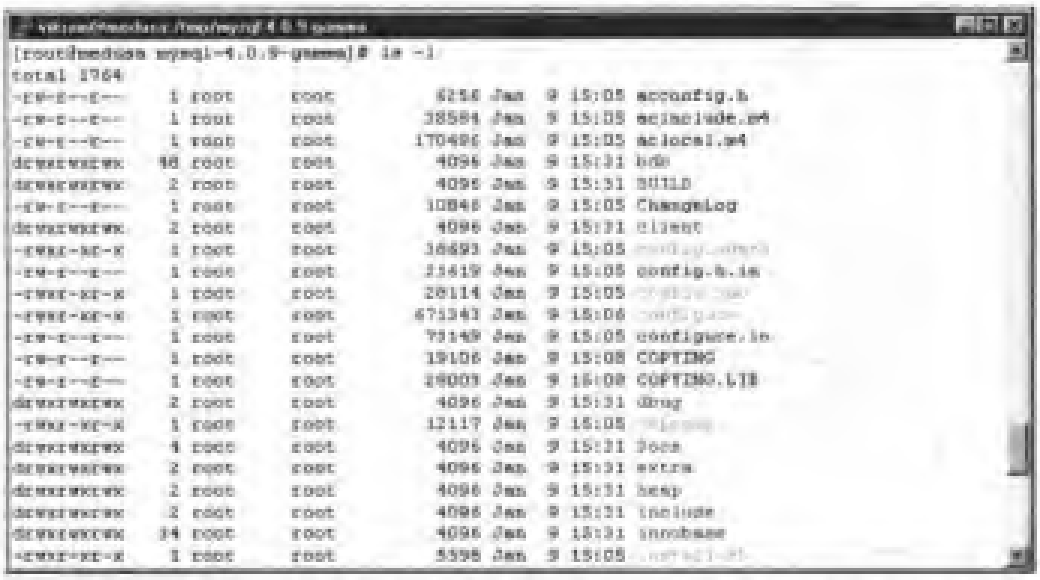

图 3.5　在 Linux 上解压缩一个 MySQL 源代码 tarball 文件获得的目录结构

可参阅第 25 页的补充资料，获得更多关于每个目录包含内容的信息。

4. 现在，通过所包含的 configure 脚本对编译过程设置变量（注意 configure 的参数 --prefix 的用法，它为编译的二进制设置默认的安装路径）。

```
[ root@host] # ./configure --prefix=/usr/local/mysql
```

可以看到当configure对编译过程所需要的变量进行配置和设置时，会出现几个屏幕的输出数据（图 3.6 是一个例子）。

5. 现在，使用 make 编译程序：

```
[ root@host] # make
```

将会看到屏幕上充满了各种奇怪符号和字符（见图 3.7）。

编译过程会花费相当长的时间（参阅第 28 页的补充资料，看一下经验测试，我们要等多久），所以现在可以喝杯咖啡或者查看一下电子邮件了。

既然都准备好了，就可以测试一下以保证一切正常运行。

图 3.6 在 Linux 上配置 MySQL 源代码目录结构

图 3.7 在 Linux 上建立 MySQL

注视时钟

编译 MySQL 是一个非常费时的过程，所以应该准备把 15 分钟到 60 分钟的时间花费在编译任务上。下面的表包含了一些在不同的硬件配置上编译程序所花费时间的试验性观测结果。

系统配置	编译时间（分钟）
Pentium-II@350 MHz, 64 MB RAM	45
Pentium-III@700 MHz, 256 MB RAM	30
AMD Athlon MP 1500+, SuSE Linux 7.3	8
AMD Opteron@2x1.6 GHz, UnitedLinux 1.0	7
Apple PowerMac G4@2x1.2 GHz, Mac OS X 10.2.4	14
Compaq AlphaServer DS20@500 MHz, SuSe Linux 7.0	17
HP 9000/800/A500-7X, HP-UX 11.11	14
IBM RS/6000, AIX 4.3.3	35
Intel Itanium2@900 MHz, Red Hat AS 2.1	14
MIPS R5000@500 MHz, SGI IRIX 6.5	150

6. 运行下面的命令：

```
[root@host]# make tests
```

手工建立

我们可以不去配置许多对建立过程有影响的命令行选项。这里列出了比较有意思的命令选项：

- --prefix　为安装路径设置前缀。
- --without-server　取消服务器的编译，只编译 MySQL 客户程序和库。
- --localstatedir　设置 MySQL 数据库将要被存储的位置。
- --with-charset　设置默认的字符集。
- --with-debug　打开扩展的调试。
- --with-raid　激活 RAID 支持。
- --with-embedded-server　建立 libmysqld 嵌入式服务器库。
- --without-query-cache　取消查询高速缓冲存储器。
- --without-debug　取消调试例行程序。
- --with-openssl　包含 OpenSSl 支持。

使用 configure --help 命令，可得到一个完整的选项列表。

7. 把 MySQL 二进制安装到它们的新位置 /usr/local/mysql：

```
[root@host]# make install
```

图 3.8 演示在安装过程中屏幕上显示的内容。

图 3.8　在 Linux 上安装已编译的 MySQL 二进制文件

8. 使用 groupadd 和 useradd 命令创建专有 mysql 用户和组：

```
[root@host]# groupadd mysql
[root@host]# useradd -g mysql mysql
```

9. 运行程序中包含的初始化脚本 mysql_install_db，为 MySQL 运行做准备：

```
[root@host]# /usr/local/mysql/scripts/mysql_install_db
```

10. 修改 MySQL 二进制的所有权，使它由 root 用户所拥有：

```
[root@host]# chown -R root /usr/local/mysql
```

现在保证新创建的 mysql 用户对 MySQL 数据目录有读 / 写访问的权力：

```
[root@host]# chown -R mysql /usr/local/mysql/var
[root@host]# chgrp -R mysql /usr/local/mysql
```

11. 通过手工运行 mysqld 守护程序启动 MySQL 服务器：

```
[root@host]# /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

MySQL 读取在 /usr/ local/mysql/var 创建的基本表，正常启动。

到此为止，我们可以继续学习 3.3 节的内容来检验一切是否正常了。

3.2.2 在 Windows 上安装和配置 MySQL

MySQL源代码和二进制两种形式对Windows 95/98/Me/2000/XP/NT都是可用的。通常情况下，我们希望使用二进制代码版本，它可以自动安装，允许我们在Windows系统上启动MySQL并且在几分钟以后就可以运行它。然而，如果我们喜欢自己启动系统，MySQL AB 也可使MySQL 源代码用于 Windows。在本节中，我们将研究在 Windows 98 和 Windows NT 上都可以使用的源代码版本和二进制代码版本的安装。

版本控制

为了防止混淆，本书后面内容使用的所有二进制代码都是根据下面的软件版本在Linux上建立的：

- mysqld 4.0.15-standard
- mysqld-4.1-alpha
- rpm 4.1
- gcc 3.2
- tar 1.13.25
- gunzip 1.3.3
- unzip 5.50
- make 3.79.1
- autoconf 2.53
- automake 1.6.3

从二进制版本安装 MySQL

在Windows上安装二进制版本是一个非常简单的过程，我们需要做的所有事情就是利用安装程序提供的安装向导点击我们的选择。步骤如下：

1. 以管理员身份登录（如果使用的是 Windows NT 或者 Windows 2000），并且解压缩发布档案文件，把它保存到系统上的一个临时目录中。保存以后，我们的目录看上去与图 3.9 显示的类似。

图 3.9 对 Windows 解压缩 MySQL 二进制版本创建的目录结构

2. 双击 setup.exe 文件开始安装过程。会看到一个欢迎界面（见图 3.10）。点击“Next”按钮。

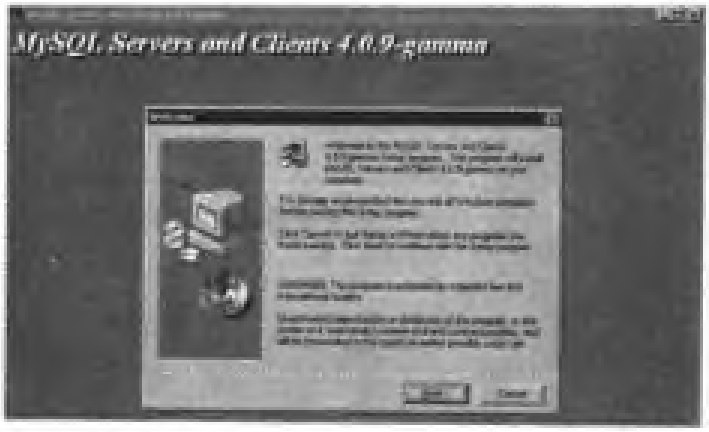

图 3.10　在 Windows 上安装 MySQL

3. 点击“Browse”按钮选择要安装 MySQL 的目录。在本例中，选择 c:\program files\mysql（见图 3.11）。点击“Next”按钮。

图 3.11　在 Windows 上选择安装目录

4. 选择要求的安装类型（见图 3.12）。点击“Next”按钮。

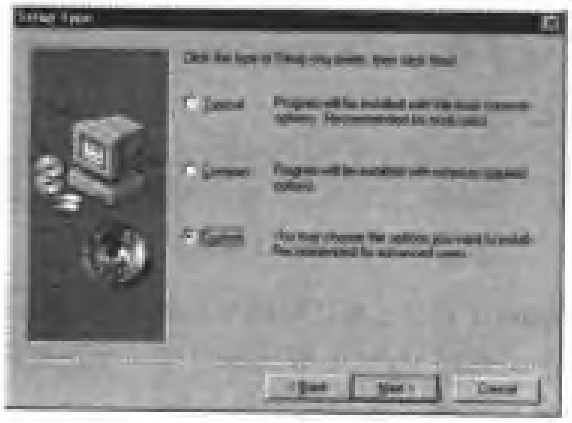

图 3.12　选择安装类型

通常选择典型安装，然而，如果想改变默认设置，则可以选择自定义安装，接着点击“Next”按钮，然后决定安装哪些组件（见图 3.13）。

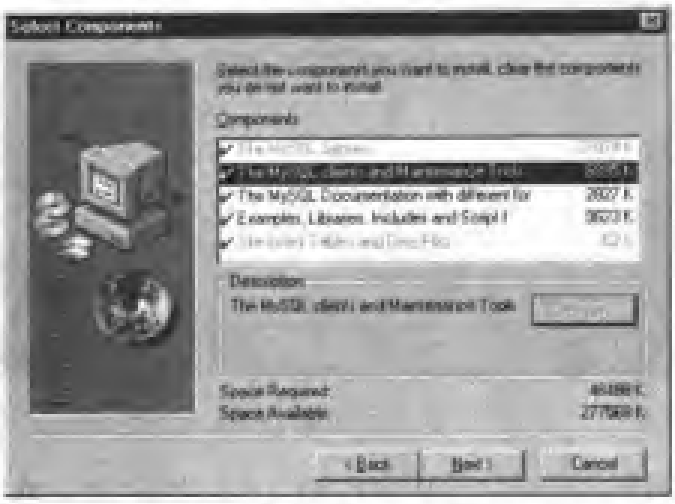

图 3.13　在 Windows 上选择自定义安装组件

5. 点击“Next”按钮，MySQL 现在开始安装到系统上（见图 3.14）。

图 3.14　Windows 上正在进行中的安装

6. 安装完成以后，应该看到如图 3.15 所示的屏幕。点击“Finish”按钮即可完成安装。

图 3.15　Windows 上安装顺利完成

现在进入安装 MySQL 的 bin 子目录，并且启动 WinMySQLadmin 工具（winmys-qladmin.exe）来启动 MySQL 服务器。这个工具对 MySQL 配置提供了一个图形用户界面（GUI），并且它是在 Windows 系统上配置 MySQL 迄今为止最简单的方法。

首次启动 WinMySQLadmin 时，系统会询问服务器运行时的用户名和密码（见图 3.16）。

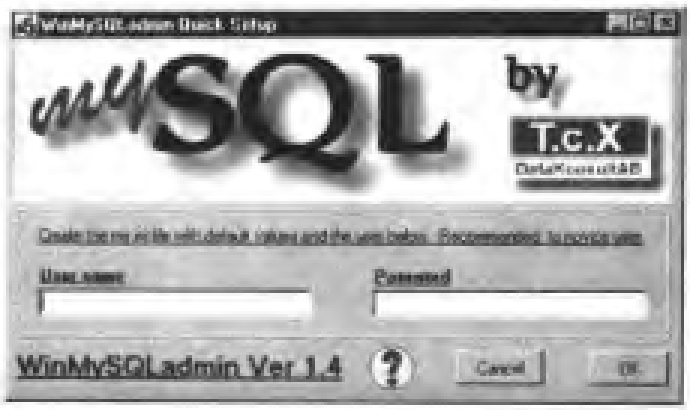

图 3.16　通过 WinMySQLadmin 在 Windows 上配置 MySQL 用户

输入这些信息以后，WinMySQLadmin 将自动创建 MySQL 配置文件（名称为 my.ini），并且用系统使用的适当值来配置它。可以通过主 WinMySQLadmin 应用窗口的“my.ini Setup”选项卡在任何时候编辑这些值（见图 3.17）。

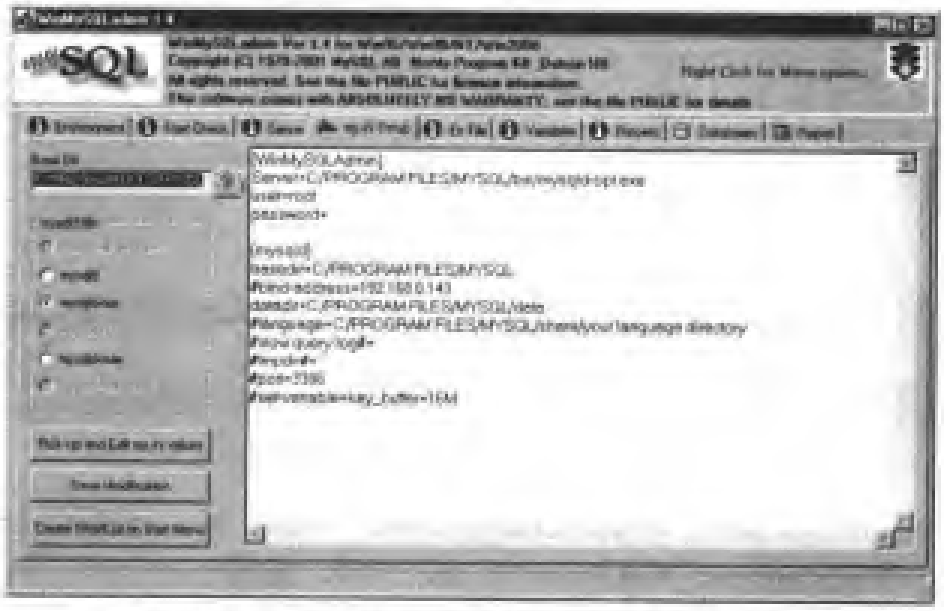

图 3.17　通过 WinMySQLadmin 在 Windows 上编辑 MySQL 配置

注释：还可以通过在 MySQL 安装的子目录 bin 中直接启动 mysqld.exe，或者用 mysqld-nt.exe 二进制文件来启动 MySQL 服务器。

一旦启动了服务器，WinMySQLadmin 会在 Windows 任务栏的右侧区域最小化为一个绿色图标。现在可以按 3.3 节描述的方法继续测试服务器，以保证一切工作正常。

请注意，可以通过右击任务栏图标，从弹出菜单（见图 3.18）中选择“Show me”命令，随时使 WinMySQLadmin 应用程序回到前台。

图 3.18　使用 WinMySQLadmin 系统托盘图标

从源代码版本安装 MySQL

从源代码文件编译 Windows 版本的 MySQL 的方法通常不推荐使用（使用提供的二进制版本非常安全，更不用说简便了），但是假设我们有一个 Visual C++ 6.0 编译器（带有 Service Pack 5 和预处理程序包）副本，这当然也是可行的。操作步骤如下：

1. 把源代码文件解压缩到系统的一个工作目录。
2. 启动 Visual C++ 编译器，从工作目录打开 mysql.dsw 工作区，会看到如图 3.19 所示的一个窗口。

图 3.19　在 Visual C++ 中的 MySQL 工作区

3. 选择“Build | Set Active Project Configuration”得到一个可以使用的配置列表。选择“mysqld - Win32 Release”（见图 3.20）。点击“OK”按钮。

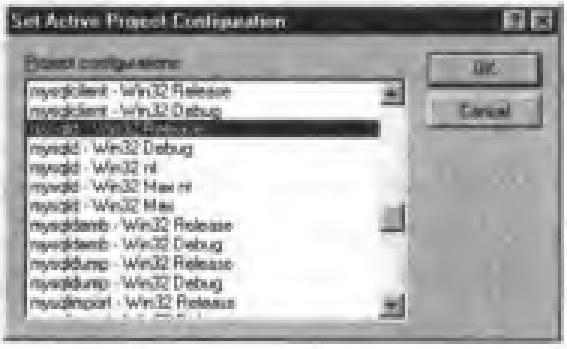

图 3.20　在 Visual C++ 中选择 MySQL 版本

4. 按 F7 键开始编译。各种 MySQL 二进制都将被编译。根据机器的性能，预计这个过程要花费 20 分钟到 1 小时的时间。在编译期间，Visual C++ 编译器窗口将显示一系列消息，比如在图 3.21 中显示的这些消息。

图 3.21　在 Visual C++ 中为 Windows 编译 MySQL

5. 编译完成以后，要创建一个单独的安装目录保存编译的二进制文件，例如 c:\program files\mysql。
6. 在这个目录下创建一个 bin 子目录，并且把编译的库和可执行文件移到这个目录。同时，也把 data，share，docs 和 support-files 目录从工作目录移到这个目录。
7. 现在，我们可以通过进入 MySQL 安装的 bin 子目录，直接启动 MySQL 服务器守护程序（mysqld.exe 或者 mysqld-nt.exe）来启动 MySQL 服务器。

服务器开始启动以后，应继续按照下面介绍的方法测试它。

3.3　测试 MySQL

MySQL 成功安装以后，基本表已经被初始化，服务器也已经启动了，我们就可以通过某些简单的测试来检测一下是否运行正常。

请注意，所有这些命令应该在 UNIX 和 Windows 命令提示符下运行。在这里我们假定从 MySQL 安装目录运行它们（按照 3.2 节的说明，安装目录或者是 UNIX 中的 /usr/local/mysql，或者是 Windows 中的 c:\program files\ mysql）。

3.3.1　使用 mysqladmin 实用程序获得服务器状态

mysqladmin 实用程序通常位于 MySQL 安装的 bin 子目录下。我们可以转到这个目录并执行下面的命令来运行它：

```
[ root@host] # mysqladmin version
```

应该看到与图 3.22 的显示相类似的输出结果。

图 3.22 调用 mysqladmin 的输出结果

3.3.2 用 MySQL 客户连接服务器

MySQL 版本包含的 MySQL 客户非常有意思地被命名为 mysql。通过切换到 MySQL 安装的 bin 目录下在提示符输入：

```
[ root@host] # mysql
```

我们将会得到一个 mysql > 提示符。

现在，我们已经连接到 MySQL 服务器，并且可以执行 SQL 命令了。下面是几个命令的实例以及它们的输出：

```
mysql> SHOW DATABASES;
+-----------+
| Database  |
+-----------+
| mysql     |
| test      |
+-----------+
2 rows in set (0.13 sec)

mysql> USE mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed

mysql> SHOW TABLES;
+------------------+
| Tables_in_mysql  |
+------------------+
| columns_priv     |
| db               |
```

```
| func            |
| host            |
| tables_priv     |
| user            |
+-----------------+
6 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM user;
+----------+
| count(*) |
+----------+
|        4 |
+----------+
1 row in set (0.00 sec)
```

3.4　安装之后的步骤

完成测试以后，还要执行两个任务才能完成 MySQL 的安装。

3.4.1　修改 MySQL root 用户口令

当MySQL第一次被安装时，MySQL的管理员或 root用户，对数据库服务器进行访问是受限制的。默认情况下，使用一个空口令初始化这个用户，通常这是一件很糟糕的事情。因此应该通过包含的mysqladmin实用程序，在UNIX中使用下面的语句来设置这个用户口令，从而尽可能快地纠正这种情况。

```
[root@host]# /usr/local/mysql/bin/mysqladmin -u root
password '<new-password>'
```

Windows 下的修改方法如下：

```
C:\> c:\program files\mysql\bin\mysqladmin -u root
password '<new-password>'
```

这个密码修改会立即生效，不需要重新启动服务器或者刷新权限表。

注释：在 UNIX 上，MySQL root 用户不同于系统 root 用户。

3.4.2　配置 MySQL，当系统引导时自动启动

在 UNIX 上，MySQL 带有启动 / 关闭脚本，我们推荐使用这种方法来启动和关闭 MySQL 数据库服务器。这个脚本名称为 mysql.server，可以在 MySQL 安装的 support-files 子目录中使用，并且可以用下面的命令来启动或关闭它：

```
[root@host]# /usr/local/mysql/support-files/mysql.server start
[root@host]# /usr/local/mysql/support-files/mysql.server stop
```

要让 MySQL 在引导装入程序时自动启动，只要把第一个脚本复制到系统的 /etc/init.d/* 目录层，然后在系统引导和关闭脚本中使用适当的参数调用它即可。

在Windows上设置自动启动MySQL，只要向启动组添加到一个mysqld服务器的连接即可。要想了解更多的信息，请参考第 13 章。

朋友的一点帮助

如果在启动MySQL服务器时出现了问题，我们可以通过阅读MySQL出错日志获得非常详细的关于故障的信息。通常可以在MySQL安装的子目录var里找到这个日志文件，它被命名为hostname.err。我们也可以通过进一步学习这个日志来发现和解决其他常见的问题，比如遗忘了超级用户口令或者不正确的路径设置。还可以访问下面的资源，获得关于如何解决在安装过程中可能遇到的问题的建议：

- **MySQL 手册** http://www.mysql.com/documentation
- **MySQL 邮件列表** http://lists.mysql.com
- **Google** http://www.google.com
- **Google 组** http://groups.google.com

如果我们正在报告一个问题或故障，请记住要使用提供的mysqlbug脚本搜集必要的系统信息并且把它记录在报告中。

3.5 小结

作为一个开放源代码应用程序，MySQL对许多平台和体系结构都可以以二进制代码和源代码两种形式使用。本章解释了MySQL不同版本之间的差别，并且根据需要对最适合的版本提出了建议。本节还在Linux和Windows这两个最通用平台上演示了MySQL的安装过程。本章为二进制代码和源代码两种版本提供了安装和配置指令，还为其他平台以及详细的故障修复的建议和帮助提供了指南性的网络在线资源。

第二部分

使　　用

第4章　SQL基础
第5章　MySQL数据类型
第6章　MySQL运算符
第7章　MySQL函数
第8章　使用数据库和表
第9章　使用数据
第10章　连接
第11章　子查询
第12章　事务处理

第4章 SQL基础

回想一下，我们就会意识到对数据库可以执行两种基本任务：输入数据和取出数据。并且在通常情况下，完成这两个功能使用的主要工具是广为人知的SQL，即结构化查询语言。

作为一个符合标准的关系数据库管理系统（RDBMS），MySQL可以非常好地识别SQL，并且甚至对SQL标准提供了某些引人关注的扩展。要有效地使用MySQL，必须能够熟练地操作SQL——这是我们与数据库服务器打交道的主要方式，并且它对帮助我们迅速有效地获得所需要的数据起到了非常重要的作用。

在本章中，将要讲解创建和向数据库输入数据的一些基本SQL命令，并且会结合实例使这些内容更加清晰易懂。如果从来没有使用过数据库，或者有害怕学习另外一种语言的想法，在这里都不用担心，因为SQL比大多数编程语言都要简单得多，我们将会非常容易地掌握它。

4.1 SQL简史

在进入SQL命令语法的叙述之前，先让我们花一点时间了解SQL的由来。

SQL开始于SEQUEL1[①]，后者即结构化英语查询语言，是IBM公司研究项目System / R的一部分。System / R是关系数据库管理系统的最初原型，1974年它诞生于IBM的San Jose实验室，SEQUEL是第一个支持多表和多用户查询的语言。

在20世纪70年代后期，SQL作为Oracle RDBMS的查询语言以商业角色问世。接下来，Ingres RDBMS也开始使用SQL。到20世纪80年代，SQL成为飞速发展的RDBMS产业的实际标准。1989年SQL成为一个通常被称为SQL-89的ANSI标准，后来在1992年得到更新，变成了SQL-92或者SQL 2，是今天大多数商业RDBMS（包括MySQL）使用的标准。

违反规则

虽然现在大多数商业RDBMS支持SQL-92标准，但是它们其中很多对规则过于随便，用所有权扩展和增加来扩充SQL（MySQL就是这样的一个例子）。通常这些增加的特性用于改进性能或者对系统添加额外的功能，可是当从一个DBMS到另一个DBMS进行移植时，它们将会导致很多问题。

MySQL背离SQL规范的完整列表可以在http://www.mysql.com/doc/en/Compatibility.html上看到。

4.2 SQL综述

作为一种语言，SQL对人类很友好，它的大部分命令与英语口语相似，所以阅读、理解和

---

① 由于法律的原因，后来名字改成了SQL。

学习它都非常容易。它的命令明确地表示为语句，并且每一个语句以“作用字”作为开始。下面的实例演示了这一点：

```
CREATE DATABASE toys;
USE toys;
SELECT id FROM toys WHERE targetAge > 3;
DELETE FROM catalog WHERE productionStatus = "Revoked";
```

我们可以看到，弄懂每个语句的作用是非常容易的事情。这个简单的特性是SQL广受欢迎并且容易学习的原因之一。

SQL 语句可以划分为三个主要的类别，每一类都与数据库管理的一个不同方面相关联：

- **用于定义数据库结构的语句**　这些语句定义了不同的数据段、数据库、表和列类型以及数据库索引之间的关系。在 SQL 规范中，称这个组件为数据定义语言（DDL），本书将在第 8 章详细讨论它。
- **用于操纵数据的语句**　这些语句控制添加和删除记录、查询和连接表、检验数据完整性。在 SQL 规范中，称这个组件为数据操纵语言（DML），本书将在第 9 章详细讨论它。
- **用于控制不同数据段的许可和访问级别的语句**　这些语句定义了数据库、表和字段的访问级别和安全权限，我们将在每个用户和每个主机基础上指定它们。在 SQL 说明书中，称这个组件为数据控制语言（DCL），本书将在第 14 章详细讨论它。

典型地，每个 SQL 语句都以分号结束，并且 SQL 处理器忽略空格、制表符和回车符。下面的两个语句意义是相同的，虽然第一个语句在一个单行上，第二个语句拆分在多个行上：

```
DELETE FROM catalog WHERE productionStatus = "Revoked";

DELETE FROM
         catalog
    WHERE productionStatus =

"Revoked";
```

4.3　(My)SQL 教程

作为不常用的基础语言，我们可以阅读一个快速教程，快速地得到更多的 SQL 基础知识。在接下来的章节中，我们将设计一组关系表，并且将创建一个数据库来存储这些表。最后在 MySQL 中重新创建数据库表，把记录插入数据库，查询数据库系统，从而得到几个亟待解决的问题的答案。

在阅读本书的同时，我们要试验跟在后面的实例。这个过程不仅使我们洞察了 MySQL 是怎样工作的，而且还提前教给我们 SQL 的基础知识，为后面的几章做准备。

4.3.1　理解 RDBMS

我们从最基础的知识讲起。每个数据库由一个或者多个表组成，这些表在行和列中构造数据，使数据变得有组织。

下面是一个具有代表性的表的实例：

member_id	fname	lname	tel	email
1	John	Doe	1234567	jdoe@somewhere.com
2	Jane	Doe	8373728	jane@site.com
3	Steve	Klingon	7449373	steve@alien-race.com
4	Santa	Claus	9999999	santa@the-north-pole.com

可以看到，一个表把数据划分为行，在每一行中有一个条目（或者记录）。如果翻到第1章中类似填充档案柜的最初数据库，会看到档案柜中的每一个文件对应于表中的一行。

每行中的数据进一步分解为单元（或者字段），每个单元包含数据的一个特定属性值。例如，如果对用户 Steve Klingon 使用一条记录，这条记录会分解为多个单独的字段：member ID，first name，last name，phone number 和 e-mail 地址。

表中的各行并没有以任何特定的顺序排列，它们可以按字母顺序、ID、成员名或者我们选择指定的其他标准进行排序。因此，在表中识别一个特定记录的方法就变得很有必要了。在这个例子中，我们用 member ID 识别每一条记录，member ID 是对应每一行记录的惟一数字值，这个数字值惟一的字段称为表的主键。

应该注意到MySQL是一个关系数据库管理系统，一个关系数据库是由多个包含相互关联信息的表组成的。SQL允许我们以各种方式从这些表中合并数据，因此允许我们创建和分析在数据中的新关系。

在第一个实例中，使用了一个简单的表，它本身是非常精美的。当我们添加更多的表和联系这些表之间的信息时，才会真正体会到这个系统内在的力量。仔细思考下面的例子，它又添加了两个表，第二个表中包含了一个可以出租的电影列表，而第三个表通过它们的主键把电影和成员连接起来。

member_id	fname	lname	tel	email
1	John	Doe	1234567	jdoe@somewhere.com
2	Jane	Doe	8373728	jane@site.com
3	Steve	Klingon	7449373	steve@alien-race.com
4	Santa	Claus	9999999	santa@the-north-pole.com

video_id	title	director
1	Star Wars: The Phantom Menace	George Lucas
2	ET	Steven Spielberg
3	Charlie's Angels	McG
4	Any Given Sunday	Oliver Stone
5	Hollow Man	Paul Verhoeven
6	Woman On Top	Fina Torres

```
+-----------+----------+
| member_id | video_id |
+-----------+----------+
|         2 |        6 |
|         4 |        2 |
|         1 |        1 |
|         1 |        2 |
|         1 |        3 |
+-----------+----------+
```

如果进一步观察第三个表，会看到它用所租借的录像连接每一个成员。从而我们知道 Jane Doe（成员 #2）已经租借了 Woman On Top（录像 #6），而 John Doe（成员 #1）决定用周末的时间看完 Star Wars（录像 #1）、ET（录像 #2）和 Charlie's Angels（录像 #3）。

换句话说，第三个表在第一个表和第二个表之间建立了一个关系，这是 RDBMS 的一个基本概念。通过建立一个或者更多的关系，使得提取数据的一个子集（一个数据片段）来回答指定的问题成为可能。

4.3.2　创建数据库

如果已经理解了概念，就可以开始讨论实质问题了。启动 MySQL 客户（注意在下面的代码清单中，输入的任何符号以黑体字出现）。

```
[user@host]# mysql -u root -p
Password: ******
```

假设一切都适当地设置好了，而且输入了正确的口令，应该看到如下面所示的一个提示符：

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 80 to server version: 4.0.9-gamma-standard
Type 'help;' or '\h' for help.
mysql>
```

这就是 MySQL 命令提示符，我们要用它输入所有的 SQL 语句。注意 MySQL 命令以一个分号或者 \g 符号结束，并且用大写和小写字母都可以输入。

因为所有的表都存储在数据库中，所以我们需要知道的第一个命令是 CREATE DATABASE 命令，它看上去如下面所示：

```
CREATE DATABASE database-name
```

可以通过创建一个名为 library 的数据库来试验它：

```
mysql> CREATE DATABASE library;
Query OK, 1 row affected (0.05 sec)
```

使用 SHOW DATABASES 命令，会看到所有可用的数据库：

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
```

```
| library  |
| mysql    |
| test     |
+----------+
3 rows in set (0.00 sec)
```

一旦获得了数据库列表，使用USE命令就可以选择想要使用的数据库，USE命令用法如下：

```
USE database-name
```

现在，我们要在刚才创建的数据库上限制我们的行为：

```
mysql> USE library;
Database changed
```

选择了数据库后，使用SHOW TABLES命令可以查看它里面可以使用的表：

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

因为这是一个新的数据库，所以没有表出现。让我们来解决它。

可以在第8章阅读到更多关于处理数据库的信息。

4.3.3 添加表

在数据库中用于创建一个新表的典型SQL命令如下：

```
CREATE TABLE table-name (field-name-1 field-type-1 modifiers,
field-name-2 field-type-2 modifiers, ... , field-name-n
field-type-n modifiers)
```

表的名字中不能包含空格、斜线和句号，除此之外，任何字符都是允许的。每个表（和它包含的数据）都以三个文件组存储在MySQL的数据目录中。

下面是创建前面实例中我们看到的成员表的示例命令：

```
mysql> CREATE TABLE members (member_id int(11) NOT NULL auto_increment,
fname varchar(50) NOT NULL, lname varchar(50) NOT NULL, tel varchar(15),
email varchar(50) NOT NULL, PRIMARY KEY (member_id));
Query OK, 0 rows affected (0.05 sec)
```

请注意，每个字段名后面紧跟一个“类型”，这样确定了这个字段所允许输入的数据的类型，并且可以用一个长度值标志这个字段的最大长度。例如，在第一行中，名称为member_id的字段后面紧跟一个最大长度为11的int类型。MySQL提供了许多不同的数据类型来解决不同的数据需求。在下面的补充资料里，概括了一些比较重要的数据类型。

数据类型

下面是使用MySQL时将会遇到的一些重要数据类型：

- **INT** 允许值在-2 147 483 648到2 147 483 647之间的一个数值类型。
- **DECIMAL** 支持浮点或小数的一个数值类型。

- DOUBLE　支持双精度浮点数的一个数值类型。
- DATE　*YYYY-MM-DD* 格式的一个日期字段。
- TIME　*HH:MM:SS* 格式的一个时间字段。
- DATETIME　*YYYY-MM-DD HH:MM:SS* 格式的一个混合日期/时间类型。
- YEAR　以 *YYYY* 或 *YY* 格式专门用于显示年份在 1901 到 2155 之间的一个字段。
- TIMESTAMP　*YYYYMMDDHHMMSS* 格式的一个时间戳类型。
- CHAR　最大长度不超过 255 个字符的定长字符串类型。
- VARCHAR　最大长度不超过 255 个字符的变长字符串类型。
- TEXT　最大长度为 65 535 个字符的字符串类型。
- BLOB　可变数据的二进制类型。
- ENUM　字符串类型，从一个预先定义的可能值列表中选取一个值。
- SET　字符串类型，从一个预先定义的可能值集合中选取 0 或更多个值。

可以在表上加入额外的约束（修饰符），增加表中的数据的一致性：

- 定义每个字段以后，可以通过放置 NULL 和 NOT NULL 修饰符，指定是否允许字段为空，或者必须使字段包含数据（非空）。
- 可以使用 PRIMARY KEY 修饰符为表指定一个主键，指定作为主键的列名要放在修饰符之后。
- 可以使用 UNIQUE 修饰符指定输入字段的值必须是惟一的（即值不重复）。
- 只适用于数字字段的AUTO_INCREMENT修饰符表明MySQL将要为这个字段自动生成一个数字（通过对前面的值增加 1）。

现在继续使用下面的 SQL 语句创建其他两个表：

```
mysql> CREATE TABLE videos (video_id int(11) NOT NULL auto_increment,
title varchar(255) NOT NULL, director varchar(255) NOT NULL,
PRIMARY KEY (video_id));
Query OK, 0 rows affected (0.05 sec)

mysql> CREATE TABLE status (member_id int(11) NOT NULL,
video_id tinyint(11) NOT NULL);
Query OK, 0 rows affected (0.05 sec)
```

如果在创建表的过程中犯了错误，则可以用 ALTER TABLE 命令来修改表的定义，如下面所示：

```
ALTER TABLE table-name ADD new-field-name new-field-type
```

另外，如果只想修改某个存在的列，则可使用下面的命令：

```
ALTER TABLE table-name MODIFY old-field-name
new-field-type modifiers
```

和创建表一样，也可以用 DROP TABLE 命令来删除表，如下所示：

```
DROP TABLE table-name
```

这里是一个实例：

```
mysql> DROP TABLE members;
Query OK, 0 rows affected (0.00 sec)
```

这样将会立即清除掉指定的表和表中所包含的数据，所以要慎重使用这个命令！

可以在第 8 章中看到更多关于处理表的信息。

4.3.4 添加记录

一旦创建了表，就可以开始把数据输入表了。完成输入的 SQL 命令是 INSERT 命令，INSERT 命令的语法如下：

```
INSERT into table-name (field-name-1, field-name2, field-name-n)
VALUES (value-1, value-2, value-n)
```

下面是一个实例：

```
mysql> INSERT INTO members (member_id, fname, lname, tel, email)
VALUES (NULL, 'John', 'Doe', '1234567', 'jdoe@somewhere.com');
Query OK, 1 row affected (0.06 sec)
```

也可以使用不指定字段名的 INSERT 语句的简短形式：

```
mysql> INSERT INTO members VALUES (NULL, 'John', 'Doe', '1234567',
'jdoe@somewhere.com');
Query OK, 1 row affected (0.06 sec)
```

有意思的是，通过在 INSERT 语句中指明字段名，就有了以我们喜欢的次序插入值的灵活性。因为这种情况的存在，下面的两个语句是等价的：

```
mysql> INSERT INTO members (member_id, fname, lname, tel, email)
VALUES (NULL, 'John', 'Doe', '1234567', 'jdoe@somewhere.com');
Query OK, 1 row affected (0.06 sec)

mysql> INSERT INTO members (fname, lname, email, tel, member_id)
VALUES ('John', 'Doe', 'jdoe@somewhere.com', '1234567', NULL);
Query OK, 1 row affected (0.00 sec)
```

没有指定的字段将被自动设置成它们的默认值。

现在已经知道了怎样插入记录，可以试着为这三个表插入样本记录，我们使用4.3.1节中的示例数据（还可以在本书的相关网站http://www.mysql-tcr.com/上找到建立这些表的SQL命令）。

4.3.5 删除和修改记录

与插入记录一样，也可以使用 DELETE 命令删除记录，如下所示：

```
DELETE FROM table-name
```

例如，命令：

```
mysql> DELETE FROM members;
Query OK, 0 rows affected (0.06 sec)
```

将从 members 表中删除所有的记录。

可以给DELETE语句添加WHERE子句来选择要删除的特定行的子集。下面的例子将只删除 member ID 为 16 的记录：

```
mysql> DELETE FROM members WHERE member_id = 16;
Query OK, 1 row affected (0.06 sec)
```

最后，有一个 UPDATE 命令可以帮助我们改变表中已存在的值，如下所示：

```
UPDATE table-name SET field-name = new-value
```

这个命令将作用于字段field-name中所有的值，把它们全改变为new_value值。如果只希望修改单个字段中的值，可以像使用 DELETE 那样使用 WHERE 子句。

下面的命令用于更新表中 John Doe 的 e-mail 地址：

```
mysql> UPDATE members SET email = 'john@somewhere.com' WHERE member_id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

通过用逗号隔开它们，还可以修改多个字段：

```
mysql> UPDATE members SET email = 'john@somewhere.com',
lname = 'Doe The First' WHERE member_id = 2;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

应注意MySQL是如何把符合查询条件的记录个数和被它改变的行的数量的反馈信息提供给我们的。

4.3.6　执行查询

一旦数据存在于数据库中，就可以处理它了。通过使用许多 SELECT 语句，MySQL 允许我们从数据库中提取特定的数据片段。

SELECT 查询的最简单形式是“全选”查询，它返回指定表中的所有记录。如下所示：

```
mysql> SELECT * FROM members;
+-----------+-------+---------+---------+---------------------------+
| member_id | fname | lname   | tel     | email                     |
+-----------+-------+---------+---------+---------------------------+
|         1 | John  | Doe     | 1234567 | jdoe@somewhere.com        |
|         2 | Jane  | Doe     | 8373728 | jane@site.com             |
|         3 | Steve | Klingon | 7449373 | steve@alien-race.com      |
|         4 | Santa | Claus   | 9999999 | santa@the-north-pole.com  |
+-----------+-------+---------+---------+---------------------------+
4 rows in set (0.00 sec)
```

星号表明我们希望看到在表中出现的所有列。如果只想在结果集中看到一个或者两个指定的列，可以在 SELECT 语句中指定列的名称，如下所示：

```
mysql> SELECT lname FROM members;
+---------+
| lname   |
+---------+
| Doe     |
| Doe     |
| Klingon |
| Claus   |
+---------+
4 rows in set (0.00 sec)
```

在大多数情况下，给希望在结果集中看到的确定的字段命名是更可取的，这样允许应用程序在表中免于结构改变，并且通常效率会比较高，因为 MySQL 只选择它需要的字段。

使用 DISTINCT 关键字可以除去重复的记录，下面的查询将 last name 为“Doe”的成员最多显示一次：

```
mysql> SELECT DISTINCT lname FROM members;
+---------+
| lname   |
+---------+
| Doe     |
| Klingon |
| Claus   |
+---------+
3 rows in set (0.05 sec)
```

当然，把数据构造成行和列的方法，将使获得一个所期望的结果集变得比较容易。主要采用的方法是 WHERE 子句（可以从以前章节中的 UPDATE 和 DELETE 语句中看到这一点）和 SELECT 语句的结果。它们允许我们为结果集定义特定的标准，不符合指定标准的记录不会出现在结果集中。

例如，假定想找出 last name 为 Doe 的所有成员的一个列表，则可使用如下命令：

```
mysql> SELECT * FROM members WHERE lname = "Doe";
+-----------+-------+-------+---------+--------------------+
| member_id | fname | lname | tel     | email              |
+-----------+-------+-------+---------+--------------------+
|         1 | John  | Doe   | 1234567 | jdoe@somewhere.com |
|         2 | Jane  | Doe   | 8373728 | jane@site.com      |
+-----------+-------+-------+---------+--------------------+
2 rows in set (0.00 sec)
```

或者假定想找出 Santa Claus 的 e-mail 地址的一个列表，则可使用如下命令：

```
mysql> SELECT email FROM members WHERE fname = "Santa";
+----------------------------+
| email                      |
+----------------------------+
```

```
| santa@the-north-pole.com | |
+---------------------------+
1 row in set (0.06 sec)
```

或者假定想找出 George Lucas 所导演的所有电影的一个列表，则可使用如下命令：

```
mysql> SELECT title, director FROM videos WHERE director = "George Lucas";
+-------------------------------+--------------+
| title                         | director     |
+-------------------------------+--------------+
| Star Wars: The Phantom Menace | George Lucas |
+-------------------------------+--------------+
1 row in set (0.06 sec)
```

（我们知道这个列表还不完全。也许应该写信给 Santa，要他提供其他的。）

使用比较和逻辑运算符

还可以使用比较和逻辑运算符来更进一步修改 SQL 查询。如果表中包含大量的数值数据，它们迟早会派上用场，比如下面的表。

```
+-------+------+---------+------------+
| name  | math | physics | literature |
+-------+------+---------+------------+
| john  |   68 |      37 |         45 |
| jim   |   96 |      89 |         92 |
| bill  |   65 |      12 |         57 |
| harry |   69 |      25 |         82 |
+-------+------+---------+------------+
```

在 MySQL 中可用的 6 个比较运算符列在表 4.1 中。

表 4.1　MySQL 比较运算符

运算符	含义
=	等于
!=	不等于
>	大于
<	小于
>=	大于 / 等于
<=	小于 / 等于

还可以使用逻辑运算符 AND，OR 和 NOT 来创建更复杂的查询。表 4.2 说明了每一个逻辑运算符的作用。

表 4.2　MySQL 逻辑运算符

运算符	含义
AND	所有指定条件都必须匹配
OR	指定条件中的任一个必须匹配
NOT	对条件取反

现在考虑前面所列的成绩表，如果想要创建一个数学成绩超过90分的所有学生的列表，应该使用如下所示的查询：

```
mysql> SELECT * FROM grades WHERE math > 90;
+------+------+---------+------------+
| name | math | physics | literature |
+------+------+---------+------------+
| jim  |   96 |      89 |         92 |
+------+------+---------+------------+
1 row in set (0.00 sec)
```

假如想确定班里最优秀的孩子（你知道这个人，他总是坐在前排，漂亮地回答每一个问题，并且在牙齿上总是有金属丝），则可用下列查询：

```
mysql> SELECT name FROM grades WHERE math > 85
AND physics > 85 AND literature > 85;
+------+
| name |
+------+
| jim  |
+------+
1 row in set (0.00 sec)
```

如果想要确定至少有一门科目不超过25分的学生，该怎么做呢？

```
mysql> SELECT * FROM grades WHERE math <= 25
OR physics <= 25 OR literature <= 25;
+-------+------+---------+------------+
| name  | math | physics | literature |
+-------+------+---------+------------+
| bill  |   65 |      12 |         57 |
| harry |   69 |      25 |         82 |
+-------+------+---------+------------+
2 rows in set (0.00 sec)
```

最后，还可以在查询中运用基本的算术运算符，以下命令演示了如何将三个成绩合在一起生成一个总的成绩：

```
mysql> SELECT name, math+physics+literature FROM grades;
+-------+------------------------+
| name  | math+physics+literature |
+-------+------------------------+
| john  |                    150 |
| jim   |                    277 |
| bill  |                    134 |
| harry |                    176 |
+-------+------------------------+
4 rows in set (0.05 sec)
```

很明显，这样的操作只有在相同类型的字段上才能执行。

使用内建函数

MySQL 还提供了许多内建函数，当我们试图得到数值总和或者指定字段的平均值时，它们会派上用场。这些函数中的第一个是有用的COUNT()函数，它计算结果集中记录的个数，并且显示这个总数。

下面的命令显示了 videos 表中的记录的总个数：

```
mysql> SELECT COUNT(*) FROM videos;
+----------+
| COUNT(*) |
+----------+
|        6 |
+----------+
1 row in set (0.00 sec)
```

当需要计算一个表中记录的总个数时，这会派上用场。

SUM()函数计算结果集合的和值，而 AVG()函数计算平均值。例如，如果想要计算 math，physics 和 literature 的平均成绩，可以使用一个如下所示的查询：

```
mysql> SELECT AVG(math), AVG(physics), AVG(literature) FROM grades;
+-----------+--------------+----------------+
| AVG(math) | AVG(physics) | AVG(literature) |
+-----------+--------------+----------------+
|   74.5000 |      40.7500 |         69.0000 |
+-----------+--------------+----------------+
1 row in set (0.00 sec)
```

可以用MIN()和MAX()函数找出一个指定字段中的最小值和最大值。下面的查询分别显示了在 math 中的最低成绩和最高成绩：

```
mysql> SELECT MIN(math) FROM grades;
+-----------+
| MIN(math) |
+-----------+
|        65 |
+-----------+
1 row in set (0.00 sec)

mysql> SELECT MAX(math) FROM grades;
+-----------+
| MAX(math) |
+-----------+
|        96 |
+-----------+
1 row in set (0.00 sec)
```

第 7 章中可以看到关于 MySQL 内建函数的更多信息。

排序和限制结果集

如果想按照一个指定字段的排序查看数据，MySQL 提供了 ORDER BY 结构。这个结构允许指定列名和希望数据显示的顺序（升序或降序）。

例如，如果想要查看 members 表中按 ID 顺序排列的数据，应该这样做：

```
mysql> SELECT * FROM members ORDER BY member_id;
+-----------+-------+---------+---------+---------------------------+
| member_id | fname | lname   | tel     | email                     |
+-----------+-------+---------+---------+---------------------------+
|         1 | John  | Doe     | 1234567 | jdoe@somewhere.com        |
|         2 | Jane  | Doe     | 8373728 | jane@site.com             |
|         3 | Steve | Klingon | 7449373 | steve@alien-race.com      |
|         4 | Santa | Claus   | 9999999 | santa@the-north-pole.com  |
+-----------+-------+---------+---------+---------------------------+
4 rows in set (0.06 sec)
```

还可以用附加的 DESC 修饰符倒序排序：

```
mysql> SELECT * FROM members ORDER BY member_id DESC;
+-----------+-------+---------+---------+---------------------------+
| member_id | fname | lname   | tel     | email                     |
+-----------+-------+---------+---------+---------------------------+
|         4 | Santa | Claus   | 9999999 | santa@the-north-pole.com  |
|         3 | Steve | Klingon | 7449373 | steve@alien-race.com      |
|         2 | Jane  | Doe     | 8373728 | jane@site.com             |
|         1 | John  | Doe     | 1234567 | jdoe@somewhere.com        |
+-----------+-------+---------+---------+---------------------------+
4 rows in set (0.00 sec)
```

通过使用LIMIT关键字，可以限制结果集中记录的个数。这个关键字带有两个参数，它们用来指定开始的行的位置和要显示的行的个数。所以查询：

```
mysql> SELECT * FROM videos LIMIT 2,2;
+----------+-----------------+--------------+
| video_id | title           | director     |
+----------+-----------------+--------------+
|        3 | Charlie's Angels | McG          |
|        4 | Any Given Sunday | Oliver Stone |
+----------+-----------------+--------------+
2 rows in set (0.00 sec)
```

将返回结果集中的第 3 行和第 4 行。

可以联合 ORDER BY 和 LIMIT 结构迅速得到表中 4 个最新的记录，举例如下：

```
mysql> SELECT * FROM videos ORDER BY video_id DESC LIMIT 0, 4;
+----------+-----------------+---------------+
| video_id | title           | director      |
+----------+-----------------+---------------+
|        6 | Woman On Top    | Fina Torres   |
|        5 | Hollow Man      | Paul Verhoeven |
|        4 | Any Given Sunday | Oliver Stone  |
|        3 | Charlie's Angels | McG           |
+----------+-----------------+---------------+
4 rows in set (0.00 sec)
```

使用通配符

MySQL还支持LIKE关键字，它用于返回来自通配符搜索的结果，当不能确定搜索数据时它可以派上用场。在一个LIKE结构中允许使用两种类型的通配符：%（百分号）字符，它用于表示零或者更多个字符的具体值；_（下划线）字符，它用于确切地表示一个字符的具体值。

假设我们想要一个 first name 包含字母“e”的所有成员的列表，则查询如下所示：

```
mysql> SELECT * FROM members WHERE fname LIKE '%e%';
+-----------+-------+---------+---------+----------------------+
| member_id | fname | lname   | tel     | email                |
+-----------+-------+---------+---------+----------------------+
|         2 | Jane  | Doe     | 8373728 | jane@site.com        |
|         3 | Steve | Klingon | 7449373 | steve@alien-race.com |
+-----------+-------+---------+---------+----------------------+
2 rows in set (0.16 sec)
```

还可以使用这个技巧把 videos 表搜索一遍，查找名字中包含“man”的电影：

```
mysql> SELECT title, director FROM videos WHERE title LIKE '%man%';
+--------------+----------------+
| title        | director       |
+--------------+----------------+
| Hollow Man   | Paul Verhoeven |
| Woman On Top | Fina Torres    |
+--------------+----------------+
2 rows in set (0.05 sec)
```

可以在第 9 章中看到关于执行查询和处理表数据的更多知识。

关于 LIKE

应该注意到，LIKE %string% 结构通常被认为是一个执行全文搜索的效率不高的亚理想方法，因为在这种情形下，MySQL不能使用关键字查找。这种情况下推荐使用全文搜索和MATCH AGAINST 命令（第 8 章有更多的解释）。

连接表

迄今为止，我们见到的所有查询都集中在一个单表上，但是SQL还允许我们一次查询两个或者更多的表，并且显示一个合并了的结果集。在学术上这称为连接（join），因为它涉及到在特殊的点“连接”不同的表来生成数据的新视图。

当使用连接时，推荐在每列的名字前面附加上它所属的表的名字（迄今为止，我们没有在所见到的任何实例中演示这一点，因为所有的列都是在一个单表上）。例如，可以使用members.fname 来表示在表 members 中名字为 fname 的列，还可以使用 status.video_id 表示在表 status 中的 video_id 列。

下面是一个简单连接的例子：

```
mysql> SELECT member_id, video_id, fname FROM status, members WHERE status.
member_id = members.member_id;
+-----------+----------+-------+
| member_id | video_id | fname |
+-----------+----------+-------+
|         1 |        1 | John  |
|         1 |        2 | John  |
|         1 |        3 | John  |
|         2 |        6 | Jane  |
|         4 |        2 | Santa |
+-----------+----------+-------+
5 rows in set (0.00 sec)
```

此时，status 和 members 表通过共同的列 member_id 连接在一起。

像使用任何 SELECT 语句那样，可以从连接的表中指定我们想看到的列：

```
mysql> SELECT fname, lname, video_id FROM members, status WHERE members.
member_id = status.member_id;
+-------+-------+----------+
| fname | lname | video_id |
+-------+-------+----------+
| Jane  | Doe   |        6 |
| Santa | Claus |        2 |
| John  | Doe   |        1 |
| John  | Doe   |        2 |
| John  | Doe   |        3 |
+-------+-------+----------+
5 rows in set (0.16 sec)
```

还可以把三个表连接到一起。下面的例子使用status表，结合成员信息和录像详细资料，生成了一个复合表，它显示了哪些成员有哪些录像：

```
mysql> SELECT fname, lname, title FROM members, videos, status WHERE status.
member_id = members.member_id AND status.video_id = videos.video_id;
+-------+-------+-------------------------------+
| fname | lname | title                         |
+-------+-------+-------------------------------+
| Jane  | Doe   | Woman On Top                  |
| Santa | Claus | ET                            |
| John  | Doe   | Star Wars: The Phantom Menace |
| John  | Doe   | ET                            |
| John  | Doe   | Charlie's Angels              |
+-------+-------+-------------------------------+
5 rows in set (0.17 sec)
```

可以在第 10~12 章获得关于数据检索和操作的更高级知识，比如连接、子查询和事务等。

注意连接的效率

当连接表时，保证每个连接有一个相关联的、允许使用关键字的约束条件是很重要的。否则，当表增大时性能会以指数级降低。

额外资源

对学习更多的SQL知识感兴趣吗？这里有几个在线和离线资源，有助于我们学习：

- **SQL教程** http://www.w3schools.com/sql/default.asp
- **SQL轻松入门** http://www.w3schools.com/sql/default.asp
- **SQL网上交互资源** http://www.sqlcourse.com
- **SQL.org入门** http://www.sql.org/
- **SQL入门指南（第二版）** Robert Sheldon著
  （ISBN：0072228857），McGraw-Hill\Osborne
- **SQL完全手册（第二版）** James Groff和Paul Weinberg著
  （ISBN: 0072225599），McGraw-Hill\Osborne
- **关系数据库和SQL程序设计入门**
  Christopher Allen、Simon Chatwin和Catherine Creary著
  （ISBN: 0072229241），McGraw-Hill Technology Education

用别名给表命名

如果需要一次又一次地书写很长的表名，显然不是一种好的方法。幸好可以给每个表指派简单的别名。下面的例子分别为表members，status和videos指派了别名m，s和v。

```
mysql> SELECT m.fname, m.lname, v.title FROM members m, status s,
videos v WHERE s.member_id = m.member_id AND s.video_id = v.video_id;
+-------+-------+-----------------------------+
| fname | lname | title                       |
+-------+-------+-----------------------------+
| Jane  | Doe   | Woman On Top                |
| Santa | Claus | ET                          |
| John  | Doe   | Star Wars: The Phantom Menace |
| John  | Doe   | ET                          |
| John  | Doe   | Charlie's Angels            |
+-------+-------+-----------------------------+
5 rows in set (0.00 sec)
```

4.4 小结

通过以上简短的篇幅简要介绍了SQL，它的历史、特性和语法。我们进入了一个快餐式的语言教程，展示了怎样创建数据库和表；如何插入、修改和删除记录；如何执行查询。本章还介绍了如何创建返回表中所有记录的简单查询，然后使用运算符、通配符、连接和内建函数修改这些简单的查询，过滤出我们需要的精确数据。

本章是对MySQL的一个综述，并且作为以后更详细资料的一个初级读本。在以后的几章中，本章的介绍性资料将被深入探讨，并把重点放在MySQL的SQL特别术语上。

第5章　MySQL数据类型

经过第4章简短的SQL教程之后，应该对SQL的功能以及它是怎样适合MySQL数据库所管理和使用的大量情形有了一个清晰的概念。有了较为粗略的介绍性知识之后，现在开始深入地研究MySQL命令集的细节问题。

从前面的一章中已经知道，MySQL表的每一列跟它的主属性列一样都结合一个数据类型。在保证MySQL数据库中数据的完整性和使数据比较容易使用和操作方面，数据类型充当了重要角色。本章将结合实例描述MySQL提供的各种数据类型。

5.1　数据类型的要求和基本原理

在进入数值和字符串数据类型的叙述之前，让我们回顾并且分析一下MySQL（或者其他任何RDBMS）中的数据分类在现实世界中的好处。

RDBMS中数据分类的重要论据是与效率（数据存储）和速度（数据检索）联系在一起的。要理解这一点，只需考虑反面的论据：假设有一个不存在数据类型的系统，所有的数据都作为字符或字符串存储。在这样的系统中，每一个数字（整数、日期和货币单位）都将作为一个字符串存储，占据的磁盘空间等于字符串中字节的个数。因此，数字12 345 678 987 654 321要占用17个字节，而日期06-04-1978将占用10个字节（如果用单独的列存储日期的每个元素，占用的空间会更多）。

更进一步说，表示为字符串的数字或日期不容易执行操作，相加、相减或相乘这个数据将是一个复杂而又笨重的任务，它要求设计者首先把字符串转换成数值类型，然后在它上面执行计算。很明显，在表示为字符串的数字或日期上执行比较或相等测试也将变得非常困难。比较测试13 > 2为真是显而易见的，但是不论我们怀着多大的自信，测试"13" > "2"是否为真是非常困难的。同样，从表示为01-04-2003的2003年4月1日到表示为01-09-2004的2004年9月1日之前的时间间隔是很明显的，但是当这两个日期都表示为字符串并且系统不知道怎样从它们之中分离日、月和年时，完成这个计算是非常困难的。

数据分类解决了所有这些问题。通过给数据的不同类型强加一个一致的模式，使系统能够识别一个特定的数据属于一个特定的类型，因此可以使用这个类型的特性来操作数据。例如，如果01-04-2003标记属于DATE类型，则系统让内置的规则通知它如何处理一个DATE类型，对这个值进行操作（与其他DATE类型相比较，为它添加天或月，并且确定有效值）就变得比较容易了。

强大的数据分类把每个类型与特定的行为联系在一起，执行这些行为可以预防人为错误。最常见的错误是把字符串和数字加在一起。一个弱类型的语言会许可这一点，强类型的语言会立即指出它违反了系统的完整性，并且拒绝承认它。数据分类在不同数据如何相互作用方面排除了不确定性，它可以减少开发者不得不编写的维护系统完整性的代码的数量。

数据分类还可以更有效地利用空间，导致了更小的存储请求，同时提高了性能。例如，如果系统能够把值 12 345 678 987 654 321 识别为一个数字，而不是一个字符串，它将以更有效的形式（一个 8 字节的整数而不是一个 17 字节的字符串）来存储这个数字，因此占用了更少的空间，并且使得在它上面执行的算术运算更加简单。性能效果同样显著：一个 8 字节的整数值处理起来要比一个 17 字节的字符串值快速。考虑节省的空间和提高的速度，我们就会知道为什么使用数据类型有更大的优势。

如同宝剑一样，数据分类也存在利弊。性能的合理使用可以导致更小的数据库和表、有效的索引和更快速地执行查询，而类型的无节制的、过度的使用将导致膨胀的表、存储空间的浪费、低效的索引和在性能上的退化。因此，在实现一个数据库驱动应用程序之前，明智的数据库设计者会全面了解系统中可用的各种数据类型以及它们的局限性和长处。没有数据分类的系统在时间和金钱方面都将是昂贵的。

下面继续探讨用于 MySQL 的各种数据类型。

5.2　MySQL 数据类型

从前面一章中可以知道，表中的每一个字段有一个与它相关联的数据类型。这个数据类型定义了字段中可以保存的数据的种类，并且有助于加强表数据的一致性和完整性。MySQL 支持很多数据类型，它们可以分成下面 4 个主要类别：

- 数值类型
- 字符串类型
- 日期和时间类型
- 复合类型

下面几节详细分析这些分类中的每一种类型。

5.2.1　数值类型

MySQL 的数值数据类型可以大致划分为两个类别，一个是整数，另一个是浮点数或小数。许多不同的子类型对这些类别中的每一个都是可用的，每个子类型支持不同大小的数据，并且 MySQL 允许我们指定数值字段中的值是否有正负之分或者用零填补。

表 5.1 列出了各种数值类型以及它们的允许范围和占用的内存空间。

表 5.1　MySQL 数值类型

类型	大小	范围（有符号）	范围（无符号）	用途
TINYINT	1 字节	(-128, 127)	(0, 255)	小整数值
SMALLINT	2 字节	(-32 768, 32 767)	(0, 65 535)	大整数值
MEDIUMINT	3 字节	(-8 388 608, 8 388 607)	(0, 16 777 215)	大整数值
INT 或 INTEGER	4 字节	(-2 147 483 648, 2 147 483 647)	(0, 4 294 967 295)	大整数值
BIGINT	8 字节	(-9 223 372 036 854 775 808, 9 223 372 036 854 775 807)	(0, 18 446 744 073 709 551 615)	极大整数值

（续表）

类型	大小	范围（有符号）	范围（无符号）	用途
FLOAT	4 字节	(-3.402 823 466 E+38, 1.175 494 351 E-38),0, (1.175 494 351 E-38, 3.402 823 466 E+38)	0,(1.175 494 351 E-38, 3.402 823 466 E+38)	单精度浮点数值
DOUBLE 或 REAL, DOUBLE PRECISION	8 字节	(1.797 693 134 862 315 7 E+308, 2.225 073 858 507 201 4 E-308),0, (2.225 073 858 507 201 4 E-308, 1.797 693 134 862 315 7 E+308)	0,(2.225 073 858 507 201 4 E-308, 1.797 693 134 862 315 7 E+308)	双精度浮点数值
DECIMAL 或 DEC NUMERIC	对 DECIMAL (M, D), 如果 M > D, 为 M+2, 否则为 D+2	依赖于 M 和 D 的值	依赖于 M 和 D 的值	小数值

INT 类型

在 MySQL 中支持的 5 个主要整数类型是 TINYINT，SMALLINT，MEDIUMINT、INT 和 BIGINT。这些类型在很大程度上是相同的，只有它们存储的值的大小是不相同的。仔细考虑下面的例子：

```
mysql>  CREATE TABLE data (fti TINYINT, fsi SMALLINT,
fmi MEDIUMINT, fi INT, fbi BIGINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO data VALUES (123456789, 123456789,
123456789, 123456789, 12345678987654321);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO data VALUES (-123456789, -123456789,
-123456789, -123456789, -12345678987654321);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM data;
 +------+--------+----------+------------+--------------------+
| fti  | fsi    | fmi      | fi         | fbi                |
+------+--------+----------+------------+--------------------+
|  127 |  32767 |  8388607 |  123456789 |  12345678987654321 |
| -128 | -32768 | -8388608 | -123456789 | -12345678987654321 |
+------+--------+----------+------------+--------------------+
2 rows in set (0.00 sec)
```

MySQL以一个可选择的显示宽度指示器的形式对SQL标准进行扩展，这样当从数据库检索一个值时，可以把这个值加长到指定的长度。例如，指定一个字段的类型为INT(6)，就可以保证所包含数字少于 6 个的值从数据库中检索出来时能够自动地用空格填充。如下所示：

```
mysql> CREATE TABLE data ( age INT(7) );
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO data VALUES (19);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT age FROM data;
+-------+
| age   |
+-------+
|    19 |
+-------+
1 row in set (0.06 sec)
```

小 伎 倆

看不到填充的空格？只需使用在下面“数字游戏”补充资料中讨论的ZEROFILL专用修饰符即可。

请注意，使用一个宽度指示器不会影响字段的大小和它可以存储的值的范围。

万一我们试图对一个特别的字段存储一个超出允许范围的数字，MySQL 会根据允许范围最接近它的一端截短它，然后存储截短后的值。仔细思考下面的例子，它演示了试图在一个TINYINT 字段里存储一个非常大的值：

```
mysql> CREATE TABLE data ( id TINYINT );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO data VALUES (123456789);
Query OK, 1 row affected (0.00 sec)

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

数 字 游 戏

对表 5.1 中数字的范围是如何计算的感到惊奇吗？相当简单——仅仅使用公式 $2^{n-1}\times(-1)$ 到 $2^{n-1}-1$ 就可以得到有符号的范围，0 到 2^n-1 得到无符号的范围，其中 n 是按位表示的大小。所以要计算 4 个字节（32 位）的 INT 类型的范围，使用

$$2^{31}\times(-1)\sim 2^{31}-1=-2\,147\,483\,648\sim 2\,147\,483\,647$$

得到有符号的范围，而

$$0\sim 2^{32}-1=0\sim 4\,294\,967\,295$$

得到无符号的范围。

另外，MySQL会将不合规定的值在插入表之前自动改为0。下面的例子演示了试图将一个字符串插入一个定义为INT类型的字段中：

```
mysql> CREATE TABLE data ( age INT );
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO data VALUES ("polly says golly!" );
Query OK, 1 row affected (0.00 sec)

mysql> SELECT age FROM data;
+-------+
| age   |
+-------+
|     0 |
+-------+
1 row in set (0.00 sec)
```

每个整数类型还可以接受一个或者两个下面的可以影响它的显示和存储方式的修饰符：

- UNSIGNED 修饰符规定字段只保存正的值，它可以增大这个字段支持的值的范围。
- ZEROFILL 修饰符规定 0（不是空格）可以用于填补输出值。请注意，使用这个修饰符可防止 MySQL 存储负值。

仔细思考下面的例子，它示范了这些修饰符在实际中怎样起作用：

```
mysql> CREATE TABLE data (fi INT, fiu INT UNSIGNED, fiz INT ZEROFILL,
fiuz INT UNSIGNED ZEROFILL);
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO data VALUES (10, 10, 10, 10),
    -> (-10, -10, -10, -10),
    -> (2147483647, 2147483647, 2147483647, 2147483647),
    -> (3004005006, 3004005006, 3004005006, 3004005006);
Query OK, 4 rows affected (0.06 sec)

mysql> SELECT * FROM data;
+-------------+------------+------------+------------+
| fi          | fiu        | fiz        | fiuz       |
+-------------+------------+------------+------------+
|          10 |         10 | 0000000010 | 0000000010 |
|         -10 |          0 | 0000000000 | 0000000000 |
| 2147483647  | 2147483647 | 2147483647 | 2147483647 |
| 2147483647  | 3004005006 | 3004005006 | 3004005006 |
+-------------+------------+------------+------------+
4 rows in set (0.11 sec)
```

FLOAT，DOUBLE 和 DECIMAL 类型

MySQL 支持的三个浮点类型是 FLOAT，DOUBLE 和 DECIMAL 类型。FLOAT 数值类型用于表示单精度浮点数值，而 DOUBLE 数值类型用于表示双精度浮点数值。

与整数类型一样，这些类型也带有附加参数：一个显示宽度指示器和一个小数点指示器。

例如，语句 FLOAT (5,2)规定显示的值不会超过 5 位数字，小数点后面带有 2 位数字。看下面的示范举例：

```
mysql> CREATE TABLE data ( price FLOAT(5,2) );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO data VALUES (13.6), (876.90), (-5.2), (-12345.678);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT price FROM data;
+------------+
| price      |
+------------+
|      13.60 |
|     876.90 |
|      -5.20 |
| -12345.68  |
+------------+
4 rows in set (0.00 sec)
```

对于小数点后面的数字个数超过了允许的数目的值，系统会自动将它四舍五入为最接近它的值，然后插入它。下面的例子演示了这一点：

```
mysql> CREATE TABLE data ( speed FLOAT (3,1) );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO data VALUES (123.765);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT speed FROM data;
+----------+
| speed    |
+----------+
|    123.8 |
+----------+
1 row in set (0.05 sec)
```

DECIMAL 数据类型用于要求非常高的精确度的计算中，这些类型允许指定数值的精确度和计数方法作为选择参数。精确度在这里是指为这个值保存的有效数字的总个数，而计数方法表示小数点后数字的个数。例如，语句 DECIMAL (5,2)规定了存储的值将不会超过 5 位数字，并且小数点后面有 2 位数字。看下面的示范举例：

```
mysql> CREATE TABLE data ( radius DECIMAL (6,3) );
Query OK, 0 rows affected (0.00 sec)

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

mysql> SELECT radius FROM data;
```

```
+-----------+
|    radius |
+-----------+
|     1.000 |
+-----------+
1 row in set (0.00 sec)
```

注释：MySQL 在内部把 DECIMAL 数据类型存储为字符串，更精确地保留它们的值。

忽略 DECIMAL 数据类型的精确度和计数方法修饰符将会使 MySQL 把所有标识为这个类型的字段的精确度默认为 10，计算方法默认为 0：

```
mysql> CREATE TABLE data ( f_decimal DECIMAL );
Query OK, 0 rows affected (0.00 sec)

mysql> DESCRIBE data;
+-----------+---------------+
| Field     | Type          |
+-----------+---------------+
| f_decimal | decimal(10,0) |
+-----------+---------------+
1 row in set (0.00 sec)
```

UNSIGNED 和 ZEROFILL 修饰符也可以被 FLOAT，DECIMAL 和 DOUBLE 数据类型接受，它们的结果和前一节中描述的一样。

5.2.2 字符串类型

MySQL 提供了 8 个基本的字符串类型，可以存储范围从简单的一个字符的字符串到巨大的文本块或二进制数据的字符串数据。表 5.2 列出了这些类型。

表 5.2 MySQL 字符串类型

类型	大小	用途
CHAR	0~255 字节	定长字符串
VARCHAR	0~255 字节	变长字符串
TINYBLOB	0~255 字节	不超过 255 个字符的二进制字符串
TINYTEXT	0~255 字节	短文本字符串
BLOB	0~65 535 字节	二进制形式的长文本数据
TEXT	0~65 535 字节	长文本数据
MEDIUMBLOB	0~16 777 215 字节	二进制形式的中等长度文本数据
MEDIUMTEXT	0~16 777 215 字节	中等长度文本数据
LONGBLOB	0~4 294 967 295 字节	二进制形式的极大文本数据
LONGTEXT	0~4 294 967 295 字节	极大文本数据

下面的几节将详细分析这些类型。

CHAR 和 VARCHAR 类型

在表 5.2 中列出的最简单的类型是 CHAR 类型，它用于定长字符串，并且必须在圆括号内用一个大小修饰符来定义。这个大小修饰符范围为 0 到 255，指定了要存储的值的长度——例

如，语句CHAR (10)指定了一个长度为 10 个字符的值。比指定长度小的值将会用空格适当填补；比指定长度大的值将被自动截短。下面的例子演示了这两种情况：

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

mysql> INSERT INTO data VALUES ('abcdefghijklmno');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO data VALUES ('abc');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT alphabet FROM data;
+------------+
| alphabet   |
+------------+
| abcdefghij |
| abc        |
+------------+
2 rows in set (0.00 sec)
```

CHAR类型认可一个可选择的BINARY修饰符，当用于比较运算时，这个修饰符使CHAR类型以一个二进制方式（不是传统的区分大小写）起作用。下面的举例示范了这一点：

```
mysql> CREATE TABLE data ( name CHAR(5) );
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO data VALUES ('HUGO');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM data WHERE name = 'hugo';
+--------+
| name   |
+--------+
| HUGO   |
+--------+
1 row in set (0.06 sec)
```

所以MySQL可以在字符串'hugo'和'HUGO'之间执行一个区分大小写的匹配。现在观察当我们改为一个 CHAR BINARY 类型时会出现什么情况：

```
mysql> ALTER TABLE data CHANGE name name CHAR(5) BINARY;
Query OK, 1 row affected (0.11 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

再执行这个查询一遍：

```
mysql> SELECT * FROM data WHERE name = 'hugo';
Empty set (0.00 sec)
```

因为现在字段使用一个二进制类型，所以MySQL 执行一个二进制比较，很显然由于不匹配，它以失败告终。

CHAR类型的一个变体是VARCHAR类型，它用于变长字符串，它也必须带有一个范围在0到255之间的大小指示器。然而，CHAR 类型和 VARCHAR 类型之间的差别在于 MySQL 处理这个指示器的方式：CHAR 把这个大小视为值的准确大小（用空格填补比较短的值，所以达到了这个大小），而 VARCHAR 类型把它视为最大值并且只使用了存储字符串实际上需要的字节数（增加一个额外的字节记录长度）。因而，较短的值当被插入一个语句为 VARCHAR 类型的字段时，将不会用空格填补（然而，较长的值仍然被截短）。

因为字段会根据它的内容动态地增长和缩短，所以当不能确定字段要存储多少个字符时，使用这种类型是一个很好的想法；因为 VARCHAR 类型只使用需要的最小数目的字节，所以它在数据库性能方面导致了更大的存储效率和（可能）空前的改进。

与 CHAR 类型一样，VARCHAR 类型也有一个可选的 BINARY 修饰符，它的功能跟前面的描述相同。

TEXT 和 BLOB 类型

对于比较长的字符串（即长度超过255个字符的字符串），MySQL 提供了 TEXT 和 BLOB（Binary Large Object，二进制大对象）两种类型，根据要存储数据的大小，它们都有不同的子类型。这些大数量的类型用于存储大的文本块或者像图像、声音文件那样的二进制数据。

BLOB 和 TEXT 类型在分类和比较的方式上不同：BLOB 类型区分大小写，TEXT 类型不区分大小写。MySQL 手册用“TEXT 类型是不区分大小写的 BLOB 类型”最确切地说明了这一点[①]。

大小修饰符不用于各种 BLOB 和 TEXT 子类型，比指定类型支持的最大范围大的值将被自动截短。

5.2.3 日期和时间类型

当处理日期和时间值时，MySQL 带有5个不同的数据类型可供选择，它们可以被广泛地分类为简单的日期或时间类型和混合日期/时间类型。根据数据要求的精确度，子类型在每一个分类中都可用，并且MySQL带有内置智能来识别并把多样化的输入格式转变为一个标准格式，以便于使用和操作。

表 5.3 列举了不同的日期和时间数据类型以及它们的允许范围和格式。

表 5.3 MySQL 日期类型

类型	大小（字节）	范围	格式	用途
DATE	3	1000-01-01~9999-12-31	YYYY-MM-DD	日期值
TIME	3	'-838:59:59'~'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1	1901~2155	YYYY	年份值
DATETIME	8	1000-01-01 00:00:00~9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值

① 见 MySQL 手册，http://www.mysql.com/documentation/。

（续表）

类型	大小（字节）	范围	格式	用途
TIMESTAMP	8	1970-01-01 00:00:00~ 2037 年的某个时候	YYYYMMDDHHMMSS	混合日期和时间值，时间戳

什么时候大小是重要的

MySQL引擎有时自动修改CHAR和VARCHAR字段类型来达到更好的存储效率和性能。转换的两个可能类型发生于：

- 最大字段长度少于4个字符的VARCHAR字段被自动转换为CHAR字段。
- 在至少包含一个可变长度字段的行中，最小字段长度为3个字符的CHAR字段，比如VARCHAR类型、TEXT类型和BLOB类型，被自动转换为VARCHAR字段。

接下来的几节将更加详细地讨论这些类型。

DATE，TIME 和 YEAR 类型

MySQL 用 DATE 和 YEAR 类型描述简单的日期值，而使用 TIME 类型描述时间值。这些值可以描述为字符串或不带分隔符的整数序列。如果描述为字符串，DATE类型的值应该使用连字号作为分隔符分隔开，而 TIME 类型的值应该使用冒号作为分隔符分隔开。

下面的例子说明了这些类型：

```
mysql> CREATE TABLE data ( birthday DATE );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO data VALUES ('2003-03-04'), (20030304);
Query OK, 2 rows affected (0.06 sec)

mysql> SELECT birthday FROM data;
+------------+
| birthday   |
+------------+
| 2003-03-04 |
| 2003-03-04 |
+------------+
2 rows in set (0.00 sec)

mysql> DROP TABLE data;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE data ( showtime TIME );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO data VALUES ('12:30:56'), ('12:30'), (123056);
Query OK, 3 rows affected (0.00 sec)

mysql> SELECT showtime FROM data;
```

```
+----------+
| showtime |
+----------+
| 12:30:56 |
| 12:30:00 |
| 12:30:56 |
+----------+
3 rows in set (0.00 sec)
```

请注意，没有冒号分隔符的TIME类型值可能被MySQL解释为持续时间，而不是时间戳，这样有时会使粗心的人感到迷惑，请看下面的示范举例：

```
mysql> CREATE TABLE data ( t TIME );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO data VALUES (1230);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT t FROM data;
+----------+
| t        |
+----------+
| 00:12:30 |
+----------+
1 row in set (0.00 sec)
```

虽然MySQL使用表5.3指定的格式返回日期和时间值，但是它能够接受与表中格式不完全一致的输入。在这种情形下，MySQL对输入的值应该描述的日期或时间做最大限度的估计。仔细思考下面的举例，它说明了这些估计是怎样起作用的：

```
mysql> CREATE TABLE data (
    -> f_date DATE,
    -> f_time TIME
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO data VALUES ('1978-4-6', 123403);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO data VALUES (650503,'3:4:5');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM data;
+------------+----------+
| f_date     | f_time   |
+------------+----------+
| 1978-04-06 | 12:34:03 |
| 2065-05-03 | 03:04:05 |
+------------+----------+
2 rows in set (0.06 sec)
```

MySQL 还对日期的年份中的两个数字的值，或是语句为 YEAR 类型的输入字段中的两个数字输入执行这种类型的最大限度的通译。因为所有的 YEAR 类型的值必须用 4 个数字存储，所有 MySQL 试图根据值的数字范围把两个数字的年份值转换为 4 个数字的值：把在 00~69 范围内的值转换到 2000~2069 范围内，而把 70~99 范围内的值转换到 1970~1979 之内。示范举例如下所示：

```
mysql> CREATE TABLE data ( graduation YEAR );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO data VALUES (2003), (04), (9), (53), (96);
Query OK, 5 rows affected (0.05 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT graduation FROM data;
+------------+
| graduation |
+------------+
|       2003 |
|       2004 |
|       2009 |
|       2053 |
|       1996 |
+------------+
5 rows in set (0.00 sec)
```

如果这种类型的转换不是我们想要的，则应该使用4个数字的年份值，MySQL就不会做这种数据转换了。

DATETIME 和 TIMESTAMP 类型

除了日期和时间数据类型，MySQL 还支持这两种类型的一个混合类型：DATETIME 和 TIMESTAMP 数据类型，它们可以把日期和时间作为一个单值的组成成分来存储。示范举例如下：

```
mysql> CREATE TABLE data (
    -> f_date_time DATETIME,
    -> f_time_stamp TIMESTAMP
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO data VALUES ('1999-03-12 12:30:12',
'2002-03-09 16:17:18');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO data VALUES (19990312123012, 20020309161718);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO data VALUES (NOW(), NULL);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT f_date_time, f_time_stamp FROM data;
+---------------------+----------------+
| f_date_time         | f_time_stamp   |
+---------------------+----------------+
| 1999-03-12 12:30:12 | 20020309161718 |
| 1999-03-12 12:30:12 | 20020309161718 |
| 2003-04-09 11:57:18 | 20030409115718 |
+---------------------+----------------+
3 rows in set (0.00 sec)
```

这两种类型通常用于自动存储包含当前日期和时间的时间戳，并且对执行大量数据库事务和需要建立一个调试和审查用途的审计跟踪的应用程序可以派上用场。

如果一个行中第一个字段语句为TIMESTAMP类型，而且这个字段没有被明确地指定值或者被指定了一个NULL值，MySQL将会自动用当前的日期和时间填充它。示范举例如下：

```
mysql> CREATE TABLE data ( f_time_stamp TIMESTAMP );
Query OK, 0 rows affected (0.06 sec)

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

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

mysql> SELECT f_time_stamp FROM data;
+----------------+
| f_time_stamp   |
+----------------+
| 20030409120052 |
| 20030409120058 |
+----------------+
2 rows in set (0.00 sec)
```

使用MySQL的NOW()函数在语句为DATETIME类型的字段中填充当前的日期和时间可以得到相同的结果：

```
mysql> CREATE TABLE data ( event DATETIME );
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO data VALUES (NOW());
Query OK, 1 row affected (0.05 sec)

mysql> SELECT event FROM data;
+---------------------+
| event               |
+---------------------+
| 2003-04-09 12:01:59 |
+---------------------+
1 row in set (0.00 sec)
```

有趣的不可见时间戳

与数值类型一样，通过使用一个显示宽度指示器控制TIMESTAMP类型字段的大小也是可能的。下面的表列举了各种可用的大小以及相应的格式[1]：

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

应该注意，虽然MySQL允许使用自定义的TIMESTAMP格式，但是它不提倡这种用法。原因非常简单：如果我们在一个TIMESTAMP字段上使用自定义格式，MySQL就不能够在这个字段上使用关键字。而且输出格式要通过使用在下一章中论述的DATE_FORMAT()函数才能得到。

在一个TIMESTAMP字段上使用一个显示宽度修饰语，对字段的实际大小和存储的值的范围不会产生影响。事实上，如果我们对TIMESTAMP字段设置了一个低显示宽度，然后决定通过调用ALTER TABLE命令加大它，这时以前隐藏的信息将变成可见的。示范举例如下：

```
mysql> CREATE TABLE data ( when TIMESTAMP(6) );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO data VALUES (NULL);
Query OK, 1 row affected (0.06 sec)

mysql> SELECT when FROM data;
+--------------+
| when         |
+--------------+
|       030409 |
+--------------+
1 row in set (0.06 sec)

mysql> ALTER TABLE data CHANGE when when TIMESTAMP(8);
Query OK, 1 row affected (0.11 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> SELECT when FROM data;
+--------------+
| when         |
+--------------+
|     20030409 |
+--------------+
1 row in set (0.00 sec)

mysql> ALTER TABLE data CHANGE when when TIMESTAMP(14);
Query OK, 1 row affected (0.06 sec)
```

① 见MySQL手册，http://www.mysql.com/documentation/。

```
Records: 1 Duplicates: 0 Warnings: 0

mysql> SELECT when FROM data;
+----------------+
| when           |
+----------------+
| 20030409135827 |
+----------------+
1 row in set (0.00 sec)
```

很显然，颠倒过来也是可以的。

应该注意，这个行为只对MySQL 4.1之前的版本适用，对MySQL 4.1及更高版本，TIMESTAMP类型的大小总是被设置为最大值。

在合法范围以外的日期和时间值通常被转换为最接近它的合法范围的边界点值，然后保存它，示范举例如下：

```
mysql> CREATE TABLE data (
    -> d date,
    -> t time
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO data VALUES ('10000-01-01', '9000:07:12');
Query OK, 1 row affected (0.05 sec)

mysql> SELECT d, t FROM data;
+------------+-----------+
| d          | t         |
+------------+-----------+
| 2010-00-00 | 838:59:59 |
+------------+-----------+
1 row in set (0.06 sec)
```

另外，MySQL把不合法的日期和时间值自动置为0，详细的示范举例如下：

```
mysql> INSERT INTO data VALUES ('0005-14-76', '9000:900:90');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT d, t FROM data;
+------------+----------+
| d          | t        |
+------------+----------+
| 0000-00-00 | 00:00:00 |
+------------+----------+
1 row in set (0.00 sec)
```

熟视无睹

MySQL允许不合法的日期和时间值存储在数据库中是不值得的。例如，仔细思考下面的列表，它把不合法的数据2003-02-31插入MySQL的一个DATE字段：

```
mysql> INSERT INTO data VALUES ('2003-02-31', '');
Query OK, 1 row affected (0.06 sec)

mysql> SELECT d FROM data;
+------------+
| d          |
+------------+
| 2003-02-31 |
+------------+
1 row in set (0.05 sec)
```

这样的值将会逃过 MySQL 的锐眼，因为 MySQL 的开发者没有对日期和时间值实施一个严格的检查程序。原因是什么？日期和时间值的确认被认为是应该在应用级别上处理的事情，因此，使 MySQL 负担对每次输入系统的日期或时间进行确认的额外开销是不适当的。

5.2.4 复合类型

除了在前一节中论述的基本类型以外，MySQL 还有两个复合类型 ENUM 和 SET，它们扩展了 SQL 规范。虽然这些类型在技术上是字符串类型，但是我们把它们看成不同的分类，因为它们的值必须从一个预先定义好的字符串集合中选取。一个ENUM类型从一个允许值集合中只选择单个值；SET 类型允许从可选集合中选择任意多个值。

ENUM 类型

ENUM类型在允许从集合中只选择单个值时，通常用于互斥的数据值。一个常见的例子是性别属性，它或者是男性或者是女性。示范举例如下所示：

```
mysql> CREATE TABLE data ( gender ENUM('M', 'F') );
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO data values ('M'), ('m'), ('f'), ('yy'), (NULL);
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 1

mysql> SELECT gender FROM data;
+--------+
| gender |
+--------+
| M      |
| M      |
| F      |
|        |
| NULL   |
+--------+
5 rows in set (0.06 sec)
```

可以看到，ENUM类型可以包含列于集合允许成员中的惟一的值或NULL值，任何不使用这些值而使用其他值的操作都将使 MySQL 插入一个空的字符串。如果插入值的大小写与在字段语句中相对应值的大小写不匹配，那么插入值的大小写将被自动改变为与字段语句相匹配，在这个实例中可以清晰地看到这一点。

有趣的是，虽然ENUM的元素只能指定为字符串，但是它们在系统内部可以存储为数字并且从1开始用数字做索引。一个ENUM类型最多可以包含65 536个元素，其中一个元素为MySQL保留，用来保存错误信息，这个错误值用索引0或者一个空字符串表示。

因为可以通过字符串或数字索引访问ENUM值，所以下面的两个命令是相等的：

```
mysql> SELECT COUNT(*) FROM data WHERE gender = 'M';
+----------+
| COUNT(*) |
+----------+
|        2 |
+----------+
1 row in set (0.05 sec)

mysql> SELECT COUNT(*) FROM data WHERE gender = 1;
+----------+
| COUNT(*) |
+----------+
|        2 |
+----------+
1 row in set (0.05 sec)
```

MySQL认为在ENUM类型语句中出现的惟一值是合法的，除了这些值外，其他任何插入值的企图都将失败，MySQL用错误值表示出现错误了。这意味着仅仅通过搜索包含空字符串或对应数字索引为0的行就可以很容易地查找出错误记录的位置。我们再看一下在前面论述的实例并计算出现错误的行的数目就可以示范这一点。下面的两个命令是等价的：

```
mysql> SELECT COUNT(*) FROM data WHERE gender = 0;
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.05 sec)

mysql> SELECT COUNT(*) FROM data WHERE gender = '';
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.00 sec)
```

SET类型

SET类型与ENUM类型相似但又不完全相同，SET类型允许从预先定义的字符串值集合中选取任意数目的值。语句使用ENUM类型的字段最多包含一个元素，而SET类型不同，语句为SET类型的字段可以包含0个、1个或1个以上从允许值集合中选取的值。这使SET类型可以做多个选择的选取，比如下面实例中的选取：

```
mysql> CREATE TABLE data ( pizza_topping SET ('ham', 'cheese',
'tomato', 'chicken', 'pepperoni') );
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO data VALUES ('ham');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO data VALUES ('cheese, pepperoni');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT pizza_topping FROM data;
+------------------+
| pizza_topping    |
+------------------+
| ham              |
| cheese,pepperoni |
+------------------+
2 rows in set (0.00 sec)
```

与ENUM类型一样，任何企图使用一个不在预先定义集合中的值的操作都会使MySQL插入一个空字符串。

```
mysql> INSERT INTO data VALUES ('egg');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT pizza_topping FROM data;
+------------------+
| pizza_topping    |
+------------------+
| ham              |
| cheese,pepperoni |
|                  |
+------------------+
3 rows in set (0.00 sec)
```

然而，如果试图插入一个既包含合法元素又包含非法元素的记录，如下面例子所示的一样，MySQL将会除去非法的部分，保留合法的部分：

```
mysql> CREATE TABLE data ( pizza_topping SET ('ham', 'cheese',
'tomato', 'chicken', 'pepperoni') );
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO data VALUES ('egg,cheese,tomato,bacon');
Query OK, 1 row affected (0.05 sec)

mysql> SELECT pizza_topping FROM data;
+------------------+
| pizza_topping    |
+------------------+
| cheese,tomato    |
+------------------+
1 row in set (0.00 sec)
```

一个SET类型最多可以包含64项。在SET类型中，值被存储为一个分离的位序列，这些位表示与它相对应的选取项。位表示法是创建有序元素集合的一种简单而有效的方法，它还除去重复的元素，因为在SET类型中相同的元素重复两次是不可能的。仔细思考下面的例子，它说明了这一点：

```
mysql> CREATE TABLE data ( pizza_topping SET ('ham', 'cheese',
'tomato', 'chicken', 'pepperoni') );
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO data VALUES ('ham,ham,ham,cheese');
Query OK, 1 row affected (0.05 sec)

mysql> SELECT pizza_topping FROM data;
+------------------+
| pizza_topping    |
+------------------+
| ham,cheese       |
+------------------+
1 row in set (0.00 sec)
```

这样也使查找包含错误数据的记录的位置变得很简单，需要做的就是查找包含空字符串或二进制值为0的行。下面的两个命令是等价的，它们说明了这一点：

```
mysql> SELECT COUNT(*) FROM data where pizza_topping = '';
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.05 sec)

mysql> SELECT COUNT(*) from data where pizza_topping = 0;
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.06 sec)
```

5.2.5 数据类型选择

现在已经了解了各种数据类型，花少许时间掌握如何选择适当的数据类型，对我们是非常有益的。

正如在本章前面部分讨论的那样，数据分类在使存储数据更有一致性和提高存储效率和速度方面起了重要的作用。从而，当设计一个数据库时对数据类型的选取不能掉以轻心，并且在从MySQL可以使用的多组选项做出选择之前，应该对所有涉及的问题做出充分的考虑。

下面列出了在对MySQL的列选取数据类型时需要牢记在心中的几个标准。

允许值

允许输入一个字段中的值的范围和类型是决定哪个数据类型加在这个字段上的最直接因素。例如，如果让一个列包含股票价格，很显然在这个列中不应该使用字符串或日期类型，而应该使用数值类型。具体是 FLOAT 或 DECIMAL 类型，依赖于我们需要的精确度。同样，如果用一个列存储生日，为这个列选择 DATE 类型是有意义的，而不是使用一个 VARCHAR 类型，然后在应用程序级别上编写代码把字符串的各个部分分解为日、月和年。

存储效率

考虑字段可能的值的范围并且根据这个范围选取一个适当的、大小合适的数据类型也是很重要的，尤其是在处理数值类型的时候。例如，如果我们让一个列存储在系统上注册的所有用户的年龄，则字段中的值总会落在 0 和 255 之间，因而对这个字段使用一个标准 INT 类型或 BIGINT 类型效率是很差的，因为 UNSIGNED TINYINT 类型会提供我们需要的所有功能，同时占用较少的空间（只有 1 个字节而不是 4 个或 8 个字节）。

对字符串值应用类似的逻辑：例如，如果我们知道一个特殊的字段可以最多包含一个含有 2 个字符的字符串（如美国州名的代码），那么使用一个带有长度为 2 的 CHAR 类型，而不使用 TEXT 和 BLOB 类型是很理智的，因为 TEXT 和 BLOB 类型要占用超过我们需要的更多的空间，而且并没有提供额外的功能。同样，如果我们知道一个列中惟一允许的值属于一个预先定义的集合，那么这个约束限制将会立即提示我们为这个字段选择一个 ENUM 或 SET 类型。

格式和显示要求

影响字段类型选择的另一个重要因素是存储在字段内的数据的应用格式和显示要求。例如，如果必须总是把字符串数据格式化到一个特定的长度，那么使用可以在需要位置自动用空格填充值的 CHAR 数据类型是比较好的想法（对数值数据，用 0 作为填充数字的 ZEROFILL 修饰符是等价的）。同样，如果我们让数字必须总是显示 4 个小数位，那么 DECIMAL 数据类型是一个很好的选择，因为它带有内置结构，允许我们指定它保存的值的精确度。

当处理日期和时间时，格式化也起着重要的作用。依赖于我们的应用程序是以数据库中的存储形式来显示日期和时间，还是进一步处理它们。我们可以选取一个标准 DATE 类型，它使用 YYYY-MM-DD 格式，很容易为人们所理解；或者选择一个 TIMESTAMP 类型，它使用 YYYYMMDDHHMMSS 格式，便于程序处理。

数据处理要求

数据类型的选择还应该考虑数据操作和应用处理的要求，比如说某些类型更适合于加快数据操作而不是其他方面。例如，如果我们希望把一个列中的所有值加起来，那么选择一个本身支持这种操作的类型是很重要的，比如 INT 类型。

当处理日期和时间时，这个因素会变得更为关键，因为日期和时间的比较和操作在最好的情况下也非常复杂，我们想尽可能地扩展 MySQL 的内建功能使它能够处理这些操作类型。完全从性能角度来看，当我们能够使用一个支持这些操作的内建类型来做这些工作的时候，再把日期在数据库中存储为字符串，然后手工在应用程序级别上编写代码来执行相加、相减或者比较这些值是很荒唐的事情。

排序、索引和比较要求

如果想用不同的标准对一列中所有的值进行排序，那么我们需要选择一种能够有效地执行这些功能并且带有最大限度的灵活性的类型。例如，存储为字符数据的数值和存储为数字的相同数值的排序是不同的，并且在文本字段中存储的数据值与在数据字段中存储的相同值的排序也是不同的。因而，在做操作之前研究选取的类型是否支持应用排序要求是很重要的。

索引要求也影响数据类型选择。我们想给在查询中作为选择标准的字段上加索引，这样能够加快处理时间。然而，与数值数据比较，给字符数据加索引相对来说效率不高，因为在数字上执行操作要比在字符串上执行操作快。另外，MySQL的比较老的版本把特殊的要求强加给语句为索引的字段，我们需要确保字段上保存的值与这些要求不冲突。

如果想以类似的方式对数值进行比较，我们需要保证使用的数据类型支持比较运算，并且相互兼容。比较的类型也影响使用的数据类型。例如，如果想比较字符串的值，我们需要估计这个比较运算以二进制（区分大小写）方式还是非二进制（不区分大小写）方式执行，并且应对 MySQL 的列应用适当的类型修饰符。

5.3 小结

本章标志着对MySQL命令集合进行更为详细的讨论的开始。MySQL带有大量几乎支持任何应用的数据类型。前面的章节通过对这些数据类型进行详细地研究，充分说明了这一点，并且连同实例说明了每一种类型的重要特点和需注意之处。对数据分类的需求和选择适当类型的重要性做简短描述的几节，使这些信息贯穿于上下文中，此外还对在为 MySQL 的列选择数据类型时需要牢记心中的标准做了解释。

第6章　MySQL运算符

在第4章的SQL教程中，我们已经看到，像SELECT这样的MySQL语句是如何使用比较和逻辑运算符，只检索与指定条件或条件集合相匹配的记录的。这些SQL运算符给语言增添了巨大的灵活性，可以建立复杂的表达式和子句，它们允许我们非常容易地对MySQL表中的数据执行复杂的比较和转换操作。

本章将介绍MySQL的许多运算符，解释它们的功能，并且用例子演示了如何使用它们来执行计算和比较、创建条件组、在字符串中搜索指定表达式以及对数据类型进行转换。

MySQL的运算符超过25个，每一个都有特殊的用途。这些运算符根据它们的功能可以划分为以下4种类型：

- 算术运算符
- 比较运算符
- 逻辑运算符
- 位运算符

接下来的几节将详细地对这些分类进行研究。

名字的含义是什么

我们很有趣地发现，在MySQL世界中，一个运算符和一个函数的差别仅仅在语义上。一个函数用它的名字以及在圆括号中的参数来调用，而一个运算符总是由一个或两个特殊的、有参数在中间或在一侧的符号来表示。在内部，MySQL以完全相同的方式处理函数和运算符。

6.1　算术运算符

与大多数编程语言一样，MySQL支持大多数通用的、允许我们执行计算的算术运算符。表6.1列出了在MySQL中可用的算术运算符。

表6.1　MySQL的算术运算符

运算符	作用
+	加法
-	减法
*	乘法
/	除法，返回商
%	除法，返回余数

我们从简单的运算符开始：加法和减法。+运算符用于获得一个或更多值的和，示范举例如下所示：

```
mysql> SELECT 10 + 5, 1000000 + 64763;
+--------+-----------------+
| 10 + 5 | 1000000 + 64763 |
+--------+-----------------+
|     15 |         1064763 |
+--------+-----------------+
1 row in set (0.04 sec)

mysql> SELECT 3.24 + 0.89844,0.00000000001 + 0.000000000001;
+----------------+-------------------------------+
| 3.24 + 0.89844 | 0.00000000001 + 0.000000000001 |
+----------------+-------------------------------+
|        4.13844 |                  0.000000000011 |
+----------------+-------------------------------+
1 row in set (0.04 sec)
```

-运算符用于从一个值中减去另一个值：

```
mysql> SELECT 5 - 1, 999 - 1000, 0.24 - 0.2;
+-------+------------+------------+
| 5 - 1 | 999 - 1000 | 0.24 - 0.2 |
+-------+------------+------------+
|     4 |         -1 |       0.04 |
+-------+------------+------------+
1 row in set (0.00 sec)

mysql> SELECT -46.7 - 0.3, 0.6484895563 - 0.63858546546;
+-------------+-----------------------------+
| -46.7 - 0.3 | 0.6484895563 - 0.63858546546 |
+-------------+-----------------------------+
|       -47.0 |                 0.00990409084 |
+-------------+-----------------------------+
1 row in set (0.00 sec)
```

可以用*运算符使数字相乘，得到两个或更多个值的乘积：

```
mysql> SELECT 7 * 10, 7 * 0, -11.1 * 2.5, -11000000 * 1276464646;
+--------+-------+-------------+-----------------------+
| 7 * 10 | 7 * 0 | -11.1 * 2.5 | -11000000 * 1276464646 |
+--------+-------+-------------+-----------------------+
|     70 |     0 |       -27.8 |     -14041111106000000 |
+--------+-------+-------------+-----------------------+
1 row in set (0.01 sec)
```

/运算符用一个值除以另一个值得到商：

```
mysql> SELECT 10 / 2, 12.1 / -0.01, 22 / 7, 22.00 / 7.000000;
+--------+--------------+--------+-----------------+
| 10 / 2 | 12.1 / -0.01 | 22 / 7 | 22.00 / 7.000000 |
+--------+--------------+--------+-----------------+
|   5.00 |   -1210.0000 |   3.14 |       3.14285714 |
+--------+--------------+--------+-----------------+
1 row in set (0.00 sec)
```

还可以使用运算符 % 得到一个除法运算的余数：

```
mysql> SELECT 25 % 7, -33 % 7;
+--------+---------+
| 25 % 7 | -33 % 7 |
+--------+---------+
|      4 |      -5 |
+--------+---------+
1 row in set (0.00 sec)
```

不 同 点

可以看到，MySQL 允许使用带有任意的由用户提供值的 SELECT 语句，这一点与 Oracle 不同。当选择一个不能从表中读取的值时，Oracle 要求使用 FROM DUAL 语法。

很显然，除以 0 的除法是不允许的。如果这样做，MySQL 会返回一个 NULL：

```
mysql> SELECT 9 / 0;
+--------+
| 9 / 0  |
+--------+
|   NULL |
+--------+
1 row in set (0.00 sec)
```

请注意，所有涉及整数的数学运算都使用64位精度进行计算。在这个范围之外的数字可能产生一个预料不到的结果，示范举例如下所示：

```
mysql> SELECT 10000000000000000000000000000 * 1;
+-------------------------------------+
| 10000000000000000000000000000 * 1 |
+-------------------------------------+
|      9999999999999999 1000000000000 |
+-------------------------------------+
1 row in set (0.01 sec)
```

更进一步，用字符串表示的数字在任何可能便于计算的地方都被自动地转换为字符串。当执行转换时，MySQL 遵循两个基本规则：

- 如果第一位是数字的字符串被用于一个算术运算中，那么它被转换为这个数字的值。
- 如果一个包含字符和数字混合的字符串不能被正确地转换为数字，那么它被转换成 0。

下面的实例说明了这些规则是如何执行的：

```
mysql> SELECT '500' + '00002', '8.9' * 10, '90AA' + '0';
+-----------------+------------+--------------+
| '500' + '00002' | '8.9' * 10 | '90AA' + '0' |
+-----------------+------------+--------------+
|             502 |         89 |           90 |
+-----------------+------------+--------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT 'AA90' + '1', '10X' * 1 * 'pqr';
+--------------+-------------------+
| 'AA90' + '1' | '10X' * 1 * 'pqr' |
+--------------+-------------------+
|            1 |                 0 |
+--------------+-------------------+
1 row in set (0.00 sec)
```

让我们脱离理论，来看一下这些运算符在实践中是如何工作的。考虑下面简单的MySQL表，它列出了玩具公司4种产品的成本价（厂商价，即CTM）、批发价（给零售商的价格，即CTR）、零售价（消费者的价格，即CTC）和当前存货量：

```
mysql> SELECT * FROM toys;
+----+-----------------+-------+-------+-------+----------+
| id | item            | ctm   | ctr   | ctc   | quantity |
+----+-----------------+-------+-------+-------+----------+
|  1 | Huggy bear      |  5.00 |  7.40 |  9.90 |      300 |
|  2 | Coloring book   |  3.00 |  5.00 |  7.99 |     2000 |
|  3 | Board game      | 12.00 | 20.00 | 39.90 |     1430 |
|  4 | Activity CD-ROM |  2.00 |  6.00 | 17.49 |     3780 |
+----+-----------------+-------+-------+-------+----------+
4 rows in set (0.00 sec)
```

现在当在这些原始数据上执行计算时，算术运算符可以派上用场。例如，为厂商查看每一产品的利润率：

```
mysql> SELECT item, ctr - ctm FROM toys;
+-----------------+-----------+
| item            | ctr - ctm |
+-----------------+-----------+
| Huggy bear      |      2.40 |
| Coloring book   |      2.00 |
| Board game      |      8.00 |
| Activity CD-ROM |      4.00 |
+-----------------+-----------+
4 rows in set (0.01 sec)
```

为零售商查看利润率：

```
mysql> SELECT item, ctc - ctr FROM toys;
+-----------------+-----------+
| item            | ctc - ctr |
+-----------------+-----------+
| Huggy bear      |      2.50 |
| Coloring book   |      2.99 |
| Board game      |     19.90 |
| Activity CD-ROM |     11.49 |
+-----------------+-----------+
4 rows in set (0.00 sec)
```

想要知道每个产品在厂家生产它和消费者购买它这段时间内价格上涨的情况吗？下面的列表显示了绝对值和百分率两种条目。

```
mysql> SELECT item, ctc - ctm, (ctc - ctm) / ctm * 100 FROM toys;
+-----------------+----------+-----------------------+
| item            | ctc - ctm | (ctc - ctm) / ctm * 100 |
+-----------------+----------+-----------------------+
| Huggy bear      |     4.90 |               98.0000 |
| Coloring book   |     4.99 |              166.3333 |
| Board game      |    27.90 |              232.5000 |
| Activity CD-ROM |    15.49 |              774.5000 |
+-----------------+----------+-----------------------+
4 rows in set (0.01 sec)
```

很显然，得到厂商以产品作为分类的库存货值也是可能的：

```
mysql> SELECT item, ctm * quantity FROM toys;
+-----------------+---------------+
| item            | ctm * quantity |
+-----------------+---------------+
| Huggy bear      |       1500.00 |
| Coloring book   |       6000.00 |
| Board game      |      17160.00 |
| Activity CD-ROM |       7560.00 |
+-----------------+---------------+
4 rows in set (0.00 sec)
```

由厂商提供给零售商的每种产品的折扣（以绝对值和百分率条目形式）为：

```
mysql> SELECT item, ctc-ctr, (ctc - ctr)/ctm * 100 FROM toys;
+-----------------+--------+----------------------+
| item            | ctc-ctr | (ctc - ctr)/ctm * 100 |
+-----------------+--------+----------------------+
| Huggy bear      |   2.50 |              50.0000 |
| Coloring book   |   2.99 |              99.6667 |
| Board game      |  19.90 |             165.8333 |
| Activity CD-ROM |  11.49 |             574.5000 |
+-----------------+--------+----------------------+
4 rows in set (0.00 sec)
```

当然，算术运算只能提供给我们这么多——有时候，我们还需要执行比较运算。

6.2　比较运算符

当使用SELECT查询时，我们要感谢MySQL许多比较运算符的存在，它允许我们对表达式的左边和右边进行比较。一个比较运算的结果总是1（真）、0（假）或为NULL（不能确定）。

表6.2列出了在MySQL中可以使用的各种比较运算符。

表6.2　MySQL的比较运算符

运算符	作用
=	等于
<> 或 !=	不等于
<=>	NULL安全的等于

（续表）

运算符	作用
<	小于
<=	小于等于
>	大于
>=	大于等于
BETWEEN	存在于指定范围
IN	存在于指定集合
IS NULL	为 NULL
IS NOT NULL	不为 NULL
LIKE	通配符匹配
REGEXP 或 RLIKE	正则表达式匹配

比较运算符可以用于比较数字和字符串。数字作为浮点值比较，而字符串以不区分大小写的方式进行比较（除非使用特殊的BINARY关键字——下一页的补充资料中有这方面的更多信息）。另外，正如我们在前一节中看到的那样，MySQL包含根据需要自动地把数字转换为字符串（反之亦然）的智能（下面的补充资料包含MySQL如何执行估计的技巧信息）。思考下面的例子，它说明了在不同情况下 MySQL 可以以不同的方式处理数字和字符串：

```
mysql> SELECT 6 = '6gfh', '6' = '6gfh';
+------------+--------------+
| 6 = '6gfh' | '6' = '6gfh' |
+------------+--------------+
|          1 |            0 |
+------------+--------------+
1 row in set (0.03 sec)
```

人工智能

在内部，MySQL使用由类Item衍生的一个类对象来表示表达式分析树上的每个节点。类Item有三个值估计方法：longlong Item::val_int()、double Item::val()和String* Item::val_str (string* buf)。第一种方法以64位整数上下文估计表达式的节点，第二种方法使用浮点值，第三种方法使用字符串上下文进行估计。每个父节点对象通过在计算子表达式时调用适当的Item方法来控制它的直接子节点的估计结果，并且MySQL在每个选择结果的情形下都试图做出最直观、符合标准的判断。

=运算符用于比较一个表达式的两边是否相等：

```
mysql> SELECT 2 = 2,  2.37 = 2.4, 2.415 = 2.4150000;
+-------+------------+-------------------+
| 2 = 2 | 2.37 = 2.4 | 2.415 = 2.4150000 |
+-------+------------+-------------------+
|     1 |          0 |                 1 |
+-------+------------+-------------------+
1 row in set (0.00 sec)
```

也可以对字符串进行比较：

```
mysql> SELECT 'a' = 'a', 'a' = 'A', 'A' = 'B', 'circle' = 'round';
+-----------+-----------+-----------+-------------------+
| 'a' = 'a' | 'a' = 'A' | 'A' = 'B' | 'circle' = 'round' |
```

```
+-----------+-----------+-----------+-------------------+
|         1 |         1 |         0 |                 0 |
+-----------+-----------+-----------+-------------------+
1 row in set (0.00 sec)
```

与=运算符相对立的是<>运算符，它用来检测表达式的两边是否不相等，如果不相等返回真值，相等返回假值。

```
mysql> SELECT 7 <> 7, 7 <> 8;
+--------+--------+
| 7 <> 7 | 7 <> 8 |
+--------+--------+
|      0 |      1 |
+--------+--------+
1 row in set (0.00 sec)
```

这个运算符也可以用于比较字符串：

```
mysql> SELECT 'x' <> 'X', 'x' <> 'y', '7x' <> '7y';
+------------+------------+--------------+
| 'x' <> 'X' | 'x' <> 'y' | '7x' <> '7y' |
+------------+------------+--------------+
|          0 |          1 |            1 |
+------------+------------+--------------+
1 row in set (0.00 sec)
```

还可以对 NULL 值进行比较：

```
mysql> SELECT NULL <> NULL, 0 <> NULL, 0 <> 0;
+--------------+-----------+--------+
| NULL <> NULL | 0 <> NULL | 0 <> 0 |
+--------------+-----------+--------+
|         NULL |      NULL |      0 |
+--------------+-----------+--------+
1 row in set (0.02 sec)
```

双 重 人

因为在默认情况下 MySQL 以不区分大小写的方式比较字符串，所以表达式'a'='A'的结果为真。如果想执行区分大小写的比较，可添加 BINARY 关键字，它告诉 MySQL，在它之后的字符串应该以二进制方式处理。这时，当在字符串上执行比较运算时，MySQL 将牢牢记住字符串的大小写。示范举例如下所示：

```
mysql> SELECT 'Aloha' = 'aloha', BINARY 'Aloha' = 'aloha';
+-------------------+-------------------------+
| 'Aloha' = 'aloha' | BINARY 'Aloha' = 'aloha' |
+-------------------+-------------------------+
|                 1 |                       0 |
+-------------------+-------------------------+
1 row in set (0.00 sec)
```

BINARY 关键字是 MySQL 对 CAST()函数的一个排列的简略表示。CAST()函数将在下一章详细论述。

<=, >=, <和>运算符用来比较表达式的左边是否小于或等于、大于或等于、小于还是大于它的右边：

```
mysql> SELECT 100 > 100, 100 > 10, 100 < 10, 3.14 > 3.144;
+-----------+----------+----------+--------------+
| 100 > 100 | 100 > 10 | 100 < 10 | 3.14 > 3.144 |
+-----------+----------+----------+--------------+
|         0 |        1 |        0 |            0 |
+-----------+----------+----------+--------------+
1 row in set (0.00 sec)

mysql> SELECT 10 <= 10, 10 >= 19, 10 <= 25, -50 > 50, -50 < 50;
+----------+----------+----------+----------+----------+
| 10 <= 10 | 10 >= 19 | 10 <= 25 | -50 > 50 | -50 < 50 |
+----------+----------+----------+----------+----------+
|        1 |        0 |        1 |        0 |        1 |
+----------+----------+----------+----------+----------+
1 row in set (0.00 sec)
```

按照字符串在字母表中表示相对位置的值，它们也可以按不区分大小写的方式进行比较：

```
mysql> SELECT 'a' > 'a', 'x' <= 'x', 'y' > 'Y', 'e' > 'f';
+-----------+------------+-----------+-----------+
| 'a' > 'a' | 'x' <= 'x' | 'y' > 'Y' | 'e' > 'f' |
+-----------+------------+-----------+-----------+
|         0 |          1 |         0 |         0 |
+-----------+------------+-----------+-----------+
1 row in set (0.00 sec)
```

空格问题

请注意，当比较字符串时，MySQL 忽视末尾空格、制表符和换行符。因此，表达式

```
'zebra' = 'ZEBRA'
```

和

```
'zebra'='ZEBRA
```

是相等的，并且都返回真值。

然而，前面的空格、制表符和换行符在比较运算中被保留和使用。比较表达式

```
'zebra'=  'ZEBRA'
```

和

```
'zebra' ='      ZEBRA'
```

将不会返回相同的结果。

BETWEEN 运算符用于检验一个值（或者一个求值表达式）是否存在于一个指定的范围之内：

```
mysql> SELECT 10 BETWEEN 0 AND 100, 10 BETWEEN 11 AND 100;
```

```
+----------------------+----------------------+
| 10 BETWEEN 0 and 100 | 10 BETWEEN 11 AND 100 |
+----------------------+----------------------+
|                    1 |                    0 |
+----------------------+----------------------+
1 row in set (0.00 sec)
```

如果被测试的表达式在指定的范围内（包括范围的两个端点），那么使用BETWEEN运算符的比较运算返回真值。这不仅适用于数字，对字符串同样适用，示范举例如下所示：

```
mysql> SELECT 'y' BETWEEN 'x' AND 'z', 'z' BETWEEN 'x' AND 'y';
+-------------------------+-------------------------+
| 'y' BETWEEN 'x' AND 'z' | 'z' BETWEEN 'x' AND 'y' |
+-------------------------+-------------------------+
|                       1 |                       0 |
+-------------------------+-------------------------+
1 row in set (0.00 sec)
```

可以添加NOT逻辑运算符对一个BETWEEN比较取反，这种情况只有表达式在给定范围之外时才返回真值：

```
mysql> SELECT 99 NOT BETWEEN 10 AND 50, 'f' NOT BETWEEN 'a' AND 'z';
+---------------------------+-----------------------------+
| 99 NOT BETWEEN 10 AND 50 | 'f' NOT BETWEEN 'a' AND 'z' |
+---------------------------+-----------------------------+
|                         1 |                           0 |
+---------------------------+-----------------------------+
1 row in set (0.00 sec)
```

类似地，IN运算符用于检验一个值（或者一个求值表达式）是否包含在一个指定的值集合中。示范举例如下所示：

```
mysql> SELECT 7 IN (1,2,3,4,5,6,7,8,9);
+--------------------------+
| 7 IN (1,2,3,4,5,6,7,8,9) |
+--------------------------+
|                        1 |
+--------------------------+
1 row in set (0.00 sec)
```

如果被测试的值在指定的集合中，使用IN运算符的比较运算返回真值。可以想像，我们可以对字符串运用这个运算符：

```
mysql> SELECT 'red' IN ('red', 'green', 'blue');
+----------------------------------+
| 'red' IN ('red', 'green', 'blue') |
+----------------------------------+
|                                1 |
+----------------------------------+
1 row in set (0.00 sec)
```

可以用BINARY关键字来执行区分大小写的比较运算：

```
mysql> SELECT BINARY 'ross' IN ('Chandler', 'Joey', 'Ross');
+-----------------------------------------------+
| BINARY 'ross' IN ('Chandler', 'Joey', 'Ross') |
+-----------------------------------------------+
|                                             0 |
+-----------------------------------------------+
1 row in set (0.00 sec)
```

和其他比较运算符一样，当求值表达式或不匹配集合中的所有值为NULL时，IN运算符返回NULL：

```
mysql> SELECT NULL IN (1,2), NULL IN (1,2,3,NULL);
+---------------+----------------------+
| NULL IN (1,2) | NULL IN (1,2,3,NULL) |
+---------------+----------------------+
|          NULL |                 NULL |
+---------------+----------------------+
1 row in set (0.01 sec)
```

通过添加NOT逻辑运算符，可以对IN运算符的行为取反，如果所求的表达式或值不是指定的值集合中的一个元素，那么这时返回真值：

```
mysql> SELECT 3 NOT IN (1,2);
+----------------+
| 3 NOT IN (1,2) |
+----------------+
|              1 |
+----------------+
1 row in set (0.21 sec)
```

现在，如果我们注意这一节中的举例，就会看出，通常包括一个NULL值的任何一个比较运算往往产生一个NULL结果。下面的实例扼要说明了这一点：

```
mysql> SELECT 11 = NULL, 56 <> NULL, 1 < 2 < NULL;
+-----------+------------+--------------+
| 11 = NULL | 56 <> NULL | 1 < 2 < NULL |
+-----------+------------+--------------+
|      NULL |       NULL |         NULL |
+-----------+------------+--------------+
1 row in set (0.07 sec)

mysql> SELECT NULL IN (5,NULL,30), 7 BETWEEN 0 AND NULL;
+---------------------+----------------------+
| NULL IN (5,NULL,30) | 7 BETWEEN 0 AND NULL |
+---------------------+----------------------+
|                NULL |                 NULL |
+---------------------+----------------------+
1 row in set (0.07 sec)
```

这样出现了一个重要的问题：在一个表达式中，如何测试实际 NULL 值的存在和不存在呢？答案是 IS NULL 和 IS NOT NULL 运算符，它们专门用于执行包含 NULL 值的比较运算。

思考下面的举例，它演示了如何用 IS NULL 运算符来测试一个值是否为空：

```
mysql> SELECT 88 IS NULL, 0 IS NULL, NULL IS NULL, 1=NULL IS NULL;
+------------+-----------+--------------+----------------+
| 88 IS NULL | 0 IS NULL | NULL IS NULL | 1=NULL IS NULL |
+------------+-----------+--------------+----------------+
|          0 |         0 |            1 |              1 |
+------------+-----------+--------------+----------------+
1 row in set (0.00 sec)
```

可以看到，在一个 IS NULL 测试中只有 NULL 值返回真值。尤其是这个例子中的最后一个表达式——尽管表达式 1 = NULL 包含整数 1，整个表达式仍然取值为 NULL（因为它包含一个 NULL 值），因而 IS NULL 测试返回一个肯定的结果。

可是，当使用 IS NOT NULL 运算符时，则相反的情况为真。示范举例如下所示：

```
mysql> SELECT 6478 IS NOT NULL, NULL IS NOT NULL, 1=NULL IS NOT NULL;
+------------------+-----------------+-------------------+
| 6478 IS NOT NULL | NULL IS NOT NULL | 1=NULL IS NOT NULL |
+------------------+-----------------+-------------------+
|                1 |               0 |                 0 |
+------------------+-----------------+-------------------+
1 row in set (0.00 sec)
```

还可以使用特殊的<=>运算符，MySQL 手册把它称为“NULL 安全的等于”运算符。这意味着即使当包含在比较运算中的表达式含有一个 NULL 值时，在一个正常的偏差下，MySQL 也会为比较运算返回一个真值或假值。示范举例如下所示：

```
mysql> SELECT 2 = NULL, 2 <=> NULL;
+----------+------------+
| 2 = NULL | 2 <=> NULL |
+----------+------------+
|     NULL |          0 |
+----------+------------+
1 row in set (0.00 sec)

mysql> SELECT NULL = NULL, NULL <=> NULL;
+-------------+---------------+
| NULL = NULL | NULL <=> NULL |
+-------------+---------------+
|        NULL |             1 |
+-------------+---------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT 0 = NULL, 0 <=> NULL;
+----------+------------+
| 0 = NULL | 0 <=> NULL |
+----------+------------+
|     NULL |          0 |
+----------+------------+
1 row in set (0.00 sec)
```

在后面的章节中我们将看到，当处理含有NULL数据的MySQL列时，这三个运算符会派上用场。

NULL解惑

一个空值是否与一个NULL值相同呢？下面将很快地解决这个疑惑：

```
mysql> SELECT '' IS NULL;
+-------------+
| '' IS NULL |
+-------------+
|           0 |
+-------------+
1 row in set (0.00 sec)
```

可以看到，与很多编程语言不同，MySQL对一个空值和一个NULL值做了很清晰的区分。这不是MySQL的特殊规定，而是SQL标准的一个完整部分。这也是导致新手迷惑的常见原因，所以应该注意。

如果想执行通配符数据搜索，应该使用LIKE运算符。它通过在表达式中允许使用专门的通配符字符，可以找出与指定搜索字符串全部或部分匹配的记录。思考下面的示范举例：

```
mysql> SELECT 'Roger says hello' LIKE '%ll%';
+--------------------------------+
| 'Roger says hello' LIKE '%ll%' |
+--------------------------------+
|                              1 |
+--------------------------------+
1 row in set (0.06 sec)
```

在这种情况下，通配符%指示MySQL匹配所有那些包含1个或多个子串ll的值，如hello，ball，roller等。还可以使用_符号来匹配单个字符而不是一个序列，示范举例如下所示：

```
mysql> SELECT 'MySQL' LIKE '_SQL', 'mSQL' LIKE '_SQL';
+----------------------+--------------------+
| 'MySQL' LIKE '_SQL' | 'mSQL' LIKE '_SQL' |
+----------------------+--------------------+
|                    1 |                  0 |
+----------------------+--------------------+
1 row in set (0.00 sec)
```

默认情况下，比较是以不区分大小写的方式执行的。然而，以前我们注意到，可以添加 BINARY 关键字让 MySQL 执行区分大小写的比较。下面的举例说明了它们之间的差别：

```
mysql> SELECT 'MySQL' LIKE '%sql', BINARY 'MySQL' LIKE '%sql';
+---------------------+---------------------------+
| 'MySQL' LIKE '%sql' | BINARY 'MySQL' LIKE '%sql' |
+---------------------+---------------------------+
|                   1 |                         0 |
+---------------------+---------------------------+
1 row in set (0.01 sec)
```

添加NOT逻辑运算符会使结果取反，返回与指定搜索字符串不匹配的那些记录。示范举例如下所示：

```
mysql> SELECT 'Harry' NOT LIKE '%ry', 'bee' NOT LIKE 'insect';
+------------------------+-----------------------+
| 'Harry' NOT LIKE '%ry' | 'bee' NOT LIKE 'insect' |
+------------------------+-----------------------+
|                      0 |                     1 |
+------------------------+-----------------------+
1 row in set (0.00 sec)
```

REGEXP 运算符允许我们执行更复杂的字符串比较运算，在这里使用 UNIX 正则表达式。如果找到匹配结果，REGEXP 运算返回真值。例如：

```
mysql> SELECT 'William' REGEXP '^Wil?', 'Wendy' REGEXP '^Wil';
+--------------------------+----------------------+
| 'William' REGEXP '^Wil?' | 'Wendy' REGEXP '^Wil' |
+--------------------------+----------------------+
|                        1 |                    0 |
+--------------------------+----------------------+
1 row in set (0.00 sec)

mysql> SELECT 'red' REGEXP 'red|green|blue', 'hammer' REGEXP 'ham$';
+-------------------------------+-----------------------+
| 'red' REGEXP 'red|green|blue' | 'hammer' REGEXP 'ham$' |
+-------------------------------+-----------------------+
|                             1 |                     0 |
+-------------------------------+-----------------------+
1 row in set (0.00 sec)
```

把NOT逻辑运算符添加到REGEXP运算符上将颠倒它的行为，如果找到匹配结果则返回假值：

```
mysql> SELECT 'hammer' NOT REGEXP 'ham$';
+---------------------------+
| 'hammer' NOT REGEXP 'ham$' |
+---------------------------+
|                         1 |
+---------------------------+
1 row in set (0.01 sec)
```

注意 LIKE 的效率

MySQL 编程者往往喜欢使用 LIKE 运算符来解决他们所有的烦恼，因为使用通配符似乎可以充分简化复杂的搜索查询。这样的情形具有某种欺骗性：MySQL 的 LIKE 运算符的确强大有力，并且通过使用几个字符串片断就可以查找记录，但是不过多地考虑查询的结构和数据库的索引来使用它，就会大大降低系统的性能（因为必须搜索整个表才能找到结果）。因此，当使用一个包含 LIKE 运算符的查询时，建议我们保证在查询 WHERE 子句中命名的列是加索引的，并且 WHERE 子句应包含足够多的数据来限制在开始时搜索的记录数目。

寻找模式

正则表达式是什么呢？有些热心的社区也称它为 regex，是一个用于模式匹配和置换的强有力工具。通常它们跟几乎所有的基于 *NIX 的工具联系在一起，包括像 vi 一样的编辑器，像 Perl 和 PHP 一样的脚本语言以及像 awk 和 sed 一样的 Shell 程序[①]。

正则表达式用一组特殊的字符来建立模式，然后这些模式与一个文件中的文本、输入一个应用程序的数据和来自网站上由用户填写的表的输入做比较。根据匹配的情况采取适当的行动或执行适当的程序代码。

正则表达式由标准字符和专门的定义匹配模式的元字符混合组成。例如，正则表达式 ^hell 能够匹配以 hell 开头的单词（hello 和 hellhound，而不是 shell），而正则表达式 Wi$ 能够匹配 Winne，Wimpy，Wilson 和 William，而不与 Wendy 和 Wolf 匹配。

表 6.3 显示了更多的元字符列表以及它们的意义。

表 6.3 在正则表达式中通常使用的元字符

元字符	作用
+	匹配 1 个或更多个前面字符的值
*	匹配 0 个或更多个前面字符的值
?	匹配 0 个或 1 个前面字符的值
.	匹配任意字符
^	匹配字符串的开始部分
$	匹配字符串的末尾部分
\s	匹配单个空白空间字符，包括制表符和换行符
\S	匹配空白空间字符以外的一切字符
\d	匹配 0 到 9 之间的数字
\w	匹配字母、数字和下划线字符
\W	匹配用 \w 不能匹配的任意字符

让我们看一下这些运算符在现实情形中是如何起作用的。考虑下面的 MySQL 表，它保存了关于 7 匹马在两年时间里比赛成绩的数据：

```
mysql> SELECT * FROM racedata;
+--------------+-------+-----+-------+----------+
| name         | total | win | place | unplaced |
+--------------+-------+-----+-------+----------+
| Stormtrooper |    50 |   4 |    13 |        9 |
| VioletDaisy  |     9 |   4 |     5 |        0 |
```

① 摘自 http://www.melonfire.com/community/columns/trog/article.php?id=2 中的文章 "So What's A $#!%% Regular Expression, Anyway?!"。

```
| King William |    33 |  11 |    15 |        4 |
| Remuneration |   127 |  35 |    56 |       22 |
| Pop Princess |    79 |   4 |    14 |       43 |
| Superman     |    43 |   0 |     6 |       11 |
| Baby Boo     |    67 |  56 |    10 |        1 |
+--------------+-------+-----+-------+----------+
7 rows in set (0.00 sec)
```

我们将会看到，可以使用MySQL的比较运算符从这些数据中得到一些非常有趣的结论。例如，可以得到参加过 50 多场比赛的所有马的一个列表：

```
mysql> SELECT name, total FROM racedata WHERE total > 50;
+--------------+-------+
| name         | total |
+--------------+-------+
| Remuneration |   127 |
| Pop Princess |    79 |
| Baby Boo     |    67 |
+--------------+-------+
3 rows in set (0.00 sec)
```

或者可以得到在 10 场或者更多场比赛中获胜的所有马的一个列表：

```
mysql> SELECT name, win FROM racedata WHERE win >= 10;
+--------------+-----+
| name         | win |
+--------------+-----+
| King William |  11 |
| Remuneration |  35 |
| Baby Boo     |  56 |
+--------------+-----+
3 rows in set (0.00 sec)
```

我们能够得到在它所有的比赛中要么获胜要么取得名次的马的一个列表（它们在下一次很可能还要参加比赛）：

```
mysql> SELECT name FROM racedata WHERE win + place = total;
+-------------+
| name        |
+-------------+
| VioletDaisy |
+-------------+
1 row in set (0.00 sec)
```

还可以找出哪些马完成了它们参加的每一场比赛（从而证明了不寻常的一致性）：

```
mysql> SELECT name FROM racedata WHERE win + place + unplaced = total;
+-------------+
| name        |
+-------------+
| VioletDaisy |
| Baby Boo    |
```

```
+-------------+
2 rows in set (0.00 sec)
```

还能够得到具有优秀的比赛记录的马的一个列表，例如，所有那些在它的一半或更多比赛中获胜或者取得名次的马：

```
mysql> SELECT name, win, place, total FROM racedata
WHERE ( win + place ) / total >= 0.5;
+--------------+-----+-------+-------+
| name         | win | place | total |
+--------------+-----+-------+-------+
| VioletDaisy  |   4 |     5 |     9 |
| King William |  11 |    15 |    33 |
| Remuneration |  35 |    56 |   127 |
| Baby Boo     |  56 |    10 |    67 |
+--------------+-----+-------+-------+
4 rows in set (0.00 sec)
```

或者能够得到获胜次数不超过它比赛次数的 10% 的马的一个列表：

```
mysql> SELECT name, win FROM racedata WHERE ( win / total ) < 0.1;
+--------------+-----+
| name         | win |
+--------------+-----+
| Stormtrooper |   4 |
| Pop Princess |   4 |
| Superman     |   0 |
+--------------+-----+
3 rows in set (0.00 sec)
```

我们能够掌握每匹马的参赛历史的细节，例如，查明Superman是否完成了它参加的所有比赛（从而可推测出它没有参加的比赛）：

```
mysql> SELECT name, total, win + place + unplaced FROM racedata
WHERE name = 'Superman';
+----------+-------+-----------------------+
| name     | total | win + place + unplaced |
+----------+-------+-----------------------+
| Superman |    43 |                    17 |
+----------+-------+-----------------------+
1 row in set (0.01 sec)
```

或者能够根据获胜的情况判断 King William 是否具有皇家血统：

```
mysql> SELECT name, total, win FROM racedata
WHERE name = 'King William';
+--------------+-------+-----+
| name         | total | win |
+--------------+-------+-----+
| King William |    33 |  11 |
+--------------+-------+-----+
1 row in set (0.00 sec)
```

6.3　逻辑运算符

MySQL 还有 4 个逻辑运算符，它可以测试 1 个或多个表达式（或表达式集合）的逻辑有效性。含有这些运算符的运算结果总是 1（真）、0（假）或 NULL（不能确定）。

表 6.4 列出了 MySQL 支持的逻辑运算符。

表 6.4　MySQL 中的逻辑运算符

运算符	作用
NOT 或 !	逻辑非
AND 或 &&	逻辑和
OR 或 ||	逻辑或
XOR	逻辑异或

这些运算符中最简单的运算符是NOT运算符，它对跟在它后面的逻辑测试判断取反，把真变为假，把假变为真。在前一节中已经有了几个实例。

```
mysql> SELECT NOT 1, NOT 0, NOT (2=2), NOT (100 > 200);
+-------+-------+-----------+-----------------+
| NOT 1 | NOT 0 | NOT (2=2) | NOT (100 > 200) |
+-------+-------+-----------+-----------------+
|     0 |     1 |         0 |               1 |
+-------+-------+-----------+-----------------+
1 row in set (0.00 sec)
```

AND运算符可以测试两个或更多的值（或表达式求值）的有效性，如果它的所有成分为真并且不是 NULL，它返回真值，否则返回假值。

```
mysql> SELECT (2 = 2) AND (900 < 100), ('a' = 'a') AND ('c' < 'd');
+-------------------------+----------------------------+
| (2 = 2) AND (900 < 100) | ('a' = 'a') AND ('c' < 'd') |
+-------------------------+----------------------------+
|                       0 |                          1 |
+-------------------------+----------------------------+
1 row in set (0.00 sec)
```

在这一点上，一个出人意料的变化是 OR 运算符，如果包含的值或表达式为真并且不是 NULL，它返回真值，否则返回假值。下面的例子演示并且说明了这个运算符和前一个运算符之间的差别：

```
mysql> SELECT (2 = 2) OR (900 < 100), ('a' = 'a') OR ('c' < 'd');
+------------------------+---------------------------+
| (2 = 2) OR (900 < 100) | ('a' = 'a') OR ('c' < 'd') |
+------------------------+---------------------------+
|                      1 |                         1 |
+------------------------+---------------------------+
1 row in set (0.00 sec)
```

MySQL 4.x 和更高版本还包括一个附加的 XOR 运算符，如果它的参数中的一个（不是两个）为真，它返回真值。思考下面的示范举例：

```
mysql> SELECT (1 = 1) XOR (2 = 4), (1 < 2) XOR (9 < 10);
+---------------------+----------------------+
| (1 = 1) XOR (2 = 4) | (1 < 2) XOR (9 < 10) |
+---------------------+----------------------+
|                   1 |                    0 |
+---------------------+----------------------+
1 row in set (0.00 sec)
```

为了说明逻辑运算符在过滤数据时是多么地有用，考虑下面的雇员数据库：

```
mysql> SELECT * FROM employees;
+-----+--------+--------+-----+-----+----------------+---------+
| id  | fname  | lname  | age | sex | department     | country |
+-----+--------+--------+-----+-----+----------------+---------+
|  54 | Doe    | John   |  27 | M   | Engineering    | US      |
| 127 | Jones  | Sue    |  31 | F   | Finance        | UK      |
| 113 | Woo    | David  |  26 | M   | Administration | CN      |
| 175 | Thomas | James  |  34 | M   | Finance        | US      |
| 168 | Kent   | Jane   |  29 | F   | Administration | US      |
|  12 | Kamath | Ravina |  35 | F   | Finance        | IN      |
+-----+--------+--------+-----+-----+----------------+---------+
6 rows in set (0.00 sec)
```

在MySQL逻辑运算符的帮助下，可以组合条件以获得更精确的结果。例如，我们可以得到年龄超过30岁的所有女性雇员的一个列表：

```
mysql> SELECT fname, lname, age FROM employees
WHERE sex = 'F' AND age > 30;
+--------+--------+-----+
| fname  | lname  | age |
+--------+--------+-----+
| Jones  | Sue    |  31 |
| Kamath | Ravina |  35 |
+--------+--------+-----+
2 rows in set (0.00 sec)
```

或者可以得到财务部门所有男性雇员的一个列表：

```
mysql> SELECT fname, lname FROM employees WHERE sex = 'M'
AND department = 'Finance';
+--------+-------+
| fname  | lname |
+--------+-------+
| Thomas | James |
+--------+-------+
1 row in set (0.01 sec)
```

或者甚至可以得到国家为美国或英国的所有雇员：

```
mysql> SELECT fname, lname, country FROM employees
WHERE country = 'US' OR country = 'UK';
+--------+-------+---------+
| fname  | lname | country |
+--------+-------+---------+
| Doe    | John  | US      |
| Jones  | Sue   | UK      |
| Thomas | James | US      |
| Kent   | Jane  | US      |
+--------+-------+---------+
4 rows in set (0.00 sec)
```

还可以得到一个特定的雇员的记录：

```
mysql> SELECT * FROM employees WHERE fname = 'Kent'
AND lname = 'Jane';
+-----+-------+-------+-----+-----+----------------+---------+
| id  | fname | lname | age | sex | department     | country |
+-----+-------+-------+-----+-----+----------------+---------+
| 168 | Kent  | Jane  |  29 | F   | Administration | US      |
+-----+-------+-------+-----+-----+----------------+---------+
1 row in set (0.00 sec)
```

或者可以得到所有年龄超过 25 岁的非美国雇员的一个列表：

```
mysql> SELECT fname, lname, country FROM employees
WHERE age > 25 AND NOT (country = 'US');
+--------+--------+---------+
| fname  | lname  | country |
+--------+--------+---------+
| Jones  | Sue    | UK      |
| Woo    | David  | CN      |
| Kamath | Ravina | IN      |
+--------+--------+---------+
3 rows in set (0.00 sec)
```

6.4　位运算符

MySQL 还包含 6 个专门用于位操作的运算符，见表 6.5。

表 6.5　MySQL 的位运算符

运算符	作用
&	位 AND
|	位 OR
^	位 XOR（异或）
~	位取反
>>	位右移
<<	位左移

|运算符用于执行一个位 OR 操作，而 & 用于执行一个位 AND 操作。示范举例如下所示：

```
mysql> SELECT 16 | 32, 9 | 4;
+---------+-------+
| 16 | 32 | 9 | 4 |
+---------+-------+
|      48 |    13 |
+---------+-------+
1 row in set (0.00 sec)

mysql> SELECT 30 & 10, 8 & 16;
+---------+--------+
| 30 & 10 | 8 & 16 |
+---------+--------+
|      10 |      0 |
+---------+--------+
1 row in set (0.00 sec)
```

还可以分别使用<<和>>运算符向左和向右移动位。思考下面的示范举例：

```
mysql> SELECT 1 << 7, 64 >> 1;
+--------+---------+
| 1 << 7 | 64 >> 1 |
+--------+---------+
|    128 |      32 |
+--------+---------+
1 row in set (0.00 sec)
```

^运算符执行位 XOR 操作：

```
mysql> SELECT 1 ^ 0, 0 ^ 1, 17 ^ 9, 143 ^ 66;
+-------+-------+--------+----------+
| 1 ^ 0 | 0 ^ 1 | 17 ^ 9 | 143 ^ 66 |
+-------+-------+--------+----------+
|     1 |     1 |     24 |      205 |
+-------+-------+--------+----------+
1 row in set (0.00 sec)
```

~运算符执行位取反操作，并返回 64 位整型结果：

```
mysql> SELECT ~18446744073709551614, ~1;
+----------------------+---------------------+
| ~18446744073709551614 | ~1                  |
+----------------------+---------------------+
|                    1 | 18446744073709551614 |
+----------------------+---------------------+
1 row in set (0.01 sec)
```

按规则玩游戏

当对运算符求值时，MySQL 不一定按照它们出现的次序处理它们，而是有它自己的关于哪个运算符比其他运算符有优先级的规则集。看一下下面的列表，它说明了 MySQL 的优先级规则（在同一行中的运算符有相同的优先级）。

- 'OR'
- 'AND'
- 'BETWEEN', 'CASE', 'WHEN', 'THEN', 'ELSE'
- '=', '<=>', '>=', '>', '<=', '<', '<>', 'IS', 'LIKE', 'REGEXP', 'IN'
- '|'
- '&'
- '<<', '>>'
- '-', '+'
- '*', '/', '%'
- '~'
- 'NOT'
- 'BINARY'

如果有疑问，请记住，我们可以（事实上是应该）使用圆括号忽略这些规则，就像本章实例中的一些那样。这样就减少了不明确性，并且保证运算符以我们指定的次序求值。下面的举例示范了使用圆括号时，其结果是怎样与 MySQL 表达式求值的方法不同的：

```
mysql> SELECT 2 + 4 / 3, (2 + 4) / 3;
+-----------+-------------+
| 2 + 4 / 3 | (2 + 4) / 3 |
+-----------+-------------+
|      3.33 |        2.00 |
+-----------+-------------+
1 row in set (0.00 sec)
```

下面的举例演示了在现实世界的运算中的位运算符，它计算来自一个用户提供的IP地址和网络掩码的网络号：

```
mysql> SELECT INET_NTOA(INET_ATON('192.168.1.34') & INET_ATON('255.255.
255.0'));
+-----------------------------------------------------------------+
| INET_NTOA(INET_ATON("192.168.1.34") & INET_ATON("255.255.255.0")) |
+-----------------------------------------------------------------+
| 192.168.1.0                                                     |
+-----------------------------------------------------------------+
1 row in set (0.05 sec)
```

提示：INET_ATON()函数把一个 IP 地址转换成它的数字描述，将在第 7 章中讨论。

6.5 小结

本章使我们更进一步走近 MySQL，详细讲解了在 MySQL 中建立 SELECT 表达式和子句时可用的各种运算符。MySQL 有超过 25 个不同的运算符，用于执行算术运算、比较运算、逻辑运算和位运算；本章通过结合现实世界中的简单实例和演示，研究了每一个运算符以及它们的限制条件和规则。对这些运算符的充分认识以及对它们最合适的用法的理解，将对我们使用 MySQL 的 SQL 语句来检索和更新存储数据非常有用。

第7章　MySQL函数

灵活性作为关系型数据库管理系统（RDBMS）引以为豪的特性，所以MySQL的内建函数使用户能够很容易地对表中的数据进行操作是一点也不奇怪的。惟一令人惊讶的是它有如此多的函数——MySQL包含100多个函数，从简单的数学函数到复杂的比较函数和日期操作函数。如此之多的选择，使MySQL的开发者可以用最少的代码进行复杂的操作，这也是MySQL流行的一个重要原因。

本章将概述MySQL各种各样的内建函数，结合一些代码片断和例子来解释其中一些重要的函数，这些例子解释了如何使用这些函数进行操作，如字符串的比较、日期/时间计算、算术操作、数据转化以及一些其他任务。

这些内建函数可以分为以下几组：

- 数学函数
- 聚合函数
- 字符串操作函数
- 日期和时间操作函数
- 数据加密函数
- 控制流函数
- 格式化函数
- 类型转化函数
- 系统信息函数

下面将详细介绍每类函数。

7.1　数学函数

由于MySQL包含一系列的算术操作，所以关系型数据库管理系统支持很多数学函数。表7.1列出了比较重要的一些。

表7.1　MySQL数学函数

函数	功能
ABS(*x*)	返回*x*的绝对值
ACOS(*x*)	返回*x*（弧度）的反余弦值
ASIN(*x*)	返回*x*（弧度）的反正弦值
ATAN(*x*)	返回*x*（弧度）的反正切值
CEILING(*x*)	返回大于*x*的最小整数值
COS(*x*)	返回*x*（弧度）的余弦值
COT(*x*)	返回*x*（弧度）的余切
DEGREES(*x*)	返回弧度值*x*转化为角度的结果

（续表）

函数	功能
EXP(x)	返回值 e（自然对数的底）的 x 次方
FLOOR(x)	返回小于 x 的最大整数值
GREATEST($x_1, x_2, \cdots, x_n$)	返回集合中最大的值
LEAST($x_1, x_2, \cdots, x_n$)	返回集合中最小的值
LN(x)	返回 x 的自然对数
LOG(x,y)	返回 x 的以 y 为底的对数
MOD(x,y)	返回 x/y 的模（余数）
PI()	返回 pi 的值（圆周率）
POW(x,y)或 POWER(x,y)	返回 x 的 y 次幂
RAND()	返回 0 到 1 内的随机值
RADIANS(x)	返回角度 x 转化为弧度的结果
ROUND(x,y)	返回参数 x 的四舍五入的有 y 位小数的值
SIGN(x)	返回代表数字 x 的符号的值
SQRT(x)	返回 x 的开方
SIN(x)	返回 x（弧度）的正弦值
TAN(x)	返回 x（弧度）的正切值
TRUNCATE(x,y)	返回数字 x 截短为 y 位小数的结果

我们经常使用的函数是那些获得一组数中最大和最小值的函数。考虑下面这个例子，它阐述了 GREATEST()和 LEAST()函数的功能：

```
mysql> SELECT GREATEST(100, 99, 158, 63), GREATEST(1.1, 2.2);
+----------------------------+--------------------+
| GREATEST(100, 99, 158, 63) | GREATEST(1.1, 2.2) |
+----------------------------+--------------------+
|                        158 |                2.2 |
+----------------------------+--------------------+
1 row in set (0.00 sec)

mysql> SELECT LEAST(-2, 5, 99), LEAST(1, 2, GREATEST(1, 2, 3));
+------------------+--------------------------------+
| LEAST(-2, 5, 99) | LEAST(1, 2, GREATEST(1, 2, 3)) |
+------------------+--------------------------------+
|               -2 |                              1 |
+------------------+--------------------------------+
1 row in set (0.00 sec)
```

注释：MySQL 不允许函数名和括号之间有空格。

可以使用 FLOOR()和 CEILING()函数获得接近一个数的最大和最小整数值：

```
mysql> SELECT FLOOR(-75.1), CEILING(-75.1),
FLOOR(99.99), CEILING(99.99);
+--------------+----------------+--------------+----------------+
| FLOOR(-75.1) | CEILING(-75.1) | FLOOR(99.99) | CEILING(99.99) |
+--------------+----------------+--------------+----------------+
|          -76 |            -75 |           99 |            100 |
+--------------+----------------+--------------+----------------+
1 row in set (0.00 sec)
```

或者可以使用ROUND()函数将数四舍五入到最接近的整数值：

```
mysql> SELECT ROUND(10.33), ROUND(64.5000001), ROUND(7.987654533);
+--------------+------------------+-------------------+
| ROUND(10.33) | ROUND(64.5000001) | ROUND(7.987654533) |
+--------------+------------------+-------------------+
|           10 |                65 |                 8 |
+--------------+------------------+-------------------+
1 row in set (0.00 sec)
```

TRUNCATE()函数和ROUND()函数相似但并不相同，它把一个数字截短成为一个指定小数个数的数字：

```
mysql> SELECT TRUNCATE(1.7850927343,3), TRUNCATE(-629.2383873, 10);
+-------------------------+---------------------------+
| TRUNCATE(1.7850927343,3) | TRUNCATE(-629.2383873, 10) |
+-------------------------+---------------------------+
|                   1.785 |            -629.2383873000 |
+-------------------------+---------------------------+
1 row in set (0.00 sec)
```

当我们需要一个数的绝对值时，ABS()函数使用起来非常方便：

```
mysql> SELECT ABS(-765);
+-----------+
| ABS(-765) |
+-----------+
|       765 |
+-----------+
1 row in set (0.03 sec)
```

SIGN()函数返回数字的符号，取决于返回的结果是正数（1）、负数（-1）或者零（0）：

```
mysql> SELECT SIGN(-99), SIGN(99), SIGN(0);
+-----------+----------+---------+
| SIGN(-99) | SIGN(99) | SIGN(0) |
+-----------+----------+---------+
|        -1 |        1 |       0 |
+-----------+----------+---------+
1 row in set (0.03 sec)
```

取　反

有趣的是，我们也可以给TRUNCATE()一个负值作为第二参数——看会发生什么情况：

```
mysql> SELECT TRUNCATE(1785.0927343,-2), TRUNCATE(-629.2383873, -1);
+--------------------------+---------------------------+
| TRUNCATE(1785.0927343,-2) | TRUNCATE(-629.2383873, -1) |
+--------------------------+---------------------------+
|                     1700 |                      -620 |
+--------------------------+---------------------------+
1 row in set (0.00 sec)
```

SQRT()函数返回一个数的平方根：

```
mysql> SELECT SQRT(49), SQRT(17), SQRT(1);
+----------+----------+----------+
| SQRT(49) | SQRT(17) | SQRT(1)  |
+----------+----------+----------+
| 7.000000 | 4.123106 | 1.000000 |
+----------+----------+----------+
1 row in set (0.00 sec)
```

POW()函数使一个数作为另外一个数的指数，并返回结果：

```
mysql> SELECT POW(4,4), POW(10,-2), POW(3,0), POW(0,5);
+------------+------------+----------+----------+
| POW(4,4)   | POW(10,-2) | POW(3,0) | POW(0,5) |
+------------+------------+----------+----------+
| 256.000000 |   0.010000 | 1.000000 | 0.000000 |
+------------+------------+----------+----------+
1 row in set (0.00 sec)
```

LOG()函数返回它的第一个参数的自然对数：

```
mysql> SELECT LOG(4), LOG(1), LOG(0), LOG(2.718282);
+----------+----------+--------+---------------+
| LOG(4)   | LOG(1)   | LOG(0) | LOG(2.718282) |
+----------+----------+--------+---------------+
| 1.386294 | 0.000000 |   NULL |      1.000000 |
+----------+----------+--------+---------------+
1 row in set (0.00 sec)
```

如果指定两个参数，LOG()返回以第二个参数为底的第一个参数的对数：

```
mysql> SELECT LOG(2,2), LOG(56,4), LOG(0,4);
+----------+-----------+----------+
| LOG(2,2) | LOG(56,4) | LOG(0,4) |
+----------+-----------+----------+
| 1.000000 |  0.344391 |     NULL |
+----------+-----------+----------+
1 row in set (0.01 sec)
```

注释：函数 LOG10()和 LOG2()是函数 LOG()的变体，LOG10()返回一个数以 10 为底的对数，LOG2()返回的是一个数以 2 为底的对数。

EXP()函数与 LOG()函数相反，它返回的是以 e 为底，以指定数为指数的结果：

```
mysql> SELECT EXP(9), EXP(1), EXP(0), EXP(-45);
+-------------+----------+----------+----------+
| EXP(9)      | EXP(1)   | EXP(0)   | EXP(-45) |
+-------------+----------+----------+----------+
| 8103.083928 | 2.718282 | 1.000000 | 0.000000 |
+-------------+----------+----------+----------+
1 row in set (0.01 sec)
```

SIN()、COS()和TAN()函数返回一个角度（指定为弧度）的正弦、余弦和正切值：

```
mysql> SELECT SIN(1.5708), COS(1.5708), TAN(RADIANS(45));
+-------------+-------------+-----------------+
| SIN(1.5708) | COS(1.5708) | TAN(RADIANS(45)) |
+-------------+-------------+-----------------+
|    1.000000 |   -0.000004 |        1.000000 |
+-------------+-------------+-----------------+
1 row in set (0.00 sec)
```

ASIN()，ACOS()和ATAN()函数返回一个角度（指定为弧度）的反正弦、反余弦和反正切值：

```
mysql> SELECT ASIN(1), ACOS(1), ATAN(RADIANS(45));
+----------+----------+-------------------+
| ASIN(1)  | ACOS(1)  | ATAN(RADIANS(45)) |
+----------+----------+-------------------+
| 1.570796 | 0.000000 |          0.665774 |
+----------+----------+-------------------+
1 row in set (0.00 sec)
```

如果使用的是角度而不是弧度，可以使用DEGREES()和RADIANS()函数进行转化：

```
mysql> SELECT RADIANS(90), RADIANS(180);
+-----------------+----------------+
| RADIANS(90)     | RADIANS(180)   |
+-----------------+----------------+
| 1.5707963267949 | 3.1415926535898 |
+-----------------+----------------+
1 row in set (0.00 sec)

mysql> SELECT DEGREES(1.57), DEGREES(3.14);
+----------------+----------------+
| DEGREES(1.57)  | DEGREES(3.14)  |
+----------------+----------------+
| 89.954373835539 | 179.90874767108 |
+----------------+----------------+
1 row in set (0.01 sec)
```

BIN()，OCT()和HEX()函数分别返回一个数的二进制、八进制和十六进制值，这个值作为字符串返回：

```
mysql> SELECT BIN(2), OCT(28), HEX(56);
+--------+---------+---------+
| BIN(2) | OCT(28) | HEX(56) |
+--------+---------+---------+
| 10     | 34      | 38      |
+--------+---------+---------+
1 row in set (0.00 sec)
```

最后，PI()函数返回圆周率π的值：

```
mysql> SELECT PI();
+----------+
| PI()     |
+----------+
| 3.141593 |
+----------+
1 row in set (0.00 sec)
```

RAND()函数返回 0 到 1 之间的随机数：

```
mysql> SELECT RAND(), RAND(), RAND();
+------------------+------------------+------------------+
| RAND()           | RAND()           | RAND()           |
+------------------+------------------+------------------+
| 0.97721177709961 | 0.12815972336397 | 0.70916331625466 |
+------------------+------------------+------------------+
1 row in set (0.01 sec)
```

可以通过提供一个参数（种子）使 RAND()随机数生成器生成一个指定的值：

```
mysql> SELECT RAND(17), RAND(10);
+------------------+------------------+
| RAND(17)         | RAND(10)         |
+------------------+------------------+
| 0.40833328422935 | 0.65705152196535 |
+------------------+------------------+
1 row in set (0.00 sec)
```

7.2　聚合函数

MySQL有一组函数是特意为求和或者对表中的数据进行集中概括而设计的。这些函数经常用在包含 GROUP BY 从句的 SELECT 查询中，当然，它们也可以用于无 GROUP 的查询。表 7.2 列出了这些函数。

表 7.2　MySQL 聚合函数

函数	功能
AVG(*col*)	返回指定列的平均值
COUNT(*col*)	返回指定列中非 NULL 值的个数
MIN(*col*)	返回指定列的最小值
MAX(*col*)	返回指定列的最大值
SUM(*col*)	返回指定列的所有值之和
STD(*col*) 或 STDDEV(*col*)	返回指定列的所有值的标准偏差
VARIANCE(*col*)	返回指定列的所有值的标准方差
GROUP_CONCAT(*col*)	返回由属于一组的列值连接组合而成的结果

这一类函数中最经常用到的是COUNT()函数，它计算出结果集中至少包含一个非NULL值的行数：

```
mysql> SELECT COUNT(*) FROM members;
```

```
+----------+
| COUNT(*) |
+----------+
|       45 |
+----------+
1 row in set (0.05 sec)
```

MIN()和MAX()函数返回数字集的最小或者最大值：

```
mysql> SELECT MIN(quantity) FROM inventory;
+---------------+
| MIN(quantity) |
+---------------+
|           100 |
+---------------+
1 row in set (0.00 sec)

mysql> SELECT MAX(return) FROM investments;
+-------------+
| MAX(return) |
+-------------+
|    23760.00 |
+-------------+
1 row in set (0.00 sec)
```

SUM()和AVG()函数返回一组数的总和及平均值：

```
mysql> SELECT SUM(units), AVG(return) FROM mutualfunds;
+---------------+-------------+
| SUM(units)    | AVG(return) |
+---------------+-------------+
|          7610 | 27.056000   |
+---------------+-------------+
1 row in set (0.01 sec)
```

VARIANCE()函数返回一组指定数的方差，而STD()函数返回它们的标准偏差：

```
mysql> SELECT VARIANCE(invoiceAmt) FROM invoices
WHERE customerID = 125;
+----------------------+
| VARIANCE(invoiceAmt) |
+----------------------+
|       1133926.723409 |
+----------------------+
1 row in set (0.22 sec)

mysql> SELECT STD(price) FROM stocks;
+-------------+
| STD(price)  |
+-------------+
| 19.998123   |
+-------------+
1 row in set (0.01 sec)
```

GROUP_CONCAT()函数把一个特定组（通过GROUP BY子句创建）的值连接成一个字符串。例如，如果要把下面的关于用户的表进行分组：

```
+-------+-----------+
| uname | gname     |
+-------+-----------+
| sam   | actors    |
| sue   | musicians |
| james | authors   |
| rita  | authors   |
| sue   | authors   |
| louis | actors    |
| sue   | actors    |
+------+-------+----+
7 rows in set (0.11 sec)
```

我们可以使用GROUP_CONCAT()函数返回相同信息的经过选择的、易读的描述，如下所示：

```
mysql> SELECT gname, GROUP_CONCAT(uname) AS members FROM users_groups
GROUP BY gname;
+------------+-------------------+
| gname      | members           |
+------------+-------------------+
|  authors   | james rita sue    |
|  actors    | sam louis sue     |
|  musicians | sue               |
+------------+-------------------+
3 rows in set (0.17 sec)
```

注释：VARIANCE()和 GROUP_CONCAT()函数都是 MySQL 4.1 的新增函数。

要理解这些函数的功能，考虑下表的例子，它是一个关于学生和成绩的列表：

```
mysql> SELECT * FROM marks;
+----+-------+-------+------+-----+-----+
| id | fname | lname | math | sci | lit |
+----+-------+-------+------+-----+-----+
|  1 | John  | Doe   |   36 |  73 |  34 |
|  2 | Sarah | Short |   23 |  67 |  55 |
|  3 | Joe   | Cool  |   75 |  82 |  89 |
|  4 | Mark  | Wumba |   58 |  72 |  15 |
|  5 | Tom   | Thumb |    0 |  12 |   0 |
+----+-------+-------+------+-----+-----+
5 rows in set (0.00 sec)
```

现在可以使用聚合函数来获得表中记录的信息。例如，使用COUNT()函数可以获得参加考试的学生数：

```
mysql> SELECT COUNT(*) FROM marks;
+----------+
| COUNT(*) |
```

```
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
```

或者使用COUNT()函数显示任一科成绩低于25分的学生数：

```
mysql> SELECT COUNT(*) FROM marks
WHERE (math < 25 OR sci < 25 OR lit < 25);
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
1 row in set (0.00 sec)
```

可以使用MIN()和MAX()函数获得不同科目的最高和最低成绩：

```
mysql> SELECT MAX(math), MAX(lit), MIN(sci) FROM marks;
+-----------+----------+----------+
| MAX(math) | MAX(lit) | MIN(sci) |
+-----------+----------+----------+
|        75 |       89 |       12 |
+-----------+----------+----------+
1 row in set (0.01 sec)
```

使用AVG()函数可计算每一科的平均成绩：

```
mysql> SELECT AVG(math), AVG(sci), AVG(lit) FROM marks;
+-----------+----------+----------+
| AVG(math) | AVG(sci) | AVG(lit) |
+-----------+----------+----------+
|   38.4000 |  61.2000 |  38.6000 |
+-----------+----------+----------+
1 row in set (0.00 sec)
```

7.3 字符串函数

因为MySQL数据库不仅包含数字数据，还包含字符串，因此MySQL有一套为字符串操作而设计的函数。表7.3列出了其中重要的一些。

表7.3 MySQL字符串函数

函数	功能
ASCII(char)	返回字符的ASCII码值
BIT_LENGTH(str)	返回字符串的比特长度
CHAR($x_1, x_2, \cdots, x_n$)	返回$x_1, x_2, \cdots, x_n$所代表的ASCII码值给出的字符组成的字符串
CONCAT($s_1, s_2, \cdots, s_n$)	将$s_1, s_2, \cdots, s_n$连接成字符串
CONCAT_WS(sep, $s_1, s_2, \cdots, s_n$)	将$s_1, s_2, \cdots, s_n$连接成字符串，并用sep字符间隔
INSERT(str, x, y, instr)	将字符串str从第x位置开始，y个字符长的子串替换为字符串instr，返回结果

（续表）

函数	功能
FIELD(str, s_1, s_2,…, s_n)	分析列表 s_1, s_2, …, s_n，如果发现 str，则返回 str 在列表 s_1,s_2,…, s_n 中的索引
FIND_IN_SET(str, list)	分析逗号分隔的 list 列表，如果发现 str，返回 str 在 list 中的位置
LCASE(str) 或 LOWER(str)	返回将字符串 str 中所有字符改变为小写后的结果
LEFT(str, *x*)	返回字符串 str 中最左边的 *x* 个字符
LENGTH(s)	返回字符串 str 中的字符数
LPAD(str, *n*, pad)	用字符串 pad 对 str 进行左边填补直至达到 *n* 个字符长度
LTRIM(str)	从字符串 str 中切掉开头的空格
ORD(char)	返回字符 char 的多字节安全的编码
POSITION(substr, str)	返回子串 substr 在字符串 str 中第一次出现的位置
QUOTE(str)	用反斜杠转义 str 中的单引号
REPEAT(str, *x*)	返回字符串 str 重复 *x* 次的结果
REPLACE(str, srchstr, rplcstr)	用字符串rplcstr替换字符串str中所有出现的字符串srchstr
REVERSE(str)	返回颠倒字符串 str 的结果
RIGHT(str, *x*)	返回字符串 str 中最右边的 *x* 个字符
RPAD(str, *x*, pad)	用字符串 pad 对 str 进行右边填补直至达到 *n* 个字符长度
RTRIM(str)	去除字符串 str 尾部的空格
STRCMP(s_1, s_2)	比较字符串 s_1 和 s_2
SUBSTRING(str, *x*, *y*) 或 MID(str, *x*, *y*)	返回从字符串 str 的 *x* 位置起 *y* 个字符长度的子串
TRIM(str)	去除字符串首部和尾部的所有空格
UCASE(str)或 UPPER(str)	返回将字符串 str 中所有字符转变为大写后的结果

通过 LENGTH()函数可以获得一个字符串的长度：

```
mysql> SELECT LENGTH('abracadabra');
+-----------------------+
| LENGTH('abracadabra') |
+-----------------------+
|                    11 |
+-----------------------+
1 row in set (0.00 sec)
```

BIT_LENGTH()函数返回的是字符串的比特长度而不是字符长度：

```
mysql> SELECT BIT_LENGTH('abracadabra');
+---------------------------+
| BIT_LENGTH('abracadabra') |
+---------------------------+
|                        88 |
+---------------------------+
1 row in set (0.00 sec)
```

使用RPAD()或者LPAD()函数可以分别从右边或者左边对字符串进行填补。字符串经常用空格进行填补，不过，MySQL 可以在 RPAD()和 LPAD()函数中指定一个自定义值作为第三个参数对字符串进行填补：

```
mysql> SELECT RPAD('red alert', 17, '!'), LPAD('welcome', 12, '* ');
+----------------------------+--------------------------+
| RPAD('red alert', 17, '!') | LPAD('welcome', 12, '* ') |
+----------------------------+--------------------------+
| red alert!!!!!!!!          | * * *welcome             |
+----------------------------+--------------------------+
1 row in set (0.00 sec)
```

我们可以通过LTRIM()和RTRIM()函数产生与填补相反的效果，它们分别去除一个字符串首部和尾部的空格：

```
mysql> SELECT RTRIM("right hand    "), LTRIM("    left eye");
+-------------------------+-----------------------+
| RTRIM("right hand    ") | LTRIM("    left eye") |
+-------------------------+-----------------------+
| right hand              | left eye              |
+-------------------------+-----------------------+
1 row in set (0.00 sec)
```

空格问题

谈到空格，MySQL 可以通过 SPACE()函数生成一个只包含空格的字符串：

```
mysql> SELECT SPACE(4);
+----------+
| SPACE(4) |
+----------+
|          |
+----------+
1 row in set (0.03 sec)
```

另外一个选择就是TRIM()函数，它可以让我们在剪切值时指定去除格式，还可以决定是从字符串的首部剪切、尾部剪切，还是两边都剪切：

```
mysql> SELECT TRIM('    red alert    ');
+---------------------------+
| trim('    red alert    ') |
+---------------------------+
| red alert                 |
+---------------------------+
1 row in set (0.00 sec)

mysql> SELECT TRIM(LEADING '!' FROM '!!!!!ERROR!!!');
+----------------------------------------+
| TRIM(LEADING '!' FROM '!!!!!ERROR!!!') |
+----------------------------------------+
| ERROR!!!                               |
+----------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT TRIM(BOTH '.' FROM '...welcome...');
```

```
+-----------------------------------+
| TRIM(BOTH '.' FROM '...welcome...') |
+-----------------------------------+
| welcome                           |
+-----------------------------------+
1 row in set (0.00 sec)
```

使用LEFT()和RIGHT()函数可以获得字符串的指定部分，它将从字符串的左边或右边返回字符：

```
mysql> SELECT LEFT('Hello', 4), RIGHT('Welcome to the fundraiser', 6);
+-----------------+-------------------------------------+
| LEFT('Hello', 4) | RIGHT('Welcome to the fundraiser', 6) |
+-----------------+-------------------------------------+
| Hell            | raiser                              |
+-----------------+-------------------------------------+
1 row in set (0.01 sec)
```

想得到一个字符串的中间部分吗？可以结合使用 LEFT()和 RIGHT()函数：

```
mysql> SELECT LEFT(RIGHT('market', 5), 3);
+---------------------------+
| LEFT(RIGHT('market', 5), 3) |
+---------------------------+
| ark                       |
+---------------------------+
1 row in set (0.01 sec)
```

也可以使用比较简单的 SUBSTRING()函数，它允许指定一个字符串的长度和起始位置：

```
mysql> SELECT SUBSTRING('market', 2, 3);
+--------------------------+
| SUBSTRING('market', 2, 3) |
+--------------------------+
| ark                      |
+--------------------------+
1 row in set (0.03 sec)
```

CONCAT()函数将提供的参数连接成一个字符串：

```
mysql> SELECT CONCAT('Clark', 'Kent');
+-------------------------+
| CONCAT('Clark', 'Kent') |
+-------------------------+
| ClarkKent               |
+-------------------------+
1 row in set (0.01 sec)
```

如果 CONCAT()的参数是空的，则结果也是空的：

```
mysql> SELECT CONCAT('Clark', 'Kent', NULL),
CONCAT('Clark', 'Kent', 'NULL');
```

```
+-------------------------------+-------------------------------+
| CONCAT('Clark', 'Kent', NULL) | CONCAT('Clark', 'Kent', 'NULL') |
+-------------------------------+-------------------------------+
| NULL                          | ClarkKentNULL                 |
+-------------------------------+-------------------------------+
1 row in set (0.00 sec)
```

可以通过相似（但并不完全相同）的CONCAT_WS()函数，利用指定的分隔符结合字符串，它允许我们指定第一个参数作为字符串的分隔符。例子如下：

```
mysql> SELECT CONCAT_WS(' ', 'Clark', 'Kent');
+--------------------------------+
| CONCAT_WS(' ', 'Clark', 'Kent') |
+--------------------------------+
| Clark Kent                     |
+--------------------------------+
1 row in set (0.00 sec)
```

下面是另外一个例子：

```
mysql> SELECT CONCAT_WS(', ', CONCAT_WS(' ', 'Clark', 'Kent'), 'Superman');
+-------------------------------------------------------------+
| CONCAT_WS(', ', CONCAT_WS(' ', 'Clark', 'Kent'), 'Superman') |
+-------------------------------------------------------------+
| Clark Kent, Superman                                        |
+-------------------------------------------------------------+
1 row in set (0.01 sec)
```

可以使用LOCATE()函数检验一个字符串是否包含指定的子串，它返回指定字符串中子串第一次出现的位置（如果存在）：

```
mysql> SELECT LOCATE('me', 'Robin Hood and his merry men');
+---------------------------------------------+
| LOCATE('me', 'Robin Hood and his merry men') |
+---------------------------------------------+
|                                          20 |
+---------------------------------------------+
1 row in set (0.02 sec)
```

也可以使用FIND_IN_SET()函数查找在一个大的集合中是否存在指定的字符串。作为FIND_IN_SET()的第二个参数的字符串集合必须包含一系列用逗号分开的字符串值，如果存在这个值，该函数将返回相对应值的位置，位置从1开始计算。如：

```
mysql> SELECT FIND_IN_SET('Ross',
'Rachel,Phoebe,Monica,Ross,Joey,Chandler');
+-----------------------------------------------------------------+
| FIND_IN_SET('Ross', 'Rachel,Phoebe,Monica,Ross,Joey,Chandler') |
+-----------------------------------------------------------------+
|                                                               4 |
+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

还可以使用STRCMP()函数比较字符串，如果两个参数相同，则返回0，如果第一个大于第二个则返回1，否则返回-1（字符串比较是不区分大小写的，除非指定BINARY关键字或者涉及到的值是BLOB）。如：

```
mysql> SELECT STRCMP('y', 'z'), STRCMP('e', BINARY 'E');
+-----------------+-------------------------+
| STRCMP('y', 'z') | STRCMP('e', BINARY 'E') |
+-----------------+-------------------------+
|               -1 |                       1 |
+-----------------+-------------------------+
1 row in set (0.00 sec)
```

REPLACE()函数可以搜索一个字符串的指定子串，然后用另一个子串代替：

```
mysql> SELECT REPLACE('I am Joe Cool', 'Cool', 'Camel');
+-----------------------------------------+
| REPLACE('I am Joe Cool', 'Cool', 'Camel') |
+-----------------------------------------+
| I am Joe Camel                          |
+-----------------------------------------+
1 row in set (0.02 sec)
```

INSERT()函数可以把一个字符串的指定部分（定义了起始位置和长度）替换为一个新值：

```
mysql> SELECT INSERT('I am Joe Cool', 10, 4, 'Camel');
+---------------------------------------+
| INSERT('I am Joe Cool', 10, 4, 'Camel') |
+---------------------------------------+
| I am Joe Camel                        |
+---------------------------------------+
1 row in set (0.01 sec)
```

REPEAT()函数用于重复字符串：

```
mysql> SELECT REPEAT('ha ', 6);
+--------------------+
| REPEAT('ha ', 6)   |
+--------------------+
| ha ha ha ha ha ha  |
+--------------------+
1 row in set (0.02 sec)
```

可以使用REVERSE()函数颠倒字符串：

```
mysql> SELECT REVERSE(REPEAT('ha ', 6));
+---------------------------+
| REVERSE(REPEAT('ha ', 6)) |
+---------------------------+
|  ah ah ah ah ah ah        |
+---------------------------+
1 row in set (0.00 sec)
```

可以使用UCASE()函数把字符串的字符转变为大写，使用LCASE()函数把字符串的字符转变为小写：

```
mysql> SELECT UCASE('Little Bo Peep'), LCASE('DotCom');
+-------------------------+-----------------+
| UCASE('Little Bo Peep') | LCASE('DotCom') |
+-------------------------+-----------------+
| LITTLE BO PEEP          | dotcom          |
+-------------------------+-----------------+
1 row in set (0.00 sec)
```

ASCII()函数返回指定字符的 ASCII 值：

```
mysql> SELECT ASCII('A'), ASCII('a');
+------------+------------+
| ASCII('A') | ASCII('a') |
+------------+------------+
|         65 |         97 |
+------------+------------+
1 row in set (0.00 sec)
```

ORD()函数返回指定字符的数字编码［对于多字节的字符串，经常使用这个函数替换ASCII()函数］。

```
mysql> SELECT ORD('À');
+----------+
| ORD('À') |
+----------+
|      192 |
+----------+
1 row in set (0.01 sec)
```

7.4 日期和时间函数

因为MySQL有很多日期和时间数据类型，所以RDBMS有相当多的操作日期和时间的函数是丝毫不奇怪的。MySQL的日期和时间函数比其他RDBMS的日期和时间函数更通用，表7.4阐述了这个事实，列出了其中比较重要的日期和时间函数。

表 7.4 MySQL 日期和时间函数

函数	功能
CURDATE()或 CURRENT_DATE()	返回当前的日期
CURTIME()或 CURRENT_TIME()	返回当前的时间
DATE_ADD(date, INTERVAL int keyword)	返回日期date加上间隔时间int的结果（int必须按照关键字进行格式化）
DATE_FORMAT(date, fmt)	依照指定的 fmt 格式格式化日期 date 值
DATE_SUB(date, INTERVAL int keyword)	返回日期date减去间隔时间int的结果（int必须按照关键字进行格式化）
DAYOFWEEK(date)	返回 date 所代表的一星期中的第几天（1~7）
DATOFMONTH(date)	返回 date 是一个月的第几天（1~31）

（续表）

函数	功能
DAYOFYEAR(date)	返回 date 是一年的第几天（1~366）
DAYNAME(date)	返回 date 的星期名
EXTRACT(keyword FROM date)	返回日期 date 的指定部分，即 keyword 指定的
FROM_DAYS(*x*)	返回一个日期，它是由年份 0 加上 *x* 天产生的
FROM_UNIXTIME(ts, fmt)	根据指定的 fmt 格式，格式化 UNIX 时间戳 ts
HOUR(time)	返回 time 的小时值（0~23）
MINUTE(time)	返回 time 的分钟值（0~59）
MONTH(date)	返回 time 的月份值（1~12）
MONTHNAME(date)	返回 date 的月份名
NOW()	返回当前的日期和时间
QUARTER(date)	返回 date 在一年中的季度（1~4）
PERIOD_ADD(date, mon)	返回日期 date 增加 mon 月份的结果
PERIOD_DIFF(date1, date2)	返回日期 date1 和 date2 相差的月份
SECOND(time)	返回 time 的秒值（0~59）
SEC_TO_TIME(*x*)	把秒值 *x* 转换为易读的时间值
TIME_FORMAT(time, fmt)	依照指定的 fmt 格式格式化时间 time 值
TIME_TO_SEC(time)	将时间 time 转化为秒数
TO_DAYS(date)	返回从 0 年到日期 date 的天数
UNIX_TIMESTAMP(date)	返回日期 date 所代表的 UNIX 时间戳
WEEK(date)	返回日期 date 为一年中的第几周（0~53）
YEAR(date)	返回日期 date 的年份（1000~9999）

和平时一样，我们从基础开始，考虑如何获得当前的日期和时间。这将通过NOW()函数完成，它将以 YYYY-MM-DD HH:MM:SS 的格式返回当前的日期和时间：

```
mysql> SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2003-05-15 11:13:03 |
+---------------------+
1 row in set (0.01 sec)
```

CURTIME()和CURDATE()函数比NOW()函数更为具体化，它们分别返回的是当前的时间和日期：

```
mysql> SELECT CURTIME(), CURDATE();
+-----------+------------+
| CURTIME() | CURDATE()  |
+-----------+------------+
| 11:14:38  | 2003-05-15 |
+-----------+------------+
1 row in set (0.00 sec)
```

通过UNIX_TIMESTAMP()函数可以获得UNIX时间戳（1970-01-01 00:00:00和当前时间之间的秒数）。如果没有提供参数，时间戳返回的将是和当前时间有关的值；如果提供了有效的日期作为参数，时间戳将返回和该日期相关的值。

```
mysql> SELECT UNIX_TIMESTAMP(), UNIX_TIMESTAMP(20040604);
+------------------+-------------------------+
| UNIX_TIMESTAMP() | UNIX_TIMESTAMP(20040604) |
+------------------+-------------------------+
|       1052977855 |               1086287400 |
+------------------+-------------------------+
1 row in set (0.00 sec)
```

可以使用相反的FROM_UNIXTIME()函数把UNIX时间戳转换为MySQL易读的日期和时间，这个函数接受一个UNIX时间戳，返回一个日期值：

```
mysql> SELECT FROM_UNIXTIME(1052977855);
+---------------------------+
| FROM_UNIXTIME(1052977855) |
+---------------------------+
| 2003-05-15 11:20:55       |
+---------------------------+
1 row in set (0.01 sec)
```

YEAR()函数分析一个日期值并且返回其中关于年的部分：

```
mysql> SELECT YEAR(20030415012345), YEAR('1967-09-21');
+----------------------+--------------------+
| YEAR(20030415012345) | YEAR('1967-09-21') |
+----------------------+--------------------+
|                 2003 |               1967 |
+----------------------+--------------------+
1 row in set (0.00 sec)
```

MONTH()和MONTHNAME()函数分别以数值和字符串的格式返回月的部分：

```
mysql> SELECT MONTH(20030415012345);
+-----------------------+
| MONTH(20030415012345) |
+-----------------------+
|                     4 |
+-----------------------+
1 row in set (0.00 sec)

mysql> SELECT MONTHNAME('1967-09-21'), MONTHNAME(20010604);
+-------------------------+---------------------+
| MONTHNAME('1967-09-21') | MONTHNAME(20010604) |
+-------------------------+---------------------+
| September               | June                |
+-------------------------+---------------------+
1 row in set (0.00 sec)
```

DAYOFYEAR()，DAYOFWEEK()和DAYOFMONTH()函数分别返回这一天在一年、一星期，以及一个月中的序数：

```
mysql> SELECT DAYOFYEAR(20030604), DAYOFMONTH('2003-02-14');
+---------------------+-------------------------+
| DAYOFYEAR(20030604) | DAYOFMONTH('2003-02-14') |
```

```
+---------------------+-------------------------+
|                 155 |                      14 |
+---------------------+-------------------------+
1 row in set (0.01 sec)

mysql> SELECT DAYOFWEEK(20020503);
+---------------------+
| DAYOFWEEK(20020503) |
+---------------------+
|                   6 |
+---------------------+
1 row in set (0.01 sec)
```

和MONTHNAME()函数一样，可以使用DAYNAME()函数获得所给出的有效日期代表的星期名，这个星期名用字符串表示：

```
mysql> SELECT DAYNAME('2003-02-14');
+------------------------+
| DAYNAME('2003-02-14') |
+------------------------+
| Friday                 |
+------------------------+
1 row in set (0.00 sec)
```

WEEK()函数返回指定的日期是一年的第几个星期，而YEARWEEK()函数返回指定的日期是哪一年的哪一星期：

```
mysql> SELECT WEEK('2004-03-01'), YEARWEEK(20040301);
+--------------------+--------------------+
| WEEK('2004-03-01') | YEARWEEK(20040301) |
+--------------------+--------------------+
|                  9 |             200409 |
+--------------------+--------------------+
1 row in set (0.00 sec)
```

HOUR()，MINUTE()和SECOND()函数分析时间值，并且分别返回小时、分钟和秒的部分：

```
mysql> SELECT HOUR(182300), MINUTE('14:56'), SECOND(123411);
+--------------+----------------+----------------+
| HOUR(182300) | MINUTE('14:56') | SECOND(123411) |
+--------------+----------------+----------------+
|           18 |             56 |             11 |
+--------------+----------------+----------------+
1 row in set (0.00 sec)
```

TO_DAYS()函数返回 0 年到指定日期之间的天数：

```
mysql> SELECT TO_DAYS(20000101), TO_DAYS(15820101);
+-------------------+-------------------+
| TO_DAYS(20000101) | TO_DAYS(15820101) |
+-------------------+-------------------+
|            730485 |            577814 |
+-------------------+-------------------+
1 row in set (0.00 sec)
```

FROM_DAYS()函数返回一个日期值，它是从0年开始，根据给定天数产生的：

```
mysql> SELECT FROM_DAYS(730485), FROM_DAYS(577814);
+-------------------+-------------------+
| FROM_DAYS(730485) | FROM_DAYS(577814) |
+-------------------+-------------------+
| 2000-01-01        | 1582-01-01        |
+-------------------+-------------------+
1 row in set (0.00 sec)
```

感谢上帝，这是星期日

由WEEK()和YEARWEEK()返回的值通常是在0到53之间，但是，我们可以把它修改到1到54之间，而且还可以通过选择函数的第二个参数来决定一个星期是从周日开始还是从周一开始。

SEC_TO_TIME()函数把秒数转化为一个易读的时间表示：

```
mysql> SELECT SEC_TO_TIME(60), SEC_TO_TIME(3720);
+-----------------+-------------------+
| SEC_TO_TIME(60) | SEC_TO_TIME(3720) |
+-----------------+-------------------+
| 00:01:00        | 01:02:00          |
+-----------------+-------------------+
1 row in set (0.00 sec)
```

TIME_TO_SEC()函数把一个时间值转换为秒数：

```
mysql> SELECT TIME_TO_SEC(000100), TIME_TO_SEC('24:01:10');
+---------------------+--------------------------+
| TIME_TO_SEC(000100) | TIME_TO_SEC('24:01:10') |
+---------------------+--------------------------+
|                  60 |                    86470 |
+---------------------+--------------------------+
1 row in set (0.00 sec)
```

除了可以分析日期和时间值的函数之外，MySQL还有一些有趣的函数，它们可以对日期和时间进行算术操作。其中最重要的是DATE_ADD()和DATE_SUB()函数，它们分别用来增加和减少日期值。

这两个函数的第一个参数都是开始日期值，以YYYY-MM-DD HH:MM:SS或者YYYYMMDDHHMMSS格式表示。第二个参数由三部分组成：关键字INTERVAL，用来从初始值加上或者减去的间隔值，以及用来指定间隔值格式的关键字。间隔值和后边的关键字必须与表7.5显示的一致。

表7.5 与DATE_ADD()和DATE_SUB()一起使用的关键字和格式

关键字	间隔值的格式
DAY	日期
DAY_HOUR	日期：小时
DAY_MINUTE	日期：小时：分钟
DAY_SECOND	日期：小时：分钟：秒
HOUR	小时

（续表）

关键字	间隔值的格式
HOUR_MINUTE	小时：分钟
HOUR_SECOND	小时：分钟：秒
MINUTE	分钟
MINUTE_SECOND	分钟：秒
MONTH	月
SECOND	秒
YEAR	年
YEAR_MONTH	年-月

一些例子可以更清楚阐明这一点。第一个例子是在初始的日期加上 6 个月并返回结果：

```
mysql> SELECT DATE_ADD('2004-01-01 00:00:00', INTERVAL 6 MONTH);
+-----------------------------------------------------+
| DATE_ADD('2004-01-01 00:00:00', INTERVAL 6 MONTH) |
+-----------------------------------------------------+
| 2004-07-01 00:00:00                                 |
+-----------------------------------------------------+
1 row in set (0.04 sec)
```

下一个例子向初始值增加 12 天 3 小时 45 分钟，然后显示结果时间戳：

```
mysql> SELECT DATE_ADD('2004-03-14 12:20:00',
INTERVAL '12 03:45' DAY_MINUTE);
+-----------------------------------------------------------------+
| DATE_ADD('2004-03-14 12:20:00', INTERVAL '12 03:45' DAY_MINUTE) |
+-----------------------------------------------------------------+
| 2004-03-26 16:05:00                                             |
+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

接下来的例子使用 DATE_SUB()函数从一个日期值减去 8 个小时：

```
mysql> SELECT DATE_SUB('2004-01-01 07:20:00', INTERVAL 8 HOUR);
+-----------------------------------------------------+
| DATE_SUB('2004-01-01 07:20:00', INTERVAL 8 HOUR) |
+-----------------------------------------------------+
| 2003-12-31 23:20:00                                 |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

这个例子从同样的日期值减去 13 年 4 个月：

```
mysql> SELECT DATE_SUB('2004-01-01 07:20:00',
INTERVAL '13-4' YEAR_MONTH);
+-----------------------------------------------------------------+
| DATE_SUB('2004-01-01 07:20:00', INTERVAL '13-4' YEAR_MONTH) |
+-----------------------------------------------------------------+
| 1990-09-01 07:20:00                                             |
+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

使用和DATE_ADD()与DATE_SUB()同样的关键字，EXTRACT()函数获得一个日期的指定部分：

```
mysql> SELECT EXTRACT(YEAR_MONTH FROM '2002-03-15');
+-------------------------------------+
| EXTRACT(YEAR_MONTH FROM '2002-03-15') |
+-------------------------------------+
|                                200203 |
+-------------------------------------+
1 row in set (0.08 sec)

mysql> SELECT EXTRACT(DAY_SECOND FROM '2002-03-15 16:45:07');
+-----------------------------------------------+
| EXTRACT(DAY_SECOND FROM '2002-03-15 16:45:07') |
+-----------------------------------------------+
|                                      15164507 |
+-----------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT EXTRACT(HOUR_MINUTE FROM '2002-03-15 16:45:07');
+-------------------------------------------------+
| EXTRACT(HOUR_MINUTE FROM '2002-03-15 16:45:07') |
+-------------------------------------------------+
|                                            1645 |
+-------------------------------------------------+
1 row in set (0.00 sec)
```

PERIOD_DIFF()函数返回两个日期值之间的差值（月数）：

```
mysql> SELECT PERIOD_DIFF(200302, 199802);
+-----------------------------+
| PERIOD_DIFF(200302, 199802) |
+-----------------------------+
|                          60 |
+-----------------------------+
1 row in set (0.00 sec)
```

当需要计算两个日期间的间隔时，这个函数会很方便。为了阐明这一点，考虑下面的例子，一个包含用户名字和出生日期的表：

```
mysql> SELECT * FROM birthdays;
+-------+------------+
| name  | dob        |
+-------+------------+
| raoul | 1978-06-04 |
| luis  | 1965-11-17 |
| larry | 1971-08-19 |
| moe   | 1992-01-23 |
+-------+------------+
4 rows in set (0.00 sec)
```

可以使用PERIOD_DIFF()函数获得每个用户的当前年龄，用年来表示：

```
mysql> SELECT name, dob,
ROUND(PERIOD_DIFF(DATE_FORMAT(NOW(), '%Y%m'),
DATE_FORMAT(dob, '%Y%m')) / 12, 1) AS age FROM birthdays;
+-------+------------+------+
| name  | dob        | age  |
+-------+------------+------+
| raoul | 1978-06-04 | 24.9 |
| luis  | 1965-11-17 | 37.5 |
| larry | 1971-08-19 | 31.7 |
| moe   | 1992-01-23 | 11.3 |
+-------+------------+------+
4 rows in set (0.00 sec)
```

另外一个选择是使用TO_DAYS()函数得到相同的结果，如下所示：

```
mysql> SELECT name, dob,
(TO_DAYS(NOW()) - TO_DAYS(dob)) / 365 AS age FROM birthdays;
+-------+------------+-------+
| name  | dob        | age   |
+-------+------------+-------+
| raoul | 1978-06-04 | 24.96 |
| luis  | 1965-11-17 | 37.52 |
| larry | 1971-08-19 | 31.76 |
| moe   | 1992-01-23 | 11.32 |
+-------+------------+-------+
4 rows in set (0.00 sec)
```

下面是另外一个例子——一个包含成员列表、成员借出的电影以及电影借出和归还日期的图书馆数据库：

```
mysql> SELECT * FROM library;
+---------+-------------------------+----------------+---------------+
| member  | title                   | check_out_date | check_in_date |
+---------+-------------------------+----------------+---------------+
| joe     | The Phantom Menace      | 2002-09-11     | 2002-09-14    |
| john    | Cape Fear               | 2002-07-15     | 2002-07-21    |
| johann  | Casper                  | 2003-01-01     | 2003-01-05    |
| mark    | The Godfather           | 2000-08-17     | 2000-09-16    |
| jthomas | Who Killed Roger Rabbit? | 2003-11-17     | 2003-11-25    |
+---------+-------------------------+----------------+---------------+
5 rows in set (0.04 sec)
```

现在可以查出每个电影借出多长时间：

```
mysql> SELECT member,
TO_DAYS(check_in_date) - TO_DAYS(check_out_date)
AS out_period FROM library;
+---------+------------+
| member  | out_period |
+---------+------------+
| joe     |          3 |
| john    |          6 |
```

```
| johann  |          4 |
| mark    |         30 |
| jthomas |          8 |
+---------+------------+
5 rows in set (0.00 sec)
```

假设一部电影可以借出的最长时间为4天，我们可以找到哪个成员借出电影的时间超过了最长借出时间（而且需要按超出时间罚款）：

```
mysql> SELECT member FROM library
WHERE TO_DAYS(check_in_date) - TO_DAYS(check_out_date) > 4;
+---------+
| member  |
+---------+
| john    |
| mark    |
| jthomas |
+---------+
3 rows in set (0.02 sec)
```

7.5 加密函数

MySQL 特意设计了一些函数对数据进行加密，表 7.6 列出了这些函数。

表 7.6 MySQL 加密函数

函数	功能
AES_ENCRYPT(str, key)	返回用密钥 key 对字符串 str 利用高级加密标准算法加密后的结果
AES_DECRYPT(str, key)	返回用密钥 key 对字符串 str 利用高级加密标准算法解密后的结果
DECODE(str, key)	使用 key 作为密钥解密加密字符串 str
ENCRYPT(str, salt)	使用 UNIX crypt()函数，用关键词 salt 加密字符串 str
ENCODE(str, key)	使用 key 作为密钥加密字符串 str
MD5()	计算字符串 str 的 MD5 校验和
PASSWORD(str)	返回字符串 str 的加密版本
SHA()	计算字符串 str 的安全散列算法（SHA）校验和

PASSWORD()函数用来创建一个经过加密的密码字符串，它适合于插入到MySQL的安全系统。这个加密过程是不可逆转的，和UNIX密码加密过程使用不同的算法。它主要用于MySQL的认证系统，而且 MySQL 手册建议使用 ENCODE()/DECODE()或者 AES_ENCRYPT()/AES_DECRYPT()函数而不是用户应用程序。

```
mysql> SELECT PASSWORD("secret");
+-------------------+
| PASSWORD("secret") |
+-------------------+
| 428567f408994404   |
+-------------------+
1 row in set (0.02 sec)
```

如果愿意的话，可以通过ENCRYPT()函数使用UNIX crypt()系统加密字符串，ENCRYPT()函数接受要加密的字符串和（可选的）用于加密过程的salt（一个可以惟一确定口令的字符串，就像钥匙一样）。如：

```
mysql> SELECT ENCRYPT('open sesame', 'abc');
+-------------------------------+
| ENCRYPT('open sesame', 'abc') |
+-------------------------------+
| ab/G8gtZdMwak                 |
+-------------------------------+
1 row in set (0.00 sec)
```

还可以使用ENCODE()和DECODE()函数来加密和解密字符串。ENCODE()函数有两个参数：被加密的字符串和作为加密基础的密钥：

```
mysql> INSERT INTO users
VALUES ('', 'john', ENCODE('abracadabra', 'secret_key'));
Query OK, 1 row affected (0.00 sec)
```

加密提示

只有当操作系统支持crypt()系统调用时，ENCRYPT()函数才可以工作。如果操作系统不支持它，例如Windows系列，将会导致MySQL在ENCRYPT()函数的例子中出现NULL结果。

调用ENCODE()的结果是一个二进制字符串，它以BLOB类型存储。继续前面的例子，如下所示：

```
mysql> SELECT * FROM users;
+----+-------+-------------+
| id | uname | upass       |
+----+-------+-------------+
|  2 | john  | ³q[|èBgØùM+ |
+----+-------+-------------
```

使用正确的密钥通过DECODE()函数对加密后的值进行解密：

```
mysql> SELECT id, uname, DECODE(upass, 'secret_key') FROM users;
+----+-------+-----------------------------+
| id | uname | DECODE(upass, 'secret_key') |
+----+-------+-----------------------------+
|  2 | john  | abracadabra                 |
+----+-------+-----------------------------+
1 row in set (0.01 sec)
```

比ENCODE()和DECODE()（加密程度相对比较弱）更安全的选择是AES_ENCRYPT()和AES_DECRYPT()函数，它们使用高级加密标准（AES）算法产生更复杂的密码。考虑前面例子的一个变化，它演示了如何使用这些函数：

```
mysql> INSERT INTO users
VALUES ('', 'john', AES_ENCRYPT('opensesame', 'key'));
Query OK, 1 row affected (0.06 sec)
```

```
mysql> SELECT * FROM users;
+----+-------+------------------+
| id | uname | upass            |
+----+-------+------------------+
|  1 | john  | ¢¦}¶__ ëßìn_<-á  |
+----+-------+------------------+
1 row in set (0.11 sec)
```

和ENCODE()一样，调用AES_ENCRYPT()的结果是一个二进制字符串，以BLOB类型存储。和前面的例子一样，可以使用AES_DECRYPT()函数解密这个二进制数据：

```
mysql> SELECT id, uname, AES_DECRYPT(upass, 'key') FROM users;
+----+-------+--------------------------+
| id | uname | AES_DECRYPT(upass, 'key') |
+----+-------+--------------------------+
|  1 | john  | opensesame               |
+----+-------+--------------------------+
1 row in set (0.11 sec)
```

保护自己

如果MySQL RDBMS支持SSL（安全套接字层），我们可以通过DES_ENCRYPT()和DES_DECRYPT()函数使用更强大的Triple-Data加密标准（DES）算法进行加密。

可以使用MD5()函数计算字符串的MD5校验和（128位）：

```
mysql> SELECT MD5('hard-to-guess-password');
+----------------------------------+
| MD5('hard-to-guess-password')    |
+----------------------------------+
| 169d405191827bad1443d67e9f8e5574 |
+----------------------------------+
1 row in set (0.01 sec)
```

或者使用SHA()函数计算字符串的SHA校验和（160位）：

```
mysql> SELECT SHA('hard-to-guess-password');
+------------------------------------------+
| SHA('hard-to-guess-password')            |
+------------------------------------------+
| dfe0c5966af28b26cdcb949ba11d4760f440cb00 |
+------------------------------------------+
1 row in set (0.03 sec)
```

这些函数返回的校验和是十六进制值，它们适合于认证系统中使用的口令，如下所示：

```
mysql> INSERT INTO users values('', 'john', MD5('mypass'));
Query OK, 1 row affected (0.10 sec)

mysql> SELECT * FROM users;
+----+-------+----------------------------------+
| id | uname | upass                            |
+----+-------+----------------------------------+
```

```
|   2 | john   | ³q[|èBgØùM+                        |
|   3 | john   | john                               |
|   4 | john   | a029d0df84eb5549c641e04a9ef389e5 |
+----+-------+-----------------------------------+
3 rows in set (0.06 sec)

mysql> SELECT * FROM users WHERE uname='john' AND upass=md5('mypass');
+----+-------+-----------------------------------+
| id | uname | upass                             |
+----+-------+-----------------------------------+
|   4 | john   | a029d0df84eb5549c641e04a9ef389e5 |
+----+-------+-----------------------------------+
1 row in set (0.02 sec)

mysql> SELECT * FROM users
WHERE uname='john' AND upass=md5('wrongpass');
Empty set (0.00 sec)
```

7.6　控制流函数

也许你并不知道，MySQL有4个函数是用来进行条件操作的。这些函数可以实现SQL的条件逻辑，允许开发者将一些应用程序业务逻辑转换到数据库后台。表7.7列出了这些函数。

表 7.7　MySQL 控制流函数

函数	功能
CASE WHEN[test1] THEN [result1]…ELSE [default] END	如果testN是真，则返回resultN，否则返回default
CASE [test] WHEN [val1] THEN [result1]…ELSE [default] END	如果test和valN相等，则返回resultN，否则返回default
IF(test, t, f)	如果test是真，返回t；否则返回f
IFNULL(arg1, arg2)	如果arg1不是空，返回arg1；否则返回arg2
NULLIF(arg1, arg2)	如果arg1= arg2返回NULL；否则返回arg1

这些函数的第一个是IFNULL()，它有两个参数，并且对第一个参数进行判断。如果第一个参数不是NULL，函数就会向调用者返回第一个参数；如果是NULL，将返回第二个参数：

```
mysql> SELECT IFNULL(1, 2),
IFNULL(NULL, 10),
IFNULL(4 * NULL, 'false');
+--------------+-----------------+--------------------------+
| IFNULL(1, 2) | IFNULL(NULL, 10) | IFNULL(4 * NULL, 'false') |
+--------------+-----------------+--------------------------+
|            1 | 10              | false                    |
+--------------+-----------------+--------------------------+
1 row in set (0.00 sec)
```

NULLIF()函数将会检验提供的两个参数是否相等，如果相等，则返回NULL，如果不相等就返回第一个参数：

```
mysql> SELECT NULLIF(1,1), NULLIF('a', 'b'), NULLIF(2+3, 4+1);
+-------------+-----------------+-----------------+
| NULLIF(1,1) | NULLIF('a', 'b') | NULLIF(2+3, 4+1) |
+-------------+-----------------+-----------------+
|        NULL | a               |            NULL |
+-------------+-----------------+-----------------+
1 row in set (0.01 sec)
```

和许多脚本语言提供的IF()函数一样，MySQL的IF()函数也可以建立一个简单的条件测试。这个函数有三个参数，第一个是要被判断的表达式，如果表达式为真，IF()将会返回第二个参数；如果为假，IF()将会返回第三个参数。如：

```
mysql> SELECT IF(1 < 10, 2, 3), IF(56 > 100, 'true', 'false');
+----------------+-----------------------------+
| IF(1 < 10, 2, 3) | IF(56 > 100, 'true', 'false') |
+----------------+-----------------------------+
|              2 | false                       |
+----------------+-----------------------------+
1 row in set (0.00 sec)
```

IF()函数在只有两种可能结果时才适合使用。然而，在现实世界中，这是不够的，我们可能发现在条件测试中会需要多个分支。在这种情况下，MySQL 提供了 CASE 函数，它和 PHP 及 Perl 编程语言的 switch-case 条件例程一样。

CASE 函数的格式有些复杂，通常如下所示：

```
CASE [ expression to be evaluated]
    WHEN [ val 1] THEN [ result 1]
    WHEN [ val 2] THEN [ result 2]
     ...
    WHEN [ val n] THEN [ result n]
    ELSE [ default result]
END
```

这里，第一个参数是要被判断的值或表达式，接下来的是一系列的WHEN-THEN块，每一块的第一个参数指定要比较的值，如果为真，就返回结果。所有的WHEN-THEN块将以ELSE块结束，当END结束了所有外部的CASE块时，如果前面的每一个块都不匹配就会返回ELSE块指定的默认结果。如果没有指定ELSE块，而且所有的WHEN-THEN比较都不是真，MySQL将会返回 NULL。

CASE 函数还有另外一种句法，有时使用起来非常方便。如下所示：

```
CASE
    WHEN [ conditional test 1] THEN [ result 1]
    WHEN [ conditional test 2] THEN [ result 2]
    ELSE [ default result]
END
```

这种情况下，返回的结果取决于相应的条件测试是否为真。

下面的几个例子将清楚地介绍这一点：

```
mysql> SELECT CASE 'green'
WHEN 'red' THEN 'stop'
WHEN 'green' THEN 'go'
END;
+-----------------------------------------------------------------+
| CASE 'green' WHEN 'red' THEN 'stop' WHEN 'green' THEN 'go' END |
+-----------------------------------------------------------------+
| go                                                              |
+-----------------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT CASE 9 WHEN 1 THEN 'a' WHEN 2 THEN 'b' ELSE 'n/a' END;
+-----------------------------------------------------------+
| CASE 9 WHEN 1 THEN 'a' WHEN 2 THEN 'b' ELSE 'n/a' END |
+-----------------------------------------------------------+
| n/a                                                       |
+-----------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT CASE WHEN (2 + 2) = 4 THEN 'OK'
WHEN (2 + 2) <> 4 THEN 'NOT OK' END AS status;
+--------+
| status |
+--------+
| OK     |
+--------+
1 row in set (0.16 sec)
```

回顾本章前面的一个例子——学生和他们的成绩，我们可以使用IF()函数找到每一门课哪个学生及格哪个学生不及格，然后显示相应的结果：

```
mysql> SELECT CONCAT_WS(' ', fname, lname) AS name,
IF((math) / 100 < 0.35, 'fail in math', 'pass in math') AS result
FROM marks;
+-------------+--------------+
| name        | result       |
+-------------+--------------+
| John Doe    | pass in math |
| Sarah Short | fail in math |
| Joe Cool    | pass in math |
| Mark Wumba  | pass in math |
| Tom Thumb   | fail in math |
+-------------+--------------+
5 rows in set (0.00 sec)
```

如果需要根据每个学生的行为指定分数，使用CASE函数就可以很简单地实现，如下所示：

```
mysql> SELECT fname, lname, (math + sci + lit) AS total,
CASE WHEN (math + sci + lit) < 50 THEN 'D'
WHEN (math + sci + lit) BETWEEN 50 AND 150 THEN 'C'
WHEN (math + sci + lit) BETWEEN 151 AND 250 THEN 'B'
```

```
ELSE 'A' END
AS grade FROM marks;
+-------+-------+-------+-------+
| fname | lname | total | grade |
+-------+-------+-------+-------+
| John  | Doe   |   143 | C     |
| Sarah | Short |   145 | C     |
| Joe   | Cool  |   246 | B     |
| Mark  | Wumba |   145 | C     |
| Tom   | Thumb |    12 | D     |
+-------+-------+-------+-------+
5 rows in set (0.00 sec)
```

或者我们可以使用一系列嵌套的IF()函数：

```
mysql> SELECT fname, lname, (math + sci + lit) AS total,
IF((math + sci + lit) < 50,'C',
IF((math + sci + lit) BETWEEN 50 AND 150,'B',
IF ((math + sci + lit) BETWEEN 151 and 250, 'C', 'A')))
AS grade FROM marks;
+-------+-------+-------+-------+
| fname | lname | total | grade |
+-------+-------+-------+-------+
| John  | Doe   |   143 | B     |
| Sarah | Short |   145 | B     |
| Joe   | Cool  |   246 | C     |
| Mark  | Wumba |   145 | B     |
| Tom   | Thumb |    12 | C     |
+-------+-------+-------+-------+
5 rows in set (0.01 sec)
```

考虑另外一个例子，由用户数据库组成的认证系统：

```
mysql> SELECT * FROM users;
+-----+-------+---------------+
| uid | uname | upass         |
+-----+-------+---------------+
|   3 | mark  | ts0o378GTHJTw |
|   2 | joe   | tsuYK6Jh0iVvI |
|   4 | sue   | tsEh.9z5n8FQc |
+-----+-------+---------------+
3 rows in set (0.00 sec)
```

这个表中存储的用户口令是使用MySQL的ENCRYPT()函数进行加密的。

现在，我们使用这个系统进行认证，有两种选择：在应用程序层放置业务逻辑来进行加密和验证，或者通过MySQL内建的IF()函数进行认证。因为本书是关于MySQL的，所以就采用第二种方法：

```
mysql> SELECT IF(ENCRYPT('sue' 'ts') = upass, 'allow', 'deny')
AS loginResult FROM users WHERE uname = 'sue';
```

```
+-------------+
| loginResult |
+-------------+
| deny        |
+-------------+
1 row in set (0.00 sec)

mysql> SELECT IF(ENCRYPT('secret', 'ts') = upass, 'allow', 'deny')
AS loginResult FROM users WHERE uname = 'sue';
+-------------+
| loginResult |
+-------------+
| allow       |
+-------------+
1 row in set (0.00 sec)
```

在这种情况下，用户输入的是简单的ENCRYPT()加密的口令，它与同时具有验证口令和与应用层的结果通信的双重任务的MySQL IF()测试结合。因而应用层的负荷立即减少：现在只需要输入用户提供的明文口令到终端（通过前面的 SELECT 查询），读出输出结果，然后允许或者拒绝通过，而不是必须执行加密过程，向 MySQL 输入数据，再进行核查。

7.7　格式化函数

MySQL 还有一些函数是特意为格式化数据设计的，如表 7.8 所示。

表 7.8　MySQL 格式化函数

函数	功能
DATE_FORMAT(date, fmt)	依照字符串 fmt 格式化日期 date 值
FORMAT(*x*, *y*)	把 *x* 格式化为以逗号隔开的数字序列，*y* 是结果的小数位数
INET_ATON(ip)	返回 IP 地址的数字表示
INET_NTOA(num)	返回数字所代表的 IP 地址
TIME_FORMAT(time, fmt)	依照字符串 fmt 格式化时间 time 值

其中最简单的是FORMAT()函数，它可以把大的数值格式化为以逗号间隔的易读的序列。FORMAT()的第一个参数是被格式化的数据，第二个参数是结果的小数位数：

```
mysql> SELECT FORMAT(999999999.868595049, 2), FORMAT(-4512,6);
+--------------------------------+-----------------+
| FORMAT(999999999.868595049, 2) | FORMAT(-4512,6) |
+--------------------------------+-----------------+
| 999,999,999.87                 | -4,512.000000   |
+--------------------------------+-----------------+
1 row in set (0.00 sec)
```

DATE_FORMAT()和TIME_FORMAT()函数可以用来格式化日期和时间值。它接受日期或者时间值和一个指定结果格式的格式化字符串。这个格式化字符串包含特殊的符号，它用来改变结果值的显示，表 7.9 列出了这些符号。

注释： 我们可以从 MySQL 手册获得这些特殊符号的更完全的列表，它位于 http://www.mysql.com/documentation/。

表 7.9 MySQL 日期/时间格式化代码

函数	功能
%a	缩写的星期名（Sun，Mon…）
%b	缩写的月份名（Jan，Feb…）
%d	月份中的天数
%H	小时（01，02…）
%I	分钟（00，01…）
%j	一年中的天数（001，002…）
%m	月份，2 位（00，01…）
%M	长型月份的名字（January，February…）
%p	AM 或 PM
%r	时间，12 小时的格式
%S	秒（00，01…）
%T	时间，24 小时的格式
%w	一周中的天数（0，1…）
%W	长型星期的名字（Sunday，Monday…）
%Y	年份，4 位

下面是应用这些函数的一些例子：

```
mysql> SELECT DATE_FORMAT(NOW(), '%W, %D %M %Y %r');
+----------------------------------------+
| DATE_FORMAT(NOW(), '%W, %D %M %Y %r') |
+----------------------------------------+
| Thursday, 15th May 2003 01:38:44 PM   |
+----------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT(19980317, '%d/%m/%Y');
+-----------------------------------+
| DATE_FORMAT(19980317, '%d/%m/%Y') |
+-----------------------------------+
| 17/03/1998                        |
+-----------------------------------+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT(19690906070134, '%s %i %h %d %m %y');
+-------------------------------------------------+
| DATE_FORMAT(19690906070134, '%s %i %h %d %m %y') |
+-------------------------------------------------+
| 34 01 07 06 09 69                               |
+-------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT TIME_FORMAT(19690609140256, '%h:%i %p');
+----------------------------------------+
| TIME_FORMAT(19690609140256, '%h:%i %p') |
+----------------------------------------+
```

```
| 02:02 PM                                |
+-----------------------------------------+
1 row in set (0.00 sec)
```

MySQL 的 INET_NTOA()和 INET_ATON()函数可以分别把 IP 地址转化为数字或者进行相反的操作。如下面的例子所示：

```
mysql> SELECT INET_ATON('192.168.2.10');
+---------------------------+
| INET_ATON('192.168.2.10') |
+---------------------------+
|                3232236042 |
+---------------------------+
1 row in set (0.05 sec)

mysql> SELECT INET_NTOA(3232236042);
+-----------------------+
| INET_NTOA(3232236042) |
+-----------------------+
| 192.168.2.10          |
+-----------------------+
1 row in set (0.05 sec)
```

7.8　类型转化函数

为了进行数据类型转化，MySQL 提供了 CAST()函数，它可以把一个值转化为指定的数据类型。MySQL 4.1 支持以下几种类型：

- BINARY
- CHAR
- DATE
- TIME
- DATETIME
- SIGNED
- UNSIGNED

通常情况下，当使用数值操作时，字符串会自动地转化为数字，因此下面两种操作是相同的：

```
mysql> SELECT 1 + '99';
+----------+
| 1 + '99' |
+----------+
|      100 |
+----------+
1 row in set (0.03 sec)

mysql> SELECT 1 + CAST('99' AS SIGNED);
```

```
+-------------------------+
| 1 + CAST('99' AS SIGNED) |
+-------------------------+
|                     100 |
+-------------------------+
1 row in set (0.00 sec)
```

类似地，字符串可以指定为BINARY类型，这样它们的比较操作就成为大小写敏感的。使用CAST()函数指定一个字符串为BINARY和在字符串前面使用BINARY关键词具有相同的作用。如：

```
mysql> SELECT 'f' = BINARY 'F', 'f' = CAST('F' AS BINARY);
+-----------------+-------------------------+
| 'f' = BINARY 'F' | 'f' = CAST('F' AS BINARY) |
+-----------------+-------------------------+
|               0 |                       0 |
+-----------------+-------------------------+
1 row in set (0.00 sec)
```

有趣的是，我们可以强制许多日期和时间函数［包括NOW()，CURTIME()和CURDATE()函数］，把它们返回的值作为一个数而不是字符串输出，只需要在数字的环境中使用这些函数或者把它们转化为数字。如：

```
mysql> SELECT CAST(NOW() AS SIGNED INTEGER), CURDATE() + 0;
+-------------------------------+---------------+
| CAST(NOW() AS SIGNED INTEGER) | CURDATE() + 0 |
+-------------------------------+---------------+
|                 20030515111705 |      20030515 |
+-------------------------------+---------------+
1 row in set (0.00 sec)
```

当我们要把数据移动到一个新的RDBMS时，CAST()函数就显得尤其有用，因为它允许我们把值从旧数据类型转变为新的数据类型，以使它们更适合新的系统。为了进行说明，考虑下面的表（不是MySQL的），它以字符串形式存储日期值：

```
mysql> SELECT * FROM old;
+-------------------------------+-----+-------+------+------+-----+
| email                         | day | month | year | hour | min |
+-------------------------------+-----+-------+------+------+-----+
| mark@some.domain.net          | 15  | 03    | 1999 | 13   | 42  |
| armyguy@free.mail.service.com | 31  | 12    | 2001 | 19   | 59  |
| debuz@some.unknown.tld        | 08  | 11    | 1996 | 03   | 37  |
+-------------------------------+-----+-------+------+------+-----+
3 rows in set (0.00 sec)
```

要把这些数据移动到MySQL中，第一步要先创建一个表来存储这些数据：

```
mysql> CREATE TABLE new (
email varchar(255) NOT NULL default '',
ts datetime NOT NULL default '0000-00-00 00:00:00',
```

```
PRIMARY KEY (email) )
TYPE=MyISAM;
Query OK, 0 rows affected (0.05 sec)
```

然后从旧的表中把数据导入新表，并把值转变为合适的 MySQL 数据类型：

```
mysql> INSERT INTO new(email, ts) SELECT email,
CAST(CONCAT(year,month,day,hour,min,'00') AS DATETIME) FROM old;
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM new;
+-------------------------------+---------------------+
| email                         | ts                  |
+-------------------------------+---------------------+
| mark@some.domain.net          | 1999-03-15 13:42:00 |
| armyguy@free.mail.service.com | 2001-12-31 19:59:00 |
| debuz@some.unknown.tld        | 1996-11-08 03:37:00 |
+-------------------------------+---------------------+
3 rows in set (0.00 sec)
```

7.9　系统信息函数

MySQL 还具有一些特殊的函数用来获得系统本身的信息，如表 7.10 所示。

表 7.10　MySQL 系统信息函数

函数	功能
DATABASE()	返回当前数据库名
BENCHMARK(count,expr)	将表达式 expr 重复运行 count 次
CONNECTION_ID()	返回当前客户的连接 ID
FOUND_ROWS()	将最后一个 SELECT 查询（没有以 LIMIT 进行限制结果）返回的记录行数返回
GET_LOCK(str, dur)	获得一个由字符串 str 命名的并且有 dur 秒延时的锁定
IS_FREE_LOCK(str)	检查以 str 命名的锁定是否释放
LAST_INSERT_ID()	返回由系统自动产生的最后一个 AUTOINCREMENT ID 的值
MASTER_POS_WAIT(log, pos, dur)	锁定主服务器 dur 秒直到从服务器与主服务器的日志 log 指定的位置 pos 同步
RELEASE_LOCK(str)	释放由字符串 str 命名的锁定
USER()或 SYSTEM_USER()	返回当前登录用户名
VERSION()	返回 MySQL 服务器的版本

DATABASE()，USER()和 VERSION()函数可以分别返回当前所选数据库、当前用户和 MySQL 版本的信息：

```
mysql> SELECT DATABASE(), VERSION(), USER();
+------------+------------------+-----------------+
| DATABASE() | VERSION()        | USER()          |
+------------+------------------+-----------------+
|            | 4.0.12-max-debug | root@domain.com |
+------------+------------------+-----------------+
1 row in set (0.11 sec)
```

CONNECTION_ID()函数检索访问服务器的客户的连接ID，而LAST_INSERT_ID()函数返回向AUTOINCREMENT字段中插入数据时由MySQL自动产生的最后一个ID。当处理多个表之间的连接或者需要使用一个表的记录ID作为另一个表的密钥时，使用这个函数就会比较方便。如下所示：

```
mysql> INSERT INTO users VALUES ('', 'john', 'john');
Query OK, 1 row affected (0.03 sec)

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                3 |
+------------------+
1 row in set (0.03 sec)

mysql> INSERT INTO departments VALUES (55, LAST_INSERT_ID());
Query OK, 1 row affected (0.03 sec)
```

BENCHMARK()函数通过服务器判断特定表达式指定的次数来使开发者评价MySQL的性能。函数经常返回0，以及在连接的客户端经过的时间报告，这个经过时间值可以用来推断服务器的性能。如：

```
mysql> SELECT BENCHMARK(999999, LOG(RAND() * PI()));
+-------------------------------------+
| BENCHMARK(999999, LOG(RAND() * PI())) |
+-------------------------------------+
|                                   0 |
+-------------------------------------+
1 row in set (2.08 sec)
```

这个例子中，MySQL计算LOG(RAND()*PI())表达式999 999次需要2.08秒。在一定的间隔时间或者不同的间隔时间执行这个函数，开发者可以获得用来衡量服务器性能的原始数据。

MySQL还有三个特意为实现客户之间的应用层锁定设计的函数，这些客户是运行在单个机器上的。GET_LOCK()函数有两个参数，锁定的名字和持续的时间（秒），然后尝试创建具有那个名字的锁定，并指定秒数。如果成功，GET_LOCK()返回1；如果不成功，返回0或者NULL。如：

```
mysql> SELECT GET_LOCK('20030907', 60);
+-------------------------+
| GET_LOCK('20030907', 60) |
+-------------------------+
|                       1 |
+-------------------------+
1 row in set (0.06 sec)
```

RELEASE_LOCK()函数进行逆转，释放给定名字的锁定，如果成功返回1，失败返回0，错误返回NULL。如：

```
mysql> SELECT RELEASE_LOCK('20030907');
+--------------------------+
| RELEASE_LOCK('20030907') |
+--------------------------+
|                        1 |
+--------------------------+
1 row in set (0.00 sec)
```

IS_FREE_LOCK()函数检查给定名字的锁定是否存在，而且根据这个锁定是自由的或者是正在使用的情况返回 1 或者 0。如：

```
mysql> SELECT IS_FREE_LOCK('20030907');
+--------------------------+
| IS_FREE_LOCK('20030907') |
+--------------------------+
|                        0 |
+--------------------------+
1 row in set (0.06 sec)

mysql> SELECT RELEASE_LOCK('20030907');
+--------------------------+
| RELEASE_LOCK('20030907') |
+--------------------------+
|                        1 |
+--------------------------+
1 row in set (0.00 sec)

mysql> SELECT IS_FREE_LOCK('20030907');
+--------------------------+
| IS_FREE_LOCK('20030907') |
+--------------------------+
|                        1 |
+--------------------------+
```

任何时间都不能有多于 1 个的锁定处于活动状态，多次调用GET_LOCK()将释放前面产生的锁定，如下例所示：

```
mysql> SELECT GET_LOCK('a', 60);
+-------------------+
| GET_LOCK('a', 60) |
+-------------------+
|                 1 |
+-------------------+
1 row in set (0.00 sec)

mysql> SELECT IS_FREE_LOCK('a');
+-------------------+
| IS_FREE_LOCK('a') |
+-------------------+
```

```
|                 0 |
+-------------------+
1 row in set (0.00 sec)

mysql> SELECT GET_LOCK('b', 60);
+-------------------+
| GET_LOCK('b', 60) |
+-------------------+
|                 1 |
+-------------------+
1 row in set (0.00 sec)

mysql> SELECT IS_FREE_LOCK('a');
+-------------------+
| IS_FREE_LOCK('a') |
+-------------------+
|                 1 |
+-------------------+
1 row in set (0.00 sec)
```

GET_LOCK()，RELEASE_LOCK()和IS_FREE_LOCK()函数使多个MySQL客户的同步活动变得非常容易，使用普通锁定的客户可以使用这些函数避免和其他用户混淆。

FOUND_ROWS()函数返回最后一个SELECT查询进行检索的总行数。将SQL_CALC_FOUND_ROWS和LIMIT结构结合尤其有用，它可以“浏览”查询集（结合LIMIT）的结果，同时显示检索［使用SQL_CALC_FOUND_ROWS和FOUND_ROWS()］的记录的总数。考虑下面简单的例子，它对此进行了说明：

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM racedata LIMIT 0,4;
+--------------+-------+-----+-------+----------+
| name         | total | win | place | unplaced |
+--------------+-------+-----+-------+----------+
| Stormtrooper |    50 |   4 |    13 |        9 |
| VioletDaisy  |     9 |   4 |     5 |        0 |
| King William |    33 |  11 |    15 |        4 |
| Remuneration |   127 |  35 |    56 |       22 |
+--------------+-------+-----+-------+----------+
4 rows in set (0.01 sec)

mysql> SELECT FOUND_ROWS();
+--------------+
| FOUND_ROWS() |
+--------------+
|           71 |
+--------------+
1 row in set (0.00 sec)
```

在这个例子中，第一个查询只能检索到结果集中有4条记录的有限子集。不过，只要在查询中使用SQL_CALC_FOUND_ROWS，MySQL就会记录结果集总行数，并允许通过调用

FOUND_ROWS()来检索这个数。因而第二个查询可以不通过重新检查表就从服务器返回一个统计数字。

最后，MASTER_POS_WAIT()函数主要是用在MySQL主从关系的从服务器端的一个专门函数。MASTER_POS_WAIT()函数有三个参数——日志名字、日志位置和超时值，并且锁定主服务器直到从服务器更新自己到日志中指定的位置。这个函数经常用于复制环境。请看第17章中关于复制的更多内容。

7.10　小结

本章讨论了 MySQL RDBMS 的多种内建函数，把它们分为多个类别，并用例子和图示进行了解释。

正如本章的最后几页所表明的那样，MySQL 有大量函数，允许用户进行从简单（获得当前的日期和时间，进行简单的数学和字符串操作）到复杂（三角运算，日期和时间计算，数据加密）的操作。本章主要是希望利用解释和图示让大家熟悉其中重要的一些函数。这些例子，无论大小，都是用来阐述这些函数如何帮助开发者把业务逻辑从应用层转换到数据库层的，这种转换可以提高系统性能。

第8章　使用数据库和表

也许还记得本书前面章节所讲的，SQL规范大概可以分为三部分：数据定义语言（DDL）、数据操纵语言（DML）和数据控制语言（DCL）。本章全部集中在DDL部分，讨论和数据库及表管理相关的MySQL命令。除了提供关于如何向系统添加、删除MySQL数据库和表的信息以外，还将讨论MySQL中可用的多种表类型的优点。这些表从旧的ISAM表到新的事务安全InnoDB表都有涉及。本章同样会提供指导来帮助MySQL的开发者，为他们的应用提供合适的选择。

8.1　创建数据库

我们已经看到，MySQL的任何事情都是以数据库开始的，这个数据库作为盛放表和表中记录的容器。通过CREATE DATABASE命令可以创建一个新的MySQL数据库。假设已经通过mysql命令行客户机连接到一个MySQL服务器，可以自己尝试下面的命令：

```
mysql> CREATE DATABASE db1;
Query OK, 1 row affected (0.05 sec)
```

这样就创建了一个名为db1的数据库。

在文件系统中，MySQL的数据存储区将以目录方式表示MySQL数据库。因此，上面命令中的数据库名字必须与操作系统（OS）的约束的目录名字的字母相一致。例如，Microsoft Windows不允许文件和目录名中有\, /, :, *,?, ",<,>, |这些符号，在MySQL数据库名字中这些字母会被自动删除。

数据库的名字不能超过64个字符，包含特殊字符的名字或者是全部由数字或保留字组成的名字必须用单引号包起来。考虑下面的情况：

```
mysql> CREATE DATABASE 875748;
ERROR 1064: You have an error in your SQL syntax.  Check the
manual that corresponds to your MySQL server version for the
right syntax to use near '875748' at line 1

mysql> CREATE DATABASE '875748';
Query OK, 1 row affected (0.06 sec)

mysql> CREATE DATABASE 'group';
Query OK, 1 row affected (0.00 sec)
```

显而易见，MySQL不允许两个数据库使用相同的名字。每一次这样的尝试都会导致错误的产生，如下所示：

```
mysql> CREATE DATABASE db1;
Query OK, 1 row affected (0.03 sec)

mysql> CREATE DATABASE db1;
ERROR 1007: Can't create database 'db1'. Database exists
```

当使用自动脚本创建数据库时，像前面那样的错误将会潜在地破坏程序流。通过使用MySQL 3.22以及更高版本支持的IF EXISTS和IF NOT EXISTS从句可以避免此类错误。考虑经过修改了的前面的例子，它阐明了这些子句如何帮助我们创建得体的SQL：

```
mysql> CREATE DATABASE db1;
ERROR 1007: Can't create database 'db1'. Database exists

mysql> CREATE DATABASE IF NOT EXISTS db1;
Query OK, 0 rows affected (0.00 sec)
```

在第二个例子中，IF NOT EXISTS从句的使用强制MySQL在不存在数据库的情况下执行此命令，这将进行更加合适的处理并且不产生错误。

数据库名字

通常情况下，应以大写字母作为数据库名字的开头，并且保证数据库名字只由字母数字和下划线组成。虽然MySQL可以处理比较奇怪的名字，但是其他应用程序可能不是这么宽宏大量的。

8.2　选择需要的数据库

使用USE语句将会选择一个数据库，使它成为所有事务的当前数据库。考虑下面的例子，它指明选择数据库db1作为所有即将进行的查询的默认数据库：

```
mysql> USE db1;
Database changed
```

一旦数据库以这种方式被选择，它将变成所有在数据库连接上执行查询的默认数据库，下面的第一个命令，与第二和第三个命令组合的作用是相同的：

```
mysql> SELECT Host, Db, User FROM mysql.db;
+------+-------+------+
| Host | Db    | User |
+------+-------+------+
| %    | test% |      |
+------+-------+------+
1 row in set (0.00 sec)

mysql> USE mysql;
Database changed

mysql> SELECT Host, Db, User FROM db;
+------+-------+------+
| Host | Db    | User |
+------+-------+------+
| %    | test% |      |
+------+-------+------+
4 rows in set (0.06 sec)
```

当然，使用USE命令选择一个数据库并不意味着不可以在同一会话期访问其他数据库。设置表名的前缀为数据库名，就能在其他数据库中使用这个表，因为MySQL知道如何找到它，如下所示：

```
mysql> USE db1;
Database changed

mysql> SELECT User FROM mysql.user;
+-------+
| User |
+-------+
|       |
| root |
|       |
| root |
+-------+
4 rows in set (0.06 sec)
```

8.3 删除数据库

可以使用DROP DATABASE命令删除一个数据库以及里面的所有表。考虑下面的例子，它删除了名为 db1 的数据库：

```
mysql> DROP DATABASE db1;
Query OK, 0 rows affected (0.02 sec)
```

注意，这个命令必须小心使用，因为它将删除指定数据库的所有表（包括其中的数据）和数据库的目录。

我们可以使用 IF EXISTS 子句以避免删除不存在的数据库时出现的 MySQL 错误信息：

```
mysql> DROP DATABASE library;
ERROR 1008: Can't drop database 'library'. Database doesn't exist

mysql> DROP DATABASE IF EXISTS library;
Query OK, 0 rows affected (0.00 sec)
```

8.4 创建表

一旦创建了数据库，下一步就是创建一个或者更多的表来丰富数据库。SQL用来在数据库中创建表的命令如下：

```
CREATE TABLE table-name (field-definition, field-definition, ...)
 TYPE=table-type;
```

适用于数据库名字的规则同样也可以适用于表名和字段名。每个表（和它包含的数据）在 MySQL 的data 目录里都以三个文件存储（假定使用默认的 MyISAM 表类型）。可以通过用逗号分隔的不同字段的描述来为表指定尽可能多的字段。字段和表名字中如果有 MySQL 保留字也必须用单引号包起来。

下面这个例子讲述了如何使用这个命令创建一个简单的表来存储用户名和口令：

```
mysql> CREATE TABLE users (id INT(3), name VARCHAR(8), pass VARCHAR(20));
Query OK, 0 rows affected (0.08 sec)
```

下面是一个更加复杂的表，它不仅指定了字段和表数据的约束，还定义了表类型：

```
mysql> CREATE TABLE firewall (
    -> host VARCHAR(11) NOT NULL,
    -> port SMALLINT(4) NOT NULL,
    -> access ENUM('DENY', 'ALLOW') NOT NULL,
    -> PRIMARY KEY(host, port)
    -> ) TYPE=HEAP;
Query OK, 0 rows affected (0.04 sec)
```

或者采用下面的方式：

```
mysql> CREATE TABLE menu (
    -> id INT(10) UNSIGNED DEFAULT '0' NOT NULL,
    -> label VARCHAR(255) NOT NULL,
    -> mpl TINYINT(2) UNSIGNED DEFAULT '0' NOT NULL,
    -> parent INT(10) DEFAULT '0' NOT NULL,
    -> link VARCHAR(255),
    -> PRIMARY KEY (id)
    -> );
Query OK, 0 rows affected (0.04 sec)
```

从前面的例子可以看到，CREATE TABLE 命令中的字段定义由字段的惟一名字以及跟在后面的指定字段中数据类型的字段类型组成。这样的字段定义可以使用单独的CREATE TABLE命令来实现。定义后面跟随的是表级约束和表类型定义。

下面更详细地介绍这些内容。

8.4.1　字段类型

MySQL 提供了很多不同的数据类型来处理不同的数据需求。比较重要的一些总结在表8.1中，并在第 5 章中做了详细的讨论。

表 8.1　MySQL 数据类型

类型	描述
INT	一种数值类型，值的范围为 -2 147 483 648~2 147 483 647
DECIMAL	一种数值类型，支持浮点数或者小数
DOUBLE	一种数值类型，支持双精度浮点数
DATE	YYYYMMDD 格式的日期字段
TIME	HH:MM:SS 格式的时间字段
DATETIME	YYYYMMDD HH:MM:SS 格式的日期 / 时间类型
YEAR	以 YYYY 或者 YY 格式，范围在 1901~2155 之间的指定年的字段
TIMESTAMP	以 YYYYMMDDHHMMSS 格式的时间戳类型
CHAR	最大长度为 255 个字符并且有固定长度的字符串类型
VARCHAR	最大长度为 255 个字符但是变长的字符串类型
TEXT	最大长度为 65 535 个字符的字符串的类型
BLOB	可变数据的二进制类型
ENUM	可以从定义数值的列表上接受值的数据类型
SET	可以从定义数值的集合上接受 0 个或者多个值的数据类型

8.4.2　字段约束

每个字段定义都包含附加约束或者修饰符，这些可以用来增加对所输入数据的约束。

表的类型

CREATE TABLE 命令的 TYPE 子句是可选的，同时 MyISAM 表类型是 MySQL 表默认的类型。关于表类型的更详细的讨论，参见 8.4.6 节。

NULL 和 NOT NULL 修饰符

可以在每个字段定义后面放置 NULL 或 NOT NULL 修饰符来指定这个字段是否可以为空还是必须填上数据。MySQL默认情况下指定为NULL修饰符。如果一个字段指定为NOT NULL，MySQL 不允许向该字段插入空值。

例如，如果在创建的用户表的不同字段中强制使用 NOT NULL 规则，表定义如下：

```
mysql> CREATE TABLE users (
    -> id INT(3) NOT NULL,
    -> name VARCHAR(8) NOT NULL,
    -> pass VARCHAR(20) NOT NULL
    -> );
Query OK, 0 rows affected (0.01 sec)
```

就不能向这样的表中插入包含 NULL 值的记录：

```
mysql> INSERT INTO users (id, name, pass) VALUES ('1', 'john', NULL);
ERROR 1048: Column 'pass' cannot be null
```

但是在自增列和 TIMESTAMP 字段中，这个规则并不适用。向这些字段中插入 NULL 值，将会导致插入下一个自动增加的值或者当前时间戳。

DEFAULT 修饰符

可以使用DEFAULT修饰符为字段指定一个默认值。当插入记录时，如果没有向字段中插入值就会使用默认值。考虑下面的例子，它创建了连接雇员和部门的表。默认情况下，雇员分配在 MIS 部门：

```
mysql> CREATE TABLE memberships (
    -> user VARCHAR(8) NOT NULL,
    -> dept VARCHAR(25) DEFAULT 'MIS'
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO memberships (user, dept) VALUES ('joe', 'HR');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO memberships (user) VALUES ('john');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM memberships;
+------+------+
| user | dept |
+------+------+
| joe  | HR   |
| john | MIS  |
+------+------+
2 rows in set (0.00 sec)
```

如果在一个字段中没有指定DEFAULT修饰符，MySQL会依据这个字段是NULL还是NOT NULL 自动设置默认值。如果指定字段可以为 NULL，MySQL 默认插入 NULL。如果为 NOT NULL 字段，MySQL 对于数值类型插入 0，字符串类型插入空字符串，时间戳类型插入当时的日期和时间，ENUM 类型插入枚举组的第一条。

AUTO_INCREMENT 修饰符

AUTO_INCREMENT修饰符只适用于INT字段，表明MySQL应该自动为该字段生成一个数（每次在前一个值的基础上加1）。对于主键，这是非常有用的。因为它允许开发者使用MySQL为每条记录创建惟一的标识符。

考虑下面的例子，它阐述了如何为每条记录自动分配惟一的数字 ID：

```
mysql> CREATE TABLE items (
    -> id INT(5) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> label VARCHAR(255) NOT NULL
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO items (label) VALUES ('CFZ 11A89-0 adapter');
Query OK, 1 row affected (0.04 sec)

mysql> INSERT INTO items (label) VALUES ('R645u connector');
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO items (label) VALUES ('Crimping pliers');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT id, label FROM items;
+-----+---------------------+
| id  | label               |
+-----+---------------------+
|   1 | CFZ 11A89-0 adapter |
|   2 | R645u connector     |
|   3 | Crimping pliers     |
+-----+---------------------+
3 rows in set (0.00 sec)
```

注释： MySQL 的表只能有一个 AUTO_INCREMENT 字段，而且这个字段必须被定义为键。

除了字段级的约束，MySQL 也允许表级的约束，比如主键和外键、索引和惟一约束。这些约束置于 CREATE TABLE 命令的字段定义的后面。下面将详细介绍这些约束。

8.4.3　索引

为了加快搜索速度，减少查询执行时间，MySQL 允许我们为一个表的特定字段设置索引。“索引”在这里的意思与现实世界中的一样。与每本书结尾的索引的概念相同，一个索引就是字段值的一个列表，它用来简化查询中定位特定记录的任务。

如果没有索引，MySQL 就需要浏览表的每一行来寻找和特定查询相匹配的记录。对于小的表来说，这也许不会造成明显的速度减慢，但是随着表的增大，完全地浏览整个表可能增加

查询的时间。索引可以很大程度上提高速度：利用索引，MySQL可以不用浏览所有的表，而是查看索引以后，跳到表的合适位置。当看到与指定搜索条件相匹配的记录时，读出这个索引要比浏览整个表快得多。这是因为索引比较小，可以加快搜索（可以在第16章看到更多关于索引如何影响性能的内容）。

但是，索引有两个非常严重的缺点：它占据了一定的磁盘空间，而且它影响INSERT，UPDATE和DELETE执行的时间，因为当表记录被添加、更新或者删除时，索引每次都必须更新。但是，大多数情况下，这些理由不能阻止我们使用索引：磁盘越来越便宜，而且MySQL包含了很多优化技术来缩减用于更新索引和搜索索引的时间（请看下面的补充资料，可以了解更多的信息）。

散列效果

在InnoDB表中，MySQL使用智能插入缓冲的方式，通过在特定的插入缓冲中维护一个变化列表，然后一次性（而不是多次）更新所有变化的索引来减少向InnoDB索引所进行的磁盘写操作的次数。MySQL还会根据正在执行的查询的格式尝试将基于磁盘的B树索引转变为合适的散列索引（搜索得更快）。

对于经常使用WHERE，ORDER BY和GROUP BY这些子句进行SELECT查询的字段和用于连接表的字段，我们推荐使用索引。

可以通过向CREATE TABLE语句中添加INDEX修饰符来创建索引，如下所示：

```
mysql> CREATE TABLE stocks (
-> symbol CHAR(4) NOT NULL,
-> purchase_price FLOAT(6,2) NOT NULL,
-> purchase_date DATE NOT NULL,
-> INDEX sym (symbol)
-> );
Query OK, 0 rows affected (0.00 sec)
```

上面的语句为stocks表的symbol字段创建了名为sym的索引。

通过指定多个INDEX修饰符可以为多个字段添加索引，如下所示：

```
mysql> CREATE TABLE users (
-> id INT(4) NOT NULL AUTO_INCREMENT,
-> fname VARCHAR(50) NOT NULL,
-> lname VARCHAR(50) NOT NULL,
-> INDEX (fname), INDEX (lname), PRIMARY KEY (id)
-> );
Query OK, 0 rows affected (0.05 sec)
```

通过连接所有被索引的字段（最多不能超过15个）创建的多个索引，可以用逗号间隔字段名来创建，如下例所示：

```
mysql> CREATE TABLE users (
-> id INT(4) NOT NULL AUTO_INCREMENT,
-> fname VARCHAR(50) NOT NULL,
-> lname VARCHAR(50) NOT NULL,
-> INDEX (fname, lname), PRIMARY KEY (id)
-> );
```

```
Query OK, 0 rows affected (0.05 sec)
```

可以使用 CREATE INDEX 命令来向已存在的表添加索引。句法如下：

```
CREATE INDEX index-name ON table-name (field-name, field-name, ...);
```

下面这个例子为 sysusers 表的 uname 字段创建了名为 username 的索引：

```
mysql> CREATE INDEX username ON sysusers (uname);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

对于 BLOB 和 TEXT 字段，也可以强制性地指定字符的个数来为它们添加索引。这可以通过向CREATE INDEX语句中的字段名字后的圆括号中插入需要的索引长度来实现。例子如下：

```
mysql> CREATE INDEX synopsis ON books (synopsis(100));
Query OK, 83 rows affected (0.15 sec)
```

在 CHAR 和 VARCHAR 字段中也可以有选择地指定这样的部分索引。

注释：如果在 CREATE INDEX 或 CREATE TABLE 中没有指定索引的名字，MySQL 将会自动使用相应的字段名为索引命名。

如果发现不再需要索引时，可以使用 DROP INDEX 命令删除它，如下所示：

```
DROP INDEX index-name ON table-name;
```

下面这个例子删除了在前一个例子中创建的 username 索引：

```
mysql> DROP INDEX username ON sysusers;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

除了常规索引类型，MySQL 还支持另外两种索引：UNIQUE 索引和 FULLTEXT 索引，下面的内容对它们进行了讨论。

UNIQUE 索引

我们可以通过 CREATE TABLE 和 CREATE INDEX 命令的 UNIQUE 修饰符来指定输入字段中的值必须是惟一的，也就是不能与其他行重复。一旦一个字段被标志为 UNIQUE，任何向它输入重复数据的尝试都会失败。如：

```
mysql> CREATE TABLE users (
-> username VARCHAR(8),
-> userpass VARCHAR(20),
-> UNIQUE(username)
-> );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO users (username, userpass) VALUES ('jb', 'secret');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO users (username, userpass) VALUES ('jb', 'hidden');
ERROR 1062: Duplicate entry 'jb' for key 1
```

请注意，UNIQUE 字段允许存储 NULL 值，如下例所示：

```
mysql> CREATE TABLE users(
-> username VARCHAR(8) NOT NULL,
-> email VARCHAR(255),
-> UNIQUE(email)
-> );
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO users (username, email) VALUES ('jb',
'jb@some.domain');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO users (username, email) VALUES ('jimbozak',
'jb@some.domain');
ERROR 1062: Duplicate entry 'jb@some.domain' for key 1

mysql> INSERT INTO users (username, email) VALUES ('john', NULL);
Query OK, 1 row affected (0.00 sec)
```

FULLTEXT 索引

MySQL 3.23.23以及更高版本支持一种特殊类型的索引，它是为在MyISAM表中进行全文检索而特意设计的，称为FULLTEXT索引。它可以对纯文本字符串中已经经过索引的列进行查询，并且只返回那些包含与搜索字符串相类似的记录。当执行全文检索时，MySQL计算表记录和要搜索字符串之间的相似程度，然后只返回那些相似度比较高的记录。如：

```
mysql> ALTER TABLE books ADD FULLTEXT INDEX (synopsis);
Query OK, 83 rows affected (0.55 sec)
Records: 83  Duplicates: 0  Warnings: 0
```

注释：FULLTEXT 索引只支持 MyISAM 表。

一旦创建了索引，把要搜索的字符串作为AGAINST()函数的参数，就可以使用MATCH()函数搜索它。考虑下面的例子：

```
mysql> SELECT title, MATCH(synopsis) AGAINST ('suspense') AS score FROM books
LIMIT 0, 10;
+-----------------------------------+-----------------+
| title                             | score           |
+-----------------------------------+-----------------+
| The Prometheus Deception          |               0 |
| Dark Hollow                       | 2.5951748101926 |
| Easy Prey                         |  2.703356073143 |
| Prayers For Rain                  | 2.8519631063088 |
| Roses Are Red                     | 2.8209489868374 |
| Personal Injuries                 |               0 |
| Demolition Angel                  |               0 |
| Code To Zero                      |               0 |
| Adrian Mole: The Cappuccino Years |               0 |
| The Bear And The Dragon           |               0 |
+-----------------------------------+-----------------+
10 rows in set (0.11 sec)
```

传递给MATCH()函数的参数必须是一个字段列表，它准确映射了表的某些FULLTEXT索引。然后MATCH()函数计算要搜索的字符串和表里每条记录的命名字段之间的相似度。按照http://www.mysql.com/documentation/上MySQL手册的说明，该函数在一些列参数的基础上为相似度评分，这些参数如下：

- 每一行词的个数
- 每一行惟一词的个数
- 集合中所有词的总个数
- 包含一个特定词的行数

相似度分数为0，表示比较的值一点也不相似。

在表的所有记录中出现概率大于50%的词（所谓的无用词）会被忽略，认为与全文搜索的目的无关。同样，出现频率多的词对于索引的作用要比出现频率少的词的权重小。

当决定哪些词应该包含在FULLTEXT索引中时，可以通过ft_stopword_file，ft_min_word_len和ft_max_word_len服务器配置变量来分别调节无用词列表、最小单词长度和最大单词长度。请看第13章关于如何改变服务器变量的更多信息。

通常情况下，我们可以在一个WHERE从句中使用MATCH()函数来检索那些具有高相似度的记录，如下例所示：

```
mysql> SELECT title, author FROM books WHERE MATCH (synopsis) AGAINST
('suspense');
+------------------+------------------+
| title            | author           |
+------------------+------------------+
| Prayers For Rain | Dennis Lehane    |
| Roses Are Red    | James Patterson  |
| Easy Prey        | John Sandford    |
| Dark Hollow      | John Connolly    |
+------------------+------------------+
4 rows in set (0.06 sec)
```

布尔搜索

在MySQL 4.0.1以及更高版本中，我们可以添加IN BOOLEAN MODE修饰符或者向AGAINST()函数的参数添加一个或者多个布尔操作符来执行对FULLTEXT索引的布尔搜索。下面的例子阐述了这些问题。第一个例子返回了synopsis字段中包含“crime”和“suspense”词的所有记录，第二个例子列出了在synopsis字段中包含“romance”而不是“teenage”或者“period”的所有记录。

```
mysql> SELECT title, author FROM books WHERE MATCH (synopsis) AGAINST ('+suspense
+crime' IN BOOLEAN MODE);
+-------------------------+------------------+
| title                   | author           |
+-------------------------+------------------+
| Void Moon               | Michael Connelly |
| Burnt Sienna            | David Morrell    |
| Cradle And All          | James Patterson  |
```

```
| The Prometheus Deception | Robert Ludlum     |
| Roses Are Red            | James Patterson   |
| Mystic River             | Dennis Lehane     |
| Right As Rain            | George Pelecanos  |
+--------------------------+-------------------+
7 rows in set (0.00 sec)

mysql> SELECT title, author FROM books WHERE MATCH (synopsis) AGAINST
('+romance -teenage -period' IN BOOLEAN MODE);
Empty set (0.00 sec)
```

从零开始

当我们修改影响FULLTEXT索引的服务器变量时，激活修改后重建的索引是必要的。一种重建的方法是删除它，然后重新创建。不过，MySQL手册支持另外一种更快的选择：使用带有QUICK参数的REPAIR TABLE命令来自动重建索引，如下例所示：

```
mysql> REPAIR TABLE books QUICK;
+----------------+--------+----------+----------+
| Table          | Op     | Msg_type | Msg_text |
+----------------+--------+----------+----------+
| db20139a.books | repair | status   | OK       |
+----------------+--------+----------+----------+
1 row in set (0.06 sec)
```

请注意，FULLTEXT索引是MySQL的新增功能，而且对于大的表它能很好地工作。对于索引，小表没有足够的数据值来进行优化操作。另外，它们还不能对产生的环境都进行完全的优化，所以需要有节制地使用。

提示： 为了更快地进行全文检索，在输入数据以后，应使用CREATE INDEX或者ALTER TABLE命令向一个表中添加FULLTEXT索引，而不是在创建表时产生索引。

8.4.4 主键

可以使用PRIMARY KEY约束为一个表指定主键。在一个设计好的数据库中，主键对每条记录都是不变的、惟一的标识符。如果一个键被声明为主键，这通常表示主键内的值很少被修改。

主键约束被认为是NOT NULL和UNIQUE约束最好的结合，因为它要求指定字段的值不能是空，也不能与其他行重复。考虑下面的例子，它通过创建一个主键为sku的存货清单表阐述了这件事：

```
mysql> CREATE TABLE inventory (
    -> sku INT(3) NOT NULL,
    -> item VARCHAR(255) NOT NULL,
    -> PRIMARY KEY(SKU)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO inventory (sku, item) VALUES (99, 'Ancient Ming vase');
Query OK, 1 row affected (0.07 sec)

mysql> INSERT INTO inventory (sku, item) VALUES (99, 'Gold sceptre with
rubies');
```

```
ERROR 1062: Duplicate entry '99' for key 1

mysql> INSERT INTO inventory (sku, item) VALUES (NULL, 'Gold sceptre with
rubies');
ERROR 1048: Column 'sku' cannot be null
```

在这个例子中，因为sku列被定义为主键，所以MySQL不允许在该字段中有重复的值或者NULL值。这就使数据库管理员能够保证表中已命名的每一条记录都有惟一的sku值，从而使存储的数据具有较高的一致性。

主键约束可以用于一个单独的字段，也可以用于多字段的组合。考虑下面的例子，它通过建立一个包含组合主键的表来说明：

```
mysql> CREATE TABLE firewall (
    -> host VARCHAR(11) NOT NULL,
    -> port SMALLINT(4) NOT NULL,
    -> access ENUM('DENY', 'ALLOW') NOT NULL,
    -> PRIMARY KEY(host, port)
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql> INSERT INTO firewall (host, port, access) VALUES ('202.65.3.87', 21,
'DENY');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO firewall (host, port, access) VALUES ('202.65.3.87', 21,
'ALLOW');
ERROR 1062: Duplicate entry '202.65.3.87-21' for key 1

mysql> INSERT INTO firewall (host, port, access) VALUES ('202.65.3.87', 80,
'DENY');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO firewall (host, port, access) VALUES ('127.0.0.1', 21,
'ALLOW');
Query OK, 1 row affected (0.00 sec)
```

在这种情况下，表规则允许主机IP地址或者端口的重复，但不能同时重复。当一条记录可以被它的属性组合惟一标识而不是通过一个属性标识时，这变得非常方便。

8.4.5　外键

像MySQL这样的关系型数据库管理系统，它们的基础是在数据库的表之间创建关系的能力。通过方便地在不同表中建立记录到记录的联系，RDBMS可以利用不同的方法分析数据，同时保持数据库以系统的方式、最小的冗余进行组织。

这些关系基本上依靠外键进行管理，在关系中所有表中具有相同含义的字段作为公共部分来连接不同表中的记录。外键关系可以是一对一的，一个表的记录只能与另一个表的一条记录连接，或者是一对多的，一个表的记录与另一个表的多条记录连接。

键的别名

MySQL中，术语“键”和“索引”是等同的。因而，MySQL会自动为所有表的主键字段进行索引，但是外键字段必须由用户进行明确地索引。

图 8.1 说明了一对一关系：SERVICE_MASTER 和 SERVICE_INFO 之间的关系，通过惟一的 serviceID 字段确立了二者的关系。

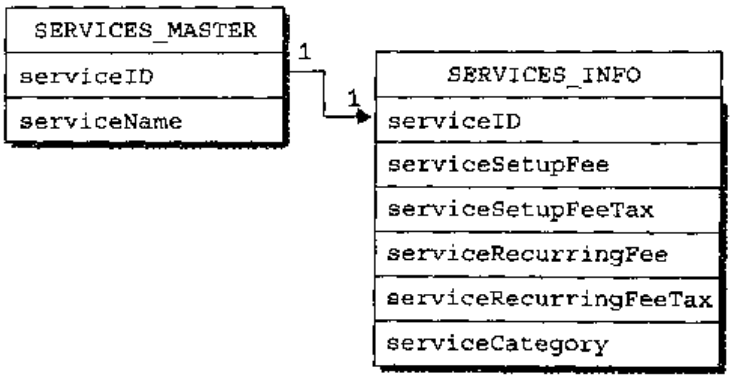

图 8.1 表间的一对一关系

图 8.2 说明了一对多关系：一个公司和它的分支机构的列表，通过惟一的 companyID 字段在两个表之间进行了连接。

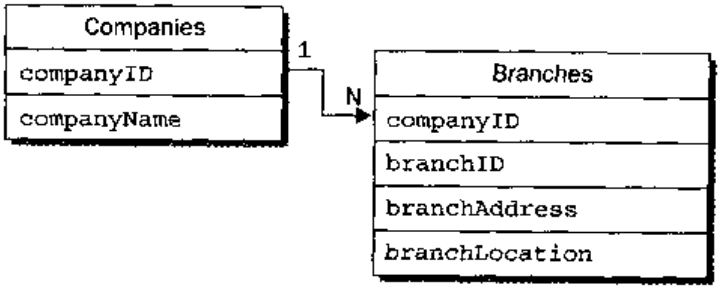

图 8.2 表间的一对多关系

当外键与另一个表的字段有关系而且这种关系是惟一时，这个系统就称为处于参照完整的状态。也就是说，如果一个字段在所有的表中只出现一次，而且每一个表的这个字段的变化都会影响其他表，这时就存在参照完整性。

参照完整性的概念是一个基本的概念，而且当设计的数据库含有多个表时这个概念就变得很重要。当使用外键来建立一个表到另一个表的连接时，参照完整性就会在插入新记录和更新已存在的记录时强加约束。例如，如果一个表的某个字段只接受某种类型的值，另外的表把该字段作为它们的外键，这时就会向相关表中强加某种约束。同样，参照完整性要求作为外键的字段变化时（删除或者插入），需要在所有的相关表中都有所反映。

现在大部分数据库都会自动地考虑这些。例如，如果使用 Microsoft Access，就能看到这一点。如果数据库不能自动地执行这个功能，保持参照完整性的任务就必须由应用程序的开发者完成，这些开发者必须手动地更新所有相关表来反映外键值的变化。

MySQL 在这个特性上持观望状态，它允许使用外键，但是为了完整性检验的目的，在除了 InnoDB 表类型之外的所有表类型中都忽略了这个功能（从 MySQL 3.23.44 开始）。这可能看起来有些异常，但是原因是非常实际的：对于数据库的所有外键的每次插入、更新和删除后，进行完整性检查是一个耗费时间和资源的过程，它可能影响性能，特别是当处理复杂的或者是缠绕的连接树时。因而，用户可以在表的基础上，选择适合于特定需求的最好结合。

在下面的例子中，employees.fk_department 字段是一个外键，与 departments.id 主键连接。注意FOREIGN KEY…REFERENCES修饰符中指定关系的方式。FOREIGN KEY部分指定关系的一端（当前表的字段名），REFERENCES 部分指定关系的另一端（参照表的字段名）。

当创建一个表时，通过FOREIGN KEY…REFERENCES修饰符，使用与定义一个主键相同的方法来定义外键。下面的例子通过创建以一对多关系连接的两个 InnoDB 表来阐述，它们使用部门 ID 进行连接：

```
mysql> CREATE TABLE departments (
-> id INT(4) NOT NULL,
-> name VARCHAR(50) NOT NULL,
-> PRIMARY KEY (id)
-> ) TYPE=INNODB;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE employees (
-> id SMALLINT(6) NOT NULL,
-> name CHAR(255) NOT NULL,
-> fk_department INT(4) NOT NULL,
-> PRIMARY KEY (id),
-> INDEX (fk_department),
-> FOREIGN KEY (fk_department) REFERENCES departments (id)
-> ) TYPE=INNODB;
Query OK, 0 rows affected (0.16 sec)
```

句法错误

当创建外键参照时，我们可能遇到以下关于错误代码为 150 的错误信息：

```
ERROR 1005: Can't create table './test/employees.frm' (errno: 150)
```

如果看到这样的信息，它表示外键声明发生了错误。为避免此类错误，应确保参照的表和字段是存在的，组成外键的字段被索引，声明的句法是正确的。也可以通过使用 SHOW INNODB STATUS 命令或者浏览关于外键错误的输出信息来获得关于错误的更多信息。

做出选择

如果需要更好的性能，并且不需要完整性检查，我们可以选择使用 MyISAM 类型，如果想要在表中强制进行外键完整性约束，而且还要在这种需要的基础上保持好的性能，就要选择 InnoDB 类型。在 8.4.6 节将详细地讨论这两种表类型。

一旦建立了一个外键，MySQL 只允许向 employees.fk_department 字段输入那些也存在于 departments.id 字段的值。继续前面的例子，让我们看看它如何工作的：

```
mysql> INSERT INTO departments (id, name) VALUES (101, 'Engineering'), (102,
'Administration'), (103, 'Finance');
Query OK, 3 rows affected (0.05 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> INSERT INTO employees (id, name, fk_department) VALUES (57, 'John D', 102);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO employees (id, name, fk_department) VALUES (101, 'Thomas
E', 110);
ERROR 1216: Cannot add or update a child row: a foreign key constraint fails
```

因为 *ID* 为 110 的部门不存在，所以 MySQL 在雇员表中的 fk_ department 字段拒绝接受具有这个值的记录。在这种方式下，外键约束可以很大程度地帮助我们对数据库的表强加数据完整性，这样会减少“坏”的或者不一致的字段值出现的次数。

当以这种方式使用外键连接表时，必须清楚下面的三个约束：

- 关系中的所有表必须是 InnoDB 表。在非 InnoDB 表中，MySQL 将会忽略 FOREIGN KEY…REFERENCES 修饰符。
- 用于外键关系的字段必须在所有的参照表中进行明确地索引，InnoDB 不能自动地创建索引。
- 在外键关系中，字段的数据类型必须相似，这对于大小和符号都必须匹配的整数类型尤其重要。

需要注意的是：即使表存在外键约束，MySQL 还允许我们删除表，并且不会产生错误（即使这样做可能会破坏更早创建的外键）。实际上，在 4.0.13 以前的 MySQL 版本中，删除表是惟一删除外键的方法。不过，MySQL 4.0.13 以及更高的版本通过 ALTER TABLE 命令提供了一种从表中删除外键的比较缓和的方法。句法如下：

```
ALTER TABLE table-name DROP FOREIGN KEY key-id;
```

通过 SHOW CREATE TABLE 命令可以获得 *key-id* 的值（在 8.5 节进行了详细的讨论）。考虑下面的例子，它阐述了这个过程：

```
mysql> SHOW CREATE TABLE employees;
+-----------+-----------------------------------------------------------------+
| Table     | Create Table                                                    |
+-----------+-----------------------------------------------------------------+
| employees | CREATE TABLE 'employees' (                                      |
| 'id' smallint(6) NOT NULL default '0',                                      |
|  'name' char(255) NOT NULL default '',                                      |
|  'fk_department' int (4) NOT NULL default '',                               |
|  PRIMARY KEY ('id'),                                                        |
|  KEY 'fk_department' ('fk_department'),                                     |
|  CONSTRAINT '0_18' FOREIGN KEY ('fk_department')                            |
|  REFERENCES 'departments' ('id')                                            |
|  ) TYPE=InnoDB                                                              |
+-----------+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

现在可以使用标识符 0_18 删除外键：

```
mysql> ALTER TABLE employees DROP FOREIGN KEY 0_18;
Query OK, 2 rows affected (0.03 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

打破规则

当通过一系列操作填充表时，也就是说，在 mysql 命令提示符下（或者通过 LOAD DATA INFILE 命令）转储文件是关掉外键检查的好主意，这样 MySQL 不会抱怨违背完整性。一种操作的方法是只在创建表和填充它以后再使用 ALTER TABLE 命令设置外键约束。另外一种方法是通过设置 foreign_key_checks 服务器变量为 0 来关掉键的检查。

可以在 8.6 节看到关于 ALTER TABLE 命令的更多内容，还可以在第 13 章找到修改服务器变量的内容。

自动键更新和删除

外键当然可以保证新插入的记录的完整性。但是，如果在 REFERENCES 子句中从已命名的表删除记录会怎样？在使用同样的值作为外键的辅助表中会发生什么？

显而易见，那些记录也应该被删除，否则在数据库中就会有很多杂七杂八的孤立记录。MySQL 3.23.50 以及更高的版本通过向 FOREIGN KEY…REFERENCES 修饰符添加一个 ON DELETE 子句简化了这项任务，它告诉了数据库在这种情况下如何处理孤立记录。

考虑下面的例子，它通过普通的分类 ID 连接了类别和文章：

```
mysql> CREATE TABLE categories (cid INT(4) NOT NULL, cname VARCHAR(50) NOT
NULL, PRIMARY KEY (cid)) TYPE=INNODB;
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO categories (cid, cname) VALUES (1, 'News'), (2, 'Fashion'),
(3, 'Sports');
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> CREATE TABLE articles (aid INT(4) NOT NULL, aname VARCHAR(255) NOT
NULL, fk_cid INT(4) NOT NULL, PRIMARY KEY (aid), INDEX category_id (fk_cid),
FOREIGN KEY (fk_cid) REFERENCES categories (cid) ON DELETE CASCADE) TYPE=INNODB;
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO articles (aid, aname, fk_cid) VALUES (1, 'Sample article
on winter fashions', 2);
Query OK, 1 row affected (0.05 sec)
```

下面是运行这些命令后的结果：

```
mysql> SELECT * FROM categories;
+-----+---------+
| cid | cname   |
+-----+---------+
|   1 | News    |
|   2 | Fashion |
|   3 | Sports  |
+-----+---------+
3 rows in set (0.06 sec)

mysql> SELECT * FROM articles;
+-----+-----------------------------------+--------+
| aid | aname                             | fk_cid |
+-----+-----------------------------------+--------+
```

```
|   1 | Sample article on winter fashions |      2 |
+-----+-----------------------------------+--------+
1 row in set (0.00 sec)
```

如果从categories表中删除一个类别，ON DELETE CASCADE子句能够保证与该类别相连接的所有文章也都从articles表中被删除。从下面片段可以清楚地看到这些：

```
mysql> DELETE FROM categories WHERE cname = 'Fashion';
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM categories;
+-----+--------+
| cid | cname  |
+-----+--------+
|   1 | News   |
|   3 | Sports |
+-----+--------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM articles;
Empty set (0.00 sec)
```

因而，当删除一个类别时，MySQL会自动删除与它连接的表中和该类别相关的所有记录。表8.2列出了可以跟随ON DELETE子句的4个关键字。

表8.2 DELETE子句中可用的操作

关键字	含义
CASCADE	删除包含与已删除键值有参照关系的所有记录
SET NULL	修改包含与已删除键值有参照关系的所有记录，使用NULL值替换（这只能用于已标记为NOT NULL的字段）
RESTRICT	拒绝删除要求，直到使用删除键值的辅助表被手工删除，并且没有参照时（这是默认设置，也是最安全的设置）
NO ACTION	什么也不做

注释： MySQL的ON DELETE或者ON UPDATE子句不支持SET DEFAULT行为。尽管在其他的RDBMS中，如果在外键关系中进行删除或者修改操作，SET DEFAULT行为可以用来向它的默认值创建一个字段。

下面是前面例子的变化，演示了SET NULL关键字是如何工作的：

```
mysql> DELETE FROM categories WHERE cname = 'Fashion';
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM articles;
+-----+--------+--------+
| aid | aname  | fk_cid |
+-----+--------+--------+
|   1 | sample |   NULL |
+-----+--------+--------+
1 row in set (0.00 sec)
```

当使用默认的RESTRICT关键字时，会发生什么？MySQL拒绝进行删除，因为与这个键相关的记录仍然在articles表中存在：

```
mysql> DELETE FROM categories WHERE cname = 'Fashion';
ERROR 1217: Cannot delete or update a parent row: a foreign key constraint fails
```

顾名思义，NO ACTION 关键字就是什么也不做（并且导致在 articles 表中产生孤立记录）：

```
mysql> DELETE FROM categories WHERE cname = 'Fashion';
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM categories;
+-----+---------+
| cid | cname   |
+-----+---------+
|   1 | News    |
|   3 | Sports  |
+-----+---------+
2 rows in set (0.06 sec)

mysql> SELECT * FROM articles;
+-----+-----------------------------------+--------+
| aid | aname                             | fk_cid |
+-----+-----------------------------------+--------+
|   1 | Sample article on winter fashions |      2 |
+-----+-----------------------------------+--------+
1 row in set (0.00 sec)
```

MySQL 4.0.8以及更高版本还允许通过使用ON UPDATE子句来让我们自动进行更新操作，ON UPDATE 子句和 ON DELETE 子句的工作方式相似。例如，向外键定义添加 ON UPDATE CASCADE 子句，就会告诉 MySQL，如果主表（作为外键检查的参照表）的记录进行了更新，则当前表中使用该外键值的所有记录都应该自动地更新为新值，以确保系统的一致性。

例子可能会更清楚地说明这一点。考虑下面两个表，它们通过状态代码的 ON UPDATE CASCADE 条件进行连接：

```
mysql> SELECT * FROM states;
+------+------------+
| code | name       |
+------+------------+
| AZ   | Arizona    |
| CA   | California |
| NY   | New York   |
+------+------------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM votes;
+-------+--------+
| state | total  |
+-------+--------+
| AZ    |  46292 |
| CA    | 123661 |
| NY    |    648 |
+-------+--------+
3 rows in set (0.00 sec)
```

如果我们已经或者现在更改 states 表中的代码，ON UPDATE CASCADE 子句可以保证 MySQL 自动更新 votes 表的所有记录为新状态的代码。下面的例子清楚说明了这一点：

```
mysql> UPDATE states SET code = 'AR' WHERE name = 'Arizona';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM votes;
+-------+--------+
| state | total  |
+-------+--------+
| AR    |  46292 |
| CA    | 123661 |
| NY    |    648 |
+-------+--------+
3 rows in set (0.05 sec)
```

连锁反应

请注意，通过ON UPDATE和ON DELETE规则，设置MySQL能够实现自动操作时，如果键的关系没有设置好，可能会导致严重的数据破坏。例如，如果一系列的表通过外键关系和ON DELETE CASCADE 规则连接时，任意一个主表的变化都会导致甚至只和原始删除有一些次要联系的记录在没有警告的情况下被删除。所以，我们在操作之前要检查这些规则，操作之后还要再次检查。

8.4.6 表类型

一旦完成了多种字段的定义和修饰，MySQL可以让我们使用TYPE属性指定表的类型，这个操作是可选的。如下例所示：

```
mysql> CREATE TABLE users (
-> id INT(3),
-> name VARCHAR(8),
-> pass VARCHAR(20)
-> ) TYPE=MyISAM;
Query OK, 0 rows affected (0.08 sec)
```

可以得到很多不同的表类型，每一种都具有它自己的属性和优点。下面部分将详细地讨论多种类型。

MyISAM 表类型

MyISAM表（TYPE = MYISAM）延伸了基本的ISAM类型，具有很多优化和增强的特性，也是MySQL的默认表类型。MyISAM表优化了压缩比例和速度，而且可以在不同的操作系统和平台之间进行移植（例如，同样的MyISAM表可以应用于Windows和UNIX两种操作系统）。MyISAM格式支持大表文件（大于4 GB），并且允许对BLOB和TEXT列进行索引。表和表索引可以被压缩以节省空间，当存储大的BLOB或者TEXT字段时，这个特性变得很实用。可以约束VARCHAR字段到指定的长度或者自动根据数据的情况进行动态调节，而且这种格式支持使用键前缀和使用完整的键搜索记录。

因为MySQL优化了MyISAM表，所以开发者可以向其中添加相当多的智能。MyISAM可以是定长的或者是可变长度的。MySQL在启动时为了防止崩溃，自动检查MyISAM表，甚至可以在出现错误的时候进行修复。表数据和表索引文件可以储存在不同的位置（甚至在不同的文件系统中）。而且即使是具有相当多的插入、更新和删除操作的表，智能防碎片逻辑也能保证其高性能的协作性。

兼　容

为了与旧表兼容，MySQL包含了ISAM表处理器。实际上，我们应该避免创建这种格式的表，因为新的MyISAM和InnoDB表格式在性能和特性上都优越得多。

ISAM表类型

ISAM表（TYPE = ISAM）和MyISAM表相似，但是它们缺少MyISAM格式的很多增强性能，因而不能像MyISAM类型那样提供很好的优化和执行效率。因为ISAM索引不能被压缩，它比在MyISAM中的相同索引占用较少的系统资源。ISAM索引需要较多的磁盘空间，这对于小型环境是个问题。

和MyISAM一样，ISAM表可以是固定长度的，也可以是可变长度的，但是ISAM格式的最大键长度比较小，ISAM格式处理的表不能大于4 GB，而且表不能在不同的平台间移植。另外，ISAM表容易分裂，这会降低查询速度，对数据/索引的压缩产生限制。

HEAP表类型

HEAP表（TYPE = HEAP）是内存中的表，它使用能够比较快速的散列索引（当运行INSERT查询时，独立评价指出HEAP表最少比MyISAM表快30%），因此，对于临时表可以优化。它和MyISAM或ISAM表的访问规则和使用方式一样。然而，存储在里面的数据只在MySQL服务器的生命期内存在，如果MySQL服务器崩溃或者被关掉，都会擦除其中的数据。虽然HEAP表具有性能方面的好处，但是由于它的临时性和一些其他功能限制，在实际中不可能经常使用。

耗尽RAM

因为HEAP表的大小只受到系统上可用内存的限制，MySQL具有内建保护来阻止无意识地使用所有可用内存。HEAP表不支持BLOB或TEXT列，它们也不能超过max_heap_table_size变量指定的大小。还可以通过向CREATE TABLE语句中添加MAX_ROWS属性来指定HEAP表的大小。如下例所示：

```
CREATE TABLE heapexample (id INT(4), title VARCHAR(24)) TYPE=HEAP
MAX_ROWS=10;
```

BerkeleyDB表类型

BerkeleyDB表（TYPE = BDB）在MySQL 3.23.34中第一次被引入，它是为满足MySQL开发者对事务安全表日益增长的需求而发展起来的。BerkeleyDB表具有很多有趣的特性，包括提交和回滚操作、多用户并发访问、检查点、次要索引、通过日志恢复崩溃、连续地和键控地访问数据等，这使得复杂的、基于事务的SQL有了可行的选择。

然而，BerkeleyDB表也有一些限制：它的移动比较困难（在创建时，表路径硬编码在表文件中），不能压缩表索引，而且表通常比它的MyISAM相应的表要大。由于这些原因，

InnoDB 格式很大程度上可以取代 BerkeleyDB 格式，它会消除这些问题并且提供许多附加的特性。

InnoDB 表类型

InnoDB 表（TYPE = INNODB）在 MySQL 3.23 中第一次出现，而且默认包含在 MySQL 4.x 中。InnoDB 是一个完全兼容 ACID 的、高效率的表，它完全支持 MySQL 的事务处理并且不会降低速度或者性能。精细的（行级和表级）锁定提高了 MySQL 事务处理的逼真度，同时 InnoDB 也支持无锁定读操作和多版本的特性（前一个特性只在 Oracle 中包含）。

异步输入/输出和一系列的读缓冲将提高数据检索速度，同时“伙伴算法”和 Oracle 类型的表空间可以进行文件的优化和内存的管理。InnoDB 在需要的基础上支持自动在内存上创建散列索引来提高性能，同时它使用缓冲来提高可靠性和数据库操作的速度。InnoDB 表的性能可以和 MyISAM 表相媲美（有时甚至超过）。由于它们的事务处理特性，它们在不同的操作系统和体系结构上是完全可移植的。它们一直处于一致的状态（MySQL 通过在启动时检查错误并修复错误来使它们更加健壮），所以 InnoDB 格式很快地在 MySQL 开发者中流行。对外键、提交、回滚和前滚的操作的支持，使这种表成为 MySQL 中最完善的表格式。

MERGE 表类型

MERGE 表（TYPE = MERGE）是通过把多个 MyISAM 表组合到一个单独的表来创建的一种虚拟表。只有涉及到的表具有完全相同的表结构时才能对表进行组合。字段类型或者索引的任何不同都不能进行成功的结合。MERGE 表使用组成表的索引，并且不能维持它本身的索引，在某种情况下这可以提高速度。

打　包

当处理大的 MyISAM 表时，我们可以通过压缩或者使用 MySQL 发布中包含的 myisampack 实用工具进行“打包”来减少这些表占用的空间。myisampack 创建比较小的只读表，而不会在使用智能压缩时导致任何大的性能开销。

MERGE 表允许 SELECT，DELETE 和 UPDATE 操作。在需要把不同表的数据放到一起、提高连接的性能或者在一系列表中进行搜索时，这种表很实用。

8.4.7 其他表修饰符

TYPE 属性不是控制已创建表的行为的惟一选择。MySQL 许多其他的特有属性也是可行的，表 8.3 列出了其中比较有趣的一些。

表 8.3　其他 MySQL 表修饰符

类型	描述
AUTO_INCREMENT	下一次向 AUTO_INCREMENT 字段插入的值
CHECKSUM	是否存储表校验和的布尔表示
COMMENT	表的描述性注释
MAX_ROWS	表中存储的最大行数
MIN_ROWS	表中存储的最小行数
PACK_KEYS	是否压缩表索引的布尔表示
UNION	映射到一个单独 MERGE 表的表
DATA DIRECTORY	表数据文件的位置
INDEX DIRECTORY	表索引文件的位置

在数据库中，MySQL 支持在 CREATE TABLE 命令中使用 IF NOT EXISTS 子句。在编辑脚本内容中是非常有用的，因为它可以帮助我们避免存在具有相同名字的表的错误。如：

```
mysql> CREATE TABLE IF NOT EXISTS users (
-> id INT(3),
-> name VARCHAR(8),
-> pass VARCHAR(20)
-> );
Query OK, 0 rows affected (0.00 sec)
```

8.5 复制表

在 MySQL 3.23 以及以后的版本，我们可以结合 CREATE TABLE 和 SELECT 语句从一个 SELECT 查询的结果动态地创建和填充表。例如，假设有下面的表：

```
mysql> SELECT * FROM marks;
+----+-------+-------+------+-----+-----+
| id | fname | lname | math | sci | lit |
+----+-------+-------+------+-----+-----+
|  1 | John  | Doe   |   36 |  73 |  34 |
|  2 | Sarah | Short |   23 |  67 |  55 |
|  3 | Joe   | Cool  |   75 |  82 |  89 |
|  4 | Mark  | Wumba |   58 |  72 |  15 |
|  5 | Tom   | Thumb |    0 |  12 |   0 |
+----+-------+-------+------+-----+-----+
5 rows in set (0.00 sec)
```

我们想要创建这个表的一份副本：

```
mysql> CREATE TABLE grades SELECT * FROM marks;
Query OK, 5 rows affected (0.07 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM grades;
+----+-------+-------+------+-----+-----+
| id | fname | lname | math | sci | lit |
+----+-------+-------+------+-----+-----+
|  1 | John  | Doe   |   36 |  73 |  34 |
|  2 | Sarah | Short |   23 |  67 |  55 |
|  3 | Joe   | Cool  |   75 |  82 |  89 |
|  4 | Mark  | Wumba |   58 |  72 |  15 |
|  5 | Tom   | Thumb |    0 |  12 |   0 |
+----+-------+-------+------+-----+-----+
5 rows in set (0.00 sec)
```

可以通过在 SELECT 查询中指定字段来限制出现在副本表中的字段：

```
mysql> CREATE TABLE students SELECT fname, lname FROM marks;
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM students;
```

```
+-------+-------+
| fname | lname |
+-------+-------+
| John  | Doe   |
| Sarah | Short |
| Joe   | Cool  |
| Mark  | Wumba |
| Tom   | Thumb |
+-------+-------+
5 rows in set (0.00 sec)
```

向查询中添加可选的 WHERE 子句，可以向新表中只放置原始数据的一个子集：

```
mysql> CREATE TABLE grades SELECT * FROM marks WHERE (math + sci) > 100;
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM grades;
+----+-------+-------+------+-----+-----+
| id | fname | lname | math | sci | lit |
+----+-------+-------+------+-----+-----+
|  1 | John  | Doe   |   36 |  73 |  34 |
|  3 | Joe   | Cool  |   75 |  82 |  89 |
|  4 | Mark  | Wumba |   58 |  72 |  15 |
+----+-------+-------+------+-----+-----+
3 rows in set (0.00 sec)
```

或者通过使用 SELECT 语句创建已存在表的空副本，并且返回一个空结果集：

```
mysql> CREATE TABLE grades SELECT * FROM marks WHERE 0 = 1;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM grades;
Empty set (0.00 sec)

mysql> DESCRIBE grades;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| id    | tinyint(4) |      |     | 0       |       |
| fname | char(50)   |      |     |         |       |
| lname | char(50)   |      |     |         |       |
| math  | int(11)    |      |     | 0       |       |
| sci   | int(11)    |      |     | 0       |       |
| lit   | int(11)    |      |     | 0       |       |
+-------+------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

注释： CREATE TABLE…SELECT 结构不能从源表中复制键，需要手工创建。

MySQL 4.1 提供另外一种完成前面例子的方法：

```
mysql> CREATE TABLE grades LIKE marks;
```

```
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

可以使用 SHOW CREATE TABLE 命令来检索由 SQL 命令创建的表。在需要时，这些信息可以用来复制表结构。考虑下面的例子，它对创建特定 MySQL 主机授权表的 CREATE TABLE 句法进行了说明：

```
mysql> SHOW CREATE TABLE mysql.host;
+--------+-----------------------------------------------------------------+
| Table  | Create Table                                                    |
+--------+-----------------------------------------------------------------+
| host   | CREATE TABLE `host` (                                           |
|        | `Host` char(60) binary NOT NULL default '',                     |
|        | `Db` char(64) 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',                 |
|        | `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',                |
|        | `Create_tmp_table_priv` enum('N','Y') NOT NULL default 'N',     |
|        | `Lock_tables_priv` enum('N','Y') NOT NULL default 'N',          |
|        | PRIMARY KEY  (`Host`,`Db`)) TYPE=MyISAM                         |
|        | COMMENT='Host privileges;  Merged with database privileges'     |
+--------+-----------------------------------------------------------------+
1 row in set (0.06 sec)
```

前面例子的结果是由 CREATE TABLE 和 SELECT 语句结合创建的一个混合表，这个表包含新字段和来自 SELECT 结果集的字段。考虑下面的例子，它创建了一个包含现有表（通过 SELECT 查询获得）中的两列和新的两列的表（在 CREATE TABLE 中定义）：

```
mysql> CREATE TABLE students (dob DATE NOT NULL,
sex SET('M', 'F') NOT NULL) SELECT fname, lname FROM marks;
Query OK, 5 rows affected (0.04 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM students;
+------------+-----+-------+-------+
| dob        | sex | fname | lname |
+------------+-----+-------+-------+
| 0000-00-00 |     | John  | Doe   |
| 0000-00-00 |     | Sarah | Short |
| 0000-00-00 |     | Joe   | Cool  |
| 0000-00-00 |     | Mark  | Wumba |
| 0000-00-00 |     | Tom   | Thumb |
+------------+-----+-------+-------+
5 rows in set (0.01 sec)
```

临 时 表

MySQL 允许我们使用 CREATE TEMPORARY TABLE 命令创建临时表。这种表之所以这样称呼是因为它只在 MySQL 会话期内存在，当客户关闭了与 MySQL 服务器的连接时，这种表就会被自动删除。这种表对于短暂的、基于会话的数据、计算或数据的临时存储是非常方便的。因为它是依赖于会话的，所以两个不同的会话使用相同的表名字是不冲突的。

下面是一个例子：

```
mysql> CREATE TEMPORARY TABLE session_key (skey INT(6) NOT NULL
PRIMARY KEY);
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO session_key VALUES (52347382934);
Query OK, 1 row affected (0.07 sec)

mysql> SELECT * FROM session_key;
+------------+
| skey       |
+------------+
| 2147483647 |
+------------+
1 row in set (0.00 sec)

mysql> exit
$ mysql
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 25 to server version: 4.0.12-max-debug

mysql> SELECT * FROM session_key;
ERROR 1146: Table 'db1.session_key' doesn't exist
```

因而，session_key 表只在客户连接时间段内是可用的。一旦中止了客户会话，MySQL 会自动删除临时表以及它的内容。

8.6 修改表

可以使用 ALTER TABLE 命令来更改已存在的表，它允许在不重建整个表的情况下添加、删除或者修改表字段。如下所示：

```
ALTER TABLE table-name (action field-definition,
action field-definition,···);
```

这里的 action 可以是关键字 ADD，DROP，ALTER 或者 CHANGE 中的任一个，而且后面跟的是与 CREATE TABLE 命令使用的相同的字段定义。这个定义包含要修改的字段的名字和（根据操作）由新字段名字、类型和约束组成的字段定义。考虑下面的例子，它通过首先创建一个表，然后向其中添加新列来说明：

```
mysql> CREATE TABLE members (mid INT(3), mname CHAR(8),
mpass VARCHAR(25));
Query OK, 0 rows affected (0.04 sec)

mysql> ALTER TABLE members ADD memail varchar(255) NOT NULL;
```

```
Query OK, 0 rows affected (0.15 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE members;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| mid    | int(3)       | YES  |     | NULL    |       |
| mname  | varchar(8)   | YES  |     | NULL    |       |
| mpass  | varchar(25)  | YES  |     | NULL    |       |
| memail | varchar(255) |      |     |         |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

下面的例子向表添加一个新的主键：

```
mysql> ALTER TABLE members ADD PRIMARY KEY (memail);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE members;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| mid    | int(3)       | YES  |     | NULL    |       |
| mname  | varchar(8)   | YES  |     | NULL    |       |
| mpass  | varchar(25)  | YES  |     | NULL    |       |
| memail | varchar(255) |      | PRI |         |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0.02 sec)
```

修改已存在字段的名字和类型也很简单：

```
mysql> ALTER TABLE members CHANGE mid id INT(8) AUTO_INCREMENT UNIQUE;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE members;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| id     | int(8)       |      | UNI | NULL    | auto_increment |
| mname  | varchar(8)   | YES  |     | NULL    |                |
| mpass  | varchar(25)  | YES  |     | NULL    |                |
| memail | varchar(255) |      | PRI |         |                |
+--------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

下面是删除字段或者键：

```
mysql> ALTER TABLE members DROP mpass;
Query OK, 0 rows affected (1.42 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> DESCRIBE members;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| id     | int(8)       |      | UNI | NULL    | auto_increment |
| mname  | varchar(8)   | YES  |     | NULL    |                |
| memail | varchar(255) |      | PRI |         |                |
+--------+--------------+------+-----+---------+----------------+
3 rows in set (0.07 sec)
```

修　改

当修改字段类型时，MySQL 允许把 CHANGE 关键字替换为 MODIFY 关键字。两个字的区别很小：CHANGE 关键字要求在修改表时指定旧的和新的字段名，MODIFY 关键字只是修改字段的类型而不会更改它的名字。

通过下面的两个命令进行说明，这两个命令是相同的：

```
ALTER TABLE members CHANGE id id INT(8) AUTO_INCREMENT UNIQUE;
ALTER TABLE members MODIFY id int(8) AUTO_INCREMENT UNIQUE;
```

请注意，当我们把一个字段的类型变化或者修改成另一个时，MySQL 自动尝试把字段中的数据转变为新的类型。

可以使用 ALTER TABLE 命令重新命名表，如下例所示：

```
mysql> ALTER TABLE members RENAME TO memberlist;
Query OK, 0 rows affected (0.00 sec)
```

或者使用具有相同效果的 RENAME TABLE 命令：

```
mysql> RENAME TABLE members TO memberlist;
Query OK, 0 rows affected (0.01 sec)
```

可以使用 SET DEFAULT 和 DROP DEFAULT 子句为一个字段设置（或删除）默认值，还可以使用 ALTER TABLE 命令的 FIRST 和 AFTER 子句控制字段的位置：

```
mysql> ALTER TABLE members ADD status INT(1) AFTER id;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE members ALTER status SET DEFAULT 1;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE members;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| id     | int(8)       |      | UNI | NULL    | auto_increment |
| status | int(1)       | YES  |     | 1       |                |
| mname  | varchar(8)   | YES  |     | NULL    |                |
| memail | varchar(255) |      | PRI |         |                |
+--------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

向一个表添加 AUTO_INCREMENT 字段会使表中所有存在的记录被自动编号。例如，有这样的一个表：

```
mysql> SELECT * FROM movies;
+--------------------------+
| title                    |
+--------------------------+
| The Phantom Menace       |
| Cape Fear                |
| Casper                   |
| The Godfather            |
| Who Killed Roger Rabbit? |
+--------------------------+
5 rows in set (0.01 sec)
```

我们决定为每一个字段添加一个 AUTO_INCREMENT 主键：

```
mysql> ALTER TABLE movies ADD id INT(3) AUTO_INCREMENT PRIMARY KEY FIRST;
Query OK, 5 rows affected (0.03 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

结果如下：

```
mysql> SELECT * FROM movies;
+----+--------------------------+
| id | title                    |
+----+--------------------------+
|  1 | The Phantom Menace       |
|  2 | Cape Fear                |
|  3 | Casper                   |
|  4 | The Godfather            |
|  5 | Who Killed Roger Rabbit? |
+----+--------------------------+
5 rows in set (0.01 sec)
```

当向一个表添加UNIQUE键，而且这个键包含重复的值时，我们可以通过包含或者删除可选的IGNORE从句来控制MySQL处理这种情况的方法（或者删除所有重复的记录或者放弃变更）。考虑下面的例子，一个字段的两条记录具有相同的值：

```
mysql> SELECT * FROM users;
+--------+-----------------------------+
| name   | email                       |
+--------+-----------------------------+
| john   | john@some.domain.com        |
| moe    | moetheman@some.domain.net   |
| scooty | sarah@some.domain.org       |
| jb     | john@some.domain.com        |
| tom    | ttalam@some.other.domain.com |
| john   | john@free.mail.server.com   |
+--------+-----------------------------+
6 rows in set (0.00 sec)
```

现在，如果设置name字段为UNIQUE，MySQL就会返回一个关于字段值重复的错误：

```
mysql> ALTER TABLE users CHANGE name name VARCHAR(10) NOT NULL UNIQUE;
ERROR 1062: Duplicate entry 'john' for key 1
```

然而，如果指定ALTER TABLE命令的IGNORE子句，那么MySQL就会从表中删除在那个键上具有重复值的所有记录，只保留第一条记录，如下所示：

```
mysql> ALTER IGNORE TABLE users CHANGE name name VARCHAR(10)
NOT NULL UNIQUE;
Query OK, 6 rows affected (0.02 sec)
Records: 6  Duplicates: 1  Warnings: 0

mysql> SELECT * FROM USERS;
+--------+-----------------------------+
| name   | email                       |
+--------+-----------------------------+
| john   | john@some.domain.com        |
| moe    | moetheman@some.domain.net   |
| scooty | sarah@some.domain.org       |
| jb     | john@some.domain.com        |
| tom    | ttalam@some.other.domain.com |
+--------+-----------------------------+
5 rows in set (0.01 sec)
```

可以使用ADD FOREIGN KEY子句向一个表添加外键参照。如下例所示：

```
mysql> ALTER TABLE articles ADD FOREIGN KEY (fk_cid) REFERENCES categories
(cid) ON DELETE CASCADE ON UPDATE CASCADE;
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

可以通过向ALTER TABLE命令指定一个新的TYPE子句来更改表类型。考虑下面的例子，它把users表改为InnoDB表类型：

```
mysql> ALTER TABLE users TYPE = INNODB;
Query OK, 6 rows affected (0.11 sec)
Records: 6  Duplicates: 0  Warnings: 0
```

也可以通过在ALTER TABLE命令中包含ADD INDEX或者DROP INDEX子句添加和删除索引。下面的例子进行了说明：

```
mysql> ALTER TABLE articles ADD FULLTEXT INDEX content_001 (body);
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE articles DROP INDEX content_001;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

事半功倍

如同ALTER TABLE命令语法表明的那样，MySQL允许通过使用逗号分隔不同的行为和字段定义来一次性更改多个字段。这是MySQL对于SQL-92标准的特有的扩展。严格的SQL-92标准对每个操作都需要一个ALTER TABLE声明。因为ALTER TABLE是一个比较慢的命令，因此要尽可能少地执行，但MySQL这种特有的能力能很容易地减少服务器上的性能延迟。

8.7　删除表

可以使用DROP TABLE命令删除一个表，下面的例子是使用这个命令删除表tbl1：

```
mysql> DROP TABLE tbl1;
Query OK, 0 rows affected (0.02 sec)
```

和前面一样，也可以使用IF EXISTS子句来避免要删除的表不存在时出现讨厌的MySQL错误信息：

```
mysql> DROP TABLE tbl1;
ERROR 1051: Unknown table 'tbl1'

mysql> DROP TABLE IF EXISTS tbl1;
Query OK, 0 rows affected (0.00 sec)
```

8.8　获得数据库、表、字段和索引的信息

一旦创建了数据库，我们可以使用MySQL的SHOW DATABASES语句来验证它的存在，这个语句提供了服务器当前知道的所有数据库的列表：

```
mysql> SHOW DATABASES;
+-----------+
| Database  |
+-----------+
| db1       |
| mysql     |
| test      |
+-----------+
3 rows in set (0.04 sec)
```

可以使用SHOW TABLES FROM *database-name* 命令验证数据库中表的存在：

```
mysql> CREATE TABLE tbl1 (fld1 int(1) NOT NULL);
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW TABLES;
+---------------+
| Tables_in_db1 |
+---------------+
| tbl1          |
+---------------+
1 row in set (0.01 sec)
```

可以使用DESCRIBE命令得到关于表结构（字段、字段类型、键和默认值）的信息，如下所示：

```
mysql> DESCRIBE employees;
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| id         | smallint(6)  |      | PRI | 0       |       |
| fname      | varchar(255) |      |     |         |       |
| lname      | varchar(255) |      |     |         |       |
| age        | tinyint(4)   |      |     | 0       |       |
| sex        | char(1)      |      |     |         |       |
| department | varchar(255) |      |     |         |       |
| country    | char(2)      |      |     |         |       |
+------------+--------------+------+-----+---------+-------+
7 rows in set (0.03 sec)
```

可以使用SHOW INDEX FROM *table-name*命令得到表的索引的列表，使用SHOW TABLE STATUS FROM *database-name* 命令得到数据库中表的详细信息（包括表类型、行数、最后一次更新表的日期和时间、索引和行的长度）。因为这些命令输出的信息有些长，可能需要一页来保存这些信息。熟练掌握这些命令可以让我们得到详细的信息。

8.9 小结

本章集中讨论了MySQL的数据结构，解释了如何添加、修改和删除MySQL RDBMS中的MySQL数据库和表。本章既包含了与数据库及表管理相关的SQL命令的理论知识，也使用了很多简单的例子来说明每个命令在现实世界中的用途。下面是讨论过的命令：

- CREATE DATABASE　用来创建一个新的数据库。
- USE　用来选择进行事务处理的数据库。
- DROP DATABASE　用来删除数据库以及和它相关的数据。
- CREATE TABLE（和它的多种变体）　用来定义表的结构、约束和类型。
- DROP TABLE　用来从数据库删除一个表。
- SHOW DATABASES　用来浏览服务器上可获得的数据库。
- SHOW TABLES　用来浏览数据库中的表。
- SHOW CREATE　作为在保存索引和键信息时重建表结构的基础。
- SHOW INDEX　用来浏览表的索引。
- ALTER TABLE　用来更改现有表的结构。

当然，数据定义只占所有问题的三分之一。一旦定义了数据结构，下一步就是输入数据。下一章结合与数据操作相关的MySQL命令的解释和示例，将详细讨论这方面的问题。

第9章 使用数据

一旦定义了数据库和表，下一步就是存储数据。一般情况下，这需要使用SQL的数据操纵语言（DML）来添加、更新、删除和查询数据库中的记录。因为DML命令比前一章讨论的DDL命令使用得频繁，这些命令对于保证MySQL RDBMS的使用效率非常重要。

本章集中详细地介绍DML，包括如何插入新的记录、更新或删除已存在的记录、执行查询找到与特定准则相匹配的记录、从其他数据源导入数据以及以自定义的格式向文件导出记录。

9.1 插入、更新和删除记录

MySQL提供了很多不同的命令，每一个命令都是为特定的目的设计的，用来插入、更新或者改变表的内容。这一部分讨论、解释和演示不同的命令。

9.1.1 插入记录

一旦创建了数据库和其中的表，下一步就是向它输入数据。可以通过INSERT命令来完成，如下所示：

```
INSERT INTO table-name (field-name, field-name, ···)
VALUES (field-value, field-value, ···)
```

为了说明，我们来创建一个简单的表：

```
mysql> CREATE TABLE addressbook (fname VARCHAR(255) NULL,
lname VARCHAR(255) NULL, phone VARCHAR(255) NULL,
fax VARCHAR(255) NULL, email VARCHAR(255) NULL);
Query OK, 0 rows affected (0.08 sec)

mysql> DESCRIBE addressbook;
+-------+-----------+------+-----+---------+-------+
| Field | Type      | Null | Key | Default | Extra |
+-------+-----------+------+-----+---------+-------+
| fname | char(255) | YES  |     | NULL    |       |
| lname | char(255) | YES  |     | NULL    |       |
| phone | char(255) | YES  |     | NULL    |       |
| fax   | char(255) | YES  |     | NULL    |       |
| email | char(255) | YES  |     | NULL    |       |
+-------+-----------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

然后使用INSERT语句向它输入一些数据：

```
mysql> INSERT INTO addressbook (fname, lname, phone, fax, email)
VALUES ('Rob', 'Rabbit', '674 1536', '382 8364', 'rob@some.domain');
Query OK, 1 row affected (0.00 sec)
```

现在这个表包含 Rob Rabbit 一条记录。可以调用 SELECT 命令验证它：

```
mysql> SELECT * FROM addressbook;
+-------+--------+----------+----------+-----------------+
| fname | lname  | phone    | fax      | email           |
+-------+--------+----------+----------+-----------------+
| Rob   | Rabbit | 674 1536 | 382 8364 | rob@some.domain |
+-------+--------+----------+----------+-----------------+
1 row in set (0.01 sec)
```

也可以使用INSERT语句的缩写格式，在这个语句中字段列表是未说明的。下面的例子和前面的例子类似：

```
mysql> INSERT INTO addressbook VALUES ('Rob', 'Rabbit', '674 1536',
'382 8364', 'rob@some.domain');
Query OK, 1 row affected (0.01 sec)
```

请注意，当使用缩写格式时，插入值的顺序必须与表中字段的顺序相一致（可以通过调用 DESCRIBE 命令来决定字段顺序）。

命　名

INSERT语句的缩写变化是非常普遍的，在大多数情况下，最好的方法是为要插入值的字段明确地命名。这就使应用程序允许表的结构变化，同时也消除了模糊性。

也可以插入计算或者函数调用的结果，如下例所示：

```
mysql> INSERT INTO marks (name, math, sci, total)
VALUES ('James Doe', 45, 67, math + sci);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO users (id, name, password)
VALUES (LAST_INSERT_ID() + 1, 'tom', 'secret');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO sales (id, cost, quantity, amount)
VALUES (8587, 100, 4500, cost * quantity);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO log (process, message, timestamp)
VALUES ('crond', 'Unable to send mail in /usr/local/bin/backup.sh', NOW());
Query OK, 1 row affected (0.01 sec)
```

当向表插入字符串值（以及一些数据值）时，必须使用引号，否则 MySQL 认为它们是字段名：

```
mysql> INSERT INTO addressbook (fname, lname, phone, fax, email)
VALUES ('Rob', Rabbit, '674 1536', '382 8364', 'rob@some.domain');
ERROR 1054: Unknown column 'Rabbit' in 'field list'
```

包含引用标志的值需要在前面加上反斜线：

```
mysql> INSERT INTO addressbook (fname, lname, phone, fax, email)
VALUES ('Frances', 'D\'Souza', '123 4567', '000 7574', 'fdz@some.domain');
Query OK, 1 row affected (0.11 sec)
```

不过数值不需要加引号：

```
mysql> INSERT INTO stock (cost, quantity) VALUES (760.52, 65000);
Query OK, 1 row affected (0.01 sec)
```

选择 INTO

INSERT 语句的 INTO 关键字是可选的

MySQL 也支持下面两个 INSERT 语句的非标准变化

- 在一个单独的 INSERT 语句中使用多个 VALUES()子句可以插入多条记录。例如，不进行下面的操作：

```
mysql> INSERT INTO stocks (symbol, price, quantity)
VALUES ('ABCD', 100, 4500);
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO stocks (symbol, price, quantity)
VALUES ('HYDH', 2000, 29);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO stocks (symbol, price, quantity) VALUES ('UGTS', 25, 67);
Query OK, 1 row affected (0.05 sec)
```

我们可以这样做：

```
mysql> INSERT stocks (symbol, price, quantity) VALUES ('ABCD', 100, 4500),
('HYDH', 2000, 29), ('UGTS', 25, 67);
Query OK, 3 rows affected (0.06 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

- 也可以不使用 INSERT…VALUES 格式，而是使用类似的 UPDATE 语句。它使用 SET 子句分别为每一列设置值。所以，可以不进行下面的操作：

```
mysql> INSERT INTO stocks (symbol, price, quantity)
VALUES ('HYDH', 2000, 29);
Query OK, 1 row affected (0.00 sec)
```

而是这样做：

```
mysql> INSERT INTO stocks SET symbol='HYDH', price=2000, quantity=29;
Query OK, 1 row affected (0.06 sec)
```

使用 DEFAULT 值

字段列表中没有命名的字段将会自动分配默认值。考虑下面的例子，它对此进行了说明：

```
mysql> CREATE TABLE forums (name VARCHAR(150) NOT NULL,
category VARCHAR(50) DEFAULT 'UNIX' NOT NULL,
postsPerPage SMALLINT DEFAULT 15 NOT NULL,
```

```
highlightColor VARCHAR(10) DEFAULT 'red' NOT NULL);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO forums (name) VALUES ('Apache');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO forums (name, highlightColor)
VALUES ('Sendmail', 'green');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM forums;
+----------+----------+--------------+----------------+
| name     | category | postsPerPage | highlightColor |
+----------+----------+--------------+----------------+
| Apache   | UNIX     |           15 | red            |
| Sendmail | UNIX     |           15 | green          |
+----------+----------+--------------+----------------+
2 rows in set (0.00 sec)
```

MySQL 4.0.3以及更高版本支持DEFAULT关键字，MySQL使用这个关键字为记录设置列的默认值：

```
mysql> INSERT INTO forums (name, category, postsPerPage,
highlightColor) VALUES ('MySQL', DEFAULT, DEFAULT, DEFAULT);
Query OK, 1 row affected (0.01 sec)
```

使用 AUTOINCREMENT 字段

当使用AUTOINCREMENT列创建序列时，在INSERT语句中不使用字段名将导致MySQL自动产生序列的下一个序号。这个序号作为表的主键。考虑下面的例子，通过创建一个使用AUTOINCREMENT列作为主键的表来说明：

```
mysql> CREATE TABLE users (uid TINYINT(4) NOT NULL AUTO_INCREMENT,
uname varchar(8) NOT NULL DEFAULT '', upass VARCHAR(15) NOT NULL DEFAULT '',
PRIMARY KEY (uid));
Query OK, 0 rows affected (0.08 sec)
```

然后填充数据：

```
mysql> INSERT INTO users (uname, upass) VALUES ('jim', PASSWORD('secret'));
Query OK, 1 row affected (0.14 sec)

mysql> INSERT INTO users (uname, upass) VALUES ('sarah', PASSWORD('opensesame'));
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO users (uname, upass) VALUES ('tim', PASSWORD('whiteboard'));
Query OK, 1 row affected (0.00 sec)
```

前面代码插入的每条记录，MySQL将会在uid字段插入一个惟一的、自动增加的序号：

```
mysql> SELECT * FROM users;
+-----+-------+----------------+
| uid | uname | upass          |
+-----+-------+----------------+
|   1 | jim   | 428567f40899440 |
```

```
|   2 | sarah | 2f3a61de16678ac |
|   3 | tim   | 31a1db03482b555 |
+-----+-------+-----------------+
3 rows in set (0.00 sec)
```

使用 UNIQUE 字段

当在一个UNIQUE键上插入包含重复值的记录时，我们可以控制MySQL如何处理这种情况：使用IGNORE关键字或者ON DUPLICATE KEY UPDATE子句跳过INSERT、中断操作或者更新旧记录为新值。

为了更清楚地理解这一点，考虑下面的表，它在id字段上包含了一个UNIQUE键：

```
+----+----------+---------------+
| id | label    | url           |
+----+----------+---------------+
|  1 | Home     | home.html     |
|  2 | About Us | aboutus.html  |
|  3 | Services | services.html |
|  4 | Feedback | feedback.html |
+----+----------+---------------+
4 rows in set (0.00 sec)
```

通常，如果现在插入一条违背惟一约束的记录，MySQL会中断操作，提示出错：

```
mysql> INSERT INTO menu (id, label, url) VALUES (4, 'Contact Us', 'contactus.html');
ERROR 1062: Duplicate entry '4' for key 1
```

在前面的INSERT语句添加IGNORE关键字时，如果认为语句违背了惟一约束，MySQL甚至不会尝试去执行这条语句。因此，下面的语句不会返回错误：

```
mysql> INSERT IGNORE INTO menu (id, label, url) VALUES (4, 'Contact Us',
'contactus.html');
Query OK, 0 rows affected (0.01 sec)
```

表中的数据保持不变：

```
mysql> SELECT * FROM menu;
+----+----------+---------------+
| id | label    | url           |
+----+----------+---------------+
|  1 | Home     | home.html     |
|  2 | About Us | aboutus.html  |
|  3 | Services | services.html |
|  4 | Feedback | feedback.html |
+----+----------+---------------+
4 rows in set (0.00 sec)
```

REPLACE 命令

ON DUPLICATE KEY UPDATE子句的微小变化就是REPLACE命令，它和INSERT命令采用了相同的句法，但是不像INSERT，如果在标记为UNIQUE的字段上插入包含相同值的记录将会产生错误，REPLACE将会用新值替换整条记录。

ON DUPLICATE KEY UPDATE 和 REPLACE 的区别是：前者只把已命名的字段更新为新值，后者删除了旧记录，然后用新值完全地替换它。

当有很多的INSERT语句需要被顺序地执行时，IGNORE关键字就使操作变得很方便。使用它可以保证不管哪一个INSERT包含了重复的键值，MySQL都会跳过它（而不是放弃全部操作）。

在这种情况下，我们还可以通过添加 MySQL 4.1 新增加的 ON DUPLICATE KEY UPDATE 子句，使 MySQL 自动把 INSERT 操作转换为 UPDATE 操作。这个子句必须具有需要更新的字段列表，这个列表和UPDATE语句使用的列表相同。考虑下面的例子，它对此进行了演示：

```
mysql> INSERT INTO menu (id, label, url) VALUES (4, 'Contact Us', 'contactus.
html') ON DUPLICATE KEY UPDATE label = 'Contact Us', url = 'contactus.html';
```

在这种情况下，如果 MySQL 发现表已经包含具有相同惟一键的记录，它会自动更新旧的记录为 ON DUPLICATE KEY UPDATE 从句中指定的新值。上面的命令将会产生下面的表：

```
mysql> SELECT * FROM menu;
+----+------------+----------------+
| id | label      | url            |
+----+------------+----------------+
|  1 | Home       | aboutus.html   |
|  2 | About Us   | aboutus.html   |
|  3 | Services   | services.html  |
|  4 | Contact Us | contactus.html |
+----+------------+----------------+
4 rows in set (0.01 sec)
```

使用 NULL 值

我们可以在INSERT语句中使用NULL关键字来向一个字段（只要没有标记为NOT NULL）输入NULL值。下面的例子通过向包含 NULL值的email 和 fax 字段的addressbook 表添加记录进行说明：

```
mysql> INSERT INTO addressbook (fname, lname, phone, fax, email)
VALUES ('Polly', 'Parrot', '239 1828', NULL, NULL);
Query OK, 1 row affected (0.00 sec)
```

可以使用 IS NULL 运算符来检索包含 NULL 值的所有记录的列表，如下所示：

```
mysql> SELECT * FROM addressbook WHERE email IS NULL OR fax IS NULL;
+-------+--------+----------+------+---------------+
| fname | lname  | phone    | fax  | email         |
+-------+--------+----------+------+---------------+
| Polly | Parrot | 239 1828 | NULL | NULL          |
| David | Donkey | NULL     | NULL | david@don.key |
+-------+--------+----------+------+---------------+
2 rows in set (0.00 sec)
```

9.1.2 更新记录

数据如同生命一样充满着变数——当发生改变时，会发现数据库面临着更改记录的任务，这些记录是使用INSERT语句插入的。这就是为什么MySQL也提供了UPDATE命令，它是特意为更新表中的一些或者所有记录而设计的。

下面是UPDATE命令的语法：

```
UPDATE table-name SET field-name = value, SET field-name = value, ···
WHERE conditional-test
```

空格问题

一个新手普遍的错误是认为NULL值和空字符串（''）相同。讨论两者的区别看起来好像是无中生有的事情。实际上在SQL世界里，这两者是完全不同的：NULL是指没有值，而''表示值是存在的，只不过是空值。虽然区别可能看起来很小，但是如果不仔细的话，可能会使我们陷入困境。在PRIMARY KEY字段中不能使用NULL值。

假如有下面的地址簿：

```
+--------+---------+----------+----------+-------------------------+
| fname  | lname   | phone    | fax      | email                   |
+--------+---------+----------+----------+-------------------------+
| Polly  | Parrot  | 239 1828 | NULL     | NULL                    |
| David  | Donkey  | NULL     | NULL     | david@don.key           |
| George | Giraffe | 473 1911 | 757 0000 | george@giraffesrule.tld |
| Rico   | Rat     | 193 2839 | 985 2281 | rico@bite.me            |
| Rob    | Rabbit  | 674 1536 | 382 8364 | rob@some.domain         |
+--------+---------+----------+----------+-------------------------+
5 rows in set (0.05 sec)
```

如果想要更新Polly的e-mail地址，可以调用UPDATE来完成：

```
mysql> UPDATE addressbook SET email = 'polly@whos.talking'
WHERE fname = 'Polly' AND lname = 'Parrot';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

和前面一样，可以调用SELECT进行验证：

```
mysql> SELECT * FROM addressbook;
+--------+---------+----------+----------+-------------------------+
| fname  | lname   | phone    | fax      | email                   |
+--------+---------+----------+----------+-------------------------+
| Polly  | Parrot  | 239 1828 | NULL     | polly@whos.talking      |
| David  | Donkey  | NULL     | NULL     | david@don.key           |
| George | Giraffe | 473 1911 | 757 0000 | george@giraffesrule.tld |
| Rico   | Rat     | 193 2839 | 985 2281 | rico@bite.me            |
| Rob    | Rabbit  | 674 1536 | 382 8364 | rob@some.domain         |
+--------+---------+----------+----------+-------------------------+
5 rows in set (0.00 sec)
```

可以通过在UPDATE语句中指定多个SET子句来同时更新一条记录的多个字段。考虑下面的例子，它更新了Rob Rabbit的记录：

```
mysql> UPDATE addressbook SET lname = 'Ringworm', phone = 'RING ME',
fax = NULL, email = 'robbieworm@wormplace.net' WHERE fname = 'Rob';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM addressbook;
+--------+----------+----------+----------+--------------------------+
| fname  | lname    | phone    | fax      | email                    |
+--------+----------+----------+----------+--------------------------+
| Polly  | Parrot   | 239 1828 | NULL     | polly@whos.talking       |
| David  | Donkey   | NULL     | NULL     | david@don.key            |
| George | Giraffe  | 473 1911 | 757 0000 | george@giraffesrule.tld  |
| Rico   | Rat      | 193 2839 | 985 2281 | rico@bite.me             |
| Rob    | Ringworm | RING ME  | NULL     | robbieworm@wormplace.net |
+--------+----------+----------+----------+--------------------------+
5 rows in set (0.00 sec)
```

如果省略UPDATE语句的WHERE子句，表中的每一个单独行都会受到UPDATE语句的影响。下面的例子通过使用没有WHERE子句的UPDATE命令进行了说明：

```
mysql> UPDATE addressbook SET lname = 'Ringworm', phone = 'RING ME',
fax = NULL, email = 'robbieworm@wormplace.net';
Query OK, 6 rows affected (0.00 sec)
Rows matched: 6  Changed: 6  Warnings: 0

mysql> SELECT * FROM addressbook;
+--------+----------+---------+------+--------------------------+
| fname  | lname    | phone   | fax  | email                    |
+--------+----------+---------+------+--------------------------+
| Polly  | Ringworm | RING ME | NULL | robbieworm@wormplace.net |
| David  | Ringworm | RING ME | NULL | robbieworm@wormplace.net |
| George | Ringworm | RING ME | NULL | robbieworm@wormplace.net |
| Rico   | Ringworm | RING ME | NULL | robbieworm@wormplace.net |
| Rob    | Ringworm | RING ME | NULL | robbieworm@wormplace.net |
+--------+----------+---------+------+--------------------------+
5 rows in set (0.00 sec)
```

这并不是我们通常想要的。

对新手的忠告

省略WHERE子句的UPDATE命令是一个新手普遍的错误，如果没有仔细地使用，可能会导致大量数据破坏。

在9.2节可以看到更多关于WHERE子句的信息。

精细地操作

当执行INSERT或者UPDATE操作时，可以通过使用LOW_PRIORITY和DELAYED关键字使MySQL客户执行等待，直到成功地完成操作或者排队等候服务器进行处理。

- LOW_PRIORITY关键字阻止语句的执行直到没有其他线程使用这个表，从而强制客户等待，直到语句完全被执行。在应用中可靠性是最重要的，添加这个关键字可以帮助提高SQL事务处理以及在它上面的业务逻辑的完整性。
- DELAYED关键字和LOW_PRIORITY关键字相反，把要执行的语句放在一个队列中（在服务器的存储器内）就立即让客户退出。当处理繁忙的数据库服务时，这一点在提高应用程序的性能方面尤其重要，因为它可以提高客户响应时间（但是比较冒险——如果服务器崩溃，在服务器队列中的所有请求都会丢失）。

请注意，这两个关键字不适应于InnoDB表。

和INSERT语句一样，可以使用UPDATE语句执行计算或者调用函数，然后利用这些操作的结果进行更新。下面是一些例子：

```
mysql> UPDATE marks SET total = math + sci WHERE name='James';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE users SET password = PASSWORD('secret') WHERE user = 'sally';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE menu SET label = 'Home' WHERE label = 'Main Menu';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

9.1.3 删除记录

正如可以删除、更新和替换数据一样，MySQL也允许我们使用DELETE和TRUNCATE语句删除数据。DELETE语句可以允许从表中删除记录，同时支持WHERE子句来限制只能删除和指定条件匹配的记录。如下所示：

```
DELETE FROM table-name WHERE conditional-test
```

因而，给出下面的表：

```
+-----+-------+-------+
| uid | uname | upass |
+-----+-------+-------+
|  16 | joe   | joe   |
|  17 | john  | john  |
|  18 | joe   | joe   |
|  19 | john  | john  |
+-----+-------+-------+
4 rows in set (0.00 sec)
```

我们可以使用下面的命令删除关于John的记录：

```
mysql> DELETE FROM users WHERE uid = 19;
Query OK, 1 row affected (0.00 sec)
```

使用 SELECT 浏览表来验证记录是否已删除：

```
mysql> SELECT * FROM users;
+-----+-------+-------+
| uid | uname | upass |
+-----+-------+-------+
|  16 | joe   | joe   |
|  17 | john  | john  |
|  18 | joe   | joe   |
+-----+-------+-------+
3 rows in set (0.05 sec)
```

删除和破坏

当使用没有 WHERE 子句的 DELETE 语句时必须小心，因为这将导致表中的所有记录被立即删除。这个操作是不可逆转的（除非使用的是 InnoDB 表且关掉了 AUTOCOMMIT），所以在操作之前必须保证知道自己在做什么。

可以通过省略 WHERE 子句删除表的所有记录：

```
mysql> DELETE FROM users;
Query OK, 3 rows affected (0.00 sec)
```

另外的一个方法是使用 TRUNCATE 命令，它也可以删除表的所有记录。下面的命令等价于前面的命令：

```
mysql> TRUNCATE TABLE users;
Query OK, 0 rows affected (0.05 sec)
```

如果想要删除表的所有记录，TRUNCATE 比 DELETE 要快，因为 TRUNCATE 删除了表，然后重建它，而 DELETE 删除的是记录，并没有尝试去修改表。这也是为什么当向一个使用 DELETE 清空的表插入记录时，MySQL 会记住前面产生的 AUTOINCREMENT 序列，并且继续利用它对 AUTOINCREMENT 字段编号。另一方面，TRUNCATE 后的表一直从 1 开始为 AUTOINCREMENT 字段编号。

请注意，TRUNCATE 命令不像 DELETE 命令，对于事务处理是不安全的。因此，如果我们想要执行 TRUNCATE 的表正在进行事务处理，这个命令就会退出并产生错误信息。

扩 充 表

MySQL 4.0 允许使用 UPDATE 和 DELETE 操作来扩充多个表。如下面的例子：

```
# raise invoice amounts of all customers in Massachusetts by 10%
UPDATE invoices, customers SET invoices.amount = invoices.amount + (invoices.amount
* 0.1) WHERE invoices.customerID = customers.customerID AND customers.location =
'MA';

# update a user's log-in name in all areas of a Web discussion forum
UPDATE profiles, forums, groups SET profiles.username = 'hyde', forums.username =
'hyde', groups.username = 'hyde' WHERE profiles.username = 'jekyll' AND
profiles.username = forums.username AND profiles.username = groups.username;
```

```
# delete a particular customer and all invoices generated for that accountfrom the
system
DELETE FROM customers, invoices USING customers, invoices WHERE
customers.customerID = invoices.customerID AND customers.customerID = 88;

# delete all matching employees from the employee database and departmentrecords
DELETE FROM employees, departments USING employees, departments WHERE
employees.employeeID = departments.employeeID
```

9.2 检索记录

在前面的讨论中可以看到，当向数据库写数据时，MySQL提供了许多不同的命令向数据库写数据。但是当检索数据时，它只提供了一个命令：SELECT语句。

SELECT语句是SQL开发者的军火库中最强有力的语句之一，而且幸运的是，它很容易掌握。SELECT语句可以用来判断表达式，如下所示：

```
mysql> SELECT 50 >= (3+2);
+-------------+
| 50 >= (3+2) |
+-------------+
|           1 |
+-------------+
1 row in set (0.00 sec)

mysql> SELECT 100/10, NOW();
+--------+---------------------+
| 100/10 | NOW()               |
+--------+---------------------+
|  10.00 | 2003-06-27 14:29:19 |
+--------+---------------------+
1 row in set (0.11 sec)
```

从一个或者多个表的环境中检索数据的方法如下：

```
mysql> SELECT * FROM accounts LIMIT 0, 6;
+--------------+-------------+-----------------+-----------------+---------------+
| accountNumber | accountName | accountCreatedOn | accountCreatedAt | accountBalance |
+--------------+-------------+-----------------+-----------------+---------------+
|    1265489921 | James D     | 1998-07-29      | NY              |       2346.00 |
|    2147483647 | Timothy J   | 2000-12-12      | BO              |      56347.50 |
|    5739304575 | Harish K    | 1995-09-13      | NY              |     996564.88 |
|    2173467271 | Kingston X  | 1967-09-15      | MA              |     634238.00 |
|    2312934021 | Sue U       | 1998-06-06      | MA              |         34.67 |
|    1248954638 | Ila T       | 2003-08-17      | CA              |       5373.82 |
|    2384371001 | Anil V      | 2001-11-19      | MA              |      72460.00 |
|    9430125467 | Katrina P   | 2003-07-01      | NY              |        100.00 |
|    1890192554 | Pooja B     | 2002-01-23      | FL              |      17337.12 |
|    2388282010 | Sue U       | 1998-03-27      | BO              |     388883.13 |
|    2374845291 | Jacob N     | 2003-05-01      | NY              |      18410.00 |
+--------------+-------------+-----------------+-----------------+---------------+
11 rows in set (0.05 sec)
```

前面的“抓住所有”查询是SELECT语句最简单的格式。它返回指定表的所有记录。前面SELECT语句的星号（*）是一个通配符，表示“获得所有的字段”。

9.2.1 检索指定的行和列

如果希望结果集是包含表的特定列的集合，可以在SELECT关键字（通配符*的位置）后面用逗号间隔的列表形式指定字段。下面的例子只检索了前面accounts表中的accountName和accountBalance列：

```
mysql> SELECT accountName, accountBalance FROM accounts;
+-------------+----------------+
| accountName | accountBalance |
+-------------+----------------+
| James D     |        2346.00 |
| Timothy J   |       56347.50 |
| Harish K    |      996564.88 |
| Kingston X  |      634238.00 |
| Sue U       |          34.67 |
| Ila T       |        5373.82 |
| Anil V      |       72460.00 |
| Katrina P   |         100.00 |
| Pooja B     |       17337.12 |
| Sue U       |      388883.13 |
| Jacob N     |       18410.00 |
+-------------+----------------+
11 rows in set (0.05 sec)
```

可以向SELECT语句添加一个WHERE子句，让查询返回和指定条件匹配的记录。WHERE子句必须跟着一个条件表达式，这个条件将从结果集中过滤掉不匹配的记录。

下面的例子通过检索那些账户余额小于1000的记录列表进行了演示：

```
mysql> SELECT accountName, accountBalance FROM accounts
WHERE accountBalance < 1000;
+-------------+----------------+
| accountName | accountBalance |
+-------------+----------------+
| Sue U       |          34.67 |
| Katrina P   |         100.00 |
+-------------+----------------+
2 rows in set (0.11 sec)
```

下面的命令检索了2002年创建的所有账户列表：

```
mysql> SELECT accountName, accountCreatedOn FROM accounts WHERE
YEAR(accountCreatedOn) = 2002;
+-------------+------------------+
| accountName | accountCreatedOn |
+-------------+------------------+
| Pooja B     | 2002-01-23       |
+-------------+------------------+
1 row in set (0.00 sec)
```

还可以通过向语句中添加MySQL的逻辑运算符——AND，OR和NOT创建更复杂的查询。下面的例子通过查询余额从5000到50 000的账户进行了说明：

```
mysql> SELECT accountName, accountNumber, accountBalance FROM accounts
WHERE accountBalance > 5000.00 AND accountBalance < 50000.00;
+-------------+---------------+----------------+
| accountName | accountNumber | accountBalance |
+-------------+---------------+----------------+
| Ila T       |    1248954638 |        5373.82 |
| Pooja B     |    1890192554 |       17337.12 |
| Jacob N     |    2374845291 |       18410.00 |
+-------------+---------------+----------------+
3 rows in set (0.05 sec)
```

下面例子检索了在纽约或波士顿开户的所有账户：

```
mysql> SELECT accountName, accountCreatedAt FROM accounts WHERE accountCreatedAt
= 'NY' OR accountCreatedAt = 'BO';
+-------------+------------------+
| accountName | accountCreatedAt |
+-------------+------------------+
| James D     | NY               |
| Timothy J   | BO               |
| Harish K    | NY               |
| Katrina P   | NY               |
| Sue U       | BO               |
| Jacob N     | NY               |
+-------------+------------------+
6 rows in set (0.00 sec)
```

9.2.2 使用内建函数

也可以在SELECT语句中使用MySQL的内建函数。下面的例子选择账户余额最小和最大的记录：

```
mysql> SELECT MAX(accountBalance), MIN(accountBalance) FROM accounts;
+--------------------+--------------------+
| MAX(accountBalance) | MIN(accountBalance) |
+--------------------+--------------------+
|          996564.88 |              34.67 |
+--------------------+--------------------+
1 row in set (0.00 sec)
```

下面例子算出了表中记录的总数并返回结果：

```
mysql> SELECT COUNT(*) FROM accounts;
+----------+
| count(*) |
+----------+
|       11 |
+----------+
1 row in set (0.00 sec)
```

WHERE 子句也可以使用内建函数。下面的查询产生了当年创建的所有账户列表：

```
mysql> SELECT accountName, accountCreatedOn FROM accounts WHERE
YEAR(accountCreatedOn) = YEAR(NOW());
+-------------+-----------------+
| accountName | accountCreatedOn |
+-------------+-----------------+
| Ila T       | 2003-08-17      |
| Katrina P   | 2003-07-01      |
| Jacob N     | 2003-05-01      |
+-------------+-----------------+
3 rows in set (0.06 sec)
```

下面的命令返回三年前或者更早创建的账户列表：

```
mysql> SELECT accountName, accountCreatedOn FROM accounts WHERE
YEAR( DATE_ADD(accountCreatedOn, INTERVAL 3 YEAR) ) <= YEAR( NOW() );
+-------------+-----------------+
| accountName | accountCreatedOn |
+-------------+-----------------+
| James D     | 1998-07-29      |
| Timothy J   | 2000-12-12      |
| Harish K    | 1995-09-13      |
| Kingston X  | 1967-09-15      |
| Sue U       | 1998-06-06      |
| Sue U       | 1998-03-27      |
+-------------+-----------------+
6 rows in set (0.05 sec)
```

9.2.3 为表和列取别名

可以在准备定义别名的表或者字段名字后面添加AS关键词和别名来为表和字段创建别名。当表或者字段的名字比较长时，别名就很有用。下面的例子通过把accountName和accountNumber列分别另取名为name和num来进行阐述：

```
mysql> SELECT accountNumber AS num, accountName AS name FROM accounts;
+------------+------------+
| num        | name       |
+------------+------------+
| 1265489921 | James D    |
| 2147483647 | Timothy J  |
| 5739304575 | Harish K   |
| 2173467271 | Kingston X |
| 2312934021 | Sue U      |
| 1248954638 | Ila T      |
| 2384371001 | Anil V     |
| 9430125467 | Katrina P  |
| 1890192554 | Pooja B    |
| 2388282010 | Sue U      |
| 2374845291 | Jacob N    |
+------------+------------+
11 rows in set (0.00 sec)
```

也可以对表名进行这样的操作：

```
mysql> SELECT accountName, accountCreatedAt FROM accounts AS a
WHERE a.accountName LIKE '%Poo%';
+-------------+-----------------+
| accountName | accountCreatedAt |
+-------------+-----------------+
| Pooja B     | FL              |
+-------------+-----------------+
1 row in set (0.00 sec)
```

9.2.4 限制查询结果

LIMIT关键字可以用来限制SELECT查询返回的记录的总数。下面的查询通过把结果集限制为5条记录来进行说明：

```
mysql> SELECT accountName, accountNumber FROM accounts LIMIT 5;
+-------------+---------------+
| accountName | accountNumber |
+-------------+---------------+
| James D     |    1265489921 |
| Timothy J   |    2147483647 |
| Harish K    |    5739304575 |
| Kingston X  |    2173467271 |
| Sue U       |    2312934021 |
+-------------+---------------+
5 rows in set (0.06 sec)
```

我们可以（可选地）指定从何处开始计算行的偏移量。下面的例子从第4行（注意是从第0行开始计数）开始返回6条记录：

```
mysql> SELECT accountName, accountNumber FROM accounts LIMIT 3,6;
+-------------+---------------+
| accountName | accountNumber |
+-------------+---------------+
| Kingston X  |    2173467271 |
| Sue U       |    2312934021 |
| Ila T       |    1248954638 |
| Anil V      |    2384371001 |
| Katrina P   |    9430125467 |
| Pooja B     |    1890192554 |
+-------------+---------------+
6 rows in set (0.00 sec)
```

9.2.5 对查询结果排序

通过向SELECT语句添加ORDER BY子句，根据特定的字段为查询结果排序。下面的例子根据创建日期对accounts表排序：

```
mysql> SELECT accountCreatedOn, accountNumber FROM accounts
ORDER BY accountCreatedOn;
```

```
+------------------+--------------+
| accountCreatedOn | accountNumber |
+------------------+--------------+
| 1967-09-15       |    2173467271 |
| 1995-09-13       |    5739304575 |
| 1998-03-27       |    2388282010 |
| 1998-06-06       |    2312934021 |
| 1998-07-29       |    1265489921 |
| 2000-12-12       |    2147483647 |
| 2001-11-19       |    2384371001 |
| 2002-01-23       |    1890192554 |
| 2003-05-01       |    2374845291 |
| 2003-07-01       |    9430125467 |
| 2003-08-17       |    1248954638 |
+------------------+--------------+
11 rows in set (0.05 sec)
```

获得所有记录

要想检索表从指定的地方到结尾的所有记录，只需指定 -1 作为返回的行数。例如，要返回从第 19 行到表尾部的记录，可以使用查询：

```
SELECT * from tbl LIMIT 18,-1
```

可以根据 ORDER BY 子句指定的用逗号隔开的多个字段进行排序。下面的查询先根据账户余额，然后又根据位置进行排序：

```
mysql> SELECT accountName, accountBalance, accountCreatedAt FROM accounts
ORDER BY accountBalance, accountCreatedAt;
+-------------+----------------+------------------+
| accountName | accountBalance | accountCreatedAt |
+-------------+----------------+------------------+
| Sue U       |          34.67 | MA               |
| Katrina P   |         100.00 | NY               |
| James D     |        2346.00 | NY               |
| Ila T       |        5373.82 | CA               |
| Pooja B     |       17337.12 | FL               |
| Jacob N     |       18410.00 | NY               |
| Timothy J   |       56347.50 | BO               |
| Anil V      |       72460.00 | MA               |
| Sue U       |      388883.13 | BO               |
| Kingston X  |      634238.00 | MA               |
| Harish K    |      996564.88 | NY               |
+-------------+----------------+------------------+
11 rows in set (0.05 sec)
```

可以在 ORDER BY 子句向每个字段添加 ASC 和 DESC 关键字来定制排序方法。相应字段的值以升序或者降序的顺序被排序。下面的例子以颠倒的字母顺序根据账户名进行排序：

```
mysql> SELECT accountName, accountBalance FROM accounts
ORDER BY accountName DESC;
```

```
+-------------+----------------+
| accountName | accountBalance |
+-------------+----------------+
| Timothy J   |       56347.50 |
| Sue U       |          34.67 |
| Sue U       |      388883.13 |
| Pooja B     |       17337.12 |
| Kingston X  |      634238.00 |
| Katrina P   |         100.00 |
| James D     |        2346.00 |
| Jacob N     |       18410.00 |
| Ila T       |        5373.82 |
| Harish K    |      996564.88 |
| Anil V      |       72460.00 |
+-------------+----------------+
11 rows in set (0.06 sec)
```

9.2.6 对查询结果分组

MySQL可以根据特定属性使用GROUP BY子句把结果集中的记录分组。因为以这种方式创建的组用一个行来表示（即使它包含多条记录），这个能力主要用于涉及MySQL的枚举聚合函数操作中。

为了更清楚地了解这一点，考虑下面的例子：

```
mysql> SELECT accountCreatedAt FROM accounts GROUP BY accountCreatedAt;
+------------------+
| accountCreatedAt |
+------------------+
| BO               |
| CA               |
| FL               |
| MA               |
| NY               |
+------------------+
5 rows in set (0.00 sec)
```

在这个例子中，accounts表的记录根据accountCreatedAt字段进行分组。因为表中的11条记录有5个不同的地点，所以MySQL创建5个不同的组，而且结果集中每个组是一个单独的行。

随机选择

使用RAND()函数把LIMIT关键字和ORDER BY子句结合起来解决了对MySQL邮件列表的一个或者多个普通查询——从一个表中随机选择一个或者多条记录。这个过程首先涉及到对表的随机排序，然后从结果集合中选择一个单独的记录。用SQL语言可表示为：

```
SELECT fld1, fld2 FROM tbl ORDER BY RAND() LIMIT 0, 1;
```

聚合函数可以应用于由记录组成的每个单独组。例如，下面的查询返回每个地点的账户总数：

```
mysql> SELECT accountCreatedAt AS loc, COUNT(*) FROM accounts
GROUP BY accountCreatedAt;
```

```
+-----+----------+
| loc | COUNT(*) |
+-----+----------+
| BO  |        2 |
| CA  |        1 |
| FL  |        1 |
| MA  |        3 |
| NY  |        4 |
+-----+----------+
5 rows in set (0.06 sec)
```

下面的例子返回每个地点账户的平均余额：

```
mysql> SELECT accountCreatedAt AS loc, AVG(accountBalance) FROM accounts
GROUP BY accountCreatedAt;
+-----+---------------------+
| loc | AVG(accountBalance) |
+-----+---------------------+
| BO  |       222615.312500 |
| CA  |         5373.819824 |
| FL  |        17337.119141 |
| MA  |       235577.556666 |
| NY  |       254355.218750 |
+-----+---------------------+
5 rows in set (0.05 sec)
```

可以向GROUP BY列表中添加ASC和DESC关键字以升序或者降序为结果集排序。下面的例子是前面例子的变化，它通过以颠倒的字母顺序来为地点排序：

```
mysql> SELECT accountCreatedAt AS loc, AVG(accountBalance) FROM accounts
GROUP BY accountCreatedAt DESC;
+-----+---------------------+
| loc | AVG(accountBalance) |
+-----+---------------------+
| NY  |       254355.218750 |
| MA  |       235577.556666 |
| FL  |        17337.119141 |
| CA  |         5373.819824 |
| BO  |       222615.312500 |
+-----+---------------------+
5 rows in set (0.05 sec)
```

由GROUP BY子句创建的组可以加上HAVING子句进行更进一步的约束，HAVING子句和前面讨论的WHERE子句具有相似的作用：创建一个和指定条件相匹配的结果集。考虑下面的例子，它对此进行了演示：

```
mysql> SELECT accountCreatedAt AS loc, AVG(accountBalance) FROM accounts
GROUP BY accountCreatedAt HAVING loc = 'CA';
+-----+---------------------+
| loc | AVG(accountBalance) |
+-----+---------------------+
| CA  |         5373.819824 |
```

```
+-----+----------------------+
1 row in set (0.05 sec)

mysql> SELECT accountCreatedAt, SUM(accountBalance) FROM accounts
GROUP BY accountCreatedAt HAVING SUM(accountBalance) > 100000.00;
+------------------+----------------------+
| accountCreatedAt | SUM(accountBalance) |
+------------------+----------------------+
| BO               |           445230.63 |
| MA               |           706732.67 |
| NY               |          1017420.88 |
+------------------+----------------------+
3 rows in set (0.17 sec)
```

9.2.7 使用变量

MySQL也可以让我们以用户定义的变量来存储SELECT查询的结果，以便在将来的SELECT查询中使用。它们只在客户会话期间存在，但是它们提供一个方便有效的方法来连接查询，然后在另一个查询的子句中使用一个查询的输出。

选择自己想要的

WHERE和HAVING的区别是WHERE子句对一个表的所有记录进行操作，只搜索与指定条件相匹配的记录。而HAVING子句只对经过操作的表记录进行检索，对结果集进行更进一步的筛选。通常，HAVING子句和GROUP BY子句相连，而WHERE子句和SELECT，DELETE和UPDATE语句相连。

变量赋值由:=操作符完成，它用来向一个变量分配一个值。下面的例子是把表中的最早日期赋给@oldest变量：

```
mysql> SELECT @oldest:= MIN(accountCreatedOn) FROM accounts;
+-------------------------------+
| @oldest:=min(accountCreatedOn) |
+-------------------------------+
| 1967-09-15                    |
+-------------------------------+
1 row in set (0.00 sec)
```

存储在@oldest变量中的值可以通过简单的SELECT进行检索：

```
mysql> SELECT @oldest;
+------------+
| @oldest    |
+------------+
| 1967-09-15 |
+------------+
1 row in set (0.00 sec)
```

或者使用另一个查询：

```
mysql> SELECT accountName, accountNumber, accountBalance FROM accounts WHERE
accountCreatedOn = @oldest;
```

```
+-------------+---------------+----------------+
| accountName | accountNumber | accountBalance |
+-------------+---------------+----------------+
| Kingston X  |    2173467271 |   634238.00000 |
+-------------+---------------+----------------+
1 row in set (0.00 sec)
```

变量名是大小写敏感的，它可以包含字母、数字字符、下划线和点。这些变量是类型不敏感的，一个变量可以在不需要事先进行类型声明的基础上存储字符串或者数字。

9.2.8 使用子查询

MySQL 4.1允许我们对查询进行嵌套，然后在外查询中使用内查询产生的结果集。这样就不是执行两个（或者更多）独立的查询，而是执行包含一个（或者多个）子查询的一个单独查询。

考虑下面的例子，它使用一个子查询列出所有余额高于平均余额的账户：

```
mysql> SELECT accountName, accountNumber, accountBalance FROM accounts WHERE
accountBalance > (SELECT AVG(accountBalance) FROM accounts);
+-------------+---------------+----------------+
| accountName | accountNumber | accountBalance |
+-------------+---------------+----------------+
| Harish K    |    5739304575 |      996564.88 |
| Kingston X  |    2173467271 |      634238.00 |
| Sue U       |    2388282010 |      388883.13 |
+-------------+---------------+----------------+
3 rows in set (0.16 sec)
```

这里，MySQL 以最内层的查询开始，从表的记录中首先获得平均账户余额，然后从最内层的查询向外进行外（主要）查询，用外部查询的 WHERE 子句中的内查询产生的数据进行替换来获得比平均值高的账户余额持有者的列表。

也可以在本书的第 11 章看到更多关于子查询的内容。

9.2.9 控制 SELECT 行为

可以向 SELECT 语句添加很多关键字来修改行为：

- DISTINCT 关键字删除包含结果集中具有重复值的记录。
- SQL_CALC_FOUND_ROWS关键字告诉MySQL计算符合查询（不需要考虑可能设置的任何 LIMIT）的总行数。通过调用 FOUND_ROWS()函数可以得到总行数。
- SQL_CACHE 和 SQL_NO_CACHE 关键字告诉 MySQL 查询结果是否需要高速缓存。
- SQL_BUFFER_RESULT关键字强制 MySQL 把查询结果存储到一个临时表。这使缓冲器消除了对查询所使用的表的锁定，而且结果被传送给客户，因而可以暂时被其他过程使用。
- SQL_BIG_RESULT和SQL_SMALL_RESULT关键字可以指定结果集的期望大小，因此可帮助找到最佳的方法对返回的记录进行排序和存储（基于磁盘或者内存中的临时表）。
- SQL_HIGH_PRIORITY关键字提升了与UPDATE，INSERT和DELETE语句相竞争的查询的优先级，因而可以在繁忙的数据库服务器上快速地执行查询。

在同一页面上显示

应用程序中的SQL_CALC_FOUND_ROWS关键字在将数据库内容传送到多个页面上时尤其有用，这是因为它可以用来显示所有记录的总数，甚至当前的结果只是总数中的一个小的子集。

例如，如果我们想要一次显示accounts表的5条记录，而且还想要显示记录的总数，查询如下：

```
mysql> SELECT SQL_CALC_FOUND_ROWS accountName FROM accounts LIMIT 5;
+--------------+
| accountName |
+--------------+
| James D      |
| Timothy J    |
| Harish K     |
| Kingston X   |
| Sue U        |
+--------------+
5 rows in set (0.00 sec)

mysql> SELECT FOUND_ROWS();
+---------------+
| FOUND_ROWS() |
+---------------+
| 11            |
+---------------+
1 row in set (0.00 sec)
```

这种方法返回了5条记录的子集，但是它还是通过FOUND_ROWS()函数来计算与查询相匹配的记录总数，同时这个数据是可以访问的。

加快速度

适当地使用SQL_CACHE，SQL_BUFFER_RESULT，SQL_BIG_RESULT，SQL_SMALL_RESULT和SQL_HIGH_PRIORITY关键字，可以很明显地提高MySQL服务器处理事务的速度。

9.3 复制、导入和导出记录

除了允许通过INSERT语句手工输入记录之外，MySQL提供了很多DML结构来向表自动添加记录，或者从另一个表复制或者从一个文本文件导入。也可以使用SELECT语句的一种变体把记录导入到一个文本文件，这个特性使备份或者移动数据的操作变得容易。下面的部分详细地介绍每一种方法。

9.3.1 复制记录

MySQL允许结合INSERT和SELECT语句，动态地用一个SELECT查询的结果为一个表添加数据或者从一个表复制记录到另一个表。例如，假设有一个雇员记录的表：

```
mysql > SELECT * FROM employees;
+-----+--------+--------+-----+-----+---------------+---------+
| id  | lname  | fname  | age | sex | department    | country |
```

```
+-----+--------+--------+-----+-----+----------------+---------+
|  54 | Doe    | John   |  27 | M   | Engineering    | US      |
| 127 | Jones  | Sue    |  31 | F   | Finance        | UK      |
| 113 | Woo    | David  |  26 | M   | Administration | CN      |
| 175 | Thomas | James  |  34 | M   | Finance        | US      |
| 168 | Kent   | Jane   |  29 | F   | Administration | US      |
|  12 | Kamath | Ravina |  35 | F   | Finance        | IN      |
+-----+--------+--------+-----+-----+----------------+---------+
6 rows in set (0.01 sec)
```

如果想要在一个单独的用户表中为每一个雇员创建用户记录，可以把INSERT语句和SELECT语句结合起来自动创建用户记录：

```
mysql> INSERT INTO users (uname, upass) SELECT LOWER(fname),
PASSWORD( LOWER(fname) ) FROM employees;
Query OK, 6 rows affected (0.07 sec)
Records: 6  Duplicates: 0  Warnings: 6

mysql> SELECT * FROM users;
+-----+--------+------------------+
| uid | uname  | upass            |
+-----+--------+------------------+
|   7 | john   | 2ca0ede551581d2  |
|   8 | sue    | 789d886f262f349  |
|   9 | david  | 24c0ad4a2a3d818  |
|  10 | james  | 6bd9b3d678820f6  |
|  11 | jane   | 35df0a9c5a75db4  |
|  12 | ravina | 631233cb70584bc  |
+-----+--------+------------------+
6 rows in set (0.00 sec)
```

在INSERT语句中指定的字段列表必须与SELECT子句返回的列相符。不匹配将会导致MySQL产生如下的错误：

```
mysql> INSERT INTO tbl1 (fld1, fld2) SELECT fld1, fld2, fld3 FROM tbl2;
ERROR 1136: Column count doesn't match value count at row 1
```

自然，我们也可以在SELECT语句中连上一个WHERE子句把原始表记录的一个子集复制到新的表：

```
mysql> INSERT INTO users (uname, upass) SELECT LOWER(fname),
PASSWORD( LOWER(fname) ) FROM employees WHERE country = 'US';
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 3

mysql> SELECT * FROM users;
+-----+-------+------------------+
| uid | uname | upass            |
+-----+-------+------------------+
|  10 | john  | 2ca0ede551581d2  |
|  11 | james | 6bd9b3d678820f6  |
```

```
|  12 | jane   | 35df0a9c5a75db4 |
+-----+-------+-----------------+
3 rows in set (0.00 sec)
```

解决问题

MySQL带有IGNORE关键字的INSERT…SELECT的句法支持临时表，它提供很多机会重写SELECT查询，以使它们执行得更快。

例如，我们有一个复杂的查询，它从一个指定的字段选择一组不同的值，而且MySQL引擎因为它的复杂性不能优化查询。富有创造力的SQL编程者首先通过把复杂的查询分成许多简单的查询，使优化器处理得更好，以提高性能。在第一次创建的临时表中的适当字段具有UNIQUE键，然后使用INSERT_IGNORE…SELECT句法向一个临时表插入产生的结果，结果是找到了该字段的一组不同的值，而且加快了执行查询的速度。

9.3.2 导入数据

INSERT语句不是向表中插入数据的惟一方法，MySQL还允许使用LOAD DATA INFILE命令一次性插入多条记录。这个命令可以从文本文件读出原始数据（该文件可以放置在连接的服务器上或者客户端），在列和行界定符号的基础上进行分析，然后自动产生INSERT语句来向一个表写数据。

当需要向一个数据库输入大量的信息，但是结构化的数据与SQL语句的格式不相符时，这种方法就变得非常方便。为每条记录手工创建INSERT语句是非常冗长和耗时的，LOAD DATA INFILE提供了更快的和更可靠的选择。

理解LOAD DATA INFILE的最好方法是利用例子。考虑下面空的birthdays表：

```
mysql> DESCRIBE birthdays;
+-------+-------------+------+-----+------------+-------+
| Field | Type        | Null | Key | Default    | Extra |
+-------+-------------+------+-----+------------+-------+
| name  | varchar(50) |      |     |            |       |
| dob   | date        |      |     | 0000-00-00 |       |
+-------+-------------+------+-----+------------+-------+
2 rows in set (0.05 sec)
```

包含姓名和出生日期的文本文件位于服务器的/home/me/目录中，它的内容是用逗号隔开的：

```
Alan P, 1967-09-11
John D, 1956-11-17
Emma Y, 1963-01-23
Neel K, 1977-10-11
Bryan A, 1959-07-16
```

现在，这些用逗号隔开的数据可以用下面的命令导入到前面的birthdays表：

```
mysql> LOAD DATA INFILE '/home/me/data.txt' INTO TABLE birthdays
FIELDS TERMINATED BY ',' LINES TERMINATED BY '\r\n';
Query OK, 5 rows affected (0.06 sec)
Records: 5  Deleted: 0  Skipped: 0  Warnings: 0
```

现在，数据已经正确地插入到表中，如同SELECT * 所验证的那样：

```
mysql> SELECT * FROM birthdays;
+---------+------------+
| name    | dob        |
+---------+------------+
| Alan P  | 1967-09-11 |
| John D  | 1956-11-17 |
| Emma Y  | 1963-01-23 |
| Neel K  | 1977-10-11 |
| Bryan A | 1959-07-16 |
+---------+------------+
5 rows in set (0.00 sec)
```

默认情况下，MySQL认为数据文件是在服务器上，在LOAD DATA INFILE语句指定的位置。如果想要使用客户机上的数据，可以在语句中添加LOCAL关键字，告诉MySQL在客户机的文件系统上查找文件。下面的例子通过从客户机的文件上装载数据对此进行了演示：

```
mysql> LOAD DATA LOCAL INFILE 'c:\data.txt' INTO TABLE birthdays
FIELDS TERMINATED BY ',' LINES TERMINATED BY '\r\n';
Query OK, 5 rows affected (0.06 sec)
Records: 5  Deleted: 0  Skipped: 0  Warnings: 0
```

如果数据文件的字段比表的字段少，或者如果文件的值和表中的字段序列具有不相同的顺序，可以通过在LOAD DATA INFILE命令后指定字段名来告诉MySQL如何把文件中的数据映射到表的字段。下面的addressbook表包含5列：

```
mysql> DESCRIBE addressbook;
+-------+-----------+------+-----+---------+-------+
| Field | Type      | Null | Key | Default | Extra |
+-------+-----------+------+-----+---------+-------+
| fname | char(255) | YES  |     | NULL    |       |
| lname | char(255) | YES  |     | NULL    |       |
| phone | char(255) | YES  |     | NULL    |       |
| fax   | char(255) | YES  |     | NULL    |       |
| email | char(255) | YES  |     | NULL    |       |
+-------+-----------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

有一个单独文件，包含关于姓名和e-mail地址的一个三字段列表：

```
"Cathy" "cathy@cat.nip" "Cat"
"Polly" "polly@talk.to.me" "Parrot"
"Harry" "harry@prickles.tld" "Hedgehog"
"David" "woof@doghouse" "Dog"
```

可以使用下面的命令让MySQL只把姓名和e-mail地址导入到表：

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE addressbook
FIELDS TERMINATED BY ' ' ENCLOSED BY '"' LINES TERMINATED BY '\r\n'
(fname, email, lname);
Query OK, 4 rows affected (0.05 sec)
Records: 4  Deleted: 0  Skipped: 0  Warnings: 0
```

结果如下：

```
mysql> SELECT * FROM addressbook;
+-------+----------+-------+------+--------------------+
| fname | lname    | phone | fax  | email              |
+-------+----------+-------+------+--------------------+
| Cathy | Cat      | NULL  | NULL | cathy@cat.nip      |
| Polly | Parrot   | NULL  | NULL | polly@talk.to.me   |
| Harry | Hedgehog | NULL  | NULL | harry@prickles.tld |
| David | Dog      | NULL  | NULL | woof@doghouse      |
+-------+----------+-------+------+--------------------+
4 rows in set (0.00 sec)
```

当缺少字段值时，MySQL 插入 NULL 值（如果表和字段约束允许的话）。

正确的路径

当使用服务器上的数据文件时，如果调用 LOAD DATA INFILE 时没有指定文件路径（或者只指定了一个相关路径），MySQL 会在服务器上查找文件的相应数据库目录（或者把它作为相关路径）。不过，绝对路径仍然照样使用。

一些关键字可以用来修改 LOAD DATA INFILE 命令的行为：

- LOW_PRIORITY 关键字导致在开始进行导入工作之前，服务器等待直到没有其他的线程使用该表。另一方面，在进行导入工作时，CONCURRENT 关键字允许客户在导入的过程中从表读数据（但是这个关键字只能应用于 MyISAM 表）。如：

```
mysql> LOAD DATA LOW_PRIORITY INFILE 'log.asc' INTO TABLE logdata;
Query OK, 76 rows affected (0.10 sec)

mysql> LOAD DATA CONCURRENT LOCAL INFILE 'd:\palm\addressbook.txt' INTO
TABLE addressbook;
Query OK, 106 rows affected (1.45 sec)
```

- IGNORE 关键字保证如果新记录的一个键与已存在的记录重复，则 MySQL 将会跳到下一条（而不是退出整个操作，这是发生这种情况的默认行为）。或者可以选择用数据文件的新记录代替已存在的记录。这可以通过使用关键字 REPLACE 代替 IGNORE 完成。如：

```
mysql> LOAD DATA INFILE '/tmp/users.txt' INTO TABLE users
LINES TERMINATED BY '\r\n' (uname);
ERROR 1062: Duplicate entry 'emma' for key 1

mysql> LOAD DATA INFILE '/tmp/users.txt' IGNORE INTO TABLE users
LINES TERMINATED BY '\r\n' (uname);
Query OK, 5 rows affected (0.00 sec)
Records: 6  Deleted: 0  Skipped: 1  Warnings: 0

mysql> LOAD DATA INFILE '/tmp/users.txt' REPLACE INTO TABLE users
LINES TERMINATED BY '\r\n' (uname);
Query OK, 7 rows affected (0.06 sec)
Records: 6  Deleted: 1  Skipped: 0  Warnings: 0
```

9.3.3 导出记录

正如可以使用LOAD DATA INFILE命令从一个文件向一个表导入数据一样，我们也可以使用SELECT…INTO OUTFILE结构从一个表导出记录到一个文件。这个结构可以完成常规SELECT语句做的所有事情，只不过是把结果记录集送到一个文件而已。

分 隔 符

LOAD DATA INFILE和SELECT…INTO OUTFILE命令支持使用附加关键字和指定数据文件格式的子句：

- LINES TERMINATED BY子句指定了记录结束的符号（默认情况下，\n表示新的一行）。
- FIELDS子句指定了字段分隔符，而且FIELDS子句后面必须跟随一个或者多个关键字TERMINATED BY，ESCAPED BY，ENCLOSED BY。TERMINATED BY指定了字段结束的符号（默认是\t），当读出或者写入值时，ESCAPED BY用来跳过特殊的字符（默认是反斜线），ENCLOSED BY指定包围字段值的符号（没有默认值）。

为了进行说明，考虑下面的表：

```
+-------------+----------------+
| accountName | accountBalance |
+-------------+----------------+
| James D     |        2346.00 |
| Timothy J   |       56347.50 |
| Harish K    |      996564.88 |
| Kingston X  |      634238.00 |
| Sue U       |          34.67 |
| Ila T       |        5373.82 |
| Anil V      |       72460.00 |
| Katrina P   |         100.00 |
| Pooja B     |       17337.12 |
| Sue U       |      388883.13 |
| Jacob N     |       18410.00 |
+-------------+----------------+
11 rows in set (0.00 sec)
```

下面的命令把所有的记录取出放到服务器上名为accounts.txt的文本文件内：

```
mysql> SELECT accountName, accountBalance FROM accounts
INTO OUTFILE 'accounts.txt';
Query OK, 11 rows affected (0.06 sec)
```

结果如下：

```
James D       2346.00
Timothy J     56347.50
Harish K      996564.88
Kingston X    634238.00
Sue U         34.67
Ila T         5373.82
Anil V        72460.00
```

```
Katrina P      100.00
Pooja B        17337.12
Sue U          388883.13
Jacob N        18410.00
```

显然可以使用WHERE子句（常规SELECT语句的其他子句或关键字也可以使用）来对输出进行进一步的约束。下面的例子通过只把所有余额高于5000的账户记录写到文件/tmp/high-value-accounts.txt进行演示：

```
mysql> SELECT accountNumber, accountName, accountBalance FROM accounts
WHERE accountBalance > 5000 INTO OUTFILE
'/tmp/high-value-accounts.txt'
FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\r\n';
Query OK, 5 rows affected (0.00 sec)
```

结果如下：

```
"2147483647","Timothy J","56347.50"
"5739304575","Harish K","996564.88"
"2173467271","Kingston X","634238.00"
"2384371001","Anil V","72460.00"
"2388282010","Sue U","388883.13"
```

转　储

要从一个数据库向一个文件写入二进制数据，比如BLOB字段的内容，可以用INTO DUMPFILE子句代替INTO OUTFILE子句。这使MySQL以一个单独行的格式向文件写入数据（没有字段或者记录结束符），因此避免二进制数据的破坏。

在INTO OUTFILE和INTO DUMPFILE子句中指定的文件将会写入服务器的文件系统，而且文件系统中不能已经存在该文件。因为在MySQL服务器运行过程中，用户将要向此文件进行写操作，用户必须具有向指定位置写入文件的合适的权限。为了安全，MySQL不允许以这种方式向客户文件系统写入目标文件。因而，客户应用程序需要使用外部方法从服务器进行检索。

等级和文件

登录客户必须具有FILE权限才能在服务器上使用SELECT…INTO OUTFILE或者LOAD DATA INFILE命令。第14章包含更多关于MySQL权限系统如何工作的信息。

如同LOAD DATA INFILE命令，我们可以为正在转储的数据指定字段和记录的分隔符（可以从上一页的补充资料中获得更多的信息）。下面的例子进行了演示：

```
mysql> SELECT name, phone, email, fax FROM members INTO OUTFILE '/usr/local/
data/members.txt' FIELDS TERMINATED BY '|'
ENCLOSED BY '"' ESCAPED BY '\\';
Query OK, 32 rows affected (0.00 sec)

mysql> SELECT title, author, isbn FROM books INTO OUTFILE '/home/me/books.
asc' FIELDS TERMINATED BY ','
LINES TERMINATED BY '\r\n';
Query OK, 255 rows affected (0.17 sec)
```

还可以使用 mysqldump 实用工具从一个数据库或者表取出内容到一个文件。第 15 章有更多关于如何使用这个实用程序来备份和恢复 MySQL 数据库的信息。

9.4 小结

本章深入探讨了 SQL 提供的数据操作命令（以及一些非标准的、MySQL 特有的命令），解释了如何使用它们在 MySQL 表中存储和更新数据。本章解释了以下命令：

- INSERT　用来向一个表添加新记录。
- UPDATE　用来更新已存在的记录。
- REPLACE　用来替换已存在的记录。
- DELETE 和 TRUNCATE　用来删除记录。
- SELECT（和它的许多变体）　用来检索与特定标准相匹配的特定记录子集。
- LOAD DATA INFILE　用来从一个文本文件导入数据到一个 MySQL 表。

现在我们应该对MySQL命令之间以及MySQL命令与应用程序的其他部分如何相互作用有了进一步的了解。下一章是在本章以及前一章的基础上讲解如何使用 MySQL RDBMS 的一些新的和更先进的特性（包括子查询、事务处理和查询优化）。

第10章 连　　接

本书前面章节中介绍的例子都是一次处理一个单独的表，但是在现实世界中，往往不是这样的情况。高效的要求经常使得对多个表的数据进行分割，而在多个表中创建关系可以更高效地检索数据。

通过支持在相关联的信息中创建连接，一个关系型数据库管理系统（RDBMS）不仅可以更高效地存储信息（通过删除冗余和重复），也可以显示出不同数据的未被发现的关系，并且可有效地利用这些关系。

为此，本章在前面学过的DML概念的基础上阐述SQL如何一次查询多个表，如何以不同的方式对检索到的数据进行组合。这些多表的查询涉及到连接，因为它们把两个或者多个表连接在一起，因而以复杂性、难理解性和可怕性而著称。其实它们多数是无根据的，就像我们看到的那样，连接是相当简单而强大的。本章将介绍如何使用它们与MySQL RDBMS相互作用。

10.1 什么是连接

连接是把不同表的记录链到一起的最普遍的方法。一种错误的观念认为由于MySQL的简单性和源代码开放性，使它不擅长连接。这种观念是错误的。MySQL从一开始就能够很好地支持连接，现在还以支持标准的SQL2连接语句而自夸，这种连接语句可以以多种高级方法来组合表记录。

理解连接的最好方法是利用简单的例子。考虑下面的两个表，一个名为categories，它包含一个新闻类别的列表：

```
+------+-----------------+
| cid  | cname           |
+------+-----------------+
|    1 | Sports          |
|    2 | Current Affairs |
|    3 | Business        |
|    4 | Technology      |
+------+-----------------+
```

另外一个是包含新闻标题的列表：

```
+------+-----------------------------------------------------+------------+-----+
| tid  | tname                                               | tdate      | cid |
+------+-----------------------------------------------------+------------+-----+
| 100  | Probe lands on Mars                                 | 2002-03-19 |   4 |
| 102  | Stock markets crash                                 | 2002-03-05 |   3 |
| 104  | War breaks out between men and fish                 | 2003-08-14 |   2 |
| 106  | Volcanic eruptions on Pluto                         | 2003-01-03 |   2 |
| 107  | Injured football player vows to return              | 2003-06-18 |   1 |
```

```
| 110 | Cell phones found to improve cognition among babies | 2003-09-14 |   4 |
| 113 | Leaders of day-trading scam arrested by FBI         | 2002-12-27 |   3 |
+------+-----------------------------------------------------+------------+-----+
```

可以看出，在前面的两个表中存在一个连接：cid 字段，它可以用来连接每条新闻和它的类别。

现在，我们已经知道如何在其中的任意一个检索所有记录——使用 SELECT * 查询：

```
mysql> SELECT * FROM topics;
+------+-----------------------------------------------------+------------+-----+
| tid  | tname                                               | tdate      | cid |
+------+-----------------------------------------------------+------------+-----+
| 100  | Probe lands on Mars                                 | 2002-03-19 |   4 |
| 102  | Stock markets crash                                 | 2002-03-05 |   3 |
| 104  | War breaks out between men and fish                 | 2003-08-14 |   2 |
| 106  | Volcanic eruptions on Pluto                         | 2003-01-03 |   2 |
| 107  | Injured football player vows to return              | 2003-06-18 |   1 |
| 110  | Cell phones found to improve cognition among babies | 2003-09-14 |   4 |
| 113  | Leaders of day-trading scam arrested by FBI         | 2002-12-27 |   3 |
+------+-----------------------------------------------------+------------+-----+
7 rows in set (0.11 sec)
```

可以通过添加一个 WHERE 子句指定一个必须满足的条件来过滤一些记录：例如，只显示 2003 年的标题：

```
mysql> SELECT tid, tname FROM topics WHERE YEAR(tdate) = 2003;
+------+-----------------------------------------------------+
| tid  | tname                                               |
+------+-----------------------------------------------------+
| 104  | War breaks out between men and fish                 |
| 106  | Volcanic eruptions on Pluto                         |
| 107  | Injured football player vows to return              |
| 110  | Cell phones found to improve cognition among babies |
+------+-----------------------------------------------------+
4 rows in set (0.22 sec)
```

但是，如果需要的是一些更高级的东西：例如，Current Affairs类别中的标题列表或者2003年所有运动标题的列表，情况是什么样呢？在这种情况下，只查询一个单独的表是不够的，因为类别信息和标题摘要存储在两个单独的表中。

不过，如同前面讲到的那样，在两个表中存在一个连接——cid 列，它在两个表中是等同的。在 SELECT 查询的 WHERE 子句中，categories 表的 cid 字段与 topics 表的 cid 字段相等，因此可以创建两个表的一个连接。以 SQL 语言的形式为：

```
mysql> SELECT cname, tname FROM categories, topics WHERE categories.cid =
topics.cid;
+-----------------+-----------------------------------------------------+
| cname           | tname                                               |
+-----------------+-----------------------------------------------------+
| Technology      | Probe lands on Mars                                 |
| Business        | Stock markets crash                                 |
```

```
| Current Affairs | War breaks out between men and fish                |
| Current Affairs | Volcanic eruptions on Pluto                        |
| Sports          | Injured football player vows to return             |
| Technology      | Cell phones found to improve cognition among babies |
| Business        | Leaders of day-trading scam arrested by FBI        |
+-----------------+----------------------------------------------------+
7 rows in set (0.06 sec)
```

这就是我们的第一个连接。

在这个例子中，WHERE子句用cid字段连接两个表，并显示出组合后的结果。但是，这个结果并不是我们想要的。我们只需要Current Affairs类中的标题。幸运的是，WHERE子句将会再次拯救我们：

```
mysql> SELECT tname FROM categories, topics WHERE categories.cid = topics.cid
AND categories.cname = 'Current Affairs';
+-------------------------------------+
| tname                               |
+-------------------------------------+
| War breaks out between men and fish |
| Volcanic eruptions on Pluto         |
+-------------------------------------+
2 rows in set (0.22 sec)
```

避免混乱

当连接多个表时，保证参照字段尤其是同名字段尽可能地明确是非常重要的。如果连接包含的两个或者多个表具有相同的字段名，就必须在每个字段名的前端加上表名作为前缀（如同本章中的许多例子），这样MySQL就可以精确地知道是哪个表的哪个字段。

```
SELECT name FROM employees, departments WHERE employees.id =
departments.eid;            # bad
SELECT employees.name FROM employees, departments WHERE employees.id =
departments.eid;            # good
```

前面SQL语句的WHERE子句作用在两个表结合后的结果集上，过滤后只剩下与指定标准相符合的记录。

第二个问题怎么样呢——2003年所有体育标题的列表？

```
mysql> SELECT tname FROM categories, topics WHERE categories.cid = topics.cid
AND categories.cname = 'Sports' AND YEAR(topics.tdate) = 2003;
+---------------------------------------+
| tname                                 |
+---------------------------------------+
| Injured football player vows to return |
+---------------------------------------+
1 row in set (0.22 sec)
```

因而，通过使用前面两个表隐含的连接，SQL允许我们把它们连接到一起产生一个复合的结果集，然后对其中的数据进行进一步的操作，以获得需要的信息。

10.2 连接的类型

现在已经基本理解了连接如何工作，让我们对 MySQL 支持的连接类型进行更进一步的讨论。下面是 MySQL 支持的不同连接类型：

- 交叉连接
- 内连接
- 外连接
- 自连接
- 联合

下面的部分结合例子和图示详细介绍了每一种连接类型。

10.2.1 交叉连接

连接的最简单类型是交叉连接，它是对涉及到的表相乘创建一个包含所有内容的产物。考虑下面的例子，它包含两个表，一个列出了身体特征，另一个列出了颜色：

```
Table "attribute"
+-----------+
| attribute |
+-----------+
| eyes      |
| hair      |
+-----------+
```

```
Table "color"
+-------+
| color |
+-------+
| brown |
| black |
| gray  |
+-------+
```

可以使用交叉连接把前面的两个例子结合起来，如下所示：

```
mysql> SELECT * FROM color, attribute;
+-------+-----------+
| color | attribute |
+-------+-----------+
| brown | eyes      |
| black | eyes      |
| gray  | eyes      |
| brown | hair      |
| black | hair      |
| gray  | hair      |
+-------+-----------+
6 rows in set (0.00 sec)
```

这个例子中，两个表的所有列结合起来产生了一个包含所有可能组合的结果集。这种类型的连接称为交叉连接，而且连接后的结果表的行数和用于连接的每个表的行数乘积相同。可以从前面的例子中看到这些。属性表有两行，颜色表有三行，所以结果表有 $2 \times 3 = 6$ 行。

与我们想像的一样，前面那样的交叉连接会对数据库服务器的性能产生很大的影响。为了进行说明，请看当我们向前面的连接另外添加两个表时会出现什么情况：

```
mysql> SELECT * FROM sex, shape, color, attribute;
+--------+-------+-------+-----------+
| sex    | shape | color | attribute |
+--------+-------+-------+-----------+
| male   | fat   | brown | eyes      |
| female | fat   | brown | eyes      |
| male   | long  | brown | eyes      |
| female | long  | brown | eyes      |
| male   | thin  | brown | eyes      |
| female | thin  | brown | eyes      |
...
+--------+-------+-------+-----------+
48 rows in set (0.05 sec)
```

虽然连接中用到的每个表都只包含了不到5条记录，但是最后的连接结果包含2 × 3 × 2 × 4 = 48条记录。当只处理4个表一共11条记录时，可能看起来不是大的处理，但是如果有4个表，每个包含100条记录，而且决定对它们进行交叉连接时，想一想会出现什么情况。

数据超载

因为交叉连接产生了一个很大的结果集，所以在连接中使用WHERE子句过滤掉一些记录被认为是一个很好的主意。

10.2.2 内连接

内连接是最普通的连接类型，而且是最匀称的，因为它们要求构成连接的每一部分的每个表的匹配，不匹配的行将被排除在最后的结果集之外。

内连接的最常见的例子是相等连接，也就是连接后的表中的某个字段与每个表中的都相同。这种情况下，最后的结果集只包含参加连接的表中与指定字段相符的行（本章中第一段的例子使用了相等连接）。

为了说明，考虑下面的表，它们列出了产品和每个产品的销售数量：

```
Table "products"
+-----------+-------------+
| productID | productName |
+-----------+-------------+
|         1 | apples      |
|         2 | oranges     |
|         3 | pineapples  |
|         4 | bananas     |
+-----------+-------------+
```

```
Table "sales"
+------+-----------+----------+
| id   | productID | quantity |
+------+-----------+----------+
|    1 |         3 |     2300 |
|    2 |         2 |     1500 |
|    3 |         1 |     3400 |
+------+-----------+----------+
```

因为两个表具有相同的productID字段，所以连接它们很简单：

```
mysql> SELECT productName, quantity FROM products, sales WHERE products.
productID = sales.productID;
+-------------+----------+
| productName | quantity |
+-------------+----------+
| pineapples  |     2300 |
```

```
| oranges      |      1500 |
| apples       |      3400 |
+-------------+----------+
3 rows in set (0.00 sec)
```

这里，我们同样可以使用WHERE子句进一步缩小结果集，例如下面的例子，它只列出了产品销售量超过2000的结果集：

```
mysql> SELECT productName, quantity FROM products, sales WHERE products.
productID = sales.productID AND sales.quantity > 2000;
+-------------+----------+
| productName | quantity |
+-------------+----------+
| pineapples  |     2300 |
| apples      |     3400 |
+-------------+----------+
2 rows in set (0.05 sec)
```

虽然不常见，但是基于相同列的内连接也是可以的。例如，如果我们修改products表，使它包含每个产品的平均销售量：

```
+-----------+-------------+----------+
| productID | productName | avgSales |
+-----------+-------------+----------+
|         1 | apples      |     1400 |
|         2 | oranges     |     2000 |
|         3 | pineapples  |     2500 |
|         4 | bananas     |     1000 |
+-----------+-------------+----------+
```

可以使用下面的查询找到销售量大于平均值的产品，它使用大于操作符来执行比较：

```
mysql> SELECT productName, quantity, avgSales FROM products, sales WHERE
products.productID = sales.productID AND sales.quantity > products.avgSales;
+-------------+----------+----------+
| productName | quantity | avgSales |
+-------------+----------+----------+
| apples      |     3400 |     1400 |
+-------------+----------+----------+
1 row in set (0.05 sec)
```

在这个例子中，只有sales表记录的quantity值比products表相应的avgSales值大才可以被包含在最后的结果集中。

单词游戏

为了清晰，MySQL还允许在这些操作中使用INNER JOIN和CROSS JOIN关键字而不是使用逗号。例如，下面的两个语句都产生了一个交叉连接：

```
SELECT countryName, stateName FROM country, state;
SELECT countryName, stateName FROM country CROSS JOIN state;
```

下面的两个语句都创建了一个内连接：

```
SELECT countryName, stateName FROM country, state WHERE state.cid = country.id
SELECT countryName, stateName FROM country INNER JOIN state WHERE state.cid =
country.id;
```

请注意，使用这两个关键字或者逗号，可以连接尽可能多的表。下面是有效的SQL：

```
SELECT * from country CROSS JOIN state CROSS JOIN city;
```

10.2.3 外连接

从前面可以看到内连接是对称的。为了把它包含在最后的结果集中，所有参加连接的表的行必须匹配。另一方面，外连接是不对称的——连接的一方的所有行都包含在最后的记录中，不管它们是否与连接的另外一方的行匹配。

根据连接的哪一方要被保留，SQL定义了左外连接和右外连接。在左外连接中，与WHERE子句相匹配的连接左部的表的所有记录都将出现在最后结果集中。在右外连接中，与WHERE子句相匹配的连接的右部的表的所有记录都将出现在最后结果集中。

为了清楚我们何时需要这些，考虑下面的三个表。第一个列出了用户，第二个列出了分组，第三个列出了每个用户的组成员关系。一个用户可能属于0个、1个或者更多的组。一个组可能包含0个、1个或者更多的用户。分组和用户表通过users_groups表中的uid和gid字段进行连接。

Table "users"

uid	name
100	sue
103	harry
104	louis
107	sam
110	james
111	mark
112	rita

Table "groups"

gid	name
501	authors
502	actors
503	musicians
504	chefs

Table "users_groups":

uid	gid
11	502
107	502
100	503
110	501
112	501
100	501
102	501
104	502
100	502

知道了内连接，连接这三个表，创建一个每个用户属于哪个组的易读的列表是非常容易的：

```
mysql> SELECT users.name, groups.name FROM users, groups, users_groups WHERE
users.uid = users_groups.uid AND groups.gid = users_groups.gid;
+-------+-----------+
| name  | name      |
+-------+-----------+
| sam   | actors    |
| sue   | musicians |
| james | authors   |
| rita  | authors   |
```

```
| sue   | authors    |
| louis | actors     |
| sue   | actors     |
+-------+------------+
7 rows in set (0.00 sec)
```

这是一个对称的列表，但是它没有告诉我们不属于任何组的用户和没有成员的组。

左外连接

要获得丢失的信息，需要知道左外连接是如何工作的。考虑下面的 SQL 查询：

```
SELECT * FROM users LEFT JOIN users_groups ON users.uid = users_groups.uid;
```

在英语中，这翻译为“从连接的左部（users 表）选择所有的行，对于选中的每一行，或者从右部（users_groups 表）显示相匹配的值（能够满足 ON 或者 USING 子句约束的值），或者显示 NULL 的一行”。这种类型的连接称为左连接，或者称为左外连接。

下面是结果：

```
mysql> SELECT * FROM users LEFT JOIN users_groups ON users.uid = users_groups.
uid;
+------+-------+------+------+
| uid  | name  | gid  | uid  |
+------+-------+------+------+
|  100 | sue   |  503 |  100 |
|  100 | sue   |  501 |  100 |
|  100 | sue   |  502 |  100 |
|  103 | harry | NULL | NULL |
|  104 | louis |  502 |  104 |
|  107 | sam   |  502 |  107 |
|  110 | james |  501 |  110 |
|  111 | mark  | NULL | NULL |
|  112 | rita  |  501 |  112 |
+------+-------+------+------+
9 rows in set (0.05 sec)
```

如同我们看到的那样，来自连接左部的所有记录（users）出现在最后的结果集中。右部具有相应值的记录（组成员关系）显示了该值。剩下的行是 NULL。

当需要查看一个表中的哪个值在另一个表中不具备时，这种类型的连接就使操作变得很方便。我们所需要做的就是要查看 NULL 行。从前面的例子发现，很容易就可以看到用户 harry 和 mark 不属于任何一组，因为在 users_groups 表中，他们没有任何记录。

实际上，我们甚至可以不查看输出结果，使用 SQL 的 WHERE 子句就可完成这项工作：

```
mysql> SELECT name FROM users LEFT JOIN users_groups ON users.uid = users_groups.
uid WHERE users_groups.gid IS NULL;
+-------+
| name  |
+-------+
| harry |
| mark  |
```

```
+--------+
2 rows in set (0.05 sec)
```

当查找表中已破坏的数据时，这项技术非常有用。

如果参加连接的表的字段存在而且与所有参加连接的表中的该字段具有相同的名字，就可以使用MySQL的USING子句来简化ON子句。因此可以这样做：

```
SELECT * FROM users LEFT JOIN users_groups ON users.uid = users_groups.uid;
```

下面的语句是相同的而且更易于读：

```
SELECT * FROM users LEFT JOIN users_groups USING(uid);
```

右外连接

和左外连接一样，也存在右外连接，它和左外连接刚好进行相反的操作。考虑下面的查询和输出，它进行了演示：

```
mysql> SELECT * FROM users_groups RIGHT JOIN groups USING (gid);
+------+------+------+-----------+
| gid  | uid  | gid  | name      |
+------+------+------+-----------+
|  501 |  110 |  501 | authors   |
|  501 |  112 |  501 | authors   |
|  501 |  100 |  501 | authors   |
|  501 |  102 |  501 | authors   |
|  502 |   11 |  502 | actors    |
|  502 |  107 |  502 | actors    |
|  502 |  104 |  502 | actors    |
|  502 |  100 |  502 | actors    |
|  503 |  100 |  503 | musicians |
| NULL | NULL |  504 | chefs     |
+------+------+------+-----------+
10 rows in set (0.00 sec)
```

在英语中，这翻译为“从连接的右部（groups表）选择所有的行，对于选中的每一行，或者从左部（users_groups表）显示相匹配的值（能够满足ON或者USING子句约束的值）或者显示NULL的一行。”

自然顺序

当执行一个连接时，MySQL使用ON或者USING子句指定的条件来连接表创建一个结果集，然后使用WHERE子句指定的测试条件决定哪条记录应包含在结果集中。

因此，我们必须仔细考虑查询字符串在约束中出现的位置。从WHERE子句移动一个约束到ON子句将会改变查询的意思，因而也改变了结果。一些不是结果的记录可能在一边不是NULL值，而在另一边是NULL值。

通常情况下，ON和USING子句不能用来决定最后的结果集中包含哪条记录。WHERE子句适合执行此操作。

因此，使用前面的右外连接，所有的组都将会出现在最后结果集中。拥有成员（根据左边users_groups表中相应的值暗示）的那些组将显示该值，没有成员的组显示NULL值。

通过在结果中浏览NULL值，很容易看到哪一组没有成员：

```
mysql> SELECT * FROM users_groups RIGHT JOIN groups
USING (gid) WHERE uid IS NULL;
+------+------+------+-------+
| gid  | uid  | gid  | name  |
+------+------+------+-------+
| NULL | NULL |  504 | chefs |
+------+------+------+-------+
1 row in set (0.06 sec)
```

更多的例子

假如想要找到哪些组有两个或者更多的成员，通过在前面的部分创建右连接可以很容易地找到。需要做的就是通过gid字段对记录分组，然后用COUNT()统计它们：

```
mysql> SELECT name FROM users_groups RIGHT JOIN groups
USING (gid) GROUP BY users_groups.gid HAVING COUNT(uid) > 2;
+---------+
| name    |
+---------+
| authors |
| actors  |
+---------+
2 rows in set (0.00 sec)
```

角度的问题

术语“左连接”和“右连接”根据所处的角度是可以互换的。通过改变连接中表的顺序，左连接可以转换为右连接（反过来也是一样的）。为了说明，考虑下面的两个查询：

```
SELECT * FROM c LEFT JOIN a USING (id);
SELECT * FROM a RIGHT JOIN c USING (id);
```

翻译过来，其实这两个查询说的是同一件事情：“从表c中选择所有的行，对于每一个选中的行，显示表a中相匹配的值或者NULL。”

如何找到每一组用户的总数？这看起来是一件相当容易的事，一个包含GROUP BY子句的常规的SELECT查询就能完成：

```
mysql> SELECT name, COUNT(*) FROM users_groups, groups WHERE users_groups.
gid = groups.gid GROUP BY users_groups.gid;
+-----------+----------+
| name      | COUNT(*) |
+-----------+----------+
| authors   |        4 |
| actors    |        4 |
| musicians |        1 |
+-----------+----------+
3 rows in set (0.55 sec)
```

但是对于没有成员的组不能进行计算。为了保证对所有的组都能计算，需要使用一个外连接，如同下面的左连接：

```
mysql> SELECT name, COUNT(uid) FROM groups LEFT JOIN users_groups
ON users_groups.gid = groups.gid GROUP BY groups.gid;
+-----------+------------+
| name      | COUNT(uid) |
+-----------+------------+
| authors   |          4 |
| actors    |          4 |
| musicians |          1 |
| chefs     |          0 |
+-----------+------------+
4 rows in set (0.00 sec)
```

如果一个组已经为一个用户分配了成员关系，可是这个用户不存在时，是否想要找到他呢？尝试下面的语句：

```
mysql> SELECT users_groups.uid, users_groups.gid, users.name FROM users RIGHT
JOIN users_groups USING (uid) WHERE name IS NULL;
+------+------+------+
| uid  | gid  | name |
+------+------+------+
|   11 |  502 | NULL |
|  102 |  501 | NULL |
+------+------+------+
2 rows in set (0.00 sec)
```

这是使用IS NULL测试与一个连接相结合来查找讹误的数据的一个很好的例子。在这个例子中，把users_groups表（连接的右方）的每一条记录与user表的记录进行核对，来查找匹配。结果集包含了两个NULL，说明一些组被指定了不存在的用户——表明数据可能遭到了破坏。

不同的方法

是否有兴趣看到每个用户属于哪个组？我们不能使用一个连接（或者至少不能以简洁的方式）。但是，我们可以使用GROUP_CONCAT()函数，这是MySQL 4.1新增的功能。考虑下面的例子，它对此进行了说明：

```
mysql> SELECT g.name, GROUP_CONCAT(u.name SEPARATOR ',') FROM users
AS u, groups AS g, users_groups AS ug WHERE ug.gid = g.gid AND
ug.uid = u.uid GROUP BY ug.gid;
```

10.2.4 自连接

除了交叉连接、内连接和外连接，MySQL还支持第四种类型的连接，也就是自连接。这种类型的连接把一个表与它自身进行连接，它通常用来取出表中彼此包含内连接的记录。考虑下面的表：

```
Table "menu"
+----+-----------------------+--------+
| id | label                 | parent |
+----+-----------------------+--------+
|  1 | Services              |      0 |
|  2 | Company               |      0 |
```

```
|  3 | Media Center            |      0 |
|  4 | Your Account            |      0 |
|  5 | Community               |      0 |
|  6 | For Content Publishers  |      1 |
|  7 | For Small Businesses    |      1 |
|  8 | Background              |      2 |
|  9 | Clients                 |      2 |
| 10 | Addresses               |      2 |
| 11 | Jobs                    |      2 |
| 12 | News                    |      2 |
| 13 | Press Releases          |      3 |
| 14 | Media Kit               |      3 |
| 15 | Log In                  |      4 |
| 16 | Columns                 |      5 |
| 17 | Colophon                |     16 |
| 18 | Cut                     |     16 |
| 19 | Boombox                 |     16 |
+----+-------------------------+--------+
```

这是一个简单的菜单结构，每条记录是菜单树中一个惟一的节点。每条记录有一个惟一的记录ID，而且包含一个父ID。这两个ID用来定义菜单树中分支的父子关系。所以，如果想要分层表示前面的数据，它会变成如图10.1所示的样子。

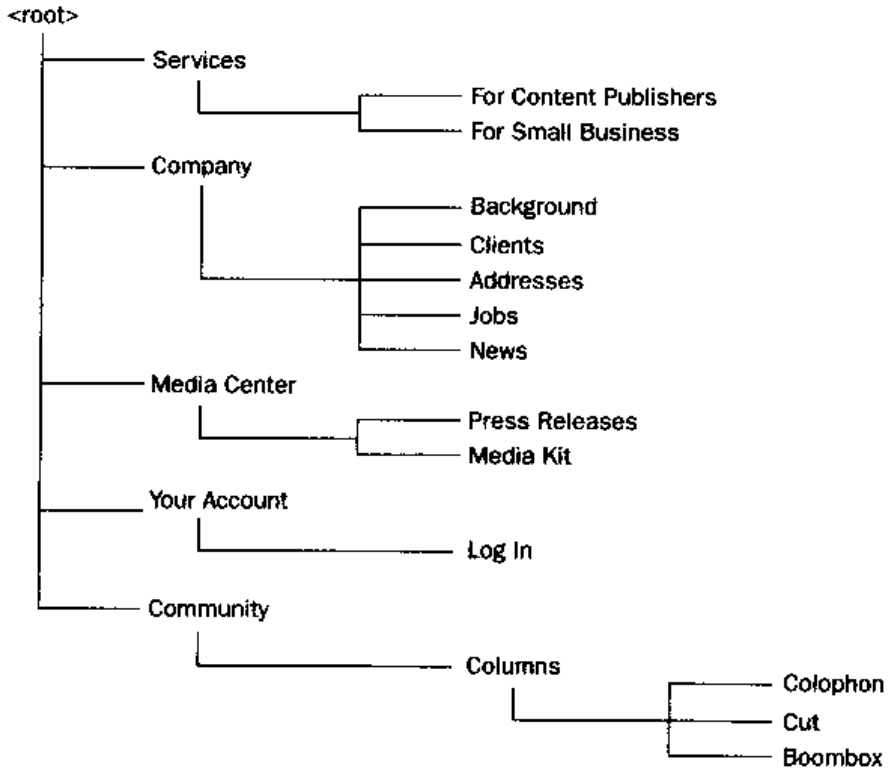

图10.1 层次菜单树

现在，假设需要显示树上所有的节点，以及它们父母的名字，也就是说，像下面这样的结果：

```
+--------------+-------------------+
| parent_label | child_label       |
+--------------+-------------------+
```

```
| Services      | For Content Publishers |
| Company       | Background             |
| Your Account  | Log In                 |
   ... snip ...
+---------------+------------------------+
```

如果仔细考虑，我们会发现没有容易的方法来获得结果集。因为所有的数据在同一个单独的表中，简单的SELECT不能实现，复杂的外连接也不行。这里需要的是一个自连接，它创建了第一个表的虚拟拷贝，然后使用一个常规的内连接来把两个表映射到一起，并且得到需要的输出，然后显示。如：

```
mysql> SELECT a.label AS parent_label, b.label AS child_label FROM menu AS a,
menu AS b WHERE a.id = b.parent;
+---------------+------------------------+
| parent_label  | child_label            |
+---------------+------------------------+
| Services      | For Content Publishers |
| Services      | For Small Businesses   |
| Company       | Background             |
| Company       | Clients                |
| Company       | Addresses              |
| Company       | Jobs                   |
| Company       | News                   |
| Media Center  | Press Releases         |
| Media Center  | Media Kit              |
| Your Account  | Log In                 |
| Community     | Columns                |
| Columns       | Colophon               |
| Columns       | Cut                    |
| Columns       | Boombox                |
+---------------+------------------------+
16 rows in set (0.33 sec)
```

最神奇的在于表的别名。前面的查询创建了menu表的两份副本，然后分别把它们重命名为a和b。结果是下面的两个虚拟表：

```
Table "a"
+----+------------------------+--------+
| id | label                  | parent |
+----+------------------------+--------+
|  1 | Services               |      0 |
|  2 | Company                |      0 |
|  3 | Media Center           |      0 |
|  4 | Your Account           |      0 |
|  5 | Community              |      0 |
|  6 | For Content Publishers |      1 |
|  7 | For Small Businesses   |      1 |
|  8 | Background             |      2 |
|  9 | Clients                |      2 |
| 10 | Addresses              |      2 |
```

```
Table "b"
+----+------------------------+--------+
| id | label                  | parent |
+----+------------------------+--------+
|  1 | Services               |      0 |
|  2 | Company                |      0 |
|  3 | Media Center           |      0 |
|  4 | Your Account           |      0 |
|  5 | Community              |      0 |
|  6 | For Content Publishers |      1 |
|  7 | For Small Businesses   |      1 |
|  8 | Background             |      2 |
|  9 | Clients                |      2 |
| 10 | Addresses              |      2 |
```

```
| 11 | Jobs                    |      2 |   | 11 | Jobs                    |      2 |
| 12 | News                    |      2 |   | 12 | News                    |      2 |
| 13 | Press Releases          |      3 |   | 13 | Press Releases          |      3 |
| 14 | Media Kit               |      3 |   | 14 | Media Kit               |      3 |
| 15 | Log In                  |      4 |   | 15 | Log In                  |      4 |
| 16 | Columns                 |      5 |   | 16 | Columns                 |      5 |
| 17 | Colophon                |     16 |   | 17 | Colophon                |     16 |
| 18 | Cut                     |     16 |   | 18 | Cut                     |     16 |
| 19 | Boombox                 |     16 |   | 19 | Boombox                 |     16 |
+----+-------------------------+--------+   +----+-------------------------+--------+
```

一旦创建了两个表，把它们连接到一起是一件很简单的事，只需使用节点ID作为共有列，就可获得以需要的格式标记子和父的一个列表。

10.2.5 联合

除了连接，MySQL 4.0以及更高的版本还支持UNION运算符，它用来把多个SELECT查询的输出连接成一个单独的结果集。大多数情况下，这个运算符用来把查询产生的结果集添加到不同的表，同时创建包括所有结果的一个单独表。

为了说明UNION运算符的使用方法，考虑下面的例子，一个表列出了交易A的股票，另一个表列出了交易B的股票：

```
Table "exchangeA"                Table "exchangeB"
+--------+--------+              +--------+-------+
| symbol | price  |              | symbol | price |
+--------+--------+              +--------+-------+
| HDGS   |   6.78 |              | POYT   | 87.10 |
| TDGB   |  78.44 |              | DFRM   |  9.43 |
| HDTE   | 123.71 |              | HTYF   | 89.70 |
| BGHU   |  12.90 |              | TDGB   | 79.00 |
| HTYF   |  90.10 |              +--------+-------+
| WERR   |  32.91 |
| YTEM   |  39.65 |
+--------+--------+
```

现在，UNION运算符可以把前面两个表组合为一个单独的表：

```
mysql> SELECT symbol, price FROM exchangeA
UNION
SELECT symbol, price FROM exchangeB;
+--------+--------+
| symbol | price  |
+--------+--------+
| HDGS   |   6.78 |
| TDGB   |  78.44 |
| HDTE   | 123.71 |
| BGHU   |  12.90 |
| HTYF   |  90.10 |
| WERR   |  32.91 |
| YTEM   |  39.65 |
```

```
| POYT    |  87.10 |
| DFRM    |   9.43 |
| HTYF    |  89.70 |
| TDGB    |  79.00 |
+--------+--------+
11 rows in set (0.22 sec)
```

可以使用一个（或者多个）WHERE子句以普通的方式对每个SELECT查询的结果进行过滤：

```
mysql> SELECT symbol, price FROM exchangeA WHERE price > 35.00
UNION
SELECT symbol, price FROM exchangeB WHERE price BETWEEN 40.00 AND 95.00;
+--------+--------+
| symbol | price  |
+--------+--------+
| TDGB    |  78.44 |
| HDTE    | 123.71 |
| HTYF    |  90.10 |
| YTEM    |  39.65 |
| POYT    |  87.10 |
| HTYF    |  89.70 |
| TDGB    |  79.00 |
+--------+--------+
7 rows in set (0.16 sec)
```

可以使用UNION运算符组合尽可能多的SELECT查询，但是必须具备两个基本的条件。首先，每个SELECT查询返回的字段的个数必须相同。第二，每个SELECT查询的字段的数据类型必须互相符合。

UNION运算符将会自动从组合结果集中消除重复的行（这个行为与在一个常规SELECT查询添加DISTINCT关键字得到的结果相似）。在前面的第一个例子已经清楚地进行了说明。结果集只包含股票的一条记录，例如HTYF和TDGB，它们代表了两次交易。如果我们想看UNION中的所有记录，可以通过向UNION运算符中添加ALL关键字来关掉MySQL自动过滤重复记录的功能。如：

```
mysql> SELECT symbol FROM exchangeA UNION ALL SELECT symbol FROM exchangeB;
+--------+
| symbol |
+--------+
| HDGS    |
| TDGB    |
| HDTE    |
| BGHU    |
| HTYF    |
| WERR    |
| YTEM    |
| POYT    |
| DFRM    |
| HTYF    |
```

```
| TDGB    |
+---------+
11 rows in set (0.17 sec)
```

普通常识

在 UNION 中向 SELECT 查询添加 ORDER BY 子句通常是没有太大意义的，因为每个查询产生的结果集对于用户是不可见的，只有最后的结果是可见的。

最后可以通过在UNION操作中查询的结尾添加ORDER BY子句对返回的结果集进行排序。不过，需要对每个SELECT查询用括号括起来，这样MySQL知道ORDER BY子句是针对最后的结果集进行的，而不是对结果集的最后一个SELECT。下面的例子进行了说明，它以颠倒的字母顺序对股票列表进行排序：

```
mysql> (SELECT symbol FROM exchangeA) UNION (SELECT symbol FROM exchangeB)
ORDER BY symbol DESC;
+---------+
| symbol  |
+---------+
| YTEM    |
| WERR    |
| TDGB    |
| POYT    |
| HTYF    |
| HDTE    |
| HDGS    |
| DFRM    |
| BGHU    |
+---------+
9 rows in set (0.22 sec)
```

10.3 小结

本章讨论了MySQL的一个强有力的数据操作特性——连接，以及结合使用这些连接如何创建复杂的和有力的查询的例子。

一个连接可以在表共有字段的基础上组合两个或者多个表，然后对结果执行SELECT查询。MySQL支持多种不同连接，包含以下几种：

- 交叉连接。它涉及到多个表，创建一个包含所有可能排列变化的组合表。
- 内连接。它只产生所有表中存在匹配的记录。
- 外连接。它产生了连接一方的所有记录，如果是空白，就用NULL来代替。
- 自连接。它通过为表另取一个名字来复制表，然后通过其他连接方式在复制品之间进行连接。
- 联合。它通过对表中涉及到的所有记录进行添加来创建一个单独的、合成的组合。

第11章 子 查 询

一般情况下，可通过添加WHERE或者HAVING子句对查询结果进行限制，这些子句包含了一个或者多个条件表达式，用来从结果集中过滤掉不相关的记录。大多数情况下，这些条件检测使用固定的常数，例如“列出年龄大于40的所有用户”或者“显示1月到6月的所有发货单”，这使它们很容易被写出和维护。

不过，当条件检测的查询使用的是另一个查询生成的值时，经常会产生新情况，例如，“列出大于用户平均年龄的所有用户”或者“显示出最小顾客组中的最大发货单”。在所有这样的例子中，一个查询的结果决定于另一个查询产生的数据，在外查询的条件测试中使用常量是不行的。

在MySQL 4.1以前的版本中，惟一解决此类问题的方法是分别执行每一个查询，使用来自另一个查询的条件子句的数据。MySQL 4.1（以及更高版本）通过引进MySQL邮件列表中要求最频繁的特性之一——子查询来处理这件事。

用户留心

子查询虽然很有用，但是很大程度上影响了MySQL RDBMS的性能。这是因为到目前为止，子查询只由MySQL 4.1-alpha版本支持，性能在任何规模的数据中都处于次要的位置。在现在的版本中，我们可以使用连接、联合、多表更新或者删除以及临时表来替代子查询，但是我们希望将来可以改进RDBMS在这方面的性能。

11.1 什么是子查询

顾名思义，子查询就是一个SELECT查询是另一个查询的附属。MySQL 4.1可以嵌套多个查询，在外面一层的查询中使用里面一层查询产生的结果集。这样就不是执行两个（或者多个）独立的查询，而是执行包含一个（或者多个）子查询的单独查询。

当遇到这样的多层查询时，MySQL从最内层的查询开始，然后从它开始向外向上移动到外层（主）查询。在这个过程中每个查询产生的结果集都被赋给包围它的父查询，接着这个父查询被执行，它的结果也被指定给它的父查询。

除了结果集经常由包含一个或者多个值的一列组成外，子查询和常规SELECT查询的执行方式一样。子查询可以用在任何可以使用表达式的地方，它必须由父查询包围，而且，如同常规的SELECT查询，它必须包含一个字段列表（这是一个单列列表）、一个具有一个或者多个表名字的FROM子句以及可选的WHERE，HAVING和GROUP BY子句。

为了弄清楚子查询的作用，我们来考虑一个简单的例子，一个虚构的公司账目数据库，它包含下面4个表：

1. services表：虚拟的公司为用户提供了一些外包服务，每一项服务都与费用和惟一的服务ID相联系。这些信息存储在services表中，如下所示。

sid	sname	sfee
1	Accounting	1500.00
2	Recruitment	500.00
3	Data Management	300.00
4	Administration	500.00
5	Customer Support	2500.00
6	Security	600.00

2. clients 表：公司还有一个存储在独立 clients 表内的当前客户的列表。每个客户通过惟一的顾客 ID 识别。

cid	cname
101	JV Real Estate
102	ABC Talent Agency
103	DMW Trading
104	Rabbit Foods Inc
110	Sharp Eyes Detective Agency

3. branches 表：每个顾客可能属于一个或者多个分支机构。branches 表列出了每个顾客所属的分支机构，以及每个分支机构的位置。每个分支机构有一个描述、一个惟一的分支机构 ID 和一个参照顾客 ID 的外键。

bid	cid	bdesc	bloc
1011	101	Corporate HQ	CA
1012	101	Accounting Department	NY
1013	101	Customer Grievances Department	KA
1041	104	Branch Office (East)	MA
1042	104	Branch Office (West)	CA
1101	110	Head Office	CA
1031	103	N Region HO	ME
1032	103	NE Region HO	CT
1033	103	NW Region HO	NY

4. branches_services 表：这个表列出了每个分支机构提供的服务，而且表中包含了分支机构 ID 和服务 ID（分别是 branches 和 services 表的外键）。

bid	sid
1011	1
1011	2
1011	3

```
| 1011 |    4 |
| 1012 |    1 |
| 1013 |    5 |
| 1041 |    1 |
| 1041 |    4 |
| 1042 |    1 |
| 1042 |    4 |
| 1101 |    1 |
| 1031 |    2 |
| 1031 |    3 |
| 1031 |    4 |
| 1032 |    3 |
| 1033 |    4 |
+------+-----+
```

现在，我们已经知道可以使用简单的SELECT * FROM查询获得表中所有记录的列表，例如关于所有客户的列表：

```
mysql> SELECT * FROM clients;
+-----+----------------------------+
| cid | cname                      |
+-----+----------------------------+
| 101 | JV Real Estate             |
| 102 | ABC Talent Agency          |
| 103 | DMW Trading                |
| 104 | Rabbit Foods Inc           |
| 110 | Sharp Eyes Detective Agency |
+-----+----------------------------+
5 rows in set (0.22 sec)
```

还可以向SELECT语句添加一个连接和一个WHERE子句对记录列表进行过滤，只剩下与指定标准相匹配的记录，例如分支机构在加利福尼亚的所有客户的列表：

```
mysql> SELECT cname, bdesc, bloc FROM clients, branches WHERE clients.cid =
branches.cid AND branches.bloc = 'CA';
+----------------------------+---------------------+------+
| cname                      | bdesc               | bloc |
+----------------------------+---------------------+------+
| JV Real Estate             | Corporate HQ        | CA   |
| Rabbit Foods Inc           | Branch Office (West) | CA   |
| Sharp Eyes Detective Agency | Head Office         | CA   |
+----------------------------+---------------------+------+
3 rows in set (0.00 sec)
```

对于比较复杂的事情怎么样？我们需要一个Rabbit Foods Inc所属的分支机构的列表。现在可以通过运行两个SELECT查询实现。首先得到Rabbit Foods Inc的顾客ID，然后使用另一个查询中的ID（104）得到与这个顾客相连接的分支机构列表：

```
mysql> SELECT cid FROM clients WHERE cname = 'Rabbit Foods Inc';
+-----+
| cid |
+-----+
```

```
| 104 |
+-----+
1 row in set (0.17 sec)

mysql> SELECT bdesc FROM branches WHERE cid = 104;
+----------------------+
| bdesc                |
+----------------------+
| Branch Office (East) |
| Branch Office (West) |
+----------------------+
2 rows in set (0.17 sec)
```

通过对 clients 和 branches 表进行相等连接：

```
mysql> SELECT bdesc FROM branches, clients WHERE clients.cid = branches.cid
AND clients.cname = 'Rabbit Foods Inc';
+----------------------+
| bdesc                |
+----------------------+
| Branch Office (East) |
| Branch Office (West) |
+----------------------+
2 rows in set (0.22 sec)
```

或者使用一个子查询：

```
mysql> SELECT bdesc FROM branches WHERE cid = (SELECT cid FROM clients WHERE
cname = 'Rabbit Foods Inc');
+----------------------+
| bdesc                |
+----------------------+
| Branch Office (East) |
| Branch Office (West) |
+----------------------+
2 rows in set (0.22 sec)
```

因而，一个子查询可以把两个或者多个查询组合成一个语句，然后在一个查询的条件从句中使用另一个查询的结果。

一个子查询必须返回一个单列的结果，否则 MySQL 将不知道如何处理结果集。考虑下面的例子，它通过使一个子查询返回一个两列的结果集进行了演示：

```
mysql> SELECT bdesc FROM branches WHERE cid = (SELECT cid, cname FROM
clients WHERE cname = 'Rabbit Foods Inc');
ERROR 1239: Cardinality error (more/less than 1 columns)
```

只要符合前面讨论的基本规则，可以嵌套任意多层的子查询。考虑下面的例子，它通过列出 Sharp Eyes Detective Agency 使用的服务进行了演示：

```
mysql> SELECT sname FROM services WHERE sid = (SELECT sid FROM
branches_services WHERE bid = (SELECT bid FROM branches WHERE cid =
(SELECT cid FROM clients WHERE cname = 'Sharp Eyes Detective Agency')));
```

```
+------------+
| sname      |
+------------+
| Accounting |
+------------+
1 row in set (0.28 sec)
```

11.2 子查询的类型

子查询有不同的使用方法：

- 在一个 WHERE 或者 HAVING 子句中使用。
- 与比较和逻辑操作符一起使用。
- 与 IN 成员测试一起使用。
- 与 EXISTS 布尔测试一起使用。
- 在一个 FROM 子句中使用。
- 与连接一起使用。
- 与 UPDATE 与 DELETE 查询一起使用。

下面的几个部分详细介绍这些类型。

11.2.1 子查询和 WHERE/HAVING 子句

MySQL 可以在 WHERE 子句（对 SELECT…WHERE 返回的记录进行约束）或者 HAVING 子句（对 SELECT…GROUP BY 创建的组进行约束）中包含子查询。包含在圆括号内的子查询，优先级高于比较和逻辑操作符、IN 操作符或者 EXISTS 操作符。

子查询和比较运算符

如果一个子查询产生了一个单独的值，就可以使用 MySQL 的比较操作符对结果和外部查询的 WHERE 或者 HAVING 子句指定的条件表达式进行比较。为了进行说明，假如我们需要所有属于两个分支机构的顾客的列表。一般情况下，首先需要获得每个顾客的分支机构数：

```
mysql> SELECT cid, COUNT(bid) FROM branches GROUP BY cid;
+-----+------------+
| cid | count(bid) |
+-----+------------+
| 101 |          3 |
| 103 |          3 |
| 104 |          2 |
| 110 |          1 |
+-----+------------+
4 rows in set (0.27 sec)
```

然后使用 HAVING 子句过滤出只有两个机构的记录：

```
mysql> SELECT cid, COUNT(bid) FROM branches GROUP BY cid HAVING COUNT(bid) = 2;
+-----+------------+
| cid | count(bid) |
```

```
+-----+-----------+
| 104 |         2 |
+-----+-----------+
1 row in set (0.16 sec)
```

然后根据客户 ID 从 clients 表中获得客户的名字：

```
mysql> SELECT cname FROM clients WHERE cid = 104;
+------------------+
| cname            |
+------------------+
| Rabbit Foods Inc |
+------------------+
1 row in set (0.22 sec)
```

下面的子查询可以一步就完成前面三步的工作：

```
mysql> SELECT cname FROM clients WHERE cid = (SELECT cid FROM branches
GROUP BY cid HAVING COUNT(bid) = 2);
+------------------+
| cname            |
+------------------+
| Rabbit Foods Inc |
+------------------+
1 row in set (0.28 sec)
```

在这个例子中，首先执行内部的查询——这个查询通过顾客ID以及计算每个组记录（分支部门）的总个数对分支进行分组。那些有两个分支部门的顾客通过使用一个HAVING子句很容易就被过滤出，同时相应的顾客 ID 返回给主查询。然后把 ID 映射到 clients 表中，并返回相应的顾客名字。

如何挑选出服务费用最高的顾客？

```
mysql> SELECT cname, bdesc FROM clients, branches, branches_services, ser-
vices WHERE services.sid = branches_services.sid AND clients.cid = branches.
cid AND branches.bid = branches_services.bid AND sfee = (SELECT MAX(sfee)
FROM services);
+----------------+--------------------------------+
| cname          | bdesc                          |
+----------------+--------------------------------+
| JV Real Estate | Customer Grievances Department |
+----------------+--------------------------------+
1 row in set (0.01 sec)
```

注释： 在使用子查询时，我们甚至可以使用 AND 或者 OR 逻辑运算符对条件测试添加进一步的约束，或者使用 NOT 逻辑符逆转它。

也可以在一个HAVING子句中的比较运算符前使用子查询，从而用它对父查询创建的组进行过滤。一个应用的例子是找到使用所有服务多于 50% 的分支机构：

```
mysql> SELECT bid FROM branches_services GROUP BY bid HAVING COUNT(sid)
> (SELECT COUNT(*) FROM services)/2;
```

```
+------+
| bid  |
+------+
| 1011 |
+------+
1 row in set (0.22 sec)
```

还可以添加一个快速的内连接来得到分支机构的名字和顾客名字。如：

```
mysql> SELECT c.cid, c.cname, b.bid, b.bdesc FROM clients AS c, branches AS
b, branches_services AS bs WHERE c.cid = b.cid AND b.bid = bs.bid GROUP BY bs.
bid HAVING COUNT(bs.sid) > (SELECT COUNT(*) FROM services)/2;
+-----+----------------+------+--------------+
| cid | cname          | bid  | bdesc        |
+-----+----------------+------+--------------+
| 101 | JV Real Estate | 1011 | Corporate HQ |
+-----+----------------+------+--------------+
1 row in set (0.28 sec)
```

或者，使用一个子查询得到使用所有服务的分支机构的列表，如下面的例子所示：

```
mysql> SELECT branches.bid, COUNT(sid) FROM branches, branches_services WHERE
branches.bid = branches_services.bid GROUP BY branches.bid HAVING COUNT(sid)
= (SELECT COUNT(*) FROM services);
Empty set (0.04 sec)
```

如果我们看这些原始数据，将会看到没有一个分支部门使用所有服务。但是，我们可以上升一级，看是否有客户通过他们所在的分支机构使用所有的服务，用下面的查询：

```
mysql> SELECT clients.cname FROM clients, branches, branches_services
WHERE branches.bid = branches_services.bid AND branches.cid = clients.cid
GROUP BY clients.cid HAVING COUNT(sid) = (SELECT COUNT(*) FROM services);
+----------------+
| cname          |
+----------------+
| JV Real Estate |
+----------------+
1 row in set (0.01 sec)
```

子查询和IN运算符

只有子查询返回的结果列包含一个值时，比较运算符才适用。假如一个子查询返回的结果集是值的列表，这时比较运算符就必须用IN运算符代替。

IN运算符可以检测结果集中是否存在某个特定的值，如果检测成功就执行外部的查询。为了说明，假设我们需要某个分支机构（设分支ID为1031）使用的所有服务的列表。一般情况下，我们首先需要得到这个分支的所有服务ID的列表：

```
mysql> SELECT sid FROM branches_services WHERE bid = 1031;
+-----+
| sid |
+-----+
|   2 |
```

```
|   3 |
|   4 |
-----+
3 rows in set (0.16 sec)
```

然后在 services 表中查询每个服务 ID 相应的名字：

```
mysql> SELECT sname FROM services WHERE sid = 2;
+--------------+
| sname        |
+--------------+
| Recruitment  |
+--------------+
1 row in set (0.28 sec)

mysql> SELECT sname FROM services WHERE sid = 3;
+------------------+
| sname            |
+------------------+
| Data Management  |
+------------------+
1 row in set (0.17 sec)

mysql> SELECT sname FROM services WHERE sid = 4;
+----------------+
| sname          |
+----------------+
| Administration |
+----------------+
1 row in set (0.11 sec)
```

使用一个子查询和 IN 测试，会变得很冗长。如：

```
mysql> SELECT sname FROM services WHERE sid IN (SELECT sid FROM
branches_services WHERE bid = 1031);
+------------------+
| sname            |
+------------------+
| Recruitment      |
| Data Management  |
| Administration   |
+------------------+
3 rows in set (0.27 sec)
```

在这个例子中，MySQL 从 services 表中选择与子查询返回的服务 ID 集合相匹配的记录。这个例子的一个变化可以获得所有使用 Accounting 服务的分支机构列表（服务 ID = 1）：

```
mysql> SELECT bdesc FROM branches WHERE bid IN (SELECT bid FROM
branches_services WHERE sid = 1);
+-----------------------+
| bdesc                 |
+-----------------------+
| Corporate HQ          |
```

```
| Accounting Department |
| Branch Office (East)  |
| Branch Office (West)  |
| Head Office           |
+-----------------------+
5 rows in set (0.17 sec)
```

这并不是很有用。这里可能合适的是每个分支的顾客名字——通过添加一个连接很简单地就可以完成：

```
mysql> SELECT cname, bdesc FROM branches, clients WHERE branches.bid IN
(SELECT bid FROM branches_services WHERE sid = 1) AND clients.cid =
branches.cid;
+----------------------------+----------------------+
| cname                      | bdesc                |
+----------------------------+----------------------+
| JV Real Estate             | Corporate HQ         |
| JV Real Estate             | Accounting Department |
| Rabbit Foods Inc           | Branch Office (East) |
| Rabbit Foods Inc           | Branch Office (West) |
| Sharp Eyes Detective Agency | Head Office          |
+----------------------------+----------------------+
5 rows in set (0.16 sec)
```

如果只需要顾客列表，则只需添加DISTINCT关键字：

```
mysql> SELECT DISTINCT cname FROM branches, clients WHERE branches.bid IN (SELECT
bid FROM branches_services WHERE sid = 1) AND clients.cid = branches.cid;
+----------------------------+
| cname                      |
+----------------------------+
| JV Real Estate             |
| Rabbit Foods Inc           |
| Sharp Eyes Detective Agency |
+----------------------------+
3 rows in set (0.17 sec)
```

或者使用IN测试再次改写查询：

```
mysql> SELECT cname FROM clients WHERE cid IN (select cid from branches where
bid IN (SELECT bid FROM branches_services WHERE sid = 1));
+----------------------------+
| cname                      |
+----------------------------+
| JV Real Estate             |
| Rabbit Foods Inc           |
| Sharp Eyes Detective Agency |
+----------------------------+
3 rows in set (0.22 sec)
```

假如我们需要一个重要顾客的列表——在每个分支机构中每月账单超过2000的所有顾客。可以使用包含一个连接、一个GROUP BY子句和IN运算符的子查询来获得这个信息。

```
mysql> SELECT cname FROM clients WHERE cid IN (SELECT cid FROM branches
WHERE bid IN (SELECT bid FROM branches_services AS bs, services AS s
WHERE bs.sid = s.sid GROUP BY bid HAVING SUM(sfee) >= 2000));
+------------------+
| cname            |
+------------------+
| JV Real Estate   |
| Rabbit Foods Inc |
+------------------+
2 rows in set (1.32 sec)
```

子查询和 EXISTS 运算符

EXISTS运算符可以用来检查一个子查询是否产生了结果，这使得可以只在EXISTS检测返回真时才执行外部的查询。

各有千秋

和比较运算符一样，我们可以使用NOT关键字来对IN运算符返回的结果进行逆转，即返回那些与子查询产生的结果集不匹配的记录。为了说明这一点，我们把前面的查询：

```
SELECT cname FROM clients WHERE cid IN (SELECT cid FROM branches
WHERE bid IN (SELECT bid FROM branches_services AS bs, services AS s
WHERE bs.sid = s.sid GROUP BY bid HAVING SUM(sfee) >= 2000));
```

改写成下面的类似方式，它们返回相同的结果集：

```
SELECT cname FROM clients WHERE cid IN (SELECT cid FROM branches
WHERE bid NOT IN (SELECT bid FROM branches_services AS bs, services
AS s WHERE bs.sid = s.sid GROUP BY bid HAVING SUM(sfee) < 2000));
```

下面是一个简单的例子：

```
mysql> SELECT * FROM clients WHERE EXISTS (SELECT bid FROM branches_services
GROUP BY bid HAVING COUNT(sid) >= 5);
Empty set (0.17 sec)
```

在这个例子中，因为子查询返回了一个空的结果集——没有一个分支机构使用5个以及更多的服务，EXISTS 测试将返回错误，而且外部的查询不会执行。

当我们想让前面的子查询返回值时看看会发生什么情况：

```
mysql> SELECT * FROM clients WHERE EXISTS (SELECT bid FROM
branches_services GROUP BY bid HAVING COUNT(sid) >= 4);
+-----+-----------------------------+
| cid | cname                       |
+-----+-----------------------------+
| 101 | JV Real Estate              |
| 102 | ABC Talent Agency           |
| 103 | DMW Trading                 |
| 104 | Rabbit Foods Inc            |
| 110 | Sharp Eyes Detective Agency |
+-----+-----------------------------+
5 rows in set (0.27 sec)
```

在这个例子中，因为一些分支使用了4个或者更多的服务，内查询将会返回至少包含一行的结果集，EXISTS 检测将会返回真，外查询就被执行。

前面的外查询返回的结果集没有列出多于4个服务的顾客，注意到这一点是很重要的。结果只是一个所有顾客的列表，而且只是因为内查询产生了结果集它才返回值。在前面的例子中，内查询返回的结果集是不重要的，我们可以使用下面的查询完成同样的事情：

```
mysql> SELECT * FROM clients WHERE EXISTS (SELECT 1);
+-----+-----------------------------+
| cid | cname                       |
+-----+-----------------------------+
| 101 | JV Real Estate              |
| 102 | ABC Talent Agency           |
| 103 | DMW Trading                 |
| 104 | Rabbit Foods Inc            |
| 110 | Sharp Eyes Detective Agency |
+-----+-----------------------------+
5 rows in set (0.16 sec)
```

EXISTS 运算符经常用于 SQL 称为外部参照的环境中。

外部参照

多数情况下，如果一个子查询每次运行产生相同的结果集，那么它就是有意义的、性能好的，只需要执行一次，就可以使用这个结果集来对主查询产生的记录进行测试。但是，如果一个子查询在它的子句中使用主查询的一个字段就会产生问题。像这样的参照，子查询使用它外部查询的一个字段，称为外部参照，而相应的子查询称为相关子查询，因为它与包含在它之内的一个或者多个查询的结果集相关。

当在一个子查询中出现外部参照时，MySQL 必须根据外查询产生的每条记录对子查询运行一次，因而检测子查询的次数就和外查询结果集中记录的个数一样多。在这样的环境中，EXISTS 运算符可以很方便地过滤出最后结果集中的某些记录。

例子可以更清楚地说明这些问题。考虑下面的例子，它结合外部参照重新运行了前面的例子来使结果集更有用：

```
mysql> SELECT bid, bdesc FROM branches WHERE EXISTS (SELECT bid FROM
branches_services WHERE branches.bid = branches_services.bid GROUP BY bid
HAVING COUNT(sid) >= 4);
+------+--------------+
| bid  | bdesc        |
+------+--------------+
| 1011 | Corporate HQ |
+------+--------------+
1 row in set (0.22 sec)
```

如果我们查看数据，可以看到，实际上只有一个分支使用了4个或者更多的服务（分支 ID 为 1011）。

在这个例子中，因为内查询包含了外查询的一个字段的参照，MySQL 不能只运行内查询一次（就像平时所做的）。它必须一次又一次地运行，对外部表的每一行都运行一次，代替来自于查询中的那一行的字段的值，然后根据相应的子查询是否返回一个结果集来决定在最后的

结果集中是否包含那个外部行。这明显地影响到性能，所以外部参照应该避免，除非遇到绝对需要的情况。

考虑资源

外部参照突出了查询优化中的一个有趣问题。在涉及到外部参照的情况下，MySQL必须对外查询产生的记录重新运行内查询，因为内查询使用的是来自外查询结果集中的值。这就占用了大量的服务器资源，同时潜在地降低了性能。所以，在子查询中应该尽可能避免外部参照，应该研究连接数据的另一种方法(例如，自连接或者联合)，因为它们经常花费较少的时间和资源。

例如，考虑前面例子的相关子查询：

```
SELECT bid, bdesc FROM branches WHERE EXISTS (SELECT bid FROM branches_services
WHERE branches.bid = branches_services.bid GROUP BY bid HAVING COUNT(sid) >= 4);
```

下面的连接，以一种更优化的方式产生相同的结果：

```
SELECT branches.bid, branches.bdesc FROM branches_services, branches WHERE
branches_services.bid = branches.bid GROUP BY bid HAVING COUNT(sid) >= 4;
```

11.2.2 子查询和FROM子句

还可以在一个SELECT语句的FROM子句中把子查询产生的结果作为一个表。为了说明这一点，假设我们想要知道每个分支机构使用的服务的平均数。因此，我们要做的第一件事是对分支分组，然后计算出每个部门的服务总数。如：

```
mysql> SELECT bid, COUNT(sid) AS stotal FROM branches_services GROUP BY bid;
+------+--------+
| bid  | stotal |
+------+--------+
| 1011 |      4 |
| 1012 |      1 |
| 1013 |      1 |
| 1031 |      3 |
| 1032 |      1 |
| 1033 |      1 |
| 1041 |      2 |
| 1042 |      2 |
| 1101 |      1 |
+------+--------+
9 rows in set (0.17 sec)
```

然后计算每个分支服务的平均数。可用子查询：

```
mysql> SELECT AVG(z.stotal) FROM (SELECT bid, COUNT(sid) AS stotal FROM
branches_services GROUP BY bid) AS z;

+---------------+
| avg(z.stotal) |
+---------------+
|        1.7778 |
+---------------+
1 row in set (0.21 sec)
```

因而，内查询产生的结果表用在外查询的FROM子句中。这样的表在SQL中称为导出表。

请注意，当以这种方式使用子查询的结果时，内查询产生的结果表必须首先另取一个表名，否则MySQL不知道如何参照其中的列。如果在没有为表另取名的情况下重新运行前面的查询，看会发生什么：

```
mysql> SELECT AVG(stotal) FROM (SELECT bid, COUNT(bid) AS stotal FROM
branches_services GROUP BY bid);
ERROR 1246: Every derived table must have its own alias
```

现在更进一步。如果需要列出服务数高于平均数的分支，该怎么办？

```
mysql> SELECT bid FROM branches_services GROUP BY bid HAVING COUNT(sid) > 1.7778;
+------+
| bid  |
+------+
| 1011 |
| 1031 |
| 1041 |
| 1042 |
+------+
4 rows in set (0.28 sec)
```

为了使它成为动态的查询，需要把前面的两个查询组合成下面的复杂查询：

```
mysql> SELECT bid FROM branches_services GROUP BY bid HAVING COUNT(sid)
> (SELECT AVG(z.stotal) FROM (SELECT bid, COUNT(bid) AS stotal FROM
branches_services GROUP BY bid) AS z);
+------+
| bid  |
+------+
| 1011 |
| 1031 |
| 1041 |
| 1042 |
+------+
4 rows in set (0.28 sec)
```

向它添加另一个查询和一个连接，可以获得和那些分支相应的顾客名字：

```
mysql> SELECT DISTINCT cname FROM clients, branches, (SELECT bid FROM
branches_services GROUP BY bid HAVING COUNT(sid) > (SELECT AVG(z.stotal)
FROM (SELECT bid, COUNT(bid) AS stotal FROM branches_services GROUP BY bid)
AS z)) AS x WHERE clients.cid = branches.cid AND branches.bid = x.bid;
+------------------+
| cname            |
+------------------+
| JV Real Estate   |
| DMW Trading      |
| Rabbit Foods Inc |
+------------------+
3 rows in set (0.33 sec)
```

11.2.3 子查询和连接

像本章前面的例子所阐述的那样，子查询和连接同时存在，为什么要限制自己只使用内连接呢？我们可以使用任意一个连接与子查询结合，有创意的想法是我们可以协同共用两者来执行一些复杂的查询。

例如，我们可以结合一个左连接得到 branches 表中没有登记的所有客户的列表：

```
mysql> SELECT cname FROM clients LEFT JOIN branches ON clients.cid =
branches.cid WHERE branches.bid IS NULL;
+-------------------+
| cname             |
+-------------------+
| ABC Talent Agency |
+-------------------+
1 row in set (0.17 sec)
```

在这个例子中，查询将保留连接左边的所有记录（客户表），并对右边的每一条记录插入 NULL（对不符合连接条件的分支）。最后的结果如何？所有没有分支机构的客户将用NULL值标记（然后可以使用 IS NULL 测试进行分离）。

另外一种方法是使用具有子查询的右连接：

```
mysql> SELECT cname FROM clients WHERE cid = (SELECT clients.cid FROM branches
RIGHT JOIN clients ON clients.cid = branches.cid WHERE branches.bid IS NULL);
+-------------------+
| cname             |
+-------------------+
| ABC Talent Agency |
+-------------------+
1 row in set (0.22 sec)
```

在这个例子中，连接的右边的所有记录（客户）将被保留，没有分支机构的客户记录将标记为 NULL。子查询将返回相应客户的 ID，然后它可以映射到 clients 表，得到客户的名字。

有趣的是，在大多数情况下，子查询可以写成连接，反之也可以。考虑下面的例子，它通过子查询列出了使用 Recruitment 服务的分支部门：

```
mysql> SELECT bid FROM branches_services WHERE sid = (SELECT sid FROM
services WHERE sname = 'Recruitment');
+------+
| bid  |
+------+
| 1011 |
| 1031 |
+------+
2 rows in set (0.22 sec)
```

下面的例子使用连接完成了同样的事情：

```
mysql> SELECT bs.bid FROM branches_services AS bs, services AS s WHERE
s.sid = bs.sid AND s.sname = 'Recruitment';
```

```
+------+
| bid  |
+------+
| 1011 |
| 1031 |
+------+
2 rows in set (0.17 sec)
```

也可以把子查询写成自连接，反之也可以。考虑下面的例子，它使用一个子查询列出了同一个地方的所有分支机构，并把它们作为 JV Real Estate 的公司总部：

```
mysql> SELECT bid, bdesc, bloc FROM branches WHERE bloc = (SELECT bloc
FROM branches WHERE bid = 1101 AND cid = 110);
+------+----------------------+------+
| bid  | bdesc                | bloc |
+------+----------------------+------+
| 1011 | Corporate HQ         | CA   |
| 1042 | Branch Office (West) | CA   |
| 1101 | Head Office          | CA   |
+------+----------------------+------+
3 rows in set (0.22 sec)
```

下面的语句是等价的，它使用一个自连接完成了同样的事情：

```
mysql> SELECT table1.bid, table1.bdesc, table1.bloc FROM branches AS
table1, branches AS table2 WHERE table2.bid = 1101 AND table2.cid = 110
AND table1.bloc = table2.bloc;
+------+----------------------+------+
| bid  | bdesc                | bloc |
+------+----------------------+------+
| 1011 | Corporate HQ         | CA   |
| 1042 | Branch Office (West) | CA   |
| 1101 | Head Office          | CA   |
+------+----------------------+------+
3 rows in set (0.16 sec)
```

11.2.4 子查询和其他 DML 语句

到目前为止，我们看到的例子只是在 SELECT 语句中使用子查询。但是，子查询也可以用来约束 UPDATE 和 DELETE 语句。

一个简单的例子可以解释这些问题。假设我们想要删除所有使用 Recruitment 服务（service ID 为 2）的分支。一般情况下，首先要在 branches_services 表中查找使用该项服务的分支：

```
mysql> SELECT * FROM branches_services WHERE sid = 2;
+------+-----+
| bid  | sid |
+------+-----+
| 1011 |   2 |
| 1031 |   2 |
+------+-----+
2 rows in set (0.16 sec)
```

然后从 branches 表中删除相应的分支记录：

```
mysql> DELETE FROM branches WHERE bid = 1011;
Query OK, 1 row affected (0.00 sec)

mysql> DELETE FROM branches WHERE bid = 1031;
Query OK, 1 row affected (0.00 sec)
```

也可以使用一个子查询把前面的两个操作结合成单独的一个，如下所示：

```
mysql> DELETE FROM branches WHERE bid IN (SELECT bid FROM
branches_services WHERE sid = 2);
Query OK, 2 rows affected (0.00 sec)
```

去除查询弊端

MySQL 在优化连接方面要比子查询做得好，所以如果我们发现在 MySQL 服务器上平均负载达到了不能接受的水平，应检查应用代码，然后把子查询重新写成连接或者连接序列。

如何删除所有分支机构中产生的服务费用等于或低于 500 的顾客？

```
mysql> DELETE FROM clients WHERE cid IN (SELECT DISTINCT b.cid FROM
branches AS b, branches_services AS bs, services AS s WHERE b.bid =
bs.bid AND bs.sid = s.sid GROUP BY bs.bid HAVING SUM(sfee) <= 500);
Query OK, 1 row affected (0.28 sec)
```

在这个例子中，内查询根据分支ID对不同的分支分组，根据每个分支使用的所有服务计算产生的服务总收入，然后列出总收入小于或者等于500的记录。最后外查询使用相应的客户ID执行在 clients 表上的 DELETE 操作。

可以在UPDATE语句中以相同的方式使用子查询。假设想要找到在三个或者更多个分支机构中使用的服务：

```
mysql> SELECT sid FROM branches_services GROUP BY sid HAVING COUNT(bid) >= 3;
+-----+
| sid |
+-----+
|   1 |
|   3 |
|   4 |
+-----+
3 rows in set (0.11 sec)
```

然后为那些服务增长 25% 的费用使用子查询：

```
mysql> UPDATE services SET sfee = sfee + (sfee * 0.25) WHERE sid = 1;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE services SET sfee = sfee + (sfee * 0.25) WHERE sid = 3;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE services SET sfee = sfee + (sfee * 0.25) WHERE sid = 4;
```

```
Query OK, 1 row affected (0.05 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

可以把前面的操作结合到下面的语句中：

```
mysql> UPDATE services SET sfee = sfee + (sfee * 0.25) WHERE sid IN
(SELECT sid FROM branches_services GROUP BY sid HAVING COUNT(bid) >= 3);
Query OK, 3 rows affected (0.22 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

另外一个例子。假设想要让所有位于加利福尼亚的所有分支使用Security服务而不是Administration服务。使用子查询，这件事就非常容易了：

```
mysql> UPDATE branches_services SET sid = 6 WHERE sid = 4 AND bid IN
(SELECT bid FROM branches WHERE bloc = 'CA');
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

在这个例子中，内查询只考虑那些加利福尼亚的分支ID，然后把这个列表提供给外查询，外查询更新branches_services表中的相应记录。请注意如何划分每个需要更新的行的选择标准：内查询列出了加利福尼亚的所有记录，外查询过滤出只使用Administration服务的记录。

如果想要更深入一步，向不同的SET子句添加子查询（这是非常完美的SQL），它和WHERE子句的子查询以相同的方式工作。如：

```
mysql> UPDATE branches_services SET sid = (SELECT sid FROM services WHERE
sname =  'Security') WHERE sid = (SELECT sid FROM services WHERE sname =
'Administration') AND bid IN (SELECT bid FROM branches WHERE bloc = 'CA');
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

UPDATE和DELETE语句的循环引用

现在，我们假设前面讨论的系统是同步的，branches表中的每个客户必须至少属于一个分支机构。最快的验证方法是使用一个左连接，而且对每个NULL值进行仔细检查。如：

```
mysql> SELECT clients.cid, clients.cname, branches.bid, branches.bdesc FROM
clients LEFT JOIN branches USING (cid);
+-----+----------------------------+------+-------------------------------+
| cid | cname                      | bid  | bdesc                         |
+-----+----------------------------+------+-------------------------------+
| 101 | JV Real Estate             | 1011 | Corporate HQ                  |
| 101 | JV Real Estate             | 1012 | Accounting Department         |
| 101 | JV Real Estate             | 1013 | Customer Grievances Department |
| 102 | ABC Talent Agency          | NULL | NULL                          |
| 103 | DMW Trading                | 1031 | N Region HO                   |
| 103 | DMW Trading                | 1032 | NE Region HO                  |
| 103 | DMW Trading                | 1033 | NW Region HO                  |
| 104 | Rabbit Foods Inc           | 1041 | Branch Office (East)          |
| 104 | Rabbit Foods Inc           | 1042 | Branch Office (West)          |
| 110 | Sharp Eyes Detective Agency | 1101 | Head Office                   |
+-----+----------------------------+------+-------------------------------+
10 rows in set (0.00 sec)
```

非常明显，系统不处在一致的状态——对于顾客 ABC Talent Agency 有一条记录，但是这个顾客在一个或者多个分支机构中没有相应的记录。为了让系统处于一致的状态，必须通过顾客 ID 从 clients 表中删除错误的记录。如：

```
mysql> DELETE FROM clients WHERE cid = 102;
```

根据前面的例子可以想到，前面的两步可以使用下面的子查询组合成一步：

```
mysql> DELETE FROM clients WHERE cid = (SELECT clients.cid FROM clients LEFT
JOIN branches USING (cid) WHERE bid IS NULL);
ERROR 1093: You can't specify target table 'clients' for update in FROM clause
```

但是这是错误的。

MySQL 不允许在使用子查询读数据的同时删除或者更新相同的表数据，如果这样做，就有可能使子查询查找的行已经被删除或者改变了。因此，在外部 DELETE 或者 UPDATE DML 语句中命名的表不能在内查询的 FROM 子句中出现（这就是前面的错误信息，MySQL 很简洁地进行了叙述）。

执行前面的任务的一个更合适的方法是使用具有外部参照的 EXISTS 测试，如下所示：

```
mysql> DELETE FROM clients WHERE NOT EXISTS (SELECT * FROM branches WHERE
branches.cid = clients.cid);
Query OK, 1 row affected (0.11 sec)
```

11.3 小结

子查询是嵌套的 SELECT 查询，SELECT 查询的结果作为包含在它外面的查询的条件。子查询可以出现在表达式的任何地方，也可以用在很多地方，包括 WHERE 或者 HAVING 子句、比较或者逻辑运算符、IN 成员测试、EXISTS 运算符、外部查询的 FROM 子句中以及其他像 UPDATE 和 DELETE 这样的 DML 命令。

但是，这种方便也是要付出代价的。不正确的书写子查询可能导致 RDBMS 的负荷过大，大幅度地降低性能，破坏了这种特性提供的所有好处。所以强烈建议使用其他方法来获得需要的数据，包括连接、联合以及其他 SQL 结构来保证应用程序的最佳性能和 RDBMS 的最小资源浪费。

第12章 事务处理

通常情况下，每个MySQL查询的执行是互相独立的，不必考虑哪个查询在前，哪个在后。例如，一系列的INSERT或者UPDATE语句被顺序地执行，不必考虑任何一个查询是否失败或者产生错误。这是因为MySQL认为每个查询是一个自我包含的单元，与在它之前或者之后的查询没有任何关系。

大多数情况下，这种无状态的方法能够很好地工作，特别是对于与简单业务逻辑相关的小型和中型的应用程序。但是在比较复杂的情况下，通过一组SQL语句执行的行为或者必须全部成功，或者必须全部撤销，这时这种方法就有些不足了。在这种情况下，不仅序列中的动作互相依靠（因而不可能完全独立地执行），而且序列中某个任务的失败意味着需要中断整个动作序列，同时撤销在同一个序列中前面的动作做出的变化，返回到数据库早期的状态。

因为无状态的方式独立地对待每个查询，没有关于它的祖先的知识或假设，它不能恰当地处理类似于或者全有或者全无的情况。因此设计了另外一种模型来满足这种需求，能够把一系列的SQL语句组合一个单独的单元或者事务。

商业上的关系型数据库管理系统（RDBMS），比如Oracle和Microsoft SQL Server，它们支持事务模型，就像PostgreSQL一样拥有开放资源。直到最近，MySQL仍然不具有这个特性。但是在MySQL 3.23版本中已经开始了最原始的实现，在4.x版本中有了更先进的结构，今天，MySQL包含的表处理程序以与其他商业RDBMS产品相同的方式管理事务。

本章深入讨论MySQL的事务模型，解释什么是事务模型，它是如何工作的，它如何帮助创建一些更强健的应用程序。本章还使用另一种方法来研究MySQL的事务模型，解释它如何通过使用MySQL的表锁定和旧的非事务表类型来完成类似的功能。

12.1 什么是事务

在MySQL的环境中，一个事务由作为一个单独单元的一个或者多个SQL语句组成。这个单元中的每个SQL语句是互相依赖的，而且单元作为一个整体是不可分割的。如果单元中的一个语句不能成功地完成，整个单元就会回滚，所有影响到的数据将返回到事务开始以前的状态。因而，只有事务中的所有语句都被成功地执行才能说这个事务被成功地执行。

我们可能会发现很难找到这种个体与整体同进退的情况，但在在现实中，事务充满在我们周围——银行交易，股票交易，网上购物，库存品控制，到处都有这种情况发生。在所有这些例子中，事务的成功取决于这些相互依赖的行为是否能够被成功地执行，是否互相协调。其中的任何一个失败必须取消整个事务，系统返回到事务处理以前的状态。

理解这件事的最好方法是使用一个简单的例子。考虑股票市场中的一次交易（见图12.1），交易者A卖出400股ACME公司的股票给交易者B。

在喧嚣的交易场所后面是一个复杂的数据库系统用来跟踪所有的交易。在这个系统中，像前面这样的交易只有当交易者A的账户卖出400股ACME公司的股票，同时交易者B的账户入

账这些股票，交易才被认为是完成了。如果前面两步的任何一步失败，那么将会陷入尴尬境地——ACME公司的400股股票将没有持有者。这会很不愉快，肯定大家会同意这种观点。

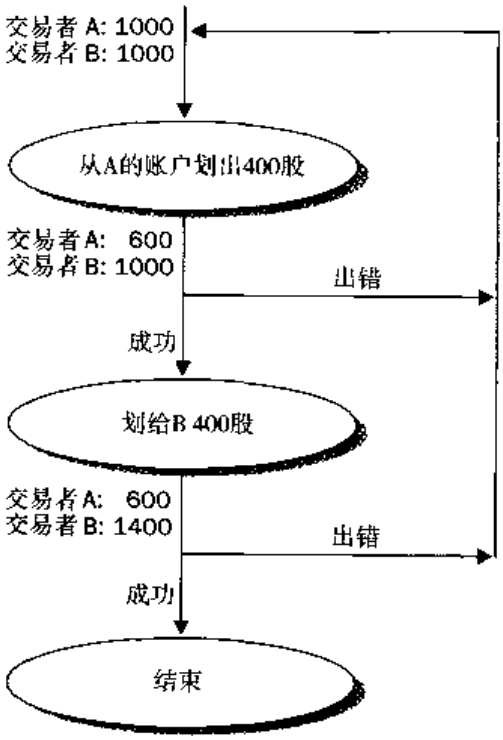

图 12.1 股票交易事务

因而，前面例子中的 ACME 公司的 400 股股票从交易者 A 到交易者 B 的转移被认为是一个事务——几个SQL语句组成的一个单独的工作单元（从交易者A的账户记录删除400股，向交易者B的记录添加400股，对两个交易者执行计算，然后保存变化）。如同前面的事务定义所说，所有这些语句都应该成功地执行。如果其中任何一个失败，交易就被撤销，所以系统返回到早期不变的状态。也就是说，谁是400股股票的持有者，在任何时候都不能是模棱两可的。

另外一个例子：向公司添加一个新的雇员（见图12.2）。这里的过程由三个基本步骤组成：在雇员数据库中为每个雇员创建一条记录，为雇员分配部门，建立雇员的工资记录。如果这三步中的任何一步失败，例如为新成员分配的雇员ID已经被其他人使用或者输入到工资系统中的值太大，系统就必须撤销在失败之前所有的变化，删除所有不完整记录的踪迹，避免以后的不一致和计算失误。

前面的三项任务构成了一个事务。任何一个任务的失败都会导致整个事务被撤销，系统返回到以前的状态。

12.1.1 事务和ACID属性

通过InnoDB和BDB表类型，MySQL事务系统能够完全满足事务安全的ACID测试。旧的表类型，比如MyISAM类型，不支持事务。因此，这种系统中的事务只能通过直接的表锁定实现（在12.5.1节可以得到更详细的说明）。

术语“ACID”是一个简称，指出每个事务型RDBMS必须遵循的4个属性。为了验证是否满足ACID，一个RDBMS必须具有下面的特征：原子性、一致性、孤立性和持久性。在下面的部分将详细讨论它们。

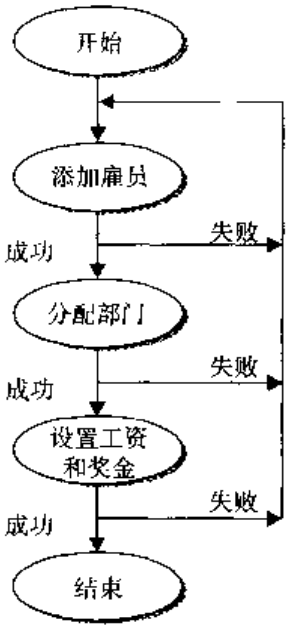

图 12.2 涉及多个互相依赖行为的事务

原子性

原子性意味着每个事务都必须被认为是一个不可分割的单元。假设一个事务由两个或者多个任务组成，其中的语句必须同时成功才能认为事务是成功的。如果事务失败，系统将会返回到事务以前的状态。

参照前面股票交易的例子，原子性意味着交易者A卖出股票和交易者B购买同样的股票不能单独发生，两个事务必须同时发生才被认为是完整的。类似地，在添加雇员这个例子中，原子性就是指如果没有创建雇员相应的工资表和部门记录，就不可能向雇员数据库添加雇员。

原子的执行是一个或者全部发生或者什么也没有发生的命题。在一个原子操作中，如果事务中的任何一个语句失败，前面执行的语句都将被返回，以保证数据的整体性没有受到影响。这在一些关键系统中尤其重要，现实世界的应用程序（比如金融系统）执行数据输入或者更新，必须保证不出现数据丢失或数据错误，以保证数据安全性。

一致性

不管事务是完全成功完成还是中途失败，当事务使系统处于一致的状态时存在一致性。参照前面的股票交易例子，一致性是指卖方账户结果的每次减少和买方账户的增加处于相等的状态。如果一个事务从交易者A的账户上减去了400股，但是只在交易者B的账户上增加了300股，这就违背了一致性约束，因为系统中股份的总数发生了变化。类似地，一致性属性要保证如果从系统中删除了一个雇员，则所有和该雇员相关的数据，包括工资数据和组的成员资格也要被删除。

在MySQL中，一致性主要由MySQL的日志机制处理，它记录了数据库的所有变化，为事务恢复提供了跟踪记录。如果系统在事务处理中间发生错误，MySQL恢复过程将使用这些日志来发现事务是否已经完全成功地执行，是否需要返回。因而一致性属性保证了数据库从不返回一个未处理完的事务。

孤立性

孤立性是指每个事务在它自己的空间发生，和其他发生在系统中的事务隔离，而且事务的结果只有在它完全被执行时才能看到。即使在这样的一个系统中同时发生了多个事务，孤立性原则保证某个特定事务在完全完成之前，其结果是看不见的。例如，在前面的股票交易例子中，孤立性是指两个交易者之间的事务和交易中的其他事务是不互相影响的，而且它的结果只有在被完成时才能显示给公众。

当系统支持多个同时存在的用户和连接时（如MySQL），这就尤其重要。如果系统不遵循这个基本规则，就可能导致大量数据的破坏，比如每个事务的各自空间的完整性很快地被其他冲突事务所侵犯。

当然，在现实世界中，获得绝对孤立性的惟一方法是保证在任意时刻只能有一个用户访问数据库。当处理像MySQL这样多用户的RDBMS时，这不是一个实际的解决方法。但是，大多数事务系统使用页级锁定或者行级锁定来孤立不同事务之间的变化，这是要以降低性能为代价的。例如，MySQL的BDB表处理程序使用页级锁定来保证处理多个同时发生的事务的安全，InnoDB表处理程序使用更好的行级锁定（请参阅下面的补充资料中关于更多不同表类型使用的锁定级的信息）。

持久性

持久性是指即使系统崩溃，一个提交的事务仍在坚持。当一个事务完成，数据库的日志已经被更新时，持久性就开始发生作用。大多数RDBMS产品通过保存所有行为的日志来保证数据的持久性，这些行为是指在数据库中以任何方法更改数据。数据库日志记录了所有对于表的更新、查询、报表等。

如果系统崩溃或者数据存储介质被破坏，通过使用日志，系统能够恢复在重启前进行的最后一次成功的更新，反映了在崩溃时处于执行过程的事务的变化。在前面股票转移的例子中，持久性意味着一旦股份从交易者A成功地转移给交易者B，即使系统不断地崩溃，也能反映那时的状态。

MySQL通过保存一条记录事务过程中系统变化的二进制事务日志文件来实现持久性。如果遇到硬件破坏或者突然的系统关机，在系统重启时，通过使用最后的备份和日志就可以很容易地恢复丢失的数据。

默认情况下，InnoDB表是100%持久的（也就是说，所有在崩溃前系统所进行的事务在恢复过程中都可以可靠地恢复）。MyISAM表提供部分持久性，所有在最后一个FLUSH TABLES命令前进行的变化都能保证被存盘。

锁　定

MySQL支持很多不同的表类型，而且对于不同的类型，锁定机制也是不同的。因此，理解不同级的锁定是使用MySQL的非事务表实现伪事务环境的基本条件。

- **表锁定**　一个特殊类型的访问，整个表被客户锁定。根据锁定的类型，其他客户不能向表中插入记录，甚至从中读数据也受到限制。
- **页锁定**　MySQL将锁定表中的某些行（称做页）。被锁定的行只对锁定最初的线程是可行的。如果另外一个线程想要向这些行写数据，它必须等到锁被释放。不过，其他页的行仍然可以使用。

- **行锁定** 行级的锁定比表级锁定或者页级锁定对锁定过程提供了更精细的控制。在这种情况下，只有线程使用的行是被锁定的。表中的其他行对于其他线程都是可用的。在多用户的环境中，页层的锁定降低了线程间的冲突，可以使多个用户同时从一个相同的表读数据甚至写数据。不过，必须平衡这种灵活性，实际上它还有这三层锁定的最高性能开销。

MyISAM表类型只支持表级的锁定，当涉及到大量的读操作而不是写操作时，它提供了比行级和页级锁定更好的性能。BDB表类型支持页级锁定，InnoDB表类型在事务中自动执行行级的锁定。

我们可以在MySQL手册中看到更多锁定的方法，可查看http://www.mysql.com/doc/en/Locking_methods.html。

12.2 事务的生命周期

现在已经知道了什么是事务，让我们再看看事务如何工作。MySQL有许多与开始、结束和返回事务相关的命令。这一节将详细地介绍它们。

请注意，MySQL的InnoDB和BDB表类型生来就支持事务，也就是说，下面的命令只适用于这两种表类型。MySQL中新表的默认类型是MyISAM，但是可以向CREATE TABLE命令添加可选的TYPE子句，告诉MySQL我们需要InnoDB表。如下所示：

```
CREATE TABLE table-name (field-definitions) TYPE = INNODB;
```

对于已存在的表，我们可以通过对ALTER TABLE命令指定一个TYPE子句来改变表类型。如下所示：

```
ALTER TABLE table-name TYPE = INNODB;
```

防患于未然

ALTER TABLE命令先备份表中的数据，删除它，使用指定的要修改的名字重建它，然后重新插入备份的记录。任一步骤的失败将会导致数据的丢失或者破坏。因此，一个好的方法是在使用ALTER TABLE命令之前，先创建一个表备份。请参阅第8章关于ALTER TABLE命令的更多信息。

为了初始化一个事务，并告诉MySQL所有随后的SQL语句需要被认为是一个单元，MySQL提供了START TRANSACTION命令来标记一个事务的开始。如：

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

也可以使用BEGIN或者BEGIN WORK命令来初始化一个事务。通常情况下，START TRANSACTION命令后面跟随的是组成事务的SQL语句。假设这里的事务是要向一个系统添加一个新雇员，涉及到的步骤包括为每个雇员创建一条记录，为雇员分配部门，指定每个雇员的工资和奖金。如：

```
mysql> INSERT INTO employees (ename, enationality)
VALUES ('Tim Jonez', 'US');
Query OK, 1 row affected (0.13 sec)
```

```
mysql> INSERT INTO departments (eid, dept)
VALUES (LAST_INSERT_ID(), 'Accounts');
Query OK, 1 row affected (0.03 sec)

mysql> INSERT INTO payroll (eid, pbasic, pbonus)
VALUES (LAST_INSERT_ID(), 36000.00, 2500.00);
Query OK, 1 row affected (0.02 sec)
```

现在查看这些表，通过 SELECT 查询看一看数据是否被正确地输入：

```
mysql> SELECT * FROM employees;
+-----+-----------+--------------+
| eid | ename     | enationality |
+-----+-----------+--------------+
|   1 | Tim Jonez | US           |
+-----+-----------+--------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM payroll;
+-----+----------+---------+
| eid | pbasic   | pbonus  |
+-----+----------+---------+
|   1 | 36000.00 | 2500.00 |
+-----+----------+---------+
1 row in set (0.05 sec)
```

一旦 SQL 语句被执行，就可以使用 COMMIT 命令把整个事务保存在磁盘上，或者使用 ROLLBACK 命令撤销所有的变化。我们首先尝试第二种选择：

```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.02 sec)
```

如何证明这些变化已经被撤销呢？

```
mysql> SELECT * FROM employees;
Empty set (0.01 sec)

mysql> SELECT * FROM departments;
Empty set (0.00 sec)

mysql> SELECT * FROM payroll;
Empty set (0.00 sec)
```

如果事务包括事务表和非事务表的变化，非事物表的事务处理部分是不能使用ROLLBACK命令撤销的。在这种情况下，MySQL 将返回一个错误，通知出现一个不完全撤销，如下所示：

```
mysql> ROLLBACK;
ERROR 1196: Some non-transactional changed tables couldn't be rolled back
```

现在再一次执行这个事务，这次要注意如何保存它。如：

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

mysql> INSERT INTO employees (ename, enationality) VALUES ('Tim Jonez', 'US');
```

```
Query OK, 1 row affected (0.13 sec)

mysql> INSERT INTO departments (eid, dept) VALUES (LAST_INSERT_ID(), 'Accounts');
Query OK, 1 row affected (0.03 sec)

mysql> INSERT INTO payroll (eid, pbasic, pbonus) VALUES (LAST_INSERT_ID(),
36000.00, 2500.00);
Query OK, 1 row affected (0.02 sec)
```

一个有趣的实验是打开和服务器相连的另一个客户机，检查前面的SQL查询是否导致数据库的变化。如：

```
mysql> SELECT * FROM employees;
Empty set (0.06 sec)

mysql> SELECT * FROM departments;
Empty set (0.03 sec)

mysql> SELECT * FROM payroll;
Empty set (0.04 sec)
```

如果想知道在一个客户机会话中插入的记录为什么不在另一个里面显示，不要过份担心——实际上，这是一个孤立的例子。在前面部分曾讲到，孤立性意味着只有整个事务被成功地执行时，事务的结果才是可见的。因为事务仍在处理的过程中，还没有被保存在磁盘上，对于同一个数据库的其他用户是不可见的（如果需要事务之间的可见性，可以通过设置不同的孤立级得到，在12.3.2节将详细讨论）。

一旦数据被正确地插入，这时可以使用COMMIT命令保存各种表的变化：

```
mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

COMMIT命令标记了事务块的结束。一旦事务交给了数据库，提交的数据对其他客户会话是可见的。

图12.3使用一个简单的流程图总结了一个事务的生命周期。

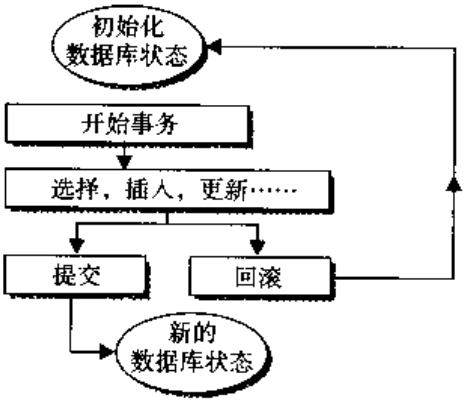

图12.3 事务生命周期

注意MySQL使用了一个平面事务模型：嵌套的事务是不允许的，在第一个里面使用START TRANSACTION或者BEGIN后，当第二个事务开始时，自动地提交第一个事务。同样，许多其他的MySQL命令运行时都会隐藏执行一个COMMIT命令，下面是一个简要的列表：

- DROP DATABASE/DROP TABLE
- CREATE INDEX/DROP INDEX
- ALTER TABLE/RENAME TABLE
- LOCK TABLES/UNLOCK TABLES
- SET AUTOCOMMIT = 1

循序渐进

MySQL 比较新的版本支持 savepoints（保存点），它可以使我们在一个事务中在关键的地方做标记，如果需要的话，可以部分地撤销。当处理一个长的事务时，这就变得很方便——如果发生错误，一个保存点可以让我们只撤销产生错误的部分，而不是整个事务。

保存点仍然在发展中，所以我们需要参考 MySQL 手册来获得关于这个特性的最新消息。

12.3 控制事务行为

MySQL 提供了两个变量来控制事务行为——AUTOCOMMIT 变量和 TRANSACTION ISOLATION LEVEL 变量。下面的部分详细地介绍这两个变量。

12.3.1 自动提交

默认情况下，MySQL 的 SQL 查询一旦被执行，就会自动向数据库提交结果。这就涉及到自动提交模式，也是我们不需要使用 START TRANSACTION 语句开始 MySQL 会话或者使用一个 COMMIT 或者 ROLLBACK 结束的原因。也就是说，MySQL 把每个查询作为一个单语句事务来对待。

这个默认的行为可以通过特定的 AUTOCOMMIT 变量进行修改，这个变量控制了 MySQL 的自动提交模式。下面的片段通过在每个 SQL 交互后提交一个 COMMIT 命令关掉 MySQL 行为：

```
mysql> SET AUTOCOMMIT = 0;
Query OK, 0 rows affected (0.02 sec)
```

随后，表的更新将不会被保存，直到明确地发出了一个 COMMIT 命令。实际上，如果我们没有发布 COMMIT，就终止了 MySQL 会话，数据库将会自动执行 ROLLBACK 撤销会话中所有的变化，因而否定了会话期间的所有工作。下面的例子进行了说明：

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

mysql> SELECT * FROM gallery;
Empty set (0.00 sec)

mysql> INSERT INTO gallery (id, filename, dsc)
VALUES (52, 'dog.gif', 'Me and my pooch');
Query OK, 1 row affected (0.00 sec)

mysql> exit
Bye
```

速度碰撞

在自动提交模式开启的情况下，在一个连接上执行许多小的更新会经常导致性能变差。

现在开始一个新的会话：

```
[user@host] $ mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14 to server version: 4.0.12-max-debug

mysql> SELECT * FROM gallery;
Empty set (0.01 sec)
```

通过重新设置AUTOCOMMIT变量到它的初始状态，来返回到最初的自动提交模式：

```
mysql> SET AUTOCOMMIT = 1;
Query OK, 0 rows affected (0.00 sec)
```

当转换自动提交模式到开的状态时，MySQL将会自动发出一个COMMIT，同时保存所有开放的事务。

可以使用SELECT随时获得AUTOCOMMIT变量的当前值。如下所示：

```
mysql> SELECT @@autocommit;
+--------------+
| @@autocommit |
+--------------+
|            0 |
+--------------+
1 row in set (0.00 sec)
```

AUTOCOMMIT变量是一个会话变量，当一个新的客户会话开始的时候，将默认为1，没有方法把它在新的客户会话开始时的默认值设为0。可以参考第13章，得到关于修改和浏览服务器变量的更多信息。

错误类型

AUTOCOMMIT变量只影响像InnoDB这样的事务表类型。当处理像MyISAM这样的非事务表类型时，AUTOCOMMIT变量不起作用，这些表的变化立即就被保存。

12.3.2 事务孤立级

前面曾提到，事务型RDBMS的一个最重要的属性之一就是它可以“隔离”在服务器上正在处理的不同的会话。在单用户的环境中，这个属性无关紧要：没有东西用来隔离，因为在任意时刻只有一个会话处于活动状态。不过，在复杂的现实环境中，这种假设的情况是不存在的。

在多用户环境中，许多RDBMS会话在任一给定时刻都是活动的。例如，在前面讨论的股票交易例子中，不可能在某一指定时间只发生一项交易事务。大多数是同时发生成百上千个交易。在这种情况下，RDBMS能够孤立事务是很重要的，这样它们不互相影响，同时保证数据库性能不受到影响。

为了了解孤立的重要性，有必要花些时间来考虑如果不强加孤立会发生什么。如果没有事务的孤立性，不同的SELECT语句将会在同一个事务的环境中检索到不同的结果，因为在这个期间，基本的数据已经被其他事务所修改。这将导致不一致性，同时很难相信结果集，从而不

能利用查询结果作为计算的基础。因而孤立性强制对事务进行某种程度的隔离，保证应用程序在事务中看到一致的数据。

基于 ANSI/ISO SQL 规范，MySQL 提供了下面 4 种孤立级：

- SERIALIZABLE（序列化）
- REPEATABLE READ（可重复读）
- READ COMMITTED（提交的读）
- READ UNCOMMITTED（未提交的读）

这些事务孤立级决定了其他事务能“看到”正在进行的事务的程度，同时以分层顺序安排，先进行最安全的，然后逐渐向下到最低安全级（最易出现问题）的。这些孤立级可以通过 TRANSACTION ISOLATION LEVEL 变量来操作，这个变量在 12.3.3 节详细讨论。

现在看看每个孤立级是做什么的。

序列化

SERIALIZABLE孤立级通过一个接一个顺序地执行当前的事务提供了事务之间最大程度的隔离，图 12.4 说明了这一点。

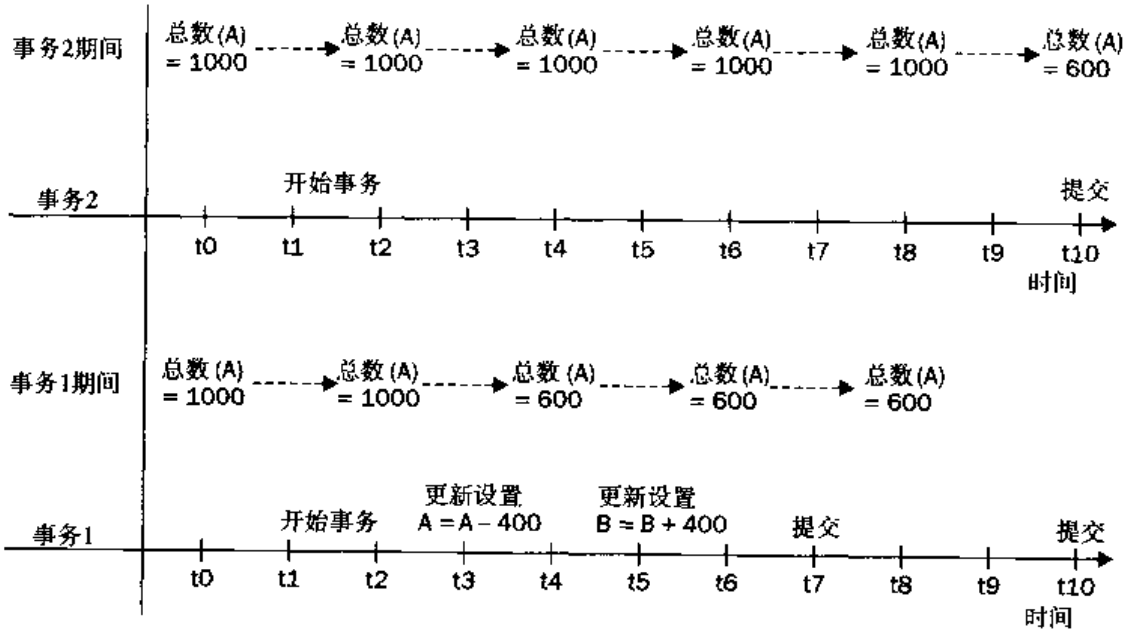

图 12.4 SERIALIZABLE 孤立级

图中表示交易者 A 卖出 400 股给交易者 B。要执行这个事务，系统必须从交易者 A 的账户上减去 400 股，然后把这 400 股加到交易者 B 的账户上。假设两个交易者在交易的最初都有等值的 1000 股，一个成功的交易将使交易者 A 的账户持有 600 股，交易者 B 的账户持有 1400 股。

在这个例子中，因为交易的孤立级是最大的，第二个交易只能在交易完成以后看到交易者 A 和 B 账户上的变化。在交易的过程中更新已存在的行或者添加新行都是不可见的。实际上，这种方法是大多数用户期待事务进行工作的方法。不过，它是要付出代价的：如果每个事务都以这种孤立级运行就会影响 MySQL 的性能，因为需要使用大量的资源来使大量事务在任一时刻不被看到。

可重复读

对于应用程序，如果希望稍微在安全上做一些妥协来获得更好的性能，MySQL提供了REPEATABLE READ孤立级。在这一级上，事务不会被看成是一个序列。不过，当前在执行事务的变化仍然不能看到，直到得到结果为止。

这看起来和序列化孤立级很相似。但是，它有一个很重要的不同点：虽然一个事务不能看到其他正在执行的当前事务做出的变化，但是它可以看到新的记录（其他事务添加到数据库的）。这有时被称做虚拟读或者虚拟插入，因为只在事务通过INSERT语句向数据库插入一个新记录时它才发生。当执行一个SELECT查询时，其他同一时间正在进行的事务将会突然看到这个新的虚幻的行。如果事务对于表的记录执行计算或者使用总和函数时就会导致问题——一个新行的突然出现将会产生意料不到的、不一致的结果。

图 12.5 显示了虚构的读操作是什么样子的。

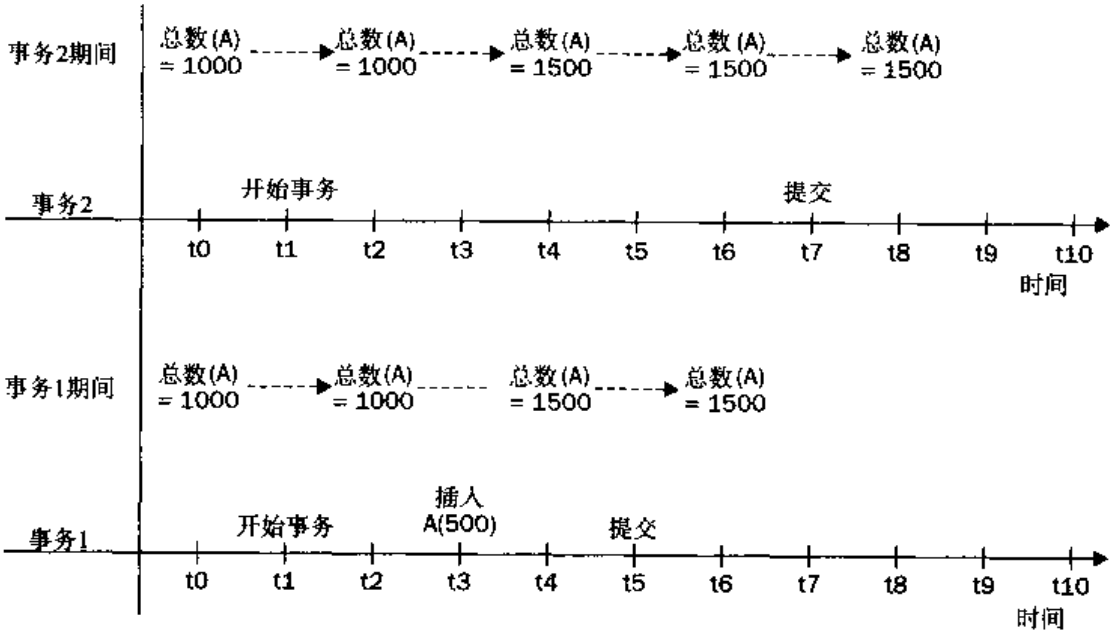

图 12.5　REPEATED READ 孤立级和虚构的读操作

在这个例子中，第二个事务可以看到第一个事务添加的新行。这就导致第二个事务在进行中可以看到的数据突然变化：在时间点 t2 和 t4 之间，交易者 A 的账户的股票总数多了 500 股，即使事务最初的变化没有被提交。这种变化是由于事务过程中新的虚幻行的突然出现产生的。

提交的读

READ COMMITTED孤立级的安全性比REPEATABLE READ孤立级的安全性要差。不仅处于这一级的事务可以看到其他事务添加的新记录，而且其他事务对现存记录做出的修改一旦被提交，也可以看到。也就是说，这意味着在事务处理期间，如果其他事务修改了相应的表，那么同一个事务的多个 SELECT 语句可能返回不同的结果。图 12.6 进行了说明。

假如处于 READ COMMITTED 孤立级，那么在第一个事务执行的过程中，第二个事务看到交易者 A 的账户还是 1000 股。但是，一旦第一个事务被提交，第二个事务即使仍在执行过程中，将会看到交易者 A 的账户是 600 股，交易者 B 的账户是 1400 股。

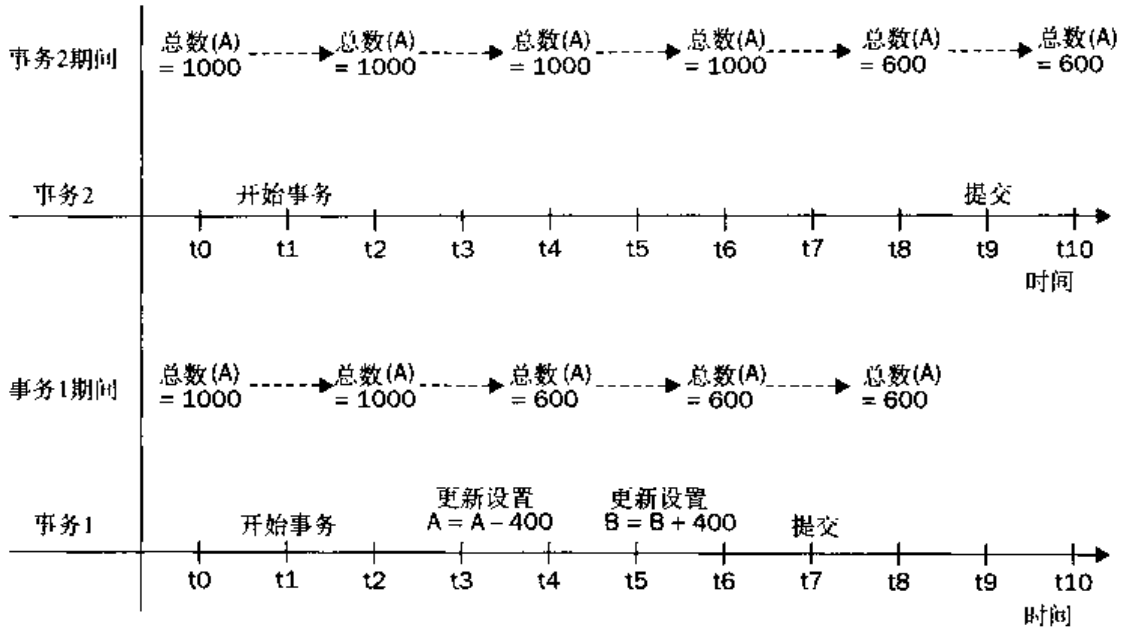

图 12.6 READ COMMITTED 孤立级和不能重复的读操作

很明显，这是一个问题，如果第二个事务在同一个事务中看到两个不同的结果集，将会不知道哪一个是正确的。预测和假设一下，如果不是一个单独的事务，而是很多事务进行修改数据库，我们将看到一个事务执行的每个查询将会产生一个不同的结果集（因而这种情况称为不能重复的读操作）。

未提交的读

READ UNCOMMITTED 孤立级提供了事务之间最小程度的隔离。除了容易产生虚幻的读操作和不能重复的读操作外，处于这个孤立级的事务可以读到其他事务还没有提交的数据。如果这个事务使用其他事务未提交的变化作为计算的基础，然后那些未提交的变化被它们的父事务撤销，这就导致了大量的数据变化。

考虑图12.7。如果我们假设第二个事务处于READ UNCOMMITTED的事务孤立级，就可以在第一个事务的过程中看到第一个事务做出的变化，因此，在它还没有提交给系统时，就可以看到交易者A的账户有600股。如果交易者A和交易者B之间的事务通过ROLLBACK依次被撤销，那么第二个事务将会在错误的数据上进行操作（因此，这种错误的类型称为脏读）。

12.3.3 修改事务孤立级

从 MySQL 4.0.5 开始，可以使用 TRANSACTION ISOLATION LEVEL 变量修改事务孤立级。MySQL 默认为 REPEATABLE READ 孤立级。可以使用 SET 命令改变它，如下例所示：

```
mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
Query OK, 0 rows affected (0.00 sec)
```

可以使用 SELECT 随时获得 TRANSACTION ISOLATION LEVEL 变量的当前值，如下所示：

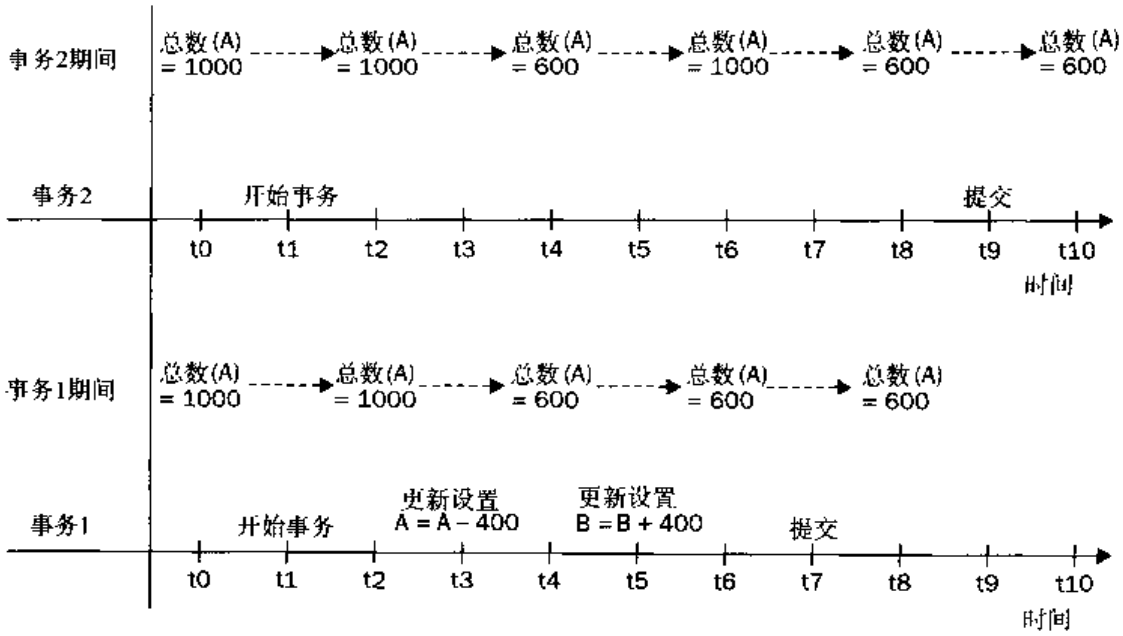

图 12.7　READ UNCOMMITTED 孤立级和一个脏读操作

```
mysql> SELECT @@tx_isolation;
+-----------------+
| @@tx_isolation  |
+-----------------+
| REPEATABLE-READ |
+-----------------+
1 row in set (0.00 sec)
```

默认情况下，TRANSACTION ISOLATION LEVEL 变量的值是基于每个会话设置的，但是可以通过向 SET 命令行添加 GLOBAL 关键字对所有的会话进行全局设置，如下所示：

```
mysql> SET GLOBAL TRANSACTION ISOLATION LEVEL READ COMMITTED;
Query OK, 0 rows affected (0.00 sec)
```

请注意，我们需要 SUPER 优先权来执行这个操作。第 14 章有关于如何在 MySQL 访问控制系统中获得这个（或者其他）优先权的更多内容。

12.4　事务和性能

因为支持事务的数据库在保持不同用户彼此孤立方面要比非事务数据库难，所以这自然地反映了系统的性能。如果由于事务日志的使用导致了系统的失败时，遵从 ACID 规则，特别是和保持数据一致性相关的规则，将会使这样的事务型系统增加过多的负担。MySQL 对于这条规则也不例外——其他事情是相同的，非事务型 MyISAM 表比事务型 InnoDB 和 BDB 表类型要快得多（不过大多数情况下，根据查询的类型和查询的组合发生变化。InnoDB 能够很快地完成很多读操作和写操作的组合，但是 MyISAM 在表的扫描和进行大的删除时要比 InnoDB 快）。

也就是说，如果必须使用事务型表类型时，我们仍然要做一些事来保证事务不会向系统添加不适当的负担。

12.4.1 使用小事务

也许是陈词滥调，在复杂的事务世界中，KISS原则（Keep It Simple, Stupid！保持简单，傻瓜！）是特别有用的。这是因为MySQL使用了行级的锁定机制来避免同时发生的事务编辑数据库中的同一条记录以及可能的破坏。行级的锁定机制防止多个事务同时访问一个行——这保护了数据，但是导致其他事务等待，直到初始事务的锁定完成了它的工作。只要事务很小，这个等待时间就不是很重要了。

然而，当处理大的数据库和许多复杂的事务时，多个事务长时间等待彼此释放锁定很明显地影响了性能。

设定默认事务孤立级

我们还可以使用mysqld服务器过程特定的transaction-isolation参数来设定开始时的默认事务孤立级。

所以，保持事务尽可能地小通常被认为是一个很好的主意，而且要使它们快速地变化和退出，这样其他处于队列中的事务就不会过度地被推迟。在应用程序层，有两个普遍的策略：

- 保证所有要求的用户输入在发出START TRANSACTION命令之前都是可行的。通常情况下，新的应用程序设计者在不知道是否需要的值都是可行的之前就初始化了一个事务。其他在同一时间初始化的事务必须等待用户输入要求的数据，应用程序处理它，再要求更多的数据等等。在单用户环境中，这种延迟不会因为没有其他事务访问数据库而产生影响。但是，在多用户的环境中，一个事务的延迟对队列中所有的事务都产生了影响，导致严重的性能退化。
- 尝试把大的事务分成小的事务然后分别执行。这将保证每个子事务很快地执行，释放系统资源，用来保持系统的状态。

12.4.2 选择合适的孤立级

当我们从无保护的READ UNCOMMITTED孤立级转移到更安全的SERIALIZABLE级时，RDBMS的性能也要受到影响。原因很简单：我们要求系统提供越强的数据完整性，它就越需要做更多的工作，运行的速度也就越慢。因此，作为一个数据库管理员或者系统分析员，需要在RDBMS的孤立性需求和性能之间协调。

在SERIALIZABLE孤立级，RDBMS顺序地执行事务，因此，提供了防止数据破坏最高级的保护。不过，因为这涉及到要等待其他事务释放锁定，它很大程度上降低了应用程序的速度。最后的一个孤立级，READ UNCOMMITTED孤立级允许并行的事务看到彼此所做出的还未保存的变化，在可能存在数据不一致的情况下提供了改善的性能。图12.8说明了事务安全性和性能之间相反的关系。

MySQL默认的是REPEATABLE READ孤立级。这个孤立级适用于大多数应用程序，只有在我们的应用程序有具体的对于更高或者更低孤立级的要求时才需要改动。没有一个标准公式来决定哪个孤立级适用于我们的应用程序——大多数情况下，这是一个主观的决定，它是基于应用程序的容错能力和应用程序开发者对于潜在数据错误的影响的判断。孤立级的选择对于每个应用程序也是没有标准的。例如，同一个应用程序的不同事务基于执行的任务需要不同的孤立级。

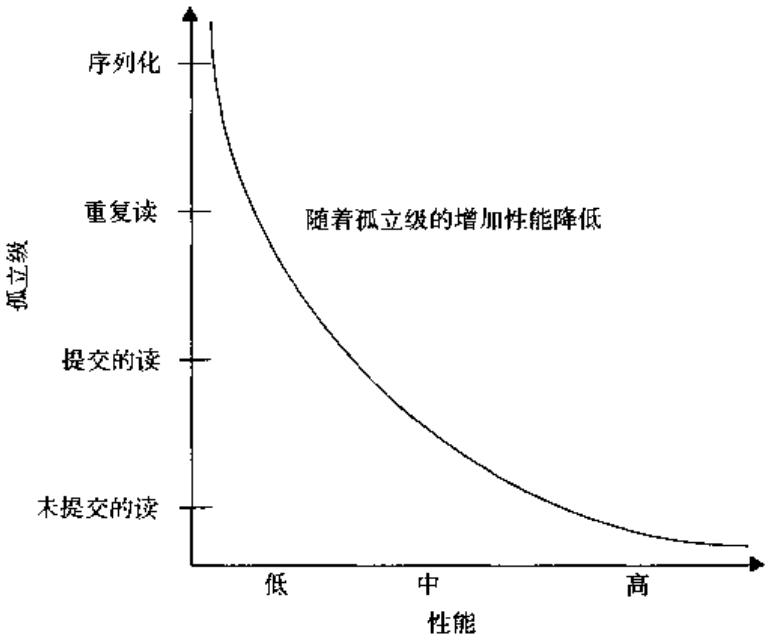

图 12.8　事务孤立级和性能之间的关系

12.4.3　避免死锁

没有对死锁简单的回顾，就不可能完成对事务性能的讨论。如果我们熟悉操作系统编程，就知道什么是死锁——两个进程同时访问同样的资源，每一个都在等待另一个完成，在这种情况下，两个进程都被锁定。

在一个事务环境中，当两个或者多个处于不同序列的客户同时想要更新相同的数据时，就会发生死锁。为了说明这一点，考虑图 12.9，两个不同的事务工作在同一个表上，但是两个事务处于不同的序列。

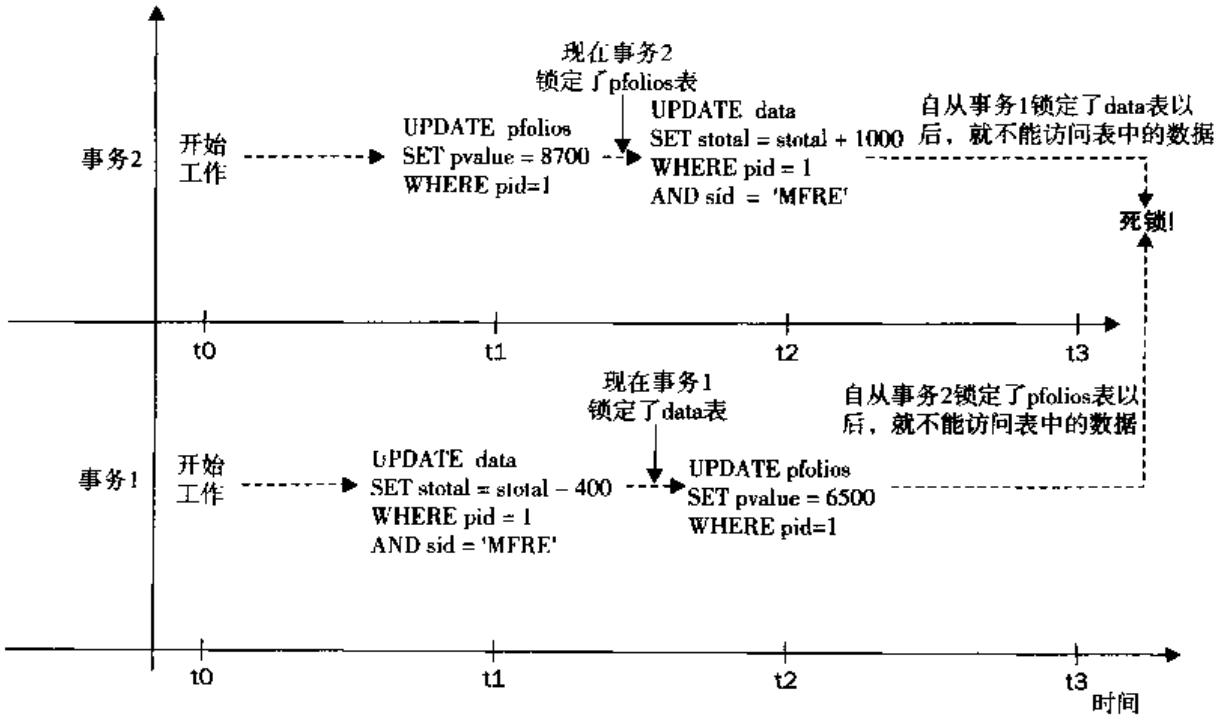

图 12.9　事务死锁

第一个事务尝试从股票账户上减去400股，第二个尝试添加1000股，但是两个事务同时开始。第一个事务的过程是：(1) 从股票账户上减去400股，(2) 更新股票账目表；第二个事务尝试：(1) 更新股票账目表来反映降低的值，(2) 从股票账目上减少1000股。

从前面的例子清楚地看到，当每个事务等待另一个事务完成对它所需要访问的表的操作时，就会导致死锁。如果没有被解决，像前面这样的死锁将导致每个事务毫无目的地等待另一个事务释放对数据的锁定。幸运的是，MySQL的InnoDB表处理程序具有检查死锁情况的内建智能。如果发现了一个死锁，InnoDB表处理程序立刻通过撤销其中的一个事务（或者释放锁定）来解决死锁情况，因而使另一个能够继续进行到逻辑结束。像前面一个例子所输出的那样，通过一个错误信息通知客户拥有的是已撤销的事务。

12.5 具有非事务表的伪事务

到现在为止，我们已经看到InnoDB和BDB表环境中的事务，这是MySQL天生就支持ACID规则事务的仅有的表类型。旧的MySQL表类型，在许多安装MySQL的环境中仍然在使用，它们不支持事务，但是MySQL仍然可以使用户通过表锁定来实现原始形式的事务。这一部分详细介绍伪事务，提供对非事物表执行安全事务的一些基本指标。

准　备

对前面死锁例子的检查，将会发现不止一个数据库问题，它是应用程序设计得差的结果：两个客户操作表的顺序完全不同就会导致死锁的产生。通常情况下，在应用程序层通过仔细地计划和设计可以避免死锁。这就保证在竞争的过程中资源能够很好地共享，能够发现循环链（比如前面所讲的），并在最早的时间解决。

一个开发者在应用程序层可以做很多基本的事情来避免死锁：在会话初期获得所有需要的锁定，一直以相同的顺序处理表，如果RDBMS在解决死锁时撤销了事务，要使用内建的恢复程序来重新执行事务。

12.5.1 用表锁定代替事务

因为MyISAM（以及其他旧的MySQL表）不支持InnoDB格式的COMMIT和ROLLBACK句法，所以每次数据库的变化都被立即保存在磁盘上。像前面所讲的那样，在单用户的环境中，这不会引出很多问题，但是在多用户的环境中，就会导致很多问题，因为它不能创建事务“泡”来使用户所做的变化孤立于其他用户所做的变化。在这种情况下，惟一一种保证不同客户能够看到一致数据的方法是强制方法：在变化的过程中阻止其他用户访问正在变化的表（通过锁定表），只在变化完成后才允许访问。

本章的前面部分已经讨论了InnoDB和BDB表，它们生来就支持行级和页级的锁定来保证同时执行的事务的安全。不过，MyISAM表类型不支持这些锁定机制。所以，要明确地设置表锁定，以避免同时存在的事务互相侵犯空间。

MySQL提供了以下实现表锁定的句法：

```
LOCK TABLES table-name lock-type, ...
```

下面的例子在users表上设置了一个只读锁定：

```
mysql> LOCK TABLE users READ;
Query OK, 0 rows affected (0.05 sec)
```

在同一时间锁定多个表是很常见的。我们通过在LOCK TABLES命令后面指定一个以逗号分开的表名字列表和锁定类型就可以很简单地实现，如下所示：

```
mysql> LOCK TABLES users READ, pfolios WRITE
Query OK, 0 rows affected (0.05 sec)
```

前面的语句以READ方式锁定了users表，以WRITE方式锁定了pfolios表。

可以使用下面的UNLOCK TABLES命令解除锁定：

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

我们不需要指出解除锁定的表的名字。MySQL会自动对前面通过LOCK TABLES锁定的所有表解除锁定。

有两种类型的表锁定：读锁定和写锁定。下面进一步讨论。

读锁定

表的读锁定是指设置锁定的线程（客户）可以从表中读数据，其他线程也可以。不过，每个线程都不能修改锁定的表，只要锁定处于活动状态就不能添加、更新或者删除记录。

下面是一个简单的例子，可以看到READ锁定如何工作。首先向users表设置一个READ锁定：

```
mysql> LOCK TABLE users READ;
Query OK, 0 rows affected (0.05 sec)
```

然后进行读操作：

```
mysql> SELECT * FROM users;
+-----+--------+
| uid | uname  |
+-----+--------+
|   2 | jim    |
|   1 | john   |
|   3 | ken    |
|   5 | philip |
|   4 | scott  |
+-----+--------+
5 rows in set (0.23 sec)
```

没有问题。现在，要进行写操作会怎么样？

```
mysql> INSERT INTO users (uid, uname) VALUES (6,'tom');
ERROR 1099: Table 'users' was locked with a READ lock and can't be updated
```

MySQL拒绝进行插入，因为表以只读的方式锁定。

其他客户要访问同一个数据库会怎么样？打开一个新的MySQL客户，然后尝试去访问同一个表。如下所示，读操作没有问题：

```
mysql> SELECT * FROM users;
+-----+--------+
| uid | uname  |
+-----+--------+
|   2 | jim    |
|   1 | john   |
|   3 | ken    |
|   5 | philip |
|   4 | scott  |
+-----+--------+
5 rows in set (0.23 sec)
```

两种锁定方式

READ 锁定的一个变体是 READ LOCAL 锁定，它和常规的 READ 锁定是不同的，因为其他线程可以执行 INSERT 语句，它和初始化锁定的线程拥有的会话不冲突。它是为了使用 mysqldump 工具对一个表同时进行多个 INSERT 操作而创建的。

写怎么样？

```
mysql> INSERT INTO users (uid, uname) VALUES (7,'paul');
```

在这种情况下，MySQL 客户将会停止，等待第一个会话释放锁定。如果返回到第一个 MySQL 会话，执行 UNLOCK TABLES 命令：

```
mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.04 sec)
```

锁定将被释放，第二个会话进行查询：

```
mysql> INSERT INTO users (uid, uname) VALUES (7,'paul');
Query OK, 1 row affected (27 min 15.11 sec)
```

写锁定

表的写锁定是指设置锁定的线程可以修改表中的数据，但是其他线程在锁定期间既不能从表中读数据也不能写数据。

下面是一个简单的例子，我们可以看到写锁定如何工作。首先向 data 表设置一个写锁定：

```
mysql> LOCK TABLE data WRITE;
Query OK, 0 rows affected (0.05 sec)
```

然后从中读数据：

```
mysql> SELECT * FROM data;
+-----+------+--------+
| pid | sid  | stotal |
+-----+------+--------+
|   1 | INFY |   9000 |
|   1 | MSFT |    100 |
|   2 | MSFT |   1000 |
|   2 | INTC |   1000 |
```

```
|    3 | INFY |     500 |
|    3 | CISC |    1000 |
|    1 | CISC |    1000 |
|    1 | ACME |    2000 |
|    1 | INTC |    2000 |
|    2 | ACME |    1000 |
+------+------+---------+
10 rows in set (0.00 sec)
```

因为表被写锁定，所以可以向其中写数据，不会有问题：

```
mysql> UPDATE data SET stotal = 5000 WHERE pid = 1 and sid = 'INFY';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

其他MySQL会话怎么样？打开一个新的客户会话，在WRITE锁定处于活动状态时尝试从相同的表读数据。

```
mysql> SELECT * FROM data;
```

现在MySQL客户在执行前面的命令前将会停止，等待第一个会话释放锁定。所以，返回到第一个会话，释放对表的锁定：

```
mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.04 sec)
```

我们看到第二个会话使用的SELECT命令现在被MySQL接受进行处理，因为data表已经不再被锁定：

```
mysql> SELECT * FROM data;
+-----+------+---------+
| pid | sid  | stotal  |
+-----+------+---------+
|    1 | INFY |    9000 |
|    1 | MSFT |     100 |
|    2 | MSFT |    1000 |
|    2 | INTC |    1000 |
|    3 | INFY |     500 |
|    3 | CISC |    1000 |
|    1 | CISC |    1000 |
|    1 | ACME |    2000 |
|    1 | INTC |    2000 |
|    2 | ACME |    1000 |
+-----+------+---------+
10 rows in set (3 min 32.98 sec)
```

锁定的优先权

在同时涉及到WRITE和READ锁定的情况下，MySQL对于WRITE锁定分配较高的优先权来保证对于表的修改能尽快地保存到磁盘上。这就降低了磁盘崩溃或者系统失败导致更新丢失的危险。

前面的例子指出了表锁定最重要的缺点之一：如果一个线程不释放它的锁定，其他试图访问这个已锁定表的尝试都会超时等待，导致了整体性能很大程度地降低。

12.5.2 使用表锁定实现伪事务

现在我们看一下如何通过使用表锁定来实现一个事务。我们将通过重写前面的一个例子进行解释，也就是向公司数据库添加一个新雇员的例子，使用锁定和MyISAM表来代替事务型代码和InnoDB表。

第一步是创建在事务期间要使用的表。在前面的例子中，涉及到的步骤包括：(1) 在employees表中为每个雇员创建一条记录，(2) 在departments表中为每个雇员分配部门，(3) 在payroll表中指定每个雇员的工资和奖金。

因为当添加一个新雇员时，这三个表都要发生变化，因此它们必须以WRITE方式被锁定，这样其他的会话就不能影响这个过程：

```
mysql> LOCK TABLES employees WRITE, departments WRITE, payroll WRITE;
Query OK, 0 rows affected (0.00 sec)
```

正如前面所解释的那样，WRITE方式是指只要锁定处于活动状态，其他会话既不能从锁定的表中读数据也不能写数据。因此，事务要尽可能地小，以避免其他事务对这个表数据的访问要求被延迟。

下一步，向不同的表插入新的记录：

```
mysql> INSERT INTO employees (ename, enationality) VALUES ('Tim Jonez', 'US');
Query OK, 1 row affected (0.13 sec)

mysql> INSERT INTO departments (eid, dept) VALUES (LAST_INSERT_ID(), 'Accounts');
Query OK, 1 row affected (0.03 sec)

mysql> INSERT INTO payroll (eid, pbasic, pbonus) VALUES (LAST_INSERT_ID(), 36000.00,
2500.00);
Query OK, 1 row affected (0.02 sec)
```

使用SELECT验证数据被正确地输入：

```
mysql> SELECT * FROM employees;
+-----+-----------+--------------+
| eid | ename     | enationality |
+-----+-----------+--------------+
|   1 | Tim Jonez | US           |
+-----+-----------+--------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM payroll;
+-----+----------+---------+
| eid | pbasic   | pbonus  |
+-----+----------+---------+
|   1 | 36000.00 | 2500.00 |
+-----+----------+---------+
1 row in set (0.05 sec)
```

释放对表的锁定，这样就完成了：

```
mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.09 sec)
```

直到表被解除锁定前，其他会话要求访问这三个已锁定表的尝试都将被迫等待。事务型方法的长处（也就是即使在事务的过程中，页级和行级的锁定允许其他客户使用相同的数据）在这里没有体现。不过，表锁定使不同会话进行的更新彼此孤立（虽然以原始的方式）。通过这些方法，可以帮助用户约束那些旧的、非事务表类型，实现一个事务型的环境。

12.6 小结

本章讨论了MySQL最重要的新特征之一：把多个SQL语句作为一个单独的事务单元，并让这个单元自动执行的能力。与记录RDBMS所有变化的内建的二进制日志结合，这个事务型能力可以以一种更安全的方式执行SQL查询，如果发生错误，就把RDBMS返回到以前的稳定状态。因而，事务型支持通过外加的SQL查询和包含对事务的支持，在使SQL应用程序更健壮、减少数据破坏的可能性方面扮演了很重要的角色。

事务可能导致RDBM资源的性能很明显地降低，因为当多个用户访问同一个数据时，需要很大的资源来使事务彼此隔离。因此，除了讨论了事务的生命周期以外，本章还解释了控制事务行为和性能的多个MySQL变量，最主要的是TRANSACTION ISOLATION LEVEL变量，它控制了事务和其他行为互相隔离的程度。

最后，因为只有新的InnoDB和BDB表类型支持事务，本章简要解释了如何使用旧的MyISAM表格式模仿事务型的环境，使得用表锁定这个原始的方式很有效果。

第三部分

管　　理

第 13 章　管理与配置
第 14 章　安全、访问控制和权限
第 15 章　维护、备份和恢复
第 16 章　性能优化
第 17 章　MySQL 复制

第13章　管理与配置

本书前面章节已经集中讨论了多有关每天使用MySQL的工作，包括创建数据库、添加数据和运行查询等，但是很少涉及这些操作的管理问题，如管理安全、分配用户权限和备份数据等。对于大多数用户来说，有了前面的技能就可以顺利工作。当然如果不是用户而是一名数据库管理员，其工作包括管理MySQL服务器方面的内容，情况将会怎样？

在这种情况下，只是了解SQL查询方面的知识还不够，还需要了解如何管理MySQL关系数据库管理系统，并且承担起责任保证系统用户一直可以使用MySQL的服务。这个角色涉及很多方面，包括保证MySQL服务器拒绝未经授权的使用或破坏；按照用户预计系统使用要求分配用户权限；对MySQL数据库实施常规检查和备份，避免数据破坏或损失；优化服务器使用，保证始终提供最佳性能。

从本章开始将涉及这方面的内容。经过下面的章节，你将了解到有关MySQL服务器管理的各方面知识，并且可以了解到如何更快、更有效地完成日常任务。本章简要介绍这方面的知识，包括一些日常任务，像服务器的启动和关闭，修改服务器默认配置和检查MySQL日志文件等。

13.1　数据库管理与MySQL

数据库管理员在一个机构的MIS团队中属于重要岗位。因为这种人员承担着确保网络数据库平稳有效使用的责任，所以这个岗位的职责描述包括：确保数据库对用户和应用程序达到24 × 7 × 365标准运行时间；实施常规备份，以便当系统崩溃时能快速恢复；调整服务器参数，以最大化服务器性能；确保数据库不受恶意破坏并拒绝未经授权的访问。如果逐项分析，这些任务都不能说十分简单。把它们合并到一起，就产生了行业中一个要求最高且最具挑战性的职位。

幸运的是，MySQL可以使用一些有效的工具来帮助数据库管理员解决每天遇到的琐事。

13.1.1　运行时间

通过MySQL开发者的精细设计，加上使用开放源代码社区的充分测试，已经能够创建一个非常平稳和高度可靠的产品，这一点数据库管理员应该坚信。MySQL可以提供最为可靠的系统和最高的运行时间，并且已经被一些公司和组织，如SAP，Motorola，Sony，Yahoo!，NASA和HP（这仅仅是很少的一部分）测试和验证，可以用于大容量和关键任务应用程序。

系统崩溃的一个常见原因涉及应用程序代码中的设计错误和缺陷。MySQL开放式源代码的历史，使得这种情况要比其他很多同类商业数据库系统发生的机会少很多。因为MySQL开发完全是在公众的关注之下，在被作为认证可以用于生产环境之前，最后版本的发布得到了世界各地用户的彻底测试，可以从中得到独特的收益，包括在不同平台上面和多种不同环境下面，

以及经过MySQL AB的名为“使我崩溃”的内部测试套件测试，因为这个套件的主要目的就是要使一个系统崩溃。这种方法导致了特别稳定且实际上没有缺陷的数据库产品。

13.1.2　数据备份

虽然MySQL可以说非常稳定和可靠，而且还可以不需要专门管理就完全能够自己运行，但是在任何生产环境下，它的基本维护和建立备份和恢复过程仍然需要管理员操作。特别是在备份方面，因为对于确保数据库的安全十分关键，应避免因为磁盘故障或系统崩溃造成数据破坏或丢失的情况出现。

应该记住，MySQL附带了很多工具，专门设计用来提高这个过程的速度，使得工作效率更高。其中一个重要工具称为mysqldump实用程序，它可以写MySQL表结构、表数据或者两者同时写，以便备份各种不同格式的文件。mysqldump的输出可以很容易地使用，从而能快速恢复一个或多个MySQL数据库，其方法是使用mysqlimport或mysql客户机命令行方式，或者通过LOAD DATA INFILE命令。在表被破坏的情况下，MySQL可以通过一个恢复实用程序的套件改善数据恢复的机会，这些实用程序十分擅长深入被破坏表的内部，或者完全修复这个表，或者修理被破坏的点，这样可以使得大多数数据得到恢复。

MySQL还配备了内置复制，可以根据预先定义的主次关系，从一个数据库服务器向另一个服务器镜像产生的变化。MySQL的早期版本只能支持单向复制，现在比较新的版本能够支持双向复制，可以更为灵活地进行镜像和保证负荷平衡。

13.1.3　安全和访问控制

MySQL配备了十分完善的访问控制和权限系统，防止未经授权的用户访问系统。这个系统使用5层权限结构，可以创建十分庞大的访问规则，这些规则在决定如何处理具体用户申请时供MySQL使用：

- 如果与MySQL权限系统中规定的访问原则匹配，则允许连接服务器。这种访问规则可以以用户和主机为基础确定，用来限制来自确定子网或IP地址范围以外的主机的访问。而这种与服务器的连接只能在用户提供了有效口令之后才能得到许可。
- 一旦连接建立，MySQL将检查用户执行的每个动作，确认是否得到实施该动作的足够的权限。用户可以被限制只能在具体数据库或字段中操作，MySQL甚至可以控制用户能够在数据库、表或字段水平上运行哪些类型的查询。

通过对用户口令的单向加密方案，系统的安全得到了进一步加强。这种加密方案最初可能十分幼稚，在MySQL的4.x版本之后已经得到了很大的改善。MySQL更新的版本还可以支持SSL，可以用来加密发送到MySQL客户-服务器连接（如查询结果）的数据，以便实现更高级的安全。

13.1.4　性能优化

数据库管理员一旦考虑到有关例程事宜，如备份数据和系统安全等，就必须重点发挥出RDBMS的最大性能。MySQL的多线程引擎和多种优化算法可以让这些活动比很多其他RDBMS更方便，通常可以非常快速地实现MySQL的完整安装，而且很少需要修改。

这就是说MySQL可以满足很多具体需要。MySQL通过系统变量产生相当多的内部参数，并且允许开发者和管理员很容易地进行修改，以便满足定制要求。很多会降低系统性能的功能，如事务、参照完整性和被保存的程序等，可以允许或禁止用户选择，这样就可以针对每个应用程序选择功能和性能的优化方案。例如ANALYZE TABLE和EXPLAIN SELECT命令可以帮助MySQL开发者设置查询的基准点，寻找性能的瓶颈，并且有些新性能，如Query Cache（可以缓存常规查询结果，将缓存数据退给调用者，而不必在每次查询时重新执行）可以在减少额外编程时帮助改善性能。

在讨论性能问题时的一个重要因素是可扩展性。很多数据库系统在处理数千条记录时工作效果十分好，但是当数量增加到数以百万计时，其性能将大幅降低。然而使用MySQL将不会出现这种问题。MySQL具有很高的可扩展性，可以处理数据很大的和复杂的数据库（包含成千上万条记录的几个GB的表）。因此使得MySQL适合各种情况，从简单的基于内容的网页形式的网站，到特别复杂的大型多变化数据网络，如可以用来电子跟踪的数据仓库或者商业智能管理系统等。

下面各节将详细介绍这些内容。先从常规服务器的管理任务开始。

13.2 基本的服务器管理和配置任务

MySQL的发布版本包括了一个命令行工具，专门用于帮助管理员执行常规任务，如修改MySQL管理员口令或重新装载MySQL权限等。这个工具称为mysqladmin，可以在MySQL安装的bin目录下找到，通常可以以下列方式调用一个或多个命令：

```
mysqladmin command
```

参见下列示例：

```
[root@host]# /usr/local/mysql/bin/mysqladmin shutdown
```

表13.1列出了mysqladmin可以支持的一些特别有用的命令。

表13.1 mysqladmin支持的命令

命令	作用
status	返回服务器状态信息
password	修改用户口令
shutdown	关闭MySQL服务器
reload	重载MySQL授权表
refresh	重置所有高速缓存和日志
variables	返回所有服务器变量值
version	返回服务器版本
processlist	返回服务器上所有进程的列表
kill	取消一个服务器进程
ping	测试服务器是否在活动状态

运行mysqladmin --help能够得到可以使用命令的一个完整列表。

在本章后面以及本节所有内容中，可以看到服务器管理将广泛地使用这个工具。

现在先来看一下一些常用的数据库管理任务，作为进入后面章节更高级内容的先导。

mysqladmin 的替代工具

如果还不熟悉或不习惯使用命令行，可以使用下列很多图形用户界面来替代mysqladmin工具：

- phpMyAdmin（http://www.phpmyadmin.net/），这是针对 MySQL 管理的一个基于浏览器的界面。
- MySQL Control Center (http://www.mysql.com/products/ mysqlcc/index.html)，这是 Windows 和 UNIX 两种平台 MySQL 管理工具的 GUI。
- WinMySQLadmin，这是在 MySQL 服务器上备有的对 MySQL 管理的 Windows GUI。

13.2.1　启动和关闭服务器

在 UNIX 上，MySQL 附带一个启动 / 关闭脚本，这是启动和关闭 MySQL 数据库服务器建议使用的方法。这个脚本名为 mysql.server，可以在 MySQL 安装的 support-files 子目录下找到。可以使用下面的方式来调用以启动 MySQL 服务器：

```
[root@host]# /usr/local/mysql/support-files/mysql.server start
```

还有一种代替方法可以直接调用服务器，称为 mysqld_safe 包装，用法如下：

```
[root@host]# /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

在没有 Windows 服务器的情况下（Windows 95/98 或 Me），启动 MySQL 服务器的最简单方法是在 Windows Explorer 中导航到自己的 MySQL 安装目录，双击 mysqld 来启动这个二进制文件，在 Start -> Run 对话框中或者在命令行 DOS 提示下键入完整的到 mysqld 二进制文件的路径来启动它，方法如下：

```
C:\> c:\mysql\bin\mysqld.exe
```

安全胜于遗憾

虽然也可以使用在安装文件的bin目录下面直接运行mysqld二进制名的方法来调用MySQL，但是并不推荐使用这种方法，可以认为使用 mysqld_safe 包装的方法比较安全，因为这个包装考虑了自动联机错误和针对文件的运行时信息，并且在出现意外的关机情况下可以重新启动 MySQL 守护程序。

注意，在 MySQ L 的比较旧的版本中，mysqld_safe 称做 safe_mysqld。

在服务器级的 Windows 设备上（Windows NT/2000/XP 和 2003）都更加适合运行 MySQL，MySQL 服务器可以作为后台服务运行。这种方法可以通过安装 mysql 服务来实现，然后使用 net start 命令来激活这个服务。为了实现这个功能，首先需要执行下面的命令，或者通过 Start -> Run 对话框，或者在命令行 DOS 窗口中键入：

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

接下来通过服务控制面板（Control Panel -> Administrative Tools -> Services），或者通过下列命令来启动这个服务：

```
C:\> net start mysql
```

根据现在运行的 Windows 版本的情况，MySQL 可以在前台启动（mysqld 控制台窗口被打开并且在 Windows 任务栏上总可以看到）或者在后台作为一种服务来启动，运行时无法看到。

如下所示，可以使用 ping 命令证实服务器正在使用 mysqladmin 工具运行。这样将返回一个状态信息，表示服务器是否被激活：

```
[root@host] # /usr/local/mysql/bin/mysqladmin ping
mysqld is alive
```

mysqld 不同的名称

MySQL 的 Windows 二进制发布包括很多不同的 mysqld 二进制名。下面是各自的功能：

- **mysqld** 常规 MySQL 服务器。
- **mysqld-nt** 常规 MySQL 服务器，可以支持 Windows NT/2000/XP 上面的命名管道。
- **mysqld-opt** 优化 MySQL 服务器，提供的一些功能可以挖掘更好的性能。
- **mysqld-max** 与 mysqld 一样，但可以支持更新、更具实验性质的性能（但是更不稳定）。
- **mysqld-max-nt** 与 mysqld-max 一样，可以额外支持 Windows NT/2000/XP 上的命名管道。

大多数情况下，根据使用平台情况，可能更希望试着使用 mysqld 或 mysqld-nt 二进制名。如果不能使用这些功能，mysqld-opt 通常可以作为第二种安全的选择。

Mysqladmin 实用程序也可以用于重载服务器的授权表，具体方法如下（本示例是针对 UNIX 平台的，对于 Windows，可以简单地更换 MySQL 安装路径即可）：

```
[root@host] # /usr/local/mysql/bin/mysqladmin reload
```

一旦服务器运行，可以在任何时间内使用 MySQL 附带的 mysqladmin 实用程序关闭它，方法是调用这个命令加上 shutdown 参数。在 UNIX 上面，具体方法如下：

```
[root@host] # /usr/local/mysql/bin/mysqladmin shutdown
mysqld is alive
```

在 Windows 上面，通常需要打开一个新的 DOS 控制台窗口，或者使用 Start -> Run 对话框，方法如下：

```
C:\> c:\mysql\bin\mysqladmin shutdown
```

在 UNIX 上面，也可以使用提供的 mysql.server 启动 / 关机脚本来关闭服务器，方法如下：

```
[root@host] # /usr/local/mysql/support-files/mysql.server stop
```

提示： 使用 kill 命令（UNIX）或任务管理器（Windows）可以立即取消 mysqld 过程，而不必立即关闭 MySQL。这种过早的关闭，如果服务器正在往磁盘上写数据，当收到终止信号时，可能造成数据丢失或者破坏。

配置 MySQL 在系统引导时自动启动

也可以把 MySQL 设置成为在系统引导时自动启动。下面是具体方法：

- 在 UNIX 上面，可以在引导时让 MySQL 自动启动，方法是复制 MySQL 发布提供的 mysql.server 脚本到系统的 /etc/init.d/* 目录中，并根据系统引导和关机脚本的具体参数调用这个脚本。
- 如果需要在 Windows 95，Windows 98 或 Windows Me 上自动启动 MySQL，可以简单地添加一个指向 mysqld 服务器二进制名的快捷方式到 Startup 组。然后每当 Windows 启动

时都会调用该服务器。在 Windows NT，Windows 2000 或 Windows XP 上，同样的程序还包括安装 MySQL 作为 Windows 控制面板上的一项服务，方法是使用 --install 命令行选项调用 mysqld 二进制文件，并且确认在 Control Panel -> Administrative Tools -> Services 中该服务已经设置为自动启动。

使用前检查

在启动 MySQL 服务器之前，重要的是审查和修改 my.cnf（或 my.ini）文件中的运行时配置设置。一旦系统启动，服务器将读这个文件，所以任何改变都需要进行重新启动才能生效。

13.2.2　检查 MySQL 服务器状态

可以通过mysqladmin工具，使用status命令查看服务器的当前状态（服务器正常运行时间、每秒查询数量、当前打开表的数量等）。参阅下列示例：

```
[root@host]# /usr/local/mysql/bin/mysqladmin status
Uptime: 1743  Threads: 2  Questions: 31  Slow queries: 0  Opens: 13
Flush tables: 1  Open tables: 2  Queries per second avg: 0.018
```

version 命令可以提供更为详尽的总结信息，包括 MySQL 服务器的版本信息：

```
[root@host]# /usr/local/mysql/bin/mysqladmin version
/usr/local/mysql/bin/mysqladmin  Ver 8.40 Distrib 4.0.15, for pc-linux on
i686
Copyright (C) 2000 MySQL AB & MySQL Finland AB & TCX DataKonsult AB
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL license
Server version          4.0.15-standard
Protocol version        10
Connection              Localhost via UNIX socket
UNIX socket             /tmp/mysql.sock
Uptime:                 29 min 38 sec
Threads: 2  Questions: 32  Slow queries: 0  Opens: 13  Flush tables: 1
Open tables: 2  Queries per second avg: 0.018
```

还有一种类似方法是使用 VERSION()的内置功能，方法如下：

```
mysql> SELECT VERSION();
+-----------+
| VERSION() |
+-----------+
| 4.0.15    |
+-----------+
1 row in set (0.11 sec)
```

通过在mysqladmin中使用extended-status命令或者使用SHOW STATUS命令还可以得到扩展的状态信息：

```
mysql> SHOW STATUS;
+--------------------------+-------+
| Variable_name            | Value |
```

```
+---------------------------+-------+
| Aborted_clients           | 0     |
| Aborted_connects          | 1     |
| Bytes_received            | 125   |
| Bytes_sent                | 3319  |
| Com_admin_commands        | 1     |
| Com_alter_table           | 0     |
...
| Threads_cached            | 0     |
| Threads_created           | 4     |
| Threads_connected         | 1     |
| Threads_running           | 1     |
| Uptime                    | 329   |
+---------------------------+-------+
132 rows in set (0.00 sec)
```

正如所见，这种扩展状态信息提供了大量的实时状态信息。该报告包括服务器自最后一次启动以来已经收到的流量数量，发出和接收的字节数量，客户进行了多少次连接，包括中断、多少次成功连接和多少次失败连接，以及多少次异常终止。还包括了服务器自启动以来进行的查询总量，包括每种查询数量的信息（SELECT, DELETE, INSERT 等）、活动线程数量、当前客户连接数量、运行查询数量和打开表数量等。

13.2.3 管理 MySQL 客户机过程

使用SHOW PROCESSLIST命令可以得到如下的完整的连接到服务器上所有客户机过程的列表：

```
mysql> SHOW PROCESSLIST;
+----+------+-----------+---------+---------+------+--------------+-----------
---------------------------+
| Id | User | Host      | db      | Command | Time | State        | Info    |
+----+------+-----------+---------+---------+------+--------------+-----------
---------------------------+
| 10 | root | localhost | NULL    | Query   | 0    | NULL         | show
processlist                        |
| 12 | root | localhost | library | Query   | 0    | Sending data | select *
from users, categories, info |
| 14 | root | localhost | student | Sleep   | 20   |              | NULL    |
+----+------+-----------+---------+---------+------+--------------+-----------
---------------------------+
3 rows in set (0.00 sec)
```

使用 SHOW PROCESSLIST 输出，常规用户只能看到自己的线程，但是有 PROCESS 权限的用户可以看到所有正在运行的线程，具有所有能力的SUPER权限用户甚至可以使用KILL命令取消正在运行的线程。参阅下列示例：

```
mysql> KILL 12;
Query OK, 0 rows affected (0.01 sec)
```

mysqladmin 工具可以提供类似的 processlist 和 kill 命令：

```
[root@host]# /usr/local/mysql/bin/mysqladmin processlist
+----+------+-----------+---------+---------+------+-------+---------------+
| Id | User | Host      | db      | Command | Time | State | Info          |
+----+------+-----------+---------+---------+------+-------+---------------+
| 10 | root | localhost |         | Sleep   | 83   |       |               |
| 14 | root | localhost | student | Sleep   | 142  |       |               |
| 17 | root | localhost |         | Sleep   | 19   |       |               |
| 18 | root | localhost | library | Sleep   | 2    |       |               |
| 20 | root | localhost |         | Query   | 0    |       | show processlist |
+----+------+-----------+---------+---------+------+-------+---------------+
[root@host]# /usr/local/mysql/bin/mysqladmin kill 18
```

有关 MySQL 权限系统的更多信息，包括如何向用户分配权限的说明请参阅第 14 章。

13.2.4　修改服务器配置

大多数情况下，不需要修改 MySQL 默认配置。这个软件已经事先通过 MySQL AB 进行了配置，以便满足绝大多数常规需求。然而，在默认配置不适宜用户的情况下，MySQL 还提供了大量变量，这些值可以进行修改，以便满足定制要求。有些变量可以在启动 MySQL 服务器时设置，还有些变量是在服务器运行中设置的。

取消行动

应该注意，线程在收到取消信号时不会立即停止下来。然而 MySQL 给具体线程设置了一个取消标志，一旦完成操作，线程都会进行检查，不论当前执行什么操作。这种方法可以认为比立即取消更安全，因为允许线程完成操作，不论正在做什么操作，并且在终止之前解除已经创建的任何锁定。通常线程在进行每次重要读写操作之后将检查取消标志。

使用选项文件

设置 MySQL 选项的推荐方法是通过一个选项文件，该文件基本上是一个 ASCII 配置文件，包含了 MySQL 服务器在启动时读取的可变值对。MySQL 在启动时需要查找一些标准位置下的选项文件：

- **Windows**　MySQL 将在 Windows 安装目录中的名为 my.cnf 的选项文件和 C:\my.cnf 中查找启动选项。作为代替，my.cnf 文件可以采用文件名 my.ini（实际上只有在引导卷驱动不是C盘这种少见的情况下才需要），位于 Windows 目录中（C:\WINNT 或 C:\WINDOWS）。
- **UNIX**　MySQL 在 /etc/my.cnf, ~/.my.cnf 目录中和在 MySQL 数据目录中的名为 my.cnf 的文件中检查启动选项。

提示： *可以要求 MySQL 通过在 MySQL 的命令行上指定带 --defaults-file 选项的文件名，在不同位置上查找启动选项。*

选项文件的格式通常很简单，类似一个 Windows INI 文件。文件按组来划分，每个组包含可变值对。在 MySQL 命令行中指定的任何选项可以放置在这个文件中，前面不必使用双斜线。参阅下列示例：

```
[mysqld]
port=3306
```

```
skip-locking
log-bin
skip-bdb
```

所有程序针对一个选项文件

MySQL带有的所有二进制程序（不仅仅是mysqld）都可以在一个选项文件中读选项。简单地确定程序名作为选项文件的一个组（通过在方括号中的名称）并且后跟希望设置的变量。MySQL客户机程序可以使用一个特殊的[client]组，通常这个组用于保存用户名和口令连接参数。下面是这个命令工作的简单示例：

```
[client]
user=timothy
password=greenpeas
```

注释：MySQL发布中带有很多示例配置文件（可以参考my-large.cnf，my-huge.cnf，my-medium.cnf和my-small.cnf文件），应该认真掌握如何使用这些文件。

通常MySQL需要在组[mysql]和[mysqld]中寻找配置选项。在UNIX中，如果是使用mysqld_safe MySQL包装来启动MySQL，也可以使用[mysqld_safe]组来向MySQL服务器传递选项。

在这种情况下，当mysql客户机试图连接到MySQL服务器时，将可以连接一个口令为greenpeas的用户timothy。

表13.2列出了可以用于配置MySQL的十分常见和有用的选项（参阅MySQL手册可以看到完整列表）。

表13.2 mysqld命令行选项

选项	意义
ansi	使用更为严格的ANSI SQL-99句法
basedir	设置MySQL安装目录的位置
datadir	设置MySQL数据目录的位置
debug	创建一个排错文件
default-character-set	设置默认字符集
default-table-type	为新表设置默认表类型
flush	在每个SQL命令后向磁盘写变化
init-file	设置包含在启动时执行的SQL命令的文件
language	设置错误信息的语言
log	向日志写MySQL信息（连接和查询）
log-long-format	向每个日志项目添加其他信息（类似用户名和时间戳）
log-error	向日志写关键错误信息
log-warnings	向日志写警告信息
log-slow-queries	向日志写慢执行查询
log-bin	向二进制日志（用于备份和复制）写修改数据的查询
port	设置监听服务器的端口，以便客户机连接
safe-show-database	只向已登录的用户显示数据库
skip-bdb	禁止BDB表处理
skip-innodb	禁止InnoDB表处理

（续表）

选项	意义
skip-grant-tables	执行访问控制时绕过授权表
skip-networking	只允许本地请求，停止监听 TCP/IP 请求
socket	设置本地连接使用的套接字 / 命名管道名
transaction-isolation	设置默认事务隔离等级
user	确定可以作为用户运行的服务器
tmpdir	设置临时文件区的位置

注意，所有这些选项都可以在 MySQL 命令行上面指定，简单地前面使用双虚线标识选项名。参阅下列示例：

```
[root@host]# /usr/local/mysql/bin/mysqld_safe --socket=/usr/tmp/mysql.socket
--user=mysql --skip-networking &
Starting mysqld daemon with databases from /usr/local/mysql/data
```

在存在多种选项文件或相同选项指定了多次不同值的情况下，MySQL使用最后发现的值。因为MySQL在命令行变量之前读选项文件，这意味着命令行指定的选项权限优于选项文件中的选项。

使用 SET 命令

在服务器运行时，MySQL 还允许使用 SET 命令修改系统和连接变量。SET 命令的使用方法为：

```
SET variable = value, variable = value, ...
```

下面是一个示例，其中 SET 命令被用来设置新表的默认表类型：

```
mysql> SET table_type = innodb;
Query OK, 0 rows affected (0.00 sec)
```

使用SET命令的变量集，根据SET关键字后面使用的是GLOBAL或SESSION关键字，可以全局设置在所有会话上，或者只设置在当前会话上。下面的示例限制服务器在任何时间内连接 10 个客户机，并且设置读缓冲大小为 250 KB：

```
mysql> SET GLOBAL max_user_connections=10, SESSION read_buffer_size=250000;
Query OK, 0 rows affected (0.08 sec)
```

注意，SUPER 权限需要设置在 GLOBAL 变量上。在第 14 章中可以读到更多有关 MySQL 权限系统的内容。

锁好门锁

如果应用程序存放在与MySQL服务器相同的物理设备上，同时如果并不希望MySQL客户机可以从其他主机上连接，使用 --skip-networking 选项关闭 TCP/IP 的监听可以很大程度上增强 MySQL 的安全性。

表13.3列出了一些可以使用SET命令设置的重要变量（了解完整列表请参见MySQL手册，同时在第 16 章中可以得到有关这些变量的更详细介绍）。

表 13.3 MySQL 重要变量

变量	功能
autocommit	切换 MySQL 是否试图自动提交打开事务
key_buffer_size	设置索引使用的缓冲大小
table_cache	设置 MySQL 可以在任何时间内打开表的总数
table_type	设置默认表类型
concurrent_inserts	在 MyISAM 表中允许 INSERT 和 SELECT 并发
interactive_timeout	设置对话客户机连接的时限
language	设置错误信息使用的语言
lower_case_table_names	自动小写表名
sort_buffer_size	为排序结果设置缓冲最大规模
read_buffer_size	为读表设置缓冲规模
max_binlog_size	设置循环之前二进制日志的最大规模
max_connections	设置在任何具体时间内允许客户机连接的最大数量
max_user_connections	设置在任何具体时间内可激活的单独客户机连接的最大数量
max_tmp_tables	设置在任何具体时间内客户机可以打开临时表的最大数量
query_cache_type	查询高速缓存开关切换
query_cache_size	设置查询高速缓存的最大规模
tmpdir	设置临时文件区的位置
tx_isolation	设置事务独立等级

检索变量值

变量一旦已经通过SET或者通过启动选项设置，它们的值可以使用SHOW VARIABLES命令，或者调用带变量的mysqladmin命令检索。因为SHOW VARIABLES的输出可能很大，所以 MySQL 允许按照下列方式增加一个 LIKE 语句，来过滤掉一部分内容，只得到需要的变量。如：

```
mysql> SHOW VARIABLES LIKE 'table_type';
+---------------+--------+
| Variable_name | Value  |
+---------------+--------+
| table_type    | INNODB |
+---------------+--------+
1 row in set (0.01 sec)

mysql> SHOW VARIABLES LIKE '%innodb%';
+---------------------------------+-------------------------+
| Variable_name                   | Value                   |
+---------------------------------+-------------------------+
| have_innodb                     | YES                     |
| innodb_additional_mem_pool_size | 1048576                 |
| innodb_buffer_pool_size         | 8388608                 |
| innodb_data_file_path           | ibdata1:10M:autoextend  |
| innodb_data_home_dir            |                         |
| innodb_file_io_threads          | 4                       |
| innodb_force_recovery           | 0                       |
| innodb_thread_concurrency       | 8                       |
| innodb_flush_log_at_trx_commit  | 1                       |
```

```
| innodb_fast_shutdown          | ON                    |
| innodb_flush_method           |                       |
| innodb_lock_wait_timeout      | 50                    |
| innodb_log_arch_dir           | ./                    |
| innodb_log_archive            | OFF                   |
| innodb_log_buffer_size        | 1048576               |
| innodb_log_file_size          | 5242880               |
| innodb_log_files_in_group     | 2                     |
| innodb_log_group_home_dir     | ./                    |
| innodb_mirrored_log_groups    | 1                     |
| innodb_max_dirty_pages_pct    | 90                    |
+-------------------------------+-----------------------+
20 rows in set (0.00 sec)
```

同时还可以使用下列方式用 SELECT@@*variable* 的句法得到系统变量的值：

```
mysql> SELECT @@tx_isolation;
+-----------------+
| @@tx_isolation  |
+-----------------+
| REPEATABLE-READ |
+-----------------+
1 row in set (0.00 sec)
```

13.2.5　使用错误日志排错

在启动服务器存在困难或运行中出现错误的情况下，一定要检查 MySQL 错误日志来查找错误原因。正如下面简单的示例，这个日志文件保存了服务器启动和关闭的信息，包括关键错误信息和有关受到损坏的表的警告列表。

```
030919 18:17:16  Fatal error: Can't open privilege tables: Can't find file:
'./mysql/host.frm' (errno: 13)
030919 18:17:16  Aborting

030919 17:03:37  mysqld ended

030919 17:06:13  mysqld started
030919 17:06:13  InnoDB: Started
/usr/local/mysql/bin/mysqld: ready for connections.
Version: '4.0.15-standard'  socket: '/tmp/mysql.sock'  port: 3306
030926 17:40:31  Found invalid password for user: 'root'@'%'; Ignoring user
030926 17:40:31  Found invalid password for user:
'root'@'localhost.localdomain'; Ignoring user
030926 17:43:51  /usr/local/mysql/bin/mysqld: Normal shutdown

030926 17:43:52  InnoDB: Starting shutdown...
030926 17:43:53  InnoDB: Shutdown completed
030926 17:43:53  /usr/local/mysql/bin/mysqld: Shutdown Complete
```

默认情况下，这个文件在 UNIX 中称为 hostname.err，在 Windows 中称为 mysql.err，一般存放在 MySQL data 目录下面。在 MySQL 4.0.10 和以后的版本中，可以为错误日志文件确定一个定制位置，方法是在 mysqld 中添加 --log-error 参数，参阅下面示例：

```
[root@host]# /usr/local/mysql/bin/mysqld_safe --log-error=/tmp/mysqld.errors
--user=mysql &
```

有关如何修复被破坏表的内容请参阅第15章，因为这方面的信息存放在MySQL维护的其他日志文件中。

13.3 小结

本章简要介绍了MySQL数据库的管理，概括了数据库管理员需要执行的最常见任务，同时简要介绍了完成这些任务需要使用的MySQL工具。其中主要工具是mysqladmin实用程序，这个工具可以重载或关闭服务器；查看活动进程列表，得到服务器变量的当前值。在管理工作中将要使用这个工具执行很多工作。

本章接着还讨论了MySQL管理工作中的一些特别的基本任务，包括启动和停止MySQL服务器，配置服务器在引导时自动启动，获取服务器状态，管理服务器过程，通过一个配置文件或SET命令修改服务器配置，使用MySQL错误日志排除错误等。

当然这还仅仅是冰山之一角。MySQL为数据库管理员提供很多强大的帮助功能，以便能够保证服务器的效率和效益。在后面一些章节中将会深入体验这些功能。

第14章　安全、访问控制和权限

大多数用户集中在MySQL的数据库和表上面，总之可以认为这是绝大多数行为发生的位置，但是人们通常不能很深入地掌握如何处理访问权限、口令和安全等问题。使用这种方法通常能够足以进行大多数开发活动，除非你是偶然作为数据库管理员，其工作将涉及设置和保证数据库的安全，防止未经授权的使用和恶意破坏等。

应该记住，本章将涉及MySQL访问控制系统，并讨论MySQL授权表的某些内容。这些表作为服务器安全系统的一个整体部分，可以为数据库管理员提供功能更为强大、更加灵活的管理访问系统的原则。本章还讨论在MySQL访问控制系统中用户账户和口令的管理，解释口令（尤其是最重要的根口令）如何进行修改和如何重新设置丢失的超级用户口令等内容。

14.1　MySQL授权表

当第一次安装MySQL的时候，MySQL的安装程序将自动创建两个数据库：为新用户提供练习使用的模拟test数据库和包括5个MySQL授权表的mysql数据库。

从下列的示例输出可以明显看出它们的情况：

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+
2 rows in set (0.00 sec)

mysql> SHOW tables FROM mysql;
+-----------------+
| Tables_in_mysql |
+-----------------+
| columns_priv    |
| db              |
| func            |
| host            |
| tables_priv     |
| user            |
+-----------------+
6 rows in set (0.00 sec)
```

除了func表外（该表用于用户定义的功能），每个表都有不同的确定用户是否可以访问特定数据库、表或列的作用。可以根据用户名、连接主机或申请的数据库来建立访问规则。

下面各节将更加详细地讨论每种表。

14.1.1 user 表

在 5 个授权表中，最重要的表是 user 表。下面是该表包含的字段列表。

```
CREATE TABLE `user` (
  `Host` varchar(60) binary NOT NULL default '',
  `User` varchar(16) binary NOT NULL default '',
  `Password` varchar(45) 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'
```

前三个字段（称为范围字段）定义了哪些用户允许连接到数据库服务器、口令以及可以连接的主机，MySQL 使用用户加主机标识组合作为其安全系统的基础。

假设从下列表中提取：

Host	User	Password
turkey.domain.com	john	

这暗示用户 john（无口令）允许从主机 turkey.domain.com 进行连接。

也可以使用 IP 地址代替主机名。下面也是一个完全有效的项目：

```
+--------------------+--------+----------+
| Host               | User   | Password |
+--------------------+--------+----------+
| 240.56.78.99       | john   |          |
+--------------------+--------+----------+
```

用户名是什么

注意，在 Windows 或 UNIX 中，MySQL 用户与系统用户并不相同。MySQL 用户只在 MySQL RDBMS 中存在，并且在系统中不需要账户或主目录。在 UNIX 中的 MySQL 命令行客户机，默认使用当前登录的用户名连接服务器的时候，这种情况可以通过 --user 参数指定用户名进行覆盖。

在设置这种访问规则时，还有可能指定通配符。下列示例允许以一个名为 john 的用户访问，不论连接申请哪个主机：

```
+-----------------+--------+
| Host   | User | Password  |
+-----------------+--------+
| %      | john |           |
+-----------------+--------+
```

字符 % 被用做通配符。下列示例在与 loudbeep.com 域名中的任何主机连接时可以匹配任何名为 john 的用户：

```
+---------------+-----+----------+
| Host          | User | Password  |
+---------------+-----+----------+
| %.loudbeep.com     | john |     |
+---------------+-----+----------+
```

一旦用户、口令和主机被指定，有必要确定每个用户享有的权限，在这里增加了另外 21 个列（或权限字段）。表 14.1 介绍了这个字段中每个项目代表的意思。

表 14.1　MySQL 权限等级

字段	权限名	拥有该权限的用户可以执行的动作
Select_priv	SELECT	在表的行中检索，进行 SELECT 查询
Insert_priv	INSERT	进行 INSERT 操作
Update_priv	UPDATE	进行 UPDATE 操作
Delete_priv	DELETE	进行 DELETE 操作
Create_priv	CREATE	创建数据库和表
Drop_priv	DROP	DROP 数据库和表
Reload_priv	RELOAD	重载 / 刷新 MySQL 服务器
Shutdown_priv	SHUTDOWN	关闭正在运行的 MySQL 服务器
Process_priv	PROCESS	在 MySQL 服务器上跟踪一项活动
File_priv	FILE	在服务器上读和写文件
Grant_priv	GRANT	授权其他用户享有该用户享有的权限
Index_priv	INDEX	创建、编辑和删除表索引
Alter_priv	ALTER	修改表

（续表）

字段	权限名	拥有该权限的用户可以执行的动作
References_priv	REFERENCES	创建、编辑和删除外键引用
Show_db_priv	SHOW DATABASES	在服务器上浏览可以使用的数据库
Super_priv	SUPER	执行管理命令
Create_tmp_table_priv	CREATE TEMPORARY TABLES	创建临时表
Lock_tables_priv	LOCK TABLES	创建和删除表锁
Execute_priv	EXECUTE	执行被保存的程序
Repl_slave_priv	REPLICATION SLAVE	在复制内容中读主二进制日志
Repl_client_priv	REPLICATION CLIENT	在复制内容中申请主、副信息

user 表中其余字段均与 SSL 加密和每个用户使用资源限制有关，部分内容将在 14.2.1 节中介绍。

此时，请注意在 user 表中赋予每个用户的安全权限全部有效，适用于系统的每个数据库。因此，记录：

```
+-----------------+------+-----------+--------------+
| Host            | User | Password  | Delete_priv  |
+-----------------+------+-----------+--------------+
| apple.pie.com   | joe  | secret    | Y            |
+-----------------+------+-----------+--------------+
```

将暗示用户 joe 有能力在服务器中删除任何数据库中任何表上的记录，如果 joe 当时情绪不佳，这种情况就不好了。为此，大多数管理员（和 MySQL 手册）建议对每个用户将该表中的所有权限设置成 N（默认值），并且使用 host 和 db 表设置更集中的访问等级（host 表和 db 表将在 14.1.2 节中讨论）。

提示：是否清楚了如何创建用户和维护用户口令？有关详细内容请参阅 14.2 节和 14.3 节。

下面是另一个示例，其中的记录说明存在一个名为 superapple 的超级用户，完全有权限全面访问所有 MySQL 资源：

```
+-----------------+-------------+-----------+----------+
| Host            | User        | Password  | All_priv |
+-----------------+-------------+-----------+----------+
| apple.pie.com   | superapple  | secret    | Y        |
+-----------------+-------------+-----------+----------+
```

注释：在任何授权表中不能出现 All_priv 列。本示例中简单地列举这个列是为了说明在所有权限列（*_priv）中存在相同的值。

14.1.2 db 表和 host 表

db表和host表在一起使用，控制用户可以使用哪些数据库，在这些数据库中可以使用哪些操作。参阅下列一个标准 db 表中的字段：

```
CREATE TABLE `db` (
  `Host` char(60) binary NOT NULL default '',
  `Db` char(64) binary NOT NULL default '',
  `User` char(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',
  `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',
  `Create_tmp_table_priv` enum('N','Y') NOT NULL default 'N',
  `Lock_tables_priv` enum('N','Y') NOT NULL default 'N',
  PRIMARY KEY  (`Host`,`Db`,`User`),
  KEY `User` (`User`)
) TYPE=MyISAM COMMENT='Database privileges'
```

同样前三个字段是范围字段，可以连接一个具体用户和主机到一个或多个数据库。其余的权限字段用来确定用户在命名数据库中可以执行的操作类型（每个字段的意义请参阅表14.1）。

类似下列示例的记录说明了从主机 cranberry.domain.com 连接的用户 bill 只能使用数据库 darkbeast：

```
+-----------------------+------+-----------+---------+
| Host                  |  User | Db        | All_priv|
+-----------------------+------+-----------+---------+
| cranberry.domain.com  | bill | darkbeast | Y       |
+-----------------------+------+-----------+---------+
```

另一方面，在下面的记录中说明来自任何主机连接的任何用户全部可以访问 test 数据库：

```
+------+------+---------+----------+
| Host | User | Db      | All_priv |
+------+------+---------+----------+
| %    |      | test    | Y        |
+------+------+---------+----------+
```

db 表中 Host 字段如果为空，说明得到许可的主机列表应该通过第三个表（host 表）得到，方法如下：

```
CREATE TABLE `host` (
  `Host` char(60) binary NOT NULL default '',
  `Db` char(64) 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',
  `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',
```

```
  `Create_tmp_table_priv` enum('N','Y') NOT NULL default 'N',
  `Lock_tables_priv` enum('N','Y') NOT NULL default 'N',
  PRIMARY KEY  (`Host`, `Db`)
) TYPE=MyISAM COMMENT='Host privileges;  Merged with database privileges'
```

主机记录和数据库记录之间的这种分离远要比人们想像的要更为有用。在缺少host表的情况下，如果希望相同的用户根据来自不同连接的主机享有不同的权限，就需要在db表中为每个主机创建一个单独的记录，并且赋予相应的权限。然而因为host表的存在，就可以在host表中设置不同的主机名，使之连接db表中各单独的表项（Host字段为空），然后以每个主机为基础设置权限。当有一个命名主机试图进行连接时，MySQL将基于host表中对该主机设置的规则赋予这些权限。

下面是如何工作的一个示例（第一部分是来自db表，第二部分是在host表中相应的记录）：

```
+------+------+-------+
| Host | User | Db    |
+------+------+-------+
|      | jim  | title |
+------+------+-------+
+-------------------+-------+----------------+----------------+
| Host              | Db    | Select_priv    | Insert_priv    |
+-------------------+-------+----------------+----------------+
| turkey.ix6.com    | title | Y              | Y              |
+-------------------+-------+----------------+----------------+
| blackbox.glue.net | title | Y              | N              |
+-------------------+-------+----------------+----------------+
| fireball.home.net | title | Y              | Y              |
+-------------------+-------+----------------+----------------+
```

在这个示例中，jim能够从host表中记录的任何主机连接到MySQL服务器，但是所享有的权限将根据使用连接的主机不同有所不同。

14.1.3 tables_priv 表和 columns_priv 表

除了前面讨论的三个表以外，在MySQL更新的版本中还增加了两个额外的表：tables_priv表和columns_priv表。这些表允许数据库管理员分别限制在数据库中访问具体的表和一个表中具体的列。

下面是tables_priv表的具体情况：

```
CREATE TABLE `tables_priv` (
  `Host` char(60) binary NOT NULL default '',
  `Db` char(64) binary NOT NULL default '',
  `User` char(16) binary NOT NULL default '',
  `Table_name` char(60) binary NOT NULL default '',
  `Grantor` char(77) NOT NULL default '',
  `Timestamp` timestamp(14) NOT NULL,
  `Table_priv` set('Select', 'Insert', 'Update', 'Delete', 'Create',
'Drop', 'Grant', 'References', 'Index', 'Alter') NOT NULL default '',
  `Column_priv` set('Select', 'Insert', 'Update', 'References')
NOT NULL default '',
```

```
  PRIMARY KEY  (`Host`, `Db`, `User`, `Table_name`),
  KEY `Grantor` (`Grantor`)
) TYPE=MyISAM COMMENT='Table privileges'
```

下列记录将仅仅限制用户 john 在表 cream 中执行 SELECT 操作，在同一数据库中其他表中运行 SELECT 查询的任何企图将会显示错误信息：

```
+-----------------+-------+------+------------+---------------+
| Host            | Db    | User | Table_name | Table_priv    |
+-----------------+-------+------+------------+---------------+
|lost.soul.com    | db563 | john | cream      | Select        |
+-----------------+-------+------+------------+---------------+
```

同样，下面的记录将允许用户 logger 在数据库 db1 中名为 logs 的表上面执行 SELECT，INSERT 和 UPDATE 查询（但是不能进行 DELETE 操作）：

```
+-----------+-----+----------+------------+---------------------+
| Host      | Db  | User     | Table_name | Table_priv          |
+-----------+-----+----------+------------+---------------------+
| localhost | db1 | logger   | logs       | Select,Insert,Update |
+-----------+-----+----------+------------+---------------------+
```

如果需要进行更精细的控制，MySQL 还提供了一个 columns_priv 表，可以以一个单独列为基础设置可能的访问权限，请参阅以下表结构：

```
CREATE TABLE `columns_priv` (
  `Host` char(60) binary NOT NULL default '',
  `Db` char(64) binary NOT NULL default '',
  `User` char(16) binary NOT NULL default '',
  `Table_name` char(64) binary NOT NULL default '',
  `Column_name` char(64) binary NOT NULL default '',
  `Timestamp` timestamp(14) NOT NULL,
  `Column_priv` set('Select', 'Insert', 'Update', 'References')
NOT NULL default '',
  PRIMARY KEY  (`Host`, `Db`, `User`, `Table_name`, `Column_name`)
) TYPE=MyISAM COMMENT='Column privileges'
```

下面的示例列举这种情况。下面的规则说明使用账户 hr_users 登录的用户只能读 db1.employees 表中的雇员 ID、雇员名和部门三个列，而管理员（使用账户 hr_supervisors）可以浏览雇员 ID、雇员名和部门三个列，并可以更新雇员名和部门信息。

```
+-----+----------------+------------+-------------+----------------+
| Db  | User           | Table_name | Column_name |  Column_priv   |
+-----+----------------+------------+-------------+----------------+
| db1 | hr_supervisors | employees  | name        | Select, Update |
| db1 | hr_supervisors | employees  | id          | Select         |
| db1 | hr_supervisors | employees  | dept        | Select, Update |
| db1 | hr_users       | employees  | id          | Select         |
| db1 | hr_users       | employees  | name        | Select         |
| db1 | hr_users       | employees  | dept        | Select         |
+-----+----------------+------------+-------------+----------------+
```

授权表之间的关系

不同的授权表在创建广泛的访问规则时会相互影响，使MySQL可以用来确定什么时候如何处理具体用户的申请。访问控制可以作用两个阶段：连接阶段和申请阶段。

- **连接阶段** 当用户通过具体主机申请与数据库服务器的一次连接时，MySQL首先将检查该用户在user表中是否存在这个表项，用户的口令是否正确，并且该用户是否得到许可可以通过该主机进行连接。如果检查正确，将得到许可能够连接服务器。
- **申请阶段** 一旦连接得到许可，后来向服务器提出的申请（SELECT，DELETE，UPDATE以及其他查询）将首先得到处理，确保用户具有执行相应行动需要的安全权限。很多不同等级访问都是可能的，有些用户也许只能具有该表的SELECT能力，而其他则具有INSERT和UPDATE能力，但是还是不具有DELETE能力。

在MySQL授权表的结构中，user表在最前面，下面是db表和host表，最后才是tables_priv表和columns_priv表。低等级的表只能从高等级的表得到必要的范围或权限。

当MySQL决定是否允许进行具体的数据库操作时，将在全部三个表中考虑处理权限字段。开始是使用user表，并检查用户是否已经得到执行预计操作的适当权限。如果没有，db和host表将检查是否可以使用这种权限。只有在逻辑分析了不同表中的权限之后，MySQL才允许或拒绝具体的数据库申请。

当MySQL遇到管理活动方面的申请（RELOAD，PROCESS等）时，需要决定是否允许这种只能在user表中得到相应许可的行动。其他授权表不能批准这种决定。这是因为这些管理方面的权限适用于整个系统，而不是仅仅适合具体的数据库或表，所以user表只能批准相应的列。

默认设置

现在看一下MySQL中带的默认表，这样就可以理解如何运行一种例外的MySQL设置：

```
mysql> SELECT Host, User, Password FROM mysql.user;
+-----------+------+----------+
| Host      | User | Password |
+-----------+------+----------+
| localhost | root |          |
| localhost |      |          |
+-----------+------+----------+
2 rows in set (0.00 sec)

mysql> SELECT Host, Db, User FROM mysql.db;
+------+---------+------+
| Host | Db      | User |
+------+---------+------+
| %    | test    |      |
| %    | test\_% |      |
+------+---------+------+
```

“例外”的意思包括：

1. MySQL赋予用户通过本地设备localhost以root的形式进行连接，完全可以访问系统中的所有数据库。

2. MySQL 赋予任何其他用户通过本地设备进行连接，在 Windows 中完全访问所有数据库或在 UNIX 上的 test 数据库。
3. 对来自其他主机的用户则拒绝访问。

14.2　授权、废除和浏览用户权限

在 mysql 数据库中修改授权表，需要能够访问到数据库服务器的超级用户。所以这方面的第一个要素是确保自己已经得到这个级别的许可，并且可以修改表记录。

如果已经在自己的开发设备上安装好了服务器，在安装的时候应该已经被要求输入根口令（这个过程详见第3章）。如果已经注意到第3章介绍的安装说明，应该已经完成了口令设置（将口令留空的做法将给系统留下安全漏洞），并且要记下已经设置好的口令作为以后的参考。

如果需要确认已经得到必要的访问权，可以按照下列方式以 root 用户的方式登录服务器：

```
[root@host] $ mysql -u root -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
mysql>
```

现在确保已经可以在 mysql 数据库中访问授权表：

```
mysql> USE mysql;
Database changed
```

这种根等级的访问通常只能允许数据库管理员得到。其他用户得到的安全等级比较低，所以访问范围有限。每个“一般性”的用户通常是提供了自己的用户名和口令之后才能连接数据库。如前面章节所述，MySQL 授权表的目的是为那些一般性用户提供安全设置的可能，并能够定制每个用户的访问等级到比较细的范围。

MySQL 提供两种方法修改授权表中的访问权限，可以使用 INSERT，UPDATE 和 DELETE DML 查询来手工修改表中的信息，或者使用 GRANT 和 REVOKE 命令。后者的方法更好。直接修改授权表比较明智，但是只能针对非常规任务或情况，所以通常不推荐这么做。

14.2.1　使用 GRANT 和 REVOKE 命令

MySQL 授权表中为用户设置访问权限建议使用的方法是使用 GRANT 和 REVOKE 命令，这些命令是专门针对这种任务设计的。以下是这些命令的用法：

```
GRANT privilege (field-name, field-name, ...), privilege (field-name, field-
name, ...), ... ON database-name.table-name TO user@domain IDENTIFIED BY
password, user@domain IDENTIFIED BY password, ...
REVOKE privilege (field-name, field-name, ...), privilege (field-name, field-
name, ...), ... ON database-name.table-name FROM user@domain, user@domain, ...
```

为了举例说明，让我们看几个示例。第一个示例是在 db1.logs 表中为通过本地主机，使用口令 timber 连接的用户 logger 设置 SELECT，INSERT，UPDATE 和 DELETE 权限：

```
mysql> GRANT SELECT, INSERT, UPDATE ON db1.logs TO logger@localhost IDENTI-
FIED BY 'timber';
```

现在可以看到当这个用户登录到MySQL上面，并试图进行不同类型的查询的时候会发生什么：

```
[user@host] $ mysql -h localhost -u logger -p
Enter password: ******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 4.0.14
Type 'help;' or '\h' for help. Type '\c' to clear the buffer

mysql> USE mysql;
ERROR 1044: Access denied for user: 'logger@localhost' to database 'mysql'

mysql> USE db1;
Database changed

mysql> SELECT * FROM employees;
ERROR 1142: select command denied to user: 'logger@localhost' for table 'em-
ployees'

mysql> SELECT * FROM logs;
+----+----------------------------+--------+
| id | message                    | sender |
+----+----------------------------+--------+
| 34 | Apache core error          | httpd  |
| 35 | Root login failure on tty1 | pamd   |
| 36 | Root login failure on tty1 | pamd   |
+----+----------------------------+--------+
3 rows in set (0.00 sec)

mysql> INSERT INTO logs VALUES (37, 'Sendmail restart, 102 messages in queue',
'sendmail');
Query OK, 1 row affected (0.00 sec)

mysql> DELETE FROM logs WHERE id = 37;
ERROR 1142: delete command denied to user: 'logger@localhost' for table 'logs'
```

在此只有在GRANT命令中指定的命令才允许该用户使用，所有其他命令都被拒绝。每次用户申请一个具体命令时，MySQL将引用其授权表，并且如果权限原则得到许可，只有得到许可的命令才可以执行。

REVOKE命令是GRANT命令的反义，它可以撤销赋予用户的权限。请参阅下面示例，这是废除用户 logger 在 db1.logs 表中 INSERT 和 UPDATE 的权限：

```
mysql> REVOKE INSERT, UPDATE ON db1.logs FROM logger@localhost;
```

这个改变将立即生效。现在当用户logger试图向表中插入一个新记录时（这个操作以前是允许的），将发生以下情况：

```
mysql> INSERT INTO logs VALUES (38, 'System powerdown signal received', 'apmd');
ERROR 1142: insert command denied to user: 'logger@localhost' for table 'logs'
```

在数据库和表中，MySQL 允许使用通配符，接下来的查询赋予主机名为 medusa 的用户 admin 在所有数据库上的 RELOAD，PROCESS，SELECT，DELETE 和 INSERT 权限：

```
mysql> GRANT RELOAD, PROCESS, SELECT, DELETE, INSERT ON *.* TO admin@medusa
IDENTIFIED BY 'secret';
```

接下来的示例只给用户 supervisor 在 employees. compensation 表中赋予 SELECT 权限：

```
mysql> GRANT SELECT ON employees.compensation TO supervisor;
```

下一个示例进一步进行设置，分别赋予用户harry和john在表grades中具体列下的SELECT和 UPDATE 权限：

```
mysql> GRANT SELECT (id, name, subj, grade) ON db1.grades TO harry;

mysql> GRANT SELECT (id, name, subj, grade), UPDATE (name, grade) ON db1.
grades TO john;
Query OK, 0 rows affected (0.00 sec)
```

下面的示例取消了用户 tim 在数据库 db2003a 中的 CREATE 和 DROP 权限：

```
mysql> REVOKE CREATE, DROP ON db2003a.* FROM tim@funhouse.com;
```

正确处理表

在GRANT命令中命名的表和字段必须存在赋予相应表和列等级权限的先决条件。然而，这个原则在处理数据库等级权限时不能奏效。MySQL 允许赋予数据库等级的权限，即使不存在相应的数据库。这种在表和数据库等级权限方面的不协调情况成为新手犯错误的常见原因，所以一定要注意！

对数据库的权限设置和取消同样适应于表和列。接下来的示例是取消用户 webmaster 在表 menu 表中的 DELETE 权限：

```
mysql> REVOKE DELETE ON www.menu FROM webmaster@localhost;
```

下面是取消用户 sarah 在数据库 customer 的 name 和 address 列中进行 UPDATE 的权限：

```
mysql> REVOKE UPDATE (name, address) ON sales.customers FROM sarah@work.
domain.net;
```

MySQL 还提供了一个 ALL 权限等级作为“全部权限”的快捷方式。这样可以帮助更加简便地设置 GRANT 和 REVOKE 语句。下面的示例进行了说明，为在 melonfire.com 域中任何主机连接的用户 www 在数据库 web 上面赋予了全部权限：

```
mysql> GRANT ALL ON web.* TO www@'%.melonfire.com' IDENTIFIED BY 'abracadabra';
```

使用相同方法，也可以使用 REVOKE 命令取消特定用户的所有权限：

```
mysql> REVOKE ALL ON web.* FROM www@'%.melonfire.com';
```

MySQL 还为需要创建用户、但是并不赋予其任何权限的情况提供了专门的 USAGE 权限。参阅下列两个示例：

```
mysql> GRANT USAGE ON content.* TO joe@localhost;

mysql> GRANT USAGE ON *.* TO test@some.domain.com;
```

GRANT 权限

MySQL 允许用户通过 GRANT 命令中 WITH GRANT OPTION 的特殊语句授权给其他用户自己所拥有的相同的权限。当把这个语句添加到 GRANT 命令中时，使用这个命令的用户可以把自己所拥有的相同的权限赋予其他用户。参阅下列示例：

```
mysql> GRANT SELECT, DELETE, INSERT, UPDATE, CREATE, DROP, INDEX ON inventory.
* TO david@localhost WITH GRANT OPTION;
```

中间的通配符

在 GRANT 和 REVOKE 命令中的主机名或数据库名上使用像“%”和“_”类似的通配符时，MySQL 要求对这个名称使用括号。但是通配符不支持用户名。如果要创建一个匿名用户，用户名可以是空字符串，如‘’。

手工操作

当使用 GRANT 命令调用具体用户时，如果项目并不存在，则在 user 表中为该用户自动创建一个项目。但是使用 REVOKE 命令不能在 user 表中删除这个项目，即使删除的结果会导致所有用户权限被除去。因此，虽然一个用户的记录可以通过 GRANT 命令自动添加到系统，但是使用 REVOKE 命令不能自动进行删除。用户记录必须使用 DELETE 查询手工删除。

现在用户 david@localhost 已经得到了 GRANT 权限，并且可以将其权利赋予其他用户，从下列部分示例可以清楚地看到其结果：

```
mysql> SELECT Host, Db, User, Grant_priv FROM db WHERE Host = 'localhost';
+-----------+-----------+-------+------------+
| Host      | Db        | User  | Grant_priv |
+-----------+-----------+-------+------------+
| localhost | inventory | david | Y          |
+-----------+-----------+-------+------------+
1 row in set (0.00 sec)
```

现在用户 david@localhost 可以登录到 MySQL 上，并且授权给其他用户其所拥有的全部或部分权限，方法如下：

```
mysql> GRANT SELECT ON inventory.* TO joe@localhost IDENTIFIED BY 'joey2839';

mysql> SELECT Host, Db, User, Grant_priv FROM db WHERE Host = 'localhost';
+-----------+-----------+-------+------------+
| Host      | Db        | User  | Grant_priv |
+-----------+-----------+-------+------------+
| localhost | inventory | david | Y          |
| localhost | inventory | joe   | Y          |
+-----------+-----------+-------+------------+
2 rows in set (0.00 sec)
```

在标准的 REVOKE 命令中，使用 GRANT OPTION 语句可以修改 GRANT 权限，方法如下：

```
mysql> REVOKE GRANT OPTION ON inventory.* FROM david@localhost;
```

现在用户 david@localhost 不再能够授权给其他用户权限了。从 user 表部分和自己运行的 GRANT 命令输出中可以清楚地看到，参阅下列内容：

```
mysql> GRANT SELECT ON inventory.* TO joe@localhost IDENTIFIED BY 'joey2839';
ERROR 1044: Access denied for user: 'david@localhost' to database 'inventory'

mysql> SELECT Host, Db, User, Grant_priv FROM db WHERE User = 'david';
+-----------+-----------+-------+------------+
| Host      | Db        | User  | Grant_priv |
+-----------+-----------+-------+------------+
| localhost | inventory | david | N          |
+-----------+-----------+-------+------------+
2 rows in set (0.00 sec)
```

赋予用户 GRANT 权限时要特别小心。具有不同访问等级的用户可以合并在一起，因此得到比正常情况下可以得到更高等级的访问权。

限制资源使用

在 MySQL 4.x 中的新功能是能够以每个用户为基础，限制 MySQL 服务器的资源使用。其方法是通过在 user 表中增加三个新字段：max_questions，max_updates 和 max_connections，可以用来分别限制具体用户每个小时内的查询、表或记录更新和新连接的数量。这三个字段可以在 GRANT 命令中形成三个选项语句。

第一个语句是 MAX_QUERIES_PER_HOUR 语句，用来限制用户每小时运行的查询数量。参阅下列示例：

```
mysql> GRANT SELECT ON *.* TO sarah WITH MAX_QUERIES_PER_HOUR 5;
```

MAX_QUERIES_PER_HOUR语句可以控制每小时得到许可查询的总量，不论是SELECT，INSERT，UPDATE，DELETE 查询，还是其他查询。如果这么做过于概括，还可以通过 MAX_UPDATES_PER_HOUR 语句来设置对修改数据库数据查询数量的限制，参阅下例：

```
mysql> GRANT SELECT ON *.* TO sarah WITH MAX_UPDATES_PER_HOUR 5;
```

通过MAX_CONNECTIONS_PER_HOUR语句可以控制每小时用户打开新连接的数量，参阅下例：

```
mysql> GRANT SELECT ON *.* TO sarah WITH MAX_CONNECTIONS_PER_HOUR 3;
```

还可以综合使用这些语句。下列示例是一个完全有效的 GRANT 语句：

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO supervisor WITH
MAX_QUERIES_PER_HOUR 50 MAX_UPDATES_PER_HOUR 10 MAX_CONNECTIONS_PER_HOUR 4;
```

注意这种使用限制不能针对具体数据库或表。在GRANT命令中用ON *.*语句只能针对全部内容。

下面是一个 MySQL 说明资源限制已超过的具体示例：

```
mysql> INSERT INTO customers (id, name) VALUES (2892, 'Iola J');
ERROR 1226: User 'sarah' has exceeded the 'max_questions' resource (current
value: 5)
```

服务器以用户为单位有一个内部计数器，可以计算三种资源限制的每项要求。这些计数器可以使用新命令 FLUSH USER_RESOURCES 在任何时间内重置，参阅下列示例：

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

注释：执行 FLUSH 命令需要有 RELOAD 权限。

14.2.2 使用 INSERT，UPDATE 和 DELETE 命令

前面章节中介绍的GRANT和REVOKE命令都是在MySQL的授权表中进行修改的推荐方法。然而因为授权表完全属于MySQL的常规表，所以还需要使用标准的INSERT，UPDATE和DELETE查询操作它们。

因此一定能使用下列 GRANT 命令：

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE, ON recipes.* TO tom@localhost
IDENTIFIED BY 'tommygun';
```

可以使用下列两个（同义的）INSERT 命令完成相同的任务：

```
mysql> INSERT INTO user (Host, User, Password) VALUES('localhost', 'tom',
PASSWORD('tommygun'));

mysql> INSERT INTO db (Host, Db, User, Select_priv, Insert_priv, Update_priv,
Delete_priv, Create_priv, Drop_priv) VALUES ('localhost', 'recipes', 'tom',
'Y', 'Y', 'Y', 'Y', 'N', 'N');

mysql> FLUSH PRIVILEGES;
```

同样在使用下列 GRANT 命令设置一个管理用户时：

```
mysql> GRANT ALL PRIVILEGES ON *.* TO admin@localhost IDENTIFIED BY 'master';
```

也可以使用下列的 INSERT 设置：

```
mysql> INSERT INTO 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)
VALUES ('localhost', 'admin', PASSWORD('master'), 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y');

mysql> FLUSH PRIVILEGES;
```

14.2.3 浏览权限

MySQL允许使用SHOW GRANTS命令来浏览赋予具体用户的权限，使用用户名作为变量，并显示该用户授予的所有权限列表。参阅以下示例：

```
mysql> SHOW GRANTS FOR sarah@localhost;
+-----------------------------------------+
| Grants for sarah@localhost              |
+-----------------------------------------+
| GRANT USAGE ON *.* TO 'sarah'@'localhost'  |
|  IDENTIFIED BY PASSWORD '4837a3954ee01ece' |
```

```
| GRANT SELECT ON sales.customers TO          |
| 'sarah'@'localhost'                         |
+---------------------------------------------+
2 rows in set (0.00 sec)

mysql> SHOW GRANTS FOR root;
+-------------------------------------------------------------+
| Grants for root@%                                           |
+-------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' WITH GRANT OPTION |
+-------------------------------------------------------------+
1 row in set (0.06 sec)
```

提示：MySQL 的 UNIX 发布包括一个名为 mysqlaccess 的脚本，可以用来生成一个具体用户访问等级和权限的报告。键入 mysqlaccess --howto 可以看到使用这个脚本的示例。

14.2.4　重载授权表

使用 GRANT 和 REVOKE 命令的权限可以立即被激活（正如前面章节列举的示例所示）。然而通过常规 SQL 查询设置的权限需要服务器重载才能生效。使用 FLUSH PRIVILEGES 命令可以重载服务器，具体方法如下：

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

注释：需要有 RELOAD 权限才能运行 FLUSH 命令。

MySQL 通常在启动时就读取授权表，但是也可以要求 MySQL 通过 mysqladmin 工具来重载授权表，参阅下列示例。这是一个 UNIX 示例，只要简单地按照安装 MySQL 时使用的正确路径，就可以替换成为 Windows 路径：

```
[root@host]# /usr/local/mysql/bin/mysqladmin -u root reload
```

14.2.5　重置授权表

如果希望重置授权表成为最初的默认设置，可以按照下列步骤操作：

1. 如果服务器正在运行，以正常方式停止运行：

```
[root@host] $ /usr/local/mysql/support-files/mysql.server stop
```

2. 修改 MySQL 安装的数据目录，然后删除 mysql 文件夹。因为 MySQL 中的数据库是在文件系统中表现为目录形式，所以这样操作可以有效地删除授权表：

```
[root@host] $ rm -rf /usr/local/mysql/data/mysql
```

3. 在 UNIX 中，运行初始化脚本 mysql_install_db，重新安装授权表，这个脚本是与程序一道提供的：

```
[root@host]# /usr/local/mysql/scripts/mysql_install_db
```

在 Windows 中，因为初始化脚本不是二进制发布的一部分，所以需要在相同目录中重新安装该包，来转换原始授权表。

4. 在UNIX中，更改MySQL安装的数据目录，并改变新创建的MySQL目录所有权，这样该目录将成为mysql用户拥有：

```
[root@host]# chown -R mysql.mysql /usr/local/mysql/data/mysql
```

5. 重新启动服务器：

```
[root@host] $ /usr/local/mysql/support-files/mysql.server start
```

现在的MySQL授权表将被重新设置到默认值上了。于是你可以作为超级用户登录，使用GRANT和REVOKE命令进行修改。

14.3 改变用户口令

当使用GRANT命令在user表中添加一个用户时，MySQL还允许使用IDENTIFIED BY选择语句为这个用户确定一个口令。用户在登录服务器，要求访问其数据库时必须使用这个口令。当使用mysql客户机程序时，用户名和口令可以通过-u和-p变量在命令行中分别指定，参阅下列示例：

```
[user@host] $ mysql -h localhost -u logger -p
Enter password: ******
Welcome to the MySQL monitor ...
```

如果不希望提示口令（例如，如果通过一个脚本以非对话方式使用客户机），可以在-p变量后面使用命令行语句确定，参阅下例：

```
[user@host] $ mysql -h localhost -u logger -ptimber
Welcome to the MySQL monitor ...
```

口令保存在MySQL数据库中user表的Password字段中，在插入之前必须用MySQL PASSWORD()函数加密，如下例所示：

```
mysql> INSERT INTO user (Host, User, Password) VALUES('melonfire.net',
'timothy', PASSWORD('1r0ck'));

mysql> UPDATE user SET Password = PASSWORD('1r0ck') WHERE Host = 'melonfire.
net' AND User = 'timothy';

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

也可以使用MySQL SET PASSWORD命令来设置口令。下列示例与前面的示例效果相同：

```
mysql> SET PASSWORD FOR timothy@melonfire.net = PASSWORD('1rock');
```

当使用GRANT命令的IDENTIFIED BY语句或通过mysqladmin工具设置口令时，不需要先使用PASSWORD()函数来加密口令字符串。GRANT命令和mysqladmin实用程序会自动予以处理。因此下列示例与前面的示例效果相同：

```
mysql> GRANT USAGE ON *.* TO timothy@melonfire.net IDENTIFIED BY '1rock';
```

第一种方法要求FLUSH PRIVILEGES启动工作，因此不推荐使用。推荐使用第二种方法。

GRANT命令的IDENTIFIED BY语句可以选择使用。不用这个语句，为新用户创建一个GRANT可以为其设置一个空口令。这样造成系统安全漏洞，所以管理员一般都应该赋予新用户一个口令。

安全第一

MySQL手册建议不要把口令放置在mysql命令行-p选项之后，而建议在口令对话提示时填写口令，在键入口令时会掩蔽它。这样将减少有人在旁边看到口令的风险，造成在MySQL服务器上面未经授权访问账户的情况。

升级旧版本时

MySQL 4.1和更新的版本中的PASSWORD()函数生成的密码更长，有41字节散列值，与老版本（使用16字节值）不兼容。因此当升级一个旧的版本到MySQL 4.1或更新的版本时，必须在MySQL安装的scripts目录中运行mysql_fix_privilege_tables脚本来升级授权表，以便能够处理更长的散列值。

当用户登录MySQL服务器，并且提供了一个口令字符串时，MySQL首先使用PASSWORD()函数加密所提供的口令字符串，然后把结果值与user表中的相应用户记录中Password字段的值进行比较。如果两个值匹配（其他访问规则许可它），用户被授权可以访问。如果不匹配，访问被拒绝。

14.3.1　设置 root 口令

在第一次安装MySQL时，访问数据库服务器局限于MySQL管理员，即root。默认情况下，用户初始化的口令为无，这种做法不好，因此需要尽快予以纠正。通过mysqladmin工具为这个用户尽可能快地设置口令，并且按照以下句法（这个示例针对UNIX系统，对于Windows来说，MySQL安装只要更换正确的路径即可）：

```
[root@host]# /usr/local/mysql/bin/mysqladmin -u root password 'new-password'
```

这个口令修改是立即生效的，不需要重新启动服务器或刷新privilege表。

使用UPDATE命令或SET PASSWORD命令，按照前面章节介绍的方法也可以修改口令。

14.3.2　重置 root 口令

如果忘记了MySQL的根口令，并且锁死了授权表，不必担心，按照下列步骤还可以重新恢复运行：

1. 以系统管理员身份（在UNIX上为root）登录到系统，终止MySQL服务器。可以在MySQL安装的support-files目录中使用mysql.server启动和关闭脚本达到这个目的，方法如下：

   ```
   [root@host] $ /usr/local/mysql/support-files/mysql.server stop
   ```

   如果是在UNIX系统下工作，它带有二进制的RPM安装或预先安装的MySQL，也可以使用/etc/rc.d/init.d/mysqld脚本停止（和启动）MySQL。
2. 接下来，使用--skip-grant-tables启动选项再次启动MySQL：

   ```
   [root@host] $ /usr/local/mysql/bin/safe_mysqld --skip-grant-tables
   --skip-networking
   ```

这样可以绕过授权表，作为MySQL根用户登录到服务器，不需要提供口令。额外的--skip-networking选项告诉MySQL不要侦听TCP/IP连接，并且保证不会在修正权限时在网络上被破解。

3. 按照前面章节介绍的方法，使用UPDATE命令，为MySQL根用户设置一个新口令：

```
[root@host]# mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 4.0.14
Type 'help;' or '\h' for help. Type '\c' to clear the buffer

mysql> USE mysql;
Database changed

mysql> UPDATE user SET Password = PASSWORD('new-password') WHERE User = 'root';
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

4. 退出服务器，关闭服务器，按照正常方式重新启动它：

```
[root@host] $ /usr/local/mysql/support-files/mysql.server stop
[root@host] $ /usr/local/mysql/support-files/mysql.server start
```

现在MySQL能够允许你使用第三步设置的新口令以根用户的身份进行登录。

注释：MySQL的联机手册中有关MySQL安全的更多内容可以从下列网址得到：

http://www.mysql.com/doc/en/General_security.html
http://www.mysql.com/doc/en/Privileges_provided.html

14.4 小结

MySQL备有5层访问控制系统，可以对每个用户得到的对数据库、表和具体字段的权限进行详细的控制。本章详细讨论了访问控制和权限系统的内容，验证了MySQL授权表，讲解了在管理用户权限中使用的GRANT和REVOKE命令。同时还验证了以单个用户为基础在服务器上限制资源使用的课题，以及修改用户口令、对超级用户丢失口令进行恢复以及重新设置授权表等内容，同时还提到了MySQL 4.x中可以使用的一些新的安全功能。

第15章　维护、备份和恢复

现在，我们已经知道MySQL相对比较容易使用，在多种生产环境下，可以认为是一种比较理想的数据库工具，它不需要技术高超的数据库管理员。尽管如此，不论安装规模多大，仍然有一定数量的基本维护需要进行。本章的目的是介绍一些这方面的基本知识，使它能够正确使用，并且在出现问题时提供建议。

15.1　维护

MySQL需要进行的大量维护工作是通过不同的日志文件进行的。日志对于需要排除故障的情况或者对于争取主动，事先避免发生问题来说都是十分基本的。保证数据库运行平稳有效的其他一些技术还包括在运行中进行检验、维护和优化各种表的技术等。本节将分别讨论这方面的内容。

15.1.1　日志记录

当MySQL服务器启动时，将要检验其所有日志记录选项是否被激活。如果已经激活，服务器启动将表明日志作为启动过程的一部分。日志文件提供了管理服务器所需要的信息。查阅这些日志的主要目的是分析性能和调查问题。文件保存在与数据文件相同的目录之中，通常在UNIX中是/usr/ local/mysql/data，在Windows中是c:\mysql\data。

虽然这些文件是完全标准的文本文件，但是日志文件有一些不同的类型：

- 错误日志
- 查询日志
- 慢查询日志
- 更新日志
- 二进制更新日志

错误日志

顾名思义，错误日志保存了服务器上发生的每个错误的记录。所以这个文件是一个基本的诊断工具，当服务器发生需要排除错误的问题时，它可以派上用场。

如果需要激活错误日志，添加--log-error选项到服务器启动命令行或选项文件中，参阅下例：

```
[root@host]# /usr/local/mysql/bin/mysqld_safe --log-error
```

下面是错误日志的一部分：

```
031024 15:19:08  InnoDB: Started
/usr/local/mysql/bin/mysqld: ready for connections.
Version: '4.0.15-standard'  socket: '/tmp/mysql.sock'  port: 3306
InnoDB: Setting log file ./ib_logfile0 size to 5 MB
```

```
InnoDB: Database physically writes the file full: wait...
031024 15:19:07  InnoDB: Log file ./ib_logfile1 did not exist: new to be
created
InnoDB: Setting log file ./ib_logfile1 size to 5 MB
InnoDB: Database physically writes the file full: wait...
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
031024 15:42:02  /usr/local/mysql/bin/mysqld: Normal shutdown
031024 15:42:02  InnoDB: Starting shutdown...
031024 15:42:04  InnoDB: Shutdown completed
031024 15:42:04  /usr/local/mysql/bin/mysqld: Shutdown Complete
```

这个日志的文件名一般都是以.err 扩展名为结尾的。

查询日志

查询日志是另一个有用的日志，因为它（令人震惊地）保存了客户机发给服务器的每个查询的踪迹。它还可以显示客户机连接服务器以及这些客户机所做操作的细节内容。如果希望监督以排错为目的的活动，就应该激活查询日志，方法是添加--log选项到服务器启动命令行或选项文件。

```
[root@host]# /usr/local/mysql/bin/mysqld_safe -log
```

下面是查询日志部分内容的示例：

```
Time                 Id Command    Argument
031116 18:10:26       1 Connect     root@localhost on
031116 18:10:32       1 Query       show databases
031116 18:10:35       1 Init DB     test
                      1 Query       show databases
                      1 Query       show tables
031116 18:10:38       1 Query       show tables
031116 18:10:45       1 Init DB     mysql
                      1 Query       show databases
                      1 Query       show tables
                      1 Field List  columns_priv
                      1 Field List  db
                      1 Field List  func
                      1 Field List  host
                      1 Field List  tables_priv
                      1 Field List  user
031116 18:10:48       1 Query       show tables
031116 18:10:52       1 Query       select * from user
031116 18:10:56       1 Quit
```

这个日志的默认文件名是主机名加.log 扩展名。

慢查询日志

慢查询日志也是一个有关日志的文件，可以列出超过预先设定时间量（根据long_query_time变量确定）的所有查询，任何超过这个值的查询都将列在这个日志中。如果需要查找优化性能的办法，这个日志是开始工作的一个好地方。

注释：查询优化内容将在第 16 章中详细讨论。

通常需要查看这个日志中的查询，作为进行修改的方案，以便减轻对服务器性能的影响。应该记住，虽然查询占用的时间长度可能是一些因素的后果，而不是程序中存在的问题。如果服务器存在问题比较多，通常占用时间很长的查询都会出现在这个日志中。

慢查询日志可以使用 mysqld 命令行中的 --log-slow-queries 选项来激活，参阅下列示例：

```
[root@host]# /usr/local/mysql/bin/mysqld_safe --log-slow-queries
```

该日志的默认文件名是主机名后带 -slow 的后缀以及.log 扩展名。

更新日志

涉及查询日志的另一个日志是更新日志，可以显示改变具体表的所有查询。与 UPDATE，CREATE TABLE 和 DROP TABLE 同时使用的 INSERT，REPLACE，DELETE，GRANT 和 REVOKE语句都属于这个类型，然而SELECT语句没有记录在内。更新日志的项目只有在修改发生之后才会产生，如果没有更新，不论什么原因，都不会被登记在内。

当启动 MySQL 服务器时，使用 --log-update 选项，更新日志便创建起来，参阅下面示例：

```
[root@host]# /usr/local/mysql/bin/mysqld_safe --log-update
```

如果需要重新构建一个表，面该表在最近一次备份之后又进行了修改，这个日志十分有用。在数据库受到破坏的情况下，可以根据备份进行恢复，然后按照更新日志重新创建记录到的查询，这样可以把系统恢复到受到破坏之前的状况。

二进制更新日志

在比较新的 MySQL 版本中（3.23.14 以后的版本）还可以支持以二进制格式保存的更新日志变体，而不是文本格式。这是一种更为有效的保存数据的格式，这种二进制更新日志还可以比标准更新日志记录得更详细。名为 mysqlbinlog 的实用程序可以把二进制日志转换回文本文件，这样就可以进行阅读。在启动 MySQL 之后，使用 --log-bin 选项就可以创建一个二进制更新日志了，参阅下面示例：

```
[root@host]# /usr/local/mysql/bin/mysqld_safe --log-bin
```

这个日志的默认文件名是主机名加 -bin 后缀，然后有一个分辨日志顺序的数字。

注释：如果设置了主副服务器以便进行复制，必须启动二进制更新日志（有关复制的内容详见第 17 章）。

如果需要刷新日志，可利用 FLUSH LOGS 命令。这个命令可以导致服务器关闭，然后重新打开日志文件。对于二进制日志来说，这个命令关闭了当前的日志，并使用一个新的顺序编号创建新日志，如果需要，旧日志可以归档。

当需要考虑日志截止和循环问题时，日志刷新的价值就更明显了，这方面的内容参阅下一节。

时间延误

更新作为事务的部分内容不会立即执行，文件将被保存在高速缓存中，直到该事务被提交。MySQL 服务器一旦收到 COMMIT 命令，整个事务首先被写到二进制日志中，然后把修改保存到数据库。如果事务的一部分没有执行，不论何种原因，整个事务都会被回滚，并且在二进制日志中不会写上该变化。

日志截止和循环

如果服务器很忙，这些日志很快将变得很大（会充满整个磁盘）。所以日志必须通过截止日期和循环进行管理，保证日志是一个助手，而不是累赘。

日志循环是缓解出现这种情况的一种方法。日志循环是通过创建一个日志文件的有限数字实现，然后进行覆盖替换，这样每次循环都会剔除最旧的文件。例如，如果有一个文件名为query_log，第一次循环发生，其文件重新命名为query_log.1，新的query_log文件被创建。在接下来的循环中，query_log.1被重新命名为query_log.2，query_log被重新命名为query_log.1，并且创建了一个新的query_log文件。当循环到达最后一轮时，最旧的文件被覆盖。可以保存信息的数量完全取决于循环的长短以及创建文件的数量。这些编号根据实际情况可以变化，但是常见的方式是每天创建新日志，在一个循环中使用7轮，每一轮为一周中的一天。

日志循环因为可以通过文件名分辨日志，不是通过编号，所以十分方便。但是如果使用的是数字编码的日志文件，按照前面介绍的方法重新命名文件可能会搞乱编码顺序。在这种情况下，编号后的日志可以设置截止日期来进行管理，截止日期一到，这些文件将被删除或归档。这种情况称为日志截止。

本书涉及了一些日志截止的内容。进一步的有关信息请参阅MySQL手册，访问下列网址：http:// www.mysql.com/doc/en/ Log_file_maintenance.html。

15.2 检查和修理表

也许需要根据备份资料恢复被破坏的表（或者甚至是整个数据库），如果表是偶然受到毁坏或删除，可以使用更新日志。在毁坏相对比较小的情况下，MySQL可以提供几种修复表的选项。在下一节中将介绍如果出现这种情况，应该如何去做。

15.2.1 检查表错误

如果怀疑出现了某些问题，应该做的第一件事就是检查表中的错误。myisamchk实用程序就是检查表使用的一种方法。命令myisamchk后边是表文件的名称，可以看到这个实用程序是如何调用的。

```
myisamchk table-file
```

因为myisamchk需要专门访问该表，所以一个好办法是在运行之前先让服务器脱机。不需要担心这种方法在客户机之间协调访问的问题。此外在检查表错误时，还可以运行几种选项，参阅表15.1。

表15.1 myisamchk表检查的额外选项

选项	名称	说明
F	快速检查	只检查非正常关闭的文件
m	中速检查	更详细的检查
e	扩展检查	最慢速度、最完整的检查
C	变更检查	只检查自上次检查以来有变化的文件
w	等待检查	等待锁被删除之后进行检查

下面的示例是使用扩展选项运行myisamchk。如果希望这么操作，就不要使用很大的表来查看选项如何工作，因为这样将使服务器工作很长时间。如果使用扩展选项没有发现错误，可以确定该表没有问题。如：

```
[root@host]# /usr/local/mysql/bin/myisamchk -e db1/accounts
Checking MyISAM file: db1/accounts
Data records:      14    Deleted blocks:       0
- check file-size
- check key delete-chain
- check record delete-chain
- check index reference
- check records and index references
```

myisamchk的弱点是在进行诊断时，该数据库检查工具需要锁定客户机，但是客户机不能在运行myisamchk时锁住被检查的表。在比较大的表中，myisamchk可能要花费数分钟的时间进行检查，这可能是个问题。

一种代替方法是设置myisamchk可以使用很大的缓存空间（使用myisamchk --help可以看到修改不同缓存空间的选项）。还有一种代替方法是使用不同的方法检查表，即CHECK TABLE命令。

myisamchk实用程序要求专门访问正在进行检查的表，因为程序可以直接使用表文件工作。另一方面，CHECK TABLE命令也可以使用服务器检查表。这意味着不需要做很多工作，因为没有关闭服务器，也不需从表中删除所有锁定。

下面是这个命令的使用方法：

```
CHECK TABLE table-name, ...
```

下面是命令运行的示例：

```
mysql> CHECK TABLE accounts;
+---------------+-------+----------+----------+
| Table         | Op    | Msg_type | Msg_text |
+---------------+-------+----------+----------+
| db1.accounts  | check | status   | OK       |
+---------------+-------+----------+----------+
1 row in set (0.00 sec)
```

提示：在这种情况下，还可以在CHECK TABLE命令中添加关键字FAST，MEDIUM，EXTENDED和CHANGED，以便执行希望使用的检查种类。

也许有人会提出为什么不在整个过程中运行CHECK TABLE，而是执行myisamchk？主要原因是，在使用CHECK TABLE时服务器承担了所有工作。如果服务器关闭，CHECK TABLE就不是一个选项。另一方面，myisamchk在文件级工作，即使服务器关闭也能工作。因为CHECK TABLE是一个SQL命令，只能通过客户机进行发送，服务器必须在运行中，才能接受这个命令。如果做了这种选择，就要尽一切可能让MySQL来承担工作。

注释：myisamchk不能在InnoDB和BDB表上工作，在这些类型的表上面需要使用CHECK TABLE命令。

修理表

表检查完毕之后，如果发现存在错误，于是必须修理这个表。最好的办法就是在进行修理之前先复制出现问题的表。如果第一种方案不能解决问题，可以试着使用不同方法来进行修复。

前面介绍的myisamchk工具还可以用来修理被毁坏的表。使用-r选项加表名，就可以开始修复过程。如：

```
myisamchk -r table-file
```

下面是一个示例：

```
[root@host]# /usr/local/mysql/bin/myisamchk -r test/users
- recovering (with keycache) MyISAM-table 'test/users'
Data records: 16
```

如果 -r 选项不能解决问题，可以使用 -o 选项来慢速修复表。还可以使用其他选项，表 15.2 介绍了这些方法的功能。

表 15.2 myisamchk 额外的表修理选项

选项	名称	说明
-r	修理和恢复	标准恢复方法
-o	恢复的安全模式	慢速，完全恢复
-q	快速恢复	只检查索引，不检查数据文件

正如上一节所述，记住 myisamchk 工具在文件级工作，因此需要所有锁被删除并且所有客户被排除。

在检查表时，应该首先尝试最快的选项，如果不行再转而使用比较慢的选项。可能会发现有很多常见问题可以不需要使用很慢的选项就可以修复。如果在运行最强的修复可能性之后，问题仍然存在，将不得不从备份文件恢复该表。恢复方面的内容详见 15.3.2 节。

在修复表时可以使用的另一个选项是 REPAIR TABLE 命令，使用方法如下：

```
REPAIR TABLE table-name, ...
```

与 myisamchk 类似，可以使用选项 QUICK 或 EXTENDED 关键字来设置修理类型，只需在 REPAIR TABLE 语句后面简单地添加选项名即可。参阅下面示例：

```
mysql> REPAIR TABLE users QUICK;
+------------+--------+----------+----------+
| Table      | Op     | Msg_type | Msg_text |
+------------+--------+----------+----------+
| test.users | repair | status   | OK       |
+------------+--------+----------+----------+
1 row in set (0.27 sec)
```

优化表

在多数情况下，优化表的做法很值得。常见的示例是如果表相当活跃，特别是在很多地方需要删除时，在这种情况下，表可能很快就变得支离破碎了，其结果是导致性能的下降。运行 OPTIMIZE TABLE 命令可以刷新这些被删除的记录，因此能清理出空间。

下面是 OPTIMIZE TABLE 命令的使用方法：

```
OPTIMIZE TABLE table-name, ...;
```

例如以下命令优化了表 water：

```
mysql> OPTIMIZE TABLE water;
+------------+----------+----------+----------+
| Table      | Op       | Msg_type | Msg_text |
```

```
+------------+----------+----------+----------+
| test.water | optimize | status   | OK       |
+------------+----------+----------+----------+
1 row in set (0.11 sec)
```

OPTIMIZE TABLE命令的作用就好像母亲整理你的房间。除了抛弃旧的、被删除的文件以外，还可以排序有索引的文件，把变量表行的内容设置到连续空间中，更新表统计信息。然而应该记住，这个表已经被锁定，在进行操作的时候不能被客户机访问。

不同的应用场合

可以使用myisamchk或者REPAIR TABLE来修复被毁坏的表，但是应该记住（正如前面在介绍CHECK TABLE命令内容时讨论的），为了使用REPAIR TABLE，服务器必须正在运行，如果服务器已经停机，就只能使用myisamchk了。

15.3　备份和恢复

除了日志和表优化以外，任何数据库管理员的基本任务还包括确定数据得到保护不会丢失。这项工作包括对数据库的常规备份和测试恢复。当发生灾难时（并且这不属于错误），如果按照下面章节建议的步骤进行，就可以很好地处理这些问题。

15.3.1　备份数据库和表

MySQL发布中都带有一个名为mysqldump的实用程序，可以用来进行备份整个数据库和数据库中的单独表到一个文本文件。除了明显需要备份的数据以外，如果需要导出数据库内容到不同的RDBMS上面，或者如果需要简单快速地移动一个系统的信息到其他系统上，这个操作也十分有用。如：

```
mysqldump database-name
```

下面是使用示例：

```
[user@host]# /usr/local/mysql/bin/mysqldump db1 --user=john --password=hoonose
```

这个过程将在屏幕上显示了示例数据库db1的整个内容。其输出情况参阅图15.1。

注意在图15.1中，SQL语句包括在mysqldump的输出中，方便了重建文本或表。与mysql命令一样，还需要使用--user和--password选项来确定一个授权用户和口令，以便执行转储功能。

清洁的空间

如果备份的数据已经被破坏，最好的方法是在恢复之前执行DROP TABLE或DROP DATABASE命令。这样给恢复工作创造了一个清洁的空间。幸运的是，mysqldump实用程序可以做到这些。如果看一下通过调用mysqldump产生的SQL语句，就可以知晓可以包含哪些命令。

如果不需要把整个数据库转储，该如何操作？可以进行简单的修改，指定这些数据库中哪些表应该备份。请参阅下列示例：

```
[user@host]# /usr/local/mysql/bin/mysqldump db1 name address --user=john --
password=hoonose
```

图 15.1 mysqldump 命令的输出

这个命令只是转储数据库 db1 中 name 和 address 表的内容。

在实际操作中，可能需要保存 mysqldump 的输出到文件，而不是在控制台上翻滚查看。不论是在 UNIX 还是 Windows，都可以通过重定向运算符操作，参阅下列示例：

```
[user@host]# /usr/local/mysql/bin/mysqldump db1 name address > mydump.txt
```

这个命令的结果是一个文本文件，名为 mydump.txt，包含需要使用的命令，以便重新创建数据库 db1 中的 name 和 address 表。

如果需要同时备份一个以上的数据库，可以使用 -B 选项，参阅以下示例：

```
[user@host]# /usr/local/mysql/bin/mysqldump -B db1 db2
```

选择分隔符

在使用 mysqldump 实用程序时，可以控制使用字符，在列输出中包含和分隔字段，方法是添加下列的选项：--fields-enclosed-by，--fields-terminated-by，--fields-escaped-by 和 --lines-terminated-by。在导入它们之前，如果需要移动被转储数据到需要以定制格式编码记录的系统时，这十分有用。

在这种情况下没有指定表，因为使用-B选项备份超过一个以上的数据库时，整个数据库将被转储。单独的表不能使用这种操作。

假设希望使用相同结构创建表，但是数据不同于前一个表，将怎样处理？mysqldump 实用程序同样可以解决问题。-d 选项可以产生相同格式的表，但是内容是空的。如果希望看到这个行动，可以使用下列命令：

```
[user@host]# /usr/local/mysql/bin/mysqldump -d db1 users > users.structure
```

这样将生成一个转储文件，包含了 SQL 命令，可以创建 user 表的一个空副本。

另一方面，如果只需要表的内容，例如把内容转储到一个不同的表上，这时可以再次使用 mysqldump，但是选项是 -t。这样将产生一个包含所有已经在表中运行的 INSERT 语句文件。其中不能复制的内容是创建表使用的指令。

参见下列实例：

```
[user@host]# /usr/local/mysql/bin/mysqldump -t db1 employees > employees.data
```

现在 employees 表中记录已经准备好导入到任何可以识别 SQL 的应用程序中了。

15.3.2　从备份恢复数据库和表

有关这方面内容的大多数专著都强调定期备份数据（以及正确操作）的重要性，但是恢复数据的问题是一个常常被忽略的过程。如果不能进行访问，备份文件将没有用处。当然应该定期根据备份进行恢复，保证数据可以在紧急情况下使用。事实上，直到已经确认备份文件可以恢复时，备份工作才能说是完全的，这种说法一点也不过分。除了应该做到心中清楚外，还应该完全熟悉恢复的过程，因为不可能指望在系统出现问题之后才去花时间学习恢复程序。

这样值得吗

应该记住进行恢复操作的用户必须得到许可，可以创建表和数据库。相应地，可能需要在前面的命令中同时使用 --user，--password 或 --host 选项。

在前面的章节中，已经学习到 mysqldump 实用程序的输出包括 SQL 语句，如 CREATE TABLE 来简化重建丢失数据的过程。正因为如此，可以使用由 mysqldump 生成的文件，并且通过 mysql 命令行客户机进行输送，以便快速重新创建丢失的数据库或表。

参阅下列示例：

```
[user@host]# /usr/local/mysql/bin/mysql db1 < mydump.txt
```

在这个示例中，mydump.txt 是一个文本文件，包含前面运行 mysqldump 的输出。这个文件的内容（SQL 命令）是通过 mysql 命令行客户机运行的，使用标准输出重新定向。

如果不能访问（或不希望访问）命令行，另一个选项是使用 SOURCE 命令，方法如下：

```
mysql> SOURCE mydump.txt
```

SOURCE命令在命名的文本文件中使用SQL指令来重建具体的数据库或表。如果希望看到恢复的结果，可以使用简单的 SELECT 语句来证实数据已经被成功地恢复。

另一个选项是使用 MySQL 命令 LOAD DATA INFILE 从文本文件中导入数据。使用这个命令，加上 INTO TABLE 关键字，可以指出希望发送数据的表。

参阅下列示例：

```
mysql> LOAD DATA INFILE '/home/me/data.txt' INTO TABLE birthdays
FIELDS TERMINATED BY ',' LINES TERMINATED BY '\r\n';
Query OK, 5 rows affected (0.06 sec)
Records: 5  Deleted: 0  Skipped: 0  Warnings: 0
```

有关 LOAD DATA INFILE 命令的细节内容参阅第 9 章。

利用恢复来移动

除了恢复丢失或被破坏的数据外，恢复程序也可用来发送表或者发送整个数据库给不同的 MySQL 服务器。恢复数据是导入或导出不同服务器之间内容的一种快捷方法。

一旦能够熟练地使用这个程序备份和恢复数据，就可以为所在单位制定一个定期备份的程序。cron 工具是一个 UNIX 计划实用程序，可以用于这种用途。这个程序允许计划 mysqldump 实用程序在规定的时间和日期内运行。在 UNIX 命令提示下键入 man cron，可以查找更多的如何使用这个工具的内容。

在 Windows NT，Windows 2000 或 Windows XP 中，可以在命令提示位置上使用 at 命令，或者使用 Task Scheduler（任务计划）工具（Control Panel -> Scheduled Tasks）来自动进行备份。

注释： cron 这个词也许令人迷惑，但是只要注意这个词是来自希腊语，表示时间，在英文中应该写成 chronos，就能理解它的含义了。使用这个词根的词还有 chronology 和 chronometer。

15.4 小结

MySQL能够普及的优势之一是使用方便，然而这个数据库系统不可能承担起所有功能。对于管理员来说，基本维护和建立好的备份和恢复过程在任何生产环境下都是需要的。本章集中讨论了采取步骤保证平稳使用 MySQL 安装的最低步骤，如使用不同的日志监督数据库，找出潜在故障部分等。本章还重复了检查和维护表的方法。最后，涉及到最为重要的内容是备份和恢复，可以使用 MySQL 中提供的多种实用程序实现。

第16章 性能优化

随着数据库的扩张，会遇到希望调整用于控制服务器的MySQL默认设置。随着处理器的提速、磁盘越来越大、越来越快，内存也得到某些补充，这些都会作用到性能，但是这些不属于本书讨论的范围。本章的内容是使用MySQL中可能使用的工具最大化服务器性能，介绍一些技术，保证查询尽可能有效地运行。

在讨论性能问题时，数据库设计也是一个应该考虑的方面。为了提高性能采取不同的优化表的策略也是本部分介绍的内容之一。应该注意的是，大多数优化方法首先应该涉及到细化查询、添加索引等。查询的优化应该在本章中优先考虑。

改善性能首要的工作就是在查询上面，特别是在经常运行的查询上面。分析一个查询，并重新编写它使之更为有效的做法十分重要。可以使用MySQL慢速查询日志（参阅第15章）了解可能细化调整查询的一些途径。

16.1 索引

在线论坛中非常多的人在寻找有关慢速查询方面的信息，而不是试图在经常访问的列上添加一个索引。正如在第8章中所提到的，经常访问的带有列的表可以通过创建索引来排列。索引指向数据库中具体数据所在的一个位置，同时在列上创建一个索引可以排列该列上的信息。当服务器需要访问该信息进行查询时，就会知晓在什么地方进行查询，因为索引指向有关位置。

索引在多表查询中甚至更为重要。如果需要花费一些时间在表中进行全表扫描，如果需要检查好几个表，可以想像需要花费多少时间。如果优化查询是一个目标，首先需要做的事就是试图实现一个索引。

决定哪些列应该加索引涉及一些因素。如果一个列涉及查找、分组、排序，索引将可以达到提高性能的效果。这样包括了联合操作部分的一些列，或者可以出现语句的列，如WHERE，GROUP BY或ORDER BY。

请参阅下列示例：

```
mysql> SELECT salary FROM empData JOIN empMaster ON empData.sid = empMaster.
id WHERE salary > 50000;
```

在此应该增加索引的列有empData.sid和empMaster.id，因为它们属于联合部分。因为salary是WHERE语句的一部分，所以增加索引也是一个好选择。

在此应该考虑的另一个因素是带很多重复值的索引不会产生很好的结果。属于“对错”选择内容的列不能通过建立索引来改善。另一方面，其值属于惟一的列（如雇员社会保险编号）可以通过索引得到很大益处。

可以使用表来联合多个非惟一的索引，以便改善性能。在前面的例子中，列empData.sid，empMaster.id和salary都可以添加索引。创建非惟一索引的数量没有限制。

假设创建索引达到了逻辑极限，就会觉得索引越多，就越方便。这是一个谬误：添加一个索引并不一定能够改善性能。例如，很小的表就不需要建立索引。此外，每个索引要在磁盘上面占有一定的额外空间，每个索引字段都要求 MySQL 在这个字段中以及数据库中的具体位置上为每条记录保存信息。在构建索引的同时，这些表开始占用更多的空间，进而索引增加了搜索速度，但是减慢了写操作，如 INSERT，DELETE 或 UPDATE。除非在数据库中使用索引，否则少量的尝试不会得到很高的性能增益。

某些管理方面的工具可以帮助用户监督索引情况，或者为添加索引提供建议。SHOW STATUS 或 mysqladmin 扩展状态命令都可以显示值，以便为索引提供参考：

- 如果索引正在工作，Handler_read_key 的值将很高。这个值代表了一个行被索引值读的次数。很低的值表明通过增加索引得到的性能改善不高，因为索引并不经常使用。
- 对于Handler_read_rnd_next的高值意味着查询运行低效，且索引建立被认为是一种补救。这个值表明按照顺序读下一行的申请次数。这种情况的发生是在表被按照次序从第一条记录到最后一条记录扫描，以便进行查询的情况。对于经常的查询，这是一种资源的浪费。一个关联索引会直接指向记录，因此不需要再扫描整个表。请注意性能差的索引还可能导致产生一个很高的值。

如果需要浏览这些情况，运行下列命令即可：

```
mysql> SHOW STATUS LIKE 'Handler_read%';
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Handler_read_first    | 1     |
| Handler_read_key      | 0     |
| Handler_read_next     | 0     |
| Handler_read_prev     | 0     |
| Handler_read_rnd      | 0     |
| Handler_read_rnd_next | 9     |
+-----------------------+-------+
6 rows in set (0.03 sec)
```

一旦已经在表中装载数据，并且按照希望的方式建立了索引，就应该在上面运行ANALYZE TABLE命令。这个命令可以分析表中的数据，并按照享有相同值的行的平均数来创建表统计。在决定哪些索引将在表连接中使用时，可以允许 MySQL 优化程序使用这个信息。如：

```
mysql> ANALYZE TABLE users, groups, users_groups;
+-------------------+---------+----------+----------+
| Table             | Op      | Msg_type | Msg_text |
+-------------------+---------+----------+----------+
| test.users        | analyze | status   | OK       |
| test.groups       | analyze | status   | OK       |
| test.users_groups | analyze | status   | OK       |
+-------------------+---------+----------+----------+
3 rows in set (0.01 sec)
```

经常运行 ANALYZE TABLE 命令是一个好办法，特别是在表中已经添加了大量的数据之后，以保证优化程序总可以使用最高效益的索引。

自然顺序

如果有一个表经常需要读，但是很少修改，于是可以试着分类这个表中的信息，而不是建立索引。使用 ORDER BY 命令，以及 ALTER TABLE 可以进行操作。这样将读得更快，效率也更高。

16.2 查询高速缓存

在运行 SELECT 查询时，MySQL 4.x 会记录下查询和返回的结果。这是通过 SELECT 每次进行查询时，在一个特殊的高速缓存（名为查询高速缓存）中保存结果集实现的。然后，当再次请求服务器做相同的查询时，MySQL 将从高速缓存中检索结果，而不是再次运行该查询。可以想像出来，这样当然提高了这个过程的速度。

虽然默认条件下是启动这个性能，但是还必须要确认查询高速缓存已经打开，方法是检查服务器的变量。参阅下列示例：

```
mysql> SHOW VARIABLES LIKE '%query_cache%';
+--------------------------+---------+
| Variable_name            | Value   |
+--------------------------+---------+
| have_query_cache         | YES     |
| query_cache_limit        | 1048576 |
| query_cache_min_res_unit | 4096    |
| query_cache_size         | 48      |
| query_cache_type         | ON      |
+--------------------------+---------+
5 rows in set (0.00 sec)
```

下面是进一步说明：

- 第一个值 have_query_cache 表明这个服务器在安装时（默认条件）已经配置进行查询高速缓存。
- query_cache_size 值表示为 48，这意味着分配给高速缓存的空间是 48 MB。如果这个值为 0，则表明查询高速缓存已经关闭。
- query_cache_type 的变量值范围从 0 到 2。值 0 或 OFF 表明查询高速缓存已经关闭，1 意味着查询高速缓存已经打开，但使用 SQL_NO_CACHE 选项的 SELECT 语句除外。DEMAND 或 2 要求根据需要按照运行带 SQL_CACHE 选项的 SELECT 语句，提供查询高速缓存。

如果希望亲自看到查询高速缓存运行中对性能的反应，分别使用和不使用查询高速缓存运行相同的查询，以便比较性能差别。下面是使用查询高速缓存的情况：

```
mysql> SELECT SQL_CACHE * FROM employees;
+------------+-----------+----------+---------+---------+
| employeeID | firstname | lastname | titleID | salary  |
+------------+-----------+----------+---------+---------+
```

```
|          1 | Marvin     | Freeman   |        1 |    41000 |
|          2 | Ed         | Bailey    |        1 | 32100.2 |
|          3 | Beth       | Trinity   |        1 | 55300.8 |
|          4 | Rachel     | Williams  |        2 | 35000.2 |
|          5 | Ian        | Stanton   |        2 | 35000.6 |
|          6 | Terry      | Thorne    |        2 |   303000 |
|          7 | Joan       | Chu       |        1 |    41000 |
|          8 | Stan       | Schowry   |        1 | 24372.1 |
|          9 | Will       | Tates     |        3 |    50000 |
|         10 | Steve      | Bobs      |        3 | 3.8e+08 |
|         59 | Ismail     | Ozkan     |        3 |    88900 |
...
+------------+-----------+----------+---------+---------+
711 rows in set (0.06 sec)
```

下面不使用高速缓存运行相同的查询，其结果是：

```
mysql> SELECT SQL_NO_CACHE * FROM employees;
+------------+-----------+----------+---------+---------+
| employeeID | firstname | lastname | titleID | salary  |
+------------+-----------+----------+---------+---------+
|          1 | Marvin     | Freeman   |        1 |    41000 |
|          2 | Ed         | Bailey    |        1 | 32100.2 |
|          3 | Beth       | Trinity   |        1 | 55300.8 |
|          4 | Rachel     | Williams  |        2 | 35000.2 |
|          5 | Ian        | Stanton   |        2 | 35000.6 |
|          6 | Terry      | Thorne    |        2 |   303000 |
|          7 | Joan       | Chu       |        1 |    41000 |
|          8 | Stan       | Schowry   |        1 | 24372.1 |
|          9 | Will       | Tates     |        3 |    50000 |
|         10 | Steve      | Bobs      |        3 | 3.8e+08 |
|         59 | Ismail     | Ozkan     |        3 |    88900 |
...
+------------+-----------+----------+---------+---------+
711 rows in set (0.84 sec)
```

如果经常运行的查询都启用查询高速缓存，将会看到性能方面的改善是十分明显的。

注意，一旦表有变化，使用这个表的高速缓存查询就变成无效，并且将从高速缓存中删除。这样防止查询从旧表上返回不准确的数据。虽然这样做使得查询高速缓存更为有用，但是经常有变化的表不能从高速缓存中得到利益。在这种情况下，可以考虑不使用查询高速缓存。这时可以添加一个选项 SQL_NO_CACHE 来实现，如前所述的 SELECT 语句。

注意具体情况

文本方式的查询将完全匹配查询高速缓存的情况，它是把任何差异作为一个新查询处理。例如，SELECT * FROM *table-name* 不会从高速缓存中的 SELECT * FROM *table-name* 上面得到结果。

16.3 查询分析

在SELECT查询开始处加上一个EXPLAIN关键字，将告诉MySQL返回一个图表，说明这

个查询如何进行处理。在这个图表里面涉及了查询将访问哪个表的信息以及查询希望返回行数的信息。这个信息可以用来查看哪些表可以添加索引，以便加快执行速度，分析瓶劲的位置。

请参考下列查询示例：

```
mysql> SELECT a.companyID, a.companyName FROM Companies a, Users b WHERE a.
companyID = b.fk_companyID AND a.numEmployees >= 0 AND a.companyName LIKE
'%i%' AND b.groupID IN (SELECT c.groupID FROM Groups c WHERE c.groupLabel =
'Executive');
+-----------+---------------------------+
| companyID | companyName               |
+-----------+---------------------------+
|      3447 | Udoozy Mining             |
|       123 | Benny Bill Chickens, Inc  |
+-----------+---------------------------+
2 rows in set (0.20 sec)
```

现在添加关键字EXPLAIN到查询开始部位，就可以得到MySQL如何进行处理的一些信息：

```
mysql> EXPLAIN SELECT a.companyID, a.companyName FROM Companies a, Users b WHERE a.
companyID = b.fk_companyID AND a.numEmployees >= 0 AND a.companyName LIKE '%i%' AND
b.groupID IN (SELECT c.groupID FROM Groups c WHERE c.groupLabel
= 'Executive');
+----+--------------------+-------+------+---------------+------+---------+--
----+------+-------------+
| id | select_type        | table | type | possible_keys | key  | key_len | ref  |
rows | Extra        |
+----+--------------------+-------+------+---------------+------+---------+--
----+------+-------------+
|  1 | PRIMARY            | a     | ALL  | NULL          | NULL |    NULL | NULL |
934 | Using where  |
|  1 | PRIMARY            | b     | ALL  | NULL          | NULL |    NULL | NULL |
134 | Using where  |
|  2 | DEPENDENT SUBSELECT | c     | ALL  | NULL          | NULL |    NULL | NULL |
621 | Using where  |
+----+--------------------+-------+------+---------------+------+---------+--
----+------+-------------+
3 rows in set (0.12 sec)
```

这里看上去似乎有些迷惑，下面我们做一下说明。EXPLAIN SELECT的结果是一个表，列出了查询中所有的SELECT，包括MySQL准备如何进行处理的内容。

- id列指出在整个查询中SELECT的位置，而table列存放着所查询的表名。
- select_type列指出查询的类型，包括没有子查询的简单查询、UNION、子查询、外部查询、外部查询中的子查询或FROM语句中的子查询等。
- type列指出连接如何执行的情况。这里存在很多的值，范围从const（连接的最佳种类，因为这意味着表只包含简单匹配记录）到ALL（最差种类，因为这意味着MySQL必须扫描每个单独记录，查找与其他连接表中的记录匹配的情况）。
- possible_keys列指出了为了提高查找速度，在MySQL中可以使用的索引。

- key 列指出实际使用的键，包括在 key_len 列中显示的键的长度。
- rows 列指出 MySQL 需要在相应表中为了成功进行查询，进行检验的行的数量。为了得出总行数，MySQL 必须扫描处理整个查询，再乘以每个表的行值。
- Extra 列包含一些其他信息，涉及 MySQL 如何处理查询，比如说，使用 WHERE 语句、使用一个索引、利用一个临时表等。

现在根据前面的输出，可以清楚地了解到为了实施查询，MySQL 将需要检查所有的命名表中所有的行。然后，MySQL 需要扫描的行总值大约为 934 × 134 × 621 = 77 721 876 行，这是一个非常大的值。

然而，通过查看 EXPLAIN SELECT 命令的输出结果，可以清楚看出其改善程度。例如，所有表的 possible_keys 列都是 NULL，这表明 MySQL 没有发现任何可以使用的索引。这样通过查阅表和在可能的位置上添加索引的方法可以迅速地进行改正：

```
mysql> ALTER TABLE Companies ADD PRIMARY KEY (companyID);
Query OK, 934 rows affected (0.18 sec)

mysql> ALTER TABLE Groups ADD INDEX (groupLabel);
Query OK, 621 rows affected (0.11 sec)
```

现在可以试着使用 EXPLAIN 再次运行查询：

```
mysql> EXPLAIN SELECT a.companyID, a.companyName FROM Companies a, Users b
WHERE a.companyID = b.fk_companyID AND a.numEmployees >= 0 AND a.companyName
LIKE '%i%' AND b.groupID IN (SELECT c.groupID FROM Groups c WHERE c.groupLabel
= 'Executive');
+----+--------------------+-------+--------+---------------+-------------+-
--------+-----------+------+-------------+
| id | select_type        | table | type   | possible_keys | key         |
key_len | ref           | rows | Extra       |
+----+--------------------+-------+--------+---------------+-------------+-
--------+---------------+------+-------------+
|  1 | PRIMARY            | b     | ALL    | NULL          | NULL        |
NULL    | NULL          |  134 |             |
|  1 | PRIMARY            | a     | eq_ref | PRIMARY       | PRIMARY     |
4       | b.fk_companyID |    1 | Using where |
|  2 | DEPENDENT SUBSELECT | c     | ref    | groupLabel    | groupLabel  |
41      | const         |    1 | Using where |
+----+--------------------+-------+--------+---------------+-------------+-
--------+---------------+------+-------------+
3 rows in set (0.03 sec)
```

正如所看到那样，现在 MySQL 可以使用新添加的索引来切断需要检查的行数。这样可以从上面表的 type 列中清楚地看到，表明 MySQL 现在可以使用 eq_ref 连接类型（最佳的连接类型，表明 MySQL 只需要从这个表中读一个单独的行，以满足连接条件）。将结果乘以行，便可以看到 MySQL 只需要扫描 134 × 1 × 1 = 134 行来处理查询，这样比前面情况（没有索引方法）要改善许多。

16.4 优化多表查询

连接就是一种多表查询，可以跨过以一个或多个共同字段方式相互连接的表来执行命令。

这通常用来建立RDBMS常规表之间的关系，并且赋予SQL程序员可以连接单独表中的记录，为相同数据创建不同视图。

子查询是一个嵌套在另一个SELECT语句中的SELECT语句。子查询经常用来把一个复杂的查询拆分成为一系列的逻辑步骤，或者使用其他查询结果回答一个查询。其结果是不需要执行两个（或更多的）单独查询，就可以执行一项包含一项（或多项）子查询的简单查询。

MySQL可以比子查询更好地优化连接，所以如果发现在自己的MySQL服务器上的负载平均值达到了无法接受的高水平，就应该检验应用程序代码，并试着重写作为连接和连接序列的子查询。例如，在下列子查询合法的情况下：

```
mysql> SELECT bid FROM branches_services WHERE sid = (SELECT sid FROM ser-
vices WHERE sname = 'Administration');
+------+
| bid  |
+------+
|  536 |
| 1031 |
| 1743 |
+------+
3 rows in set (0.22 sec)
```

下列同样的连接因为MySQL优化了算法，将运行得更快（注意运行时间不同）：

```
mysql> SELECT bs.bid FROM branches_services AS bs, services AS s WHERE s.sid
= bs.sid AND s.sname = 'Administration';
+------+
| bid  |
+------+
|  536 |
| 1031 |
| 1743 |
+------+
3 rows in set (0.17 sec)
```

这是在类型和长度上面匹配被连接字段的好办法。MySQL在使用具有不同长度和类型的连接字段索引时显现出一些不足。

也可以通过有效地使用MySQL的集合性能和修改程序来把低效的子查询转变成为更为有效的连接。参阅下列子查询：

```
mysql> SELECT customerName FROM customers WHERE customerID = (SELECT customerID
FROM invoices WHERE invoiceStatus = 'U' AND invoiceAmt = (SELECT MAX(invoiceAmt)
FROM invoices WHERE invoiceStatus = 'U'));
+---------------+
| customerName  |
+---------------+
| PowrTools Inc |
+---------------+
1 row in set (0.22 sec)
```

按照下列连接可以产生更好的效果，这样将更容易进行阅读，而且运行得会更快：

```
mysql> SELECT customerName FROM customers, invoices WHERE customers.customerID
= invoices.customerID WHERE invoices.invoiceStatus = 'U' ORDER BY invoiceAmt
DESC LIMIT 1;
+---------------+
| customerName  |
+---------------+
| PowrTools Inc |
+---------------+
1 row in set (0.12 sec)
```

如果希望避免中间使用嵌套查询，还可以使用基于会话的服务器变量。因此在下面的查询中，将列出在任何情况下高于平均账单金额的客户：

```
mysql> SELECT customerName FROM customers WHERE customerID IN (select customerID
FROM invoices WHERE invoiceAmt > (SELECT AVG(invoiceAmt) FROM invoices));
+-----------------+
| customerName    |
+-----------------+
| PowrTools Inc   |
| Fatty Foods plc |
| CoffeManiacs    |
+-----------------+
3 rows in set (0.22 sec)
```

通过把任务划分成两个查询并使用服务器端MySQL变量进行连接的方法进行相同处理：

```
mysql> SELECT @avg:=AVG(invoiceAmt) FROM invoices;
mysql> SELECT DISTINCT customerName FROM customers, invoices WHERE customers.
customerID = invoices.customerID and invoiceAmt > @avg;
+-----------------+
| customerName    |
+-----------------+
| PowrTools Inc   |
| Fatty Foods plc |
| CoffeManiacs    |
+-----------------+
3 rows in set (0.14 sec)
```

上述合并的两个查询将比第一个子查询运行得更快。

16.5 使用临时表

MySQL还允许使用CREATE TEMPORARY TABLE命令创建临时表。这种表之所以这么称呼，是因为它只针对单一MySQL会话过程而存在，当使用这些表的客户机关闭了与MySQL服务器的连接时，它将自动删除。临时表用于短暂的基于会话的数据或计算，或者是一些暂时保存的数据。并且因为表依赖于会话，所以两个不同的会话可以使用相同的表名而不会冲突。

因为临时表保存在内存中，所以要比基于磁盘的表明显地快。结果可以有效地作为中间储存区域，以便提高查询实施的速度，帮助把复杂的查询拆分成为更为简单的部件，或者作为子查询和连接支持的代替。

例如，假定有一个复杂的查询涉及从具体字段中选择一组特殊值，因为比较复杂，所以MySQL引擎无法优化这个查询。富有创造性的SQL程序员可以通过把单一复杂查询拆分成多个简单查询（便于更好地进行优化）来改善其性能，然后在适当的字段上使用UNIQUE键创建临时表之后，使用INSERT IGNORE…SELECT命令保存产生的结果到一个临时表上。结果是得到关于该字段的一组特殊值，而查询速度更快。

下面是另一个示例：假设有一个表包含一个月的交易信息，也许有30万条记录。在每天结束时，应用程序需要生成一个报告，总结当天的交易情况。在这种情况下，以天为基础对整个30万条记录运行SUM()和AVG()函数，不是一个好的办法或有效的办法。一个更为有效的解决方案是使用INSERT…SELECT，只提取当日交易情况到一个临时表中，在临时表上运行合计函数，以便产生所需要的报告，然后再删除临时表。因为临时表可能包含很多更小的记录子集，其性能将会更好，服务器负荷也能降低。

16.6 优化表设计

为了使查询能够更加精练，需要考虑表设计方面的一些因素。首先，如果经常查询的表会有很多变化，改进性能的方式是使用定长字段，而不使用变长字段。定长字段不论内容如何都将占用一定数量的空间，而变长字段可以根据数据输入情况进行自我调整。虽然使用定长字段将浪费更多的磁盘空间，但是，如果从查询速度考虑，MySQL处理定长字段要比变长字段的速度快。

如果计划使用定长字段，则应尽可能保持列宽为最低。例如在设计一个表时，创建CHAR(255)的列更容易，因为不需要再具体考虑实际到底需要多少。实际上可能发现列宽减少一半也能适合需要。减少列宽不仅仅减少了磁盘空间，而且还意味着在进行处理时的I/O过程更少。毋庸赘言，完全删除没有必要的列也可以提高性能。

改善性能的另一项技术是使用OPTIMIZE TABLE命令处理经常需要修改的表（第15章已全面讨论过）。经常的修改将导致磁盘碎片，以致需要花费额外的时间去读取没有用的空间块，以便得到希望的数据。

在考虑改善性能时，还要检查是否需要针对已经建立的所有表。刚开始设计表时，划分数据可能是一个好办法，但是额外的表意味着性能降低。经常要查看这些表，看是否有可能合并这些数据到一个表中？如果发现没有必要合并表，则应试着匹配连接的列。如果连接的列采用相同的数据类型和长度，这些查询将更为有效地运行。

规模大小问题

这里介绍一些使用定长字段时改变表的技巧。如果经常修改的表中的一个列不能定义成定长的，则可以考虑移动这个列到单独的表中，而转换原始表中的其他列为定长字段。这样做在多数情况下不会有问题，所以使用定长字段，而不是变长字段得到的性能增加也是可行的。

注意，InnoDB或BDB表处理行存储与MyISAM或ISAM表的情况不同。在这些类型的表上使用定长列，不用变长列也不会提高性能。

更多资源

查询优化本身就是一门技术，在本章有限的空间中不可能概括全部内容。进一步的信息可以查阅下列网站：http://www.mysql.com/doc/en/MySQL_Optimisation.html和http://www.mysql.com/doc/en/Tips.html。从这里可以得到更广泛的信息、提示和加速MySQL查询以及提高效率的技术。

16.7 调整服务器设置

如果希望服务器更加有效地运行，最佳解决方案是加大内存空间和使用更大更快的磁盘。然而，在大多数情况下，这些硬件技术不能算做一种选择。假设我们是在很不理想的条件下运行服务器，能够用于优化服务器、提高性能的敏感技术也很少，所以本节主要介绍一些可以改进服务器的常规技术。

正如在第13章中已经见到的，MySQL涉及很大数量的变量，其中的值可以进行修改，以便满足定制要求。有些变量在启动MySQL服务器时就能够设置好，其他一些变量可以在服务器运行时进行设置。

在调整服务器变量，以便发挥服务器的最高性能时，MySQL手册推荐首先考虑key_buffer_size和table_cache变量。

- key_buffer_size变量控制了MySQL索引缓冲可以使用内存的数量。这个值越高，索引可以使用的内存越多，性能也越好。一般情况下，可以保持这个值在服务器可使用内存总量的25%到30%。
- table_cache变量控制表高速缓存可以使用内存的数量，以及在同一时间内MySQL可以处理表的打开总量。对于有很多数据库和表的非常忙的服务器来说，应该增加这个值，以便MySQL能可靠地支持所有的请求。与此有关的变量还有max_connections变量，MySQL手册建议使用公式table_cache = max_connections × *N* 设置table_cache，其中*N*为标准连接中表的数量。

正如第13章所指出的，可以使用SET命令修改这些值，参阅下面示例：

```
mysql> SET GLOBAL table_cache=200;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@table_cache;
+---------------+
| @@table_cache |
+---------------+
|           200 |
+---------------+
1 row in set (0.01 sec)
```

一旦修改了一个全局服务器变量，在服务器关闭之前，这些变量的新值一直存在，但在服务器重新启动之后，这些变量将恢复到默认状态，这就意味着如果需要采用一个非默认的设置，就要在每次启动后手工进行设置。因为这样做十分麻烦，所以有必要知道进行永久修改的方法。这可以通过在一个选项文件中设置变量来实现（参阅第13章的介绍）。

一旦已经得到了按照希望要求设置的表高速缓存和索引缓冲，就可以将注意力集中到MySQL使用的其他不同内存缓存上面：

- 可以提高使用ORDER BY或GROUP BY语句查询的速度，通过增加MySQL分类缓冲值的方法分类结果设置，通过sort_buffer变量进行控制。还可以考虑增加read_rnd_buffer_size变量的值，以提高读分类行的速度。

- 增加 read_buffer_size 变量的值可以提高 SELECT 查询的速度，该变量值的增加扩大了 MySQL 读缓冲的规模，从而使 SELECT 能顺序地扫描表。
- 在执行一项作业时，MySQL 保存高速缓存中组成作业的语句，直到收到指示写信息到二进制日志为止，然后将这些信息提交到数据库。对于比较长或者复杂的作业，应该通过 binlog_cache_size 变量增加高速缓存的规模，以便得到更好的性能。
- 如果准备采用所谓的“批量插入”（使用一次 INSERT 命令便插入很多记录），则可以通过增加 bulk_insert_buffer_size 变量值的方法来提高速度。然而，这只能在 MyISAM 表类型中使用。
- 如果预计在服务器上会增加很多新的连接，增加 thread_cache_size 变量值的做法很好。当正在工作的客户机断线时，这个变量控制着服务器线程经过的高速缓存的大小。然后这个高速缓存的线程将重新用来服务一个新连接。这个值越高，可以被高速缓存的线程越多，对新连接申请的响应时间越合适。

16.8 基准技术

修改服务器配置和优化查询的做法都很好，但是如何验证这些修改的结果，并测试是否其性能有适当的变化？MySQL AB 提供的功能已经能满足你的要求——MySQL 使用了一个基准技术套件，名为 MySQL Benchmark Suite，这个技术正如其名称所言：其主要功能是为数据库服务器查找弱点、确认对 SQL 标准的符合情况、测量性能等。

更多的基准技术

MySQL 基准技术套件不仅仅是一种游戏，通过网站 http://www.mysql.com/portal/ software/ item-222.html，还可以得到 SuperSmack 的代替产品；通过网站 http://osdb. sourceforge.net/，可以得到开放源代码数据库基准技术版本。完整版本的基准技术工具可以通过下列网站得到：http:// www.mysql.com/portal/software/ benchmarks/index.html和http://www.wiley.com/legacy/ compbooks/ pachev/。

为了能够运行MySQL基准技术套件，必须在系统中安装Perl，包括Perl DBI套件和MySQL数据库驱动程序（DBD）。如果正在使用一个原始版本的 UNIX 系统，完全有可能已经安装好了这些套件。测试程序是否存在的简单方法是在 shell 提示后面运行下列命令：

```
[user@host]$ perl -e "use DBI"
```

如果Perl不存在错误，则说明已经安装了这个模块。如果模块没有安装，可以从下列网站：http://www.cpan.org/下载，然后根据使用平台的安装手册进行安装。在本书第20章中，可以看到更多关于这个程序的内容，第 20 章讨论了使用 Perl 和 MySQL 开发应用程序的问题。

MySQL 基准技术套件在 MySQL 安装的 sql-bench 目录下面。它包括了很多脚本，同时还有不同基准测试使用的原始数据（参阅下一页的补充资料可以了解有关基准技术的更多内容）。表 16.1 列出了脚本列表，包括每个脚本的基本说明。

启动MySQL基准技术套件最简单的方法是进入sql-bench目录，然后使用run-all-tests脚本运行所有测试，类似下面方式：

```
[user@host]# /usr/local/mysql/sql-bench/run-all-tests --server=mysql
```

测试时间

MySQL 基准技术套件包括了对 Wisconsin Benchmark 程序的支持，后者是一种被广泛接受的关系数据库系统的基准技术。受控的单独用户实验使用所谓的“综合”数据，而不是使用“真实”数据，这项基准技术主要设计用来使用公制计算时间测试每次查询的查询性能。测试情况包括选择、插入、更新、表连接和数据集合等。

表 16.1 MySQL 基准技术套件中的测试实例

脚本名	测试情况说明
copy-db	测量两个服务器之间的复制速度
crash-me	利用运行不同的查询的方式测验 SQL 服务器的能力
innotest	针对事务（InnoDB）和非事务（MyISAM）内容重点测验插入、更新和提交方面的能力
test-alter-table	测量 ALTER TABLE 命令的性能
test-big-tables	测量使用极大表的性能
test-connect	测量服务器的连接速度
test-create	测量创建和放弃表的速度
test-insert	测量向表中插入记录并在其中运行 SELECT 查询的速度
test-select	测量使用多部分列索引的 SELECT 速度
test-transactions	测量事务回滚的速度
test-wisconsin	执行 Wisconsin 基准技术测验
test-ATIS	执行 ATIS 基准技术测验

图 16.1 展示了测试输出的具体情况。

图 16.1 run-all-tests 脚本的输出

正如图中所示，这个脚本运行了表16.1中列出的各种测验的实例，并分别返回了各自占用的时间。每项测验实例细节的进一步信息可以不加任何参数通过调用相应的脚本得到，在启动之前，大多数脚本都会打印这方面的简要说明。

如果只希望运行具体测验，也可以这么做，简单地调用相应的脚本。考虑下面的示例，这里运行了 test-connect 脚本来测量连接到服务器并向其发送数据占用的时间基准：

```
[user@host]# /usr/local/mysql/sql-bench/test-connect
```

图 16.2 展示的是这个操作的具体情况。

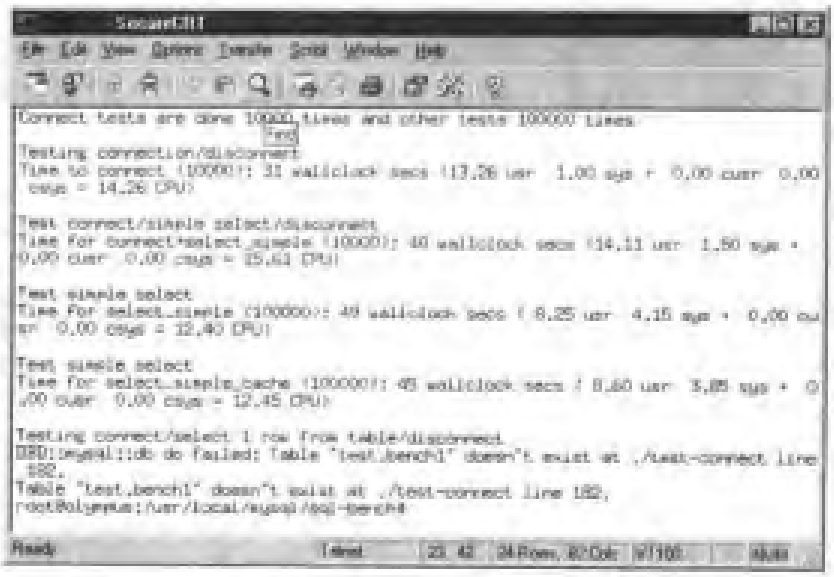

图 16.2　test-connect 脚本的输出

在 MySQL 基准技术套件的各种测验脚本中，十分著名的是 crash-me，这是专门设计用来评估 SQL 服务器能力的实用程序，它使用把服务器推到（并超过）极限的方法测试。crash-me 的工作是发送各种合法的 SQL 查询给服务器，接收其响应，确定服务器的功能设定、能力和极限。图 16.3 展示了运行 crash-me 的输出情况。

图 16.3　crash-me 运行示例

重要提示

为了测验服务器的功能性，crash-me 把数据库服务器推向极限。这在脚本运行中完全可能影响到系统的性能，并且可能造成服务器崩溃。因此使用时一定要小心，切忌在生产服务器上面运行。

如果希望看到MySQL数据库程序和其他厂家提供的数据库程序比较试验的一些结果，可以登录网站：http://www.mysql.com/information/ crash-me.php。应该注意其试验是由MySQL AB进行的，使用自己实用程序设置的准则，所以所有的结果可能存在一定的夸大。因为这些信息涉及很多基准技术，所以其他厂家也会提供自己的数据，或“独立”的基准技术数据，表明其产品的现场试验优秀。所以看到这种不同的结果就毫不奇怪了，它们只不过给数据库在线论坛添加了讨论的话题而已。

16.9 小结

本章探讨了MySQL性能优化的重要主题，讨论了一些可以帮助提高MySQL运行速度的技术和选项。这些技术涉及到查询优化的各种不同策略，包括：

- 使用索引提高频繁访问的列的查询速度。
- 使用新的MySQL查询高速缓存来改善查询响应时间。
- 使用EXPLAIN SELECT命令分析查询，理解并改善查询计划。
- 重写子查询，作为连接发挥出MySQL优化算法的优势。
- 使用服务器变量、集合性能和分类使多表查询更加有效。
- 使用INSERT…SELECT命令或临时表创建数据子集，简化复杂查询的处理过程。
- 选择适当的字段大小，从数据库中删除不必要的表。
- 调整服务器高速缓存和内存缓冲，以便得到更高的性能。
- 使用MySQL基准技术套件调整服务器性能的基准，评估变化结果。

这些技术（以及在在线MySQL手册中列出的其他技术）对于希望有效操作数据库来说是非常基本的，应该成为每一位管理员工具箱中的内容。

第17章　MySQL复制

在第15章中讨论过备份数据库的问题，可以采取数据库快速转储和复制数据库到其他位置等方式。复制是另一种单独的技术，复制的过程是动态的，要求涉及两个或更多的具有连续转换关系的服务器参与，以便使在第二台服务器上面保存原始数据库的一个备份。

本章讨论复制的基本原理，介绍如何使用MySQL设定主副复制系统，同时讲解实现这种管理所使用的命令。

17.1　复制基本原理

MySQL中的复制是在一个主数据库和一个或多个副数据库之间接近实时地同步数据的动态过程。目前建议MySQL复制使用单向方式。

执行复制的能力在MySQL发布功能中还是一项相对很新的技术，从3.23版本才开始出现。复制对于许多应用程序来说是很基本的，缺少复制支持是MySQL与其他关系数据库管理系统（RDBMS）相比较的一个缺陷。复制仍然是MySQL发展中的一项工作，所以需要尽可能地运行MySQL的最新版本。旧的版本缺少某些功能，并且对复制有一些约束，而在更新的版本中已经取消了这些约束。MySQL最适合进行单向复制，其中存在一个主服务器，一个或多个副服务器。

设定复制功能有三个理由。理由之一是创建备份数据库服务器，如果主服务器出现故障，备份服务器可以继续运行，并且是非常迅速的。对于任何有关键任务、时间敏感任务性质的单位来说，这是必须的。

使用复制的理由之二是不需要停止操作或锁定主服务器就可以进行备份。在复制发生之后，备份是在副服务器上进行的，而不是在主服务器上面。这样主服务器可以不受干扰地完成其他工作。

使用复制的理由之三是必须在多个位置上保存当前数据。如果一个单位有几个分支机构，需要使用相同数据库的当前副本进行工作，这时复制就是必要的了。

现在已经明确了为什么需要设定复制的理由，再来看一下关于复制的一些基本概念。

避免版本不同的情况

在主副服务器上面应该尽可能地使用相同版本的MySQL。版本不同有时会导致不稳定复制操作的结果发生。

17.1.1　主副关系

如前所述，复制至少需要两个服务器。服务器需要进行设定，所以第一个服务器称为主服务器，将与其他服务器建立一种关系，后者称为副服务器。主服务器上的数据库最近进行的修改定期被传送到副服务器上。通过这种复制关系，被更新的数据库可以传递到企业的多个副服

务器上面，但是只有主服务器可以在任何时间建立复制关系。从技术的角度考虑，如果需要的话，可以把副服务器提升为主服务器。

如果需要开始服务器的复制过程，主副服务器必须同步，这样数据库才能相同。实现了同步之后，要保证所有的更新都是针对主服务器的，而不是针对副服务器的，这样就避免了更新次序的混乱。

更新是通过主服务器的二进制更新日志从主服务器传递到副服务器上的。复制的基本思想是：主服务器通过二进制日志保持着对数据库变化的跟踪，同时根据相同日志上记录的变化，要求副服务器更新其数据库副本。这样二进制更新日志必须在主服务器上启用以进行复制。

在主副复制方式中，复制过程从副服务器联系到主服务器并且要求进行更新时开始。必须赋予副服务器复制的权限。副服务器通知主服务器最近更新发生的二进制日志中的点，然后开始添加新的更新。一旦操作完成，副服务器通报离开的位置，并且定期与主服务器连接，检查下一轮变化。这个过程只要启用，复制就一直延续。

17.1.2 复制线程

复制过程中涉及三个线程：在主服务器涉及一个，在副服务器涉及两个。副服务器上的I/O线程连接到主服务器，并且请求二进制更新日志。主服务器上的二进制日志转储线程发送二进制更新日志给副服务器。一旦将更新日志发送到了副服务器，I/O线程就将读取主服务器发送的数据，并将它们复制到数据目录中的中继日志中。第三个线程也在副服务器上，称为SQL线程，负责读取和执行来自中继日志的查询，保证副服务器与主服务器一致。图17.1对此进行了说明。

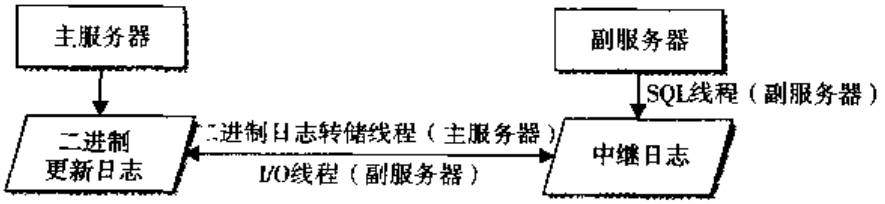

图 17.1 主副服务器复制关系

在副服务器设置的两个不同的线程是MySQL 4.0.2以后的新功能。分开线程的目的是改善性能。通过独立运行的线程，副服务器上的读写过程可以同时进行。因为运行副服务器上的SQL命令所占用的时间，要比读取和复制二进制日志到中继日志的时间长，所以把两个功能分开可以让主服务器提高性能。从主服务器发送的二进制日志更加安全，因为即使对副服务器上的所有更新还没有提交，日志的副本也已经在副服务器上了。

副服务器上的中继日志是以二进制日志相同的格式写的。默认情况下，中继日志保存在副服务器数据目录中。一旦中继日志事件全部执行，SQL线程将自动删除这个日志。当I/O线程开始启动时，会自动创建一个新的中继日志。请注意副服务器并不需要总是连接在主服务器上面，它会保持对离开主服务器时的位置跟踪，然后从该位置开始得到新的数据。

17.1.3 MySQL 复制配置

创建主副服务器以及进行配置的过程相对很直接。在本节中，假定在这些服务器上面没有进行过复制配置，并且希望复制所有的数据库。可以使用另外的方式配置复制，MySQL手册中的常见问题（FAQ）部分有此内容，网址为：http:// www.mysql.com/doc/en/ Replication_FAQ.html。

1. 第一步是授权副服务器可以连接主服务器并进行更新。这是在主服务器上进行的，方法是创建一个副服务器的账户，包括用户名和口令。在 MySQL 4.0.2 和以后的版本中，可以使用 GRANT REPLICATION 命令完成，它可以指定连接到主服务器的副服务器所使用的用户名和口令。例如，命令：

```
mysql> GRANT REPLICATION SLAVE ON *.* TO repadmin@slave
IDENTIFIED BY 'rosebud7';
```

   将授权名为 slave 的副服务器上用户 repadmin 的许可，并且赋予口令 rosebud7。

2. 一旦建立了适当的许可，接下来就是从主服务器向副服务器复制数据库。如前所述，开始时必须准确地进行复制，保证复制准确无误。这个步骤可以通过在主服务器上进行备份，然后恢复到副服务器上来完成（备份和恢复操作的详细内容，参阅第 15 章）。
   在执行这个过程中，要确保不能允许客户机执行更新操作。

注释： MySQL 4.0 及以上的版本有一个 LOAD DATA FROM MASTER 语句，可以提供一种代替方法，传输主数据库给副服务器。但是使用时有一些限制，在尝试使用这个命令之前，应该学习一下使用方法。在 17.2 节中更详细地讨论了 LOAD DATA FROM MASTER 命令的内容。

3. 接下来的步骤涉及到配置主服务器。在选项文件中添加一行语句就可以完成（参阅第 13 章中有关选项文件的信息），赋予主服务器一个复制 ID 编号。复制 ID 必须是 1 到 $2^{32}-1$ 范围内的一个惟一的值。主服务器和副服务器都必须使用惟一值。此外，还应该配置好主服务器，使之启用二进制日志。
   下面是添加到一个选项文件内容的示例：

```
[mysqld]
server-id = 3
log-bin = binary_log
```

4. 配置好这些选项之后，关机并重新启动主服务器。在启动过程中，刚刚设定的新选项将开始生效。现在，所有更新信息将被写到二进制更新日志中。

提示： 如果主服务器中的二进制日志已经启用，关闭并重新启动之前应该对该二进制日志进行备份。当重新启动时，应使用 RESET MASTER 语句来清空现有二进制日志。

5. 现在进入副服务器，并在上面修改选项文件，这样副服务器被赋予一个惟一的复制 ID。还必须提供需要连接主服务器的信息。下面的示例介绍了在副服务器选项文件中必须确定的参数：

```
[mysqld]
server-id = 7
master-host = master
master-user = repadmin
master-password = rosebud7
```

   副服务器的 server-id 必须与主服务器 ID 不同。master-host 是主服务器名。在过程（使用 GRANT 命令）的第一步上需要使用副服务器名和口令，指明 master-user 和 master-password。

警告： 如果主服务器没有在默认端口上面进行监听，还需要确定master-port选项。此外，如果主服务器和副服务器之间的连接经常失败，副服务器试图进行连接的次数将会增加，因此需要增加master-retry-count和master-connect-retry选项的值。

6. 修改完选项文件之后，最后一步是重新启动副服务器。副服务器将使用启动选项来确定如何连接主服务器。副服务器还将在数据目录中创建一个master.info文件，其中包含了所有有关复制过程的信息。在初次启动之后，副服务器将检查这个master.info文件，以便得到指示。如果希望修改复制选项，就必须放弃数据目录中当前的master.info文件，这样就可以在启动时使用新选项进行重新创建了。

注释： 如果希望在复制过程中不包含一个或多个具体的数据库，可回到主服务器的选项文件中，针对复制中不包含的每个数据库添加一个额外的行，以便确定忽略的数据库名。这个行的格式应该是：

```
binlog-ignore-db = database-name
```

17.2 管理复制过程

假设主副服务器都运行得十分稳定，有一些命令可以管理服务器之间的关系。所有这些命令都是在MySQL界面上运行的。在试验这些语句的过程中，还需要进一步学习复制的细节知识。

CHANGE MASTER 命令

CHANGE MASTER要求副服务器检查主服务器中不同的二进制日志，以便进行更新和在副服务器中写到不同的中继日志。这个语句还用于修改连接和二进制日志参数。下面是命令的使用情况：

```
CHANGE MASTER TO
  MASTER_HOST = 'master-name',
  MASTER_USER = 'slave-name',
  MASTER_PASSWORD = 'slave-password',
  MASTER_PORT = 'port-number',
  MASTER_LOG_FILE = 'binlog-filename',
  MASTER_LOG_POS = 'binlog-position',
  MASTER_CONNECT_RETRY = 'value';
  RELAY_LOG_FILE = 'relaylog-filename',
  RELAY_LOG_POS = 'value';
```

假设公司刚刚购进了一台崭新的、非常强大的、非常快的专用服务器用于数据库处理（可以想像，这样做十分有必要），希望把主服务器从旧的服务器上改到新服务器上面。下面是输入这些值的一个示例：

```
mysql>CHANGE MASTER TO
  MASTER_HOST ='MrBig',
  MASTER_USER = 'Slave1',
  MASTER_PASSWORD = 'Slav3Pass',
  MASTER_PORT = '3306',
  MASTER_LOG_FILE = 'master-bin.001',
  MASTER_LOG_POS = 7,
```

```
MASTER_CONNECT_RETRY = 15;
RELAY_LOG_FILE = 'slave-relay-bin.010',
RELAY_LOG_POS = 6084;
```

表17.1包括了这些参数的一个快速参考。注意，只有指定的参数将发生变化。如果没有指定参数，现有的值仍然没有变化。这个原则的特例是主机名和端口号参数。如果修改了任何这些参数，MySQL将假定正在修改主服务器，并且自动放弃二进制更新日志名和位置值。如果在修改主机名或端口号之后，没有提供两个新值，这些值将设置为 MASTER_LOG_FILE='' 和 MASTER_LOG_POS = 4。

表 17.1　CHANGE MASTER 选项

参数	说明
MASTER_HOST	主服务器的主机名
MASTER_USER	连接主服务器时使用的副服务器名
MASTER_PASSWORD	连接主服务器时使用的副服务器口令
MASTER_PORT	连接主服务器的端口号
MASTER_LOG_FILE	复制开始时读取的主服务器的二进制日志文件名
MASTER_LOG_POS	复制开始时读取的主服务器的二进制日志文件位置
MASTER_CONNECT_RETRY	连接操作之间等待时间的秒数
RELAY_LOG_FILE	开始复制时执行的副服务器中继日志名（版本 4.0 以上）
RELAY_LOG_POS	开始复制时执行的副服务器中继日志位置（版本 4.0 以上）

CHANGE MASTER 语句删除了现有的中继日志，并且开始一个新的日志，除非已经对 RELAY_LOG_FILE 或 RELAY_LOG_POS 定义了值。CHANGE MASTER 还更新了 master.info 和 relay-log.info 文件。

请注意，如果试图在高负荷环境下运行 CHANGE MASTER 语句，可能会丢失一些查询。如果副服务器的 SQL 线程落后 I/O 线程，这种情况可能会发生，因为当 CHANGE MASTER 命令删除中继日志时，可能会在写内容和当前应该处理什么之间出现一个间隙。

可以有两种方法处理这种情况。一种是停止副服务器线程，直到读取了所有现有的中继日志之后。可使用 STOP SLAVE IO_THREAD 命令并使用 SHOW SLAVE STATUS 和 SELECT MASTER_POS_WAIT()监督读取的过程。另一种方法是对 Relay_master_log_file 和 Exec_master_log_pos 列使用 STOP SLAVE 和 SHOW SLAVE STATUS，检查线程在主服务器二进制日志中的位置。有了这些值后，在 Master_log_file 和 Master_log_pos 选项中指定它们。这样让副服务器在前次关闭副服务器线程的具体的点上开始复制，就不会丢失记录。

PURGE MASTER 命令

PURGE MASTER语句将在一个指定的日期或日志名之前，删除主服务器上的二进制日志。这个命令可用于希望确定仍然需要处理哪些文件的情况。具体句法如下：

```
PURGE MASTER LOGS TO binary-log
```

或者

```
PURGE MASTER LOGS BEFORE date
```

注释：这个语句需要有 SUPER 权限。

假设希望清除名为bin_log.999的以前的所有主服务器二进制更新日志，可以使用下列语句：

```
mysql> PURGE MASTER LOGS TO bin_log.999;
```

START SLAVE 命令

START SLAVE 语句用于开始或恢复复制。使用 MySQL 4.0.2 和以上版本，可以支持IO_THREAD和SQL_THREAD选项。I/O线程从主服务器读查询，然后写中继日志。SQL线程读中继日志并运行查询。

请注意，如果副服务器不能连接主服务器或不能读二进制日志，可能会在不提供任何错误信息的情况自行关闭。不要因为成功发布START SLAVE命令就假设一切都很好，要使用SHOW SLAVE STATUS 监督副服务器的活动。还可以在副服务器数据目录中读.err 文件，以便确认一切正常。

STOP SLAVE 命令

STOP SLAVE 语句用于停止或结束复制。这个命令与 SLAVE START 类似，可以与IO_THREAD 和 SQL_THREAD 选项合并使用。

SHOW SLAVE STATUS 命令

SHOW SLAVE STATUS 语句提供了有关副服务器状态的信息。其句法如下：

```
mysql> SHOW SLAVE STATUS;
```

除了涉及已经讨论过的有关主服务器主机和主服务器用户的某些输出以外，SHOW SLAVE STATUS 还可以提供表 17.2 中包含的信息。

表 17.2 SHOW SLAVE STATUS 命令提供的信息

参数	说明
Connect_retry	试图连接主服务器多少次
Read_Master_Log_Pos	在主服务器二进制日志中由 I/O 线程读的当前位置
Relay_Log_File	当前中继日志文件
Relay_Log_Pos	在中继日志文件中读的当前事件位置
Relay_log_space	所有中继文件的规模

SHOW SLAVE HOSTS 命令

在版本 4.0 中，在主服务器上发布这个命令将显示一个登记作为复制的副服务器列表。这个语句可以提供的一些其他信息，参阅表 17.3。

表 17.3 SHOW SLAVE HOSTS 命令提供的信息

参数	说明
server_id	副服务器的复制 ID 编号
master_id	主服务器的复制 ID 编号
Host	副服务器主机名
Port	副服务器连接的端口号

SHOW PROCESSLIST 命令

SHOW PROCESSLIST语句显示了服务器上的有关线程的信息，这已经在第13章中做过介绍。在复制情境中，这个命令可以用来得到主副服务器双方的状态信息。对于每个线程，其输出使用不同列显示，部分命名和说明见表 17.4。

表 17.4　SHOW PROCESSLIST 命令提供的信息

参数	说明
State	包含了有关服务器正在处理内容的信息。例如在binlog 转储线程中，可以看到类似发送 binlog 信息给副服务器的信息。在 I/O 线程中，可能看到连接主服务器或请求binlog 转储的信息。在SQL线程中，常见状态是执行查询。此外还可以读中继日志的事件
db	线程数据库
Command	线程执行的当前语句
Time	当前被执行的线程长度（秒数）

```
mysql> SHOW PROCESSLIST;
+----+-----------+-------------------------+-----------+--------
-----+-------+-------------------------------------------------------
------------+------------------+
| Id | User      | Host                      | db        | Command
| Time | State
| Info             |

+----+-----------+-------------------------+-----------+-------------+----
--+-----------------------
-----------------------------------------+------------------+
| 30 | root      | localhost                 | northstar | Query       | 0    |
NULL
| SHOW PROCESSLIST |
| 33 | jay_slave | jay.vermonster.com:44527 | NULL      | Binlog
Dump | 139  | Has sent all binlog to slave; waiting for binlog to
be updated | NULL             |
+----+-----------+-------------------------+-----------+--------
-----+-------+-------------------------------------------------------
------------+------------------+
2 rows in set (0.00 sec)
```

LOAD DATA FROM MASTER 命令

LOAD DATA FROM MASTER 语句可以在 MySQL 4.0 和以上版本中使用，这是一种设置副服务器从主服务器获得数据的代替方法。使用这个命令也有一些限制条件，目前MySQL AB建议只能在主数据库很小以及主服务器对长时间不存在问题的情况设置了读锁的情况下使用。此外只有MyISAM表可以使用这个命令复制。这个命令似乎是针对未来更加频繁使用的情况设计的，这时 MySQL 5.0 不需要对主数据库锁定，以便在执行命令时还可以更新数据库。

这个语句的变体是LOAD TABLE FROM MASTER，它只能从主服务器传递一个表给副服务器。通常使用这个命令是因为副服务器上的具体表被破坏并且需要恢复时。

SHOW BINLOG EVENTS 命令

如前所述，进行复制的时候，一切信息都是基于主服务器上的二进制更新日志的。为了显示这个日志中的事件，可以在MySQL 4.0及以后的版本中使用SHOW BINLOG EVENTS语句，这个日志输出包含的信息参阅表 17.5。

表 17.5 SHOW BINLOG EVENTS 命令提供的信息

参数	说明
log_name	二进制更新日志文件名
pos	日志中事件的位置
event-type	指明事件类型

SHOW MASTER STATUS 命令

如果需要了解有关主服务器二进制日志的其他信息，可以使用 SHOW MASTER STATUS 命令，有关这个命令输出的信息参阅表 17.6。

表 17.6 SHOW MASTER STATUS 命令提供的信息

参数	说明
File	二进制更新日志文件名
position	在主服务器正在写的二进制日志中的当前的位置
binlog_do_db	二进制更新日志中记录的数据库列表
binlog_ignore_db	二进制更新日志中忽略和没有记录的数据库列表

17.3 小结

复制是 MySQL 当前被使用得相当普遍的功能之一。本章介绍了很多复制的基本概念，如主副服务器关系，二进制更新日志建立和中继日志建立等。还讨论了在主副服务器上进行复制涉及的三个线程及其行为分析。此外，还分步骤介绍了进行复制时对两个服务器的处理过程和配置。最后介绍了 MySQL 命令行中管理和监督复制使用的多种命令。这些 SQL 语句对于复制配置和排错十分有用，可以提供有关过程的大量信息。

第四部分

开　　发

第 18 章　MySQL API
第 19 章　MySQL 与 C
第 20 章　MySQL 与 Perl
第 21 章　MySQL 与 PHP

第18章　MySQL API

也许你还记得，在本书前面的介绍性材料中曾说过，MySQL之所以普及是因为得到了广泛的应用程序支持。与其他很多同类型的著名的商业软件不同，MySQL通过一系列的API向开发者开放自己的内核，因此使他们能够自由地选择自己的语言创建数据库驱动的应用程序。

后面的几章将深入探讨API，通过示例和图示解释如何使用它们来发挥MySQL的优势。本章将介绍这方面的内容，同时为读者提供可以使用的不同种类API的一些基本背景，讲解这些API能够做些什么，以及如何根据需求选择不同的API等。

18.1　MySQL API概述

应用程序编程接口（API）到底是什么？

从技术术语上讲，API是“一种描述程序员如何提出应用程序请求的公开规范”。用更通俗的话讲，API是一个接口，是两个或多个应用程序之间进行通信的渠道。一般它包括一组事先定义好的功能或方法，程序员可以调用它们，以得到某些结果或行为。在不同的应用程序组件之间或应用程序本身之间，API可以提供有效的、协调的和可以重复使用的机制。

API中最著名的一个例子就是Microsoft Windows API，几乎被现有的每一种Windows应用程序所使用。这种API公开了需要创建应用程序窗口、构建应用程序菜单、生成文件、分配内存以及以各种不同方法与操作系统（OS）相互作用的功能和方法。通过API调用这些功能，Windows可以确保其中运行的每一种应用程序都是以统一的方式运行的。

MySQL开发者已经提供了完全适合MySQL内核的API。原型的API是在3.19版本上提出来的。经过数年时间，它已逐渐发展成为一种全功能的、功能强大的工具包，能够构建定制数据驱动的应用程序。现在的MySQL API可以在多种不同语言上面使用，包括C，Java，PHP，Perl和Python（在后面的章节中，将详细讨论部分语言）。

如果要全面了解MySQL API的使用价值，可以设想一下如果没有API情况将会怎样。如果没有API为开发者提供与数据库服务器的连接、运行查询和处理结果，则与服务器进行的所有交互操作将需要通过为客户机提供的实用程序进行传送，最著名的程序是mysql客户机程序（当前是MySQL软件组中允许进行任意查询的惟一的工具）。

是否存在问题？mysql客户机程序是一种单纯的应用程序，几乎只能运行SQL查询，返回表式的结果。它不允许进行高级处理，如进入记录集、格式化记录成为HTML网络页面、有效化输入、安全数据库连接或处理错误等。

因此有时需要更加高级的功能，特别是在基于网页或数据驱动的应用程序方面，这些应用程序通常有特殊的要求，可以格式化和展示被检索出来的数据，并且可能还需要更高水平的冗余，特别是当处理数量庞大和复杂的事务时，比如一个电子商务网站或一个商务对商务（B2B）交换等。创建出一种通用的工具来处理这些情况是不可能的，特别是因为每种应用程序都具备自我特殊需求。此外，当程序员需要这些功能的时候，一种更为有效的方法是让他们能够自由

地访问需要的功能。因此无论以什么形式，他们使用这些功能的能力，都应该能够适合应用程序的具体要求。

因此通过开放 API，使得开发者能够访问到MySQL的所有功能，MySQL的开发者不但能够提供针对这种问题的广泛的解决方案，而且还可以使开放源代码社区的成员感到高兴。增加对 MySQL 内核的访问级别，可以使应用程序功能更加强大和具有创造性。开发者可以通过使他们的应用程序被 MySQL 开发组以及社区的其他人所掌握来从中得到好处。

18.2　MySQL API 组件

MySQL API 可以为开发者提供一些具体能力，下面分别简要予以介绍。

- **连接管理**　在一个应用程序能够与数据库交互并且对其中保存的数据进行查询之前，必须首先建立与数据库服务器的连接。MySQL API 包含了专门针对应用程序连接到 MySQL 数据库服务器的一系列功能，可以提供各种申请证书，并且为以后的通信开放一个渠道。一旦所有通信完成，API还提供了一些功能来终止连接和释放被使用的内存，清除事件。
- **实施查询和处理结果**　一旦连接建立，应用程序可以向服务器发送查询，并检索结果。MySQL API 提供了很多功能来简化这个过程，包括执行单个查询或者作为一个事务块的查询功能、检索和处理记录集合、收集来自集合的具体记录或来自记录的具体字段等。
- **报告错误**　MySQL API 还提供了基本错误报告能力，当连接或查询出现问题时，能向调用者传递错误代码和信息。除此之外，API还允许调用者访问详细的排错和管理信息，包括当前服务器状态和所有活动过程列表等。

注释： MySQL API 不能提供自身异常处理例程，它只能报告错误，而将处理这些错误的任务留给应用程序开发者。

- **辅助功能**　除了前面介绍的各种功能外，MySQL API 还可以提供很多所谓的管理功能，包括返回服务器的当前状态（打开连接的数量、得到处理的查询数量、服务器版本等）；列出可以使用的数据库、表、字段和过程等；管理字符集；提供详细的诊断和排错信息；通过关闭不活动的连接和终止不活动的过程来清理没有使用的内存；关闭、重启或重载服务器等。

18.3　语言支持

MySQL为很多不同的编程语言开放了API，使得可以使用自己选择的语言编写数据库驱动的应用程序。目前，MySQL提供的接口 API 编程语言有 C，C++，Java，PHP，Perl，Python，ODBC，Ruby，Eiffel 和 Tcl 等，应确保在构建自己的 MySQL 应用程序时不会因缺少选择而造成问题。

下面简要介绍 API 中各种语言的情况。

- **C 和 C++**　因为 MySQL 是用 C 语言写成的，所以对 MySQL 来说，显然 C 语言的 API 是这里介绍的各种选项中功能最全面的一种。C 语言的 API 通过 libmysqlclient 库与 MySQL 发布捆绑在一起，并且可以被 MySQL 附带的所有命令行工具使用。这些工具对

于C开发者来说是理解API精髓的一个丰富的信息源。C库也是本节中讨论其他API（Java API除外）的基础，和C++开发一样，它可以工作得很好。

注释： 使用C语言的MySQL编程的详细内容请参阅本书第19章，可以通过网站http://www.mysql.com/documentation了解更多有关MySQL C API的知识。

- **PHP** 超文本预处理器（PHP）是一种脚本语言，使用嵌套命令在HTML网页上面工作，因此可以把十分复杂的业务逻辑加入到其他静态网站上。这种语言由于其功能、灵活性和使用方便而很快得到网页开发者的青睐，因为它完全能够支持不同的RDBMS，所以成为中小型数据驱动的Web应用程序的常见选择。PHP最新版本PHP 4内建支持MySQL，并且具有全功能API，可以处理MySQL连接、查询和结果集。

注释： 本书第21章将详细讨论MySQL中使用PHP编程的方法，通过网站http://www.php.net也可以学到更多的有关PHP的MySQL API知识。

- **Perl** Perl可以通过其DBI模块支持MySQL，这个模块提供了一个普通抽象层，可以与各种不同数据库进行通信。DBI为数据库的访问、查询、结果集处理和错误处置提供了常用功能。它可以匹配专门针对数据库的数据库驱动（DBD），可以负责把这些常用功能翻译成为RDBMS可以理解的格式。文档完善的MySQL DBD允许开发者十分容易地对其Perl应用程序添加MySQL支持，这种DBD长期以来一直能够供Perl使用。

注释： 本书第20章将详细讨论MySQL中使用Perl编程的方法，通过网站http://dbi.perl.org也可以学到更多的有关Perl DBI和MySQL DBD的知识。

- **Python** MySQL-Python连接性是由Python MySQLdb模块提供的，这是“一种对可以提供Python数据库API的常见MySQL数据库服务器线程兼容的接口”。这种Python模块可以为Python开发者提供所需要的方法，以便连接MySQL数据库服务器，实施查询、处理结果和处置错误，所有这些方法都是Python能够理解的结构化面向对象的方法。

注释： 通过网站http://sourceforge.net/projects/mysql-python可以得到更多有关Python MySQLdb模块的知识。

- **Java** 目前，MySQL中的Java数据库连接性（JDBC）可以通过Connector/J驱动程序提供。这种驱动程序由MySQL AB以商业模式进行开发并提供支持，可以允许开发者通过把JDBC命令翻译成为MySQL本机格式的方式，在Java应用程序上添加MySQL支持（使用与Perl DBI类似的双层方式）。最新版本的Connector/J驱动程序可以提供对各种不同字符集（包括Unicode）的功能特点和支持。

注释： 通过网站//www.mysql.com/products/connector-j可以得到更多有关Connector/J的知识。

- **ODBC** 通过32位的MyODBC驱动程序，MySQL可以支持Linux和Windows两种平台的开放数据库连接性（ODBC），使得通过任何兼容ODBC的应用程序可以连接到MySQL，包括类似Microsoft Word和Microsoft Excel的应用程序。这种MyODBC驱动程序可以实现ODBC API所有的功能，处理MySQL的连接请求，实施SQL查询和把结果返回给调用者。

注释： 通过网站http://www.mysql.com/products/myodbc可以得到更多有关MyODBC的知识。

18.4 选择 API

在开发应用程序时，因为可以使用的方法非常繁多，因此会对到底使用哪种API感到茫然。一般情况下，开始时会遇到问题，需评估不同选项，看哪种可以作为最佳解决方案（最有效和费用效益比高）。前面列出的每种编程语言都有其自身的优势和弱点。所以一定要经过仔细考虑各种因素（技术和非技术方面的）之后才能做出决定。

平台和应用程序要求

通常，正在开发使用的平台决定了对API或编程语言的选择。虽然大多数语言都能够支持Intel 80x86结构，但是对一些不太普及的体系结构，选择起来可能就没有这么宽松了。这种情况下，在选择API时，能够使用的语言和开发环境将起到决定性作用。例如C编译程序可以在大多数体系结构中使用，但是PHP和Perl解释程序可能就不行。

在决定选择编程语言和API时，应用程序的要求也会起到重要作用。不同的语言可能具有不同的特点，如Perl对于文本处理方面具有优势，而C和C++可以提供对内存管理的更精密的控制。重要的是要利用语言的长处，满足应用程序的实际要求。

如果正在编写的应用程序要求一种带有很小内存覆盖的设备，比如使用闪存卡，则可以选择使用C API，因为使用C可以提供很小的快速的二进制控制，在功能不是很强大的设备上，能够提供更为广泛的资源管理控制。然而这种功能可能对基于MySQL的Web应用程序不适合，因为其中的内存管理因素经常放在速度、可靠性和可以提高开发速度的高级结构能力因素的后面。

性能

十分明显，应用程序的性能在选择API和语言中是一个十分重要的决定性因素，总体目标是让应用程序更快，更加方便用户。假设其他条件都相同，则基本的决定因素就是希望对于应用程序使用一种编译语言（如C），还是一种解释语言（如Perl）。

两种方法都有其优缺点。编译代码的执行速度通常要比解释型代码快，因为执行编译程序时，就已经把源代码转换成机器语言，因此与解释型代码相比，其速度有了实质性的改善，处理速度会很快。但是，解释型语言是一种很好的代码开发和排错工具，因为这些代码在每个阶段的测试之前不需要进行编译。

虽然解释型语言可以提供更快的开发时间等优点，但是编译语言允许对资源使用和内存分配进行更精细的控制，这种能力在资源紧俏、内存缺乏的平台上工作时可能很有用。编译代码在结构之间一般不可移植，应用程序在即将运行的每种结构上需要进行重新编译，而解释型程序则可以在任何平台或结构上面运行，只要源代码的解释型程序是可用的。

在此介绍的各种API中，C和C++提供了最快的速度，可以根据应用程序的要求，与Java，PHP和Perl交互使用。对于需要占用大量时间与MySQL服务器通信的应用程序，PHP因为可以提供很多的时间余量，所以更容易在性能上取胜。这是因为PHP能够最直接地访问本地C API调用，而Perl和Java必须要承担DBI或JDBC层的开销。

简单性

与实际生活一样，KISS原则（尽量简单扼要）适用于很多软件工程，简单化和容易使用的原则在决定使用哪种API时成为重要因素。大多数实际的项目开发时间都很紧张，费用压缩，

即使在具体语言可能在技术上最适合应用程序的情况下，如果开发的代码属于时间和资源紧密型的，也可能无法通过成本效益测试。在这种情况下，时间和费用的因素可能要比使用最优化的技术更重要，退而求其次可能会使解决方案更简单。

以C和C++为例。虽然这两种语言功能都特别强大，并且允许开发者完全控制性能的细致之处和管理资源，但是使用它们来编写简单应用程序，通常是件相当麻烦的任务，需要更多时间和精力。比较而言，MySQL内置支持的PHP可以提供相当多的高水平结构，使得基于MySQL的应用程序占用系统时间很小。

简单化的两个重要因素是可读性和使用方便。通常PHP和Perl中的代码都比其他类似语言（C或Java）更容易阅读和理解。

当代码的新部分被嫁接到已经存在的应用程序上时，在提高开发者使用语言的方便程度上和减少传递时间方面，这种可读性可以起到很大的作用。

中间路线

Java介于编译与解释两者之间。Java代码首先将转换成字节代码，这个过程类似于编译。然后这种字节代码被所谓的“虚拟设备”进行解释并运行。这种两个层次的方法从两个方面可以得到最佳和最遭的结果：Java程序是出名的可移植和安全的程序，但是这种额外的步骤可能导致速度方面特别明显的降低。

可提供的库和工具

可能影响选择API的另一个重要因素是可以为这种语言提供的十分复杂的库模块。这种额外的库的提供实际上可以减少开发新功能时花费的时间，而同时通过使用经过测试的、可以公开使用的代码模块改善了可靠性。一般情况下，开发者更喜欢带有大量可以自由使用的库函数的语言，因为就不必需要每次向应用程序添加新特性或功能时重新进行编码。

Java，Perl和PHP的普及，导致了大量在线式的代码库，包含从文件操作到数据加密等各种各样的库模块。这些代码库大大增加了语言的价值，经常使得开发者不选择C语言，因为它带的库太少。

除了预置代码的可用性以外，开发工具的质量和数量也影响着对语言的选择。使用比较复杂的工具，根据高级标准能自动产生代码的语言，或者包含强大的编译程序、排错程序和开发环境的语言，都属于带有原始工具箱语言的首选标准。虽然可以使用任何语言的简单文本编辑器来编写代码，但是如果有IDE，就可以通过简单的点击操作来完成任务，很明显应考虑开发语言的简单性，选择对应用程序整体开发时间有影响的开发语言。

可移植性

可移植性也是在选择语言时需要考虑的一个重要因素。很明显，只能在单一处理器或结构上使用的语言要比可以跨多种处理器和结构工作的语言的价值低。如果应用程序将计划在不同平台上使用，则在实现编写代码之前，需要探讨一下可移植性方面的问题，因为如果处理不当，到了后面阶段问题将会相当严重。

如果应用程序的可移植性是一个考虑因素，通常不能忽略C语言。C编译程序可以在现在的各种平台和结构上使用，从Intel 80x86 CPU到Motorola 68x00 CPU都可以，使用C语言编写的代码也可以运行在其他平台上，不会出现太多困难。如果应用程序的可移植性是一

个考虑因素，Java也是一个好的选择，毕竟，它是最先倡导“一次编写，到处运行”理论的语言。

成熟性

在决定使用一个API时，应该考虑的因素还包括这种语言的成熟性。一种成熟的语言可以满足公开标准，可以支持开发者社区，而且还能够得到广泛的文档支持和在线社区的援助。虽然它们属于非技术因素，但是这些因素在决定选择语言时将起到决定性的作用。十分明显，文档越多、支持越广，也就越容易提高应用程序的开发速度，从而能确保按时提交产品。

类似C，Java和PHP等现代语言可以配备广泛的文档支持，不论是在线式还是离线式的，这些语言周围都聚拢了大量热衷的开发者社区。选择使用这些语言创建的API通常是一个好主意，特别是如果你是一名初学者，或者应用程序运行的环境不理想，更应该考虑这类语言。因为在遇到困难的时候，可以进行学习，查阅离线手册和在线知识库。文档和支持少的语言通常会在某些问题上让你绕圈子，不论这些问题是否属于十分复杂的技术问题。

费用

一个重要的因素且经常在纯技术讨论中会被忽略的因素是：使用某种语言需要的开发工具的费用。开放源代码运动已经保证大多数基本工具是免费的，如编译程序和连接程序等，至少在大多数*NIX环境中是这样的。然而Windows用户可能仍然需要购买某些语言的开发工具。此外，根据计划使用情况，还可能需要购买一些应用程序中将包含的插件模块或库的商业许可。开放源代码也能够帮助减少这方面的费用，但是不能免除所有费用。

这种计算项目所需工具的真正费用是一件重要工作，并且有时会限制使用针对该任务特别重要的某些语言或API，例如类似C和Java语言可能需要投资一些高质量的商业编译程序。PHP和Perl属于开放源代码社区中真正的支持者，可能只需要很少的投资，并且因此毫无疑问地导致在中小规模软件组织中和一些希望降低软件许可开支的组织中更加普及。

18.5　小结

本章介绍了MySQL API。读者已经学习了这方面的要求和基本原理，并且基本了解了API的能力。同时还学习了各种API可以使用的不同语言，并且提供了一些简单说明来帮助你选择合适的语言。在后面的章节中，还将进一步探讨API，并使用它们来构建一些真实的MySQL应用程序。

第19章　MySQL与C

在前面的章节中，已经花了大量篇幅来介绍MySQL的各种命令行实用程序，包括使用mysqldump实用程序检索数据库内容，使用mysqladmin实用程序执行管理任务，运行SQL查询的mysql命令行客户机等等。所有这些工具共享着某些共同属性：它们可以提高速度；可以单独执行某一具体任务；可以全部都工作在客户-服务器模式下；它们全部使用C语言编写。

正如第18章中提到的，MySQL带有一个C语言的API，能够使用C语言快速简便地编写客户机应用程序，以便与数据库服务器交互。这种C API被打包成一个库，全部包含在标准的MySQL发布中，并且可以被MySQL携带的所有命令行工具所使用（包括前面提到的三种实用程序）。鉴于这种关系，毫不奇怪C API可能是各种MySQL API中功能最为全面的一种，事实上，它可以作为所有其他MySQL API（除了Java API）的工作基础。

本章将向读者介绍MySQL的C API，并且概要介绍如何使用API提供的例程和结构编写MySQL客户机应用程序。除了介绍使用C API与MySQL服务器进行连接和通信的进程，本章还将详细讨论使用API功能进行连接管理、实施查询、处理结果集和错误处置等内容。同时还提供充分的示例代码，使读者熟悉使用C语言构建MySQL客户机应用程序的方法。

19.1　C语言

首先简要介绍一下C语言，包括其历史与沿革，同时还要讨论使用MySQL和C API所需的工具。

19.1.1　历史与沿革

大多数C语言的热衷者都希望看到所有程序都以C代码开始第一行。事实上C语言的出现并不为人所瞩目：它是在20世纪70年代由Dennis Ritchie和Kenneth Thompson在贝尔实验室开发出来的。C语言作为UNIX操作系统上的一种高级应用程序语言，基本上是从BCPL起源的，后者是1967年创建的一种与C语言类似的语言（虽然没有推出）。到了70年代后期，Brian Kernighan和Dennis Ritchie公布了第一个正式的语言描述，并且引起贝尔实验室园区以外各界的关注。

计算机厂家受到其功能的影响，开始推广使用这种语言，同时不同结构和操作系统的C编译程序也缓慢地开始显现出来。1983年，行业中注意到已经广泛地采用C语言的情况，ANSI学会成立了一个委员会负责开发一个正式标准的C语言。现在虽然各种C编译程序仍然保持着编辑程序专用功能的子集，但是大多数都能产生符合ANSI C的代码。

在C语言首次创建以后的30多年中，Ritchie和Thompson创建的语言仍然十分强大，包括C编译程序可以应用在各种单独的操作系统上面。UNIX操作系统的成长已经取得了极大的成功。现在几乎所有的UNIX系统都是默认使用C编译程序，并且很多最有用的和最基础的UNIX命令都是使用C语言的代码。因为这种语言很小，定义完整，足以灵活地支持广泛的应用程序，

所以在开发者中的普及性不断增加。现在C语言被用做应用程序的编程，教师用它作为一种工具教授编程基础，在以十亿美元计的项目上，工程师们也选择它作为实现语言。

19.1.2　安装

如果需要把C语言源代码编译成可执行程序，需要使用C编译程序。建议在UNIX系统中使用的C编译程序是GNU编译程序集合（GCC），其中包括对C，C++和Java的支持，并可以通过因特网免费获得Windows和UNIX平台版本。源代码和预编译的二进制代码可以通过http://gcc.gnu.org/得到，同时包括安装指南。虽然可以按照读者选择的平台免费使用不同的编译程序，但本章中的示例假设使用Linux的GCC 3.3。

详细讨论GCC的安装不是本章的范围。如果读者是GCC的新手，应该花一些时间阅读下载文件中的安装指南以及网站中的说明，这些文件可能提供很大的帮助，让你得到自己的GCC开发环境并运行。在遇到困难的情况下，可以在网址http://gcc.gnu.org/onlinedocs/得到在线手册，在网址http://gcc.gnu.org/faq.html中可得到常见问题解答。

如果使用的是UNIX系统，快速检查GCC是否已经在系统中安装好的方法是使用UNIX的which命令。如果可以使用GCC，其程序应该能够返回gcc二进制码的完整路径。

```
[user@host]$ which gcc
/usr/bin/gcc
```

19.2　MySQL与C语言

如果要简化构建MySQL客户机应用程序的任务，MySQL AB提供了一个C API，可以提供一个低等级界面，负责完成涉及SQL服务器交互的大多数常见任务：数据库连接、查询实施、结果集处理和错误处置等。这种C API是通过两个组件实现的：

- 头文件mysql.h，由结构和功能说明组成。
- MySQL库libmysqlclient(UNIX)或mysqlclient.lib (Windows)，包括了C的实际功能。

特殊处理

如果需要在Windows上构建MySQL客户机，需要避免一些额外的圈套。有关这方面的进一步信息以及构建Windows时的遇到问题的排除方法，见网址http://www.mysql.com/doc/en/Windows_client_compiling.html。

如果计划在Windows平台上编译MySQL客户机，还需要有第三个组件：

- 头文件my_global.h，提供Windows需要的额外定义。

所有这些文件都包括在标准MySQL源代码或二进制tarball中，以及在MySQL的二进制RPM中。

- 如果需要使用一个tarball（假设MySQL安装在/usr/local/mysql目录下），则MySQL C库可以在/usr/local/mysql/lib中找到，而头文件可以在/usr/local/mysql/include中发现。
- 如果使用RPM安装了MySQL，其库将位于/usr/lib/mysql/中，头文件在/usr/include/mysql/中。

通常应该使用-L和-I选项，告诉C语言编译器到什么地方查找这些文件，而使用-lmysqlclient选项将连接 MySQL 库的连接程序。

通过示例学习

是否需要查找如何使用 MySQL 的 C API 构建 MySQL 客户机应用程序的例子？ MySQL 源代码发布中大多数客户机都是使用C语言编写的，其源代码都可以免费获得。可以查阅MySQL的源代码发布的子目录client/，这里包括了 MySQL 工具的代码，如mysqldump，mysqlimport 以及 mysql 命令行客户机。

理解如何使用MySQL C API的最佳途径是通过编写一个简单客户机，进行编译和运行。打开最适合的文本编辑器中的一个文本文件，同时输入以下的代码行：

```
#include <stdio.h>
#include <mysql.h>

int main()
{
        /* declare structures and variables */
        MYSQL mysql;
        MYSQL_RES *result;
        MYSQL_ROW row;

        /* initialize MYSQL structure */
        mysql_init(&mysql);

        /* connect to database */
        mysql_real_connect(&mysql, "localhost", "john", "doe", "db1", 0, NULL, 0);

        /* execute query */
        mysql_query(&mysql, "SELECT title, author FROM books");

        /* get result set */
        result = mysql_store_result(&mysql);

        /* process result set */
        while((row = mysql_fetch_row(result)))
         {
fprintf(stdout, "%s - %s\n", row[0], row[1]);
         }

        /* clean up */
        mysql_free_result(result);
        mysql_close(&mysql);
}
```

完成之后，保存该文件（名为 sample.c），并且进行编译：

```
[user@host]$ /usr/bin/gcc sample.c -o sample.bin -I/usr/local/mysql/include
-L/usr/local/mysql/lib -lmysclient -lz
```

-lmysqlclient选项告诉连接程序连接 libmysqlclient 库，而 -I 和 -L 选项告诉编译程序在什么位置分别查找MySQL头文件和库，-o 选项告诉编译程序分配给最终可执行二进制文件的名称。如果没有使用这些变量，GCC 则使用默认值 a.out。

编译选项解释

是否会对为什么要在前面的命令行中添加-lz选项感到奇怪？这是因为MySQL客户机库的新版本包括了对协议压缩的支持，可以减少在客户机与服务器连接之间传输的数据量。这个功能可以通过libz库实现。如果没有告诉连接程序对此的连接选项，连接程序将产生一系列的错误信息并且中断编译过程。

一旦代码完成了编译，应该有一个 sample.bin 的二进制文件。现在就可以在命令提示符下运行这个二进制文件。如果运行正确，应该连接到 MySQL 服务器，运行查询并打印出结果：

```
[user@host]$ ./sample.bin
Mystic River - Dennis Lehane
The Memoirs Of Sherlock Holmes - Arthur Conan Doyle
Hannibal - Thomas Harris
The Dilbert Principle - Scott Adams
Life Of Pi - Yann Martel
The Lake House - James Patterson
Bleachers - John Grisham
Mortal Prey - John Sandford
```

下面进一步分析前面的代码，以便理解使用 MySQL C API 把 C 程序与 MySQL 数据库连接，并且从中检索数据的过程。

1. 程序以传统方式开始，首先包括需要的所有头文件。因为这里准备使用某些 MySQL 专用结构，所以这个列表中已经包含了 mysql.h 头文件。

```
#include <stdio.h>
#include <mysql.h>
```

2. 接下来，需要定义 main()函数，并初始化在后面步骤中将要使用的变量和结构。

```
/* declare structures and variables */
MYSQL mysql;
MYSQL_RES *result;
MYSQL_ROW row;
```

每个结构在一个 MySQL 客户机应用程序的工作流中都起一个特定的作用。最前面的 MYSQL 是最重要的，因为它起到对所有服务器的交互作用的主要点的作用。服务器返回的结果集保存在特殊的 MYSQL_RES 结果集结构中，从这个结构提取的记录在 MYSQL_ROW 结构中表示出来。

3. 要开始与MySQL数据库服务器的通信，首先需要使用mysql_init()结构来初始化MYSQL结构。

```
mysql_init(&mysql);
```

一旦操作完成，可在服务器中使用 mysql_real_connect()方法打开一个连接。

```
mysql_real_connect(&mysql, "localhost", "john", "doe", "db1", 0, NULL, 0);
```

这个函数要求使用很多不同变量：运行 MySQL 服务器的主机名（"localhost"）、连接数据库名（"db1"）和 MySQL 用户名（"john"）及口令（"doe"）等。如果连接成功，这些

函数会返回一个数据库连接句柄，在与数据库通信过程中，可以在整个程序中使用。19.2.1 节中将详细进行讨论。

4. 一旦与服务器的连接建立，mysql_query()函数将被用来发送查询到服务器上并执行。

```
mysql_query(&mysql, "SELECT title, author FROM books");
```

这种查询可以是MySQL服务器能够接受的任何有效SQL查询。请注意，发送到服务器的查询字符串包括一个终止符或\g符号。19.2.2 节将详细介绍mysql_query()命令，可以进一步了解如何用于与带 SELECT 和无 SELECT 的查询相连接的详细内容。

5. 一旦查询已经进行，产生的结果集（如果该查询是 SELECT 查询）会带一个包含mysql_store_result()函数的变量（实际上是一个指向 MYSQL_RES 结构的指针）。

```
result = mysql_store_result(&mysql);
```

现在可以通过mysql_ fetch_row()函数访问结果集中包含的记录，该函数可以每次读一条记录，使之进入特殊的 MYSQL_ROW 结构。这个结构仅仅是一个数组，每个元素代表记录的一个字段，因此可以使用下列标准数组表示法进行访问：

```
while((row = mysql_fetch_row(result)))
{
        fprintf(stdout, "%s - %s\n", row[0], row[1]);
}
```

一旦结果集中的所有记录使用mysql_fetch_row()进行了检索，函数返回一个NULL结果，然后函数mysql_free_result()将负责重新分配结果集使用的内存，而函数mysql_ close()则负责关闭数据库连接。

```
mysql_free_result(result);
mysql_close(&mysql);
```

有经验的开发者将会注意到前面的示例应用程序没有包含任何错误检验。这是为了简便，而在前面的示例中省略了异常处理代码，但是不必担心，在后面章节中将讨论这些问题。

还应该注意，本示例应用程序只是在其需要的数据库和表已经存在的情况下工作。可以从本书的相关网站http://www.mysql-tcr.com得到所需要的SQL代码来创建这些表（包括本章和下面几章中需要的源代码的电子版）。

在接下来的示例中还将看到，前面介绍的 5 个步骤产生了一个基本过程，可以在使用 MySQL C API 构建数据驱动应用程序时反复使用。下面几节将更详细地探讨这些步骤。

19.2.1 连接管理

与 MySQL DBMS 建立连接基本上需要两个步骤。第一个步骤是调用 mysql_init()函数，初始化 MYSQL 结构。这个结构作为今后与服务器交互的基本点。

```
mysql_init(&mysql);
```

接下来，可以使用函数 mysql_real_connect()建立与服务器的连接。

```
mysql_real_connect(&mysql, "localhost", "joe", "secret",
"db1", 0, NULL, 0);
```

正如前面的代码片段所示，函数mysql_real_connect()必须为此提供多个不同的参数，以便初始化一个成功的连接，这些函数包括：前面使用mysql_init()初始化的MYSQL结构、运行MySQL服务器的主机名、登录服务器需要的用户名和口令以及要使用的数据库名等。

改变主意

通常运行查询的数据库在连接服务器时已经通过使用函数mysql_real_connect()指定。然而，还可以通过使用函数mysql_select_db()把一个数据库名作为变量，在任何时间内按照下列方式选择一个新的数据库：

```
/* select the database named "db678" */
mysql_select_db(&mysql, "db678");
```

如果连接建立成功，这个函数将返回一个连接句柄。这个句柄可以用在后面所有与数据库的通信中。在连接失败的情况下，也许会出现这种情况，如因为登录无效或主机名错误等，该函数将返回假。这时的好办法是在继续进行操作之前，经常检查这个脚本中该函数的返回值，参阅下列简单程序：

```
#include <stdio.h>
#include <mysql.h>

int main()
{
    /* declare MYSQL struct */
    MYSQL mysql;

    /* initialize struct */
    mysql_init(&mysql);

    /* attempt a connection and print status */
   if (!mysql_real_connect(&mysql, "localhost", "tim", "abracadbra", "mysql",
0, NULL, 0))
    {
        printf("Could not connect to MySQL server\n");
    }
    else
    {
        printf("Connection successful\n");
    }

    /* close connection and clean up */
   mysql_close(&mysql);
}
```

在此，函数mysql_real_connect()试图使用函数调用中指定的访问参数打开与数据库的连接。如果操作成功，将返回真并打印适当的信息；如果不成功，将返回NULL并打印错误信息。有很多原因可能造成mysql_ real_connect()失败。网站http://www.mysql.com/ documentation上的MySQL手册提供了所有原因的文档，包括每种情况返回的错误代码以及如何处理的代码示例。

在前面的示例中，为了简化问题，用户名、口令和数据库的默认值已经被简单地硬编码到代码之中了。后面的示例将使用更为常用的方法，允许这些参数可以从配置文件中或命令行上读取。

使用函数 mysql_close()，把使用 mysql_init()创建的结构作为参数，就可以终止一个连接：

```
mysql_close(&mysql);
```

更进一步了解

如果稍加注意，可能会看到本章中的示例向 mysql_real_ connect()发送了 8 个参数，而不是前面介绍的 5 个。额外的 3 个参数如下：

- 连接 MySQL 服务器时使用的端口。
- 连接 MySQL 服务器时使用的套接字。
- 控制客户机行为使用的一个或多个可选择标志。

大多数情况下，不必担心这 3 个参数，只需将它们设为默认值即可（按照本章介绍的示例那样）。但是如果有兴趣更多地了解这些内容，可以参阅 MySQL 手册，掌握每个参数的更为详细的信息。

如果程序使用 MySQL 4.1 中的新事务功能，可以使用函数 mysql_autocommit()打开或关闭 MySQL 的自动运行模式。另一种好办法是在调用 mysql_close()之前，调用函数 mysql_commit()或 mysql_rollback()，这样事务将可以正确完成。

```
mysql_autocommit(&mysql, 0);     /* turn off autocommit mode */
/* perform transaction */
if (success) {
      mysql_commit(&mysql);       /* commit the transaction */
} else {
      mysql_rollback(&mysql);      /* rollback the transaction */
}
```

19.2.2 查询实施

一旦连接了数据库，接下来的步骤就是使用这个数据库实施一项或多项查询。C API 的实施办法就是调用 mysql_query()，它可以接受两个参数：mysql_real_connect()返回的连接句柄和需要实施的查询。

```
/* execute query and obtain result set */
mysql_query(&mysql, "SELECT * FROM employees");
```

上面的示例使用 SELECT 查询，当然函数 mysql_query()也可以使用 INSERT，UPDATE 或 DELETE 等命令，下面的程序片段演示了这一点。

```
/* execute query and obtain result set */
mysql_query(&mysql, "DELETE FROM employees WHERE user = 'john'");
```

引用特殊字符

如果查询字符串包含特殊字符，可以使用函数 mysql_real_escape_string()让 MySQL 转义这些字符。

如果查询成功，函数mysql_query()返回0；不成功则返回非0值。从异常处理的观点看，在进一步运行程序之前，测试这个返回值十分重要，如下列片段所演示的。

```
/* check if query was successful */
if (mysql_query(&mysql, "SELECT title, author FROM books") != 0)
{
        fprintf(stderr, "Error in query\n");
        exit();
}
```

下面是一个完整的示例，通过运行查询和返回状态信息，测试查询是否成功。

```
#include <stdio.h>
#include <mysql.h>

int main()
{
        /* declare structures and variables */
        MYSQL mysql;
        MYSQL_RES *result;

        /* initialize MYSQL structure */
        mysql_init(&mysql);

        /* connect to database */
        if (!(mysql_real_connect(&mysql, "localhost", "john", "doe",
"db1", 0, NULL, 0)))
        {
                fprintf(stderr, "Error in connection: %s\n");
                exit();
        }

        /* execute query */
        if (mysql_query(&mysql, "SELECT title, author FROM books") != 0)
        {
                fprintf(stdout, "Error in query\n");
        }
        else
        {
                fprintf(stdout, "Query executed successfully\n");
        }

        /* clean up */
        mysql_close(&mysql);
}
```

如果查询字符串中包含错误，函数mysql_query()将不能返回0，因此会显示适当的错误信息。可以使用函数mysql_error()（将在19.2.4节中讨论）得到更为友好的错误信息。如果没有返回错误信息，就可以继续处理结果集。

19.2.3　结果集处理

如果mysql_query()运行的查询是一个SELECT查询，或者任何其他可以返回数据的查询，函数mysql_store_result()能够用来访问查询返回的结果集并且保存在一个变量中，以便进一步处理。

```
/* get result */
result = mysql_store_result(&mysql);
```

当mysql_store_result()在MySQL连接句柄上被调用时，可以检索最后一次查询返回的完整的结果集并将其保存在MYSQL_RES结构中。这个结构可以使用MySQL的结果集处理函数检索其中的单独记录和字段。

当规模受到影响时

函数mysql_store_result()在服务器中可以检索完整的查询结果集，并且会将其保存在客户机的内存缓存中以方便检索。如果不希望采用这种方式，如遇到客户机的内存不足的情况，可考虑使用函数mysql_use_result()，它每次在服务器中只检索一个行结果集。这样就减少了客户机使用的内存数量，因为在某个时间段内只需要分配一行内存（和通信缓冲器），而不是每次都必须满足整个结果集。

因为mysql_store_result()和mysql_use_result()专门用来处理结果集，所以通常在不需要返回任何数据的语句上面不会调用这些函数，如UPDATE或INSERT语句（尽管这么做也不会有任何影响）。如果在这种查询中调用这两个函数，都会返回结果NULL。

如果在读取mysql_query()返回的结果集时存在任何错误，mysql_store_result()会返回一个NULL值。因此当使用SELECT查询操作时，在继续重复处理查询返回的结果集之前，检查该函数的返回值的做法是明智的。请参阅下列代码片段的演示。

```
/* check for errors in reading result set */
if (!(result = mysql_store_result(&mysql)))
{
        fprintf(stderr, "Error reading result set\n");
        exit();
}
```

一旦一个SELECT查询被执行并成功地得到了一个结果集，接下来的工作就是利用返回的数据。利用这些数据可以做的最基本工作就是进行查找，看其中包含了多少记录和字段。这就是使用函数mysql_num_rows()和mysql_num_fields()的作用。请参阅下列应用示例。

```
#include <stdio.h>
#include <mysql.h>

int main()
{
        /* declare structures and variables */
        MYSQL mysql;
        MYSQL_RES *result;

        /* initialize MYSQL structure */
        mysql_init(&mysql);

        /* connect to database */
        if (!(mysql_real_connect(&mysql, "localhost", "john", "doe", "db1", 0, NULL, 0)))
        {
                fprintf(stderr, "Error in connection: %s\n", mysql_error(&mysql));
                exit();
        }
```

```
        /* execute query */
        if (mysql_query(&mysql,"SELECT title, author FROM books") != 0)
        {
                fprintf(stderr, "Error in query: %s\n", mysql_error(&mysql));
                exit();
        }

        /* check for valid result set */
        if (!(result = mysql_store_result(&mysql)))
        {
                fprintf(stderr, "Error in reading result set: %s\n", mysql_error(&mysql));
                exit();
        }
        else
        {
                /* if valid result set */
                /* print number of rows and columns */
                int numRecords = mysql_num_rows(result);
                int numFields = mysql_num_fields(result);

                fprintf(stdout, "Query returned %d rows\n", numRecords);
                fprintf(stdout, "Each record contains %d fields\n", numFields);
 mysql_free_result(result);
        }

        /* clean up */
        mysql_close(&mysql);
}
```

如果查询成功，将创建起一个 MYSQL_RES 结构，这个结构可以被函数 mysql_num_rows()和 mysql_num_fields()检验，以便得到行和字段的数量。

一旦知晓了其行和字段的数量，函数mysql_fetch_row()就可以用来检索结果集中每行的实际内容。mysql_ fetch_row()的方法可以返回一个特殊的 MYSQL_ROW 结构，这是一个简单的数字化索引数组，可以表示每个行中的字段。现在访问具体字段的内容就像访问相应数组的元素一样简单。

因为函数mysql_fetch_row()每次在一个行上工作，通常用于一个循环，在查询返回的整个结果集上迭代运行，参阅如下示例。

```
#include <stdio.h>
#include <mysql.h>

int main()
{
        /* declare structures and variables */
        MYSQL mysql;
        MYSQL_RES *result;
        MYSQL_ROW row;
        int count;

        /* initialize MYSQL structure */
        mysql_init(&mysql);

        /* connect to database */
```

```
        if (!(mysql_real_connect(&mysql, "localhost", "john", "doe", "dbl", 0, NULL, 0)))
        {
                fprintf(stderr, "Error in connection: %s\n", mysql_error(&mysql));
                exit();
        }

        /* execute query */
        if (mysql_query(&mysql, "SELECT title, author FROM books") != 0)
        {
                fprintf(stderr, "Error in query: %s\n", mysql_error(&mysql));
                exit();
        }

        /* check for valid result set */
        if (!(result = mysql_store_result(&mysql)))
        {
                fprintf(stderr, "Error in reading result set: %s\n", mysql_error(&mysql));
                exit(2);
        }
        else
        {
                /* if valid result set */
                /* iterate over rows and print contents */
                int numRecords = mysql_num_rows(result);
                for (count = 0; count < numRecords; count++)
                {
                        row = mysql_fetch_row(result);
                        fprintf(stdout, "%s - %s\n", row[0], row[1]);
                }
 mysql_free_result(result);
        }

        /* clean up */
        mysql_close(&mysql);
}
```

在前面这个示例中，通过这个查询十分明显地可以看出该结果集只包含两个字段，所以代码只能硬编码以访问行数组的前两个元素。然而如果不知道结果集中每条记录包含多少字段的话，这种方法将无法奏效，例如如果使用的是SELECT * 查询，这种情况很可能遇到。

在这种情况下，采取更为通用的方法，如果不涉及硬编码结果集中的字段数量到程序中，而是在运行时使用mysql_num_fields()得到这个信息，这种做法更为有效。mysql_num_fields()在第一个循环返回的数量被当做第二个循环的基础，打印出一个行中每个字段的内容。

下面重写前面的示例，以便说明这种更为友好、更方便的方法。

```
#include <stdio.h>
#include <mysql.h>

int main()
{
        /* declare structures and variables */
        MYSQL mysql;
        MYSQL_RES *result;
```

```
    MYSQL_ROW row;
    int i;
    int j;

    /* initialize MYSQL structure */
    mysql_init(&mysql);

    /* connect to database */
    if (!(mysql_real_connect(&mysql, "localhost", "john", "doe", "db1", 0, NULL, 0)))
    {
            fprintf(stderr, "Error in connection: %s\n", mysql_error(&mysql));
            exit();
    }

    /* execute query */
    if (mysql_query(&mysql, "SELECT * FROM accounts") != 0)
    {
            fprintf(stderr, "Error in query: %s\n", mysql_error(&mysql));
            exit();
    }

    /* check for valid result set */
    if (!(result = mysql_store_result(&mysql)))
    {
           fprintf(stderr, "Error in reading result set: %s\n", mysql_error(&mysql));
           exit(2);
    }
    else
    {
           /* if valid result set */
           /* iterate over rows */
           /* for each row, iterate over fields and print contents */
           int numRecords = mysql_num_rows(result);
           int numFields = mysql_num_fields(result);
           for (i = 0; i < numRecords; i++)
           {
                   row = mysql_fetch_row(result);
                   for (j = 0; j < numFields; j++)
                   {
                           fprintf(stdout, "%s", row[j]);
                           (j != (numFields-1)) ? printf(", ") : printf("\n");
                   }
           }
           mysql_free_result(result);
    }

    /* clean up */
    mysql_close(&mysql);
}
```

使用函数mysql_fetch_field()，还有可能得到记录中每个字段属性的详细信息。这个函数的返回值是一个MYSQL_FIELD结构，其成员包含了相应字段中的信息，如名称、长度、数据类型、默认值、最大长度、键和允许值等。下面的示例可以说明（访问MySQL手册可以得到MYSQL_FIELD结构每个成员表示内容的更详细信息）。

```
#include <stdio.h>
#include <mysql.h>

int main()
{
        /* declare structures and variables */
        MYSQL mysql;
        MYSQL_RES *result;
        MYSQL_FIELD *field;

        /* initialize MYSQL structure */
        mysql_init(&mysql);

        /* connect to database */
        if (!(mysql_real_connect(&mysql, "accounts.localdomain.com", "john", "doe", "db1",
0, NULL, 0)))
        {
                fprintf(stderr, "Error in connection: %s\n",
mysql_error(&mysql));
                exit();
        }

        /* execute query */
        /* if error, display error message */
        /* else check the type of query and handle appropriately */
        if (mysql_query(&mysql, "SELECT * FROM accounts") != 0)
        {
                fprintf(stderr, "Error in query: %s\n", mysql_error(&mysql));
                exit();
        }
        else
        {
                result = mysql_store_result(&mysql);
                /* iterate over field collection
                while((field = mysql_fetch_field(result)))
                {
                    printf("name: %s, ", field->name); /* field name */
                    printf("size: %d, ", field->length); /* field length */
                    printf("type: %d, ", field->type); /* enum of field data type */
                    printf("max width: %d, ", field->max_length); /* max field value length
in this result set */
                    printf ("nulls allowed: %s, ", IS_NOT_NULL(field->flags) ? "no" :
"yes"); /* if NULL values allowed */
                    printf ("primary key: %s\n", IS_PRI_KEY(field->flags) ? "yes" : "no");
/* if field is primary key */
                }
                mysql_free_result(result);
        }

        /* clean up */
        mysql_close(&mysql);
}
```

其他方法

使用函数mysql_fetch_fields()，完全可以得到一种MYSQL_FIELD结构的数组，每个字段针对一个结果集。

这样，本节中的示例只能处理返回了结果集的查询。但是对于类似INSERT和UPDATE的SQL命令等不会返回任何数据的情况，该如何处理呢？

在不能返回结果集，但是需要对数据库中的记录进行修改（INSERT，UPDATE和DELETE等）的情况下，受到影响的行数量可以使用函数 mysql_affected_rows()得到。参阅下列示例。

```
#include <stdio.h>
#include <mysql.h>

int main()
{
        /* declare structures and variables */
        MYSQL mysql;
        MYSQL_RES *result;

        /* initialize MYSQL structure */
        mysql_init(&mysql);

        /* connect to database */
        if (!(mysql_real_connect(&mysql, "localhost", "user", "pass", "web", 0, NULL, 0)))
        {
                fprintf(stderr, "Error in connection: %s\n", mysql_error(&mysql));
                exit();
        }

        /* execute UPDATE query */
        /* return number of rows changed */
        if (mysql_query(&mysql, "UPDATE data SET vcount = vcount+1") != 0)
        {
                fprintf(stderr, "Error in query: %s\n", mysql_error(&mysql));
                exit();
        }
        else
        {
                fprintf(stdout, "Query successful, %d rows affected\n", mysql_affected_rows
(&mysql));
        }

        /* clean up */
        mysql_free_result(result);
        mysql_close(&mysql);
}
```

从以上代码段可知，查询可以使用函数mysql_field_count()返回一个结果集，以便检查结果集中期望的字段数。如果这个函数返回的是0，可以认为得不到结果集字段。通常这种情况在运行非SELECT查询时能够遇到。因此只有当需要查找最后运行什么类型的查询时，才需要函数 mysql_field_count()，因此需要为该 MySQL 客户机编写方便的控制流例程。

下面介绍的示例使用函数mysql_field_count()，它能确定运行的查询是一个SELECT查询，还是一个非 SELECT 查询，以便确定是否继续处理结果集。

```
#include <stdio.h>
#include <mysql.h>

int main()
{
        /* declare structures and variables */
        MYSQL mysql;
        MYSQL_RES *result;

        /* initialize MYSQL structure */
```

```
        mysql_init(&mysql);

        /* connect to database */
        if (!(mysql_real_connect(&mysql, "localhost", "john", "doe", "test", 0, NULL, 0)))
        {
                fprintf(stderr, "Error in connection: %s\n", mysql_error(&mysql));
                exit();
        }

        /* execute query */
        /* if error, display error message */
        /* else check the type of query and handle appropriately */
        if (mysql_query(&mysql, "INSERT INTO logs VALUES (99, NOW(), 'Apache server
shutdown') ") != 0)
        {
                fprintf(stderr, "Error in query: %s\n", mysql_error(&mysql));
                exit();
        }
        else
        {
                if (result = mysql_store_result(&mysql))
                {
                        /* is there a result? must be a SELECT then...  */
                        /* process result set */
                        fprintf(stdout, "Result set contains some fields, must be a
SELECT query\n");

                        /* free result set */
                        mysql_free_result(result);
                }
                else
                {
                        /* no result? could be either a non-SELECT query */
                        /* or a SELECT with an error */
                        if (mysql_field_count(&mysql) == 0)
                        {
                                /* non-SELECT query */
                                fprintf(stdout, "Result set contains no fields, must be
a non-SELECT query\n");
                        }
                        else
                        {
                                /* error */
                                fprintf(stderr, "Error in reading result set: %s\n",
mysql_error(&mysql));
                                exit();
                        }
                }
        }

        /* clean up */
        mysql_close(&mysql);
}
```

一旦结果集得到处理，一般应该使用函数mysql_free_ result()来释放该结果集占用的内存缓冲空间。即使产生本操作的数据库连接已经终止，没有被释放的结果集也将继续占用内存。

```
/* free result set memory */
mysql_free_result(&mysql);
```

表 19.1 列出了与处理结果集相关的辅助函数。

表 19.1　涉及结果集处理的辅助函数

函数	功能
mysql_field_seek()	跳至该行具体列
mysql_row_seek()	跳至结果集中的具体行
mysql_field_tell()	返回字段列表中字段游标的当前位置
mysql_row_tell()	返回结果集中行游标的当前位置
mysql_fetch_fields()	返回 MYSQL_FIELD 结构的一个数组
mysql_fetch_lengths()	返回当前选中行的字段值长度
mysql_insert_id()	返回产生 AUTO_INCREMENT 字段的最后 ID

19.2.4　错误处置

MySQL C API 中的各种函数都会产生一个返回值，指出它是成功还是失败。因为 API 不能自动处置错误，从而不能绕过和排除这些错误，所以错误解决机制需要由应用程序开发者在应用程序层上实现。在最简单的方式中，错误处置机制包括在一个条件测试中包装各种函数调用，如果返回的是失败代码，则显示错误信息。下列示例明确地说明了这种情况。

```
#include <stdio.h>
#include <mysql.h>

int main()
{
        /* declare structures and variables */
        MYSQL mysql;

        /* initialize MYSQL structure */
        mysql_init(&mysql);

        /* connect to database */
        if (!(mysql_real_connect(&mysql, "localhost", "john", "doe", "db1", 0, NULL, 0)))
        {
                fprintf(stderr, "Error in connection\n");
                exit();
        }

        /* execute query */
        if (mysql_query(&mysql, "SELECT FROM books") != 0)
        {
                fprintf(stdout, "Error in query\n");
                exit();
        }
        else
        {
                fprintf(stdout, "Query executed successfully\n");
        }

        /* clean up */
        mysql_close(&mysql);
}
```

在这种情况下，如果任何函数调用返回一个错误信息[如mysql_query()调用返回的信息，因为查询语句包含了明显的错误]，其中包含的条件测试将捕捉这些错误，显示错误信息并退出。

这种简单的错误处置机制可以用两个功能强大的错误跟踪功能作为补充，这些功能可以更准确地、更为友好地向用户提示错误信息。参见下列示例，其中使用函数mysql_error()和mysql_errno()重新编写了前面的示例。

```
#include <stdio.h>
#include <mysql.h>

int main()
{
        /* declare structures and variables */
        MYSQL mysql;

        /* initialize MYSQL structure */
        mysql_init(&mysql);

        /* connect to database */
        if (!(mysql_real_connect(&mysql, "localhost", "john", "doe", "dbl", 0, NULL, 0)))
        {
                fprintf(stderr, "Error in connection: %s [code %d]\n", mysql_error(&mysql),
mysql_errno(&mysql));
                exit();
        }

        /* execute query */
        if (mysql_query(&mysql,"SELECT FROM books") != 0)
        {
                fprintf(stdout, "Error in query: %s [code %d]\n", mysql_error(&mysql),
mysql_errno(&mysql));
       exit();
        }
        else
        {
                fprintf(stdout, "Query executed successfully\n");
        }

        /* clean up */
        mysql_close(&mysql);
}
```

函数mysql_errno()显示了MySQL返回的最后错误代码，而函数mysql_error()返回了最后错误信息。在前面的某些示例中，已经有过这种情况。

表19.2列出了有关错误处置的辅助函数。

表19.2 有关错误处置的辅助函数

函数	功能
mysql_debug()	向客户机日志文件发送调试错误信息
mysql_dump_debug_info()	要求服务器在服务器日志中写错误调试信息

19.2.5 辅助函数

除了前面讨论的一些函数以外，C API还包含了很多专门用来提供服务器当前状况信息以及列出可用数据库和表的函数。这些函数并不像前面章节中介绍的那些函数一样被经常使用，但是仍然值得花一些时间介绍一下。表19.3简单地列出了这些函数。

表 19.3　涉及版本和状况报告的辅助函数

函数	功能
mysql_get_server_info()	返回 MySQL 服务器版本号
mysql_get_proto_info()	返回 MySQL 协议的版本号
mysql_get_client_info()	返回 MySQL 客户机的版本号
mysql_get_host_info()	返回 MySQL 主机的信息
mysql_list_dbs()	返回 MySQL 服务器上可以使用的数据库列表
mysql_list_tables()	返回 MySQL 数据库可以使用表的列表

下面的示例演示了部分函数的使用情况。

```
#include <stdio.h>
#include <mysql.h>

int main()
{
        /* declare structures and variables */
        MYSQL mysql;
        MYSQL_RES *dbs;
        MYSQL_ROW db;
        MYSQL_RES *tbls;
        MYSQL_ROW tbl;

        /* initialize MYSQL structure */
        mysql_init(&mysql);

        /* connect to server */
        if (!(mysql_real_connect(&mysql, "localhost", "root", "secret", NULL, 0, NULL, 0)))
        {
                fprintf(stderr, "Error in connection: %s\n", mysql_error(&mysql));
            exit();
        }

      /* display client/server information */
      fprintf(stdout, "connected to server on host %s\n", mysql_get_host_info(&mysql));
      fprintf(stdout, "server version %s\n", mysql_get_server_info(&mysql));
      fprintf(stdout, "client version %s\n", mysql_get_client_info());
      fprintf(stdout, "\n");

       /* list databases */
      dbs = mysql_list_dbs(&mysql, NULL);

      /* iterate over database list and print names */
      while(db = mysql_fetch_row(dbs))
      {
            fprintf(stdout, "%s: ", db[0]);
            /* select this database for use and get table list */
            mysql_select_db(&mysql, db[0]);
            tbls = mysql_list_tables(&mysql, NULL);

            /* print tables in this database */
            while(tbl = mysql_fetch_row(tbls))
            {
                  fprintf(stdout, "%s ", tbl[0]);
            }

            fprintf(stdout, "\n");
            mysql_free_result(tbls);
```

```
    }

    mysql_free_result(dbs);

    /* close connection */
    mysql_close(&mysql);
}
```

表 19.4 列出了在某些情况下可能使用的其他一些辅助函数。

表 19.4 其他辅助函数

函数	功能
mysql_change_user()	改变当前登录的用户
mysql_options	为 MySQL 连接设置选项
mysql_escape_string()	转义在字符串中的特殊字符
mysql_shutdown()	强制关闭 MySQL 服务器
mysql_reload()	强制服务器重新装载授权表
mysql_ping()	测试服务器是否在起作用
mysql_stat()	返回服务器状态

19.3 真实范例

假设已经了解了 MySQL C API 的基础知识，现在可以来试验一下这些知识。下面的章节将展示如何在实际环境中使用 MySQL API，可以在其中构建两个 C 语言程序：一个是交互式 SQL 客户机，但是与 MySQL 发布中携带的情况不同；一个是开支跟踪程序，使用 MySQL 作为单独个人开支的存储数据库。

19.3.1 交互式 SQL 客户机

现在已经了解了MySQL配备的命令行mysql客户机（几乎在本书的每一章都演示了RDBMS的不同方面）。整体而言，这个客户机是非常简单的代码，可以连接到 MySQL 服务器上（通过命令行变量指定），用户输入指令，然后发送这些输入信息给服务器进行处理。如果输入有效，服务器将进行处理，并返回一个结果集（对于 SELECT 类查询）或者一个返回值，表示客户机的成功或失败（对于非 SELECT 类查询）。不论什么形式的数据——显示一条信息、处理和格式化结果集（如果结果集存在）或以某种方式使用这些数据，客户机都可以处理这些数据。

如果读者阅读了本章前面的内容，就会注意到所有这些任务都可能通过 MySQL C API 提供的函数进行处理。因为 API 可以承担大多数工作，它十分容易创建一个命令行客户机，并可按照默认 mysql 客户机方式模仿很多同类函数的功能。

请参考以下代码：

```
#include <stdio.h>
#include <mysql.h>

int main(int argc, char *argv[])
{

    /* declare structures and variables */
    char query[255];        /* query string */
    int i, j, h, count;     /* loop counter variables */
```

```
MYSQL mysql;            /* MYSQL struct */
MYSQL_RES *result;      /* pointer to result set struct */
MYSQL_ROW row;          /* row struct */
MYSQL_FIELD *field;     /* pointer to field struct */

/* check for command-line arguments */
/* if no arguments are passed, count is 1 */
/* for empty password, pass "" */
if (argc != 4)
{
        printf ("Usage: %s <host> <user> <password>\n", argv[0]);
        exit();
}

/* initialize MYSQL structure */
mysql_init(&mysql);

/* connect to database with command-line arguments */
if (!(mysql_real_connect(&mysql, argv[1], argv[2], argv[3], NULL, 0, NULL, 0)))
{
        fprintf(stderr, "Error in connection: %s\n", mysql_error(&mysql));
        exit();
}

  /* loop and display prompt until "exit" */
  for( ;; )
  {
          /* ask for query */
          printf("query? ");
          fgets(query, sizeof(query), stdin);
          /* if "exit", break out of loop */
          if (strcmp(query,"exit\n") == 0)
          {
                  break;
          }

          /* execute query */
          /* if error, display error message */
          /* else check the type of query and handle appropriately */
          if (mysql_query(&mysql, query) != 0)
          {
                  fprintf(stderr, "Error in query: %s\n", mysql_error(&mysql));
          }
          else
          {
                  if (result = mysql_store_result(&mysql))
                  {
                          /* SELECT query */
                          /* get some information on result set */
                          int numRecords = mysql_num_rows(result);
                          int numFields = mysql_num_fields(result);

                          /* print field names */
                          for (h = 0; h < numFields; h++)
                            {
                                  field = mysql_fetch_field(result);
                                  printf("[%s]", field->name);
                                  (h != (numFields-1)) ? printf(", ") : printf("\n");
                            }

                          /* process result set records */
```

```
                        for (i = 0; i < numRecords; i++)
                        {
                                row = mysql_fetch_row(result);
                                for (j = 0; j < numFields; j++)
                                {
                                        fprintf(stdout, "%s", row[j]);
                                (j != (numFields-1)) ? printf(", ") : printf("\n");
                                }
                        }
                          fprintf(stdout, "** Query successful, %d rows retrieved **\n",
numRecords);
                        mysql_free_result(result);
                }
                  else
                {
                        if (mysql_field_count(&mysql) == 0)
                        {
                                /* non-SELECT query */
                                /* show number of affected rows */
                                fprintf(stdout, "** Query successful, %d rows affected
**\n", mysql_affected_rows(&mysql));
                        }
                        else
                        {
                                /* else print error */
                                fprintf(stderr, "Error in reading result set: %s\n",
mysql_error(&mysql));
                          }
                }
            }
    }

    /* clean up */
    mysql_close(&mysql);
}
```

如果已经熟悉使用C语言编程，上面任何程序段都不会令你头痛。这个程序的第一部分是查找信息，确认希望连接哪些主机和数据库，然后确定使用什么用户名和口令登录。当调用程序时，这个信息应该由命令行的用户提供。基本错误检查将警告用户在缺少变量的情况下使用正确的句法。

接着，函数mysql_real_connect()调用负责连接到数据库。假设连接已经建立，一个循环将产生提示，并等待用户以SQL命令形式（这是程序的交互部分）输入。用户输入的内容将通过函数mysql_query()发送给SQL服务器进行处理（除非用户输入命令“exit”，这时在程序断开了循环的情况下，将关闭连接并终止）。

根据查询是否返回了结果集的情况，使用函数mysql_fetch_field()和mysql_fetch_row()程序或者产生一个字段列表和记录值，或者通过函数mysql_affected_rows()，打印查询影响的一系列的行总值。每个阶段上的错误处置将保证如果存在错误，将打印到标准输出设备上，包括表明该错误的可阅读的信息。

下面是这类客户机运行中的一些示例：

```
[user@host]$ ./client.bin
Usage: ./client.bin <host> <user> <password>

[user@host]$ ./client.bin localhost root ""
```

```
Error in connection: Access denied for user: 'root@localhost' (Using password: YES)
[user@host]$ ./client.bin localhost root master

query? SHOW DATABASES
[Database]
db1
inventory
mysql
test
** Query successful, 4 rows retrieved **

query? SHOW TABLES FROM DB1
[Tables_in_db1]
categories
data
employees
logs
** Query successful, 4 rows retrieved **

query? USE db1
** Query successful, 0 rows affected **

query? SELECT * FROM employees
[id], [name], [dept], [comp]
103, Miles D, Engineering, 67550.00
105, Susan J, Engineering, 73000.00
126, Joe T, Legal, 90000.00
127, John D, HR, 41000.00
134, Ingrid F, HR, 43600.00
** Query successful, 5 rows retrieved **

query? SELECT id, name FROM employees WHERE comp > 70000
[id], [name]
105, Susan J
126, Joe T
** Query successful, 2 rows retrieved **

query? UPDATE employees SET comp = comp + 5000 WHERE comp < 50000
** Query successful, 2 rows affected **

query? SELECT nothing
Error in query: Unknown column 'nothing' in 'field list'

query? exit
[user@host]$
```

19.3.2　开支跟踪程序

在这种环境下，MySQL 数据库将作为单独个人开支情况的账簿使用，处理各种不同的预先设定类别的情况。一个用户能够在这种类别中通过简单的客户界面输入开支情况。这种输入于是被转换到SQL 查询，并保存到MySQL 数据库中，从数据库可以产生以集合形式进行检索后的报告。

构建这种应用程序的首要工作是设计存放所有必要信息的表，如下所示：

```
CREATE TABLE categories (
  cid int(8) NOT NULL default '0',
  cname varchar(255) NOT NULL default '',
```

```
  PRIMARY KEY  (cid)
)
CREATE TABLE data (
  eid int(8) NOT NULL auto_increment,
  cid int(8) NOT NULL default '0',
  date date NOT NULL default '0000-00-00',
  amt float(11,2) NOT NULL default '0.00',
  PRIMARY KEY  (eid)
)
```

在此假定这个应用程序使用相当简单的一种结构。Categories 表保存了一个类别列表，每个表都有一个惟一编号表示，而data表保存着用户输入的开支记录。数据表中的每个开支项也都用一个惟一的记录ID表示，并具备三个属性：开支日期、数量和应该归属的类别（在categories表中，这是类别 ID 的一个外键）。

下面添加部分类别到 categories 表中，并在 data 表中设定一些初始项目：

```
INSERT INTO categories (cid, cname) VALUES (1,'Entertainment');
INSERT INTO categories (cid, cname) VALUES (2,'Travel');
INSERT INTO categories (cid, cname) VALUES (3,'Gifts');
INSERT INTO categories (cid, cname) VALUES (4,'Repairs');
INSERT INTO data (eid, cid, date, amt) VALUES (1,1,'2003-04-05',568.99);
INSERT INTO data (eid, cid, date, amt) VALUES (2,4,'2003-11-19',100.00);
INSERT INTO data (eid, cid, date, amt) VALUES (3,1,'2003-06-12',250.00);
INSERT INTO data (eid, cid, date, amt) VALUES (4,2,'2003-08-16',50.00);
```

根据这个数据库结构，使用 MySQL C API 创建的这个程序的代码已经成为了一项十分简单的任务，参阅下列示例。

```
#include <stdio.h>
#include <mysql.h>

int main ()
{

   /* declare variables */
   FILE *fp;  /* file pointer */
   char lineBuffer[1000]; /* line buffer */
   const char *sep = ":"; /* field separator in config file */
   char *prev_p;
   long next_p;

   char host[100];   /* database access variables */
   char user[50];
   char pass[50];
   char db[50];

   int choice;          /* menu selection */
   MYSQL mysql;         /* MYSQL struct */
   MYSQL_RES *result;   /* pointer result set struct */
   MYSQL_ROW row;       /* row struct */
   MYSQL_FIELD *field;  /* pointer to field struct */

   /* open configuration file */
   fp = fopen ("expenses.conf", "r");

   if (fp != NULL)
   {
      /* initialization */
```

```
    memset (lineBuffer, '\0', sizeof(lineBuffer));
    memset (host, '\0', sizeof(host));
    memset (user, '\0', sizeof(user));
    memset (pass, '\0', sizeof(pass));
    memset (db, '\0', sizeof(db));

    /* split by separator */
    /* read configuration data into variables */
    fscanf (fp, "%s", lineBuffer);
    if (strlen(lineBuffer) != 0)
    {
       prev_p = lineBuffer;
       strncpy(host, prev_p, next_p = strstr (prev_p, sep) - prev_p);

       prev_p =  (char *) ((long) prev_p + next_p + 1);
       strncpy(user, prev_p, next_p = strstr (prev_p, sep) - prev_p);
       prev_p =  (char *) ((long) prev_p + next_p + 1);
       strncpy(pass, prev_p, next_p = strstr (prev_p, sep) - prev_p);
       prev_p =  (char *) ((long) prev_p + next_p + 1);
       strcpy(db, prev_p);
    }
    else
    {
       fprintf(stderr, "Error: no data in configuration file\n");
       exit();
    }

    /* close file */
    fclose (fp);
 }
 else
 {
    fprintf(stderr, "Error: unable to read configuration file\n");
    exit();
 }

 /* initialize MYSQL structure */
 mysql_init(&mysql);

 /* connect to database with configuration arguments */
 if (!(mysql_real_connect(&mysql, host, user, pass, db, 0, NULL, 0)))
 {
    fprintf(stderr, "Error in connection: %s\n", mysql_error(&mysql));
    exit();
 }

 /* print menu */
 printf("\n");
 printf("[1] View expense totals\n");
 printf("[2] Add expense item\n");
 printf("[3] Quit\n");
 printf("\n");
 printf("Select an option from the list above: ");

 /* wait for user input */
 scanf("%d", &choice);

 printf("\n");

 /* based on user choice, perform actions */
 if (choice == 1)
 {
```

```
      /* perform query to get totals in each category */
      /* and print report */
     if (mysql_query(&mysql, "SELECT cname, SUM(amt) FROM categories, data WHERE categories.
cid = data.cid GROUP BY cname") != 0)
      {
         fprintf(stderr, "Error in query: %s\n", mysql_error(&mysql));
         mysql_close(&mysql);
         exit();
      }
      else
      {
         /* check to see if result set */
         /* iterate over rows and print in category = total format */
         if (result = mysql_store_result(&mysql))
         {
             int numRecords = mysql_num_rows(result);
             int i;
             for (i = 0; i < numRecords; i++)
             {
                row = mysql_fetch_row(result);
                printf ("%s = %s\n", row[0], row[1]);
             }
             printf("\n");
             mysql_free_result(result);
         }
         else
         {
            fprintf(stderr, "Error in reading result set: %s\n", mysql_error(&mysql));
            mysql_close(&mysql);
            exit();
         }
      }
   }
   else if (choice == 2)
   {
      float amt;            /* variables for insert */
      int cid;
      char date[10];

      char query[1024];    /* query string */
      int i;               /* loop counter */

      /* ask user to enter date */
      /* if error in format, exit */
       printf("Enter date (yyyy-mm-dd): ");
      if (!scanf("%s", &date))
      {
         fprintf(stderr, "Error in input: unexpected input value\n");
         mysql_close(&mysql);
         exit();
      }

      /* ask user to enter amount */
      /* if error in format, exit */
      printf("Enter amount: ");
      if (!scanf("%f", &amt))
      {
         fprintf(stderr, "Error in input: unexpected input value\n");
         mysql_close(&mysql);
         exit();
      }
```

```
        /* ask user to enter category */
        /* retrieve list of categories/IDs from database */
        /* if error in format, exit */
        printf("Enter category number (");

        /* get category numbers */
        if (mysql_query(&mysql, "SELECT cid, cname FROM categories") != 0)
        {
            fprintf(stderr, "Error in query: %s\n", mysql_error(&mysql));
            mysql_close(&mysql);
            exit();
        }
        else
        {
            /* print category names and numbers */
            if (result = mysql_store_result(&mysql))
            {
                int numRecords = mysql_num_rows(result);
                int i;
                for (i = 0; i < numRecords; i++)
                {
                    row = mysql_fetch_row(result);
                    printf (" %s=%s ", row[0], row[1]);
                }
                mysql_free_result(result);
            }
            else
            {
                fprintf(stderr, "Error in reading result set: %s\n", mysql_error(&mysql));
                mysql_close(&mysql);
                exit();
            }
        }
        printf("): ");
        if (!scanf("%d", &cid))
        {
            fprintf(stderr, "Error in input: unexpected input value\n");
            exit();
        }

        /* validate date format here */
        /* this is left as an exercise to the reader */
        /* once all the data has come in */
        /* create query string */
        sprintf(query, "INSERT INTO data (date, amt, cid) VALUES ('%s', '%f', '%d')", date,
amt, cid);
        /* perform insert */
        if (mysql_query(&mysql, query) != 0)
        {
            fprintf(stderr, "Error in query: %s\n", mysql_error(&mysql));
            mysql_close(&mysql);
            exit();
        }
        printf("\n");
    }

    /* close connection */
    mysql_close(&mysql);
    exit();
}
```

与前面的客户机不同，这个客户机不能从命令行上接受数据库连接参数。但是可以在相同目录下查找到一个名为expenses.conf的配置文件，并可以从这个文件中读到mysql_real_connect()的参数。这个配置文件相当简单：在host:user:pass:db格式下只需要包含一行即可。代码段列出的前几行涉及到单纯地读取和分析这个文件的情况。

一旦读取了配置文件，程序将试图连接到MySQL数据库，并且如果连接成功，将显示一个选项菜单。如果用户选择使用[1]，将通过常见的cid字段使用SELECT查询把类别和数据表合并到一起并在每个类别中添加数量，以便通过函数SUM()和语句GROUP BY获得合计数据。如果用户选择使用[2]，程序将询问输入日期、数量和类别ID，并且使用这些信息构建和运行INSERT查询，保存记录到MySQL数据库。在菜单提示下输入任何其他选项，都可以使程序关闭MySQL连接并退出。

下面是这个应用程序运行的几个示例：

```
[user@host]$ ./expenses.bin
[1] View expense totals
[2] Add expense item
[3] Quit
Select an option from the list above: 1
Entertainment = 1518.99
Gifts = 250.00
Repairs = 200.00
Travel = 150.00

[user@host]$ ./expenses.bin
[1] View expense totals
[2] Add expense item
[3] Quit
Select an option from the list above: 2
Enter date (yyyy-mm-dd): 2003-12-15
Enter amount: 2450.50
Enter category number ( 1=Entertainment  2=Travel  3=Gifts  4=Repairs ): 2

[user@host]$ ./expenses.bin
[1] View expense totals
[2] Add expense item
[3] Quit
Select an option from the list above: 1
Entertainment = 1518.99
Gifts = 250.00
Repairs = 200.00
Travel = 2600.50
```

19.4 小结

本章讨论了MySQL C API，解释了如何用它来构建C客户机与MySQL服务器通话。MySQL C API提供了很多不同的函数，可以简化任务，包括连接服务器、运行查询和处理结果等。本章还讨论了每个类别中的重要函数，包括如何使用这些函数的示例和说明。

本章通过讨论不常使用的基本原理，还展示了C与MySQL组合的两个实际应用程序：可以接受查询并返回结果集的类似MySQL发布中配备的交互式的SQL客户机，以及一个开支跟踪应用程序，可以接受开支信息，使用这个程序可以产生汇总报告。两个应用程序可以通过MySQL C API实现，以便说明连接数据库、检索和更新表数据以及格式化结果用于显示等任务。

第20章　MySQL与Perl

正如读者在前面章节中所见到的，MySQL的C API可以构建一些功能相当强大的客户机应用程序。然而在开发支持MySQL的应用程序时，C语言并不是惟一选择，MySQL也能够（且经常）与Perl一道使用，开发具有数据库功能的Web应用程序。

Perl是一种针对服务器端脚本的常见语言，这主要是因为它与UNIX平台的紧密关系。大多数Web服务器都运行在UNIX或类似平台上，而Perl可以运行在大多数或所有这类系统上面。这种语言功能如此强大，以致很多这种系统上的例行管理任务都可以在上面执行，其超级模式匹配技术对于迅速扫描大量数据来说是十分有用的。

本章将集中讨论Perl，展示如何可以使用这种语言来构建动态的MySQL应用程序，以便适应网站或网站以外的应用。除了详细讨论如何使用Perl来管理数据库连接，处理查询结果和处置错误以外，本章还提供了很多示例代码和脚本，以使读者掌握基本原理，使用Perl开发自己的MySQL Web应用程序。

20.1　Perl

先简要回顾一下Perl，包括其历史与演变，并且讨论使用MySQL和Perl所需的工具。

20.1.1　历史与演变

实用摘录与报告语言（Perl）的产生是因为其创建者Larry Wall需要一种工具，要能够从原始文本数据中产生报告。因为Larry Wall不满意当时一些工具的能力，如sed和awk等，所以他创建了Perl语言作为一种代替工具，使开发者可以得到一种快速且更为灵活的文本操作的工具箱。对这种语言的最准确的定义可以见Perl的手册页：这种语言的目的是实际（易于使用、有效和完整），而不是艳丽（小巧、文雅和精致）。它组合了C，sed，awk和sh的一些最佳特性（作者的观点）。

注释： Perl手册页可以从网站http://www.perldoc.com/perl5.6/pod/perl.html得到。

Perl 1.000版于1987年在USENET comp.sources新闻组上发布。从这个初期产品开始，Perl经过多年发展，已经越来越壮大。现在，虽然Perl仍然被称做是一种完整的（且令人印象深刻的）文本操作API，但是它还具有与数据库对话、解析XML、执行SSL事务以及其他广泛的功能。这种语言的高度灵活性已经把对Perl的信任树成了一面旗帜："我们又有了一条新的途径"。使用Perl，通常可以达到这个目的。

提示： 通过网站http://www.linux-mag.com/1999-10/uncultured_01.html可以阅读到Larry Wall关于Perl演变的第一手资料。

Perl是一种解释性语言，正因如此，可以让用户执行增加的、反复的开发和测试，而不需要每次修改代码时还要通过创建或修改-编译-测试-排错循环，这样可以大大缩短开发周期。

并且Perl的编程也相对容易，特别是如果已经具备一定的C语言及类似语言经验的情况下。Perl可以访问C库，并具备了针对C编写程序代码的优势，同时该语言具有极为灵活的优势，允许程序员完成常规任务。

20.1.2 安装

对于UNIX，Windows和Macintosh平台，Perl可以通过因特网免费获得。源代码、预编译二进制文件和安装手册可以通过网站http://www.perl.com/CPAN/ports/得到。虽然读者可以自由选择使用平台，但本章中的示例假设使用Linux平台的Perl 5.8.0版本。

详细讨论Perl的安装超出了本章范围。如果读者是一名Perl的新手，可以花一些时间阅读下载得到的安装手册和网站上的说明。这些文件可以极大地帮助读者进行Perl开发和运行Perl程序。如果读者遇到了困难，网站http://www.perl.com/pub/q/FAQ上的疑难问答、http://www.perl.com/pub/a/language/info/mailing-lists.html的Perl邮件列表和comp.lang.perl.misc及comp.lang.perl.moderated的USENET新闻组都可以作为参考。

如果是在UNIX系统上，检查Perl是否已经安装在系统上面的快捷方法，是使用UNIX的which命令。如果Perl可以使用，程序将返回一个指向Perl二进制文件的完整路径。

```
[user@host]$ which perl
/usr/bin/perl
```

在Windows上，可以试着检查包含可执行Perl目录的PATH环境变量。

20.2 MySQL与Perl DBI

Perl通过其DBI.pm模块支持与数据库交互，这是针对多种不同数据库（包括MySQL）的数据库抽象层。正如其名，数据库抽象层就是一种在实际数据库访问方法之上的抽取层，它允许开发者处理不同的数据库，而不必以每个数据库为基础从根本上修改代码。

利用在数据库和开发者之间设置的抽象层的方法，DBI把程序员与数据库实现细节分隔开来。如果最初编写了一个直接进行对话的脚本，假设是针对Oracle，并且后来又需要将此用到其他数据库服务器上，通常需要重新编写所有针对数据库的部分。如果使用一个类似DBI的数据库API，就可以移植这些脚本，而不需要进行多少修改。

DBI自身不负责数据库和Perl脚本之间的交互，DBI只是简单地提供一致的界面到DBD（DBD承担实际工作）。正是这个DBD负责与数据库的对话，而不是DBI。当在DBI中发出了一个功能调用时，是把这种完整的格式后的信息传输到相关的DBD上。一旦DBD通过，将得到应答，并将应答返回给调用者。

DBI还负责保持界面整洁，处理DBD自动装载、错误检查等操作。DBI的行动好像一个中间人，负责处理所有专门针对具体数据库的事件，并且允许改变该实现，而不需要开发者对此有任何担心。

DBD存在于大多数常见数据库中，包括MySQL，PostgreSQL，Oracle等。甚至还有ODBC DBD，可以用于连接类似Access等Microsoft数据库。

在开始操作之前，需要确认已经在系统上安装好了DBI。测试其存在的简单方法是在shell提示下运行下列命令：

```
[user@host]$ perl -e "use DBI"
```

如果Perl没有出现错误且没有退出，则表明已经安装了该模块。如果模块没有安装，可以通过网站http://www.cpan.org/进行下载，并且按照使用平台的安装手册进行安装。请注意应该安装好基本DBI和MySQL DBD两个模块。安装步骤如下：

```
[root@host]$ tar -xzvf DBI-1.37.tar.gz
[root@host]$ cd DBI-1.37
[root@host]$ perl Makefile.PL
[root@host]$ make
[root@host]$ make test
[root@host]$ make install

[root@host]$ tar -xzvf DBD-mysql-2.9002.tar.gz
[root@host]$ cd DBD-mysql-2.9002
[root@host]$ perl Makefile.PL
[root@host]$ make
[root@host]$ make test
[root@host]$ make install
```

要说明DBI如何进行工作，可以开始使用一个简单的示例。参阅下列数据库，它连接着图书与其作者。

title	author
Shutter Island	Dennis Lehane
The Memoirs Of Sherlock Holmes	Arthur Conan Doyle
Hannibal	Thomas Harris
The Dilbert Principle	Scott Adams

使用下列Perl脚本连接数据库，并且打印出如下数据：

```
#!/usr/bin/perl
# load module
use DBI();

# connect
my $dbh = DBI->connect("DBI:mysql:database=somedb;host=localhost", "user", "pass");

# execute query
my $sth = $dbh->prepare("SELECT * FROM books");
$sth->execute();

# iterate through resultset
while(my $ref = $sth->fetchrow_hashref())
{
        print "Title: $ref->{'title'}\nAuthor: $ref->{'author'}\n\n";
}

# clean up
$sth->finish();
$dbh->disconnect();
```

灵活选择

在大多数情况下，只要安装了MySQL，DBI的安装过程将相当直接。主要的DBI安装脚本和MySQL DBD安装脚本足以智能地帮助你找到所需文件的位置（包括MySQL客户机库文件），实际上不需要你的交互操作。一旦的确遇到了问题，包含上述两个包的文档将极为有用，可以帮助排除安装过程中的问题。

前面脚本的输出情况参阅图20.1。

图20.1 一个简单Perl DBI脚本的输出

下面来仔细分析前面的脚本，以理解Perl连接到MySQL数据库并在其中检索数据的过程。

1. 脚本开始十分简单。也许读者已经知晓，第一行是调用Perl的解释程序，然后要求解析和运行跟着的语句。

```
use DBI();
```

这是其语句的第一部分。要求装载并激活DBI模块，使得可以使用脚本中的DBI。

2. 如果要开始与MySQL数据库服务器的通信，首先需要打开与服务器的连接。为了初始化这个连接，DBI可以提供connect()方法。

```
my $dbh = DBI->connect("DBI:mysql:database=somedb;host=localhost", "user", "pass");
```

这个函数需要5个变量：使用的DBD名（mysql）；运行MySQL服务器的主机名（localhost）；连接到的数据库名(somedb)和MySQL用户名和口令。这个函数将返回一个数据库句柄，在变量$dbh中保存。这个句柄在与数据库通信过程中被用在整个脚本上。20.2.1节中将更为详细地进行讨论。

接下来，对RaiseError的调用保证了如果DBI遇到错误，将调用die()，输出一个错误信息，表示停止的原因（示例中没有体现出来）。对于一个类似前面示例的简单的脚本，可能运行得十分完美，但是对于比较复杂的脚本来说，可能会选择关闭并使用更为高级的方式来处置错误。

prepare()方法是采用SQL查询作为参数，准备好一个查询以便运行，但是还没有开始运行。然而prepare()将通过变量$sth向准备好的查询语句返回一个句柄，该句柄然后被传递到execute()方法上，execute()执行查询。

```
my $sth = $dbh->prepare("SELECT * FROM books");
$sth->execute();
```

这种查询可以是 MySQL 能够接受的任何有效 SQL 查询。

由 prepare()方法返回的语句句柄还附带了很多有趣的方法和特性，参阅 20.2.2 节可以了解更多有关如何使用与 SELECT 和非 SELECT 查询有关联的内容。

3. 一旦查询被运行，接着的动作就是使用返回的数据做某些操作。有很多方法可以迭代通过结果集和解析到不同的字段之中。前面的示例使用语句句柄的 fetchrow_hashref()方法来收集数据，作为散列引用，并进行格式化，以便进行显示。

```
while(my $ref = $sth->fetchrow_hashref())
{
        print "Title: $ref->{'title'}\nAuthor: $ref->{'author'}\n\n";
}
```

Perl DBI 还可以提供在结果集中检索字段的一些其他方法。参阅 20.2.3 节的内容可以了解使用代替方法的更多信息。

在查询存在错误的情况下，比如错误拼写列名或关键字，MySQL 服务器将终止查询运行并返回一个错误信息。参阅 20.2.4 节将更为详细地了解错误报告的内容。

4. 一旦所有数据检索完毕，函数 finish()和 disconnect()将负责撤离数据库（清空内存且清除一般性事务）。

```
$sth->finish();
$dbh->disconnect();
```

在接着的示例中可以看出，前面简要介绍的5个步骤组成了一个基本过程，在使用Perl DBI和MySQL DBD构建数据驱动应用程序中可以反复利用。下面的章节中还将更为详细地探讨这些步骤。

DBD 格式

在 DBD 名后没有标准的字符串，这个字符串随 DBD 格式有所不同。通常需要查阅 DBD 附带的文档，以便了解数据库专门使用的格式。

20.2.1　连接管理

与 MySQL DBMS 的连接是使用 connect()方法进行处置的，可以接受三种主要参数：数据源标识符、用户名和口令。

```
my $dbh = DBI->connect("DBI:mysql:database=somedb;host=localhost", "user", "pass");
```

数据源标识符还可以进一步细分成三个部分：所使用的 DBD 名、数据库名和 MySQL 服务器主机。DBD 名被用来定位和安装驱动程序，同时与用户名和口令一起使用的数据源标识符的其他部分被传递到 DBD，以便进一步进行处理。如果无法发现或安装指定的 DBD，connect()方法将停止。

通过指定 connect()字符串中的文件名的方法，可以使用 Perl 来从文件中读取连接值。考虑下列程序片段，其情况为：

```
my $dbh = DBI->connect("DBI:mysql:database=test;mysql_read_default_file=/home/
me/access.conf");
```

一个配置文件的示例情况如下：

```
[client]
host=localhost
user=joe
pass=secret
```

如果成功的连接已经建立，connect()将返回一个数据库句柄对象，这个对象可以用在后面所有的数据库通信上。这还意味着可以使用不同的connect()语句同时打开几个数据库连接，然后在不同的变量中保存返回的对象。如果需要同时访问几个数据库，这么做就很有用处，参阅下列示例：

```
#!/usr/bin/perl

# load module
use DBI();

# connect to database 1
my $dbh1 = DBI->connect("DBI:mysql:database=fy2002;host=192.168.0.76", "root",
"pass");

# execute query
my $sth1 = $dbh1->prepare("SELECT amount FROM balancesheet WHERE label =
'Closing Balance'");
$sth1->execute();

# print value
my $ref1 = $sth1->fetchrow_hashref();
print "Closing balance in 2002: $ref1->{'amount'}\n ";

# connect to database 2
my $dbh2 = DBI->connect("DBI:mysql:database=fy2003;host=192.168.0.80", "root",
"pass");

# execute query
my $sth2 = $dbh2->prepare("SELECT amount FROM balancesheet WHERE label =
'Closing Balance'");
$sth2->execute();

my $ref2 = $sth2->fetchrow_hashref();
print "Closing balance in 2003: $ref2->{'amount'}\n ";

# clean up
$dbh1->disconnect();
$dbh2->disconnect();
```

一般情况下，与服务器的连接可以在脚本生命周期内一直保持打开，一旦脚本结束将自动终止。如果希望提前或明确终止连接（很可能的情况是，如果有一个长脚本并且一旦与数据库的交互已结束，希望减少MySQL服务器的开销），数据库句柄对象的disconnect()方法可以用来终止各个连接。

```
$dbh->disconnect();
```

如果该脚本已经开始与数据库建立事务关系，或者使用begin_work()方法或者在一个connect()调用中设置'AutoCommit' => 0，就必须在调用disconnect()之前，调用commit()或rollback()方法，这样该事务才能正确完成。下面是一个简单的示例：

```
#!/usr/bin/perl

# load module
use DBI();

# variables for the transaction
# assume these have come from standard input
$name = "Tan T";
$nat = "IN";
$dept = "Human Resources";
$pbasic = 55000;
$pbonus = 7500;

# connect
my $dbh = DBI->connect("DBI:mysql:database=master;host=192.168.0.241", "root",
"secret",
 { 'RaiseError' => 1, 'AutoCommit' => 0} );

# place the transaction in an eval{} block
# so that errors can be trapped for rollback
eval
{
        # insert employee record
        $dbh->do("INSERT INTO employees (ename, enationality) VALUES ('$name',
'$nat')");

        # get autoincrement ID of inserted record
        $id = $dbh->{ 'mysql_insertid'};

        # insert department memberships
        $dbh->do("INSERT INTO departments (eid, dept) VALUES ('$id', '$dept')");

        # insert payroll data
        $dbh->do("INSERT INTO payroll (eid, pbasic, pbonus) VALUES ('$id',
'$pbasic', '$pbonus')");

        # no errors so far? commit!
        $dbh->commit();
};

# if any errors, rollback!
if ($@)
{
        print "Transaction aborted: $@";
        $dbh->rollback();
}

# clean up
$dbh->disconnect();
```

这里的事务涉及向雇员数据库中添加一名雇员，使用向三个单独的表中输入信息的方式。第一步是使用connect()方法连接数据库，请注意向connect()添加AutoCommit参数。正如在第12章中讨论过的，这是告诉MySQL关闭自动COMMIT，将所有查询转换到必须明确要求进行保存的事务上。

一旦连接已经打开，可以使用标准的INSERT语句来插入新用户的数据到系统之中。这些语句包含在一个eval{…}异常处理块之中，这样如果存在错误，可以收集到并且逐步升级到主程序中。$@ Perl变量指示的错误（保存着由eval{…}产生的错误信息）将造成一个立即的rollback()调用。如果没有错误，事务将用commit()提交。很明显，必须使用表类型支持事务，类似InnoDB表类型，才能完成这些工作。

另一种连接数据库的方法

connect()方法的一种代替方法是connect_cached()方法，可以跟踪打开的连接数据库，如果再次要求打开一个已经打开的连接，将只返回已经存在的数据库句柄对象（而不是试图打开一个新连接）。

在使用DBI时，还应该特别注意打开数据库连接的数量。所有数据库在可以处理并发客户机连接数量方面都有一定的限制。如果并发客户机的最大数量没有明确指定，那么只在需要时打开新连接，而且要尽快关闭。这对防止发生数据库服务器负载过重，以致不能接受任何新连接而被迫关闭客户机程序等情况是有必要的。

在处理连接管理时，DBI中有很多专门函数可以使用，请参阅表20.1的列表。

表20.1 涉及连接管理的辅助函数

函数	功能
clone()	复制现有数据库连接
available_drivers()	返回可以使用的DBD列表
data_sources()	通过具体的DBD返回可以使用的数据库列表
get_info()	返回关于DBD和数据库服务器能力的信息
table_info()	返回表名和类型的信息
columns_info()	返回字段名、数据类型、长度和默认值的信息
primary_key()	返回主键字段信息
table_info_all()	返回RDBMS可以支持的数据类型信息

20.2.2 执行查询

一旦数据库连接建立，接下来的步骤就是执行查询。正如前面的示例所展示的，通常可以使用DBI的prepare()和execute()方法实现。

```
my $sth = $dbh->prepare("SELECT * FROM books");
$sth->execute();
```

prepare()方法把SQL查询作为参数，并准备好运行查询，但是还是没有运行这个查询。相反，prepare()返回一个已准备完毕的查询的句柄，该查询已经被保存起来，然后传递给execute()方法实际运行。请注意传递给prepare()的查询字符串不应该使用分号作为结尾，因为通常这时是使用mysql命令行客户机。

prepare()返回的语句句柄可以提供多种有用的方法，使之能够在实际运行查询之后从被返回的结果集中提取数据（下一节将进一步讨论）。正因为如此，prepare()和 execute()的组合操作在 SELECT 查询情况下是最为常用的方法。在非 SELECT 查询情况下，Perl DBI 手册建议使用一种代替方法——do()方法，这种方法可以同时处理准备和运行一个单独语句这样的双重任务。

考虑使用其他方法

使用 Perl 一般可以使用多种方式处理事务，这和使用 MySQL 的情况类似。经常可以发现使用 MySQL 可以提供的更高级的 SQL 操作，有很多有效的方法来完成特定的任务。prepare()和 execute()组合方法就是其中之一：如果应用程序需要在一个循环中运行基本相同的查询，可以通过使用临时表和多表 UPDATE 来更为有效地运行。例如可以在一个循环中这样运行很多遍：UPDATE t1 SET *m*=? WHERE *n*=?，也可以创建一个临时表，用一个多行的 INSERT 在 t1 中为每个行赋予一组新值，然后使用多表 UPDATE 语句更新表 t1。

参阅下列示例，它说明了在一个 UPDATE 查询中使用 do()方法的情况：

```
#!/usr/bin/perl
# load module
use DBI();

# connect
my $dbh = DBI->connect("DBI:mysql:database=somedb;host=localhost", "user", "pass");

# execute query

my $rows = $dbh->do("UPDATE books SET title='Mystic River' WHERE author='Dennis
Lehane'");

# clean up
$dbh->disconnect();
```

在类似 CREATE 和 DROP 的 SQL 语句中使用 do()方法，要比使用 prepare()，然后运行 execute()方法更有意义一些。然而如果准备多次运行一个类似的查询，使用 prepare()和 execute()更为有效。

此外，do()不能用于 SELECT 查询，因为需要一个语句句柄进行进一步的数据抽取。do()只能返回该查询影响到的一些行号（出现错误时是 undef），不能够用于这样的操作。

使用 DBI 占位符

虽然针对简单的应用程序有些言过其实，但是应该知道，prepare()在某些情况下可以提供一种实质性的性能提升。很多数据库脚本都涉及准备一个单独查询（如 INSERT），然后使用不同的值反复运行的情况。然而，完全没有差别的同一个查询很少发生。例如，假设试图向数据库中插入某些数据，INSERT 语句可以在形式上类似，但是数据值却在每次过程中都有变化。在这种情况下，prepare()和 execute()必须用 DBI 占位符联合在一起使用，这样可以减少开销并且改善应用程序的性能，参阅下面介绍。

```
#!/usr/bin/perl

# load module
use DBI();
```

```
# connect
my $dbh = DBI->connect("DBI:mysql:database=somedb;host=localhost", "user",
"pass");

# execute query
$sth = $dbh->prepare("INSERT INTO books (title, author) VALUES (?,?)");

# read data from file
open(FILE, "data.txt") or die("Cannot open file");
@lines = <FILE>;
close (FILE);

# iterate over raw data
# split into component sections and
# replace placeholders in statement
foreach $line (@lines)
{
        my ($title, $author) = split (",", $line);
        chomp ($title);
        chomp ($author);
        $sth->execute($title, $author) or die ($dbh->errstr);
}

# clean up
$sth->finish();
$dbh->disconnect();
```

prepare()语句中的问号也是占位符，其作用类似变量。这些问号表示当前查询的数据值以后应该放置的位置，数据值本身来自后续的execute()语句。一般情况下，execute()语句放在一个循环中，带有每次迭代中变化的一些变量，参阅前面的代码示例（这是从一个文本文件中进行读操作）。execute()方法使用的变量值可以称为绑定值，使用占位符联系每个绑定值的过程称为绑定。

在真实情况发生时

应该注意，前面的示例要比实际操作更有意义。在实际情况下，如果希望从一个文件向表中插入数据，MySQL的LOAD DATA INFILE语句要比前面介绍的Perl脚本更快更有效。在这个示例中仅仅使用了一个Perl脚本来明确地说明占位符如何工作。

然而如果正在检索的数据在被插入到数据库（经常是从一个遗留的数据库或应用程序中提取数据）中之前，需要使用常规表达方式进行传递，Perl脚本是很方便的。在这种情况下，循环处理一个文件的内容并使用占位符插入数据的方法是一种十分有用的技术。在第8章中可以看到有关LOAD DATA INFILE语句的更多内容。

有关查询运行的其他辅助函数参见表20.2。

表20.2 有关查询运行的辅助函数

函数	功能
prepare_cached()	高速缓存一个语句句柄，以便减少准备时间
execute_array()	使用一个元组中提供的值多次运行语句（元组是一个特殊的多维数组，其每行的列数相同）

20.2.3　结果集处理

一旦查询得到运行，接下来的操作是处理返回的数据。可以使用多种语句句柄方法来重复处理结果集，并且将其解析到不同的字段中。有关语句句柄方法参阅表 20.3。

表 20.3　有关结果集处理的语句句柄方法

函数	功能
fetchrow_array()	从结果集中返回下一个行作为数组
fetchrow_arrayref()	从结果集中返回下一个行作为数组引用
fetchrow_hashref()	从结果集中返回下一个行作为散列引用
fetchall_arrayref()	返回一个引用到包含一系列嵌套数组引用的数组之中，从结果集中每个行返回一个引用

更为常见的方法之一是 fetchrow_array()方法，可以返回不同记录字段作为数组元素。下面的示例表明通过使用 fetchrow_array()重新编写第一个基本示例，介绍如何使用这种方法。

```
#!/usr/bin/perl

# load module
use DBI();

# connect
my $dbh = DBI->connect("DBI:mysql:database=somedb;host=localhost", "user",
"pass");

# execute query
my $sth = $dbh->prepare("SELECT * FROM books");
$sth->execute();

# iterate through resultset
while(my @arr = $sth->fetchrow_array())
{
        print "Title: $arr[0]\nAuthor: $arr[1]\n\n";
}

# clean up
$sth->finish();
$dbh->disconnect();
```

如果愿意直接使用引用，也可以使用以相同方式工作的 fetchrow_arrayref()方法。

```
#!/usr/bin/perl

# load module
use DBI();

# connect
my $dbh = DBI->connect("DBI:mysql:database=somedb;host=localhost", "user",
"pass");

# execute query
my $sth = $dbh->prepare("SELECT * FROM books");
$sth->execute();
```

```
# iterate through resultset
while(my $ref = $sth->fetchrow_arrayref())
{
        print "Title: $ref->[0]\nAuthor: $ref->[1]\n\n";
}

# clean up
$sth->finish();
$dbh->disconnect();
```

以数组方式检索结果集的问题，是必须知晓该字段在结果集中的次序，这样才能正确地检索到字段数据。字段次序的任何变化（在数据库表中或在查询语句中），都应该把代码拆开，并需要在应用程序变量和结果集数组索引之间重新进行分配。

更为常见（虽然不一定是更有效的）的方法是以散列方式检索结果集，这时每个键都对应一个字段名。这样检索到的数据就可以被惟一键检索，而不是一个数字索引。

DBI可以通过fetchrow_hashref()方法支持这种操作（参阅第一个示例的说明），这时是把字段数据作为散列引用提取。参阅另一个示例：

```
#!/usr/bin/perl

# load module
use DBI();

# connect
my $dbh = DBI->connect("DBI:mysql:database=somedb;host=localhost", "user",
"pass");

# execute query
my $sth = $dbh->prepare("SELECT * FROM books");
$sth->execute();

# iterate through resultset
while(my $ref = $sth->fetchrow_hashref())
{
        print "Title: $ref->{'title'}\nAuthor: $ref->{'author'}\n\n";
}

# clean up
$sth->finish();
$dbh->disconnect();
```

不是每次操作一个单独行，还可以使用fetchall_arrayref()和fetchall_hashref()方法一次得到结果行的整个集，可以向一个单独数组或散列返回一个引用。这个数组（或散列）的每个元素其自身是在最终结果集中对一个行的引用。下列示例可以更明确地进行说明：

```
#!/usr/bin/perl

# load module
use DBI();

# connect
my $dbh = DBI->connect("DBI:mysql:database=somedb;host=localhost", "user",
"pass");
```

```
# execute query
my $sth = $dbh->prepare("SELECT * FROM books");
$sth->execute();

# get all rows in result set as array
my $ref = $sth->fetchall_arrayref();

# dereference array
# iterate over contents and dereference each
# print field values
foreach $record (@$ref)
{
        print "Title: $record->[0]\nAuthor: $record->[1]\n\n";
}

# clean up
$sth->finish();
$dbh->disconnect();
```

在这种情况下，fetchall_arrayref()方法将向包含指向子数组引用的数组返回一个引用，每个引用表示最终结果集的一个单独的行。因此所需要做的就是重复使用foreach循环处理间接引用的数组，然后进一步间接引用数组中的每个元素的引用，并将每个数组元素的值打印出来。

Perl DBI还允许通过数据库处理对象的不同的select…()方法把prepare()，execute()和fetch…()等方法合并成为一种单一的方法。表 20.4 给出了这个列表。

表 20.4　涉及结果集处理的数据库处理方法

函数	功能
selectrow_array()	从结果集中返回第一行作为数组
selectrow_arrayref()	从结果集中返回第一行作为数组引用
selectrow_hashref()	从结果集中返回第一行作为散列引用
selectall_arrayref()	返回一个引用到包含一系列嵌套数组引用的引用，每个引用针对结果集的一行
selectall_hashref()	返回一个引用到包含一系列键值对的散列，每个散列代表结果集的一行。每个对的值其本身就是对包含字段数据散列的引用
selectcol_arrayref()	返回一个引用到包含结果集每个行第一个字段值的数组

前面介绍的每种数据库处理方法直接对应着一个语句的处理方法。例如，selectrow_array()方法把prepare()，execute()和fetchrow_array()方法合并成为一种单独的常规调用，可以返回一个单行结果。请参阅下列示例的说明。

```
#!/usr/bin/perl

# load module
use DBI();

# connect
my $dbh = DBI->connect("DBI:mysql:database=somedb;host=localhost", "user",
"pass");

# execute query
# print data
@row = $dbh->selectrow_array("SELECT title, author FROM books WHERE
author='Thomas Harris'");
```

```
print "Title: $row[0]\nAuthor: $row[1]\n\n";

# clean up
$dbh->disconnect();
```

20.2.4 错误处理

默认情况下，如果在数据库会话中发生错误，Perl DBI 可以自动产生一条错误信息。考虑下面示例，对查询字符串有意制造了一个错误以便说明。

```
#!/usr/bin/perl

# load module
use DBI();

# connect to database
my $dbh = DBI->connect("DBI:mysql:database=somedb;host=localhost", "user", "pass");

# execute query
my $sth = $dbh->do("INSERT INTOO sales(id, item, amt) VALUES (918, 'Spanner', 900)");

# close connection
$sth->finish();
$dbh->disconnect();
```

当运行这个脚本时，Perl 会产生下列错误信息：

```
DBD::mysql::db do failed: You have an error in your SQL syntax near 'sales(id,
 item, amt) VALUES (918, 'Spanner', 900)' at line 1 at ./insert.pl line 10.
```

这种错误信息是由 MySQL 服务器产生的，并通过 DBD 上传给主 Perl 脚本。

将 PrintError 属性添加到 connect()方法调用并设置成为“假”就可以不再显示这些错误信息。

```
#!/usr/bin/perl

# load module
use DBI();

# connect to database
my $dbh = DBI->connect("DBI:mysql:database=db2;host=localhost", "joe", "secret",
{ 'PrintError' => 0} );

# execute query
my $sth = $dbh->do("INSERT INTOO sales(id, item, amt) VALUES (918, 'Spanner', 900);

# close connection
$sth->finish();
$dbh->disconnect();
```

在这种情况下，当调用该脚本时，错误信息将被抑制，即使该 SQL 语句不被运行。

通常，如果希望自己处理错误，可以关闭 PrintError 属性，如使用定制的错误句柄或信息。考虑下面的示例，使用 die()方法把原始错误处置添加到前面的脚本中：

```
#!/usr/bin/perl

# load module
use DBI();

# connect to database
my $dbh = DBI->connect("DBI:mysql:database=db2;host=localhost", "joe", "secret",
{ 'PrintError' => 0} ) or die ("Could not connect to database");

# execute query
my $sth = $dbh->do("INSERT INTOO sales(id, item, amt) VALUES (918, 'Spanner',
 900) or die ("Could not execute query");

# close connection
$sth->finish();
$dbh->disconnect();
```

现在可以再次试着运行这个脚本。这次在do()方法调用失败时，Perl显示一个定制错误信息。

通过特殊的err和errstr属性也可以检索到MySQL服务器产生的实际错误代码和信息。Err属性保存着服务器产生的最后错误代码，而errstr属性保存着相应的可供阅读的错误信息。考虑下列变化，可以显示下列内容：

```
#!/usr/bin/perl

# load module
use DBI();

# connect to database
my $dbh = DBI->connect("DBI:mysql:database=db2;host=localhost", "joe", "secret",
{ 'PrintError' => 0} ) or die ("Could not connect to database");

# execute query
my $sth = $dbh->do("INSERT INTOO sales(id, item, amt) VALUES (918, 'Spanner',
900) or die ("Could not execute query, server returned code [" . $dbh->err . "]
and message [" . $dbh->errstr . "]");

# close connection
$sth->finish();
$dbh->disconnect();
```

这里是实际的输出情况：

```
Could not execute query, server returned code [1064] and message [You have an
error in your SQL syntax near 'sales(id, item, amt) VALUES (918, 'Spanner',
900)' at line 1] at ./insert.pl line 10.
```

可以使用略微不同的表示法访问err和errstr变量值（因为$dbh处理对象不应该存在，除非连接成功，前面object->method的表示法不能在这种情况下使用），来检查事务连接阶段中的错误。下面的另一个示例可以说明这种情况：

```
#!/usr/bin/perl

# load module
use DBI();
```

```
# connect to database
my $dbh = DBI->connect("DBI:mysql:database=db2;host=localhost", "joe",
"badpass", { 'PrintError' -> 0}) or die ("Could not connect to database,
server returned code [" . $DBI::err . "] and message [" . $DBI::errstr . "]"
);

# execute query
my $sth = $dbh->do("INSERT INTOO sales(id, item, amt) VALUES (918, 'Spanner',
900)") or die ("Could not execute query, server returned code [" . $dbh->err
. "] and message [" . $dbh->errstr . "]");

# close connection
$sth->finish();
$dbh->disconnect();
```

特别值得注意的是，err 和 errstr 属性在大多数方法调用之前都被 DBI 自动重新设置，因此只能反应所产生的最后错误代码或信息。

跟踪程序流

DBI 还提供了强大且有用的 trace()方法，在 Perl 脚本中排除不希望发生的错误行为时十分有用。这种方法可以很详细地跟踪在 DBD 中数据库会话的不同阶段上正在发生什么。Perl 为跟踪输出的过程定义了 9 个等级，第一级输出最不详细。

```
#!/usr/bin/perl

# load module
use DBI();

# connect
my $dbh = DBI->connect("DBI:mysql:database=somedb;host=localhost", "user",
"pass");

# start debug
$dbh->trace(2);

# execute query
my $sth = $dbh->prepare("SELECT * FROM mybooks");
$sth->execute();

# iterate through resultset
while(my $ref = $sth->fetchrow_hashref())
{
        print "Title: $ref->{ 'title'}\nAuthor: $ref->{ 'author'}\n\n";
}

# clean up
$sth->finish();
$dbh->disconnect();
```

其输出情况类似图 20.2 的内容。

认真看一下图 20.2 中间部分，可以了解到脚本错误的原因：一个无效的表名。

正如前面示例所显示的，通常跟踪输出直接转储到标准输出设备上。如果不需要这么做，还可以让 Perl 保存到文件中，需要指定文件名作为 trace()方法的第二个可选变量。例如，下面的代码行将直接输出到文件 /tmp/error.log：

```
$dbh->trace(2, '/tmp/error/log');
```

图 20.2　跟踪 DBI 错误

准确地排除错误

trace()方法可以与数据库和语句句柄相联系。当调用一个数据库句柄的方法时，trace()方法可以对所有涉及数据库连接的活动提供调试错误信息。当作为语句句柄方法调用时，只能跟踪涉及相应查询的活动。如果在一个脚本中有大量的 prepare()，execute()和 do()调用，这种实现对具体查询或语句跟踪数据的能力十分方便，而且可只对其中一个调用查阅调试错误输出感兴趣。

将跟踪等级置为 0 可以关闭调试错误跟踪。

```
$dbh->trace(0);
```

在得到和处理异常方面，DBI 还可以用来指定多种其他附加属性（在散列中），作为 connect()方法的第 4 种可选变量，请参阅表 20.5 的内容。

表 20.5　错误处理属性

属性	功能
PrintError	如果错误发生打印警告
RaiseError	如果错误发生提出异常
HandleError	使用定制异常句柄处理错误
TraceLevel	设置默认跟踪等级

表 20.5 中最有用的属性之一是 RaiseError 属性，可以确保如下情况：在数据库会话中如果错误发生，Perl 将提出异常（可能因为在一个 eval()块中使用结尾代码所致）并立即运行 die()。这个属性和列表中的其他属性都可以用于定制数据库和语句句柄对象的错误处理行为。

20.2.5　辅助函数

除了前面讨论过的函数之外，DBI 还包括有很多涉及结果集处理的辅助函数。这些函数可

以检索受到查询影响的行数的信息、连接结果集字段到Perl变量、转义使用查询字符串中的特殊字符等。表20.6列出了这方面的重要函数。

表 20.6 辅助函数

函数	功能
rows()	返回受到查询影响的行数
finish()	重新设置语句句柄并清空语句缓存
bind_col()	绑定结果集中的字段到Perl变量
quote()	转义字符串中的特殊字符
dump_results()	转储整个结果集到标准输出。对于调试错误查询很有用
ping()	测试RDBMS是否正在运行
looks_like_number()	返回一个布尔值，表示变量是否类似一个数字值

20.3 真实范例

现在抛开基本原理，把这些知识运用到实际应用当中。下面的章节试图举出一些实例来，以说明如何使用Perl DBI来构建一个简单的Web应用程序，其功能是显示内容并让用户提出有关这些内容的评论。在这种情况下，读者提交的内容和评论都保存在一个MySQL数据库中，通过前面讨论的DBI调用，使用Perl处理动态产生的应用程序页面。

20.3.1 设计数据库

构建这种应用程序的第一步涉及理解需求。在这种情况下，需要设定的网站结构相当简单：网站上每篇文章都用表中的一条记录来表示，并且这个表包含着每篇文章的标题、摘要和文章的完整内容。每篇文章都由一个惟一的文章编号识别，这个文章编号可以被应用程序使用，来检索文章内容，跟踪读者有关这篇文章的评论。

网站的访问者有机会发表对每篇文章的评论，这些评论都保存在MySQL表中，其中包括评论的主题和内容、发送者的电子邮件地址和时间戳。评论可以是有线索的，即用户可以对其他人以前发表的评论回复评论，并且应用程序应该使用方便阅读的树型方式显示这种有线索的评论。

基于上述这些基本假设，所开发的应用程序应该具备下列4个简单单元：

- 列出所有可以阅读文章的索引页及其说明。读者可以使用索引来选择查阅一篇文章。
- 显示选中文章全部内容的文章详细页面，如果存在，还应该包括发表的评论。这些评论可以在一个有线索的视图中显示。读者可以选择任何评论审阅或答复，甚至可以开始一个新线索。
- 显示选中评论全部内容的评论详细页面，并且允许读者进行答复。
- 接受用户评论并且将这些评论添加到评论树上的表单。

如果需要构建这样一个应用程序，就需要按照下面的方式设计表，收集全部需要的信息：

```
CREATE TABLE articles (
  aid int(8) NOT NULL auto_increment,
  atitle varchar(255) NOT NULL default '',
  adesc varchar(255) NOT NULL default '',
```

```
  abody text NOT NULL,
  atimestamp datetime default NULL,
  PRIMARY KEY  (aid)
) TYPE=MyISAM;
```

第一个表基本可以自我解释。表中包含了所有可以查阅的文章的列表，包括每篇文章赋予的惟一的文章ID。这个表还可以保存每篇文章的标题、摘要、表示该文章是什么时候添加到系统上的时间戳以及文章的全文。这个表中的数据可以由网站内容管理器、内容管理系统或者手工方式输入。

在这个表中添加若干实例文章，可能对这个应用程序顺利进行有帮助：

```
INSERT INTO articles VALUES (4,'Getting Away With Murder','A superbly choreo-
graphed musical charged with the gritty realities of sex, scandal and the
fickleness of fame','Rather than just relying on its star-studded cast to make
it a runaway success,it is Chicago\'s irreverent look at murder and manipula-
tion of the media that makes it refreshingly different from romantic, and often
sappy, musicals. The inmates\' rendition of jailhouse jazz - lithe bodies in
Lycra and latex - in particular epitomizes the essence of this sexy and often
darkly humorous satire. All in all, the movie gives its audience more than just
the old razzle-dazzle.','2003-08-19 17:34:55');

INSERT INTO articles VALUES (5,'Towering High','The second installment of the
Tolkien epic is bigger and better than its predecessors.','If you\'ve watched
The Fellowship of The Ring, you understand this is more a film about scale than
about simple storytelling. In the second part of this three-volume tome, the
story strays from that singular epic journey of the first volume into three
separate expeditions, now with more motives than one.  Frodo and Sam are on
their way to Mordor to destroy the ring, whose ...','2003-08-19 17:36:52');

INSERT INTO articles VALUES (6,'Pottermania!','Harry\'s back!','For those of
you who are Potter fans, and have been eagerly awaiting the release of the
second installment of the Harry Potter series, this movie is definitely worth
a watch. Harry Potter And The Chamber Of Secrets is packed with great visual
effects, better acting, and is much more visually appealing than its predecessor.
It also, of course, comes with a brilliant storyline.','2003-08-19 17:40:01');
```

下一步是要创建保存读者对每篇文章的评论的表，这个表中的数据将由读者通过应用程序界面输入。将在后面介绍如何构建应用程序界面。

```
CREATE TABLE comments (
  cid int(8) NOT NULL auto_increment,
  aid int(8) NOT NULL default '0',
  cemail varchar(255) default NULL,
  ctimestamp datetime default NULL,
  csubject varchar(255) NOT NULL default '',
  cpost text NOT NULL,
  creply int(8) NOT NULL default '0',
  PRIMARY KEY  (cid)
) TYPE=MyISAM;
```

输入到这个表中的每条评论都与某篇文章的ID有关联，aid列作为对前面文章表的一个外键引用。每个评论都赋予了一个编号属性：时间戳、发表评论的人的电子邮件地址、主题行、评论内容和一个表示该评论是一个新线索的开始，还是对现有评论的答复的标志。最后一个项

日是由creply字段实现的，表示该评论在讨论树中的一个等级，这样当构造线索评论树结构时，可以更容易地将一个发布与另一个发布相关联。Creply值为0表示在树中没有发布评论，换言之，这是一个讨论树的新“枝杈”。

20.3.2 构建文章索引

当表按照要求设计完毕，这时需要开始编写一些代码。需要做的第一件事就是编写一些Perl代码显示所有可以阅读文章的列表，包括每篇文章的简单摘要、与全文的连接等。这些数据都可以在MySQL数据库中使用前面讨论的Perl DBI函数进行检索。

下面是具体脚本：

```
#!/usr/bin/perl

# read in required modules
use DBI;
use CGI;

# initialize CGI object
$cgi = new CGI();

# print HTTP headers
print $cgi->header();

# print page header
print $cgi->start_html;
print <<EOH;
<font face="Verdana, Arial" size="4" color="#66CC00"> Article archive: </
font>
<p>
<font face="Verdana, Arial" size="2" color="Black">
<ul>
EOH

# connect to database
my $dbh = DBI->connect("DBI:mysql:database=db1;host=localhost", "root", "pass",
{ 'RaiseError' => 1});

# execute query
my $sth = $dbh->prepare("SELECT aid, atitle, adesc FROM articles");
$sth->execute();

# iterate through resultset
# print article list
while($ref = $sth->fetchrow_hashref())
{
      print "<li><a href=\"article.cgi?aid=$ref->{ 'aid'}\">$ref->{ 'atitle'}
</a><br>\n";
      print "$ref->{ 'adesc'}<p>\n";
}

# close database connection
$sth->finish();
$dbh->disconnect();
```

```
# print page footer
print <<EOF;
</ul>
</font>
EOF
print $cgi->end_html;
```

工具提示

应该注意，本节介绍的脚本广泛使用了 Perl CGI.pm 模块。这种CGI.pm模块通常是Web开发者在使用Perl开发Web应用程序的时候使用的，因为模块可以提供强大且灵活的API，利用简单的方法调用就可以动态地构建网页。在这些功能中，包括了一些函数，可以发送标准的或定制的HTTP报头，创建超连接、表单、框架和其他HTML部件，把文本格式化为HTML，以及在Perl变量中自动检索POST或GET数据。

下面是本节使用的一些重要的CGI.pm方法及其分别能够起到的作用：

- header()　发送HTTP报头给客户机
- start_html()　打印HTML起始标志；
- end_html()　打印HTML关闭标志；
- url()　检索当前运行脚本的URL；
- param()　从GET查询字符串或POST数据中检索变量值。

有关CGI.pm的进一步信息，可以访问网站hnp://search.cpan. org/author/JHI/ perl-5.8.0/lib/CGI.pm。

如果需要了解Perl DBI如何工作，前面的脚本应该可以相当简单地进行诠释。该脚本打开与MySQL数据库的连接，进行查询以便得到全部可用文章的列表，并且在一个整洁的列表中进行显示。每篇文章的标题都与一个单独的脚本（article.cgi）连接，并且通过URL GET方法把文章的ID编号发送给脚本。结果集将可以检索出来，并使用前面介绍过的fetchrow_hashref()方法进行处理。其最终输出情况参阅图20.3。

图 20.3　文章索引

20.3.3 检索文章内容

正如前面提到的，索引中的每篇文章都与Perl脚本article.cgi连接，脚本把收到的文章ID作为输入参数。然后这个脚本使用文章ID来检索文章的全文，再将结果以格式整齐的网页方式显示出来。请参阅下列代码：

```
#!/usr/bin/perl

# read in required modules
use DBI;
use CGI;

# initialize CGI object
$cgi = new CGI();

# print HTTP headers
print $cgi->header();

# check to see if article ID has been sent
# if it has, clean it up and use it
if (!$cgi->param('aid'))
{
      print "Unable to find article ID, terminating...";
      die;
}
else
{
      $aid = $cgi->param('aid');
}

# print page header
print $cgi->start_html;

# connect to database
my $dbh = DBI->connect("DBI:mysql:database=dbl;host=localhost", "root", "pass",
{'RaiseError' => 1});

# execute query
my $sth = $dbh->prepare("SELECT atitle, abody FROM articles WHERE aid='$aid'");
$sth->execute();

# if result available
# print article contents
# else print error
if ($sth->rows() > 0)
{
      $ref = $sth->fetchrow_hashref();
      print "<font face=\"Verdana, Arial\" size=\"4\" color=\"#66CC00\">\n";
      print $ref->{'atitle'}, "\n";
      print "</font>\n";
      print "<p>\n";
      print "<font face=\"Verdana, Arial\" size=\"2\" color=\"Black\">\n";
      print $ref->{'abody'}, "\n";
      print "</font>\n";
}
```

```
else
{
        print "Unable to retrieve article contents, terminating...";
        die;
}

# close database connection
$sth->finish();
$dbh->disconnect();

# print page footer
# with link to comment posting form
print <<EOF;
<font face="Verdana, Arial" size="2" color="Black">
<a href="post.cgi?aid=$aid">Comment</a> on this article.
</font>
EOF
print $cgi->end_html;
```

因为这个脚本要求将一篇文章ID传给适当的函数，所以第一步就是检查是否存在这个ID。CGI.pm模块可以使用自己的param()方法来大大地简化这个任务，这个方法可以用来从传递给脚本的查询字符串或被POST处理过的数据中检索具体的参数。如果没有发现文章ID，脚本将显示错误信息，并终止操作。

假设文章ID正确地传递给脚本，Perl DBI方法被用来查询数据库，以便得到文章的全文。然后将结果以适当的格式化网页方式显示出来。在页面的底部包含一个连接，用户可以通过到post.cgi脚本的连接发布对显示文本的评论。请注意这个脚本也可以通过GET方法传递文章ID。

前面这个脚本的输出请参阅图 20.4。

图 20.4　文章内容

如果读者不知道为什么前面的页面还不显示讨论的线索评论树，这种疑问并不奇怪。这时需要把这个脚本放到 20.3.5 节中添加这个功能。

20.3.4 添加评论

接下来需要构建一个表单，要求提供一些用户基本信息，并允许用户输入对文章的评论。一旦表单提交，Perl 脚本将处理这些输入进来的信息，并通过 INSERT 操作将结果保存到数据库中。

为了方便，本例将最初的表单和结果页面都嵌入使用包含 $submit 变量的相同 Perl 脚本（post.cgi），以便用来决定显示哪些页面。请参阅下列最初的表单：

```
#!/usr/bin/perl

# read in required modules
use DBI;
use CGI;

# initialize CGI object
$cgi = new CGI();

# print HTTP headers
print $cgi->header();

# print page header
print $cgi->start_html;

# check to see if form has been submitted
if (!$cgi->param('submit'))
{
      # get URL of current script
      $url = $cgi->url();

      # check for article ID
      if (!$cgi->param('aid'))
      {
            print "Unable to find article ID, terminating...";
            die;
      }
      else
      {
            $aid = $cgi->param('aid');
      }

      # check if this is a reply or new post
      if ($cgi->param('creply'))
      {
            $creply = $cgi->param('creply');
      }

      # display form
      print <<FORM;
      <form method="POST" action="$url">
      <input type="hidden" name="creply" value="$creply">
      <input type="hidden" name="aid" value="$aid">

      <table border="0" cellspacing="5" cellpadding="0">
```

```
        <tr>
        <td colspan=2>
        <font face="Verdana, Arial" size="2" color="Black"><b>Your comment:</
b></font><p>
        </td>
        </tr>
        <tr>
        <td><font face="Verdana, Arial" size="2" color="Black">Email address</
font></td><td><input type="text" name="email" size="25"></td>
        </tr>

        <tr>
        <td><font face="Verdana, Arial" size="2" color="Black">Subject</font>
</td>
        <td><input type="text" name="subject" size="25"></td>
        </tr>
        <tr>
        <td><font face="Verdana, Arial" size="2" color="Black">Comment</font>
</td>
        <td><textarea name="post" rows="5"></textarea></td>
        </tr>
        <tr>
       <td colspan=2 align=center><input type="submit" name="submit" value="Add
Comment"></td>
        </tr>
        </table>
        </form>
FORM
}
else
{
        # form submitted, process data
}

# print page footer
print $cgi->end_html;
```

这是一个相当标准的表单，包含匹配评论表中字段的字段。使用前面提到的脚本，在实际操作完成之前，将检查传递给脚本的文章 ID。然而这里与前面提到的脚本不同，该脚本还要通过判断存在或缺少 $creply 变量的方法，来检查发布的评论在回复中是作为一个现有的线索还是新线索处理。

请注意，传递给这个表单的文章 ID 以及 $creply 变量（如果存在）要作为隐含变量传递给表单处理脚本。还应该注意使用 CGI.pm 对象的 url()方法，可以返回完整的 URL 给当前脚本。因为计划要调用相同的脚本处理表单数据，所以这种方法提供了常规且可移植的方法为表单的 ACTION 属性设置值。

图 20.5 介绍了这个表单的样式。

表单一旦提交，需要调用相同的脚本来处理数据。在这种情况下，因为存在 $submit 变量，所以在脚本开始处使用的 if 条件测试将把脚本流导入条件语句的第二个分支。这样可以校验填入表单的数据并使用 INSERT 语句将数据保存到数据库中。

图 20.5 添加用户评论

```
#!/usr/bin/perl

# read in required modules
use DBI;
use CGI;

# initialize CGI object
$cgi = new CGI();

# print HTTP headers
print $cgi->header();

# print page header
print $cgi->start_html;

# check to see if form has been submitted
if (!$cgi->param('submit'))
{
      # print form
}
else
{
      # check for article ID
      if (!$cgi->param('aid'))
      {
            print "Unable to find article ID, terminating...";
            die;
      }
      else
      {
            $aid = $cgi->param('aid');
      }
```

```
        # check if this is a reply or new post
        if (!$cgi->param('creply'))
        {
              $creply = 0;
        }
        else
        {
              $creply = $cgi->param('creply');
        }

        # check for email address
        if (!$cgi->param('email'))
        {
              $email = "anonymous";
        }
        else
        {
              $email = $cgi->param('email');
        }

        # check for subject
        if (!$cgi->param('subject'))
        {
              $subject = "No subject";
        }
        else
        {
              $subject = $cgi->param('subject');
        }

        $post = $cgi->param('post');

        # connect to database
        my $dbh = DBI->connect("DBI:mysql:database=db1;host=localhost", "root",
"pass", { 'RaiseError' => 1} );

        # insert post into table
        my $sth = $dbh->do("INSERT INTO comments (aid, csubject, cpost, cemail,
creply, ctimestamp) VALUES ('$aid', '$subject', '$post', '$email', '$creply',
NOW())");

        # close database connection
        $dbh->disconnect();

        # print confirmation message
        print <<MSG;
        <font face="Verdana, Arial" size="2" color="Black">
        Your comment has been posted. <a href="article.cgi?aid=$aid">Click</a>
to read more comments</a>.
        </font>
MSG
}

# print page footer
print $cgi->end_html;
```

这个脚本简单地检查表单变量，纠正空变量，然后将数据插入到数据库中。如果这是一个新评论（与答复相对），根据本节前面部分中讨论的规则，$creply 变量被设置为0。这将在确认页面中成功插入数据结果。

20.3.5 构建线索评论索引

现在已经有了一种向系统中添加评论的方法，那么如何进行输出？根据前面章节讨论过的机制，把评论连接到具体文章上（通过两个表中的aid列）以及其他信息上面（通过comments表中creply字段，可以用来分辨具体评论的父线索）。现在需要的就是有一种方法把这种逻辑关系转换为可视效果。

准确地说，这就是build_tree()函数需要起作用的地方。这是一种具有递归效果的函数，专门设计用来针对重复出现一组具有父子节点效果的任务，并在其间建立关系。这个函数需要把文章ID和开始评论ID作为输入参数，然后使用这些信息创建针对具体文章提交评论的结构树。请参阅以下示例：

```
# this is a recursive function
# to build the comment tree
sub build_tree()
{
      # get starting comment ID and article ID
      # as input parameters
      $cid = shift;
      $aid = shift;

      # connect to database
      my $dbh = DBI->connect("DBI:mysql:database=db1;host=localhost", "root",
"pass", { 'RaiseError' => 1});

      # execute query
      my $sth = $dbh->prepare("SELECT cid, csubject FROM comments where aid
= '$aid' AND creply = '$cid'");
      $sth->execute();

      # if result available
      # print comment subject line
      # else print message stating no comments
      if ($sth->rows() <= 0 && $cid == 0)
      {
            print "<font face=Verdana size=2 color=Black><b>No comments avail-
able</b></font>";
      }
      else
      {
            # iterate over result set
            while($ref = $sth->fetchrow_hashref())
            {
                  # print each comment as list item
                  # recurse to build hierarchical tree
                  print "<ul>";
                  print "<li><a href=\"details.cgi?cid=$ref->{'cid'}
&aid=$aid\"><font face=Verdana size=2 color=Black><b>$ref->{'csubject'}</b>
</a> </li>\n";
```

```
                    build_tree($ref->{'cid'}, $aid);
                    print "</ul>";
                }
            }
    }
```

这个函数首先使用前面介绍的connect()方法打开与数据库的连接。这时还需要使用一些变量：开始构建树结构时所处的等级和文章的标识符（作为参数提交）。

如果所有信息到位，build_tree()将执行一个 SELECT 查询，并得到一个等级为0的提交列表，然后将结果打印在一个列表中。对于返回的每条记录，build_tree()将递归地调用其记录编号，再次连接到数据库，并显示所有等级 1 的提交列表。这个过程一直持续到结构树的末端。HTML <ul>列表构建可以用来保证不同等级都能够按照次序进行排列。如果在等级0处没有发现评论，将显示一条类似信息。

最终显示的列表中每个项目都连接到 details.cgi 页面上，这个页面可以显示评论的完整内容。build_tree()函数本身是在前面讨论过的 article.cgi 脚本中调用，在主要文章显示下面，有一条水平线条隔开。下面是包括这个新函数的 article.cgi 新脚本。

```
#!/usr/bin/perl

# read in required modules
use DBI;
use CGI;

# initialize CGI object
$cgi = new CGI();

# print HTTP headers
print $cgi->header();

# check to see if article ID has been sent
if (!$cgi->param('aid'))
{
      print "Unable to find article ID, terminating...";
      die;
}
else
{
      $aid = $cgi->param('aid');
}

# print page header
print $cgi->start_html;

# connect to database
my $dbh = DBI->connect("DBI:mysql:database=db1;host=localhost", "root", "pass",
{'RaiseError' => 1});

# execute query
my $sth = $dbh->prepare("SELECT atitle, abody FROM articles WHERE aid='$aid'");
$sth->execute();

# if result available
```

```
# print article contents
# else print error
if ($sth->rows() > 0)
{
      $ref = $sth->fetchrow_hashref();
      print "<font face=\"Verdana, Arial\" size=\"4\" color=\"#66CC00\">\n";
      print $ref->{'atitle'}, "\n";
      print "</font>\n";
      print "<p>\n";
      print "<font face=\"Verdana, Arial\" size=\"2\" color=\"Black\">\n";
      print $ref->{'abody'}, "\n";
      print "</font>\n";
}
else
{
      print "Unable to retrieve article contents, terminating...";
      die;
}

# close database connection
$sth->finish();
$dbh->disconnect();

# display comment tree
print "<hr>\n";
build_tree(0, $aid);
print "<hr>\n";

# print page footer
# with link to comment posting form
print <<EOF;
<font face="Verdana, Arial" size="2" color="Black">
<a href="post.cgi?aid=$aid">Comment</a> on this article.
</font>
EOF
print $cgi->end_html;

# this is a recursive function
# to build the comment tree
# refer to previous listing for code
sub build_tree()
{
      # function code here
}
```

发布或取消

可以想像出来，这个过程可以进行一定测试，试验数据库服务器的能力，特别是如果有很多等级存在。为了简便，前面的脚本没有包括任何特殊增强优化，以便降低服务器的负荷。然而，在生产环境下，在应用程序的这个问题上应该予以一定考虑。

例如，在这种情况下，可以有选择地考虑改善性能，可以以静态的HTML文件方式发布整个评论树（每当有新评论提交时，都可以进行更新），并且在应用程序的所有页面上都包含它，而不是每次页面请求时都查询数据库的整个评论树。这样可以大大降低连接和查询MySQL服务器的次数，而且不会明显影响应用程序的功能。

图 20.6 显示了这个脚本修改后的输出情况，包括了一个完整的线索评论树。

图 20.6　包括线索评论树的文章内容

20.3.6　查阅评论

这个脚本的最后部分是details.cgi，需要使用提交在URL上的值（文章ID和评论ID），以便查询数据库和返回一个包含作者名、提交评论时间以及评论的全文的单独页面（参阅图 20.7）。

图 20.7　查阅用户评论

```
#!/usr/bin/perl

# read in required modules
use DBI;
use CGI;

# initialize CGI object
$cgi = new CGI();

# print HTTP headers
print $cgi->header();
```

```
# print page header
print $cgi->start_html;

# check for article ID
if (!$cgi->param('aid'))
{
      print "Unable to find article ID, terminating...";
      die;
}
else
{
      $aid = $cgi->param('aid');
}

# check for comment ID
if (!$cgi->param('cid'))
{
      print "Unable to find comment ID, terminating...";
      die;
}
else
{
      $cid = $cgi->param('cid');
}

# connect to database
my $dbh = DBI->connect("DBI:mysql:database=db1;host=localhost", "root", "pass",
{ 'RaiseError' => 1} );

# get article title and description
my $sth = $dbh->prepare("SELECT atitle, adesc FROM articles WHERE aid =
'$aid'");
$sth->execute();

# if result available
# print article details
# else print error
if ($sth->rows() > 0)
{
        $ref = $sth->fetchrow_hashref();
        print "<font face=\"Verdana, Arial\" size=\"4\" color=\"#66CC00\">
\n";
        print $ref->{ 'atitle'} , "\n";
        print "</font>\n";
        print "<p>\n";
        print "<font face=\"Verdana, Arial\" size=\"2\" color=\"Black\">\n";
        print $ref->{ 'adesc'} , "\n";
        print "</font>\n";
}
else
{
        print "Unable to retrieve article contents, terminating...";
        die;
}

# clear statement handle
```

```
$sth->finish();

print "<hr>\n";

# now get comment details
# execute query
my $sth = $dbh->prepare("SELECT cemail, ctimestamp, csubject, cpost FROM
comments WHERE cid = '$cid'");
$sth->execute();

# print comment details
if ($sth->rows() > 0)
{
        $ref = $sth->fetchrow_hashref();
        print "<font face=\"Verdana, Arial\" size=\"2\" color=\"Black\">\n";
        print "Posted by <b>$ref->{'cemail'}</b> on $ref->{'ctimestamp'}
<br>\n";
        print "<b>Subject:</b> $ref->{'csubject'} <br>\n";
        print "<b>Comment:</b> $ref->{'cpost'} <br>\n";
        print "</font>\n";
        print "<p>\n";
}
else
{
        print "Unable to retrieve comment, terminating...";
        die;
}
print "<hr>\n";

# close database connection
$sth->finish();
$dbh->disconnect();

# print page footer
# with links to reply and view other comments
print <<EOF;
<font face="Verdana, Arial" size="2" color="Black">
<a href="post.cgi?aid=$aid&creply=$cid">Reply</a> to this post or <a
href="article.cgi?aid=$aid">read more comments</a>
</font>
EOF
print $cgi->end_html;
```

请特别注意提交评论底部的连接。如果读者希望对这个具体评论予以答复，$creply变量保存着需要答复文章的评论ID。然后该$creply变量作为额外参数提交给post.cgi，并保存在数据库中，这样完成了产生讨论树的一个循环。

进一步讨论

如果还希望了解更多有关 Perl 和 Perl DBI 的知识，可以访问下列在线资源：

- Perl 官方网站 http://www.perl.com
- CPAN 网站 http://www.cpan.com
- Perldoc 网站 http://www.perldoc.com

- DBI.pm 手册网页 http://search.cpan.org/author/TIMB/DBI-1.37/DBI.pm
- Melonfire的Perl部分的网站，包含有关文章、应用实例和教程内容，位于http://www.melonfire.com/community/columns/trog/archives.php?category=Perl
- Devshed 的 Perl 部分的网站，位于 http://www.devshed.com/Server_Side/ Perl/

20.4 小结

本章讨论了 Perl DBI 的内容，包括对 MySQL DBD 和 DBI 方法的具体引用，这样可以通过 Perl 脚本实现与 MySQL 数据库的交互操作。DBI 包括很多函数，可以处理连接管理、查询运行、结果集处理和错误处置等。本章讨论了各种类型中的一些重要方法，同时包括如何使用的示例和说明。

除了基本知识以外，本章还提供了一个组合了单独线索评论系统的 Perl 与 MySQL 的实际应用程序，可以用于内容丰富的网站。通过运行 Web 服务器和使用 CGI 作为网关接口，这个应用程序提供给用户一个界面，可以查阅、添加和答复网站上的有关评论。在后台，Perl DBI 模块处理着连接数据库的任务，检索和更新表数据，格式化结果以便显示。

第21章　MySQL与PHP

Web开发者从不缺少的就是选择。开放源代码语言和工具的广泛发展，意味着完成某些任务存在着很多种可选方式。所以虽然Perl爱好者可能使用前面章节中介绍过的Perl DBI来构建MySQL支持的应用程序，但是作为初学的开发者可能习惯超文本预处理程序（PHP）的句法和风格。PHP也是一种编程语言，在世界上数以千万计的设备上使用，正很快变成一种事实上的Web应用程序标准。

也许你还不知道，PHP是一种脚本语言，可以在HTML Web页面中使用嵌套命令工作，因此可以在一些静态的网站上添加十分复杂的业务逻辑。这种语言正快速成为数据驱动的Web应用程序的一种流行选择，因为可以广泛支持不同数据库系统，显然也包括MySQL在内。

在开始介绍本章时，将讨论与PHP一道提供的MySQL API，并且提供简要课程，介绍如何使用这种语言来构建功能强大的数据驱动的Web应用程序。除了详细讨论如何使用PHP MySQL API管理数据库连接、处理查询结果和处置错误以外，本章还包括了足够的示例代码和脚本，使读者可以掌握基本原理并能够使用PHP开发自己的MySQL支持的Web应用程序。

21.1　历史和演变

PHP最先是由Rasmus Lerdorf开发的。当1995年中期PHP/FI发布时，它是作为监视页面查阅自己的在线简历的工具，慢慢地才引起关注。PHP的这个版本可以支持一些基本的Web功能，主要是可以处理表单数据和支持mSQL数据库。

随着PHP的不断普及，语言的开发已经从Lerdorf转到一个专业程序员小组，该小组负责彻底重新编写PHP解析程序。其努力的结果是PHP 3.0，它可以支持更为广泛的数据库，包括MySQL和Oracle。PHP 4.0在2003年发布，它使用了一个新的脚本引擎，可以产生更好的性能，更加可靠和具有伸缩性，可以支持除Apache外的Web服务器，并且还具有一些新的语言特性，包括内置的会话管理和更好的OOP支持。

现在PHP被数以千万计的网站使用，根据Netcraft在网站http://www.netcraft.com上的调查，这个数量还在快速上升。PHP得到世界范围的成千上万开发者和爱好者的支持，并且经常更新新模块和新功能，保证一直能够站在Web技术的最前沿。这种语言使用简便、功能强、灵活且可靠，具有可以使得人们热衷用来进行Web开发的所有理由。

21.1.1　安装

PHP可以在*NIX和Windows两个平台上使用，通过其官方网站http://www.php.net可以得到源代码和二进制两种格式。*NIX用户可以选择直接在自己的Web服务器上编译PHP，或者根据需要激活这个语言，作为一种可插入的Web服务器模块使用。Windows的用户没有很多选择，他们可以得到一个预构建的二进制文件，需要进行配置，以便在自己的Web服务器上工作。

详细介绍PHP的安装已经超出了本章的范围。如果读者是PHP的新手，可以花一些时间在PHP网站上阅读一些很好的文档，包括在PHP发布中附带的INSTALL和README文件中的说明。这样应该保证在进入PHP开发环境和运行PHP时遇到困难更少。一旦无法使用PHP与Web服务器对话，可以通过登录网站http://www.php.net/mailing-lists.php，根据那些众多PHP的邮寄列表，寻求PHP用户的帮助。

如果你正在一个共享主机环境中工作，并且不知道Web服务器是否支持PHP，或者不知道是否自己构建的PHP包括对MySQL的支持（PHP 4.0是默认支持的），其解决方法很简单。可以在自己的Web服务器文档根目录下面简单地创建一个使用.php作为文件扩展名的文件，比如info.php，然后填入下列代码：

```
<?php
phpinfo();
?>
```

现在可以在自己的网页浏览器上面浏览一下这个脚本，将可以看到类似图21.1的页面。

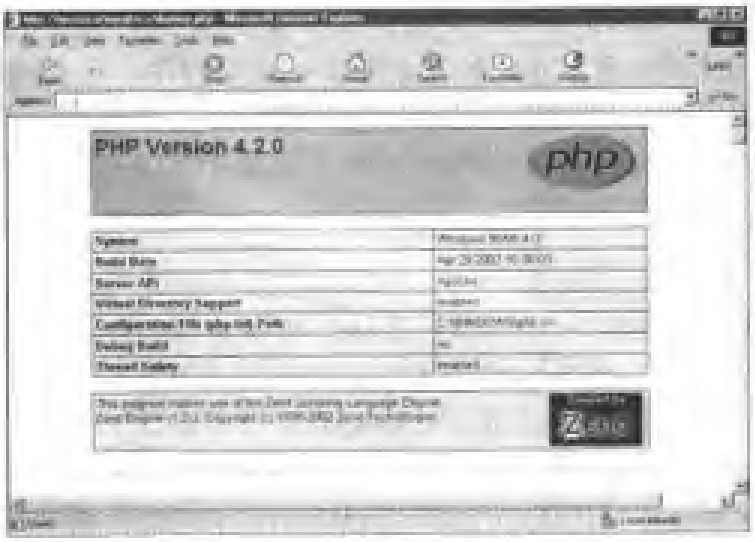

图21.1 phpinfo()命令的输出

信息过多

假如phpinfo()的输出对于你来说有些过细，还可以使用下列测试，返回一行的状态信息，指出构建的PHP是否可以支持MySQL：

```
<?php
if (extension_loaded('mysql'))
{
    echo("MySQL support present");
}
else
{
    echo("MySQL support absent");
}
?>
```

在这个页面上可以审核信息，看构建的PHP当前支持哪些功能。MySQL支持在PHP 4.0以上的所有版本中都是默认激活的。

请注意，如果自己的Web服务器没有配置可以解析带.php扩展名的文件，或者如果没有包含对PHP的支持，则前面的脚本的输出很可能是源代码。在这种情况下，应该咨询一下Web服务器文档或者系统管理员，了解有关如何配置服务器可以支持PHP的信息。

更多有关phpinfo()测试的信息可以在PHP文档以及安装说明中得到。

版本控制

请注意，本章中的示例是基于Linux和Windows 95，使用PHP 4.2.0和Apache 1.3.29创建和测试的。

21.2　MySQL与PHP的组合

从版本3.x开始，PHP已经包括了对MySQL的支持，而更早的版本需要使用特殊的配置时间变量编译PHP，才能激活MySQL模块。从版本4.x开始，PHP默认附带可以使用MySQL，其原因是PHP与MySQL组合的流行和普及。

正如第18章讨论过的，与PHP同时使用的MySQL函数具有以下4项主要目标：

1. 管理数据库连接
2. 运行查询
3. 处理查询结果
4. 提供调试和诊断信息

为了说明这些功能，考虑一个简单的示例，使用PHP连接数据库，检索一组结果，然后进行格式化以便在网页上显示。这里示例所使用的数据库包括一个简单的表，名为employees，其中存放用户登录ID及其相应的全名列表。下面是创建和初始化这个表需要的SQL查询。

```
CREATE TABLE employees (
   uname varchar(25) NOT NULL,
   fname varchar(100) NOT NULL,
   PRIMARY KEY (uname)
);

INSERT INTO employees (uname, fname) VALUES ( 'john', 'John Doe');
INSERT INTO employees (uname, fname) VALUES ( 'sarah', 'Sarah Jane');
INSERT INTO employees (uname, fname) VALUES ( 'tim', 'Tim Shaw');
INSERT INTO employees (uname, fname) VALUES ( 'jane', 'Jane Smith');
```

现在如果需要使用mysql客户机程序来连接数据库，并且查阅前表的内容，就需要使用下列方式：

```
mysql> SELECT * FROM employees;
+-------+------------+
| uname | fname      |
+-------+------------+
| john  | John Doe   |
| sarah | Sarah Jane |
| tim   | Tim Shaw   |
| jane  | Jane Smith |
+-------+------------+
4 rows in set (0.00 sec)
```

现在可以使用PHP产生相同的效果，在数据库上调用一个SELECT查询，然后在HTML页面上显示其结果：

```
<html>
<head>
<basefont face="Arial">
</head>

<body>

<?php
// set server access variables
$host = "localhost";
$user = "joe";
$pass = "secret";
$db = "db123";

// open connection to database
$connection = mysql_connect($host, $user, $pass) or die ("Unable to connect!");

// select database to use
mysql_select_db($db) or die ("Unable to select database!");

// create SQL query string
$query = "SELECT * FROM employees";

// execute query and obtain result set
$result = mysql_query($query) or die ("Error in query: $query. " . mysql_error
());

// are there any rows in the result?
if (mysql_num_rows($result) > 0)
{
     // yes
     // iterate through result set
     // format query results as table
     echo "<table cellpadding=10 border=1>";
     while($row = mysql_fetch_row($result))
     {
          echo "<tr>";
          echo "<td>" . $row[0] . "</td>";
          echo "<td>" . $row[1] . "</td>";
          echo "</tr>";
     }
     echo "</table>";
}
else
{
     // no
     // print status message
     echo "No rows found!";
}

// free result set memory
mysql_free_result($result);
```

```
// close connection
mysql_close($connection);

?>

</body>
</html>
```

其结果可以参阅图 21.2。

图 21.2　使用 PHP 检索 MySQL 数据

正如所见到的，使用PHP从数据库中得到数据涉及几个步骤，每个步骤都是预定义的PHP函数调用。下面来分析每个步骤。

需要做的第一件事是确定建立与数据库服务器连接所需的重要信息。这个信息包括服务器名、用户名和能够进行登录需要的口令，以及查询的数据库名等。这些值全部在常规PHP变量中进行设定。

```
// set server access variables
$host = "localhost";
$user = "joe";
$pass = "secret";
$db = "db123";
```

提前考虑

这些数据库登录参数将在需要MySQL服务器的应用程序的几乎所有PHP脚本中使用。常见的用法是提取这些数据到单独配置文件中，并且通过include()包含这个文件，不论PHP脚本中是否需要。不仅需要向该应用程序中添加具有可重复使用能力的元素，而且还需要减少数据库的用户名或口令变化产生的影响。对于任何改变，需要做的就是更新一个单独的配置文件，而不是编辑原始程序中的每个脚本。

如果需要开始与MySQL数据库服务器进行通信，首先需要打开与服务器的连接。PHP和数据库服务器之间的所有通信都是通过这个连接产生的。

如果需要初始化这个连接，PHP提供mysql_connect()函数。

```
$connection = mysql_connect($host, $user, $pass) or die ("Unable to connect!");
```

该函数需要三个参数：运行 MySQL 服务器的主机名、MySQL 的用户名和 MySQL 口令。如果数据库服务器和 Web 服务器都运行在同一个物理设备上，服务器名通常为 localhost。

然后这个函数将返回一个连接标识符，保存在变量 $connection 中。在与数据库通信时，这个标识符将用在整个脚本之中，在 21.2.1 节中将进一步详细讨论这个问题。

现在已经连接到数据库，接下来是选择使用的数据库。PHP 允许通过 mysql_select_db()函数实现这个步骤。

```
mysql_select_db($db) or die ("Unable to select database!");
```

mysql_select_db()函数接受数据库名作为参数，并且准备好已经选择使用的数据库。

使用被选中的数据库，这时需要通过 mysql_query()函数发送查询。

```
$query = "SELECT * FROM employees";
$result = mysql_query($query) or die ("Error in query: $query. " . mysql_error
());
```

防 护 墙

在 MySQL 函数前面使用 PHP 的“@”错误压缩操作符作为前缀调用，来隐含 PHP 的警告和错误信息是个好主意。这种方法可以隐藏公司数据库系统中的一些敏感信息。

标 点 问 题

与 mysql 命令提示的情况类似，通过 mysql_query()发送给服务器的 SQL 查询字符串应该不使用分号作为结尾。这对于新接触 PHP 的开发者来说是一个十分常见错误。

这个函数还需要一个单独的参数：需要运行的 SQL 查询。这个查询可能是任何 MySQL 服务器能够接受的有效 SQL 查询。

前面函数返回的结果集将保存在变量 $result 中。这个结果集可能包含一行或多行，一列或多列数据，取决于查询情况。在查询出现错误的情况下，比如列名或关键字拼写错误，MySQL 服务器将终止查询运行，并返回错误信息。这个信息能够通过 mysql_error()函数进行查阅。21.2.4 节可以更详细地了解错误报告的内容。

一旦得到结果集，PHP 将提供各种函数，以便处理这个结果集。这个示例展示了常用函数中的两个函数，即 mysql_num_rows()函数，可以返回结果集中的总行数；mysql_fetch_row()函数，可以用来从结果集中获得具体的行。

```
if (mysql_num_rows($result) > 0)
{
     echo "<table cellpadding=10 border=1>";
     while($row = mysql_fetch_row($result))
     {
          echo "<tr>";
          echo "<td>" . $row[0] . "</td>";
          echo "<td>" . $row[1] . "</td>";
          echo "</tr>";
     }
     echo "</table>";
}
```

因为函数mysql_fetch_row()将检索一个行作为单独的、数字化索引的值数组，所以最有效的处理方法就是使用循环重复进行收集。这样可以准确地知道这个示例做了什么，示例中把函数mysql_fetch_row()与一个while循环组合在一起，然后继续重复处理结果集中的行，在HTML表中打印每个列的内容。

不同处理方法

针对函数mysql_fetch_row()还有很多其他种类的代替方法，可以参阅21.2.3节，以便更详细地了解这方面的内容。

一旦所有行已经处理完毕，应该清空使用的内存，一般是通过调用mysql_free_result()和mysql_close()进行清空操作的。前者是清空被结果集占用的内存，而保留数据库连接，准备进一步的操作。后者是关闭数据库连接同时清空被连接标识符和结果集（如果忘记使用mysql_free_result()）使用的内存。

```
mysql_free_result($result);
mysql_close($connection);
```

正如在后面的示例所看到的，前面介绍的6个步骤构成基本过程，当使用PHP和MySQL构建数据驱动Web应用程序时，可以反复使用这个过程。

21.2.1　连接管理

与MySQL DBMS的连接是通过mysql_connect()函数进行处理的，它需要三个参数：运行MySQL服务器的主机名、MySQL用户名和MySQL口令。

```
$connection = mysql_connect($host, $user, $pass) or die ("Unable to connect!");
```

使用PEAR

读者也许注意到PHP的MySQL函数全部是使用mysql_*作为前缀，如mysql_num_rows()，mysql_fetch_row()等。认真阅读PHP手册可以发现这种原则也用在其他RDBMS上面，如Oracle专用函数就使用OCI*作为前缀，而PostgreSQL专用函数则使用pg_*打头。

十分明显，这里遇到一个重要问题：可移植性。如果数据库系统改变，就必须重新编码脚本，要使用新的RDBMS可以工作的PHP函数。这样增加了时间和工作量，同时还有可能导致实质性费用增加。

PHP试图通过数据库的抽象层的方式来解决这个问题，可以对所有可以支持的RDBMS提供统一的界面。这种数据库的抽象层通常作为免费的PEAR发布的一部分提交，可以在后端转换数据库系统，使用一种类似Perl DBI使用的方法最低限度地影响其代码。如果需要考虑应用程序的可移植性问题，就应该花一些时间来开发这个数据库抽象层，以确保后面不会遇到可移植性问题。即使不必考虑可移植性问题，类似PEAR的抽象层也是一种很好的标准化处置错误的方法，可以测量和记录查询性能，或者添加负载平衡逻辑。

有关PEAR数据库抽象层的进一步信息，可以查阅网站http://pear.php.net/。

如果成功地建立了连接，这个函数就将返回一个连接标识符。这个标识符将用于后来所有与数据库的通信。在连接失败的情况下，如因为口令错误等，其函数将返回“假”。这意味着在准备继续操作之前最好应该检查一下脚本中这个函数的返回值。下面的程序片段说明了进行这种检查的方法：

```
// attempt a connection
$connection = mysql_connect($host, $user, $pass);

// if connection failed
// display error and exit
if (!$connection)
{
     echo "Unable to connect";
     exit;
}
```

下列替代方法使用更少的几行代码也可以完成相同的任务：

```
// attempt connection or die
$connection = mysql_connect($host, $user, $pass) or die ("Unable to connect!");
```

通常与服务器的连接在脚本生命周期内一直保持打开，一旦脚本结束将自动关闭（有关这方面的忠告请参阅本小节末尾的补充资料。

有可能改变这种默认属性并且提前关闭连接。这里的示例是如果有一个很长的脚本，并希望减少在 MySQL 服务器上的开销，一旦与数据库的交互操作通过对函数 mysql_close()的调用完成，将可能立即终止指定的或最后一个打开的连接。

```
// terminate the last opened connection
mysql_close();

// terminate the connection identified by $conn
mysql_close($conn);
```

持续连接

通常一旦创建连接的 PHP 脚本停止运行，就会自动终止与 MySQL 服务器的连接。如果存在大量打开和关闭同一个数据库服务器连接的脚本，其默认特性可以导致明显的性能降低。在这种情况下，可以要求 PHP 使用 mysql_pconnect()调用替换 mysql_connect()调用的方法，就可以不必在脚本结束时终止与 MySQL 的连接，这样就打开了与服务器的“持续连接”。

这么做有什么好处？当与服务器的持续连接打开时，如果使用连接的脚本结束，这种连接也不会自动关闭。因为其连接保持活动（或者持续），所以其他连接同一个服务器的脚本也可以立即使用服务器，而不必打开新的连接。这么做可以比创建与 MySQL 服务器连接更加有效，可以减少开销，并能够在高流量服务器上面产生十分明显的性能改善。

21.2.2 查询运行

在 mysql_connect()和 mysql_close()两函数之间的代码是要去选择数据库、运行查询、处理结果和显示输出。

选择数据库的 PHP 函数是 mysql_select_db()函数，可以接受一个数据库名作为变量，还可以接受一个连接标识符。

如果没有指定，则函数默认使用最后打开的连接。下面的示例介绍如何使用这种方法：

```
// select the database named "db712"
mysql_select_db("db712"),
```

一旦选择了数据库，就可以在上面运行查询。运行查询的PHP函数是mysql_query()，它可以接受一个单独的参数，即需要运行的SQL语句。下面是一个简单的示例：

```
// execute query and obtain result set
$result = mysql_query("SELECT * FROM employees");
```

前面的示例使用一个SELECT查询，这与使用函数mysql_query()进行INSERT，UPDATE或DELETE查询一样简单。下面是这种程序的一个片段：

```
// execute query and obtain result set
mysql_query("DELETE FROM employees WHERE user = 'john'");
```

函数mysql_query()将返回结果代码，可以用来检查查询是否运行成功。

- 如果查询是一个SELECT查询或任何其他可以返回数据的查询，mysql_query()将返回一个资源标识符，包含查询结果或失败的“假”。于是这个资源标识符可以使用各种变量进行处理，详见21.2.3节。
- 对于没有返回任何数据的查询，如INSERT，UPDATE或DELETE查询，mysql_query()将返回成功或失败的信息。

在继续进行脚本运行之前，检查脚本中函数的返回值的做法很好。接下来的示例说明如何进行检查。

```
<?php

// set server access variables
$host = "localhost";
$user = "joe";
$pass = "secret";
$db = "db123";

// open connection to database
$connection = mysql_connect($host, $user, $pass) or die ("Unable to connect!");

// select database to use
mysql_select_db($db) or die ("Unable to select database!");

// create a bad SQL query string
$query = "SELECT FROM employees";

// execute query and obtain result set
$result = mysql_query($query);

if (!$result)
{
     // because the previous query contains a syntax error
     // mysql_query() will return false and
     // this branch of the conditional statement will be executed

     echo "Query failed!";
     // exit
}
```

```
else
{
     echo "Query succeeded";
     // process results
}

// close connection
mysql_close($connection);

?>
```

特殊字符的处理

在插入一些特殊字符到MySQL表中之前，应该用函数mysql_escape_string()转义这些字符。

一旦脚本打开了一个以上的数据库连接，应该知道mysql_query()将默认发送查询给最后打开的连接。如果这个默认属性不是所需要的，可以使用指定相应连接标识符的方法作为mysql_query()调用中的第二个参数，以便明确告诉mysql_query()应该使用哪个连接。下面的示例进行了演示：

```
<?php

// open two connections to databases
$conn1 = mysql_connect("my.host", "john", "secret");
$conn2 = mysql_connect("your.host", "joe", "hidden");

// select database to use
mysql_select_db("db123", $conn1);
mysql_select_db("db746", $conn2);

// create SQL query string
$query = "SELECT * FROM employees";

// execute query on $conn1 and obtain result set
$result1 = mysql_query($query, $conn1);

// check to see if query succeeded or not
if (!$result1)
{
     echo "Query on conn1 failed!";
     // exit
}
else
{
     echo "Query on conn1 succeeded";
     // process results
}

// now do the same thing for $conn2
// execute query on $conn2 and obtain result set
$result2 = mysql_query($query, $conn2);

// check to see if query succeeded or not
if (!$result2)
{
```

```
        echo "Query on conn2 failed!";
        // exit
    }
    else
    {
        echo "Query on conn2 succeeded";
        // process results
    }

    // close connections
    mysql_close($conn1);
    mysql_close($conn2);

    ?>
```

21.2.3 结果集处理

一旦结果集已经被mysql_query()返回，接下来的步骤是进行处理。在前面的示例中也许已经看到函数mysql_fetch_row()如何与一个while循环联合使用，作为值的枚举数组检索每个行，并使用编号数组索引进行显示。处理这种问题可以有不止一种方法，下面的示例可以看出，PHP提供了很多处理结果集的替代技术。

接下来的示例展示了原来mysql_fetch_row()示例的一种变形，说明如何使用PHP的函数mysql_fetch_row()和函数list()得到一个简单的值数组，然后赋予这些值到PHP变量上，这些变量可以使用在脚本的后面部分中，以避免使用数字化的数组索引。

```
<html>
<head>
<basefont face="Arial">
</head>

<body>

<?php

// set server access variables
$host = "localhost";
$user = "joe";
$pass = "secret";
$db = "db123";

// open connection to database
$connection = mysql_connect($host, $user, $pass) or die ("Unable to connect!");

// select database to use
mysql_select_db($db) or die ("Unable to select database!");

// create SQL query string
$query = "SELECT * FROM employees";

// execute query and obtain result set
$result = mysql_query($query) or die ("Error in query: $query. " . mysql_error());

// are there any rows in the result?
```

```
if (mysql_num_rows($result) > 0)
{
     // yes
     // iterate through result set
     // assign fields to variables with list()
     // format query results as table
     echo "<table cellpadding=10 border=1>";
     while(list($uname, $fname) = mysql_fetch_row($result))
     {
          echo "<tr>";
          echo "<td>" . $uname . "</td>";
          echo "<td>" . $fname . "</td>";
          echo "</tr>";
     }
     echo "</table>";
}
else
{
     // no
     // print status message
     echo "No rows found!";
}

// close connection
mysql_close($connection);

?>

</body>
</html>
```

作为一种替代，可以使用mysql_fetch_assoc()函数，以便把每个行作为包含相对列名的键和相对这些列值的值的联合数组进行检索。下面的示例展示了其具体效果：

```
<html>
<head>
<basefont face="Arial">
</head>

<body>

<?php

// set server access variables
$host = "localhost";
$user = "joe";
$pass = "secret";
$db = "db123";

// open connection to database
$connection = mysql_connect($host, $user, $pass) or die ("Unable to connect!");

// select database to use
mysql_select_db($db) or die ("Unable to select database!");
```

```
// create SQL query string
$query = "SELECT * FROM employees";

// execute query and obtain result set
$result = mysql_query($query) or die ("Error in query: $query. " . mysql_error());

// are there any rows in the result?
if (mysql_num_rows($result) > 0)
{
     // yes
     // iterate through result set
     // format query results as table
     echo "<table cellpadding=10 border=1>";
     while($row = mysql_fetch_assoc($result))
     {
          echo "<tr>";
          echo "<td>" . $row['uname'] . "</td>";
          echo "<td>" . $row['fname'] . "</td>";
          echo "</tr>";
     }
     echo "</table>";
}
else
{
     // no
     // print status message
     echo "No rows found!";
}

// close connection
mysql_close($connection);

?>

</body>
</html>
```

使用函数mysql_fetch_assoc()的另一种方法是函数mysql_fetch_object()，可以把每个行作为PHP对象进行检索。对象属性于是对应于不同的列值。

```
<html>
<head>
<basefont face="Arial">
</head>

<body>

<?php

// set server access variables
$host = "localhost";
$user = "joe";
$pass = "secret";
$db = "db123";
```

```
// open connection to database
$connection = mysql_connect($host, $user, $pass) or die ("Unable to connect!");

// select database to use
mysql_select_db($db) or die ("Unable to select database!");

// create SQL query string
$query = "SELECT * FROM employees";

// execute query and obtain result set
$result = mysql_query($query) or die ("Error in query: $query. " . mysql_error());

// are there any rows in the result?
if (mysql_num_rows($result) > 0)
{
     // yes
     // iterate through result set
     // format query results as table
     echo "<table cellpadding=10 border=1>";
     while($row = mysql_fetch_object($result))
     {
          echo "<tr>";
          echo "<td>" . $row->uname . "</td>";
          echo "<td>" . $row->fname . "</td>";
          echo "</tr>";
     }
     echo "</table>";
}
else
{
     // no
     // print status message
     echo "No rows found!";
}

// close connection
mysql_close($connection);

?>

</body>
</html>
```

通过函数mysql_fetch_field()还可以发现更多有关结果集中每个字段的情况，这将返回一个包含详细信息的字段对象，如名称、类型、长度以及数据类型等。参阅下面的示例：

```
<html>
<head>
<basefont face="Arial">
</head>

<body>

<?php

// set server access variables
```

```
$host = "localhost";
$user = "joe";
$pass = "secret";
$db = "db123";

// open connection to database
$connection = mysql_connect($host, $user, $pass) or die ("Unable to connect!");

// select database to use
mysql_select_db($db) or die ("Unable to select database!");

// create SQL query string
$query = "SELECT * FROM employees";

// execute query and obtain result set
$result = mysql_query($query) or die ("Error in query: $query. " . mysql_error());

// open outer list (for fields)
echo "<ol>";

// get the number of fields in the result set
// and iterate over them
for ($x=0; $x<mysql_num_fields($result); $x++)
{
     // for each field, retrieve the field as an object
     // and print the field name
     $obj = mysql_fetch_field($result, $x);
     echo "<li>" . $obj->name . "</li>";

     // open inner list (for field properties)
     echo "<ul>";

     // for each field object, display a list of all
     // object properties and values as a sub-list
     $vars = get_object_vars($obj);
     foreach ($vars as $key=>$value)
     {
          echo "<li>$key : $value</li>";
     }

     // close inner list
     echo "</ul>";
}

// close outer list
echo "</ol>";

// close connection
mysql_close($connection);

?>

</body>
</html>
```

正如所见到的，函数mysql_fetch_field()接受了一个参数以及结果集中字段的偏移，并且返回有关这个字段的不同的一组信息。这个信息表示出一个对象的属性，并且可以使用标准OOP表示法进行访问，与使用前面讨论的函数mysql_fetch_object()一样。

前面脚本的输出类似图21.3的形式。

图21.3 使用函数mysql_fetch_object()检索字段详细信息

除了这些函数以外，PHP还包括很多有关结果集处理的辅助函数。这些函数可以跳过结果集中不同的记录或字段、检索有关发现结果数量的信息、受到查询影响的行数、记录中字段的数量、名称和数据类型等。表21.1列出了这个方面的一些重要函数。

表21.1 有关结果集处理的辅助PHP函数

函数	功能
mysql_num_rows()	返回结果集的行数
mysql_affected_rows()	返回受到最后查询影响的行数
mysql_num_fields()	返回结果集的列数
mysql_field_len()	返回字段长度
mysql_field_name()	返回字段名
mysql_field_type()	返回字段数据类型
mysql_field_flags()	返回与字段有关联的标志
mysql_insert_id()	为了自动增量字段返回最后插入记录的ID
mysql_data_seek()	移动到结果集中不同的记录
mysql_field_seek()	移动到当前记录中的不同字段

有关对象

可以深入了解任何带有print_r()和var_dump()方法的PHP对象。请在前面脚本中创建的$obj对象上进行试验。也许你会对看到的情况感到吃惊。

21.2.4 错误处置

PHP的MySQL API带有一些功能强大的错误跟踪函数，可以加速开发时间。参阅下面的示例，其中在SELECT查询字符串中包含了一个故意的错误：

```
<?php

// connect
$connection = mysql_connect("localhost", "joe", "secret") or die("Invalid
server or user");
mysql_select_db("db123", $connection) or die("Invalid database");

// query
$query = "SELECT FROM employees";

// result
$result = mysql_query($query);

if(!$result)
{
        $error_number = mysql_errno();
        $error_msg = mysql_error();
        echo "MySQL error $error_number: $error_msg";
}

// clean up
mysql_close($connection);

?>
```

其输出情况类似图 21.4。

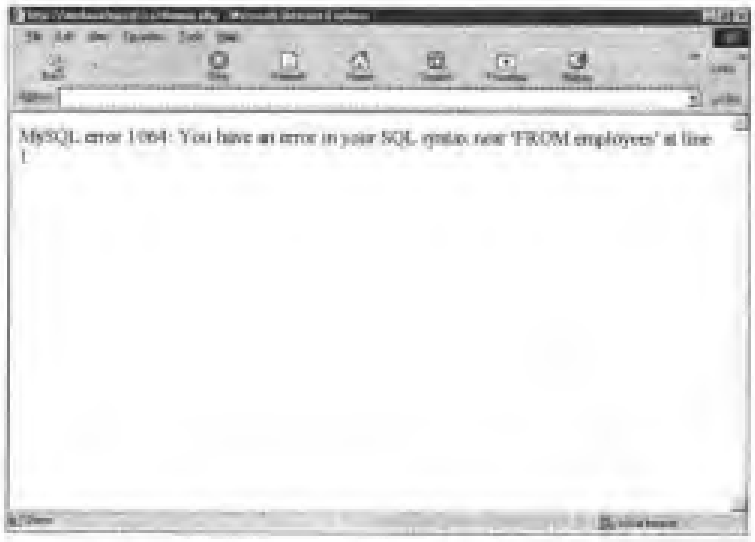

图 21.4　mysql_error()和 mysql_errno()的错误报告

如果错误是在 SQL 语句之中，函数 mysql_errno()显示了 MySQL 返回的错误代码，而函数 mysql_error()则返回实际的错误信息。将两个函数引入，将会发现它们能够大大减少修复错误的时间。

21.2.5　辅助函数

PHP除了有前面讨论过的函数以外，还带有很多专门用来提供服务器当前状况的信息以及列出可用数据库和表的函数。这些函数不像前面章节中介绍的函数那样常用，但是仍然值得花一些时间来研究一下。

表21.2列出了这方面的一些重要函数，而接下来的代码显示它们如何运行。

表 21.2 有关版本和状态报告的辅助函数

函数	功能
mysql_get_server_info()	返回 MySQL 服务器的版本号
mysql_get_proto_info()	返回 MySQL 协议的版本号
mysql_get_client_info()	返回 MySQL 客户机的版本号
mysql_get_host_info()	返回 MySQL 主机的信息
mysql_list_dbs()	返回 MySQL 服务器上可以使用的数据库列表
mysql_list_tables()	返回指定 MySQL 数据库中可以使用表的列表

```
<html>
<head>
<basefont face="Arial">
</head>
<body>

<?php

// set server access variables
$host = "localhost";
$user = "joe";
$pass = "secret";
$db = "db123";

// open connection to database
$connection = mysql_connect($host, $user, $pass) or die ("Unable to connect!");

// get MySQL server version
echo "MySQL server: " . mysql_get_server_info($connection) . "<br>";

// get MySQL host information
echo "MySQL host: " . mysql_get_host_info($connection) . "<br>";

// get MySQL client version
echo "MySQL client: " . mysql_get_client_info() . "<br>";

// get MySQL protocol version
echo "MySQL protocol: " . mysql_get_proto_info($connection) . "<p>";

// get list of available databases and tables
$dbs = mysql_list_dbs($connection);
echo "Available databases and tables:";
echo "<ul>";

// iterate over database list
while ($dbRow = mysql_fetch_array($dbs))
{
     // print database name
     echo "<li>" . $dbRow[0] . "</li>";

     // for each database, get list of tables within it
     $tables = mysql_list_tables($dbRow[0], $connection);
     echo "<ul>";
     // iterate over table list
     while ($tableRow = mysql_fetch_array($tables))
     {
```

```
            // print table name
            echo "<li>" . $tableRow[0] . "</li>";
        }
        echo "</ul>";
    }

    echo "</ul>";

    // close connection
    mysql_close($connection);

    ?>

    </body>
    </html>
```

图 21.5 显示了输出的可能情况。

图 21.5　检索 MySQL 服务器信息以及数据库和表的列表

21.3　真实范例

假设读者已经掌握了基本原理，现在可以将这些知识很好地应用了。本节将把前面几节中学到的所有知识汇总起来，以便构建一个简单的MySQL支持的Web应用程序，这是一个书签库，可以允许用户在数据库中保持自己喜爱网站的列表，以便展示PHP的MySQL函数的实际使用情况。

21.3.1　设计数据库

构建这种应用程序的第一步是了解需求。在这个应用程序中，每个用户将使用惟一的登录ID，这可以作为一个键，涉及到指向书签列表的用户。用户将能够使用简单的基于网页的界面查阅、添加或删除其书签。

基于这种需求，可以设计一个简单的数据库，作为这个应用程序的基础。这里使用的数据库包括两个表：bookmark_users 和 bookmark_urls，其中分别保存用户及其相应书签的列表。下面是创建和初始化这些表需要的 SQL 查询代码：

```
CREATE TABLE bookmark_users (
  uid tinyint(4) NOT NULL auto_increment,
  uname varchar(15) NOT NULL default '',
```

```
  PRIMARY KEY  (uid),
  UNIQUE KEY uid (uid),
  UNIQUE KEY uname (uname)
) TYPE=MyISAM;

INSERT INTO bookmark_users (uid, uname) VALUES (1, 'bill');

INSERT INTO bookmark_users (uid, uname) VALUES (2, 'john');

CREATE TABLE bookmark_urls (
  fk_uid tinyint(4) NOT NULL default '0',
  label varchar(255) NOT NULL default '',
  url varchar(255) NOT NULL default ''
) TYPE=MyISAM;


INSERT INTO bookmark_urls (fk_uid, label, url)
VALUES (1, 'Melonfire', 'http://www.melonfire.com/');

INSERT INTO bookmark_urls (fk_uid, label, url)
VALUES (1, 'Developer Shed', 'http://www.devshed.com/');

INSERT INTO bookmark_urls (fk_uid, label, url)
VALUES (1, 'PHP', 'http://www.php.net/');

INSERT INTO bookmark_urls (fk_uid, label, url)
VALUES (2, 'Yahoo', 'http://www.yahoo.com/');

INSERT INTO bookmark_urls (fk_uid, label, url)
VALUES (2, 'Slashdot', 'http://www.slashdot.org');
```

这样将为两个虚构用户：bill 和 john 创建两个带用户名、网站标题和网站 URL 的表。两个表通过 uid 外键列相互有关，连接 bookmark_users 中的用户到 bookmark_urls 的 URL 上。

21.3.2 检索数据

随着数据库的建立，将简单脚本汇集到一起，返回用户列表及其URL列表，并且在整齐的<table>中全部显示出来，实现这些都相当容易。

```
<html>
<head>
<basefont face="Arial">
</head>

<body>

<?php

// set server access variables
$host = "localhost";
$user = "joe";
$pass = "secret";
$db = "db123";

// open connection to database
```

```
$connection = mysql_connect($host, $user, $pass) or die ("Unable to connect!");

// select database to use
mysql_select_db($db) or die ("Unable to select database!");

// create SQL query string
$query = "SELECT uname, label, url FROM bookmark_users, bookmark_urls
WHERE bookmark_users.uid = bookmark_urls.fk_uid";

// execute query and obtain result set
$result = mysql_query($query) or die ("Error in query: $query. "
. mysql_error());

// are there any rows in the result?
if (mysql_num_rows($result) > 0)
{

     // yes
     // iterate through result set
     // format query results as table
     echo "<table width=450 border=1 cellspacing=0 cellpadding=10>";
     echo "<tr>";
     echo "<td align=left><b>User</b></td>";
     echo "<td align=left><b>Bookmark</b></td>";
     echo "</tr>";

     while($row = mysql_fetch_assoc($result))
     {
          echo "<tr>";
          echo "<td align=left>" . $row['uname'] . "</td>";
          echo "<td align=left><a href=" . $row['url'] . ">" . $row['label']
  "</a></td>";
          echo "</tr>";
     }
     echo "</table>";

}
else
{
     // no
     // print error message
     echo "No data found!";
}

// close connection
mysql_close($connection);

?>

</body>
</html>
```

正如在前面的示例中一样，脚本首先创建与数据库的连接。查询都进行了编辑，其结果集被返回到浏览器上。在这种情况下，因为存在的数据要比以前多，所以最好是使用函数

mysql_fetch_assoc()。应该记住，这个函数将从数据库中返回值，作为枚举数组，使得可以根据列名访问每个元素。

图 21.6 显示了前面脚本输出的式样。

User	Site #1	Site #2	Site #3
john	Melonfire	Devshed	PHP.Net
bill	Yahoo	Slashdot	32Bit.com

图 21.6 从一个 MySQL 数据库中检索被保存的书签

21.3.3 添加数据

至此，应用程序只能使用 SELECT 查询从数据库中提取信息，该如何向里面放置信息呢？接下来的脚本展示出如何使用一个 HTML 表单向数据库中插入数据。

```
<html>
<head>
<basefont face=Arial>
</head>
<body>
<center>

<?php
// has the form been submitted?
// no
if (!$_POST['submit'])
{
?>

<h3>Enter your bookmarks:</h3>
<form method="POST" action="<?php echo $_SERVER['PHP_SELF']; ?>">
<table>

<tr>
<td>Username</td>
<td><input name="uname" length="10" maxlength="30"></td>
</tr>

<tr>
<td>Web site label</td>
<td><input name="label" length="30" maxlength="30"></td>
</tr>
```

```
<tr>
<td>Web site URL</td>
<td><input name="url" length="30"></td>
</tr>

<tr>
<td colspan="2" align="center">
<input type="submit" name="submit" value="Submit">
</td>
</tr>

</table>
</form>
<?php
}
// yes
else
{

    // get POST data into regular PHP variables
    $uname = $_POST['uname'];
    $label = $_POST['label'];
    $url = $_POST['url'];

    // validate all POST data to ensure that it is
    // of the correct type and in valid format
    // ideally more complex validation should be performed here
    // for example, verifying the syntax of the URL
    // this is left as an exercise for the reader
    if (empty($uname)) { die("Error in USERNAME field!"); }
    if (empty($label)) { die("Error in LABEL field!"); }
    if (empty($url)) { die("Error in URL field!"); }


    // set server access variables
    $host = "localhost";
    $user = "joe";
    $pass = "secret";
    $db = "db123";

    // open connection to database
    $connection = mysql_connect($host, $user, $pass) or die ("Unable to
connect!");

    // select database to use
    mysql_select_db($db) or die ("Unable to select database!");

    // check to see if user already exists

    // create SQL query string
    $query = "SELECT uid from bookmark_users WHERE uname = '$uname'";

    // execute query and obtain result set
    $result = mysql_query($query) or die ("Error in query: $query. " .
mysql_error());
```

```
    if (mysql_num_rows($result) > 0)
    {
        // user exists
        // get UID
        $row = mysql_fetch_assoc($result);
        $uid = $row['uid'];
    }
    else
    {
        // user does not exist
        // create an entry for the user via an INSERT
        $query2 = "INSERT INTO bookmark_users (uname) VALUES ('$uname')";
        mysql_query($query2) or die ("Error in query: $query2. " . mysql_error());

        // and get the UID of the just-inserted record
        $uid = mysql_insert_id($connection);

    }

    // free result set memory
    mysql_free_result($result);

    // now insert the bookmark

    // create SQL query string
    $query = "INSERT INTO bookmark_urls (fk_uid, label, url) VALUES('$uid',
'$label', '$url')";

    // execute query and obtain result set
    $result = mysql_query($query) or die ("Error in query: $query. "
mysql_error());

    // close connection
    mysql_close($connection);
?>

<h3>Success!</43>
<table>
<tr>
<td>
<?php echo $_POST['uname']; ?>'s bookmarks have been saved.
</td>
</tr>

<?php
}
?>

</center>
</body>
</html>
```

如上所示，向数据库插入一条记录是很简单的，只需要简单地启动查询并运行其INSERT语句即可。正如前面介绍的，查找INSERT命令是否成功的惟一办法是检查变量$result值。如果变量值为假，则说明出现了问题。

图 21.7 和图 21.8 表明前面脚本输出的实际情况。

图 21.7　一个 HTML 表单可以让用户输入书签

图 21.8　处理和插入用户提供的书签到 MySQL 数据库的结果

并不完全真实

有经验的 Web 开发人员可能已经注意到，本节中的案例分析还不很完整。这种案例分析不能对每个页面上的用户认证、对会话管理、输入有效性和其他真正的 Web 开发中的内容来说并不充分。因为这些部分已经有意删除了，所以无法解决案例分析的主要目的：使用 PHP 构建一个数据驱动的 Web 应用程序。因此本节的代码完全是一种举例说明，特别忠告读者不要把这些示例用到实际环境中。

如果对如何构建安全的、功能完整的 Web 应用程序有兴趣，可以到 Melonfire 的 PHP 社区部分中查找更为详细的信息（包括更为实际的案例分析）。也可以到网站 http://www.melonfire.com/community/columns/trog/archives.php?category = PHP 中查找。

21.3.4　删除数据

十分明显，如果应用程序允许用户添加记录，还应该允许他们删除记录。下面的脚本将介绍这方面的内容。脚本将展示使用用户名作为键来删除记录的操作。再次说明，其基本原理是相同的，只是查询字符串变化了。

```
<html>
<head>
<basefont face=Arial>
</head>
<body>
<center>

<?php
// has the form been submitted?
// no
if (!$_POST['submit'])
{
?>

<h3>Enter your username:</h3>
<form method="POST" action="<?php echo $_SERVER['PHP_SELF']; ?>">
```

```
<table>

<tr>
<td>Username</td>
<td><input name="uname" length="10" maxlength="30"></td>
</tr>

<tr>
<td colspan="2" align="center">
<input type="submit" name="submit" value="Submit">
</td>
</tr>

</table>
</form>
<?php

}
// yes
else
{

    // get POST data into regular PHP variables
    $uname = $_POST['uname'];

    // validate form input
    if (empty($uname)) { die("Error in USERNAME field!"); }

    // set server access variables
    $host = "localhost";
    $user = "joe";
    $pass = "secret";
    $db = "db123";

    // open connection to database
    $connection = mysql_connect($host, $user, $pass) or die ("Unable to
connect!");

    // select database to use
    mysql_select_db($db) or die ("Unable to select database!");

    // check to see if user already exists

    // create SQL query string
    $query = "SELECT uid from bookmark_users WHERE uname = '$uname'";

    // execute query and obtain result set
    $result = mysql_query($query) or die ("Error in query: $query. " .
mysql_error());

    if (mysql_num_rows($result) > 0)
    {
        // user exists
        // get UID
        $row = mysql_fetch_assoc($result);
        $uid = $row['uid'];
```

```
    }
    else
    {
        die ("No such user!");
    }

    // free result set memory
    mysql_free_result($result);

    // now delete the user
    $query = "DELETE FROM bookmark_users WHERE uid = '$uid'";

    // execute query and obtain result set
    $result = mysql_query($query) or die ("Error in query: $query. " .
mysql_error());

    // next delete the bookmarks
    $query = "DELETE FROM bookmark_urls WHERE fk_uid = '$uid'";

    // execute query and obtain result set
    $result = mysql_query($query) or die ("Error in query: $query. " .
mysql_error());

    // close connection
    mysql_close($connection);
?>

<h3>Success!</43>
<table>
<tr>
<td>
<?php echo $_POST['uname']; ?>'s bookmarks have been deleted.
</td>
</tr>

<?php
}
?>

</center>
</body>
</html>
```

图 21.9 展示了初始表单的样式，而图 21.10 展示的是删除书签后的结果。

深入探讨

如果希望更深入地学习有关 PHP 的内容，可以访问下列在线资源：

- PHP 官方网站 http://www.php.net/。
- PHP 手册 http://www.php.net/manual/。
- Melonfire 的 PHP 部分，包括有关文章、案例分析和教材，位于 http://www.melonfire.com /community/columns/trog/archives.php?category=PHP。
- Devshed 的 PHP 部分，位于 http://www.devshed.com/Server_Side/PHP/。
- PHPBuilder，位于 http://www.phpbuilder.com/。

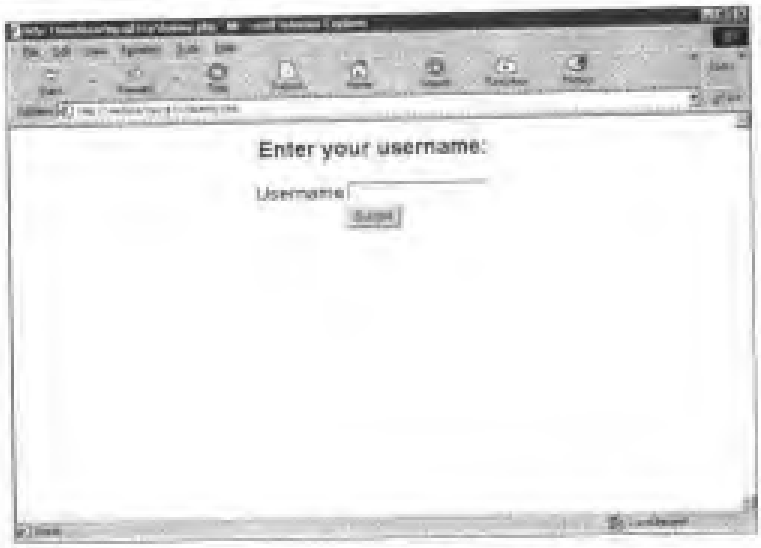

图 21.9 一个 HTML 表单使得用户可以删除书签

图 21.10 被用户删除书签后的结果

21.4 小结

本章讨论了 PHP 函数，使得开发者可以连接其 PHP 脚本到 MySQL 数据库服务器上，然后使用脚本进行交互操作、运行查询和处理结果记录集。更新版本的 PHP 通过不同的灵活的函数实现了对 MySQL 的内置支持。本章还展示了大多数函数，并提供了代码片段和完整的脚本。

本章介绍了 PHP 语言并继而详细讨论了 MySQL API 的 4 个主要部分：连接管理、查询运行、结果集处理和错误报告，最后展示了一些真实的 PHP MySQL 函数的应用程序，并使用这些函数构造了一个数据库驱动的书签应用程序。

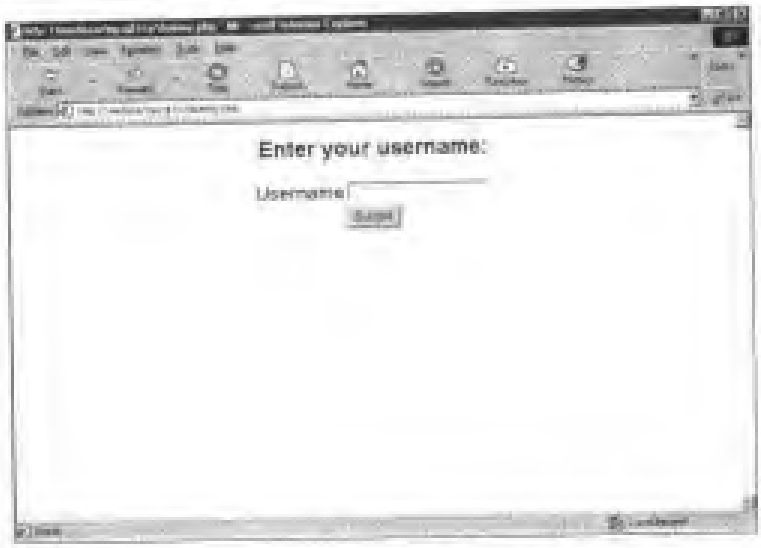

图 21.9 一个 HTML 表单使得用户可以删除书签

图 21.10 被用户删除书签后的结果

21.4 小结

本章讨论了 PHP 函数，使得开发者可以连接其 PHP 脚本到 MySQL 数据库服务器上，然后使用脚本进行交互操作、运行查询和处理结果记录集。更新版本的 PHP 通过不同的灵活的函数实现了对 MySQL 的内置支持。本章还展示了大多数函数，并提供了代码片段和完整的脚本。

本章介绍了 PHP 语言并继而详细讨论了 MySQL API 的 4 个主要部分：连接管理、查询运行、结果集处理和错误报告，最后展示了一些真实的 PHP MySQL 函数的应用程序，并使用这些函数构造了一个数据库驱动的书签应用程序。